Simulations-Modell für die Funkkommunikation in Straßenszenarien

Bachelorarbeit

von

Markus Michel

aus

Düsseldorf

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

6. Februar 2006

Betreuer:
Dipl.-Inf. Björn Scheuermann
Dipl. Wirtsch.-Inf. Christian Lochert

Danksagung

An dieser Stelle möchte ich mich besonders bei meinen beiden Betreuern Björn Scheuermann und Christian Lochert bedanken. Während der Bearbeitungsphase der Bachelorarbeit hatten sie bei Verständnisfragen oder sonstigen Problemen fast immer Zeit für mich. Durch ihre Erfahrungen mit wissenschaftlichen Veröffentlichungen konnten sie mir in der heißen Phase helfen meine Bachelorarbeit inhaltlich und strukturell zu optimieren.

Weiterhin möchte ich Alfonso Cervantes recht herzlich danken. Er hat für mich meinen Quellcode in den ns-2 implementiert und dadurch die Ausgabe von Simulationsergebnissen ermöglicht.

Als letztes danke ich allen Mitkommilitonen, die Interesse an meiner Arbeit gezeigt haben und auch in schwierigen Situationen als Ratgeber zu Verfügung standen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Mobile Netzwerke

Herkömmliche Netzwerke sind infrastrukturabhängig. Sie bestehen aus mehreren Endgeräten, die über Netzwerkkabel oder per WLAN miteinander verbunden sind. Die Übermittlung von Paketen erfolgt in diesen Netzen über zentrale Einheiten, die sogenannten Router. Diese Netzwerke sind für Datenübertragungen in mehr oder weniger statischen Umgebungen ausgelegt. In Szenarien mit sich bewegenden Endgeräten sind sie nur bedingt einsetzbar, denn um große Gebiete abzudecken, sind viele Funkstationen notwendig. Nach diesem Prinzip sind beispielsweise das GSM- und das UMTS-Netz für die Mobilkommunikation aufgebaut.

Mobile Ad-Hoc Netzwerke (MANETs) funktionieren nach einem völlig anderen Prinzip. Dies sind drahtlose Netzwerke, die sich aus mehreren mobilen Endgeräten zusammensetzen. Da sie keine Infrastruktur besitzen, und die Funkreichweite jedes Knotens begrenzt ist, muss die Kommunikation zwischen den einzelnen Teilnehmern des Netzwerkes kooperativ erfolgen. Ein Knoten agiert also gleichzeitig sowohl als normales Endgerät (Daten senden/empfangen) als auch als Router (Weiterleiten (Forwarden) von Paketen der anderen Knoten). Unterschieden werden können zwei Arten von Ad-Hoc Netzwerken.

Bei *Single-Hop Netzen* erfolgt die Kommunikation nur direkt von einem Teilnehmer zum nächsten. Diese Technik kommt beispielsweise bei modernen mobilen Spielekonsolen wie der Sony PSP [Kar] zum Einsatz. Erweiterte Funktionalitäten stellen mobile Ad-Hoc Netzwerke bereit. Hier kann die Kommunikation zwischen zwei Geräten auch über mehrere Teilnehmer hinweg erfolgen (*Multi-Hop*). Dadurch lassen sich größere Distanzen überwinden, weil man nicht so stark auf die Sendereichweite der Teilnehmer beschränkt ist. Diese Technik wird beim Militär und bei komplexen *Sensornetzwerken* [ASSC02], bei denen Kleinstrechner mit Sensoren ausgestattet Aufgabenbereiche, wie etwa ein Tsunamifrühwarnsystem übernehmen können, verwendet. Multi-Hop Netzwerke, die heutzutage meist aus Endgeräten wie Notebooks, PDAs oder Handys bestehen, sind also in vielen Bereichen einsetzbar.

1.2 Fahrzeug-zu-Fahrzeug-Kommunikation

Der Lehrstuhl für Rechnernetze beschäftigt sich derzeit mit einem weiteren Anwendungsgebiet, der Fahrzeug-zu-Fahrzeug-Kommunikation. Dabei werden Fahrzeuge auf Basis eines MANETs zu einem *Vehicular Ad-Hoc Network (VANET)* zusammengefügt. Ziel ist hier die Bildung eines Assistenzsystems für Autofahrer, in dem die einzelnen Fahrzeuge miteinander kommunizieren. Dieses System soll den Fahrer frühzeitig über bestimmte Verkehrssituationen wie etwa einen gerade stattgefunden Unfall informieren. Der Fahrer kann nun viel früher als heutzutage auf die Situation reagieren, was die Fahrsicherheit enorm steigern kann. Das computergestützte System kann also ein deutlich verbessertes vorausschauendes Fahrverhalten des Fahrers ermöglichen.

Da momentan nur Prototypen der Hersteller über die notwendige Technik verfügen, versucht der Lehrstuhl mit dem Verkehrssimulator VISSIM [PTV04] in Kombination mit dem Netzwerksimulator ns-2 [ns2] Straßenszenarien möglichst realitätsnah zu simulieren. Anschließend können aus den Ergebnissen dieser Tests unter anderem brauchbare Routingalgorithmen [LHT+03] entwickelt werden, die so im richtigen Straßenverkehr in den Fahrzeugen eingesetzt werden können. Momentan erfolgt die Kommunikation zwischen den einzelnen Fahrzeugen auf Basis einer vereinfachten Modellierung. Ein Fahr-

zeug darf Informationen zu einem anderen Fahrzeug schicken, wenn es innerhalb seines Kommunikationsradius – sprich der Sendereichweite seiner Funkantenne – liegt. In der Wüste oder auf anderen vollständig freien Flächen ist eine solche Vorgehensweise angebracht, nicht jedoch in einer belebten Stadt. Das Problem sind Hindernisse, die zwischen den Fahrzeugen liegen. Dies können beispielsweise Gebäude, Litfaßsäulen, Bäume, Ampeln oder Verkehrsschilder sein. Diese würden in realen Umgebungen das Funksignal nach dem Prinzip der *Mehrwegeausbreitung* [Wik] reflektieren, abschwächen oder anderweitig stören.

1.3 Problemstellung

Das Ziel dieser Bachelorarbeit ist es, den Realitätsgrad der Verkehrssimulationen zu verbessern. Es sollen Simulationen ermöglicht werden, die *obstacle modelling* beherrschen. *Obstacles* stellen das Komplement des Straßennetzes, d.h. alle Hindernisse abseits der Straßen, dar. Hauptaufgabe ist dabei die Entwicklung eines Algorithmus, der überprüft, ob die Kommunikation zwischen zwei Fahrzeugen durch irgendwelche Hindernisse behindert wird und anschließend entscheidet, ob eine Datenübertragung stattfinden darf oder nicht. Der Algorithmus muss dafür die Positionen der beiden Fahrzeuge wissen. Er benötigt darüber hinaus Zugriff auf die Straßendaten. Dies erfolgt idealerweise mit Hilfe einer speziellen Speicherstruktur, bei der es möglich ist nur die Straßen zu betrachten, die im Bereich zwischen den beiden Fahrzeugen liegen. Deshalb müssen die Straßendaten aus einer Szenariodatei des Simulators entsprechend eingelesen und aufbearbeitet werden. Abschließend soll untersucht werden, wie sich die Verwendung des Algorithmus auf die Simulationsergebnisse auswirkt.

1.4 Aufbau der Arbeit

Kapitel 2 gibt dem Leser einen grundlegenden Einblick in die Fahrzeug-zu-Fahrzeug Kommunikation. Dabei wird auf die beiden in dieser Arbeit in Kooperation eingesetzten Simulatoren, den Verkehrssimulator VISSIM und den Netzwerksimulator ns-2, sowie auf mathematische Grundlagen eingegangen. Kapitel 3 beschäftigt sich mit dem entwickelten Algorithmus. Hier wird genauer erklärt, wie die Straßendaten zunächst entspre-

chend aufgearbeitet eingelesen werden, und anschließend vom Algorithmus verarbeitet werden. Kapitel 4 geht näher auf Simulationen bei der Verwendung des entworfenen Algorithmus ein und wertet die dabei entstandenen Ergebnisse aus. Kapitel 5 schließt die Arbeit mit einen Blick auf mögliche Optimierungen/Erweiterungen sowie einem Fazit ab.

Kapitel 2

Grundbegriffe

2.1 Grundsätzliches

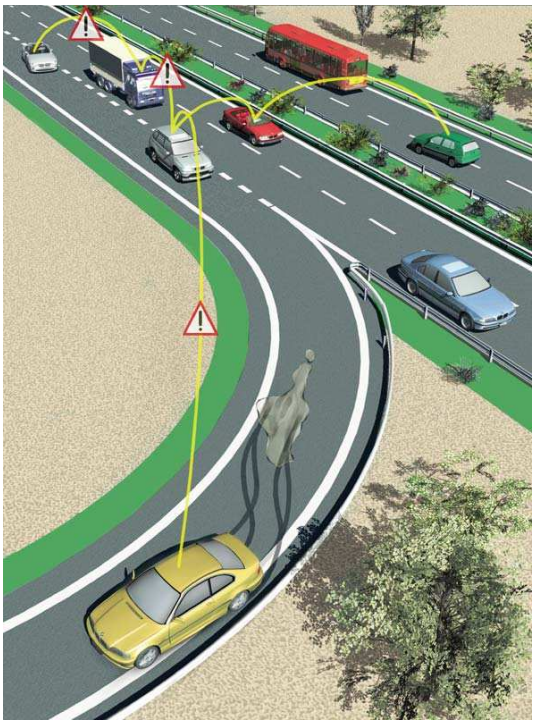
Große Automobilkonzerne wie BMW, Daimler-Chrysler oder Volkswagen sind ständig daran interessiert, Funktionen in die von Ihnen gebauten Fahrzeuge zu integrieren, die entweder den Komfort oder die Fahrsicherheit der Insassen steigern. In diesem Interesse wurde ein Verbund aus mehreren Firmen aus diesem Bereich auf EU-Ebene, das *Car2Car Communication Consortium* [Car], speziell für die Fahrzeug-zu-Fahrzeug-Kommunikation gegründet, um einen zwischen den einzelnen Herstellern kompatiblen Standard zu schaffen. In Deutschland ist mit *FleetNet* [Dai] und seinem Nachfolger *NOW* [NOW] ein ähnliches Projekt angelaufen.

Diese Projekte basieren auf dem Funkstandard IEEE 802.11 [IEE97], der nach [KS01] in der Variante 802.11b gründsätzlich für die Fahrzeug-zu-Fahrzeug-Kommunikation geeignet ist. Ein spezieller eigener Standard ist mit 802.11p sogar bereits in Planung [Fes05].

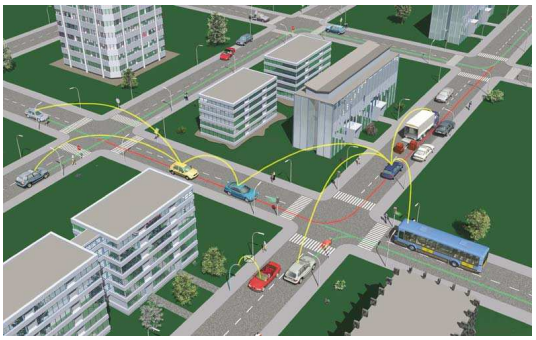
2.2 Mögliche Anwendungsgebiete

Die Anwendungsgebiete für die Fahrzeug-zu-Fahrzeug-Kommunikation sind recht vielfältig. Denkbar wäre es beispielweise, das Senden eines Notrufs oder einer Unfallwar-

nung, die Bereitstellung eines schmalbandigen Internetzugangs für die Abfrage von E-mails etc., oder die automatisierte elektronische Mauterfassung mit Hilfe dieser Technik zu realisieren.



(a) Gefahrensituation in der Autobahnausfahrt [BMW03]



(b) Erfassung der Verkehrslage [BMW03]

Abbildung 2.1: Beispielsituationen für die Fahrzeug-zu-Fahrzeug-Kommunikation

In Abbildung 2.1(a) ist ein Fahrzeug auf einer Autobahnausfahrt auf einer Ölspur gerutscht. Sensoren im Fahrzeug erkennen die vorliegende Situation, und senden eine Warnung über ein mobiles Ad-Hoc Netzwerk an die Fahrzeuge in der Umgebung. In Abbildung 2.1(b) blockiert ein LKW durch Lieferaktivitäten eine Straße vollständig. Ein sich dort befindendes Fahrzeug stellt dies fest, und teilt den anderen Fahrzeugen mit, dass diese besser eine andere Verkehrsroute wählen sollten. Damit kann ein unnötiger Stau vermieden werden.

2.3 Die Simulatoren

Die momentane Ausstattungsrate an Fahrzeugen, die über die notwendige Technik zur Durchführung von Fahrzeug-zu-Fahrzeug-Kommunikation verfügen, tendiert momentan aufgrund des recht frühen Entwicklungsstadiums nahezu gegen Null. Um die eingesetzte Technik trotzdem verbessern zu können, wird auf Verkehrssimulatoren zurückgegriffen. Sie bieten eine realistische Umgebung für die Simulation eines mobilen Multi-Hop Ad-Hoc Netzwerks. Der Markt bietet eine Reihe von Simulatoren, die sich teilweise stark im Umfang der Simulation unterscheiden. Manche von ihnen sind dabei auf bestimmte Teilbereiche spezialisiert, und deshalb für eine generalisierte Betrachtungsweise nicht verwendbar.

Mögliche Simulationsgebiete sind der Verkehrsfluss von Automobilen, Bussen, Eisen- und Straßenbahnen, und die Simulation von Fußgängerbewegungen. Beispiele sind CORSIM, TRANSYT-7F und VISSIM (Vergleich in [MFCO00]), der vom Lehrstuhl für Rechnernetze verwendet wird. Im Folgenden wird dieser Simulator etwas genauer vorgestellt.

2.3.1 Der Verkehrssimulator VISSIM

VISSIM [PTV04] bietet eine mikroskopische, verhaltensbasierte und vor allem sehr detaillierte Simulation sowohl von inner- als auch von außerörtlichen Gebieten an. Modelliert werden kann dabei der Individualverkehr und jeglicher öffentlicher Personennahverkehr. VISSIM unterteilt sich in zwei Hauptprogramme: das Verkehrsflussmodell zur Visualisierung der Simulation – wahlweise in 2D oder in 3D – sowie die Signalsteuerung, die Ampelanlagen und das generelle Fahrverhalten von Fahrzeugen, die von Punkt A nach Punkt B wollen, koordiniert. Dem Benutzer können bei Bedarf zusätzliche Informationen der Simulation wie etwa statistische Informationen über Wartezeiten an einer Ampel etc. ausgegeben werden.

Um eine realistische Simulation von Szenarien zu gewährleisten, verwendet VISSIM das *psycho-physische Wahrnehmungsmodell* von Wiedemann [Wie74]. Nach diesem Modell

passt jedes Fahrzeug seine Geschwindigkeit – wie es im normalen Straßenverkehr üblich ist – an auf seiner Strecke liegende Fahrzeuge an (*Folgemodell*). Das Ausmaß der Anpassung wird durch Faktoren wie die aus der aktuellen Geschwindigkeit abgeleitete Wahrnehmungsschwelle und dem Abstand zum Vordermann errechnet.

Zur verbesserten Genauigkeit trägt auch bei, dass VISSIM anstatt eines einfachen Fahrzeugs *Fahrer-Fahrzeug-Einheiten* durch das Szenario bewegt. Bei diesen können sowohl Fahrer als auch Fahrzeug verschiedene dynamische Eigenschaften aufweisen. So können Fahrer ein unterschiedliches Fahrverhalten, Fahrzeuge hingegen unterschiedliche maximal erreichbare Geschwindigkeiten haben.

Durch eine offene C-Programmierschnittstelle ist es möglich, eine individuelle Signalsteuerung für die Verkehrsimulation zu verwenden, und anderweitig Einfluss auf die Simulation zu nehmen.

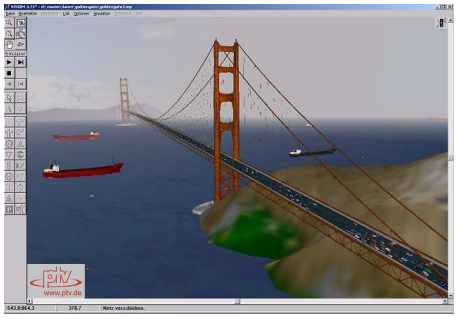


Abbildung 2.2: 3D-Visualisierung eines VISSIM Szenarios von San Fransicso [PTV05]

2.3.2 Der Netzwerksimulator ns-2

Es gibt sowohl freie Netzwerksimulatoren wie den *ns-2* [ns2], den *The Georgia Tech Network Simulator* [Ril03] und den *NCTUns 2.0* [Sim02] als auch kommerzielle Netzwerksimulatoren wie *OPNET* [OPN], den *CCNA network simulator* [Rou] oder *GlomoSim* [ZBG98]. Aus Gründen der guten Kombinationsmöglichkeit mit VISSIM wird am Lehrstuhl der Netzwerksimulator ns-2 eingesetzt.

Der ns-2 ist ein ereignisbasierter Netzwerksimulator, der in objektorientierter Form in C++ geschrieben ist. Zur Ausführung von Befehlsskripten ist ein OTcl-Interpreter integriert. Der Simulator bietet eine gute Unterstützung für mobile Endgeräte (Module für Link-Layer, MAC und Shared Channel) inklusive der gängigen Routingprotokolle für mobile Netze (DSR [IET04], AODV [PBR03], DSDV [PB94] und TORA [PC97]) an. Damit eignet er sich ausgezeichnet zur Simulation von mobilen Ad-Hoc Netzwerken.

Der normale Ablauf bei der Simulation eines *Inter Vehicle Communication (IVC)*-Szenarios wäre es, zuerst anhand des gewählten Verkehrssimulators Bewegungsmuster der beteiligten Fahrzeuge zu erstellen. Anschließend wird dann das Programm für den Nachrichtenaustausch – also in unserem Fall der Netzwerksimulator – innerhalb des VANET nach einem entsprechenden Routingprotokoll die Kommunikation zwischen den Fahrzeugen auf statische Weise durch schrittweise Abarbeitung der Bewegungsmuster durchgeführt. Diese Herangehensweise ist suboptimal, weil während der Laufzeit dynamische Änderungen der Bewegungsmuster nicht akzeptiert werden können.

Diese Probleme lassen sich umgehen, indem man die Simulation mit einer dynamischen Variante durchläuft. Dazu müssen beide Simulationsteile miteinander verlinkt werden. Der Ablauf der Simulation läuft hier nach einem anderen Prinzip ab. Der Netzwerksimulator führt bis zum Ende der Simulation zwei Schritte im Wechsel aus: die Anfrage des nächsten Zustandes im Bewegungsmuster und den Kommunikationsaustausch auf Basis des aktuell gültigen Zustandes des Bewegungsmusters. Zwingende Voraussetzung für die Verlinkung der beiden Simulationsteile ist es, dass eine zeitliche Synchronität zwischen den einzelnen Komponenten besteht. Eine Implementierung dieser Simulationsvariante zeigt [LCS+05], die im Folgenden näher erläutert wird.

2.3.3 Das Zusammenspiel der beiden Simulatoren

Die Architektur setzt sich aus vier Hauptkomponenten zusammen: VISSIM als Verkehrssimulator, der Simulation Control, dem Netzwerksimulator ns-2 und Matlab/Simulink (Abbildung 2.3). Der ns-2 wurde dabei um Funktionalitäten zur Synchronisation erwei-

tert. Zu Beginn der Simulation wird mit einer neuen Instanz des ns-2 eine Verbindung zu VISSIM aufgebaut und ein Timer gesetzt. Läuft dieser aus, fordert das Objekt solange, wie die Simulation nicht beendet wurde, von VISSIM immer und immer wieder die nächsten Bewegungskoordinaten des Bewegungsmusters an. Damit bestimmt der ns-2 den zeitlichen Ablauf der gesamten Simulation.

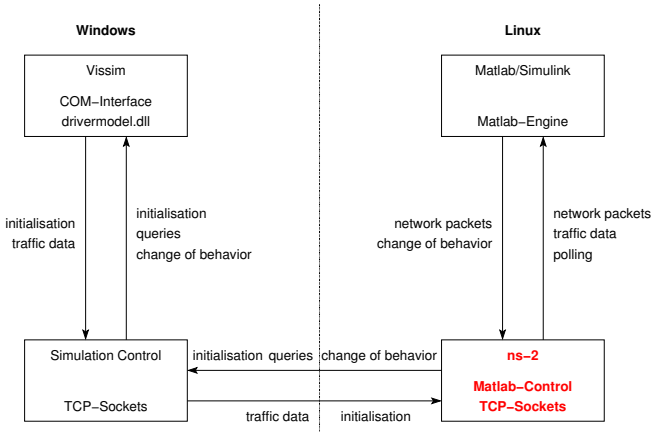


Abbildung 2.3: Darstellung der Simulationsumgebung [LCS+05]

Da VISSIM stets auf die nächste Anfrage des ns-2 warten muss, ist es möglich, während der Abarbeitung des aktuell betrachteten Bewegungsmusters den entsprechenden nächsten Schritt in VISSIM zu modifizieren.

Die zwei Schnittstellen von VISSIM

VISSIM bietet zwei Schnittstellen an, die sich für den Zugriff aus anderen Anwendungen heraus eignen: das Component Object Model (COM) Interface sowie die DLL-Datei *drivermodel.dll*. Das COM-Interface bietet einen wesentlich größeren Kontrollumfang, ist aber auch schwieriger zu handhaben. Deshalb wird es hier nur für Funktionen verwendet, die das zweite Interface nicht anbietet (Start/Stop der Simulation).

Durch die *drivermodel.dll* ist es möglich, während der Laufzeit Einfluss auf das Fahrverhalten der Fahrer zu nehmen. So kann beispielweise auf Anwendungsebene definiert werden, dass im Szenario ein Unfall stattgefunden haben soll, und die Fahrzeuge entsprechend zu reagieren haben. Da VISSIM Windows-basiert, der ns-2 hingegen Unix-basiert

ist, müssen beide Teile der Simulation jeweils auf einem separaten Computer laufen. Die Kommunikation zwischen beiden Rechnern übernimmt ein eigenständig Programm, die Simulation Control. Simulink ermöglicht die graphische Zusammenstellung von zu simulierenden Modellen. Es generiert zudem auf Wunsch aus der Simulation C-Code, der nach ausreichend erfolgreichen Simulationsergebnissen direkt in der Software der „real-world Produkte“ verwendet werden kann.

2.4 Die projektive Geometrie

Der Algorithmus, der in Kapitel 3.2 genauer beschrieben wird, benötigt während seiner Ausführung ein Verfahren, das in der Lage ist, den Schnittpunkt zwischen zwei Geraden zu bestimmen. Der dafür notwendige rechnerische Aufwand lässt sich stark reduzieren, wenn man die Geometrie aus einem anderen Blickwinkel betrachtet.

Die euklidische Geometrie ist in der Lage, geometrische Zusammenhänge darzustellen. Auf dieser Geometrie basieren viele Formeln, mit deren Hilfe man beispielsweise den Abstand zwischen zwei Punkten oder den Schnittwinkel zwischen zwei sich schneidenden Geraden bestimmen kann. Gleichzeitig sind einige geometrische Gesetzmäßigkeiten fest definiert. So haben etwa zwei zueinander parallel verlaufende Geraden im euklidischen Raum keinen gemeinsamen Schnittpunkt. Die Winkel innerhalb eines geometrischen Objektes bleiben im euklidischen Raum unter Transformation, Rotation und Translation zueinander invariant.

Betrachtet man den Abbildungsprozess der Linse einer Kamera, stellt man erstaunlicherweise fest, dass sich die Geometrie hier etwas anders als erwartet verhält. So sind zum Beispiel Längenverhältnisse oder Winkel „falsch“ dargestellt. Außerdem können sich hier parallele Geraden schneiden. Das liegt daran, dass hier mit der projektiven Geometrie [RK98] eine andere Darstellungsart der geometrischen Zusammenhänge verwendet wird.

2.4.1 Grundlegendes Wissen

Die projektive Geometrie bildet eine Obermenge zur euklidschen Geometrie. Zur Darstellung verwendet die projektive Geometrie *homogene Koordinaten*. Diese lassen sich leicht aus den euklidischen Koordinaten herleiten. Ein Punkt (x,y) wird in Richtung der W-Achse um 1 verschoben. Dabei ergibt sich der Wert $(x,y,1)$. Für diesen Punkt gilt die Eigenschaft der *uniformen Skalierung*:

$$\forall \alpha \neq 0 : (x,y,w) = (\alpha x, \alpha y, \alpha w)$$

Anschaulich betrachtet wird jeder zweidimensionale Punkt (x,y) aus dem euklidischen Raum im projektiven Raum auf einer Ebene mit der W-Koordinate 1 plaziert. Zu jedem dieser Punkte existiert eine Gerade, die ausgehend vom Ursprung des Koordinatensystems die Ebene genau in diesem Punkt (x,y) schneidet (Abbildung 2.4). Diese Gerade kann also dazu verwendet werden, diesen Punkt (x,y) zu repräsentieren.

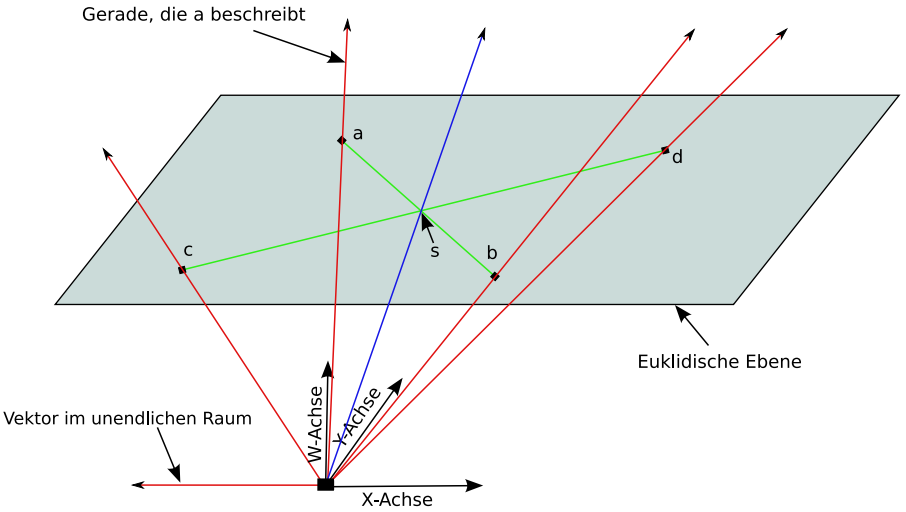


Abbildung 2.4: Darstellung des euklidischen Raumes in der projektiven Geometrie

Die projektive Geometrie kann auch Punkte im Unendlichen darstellen. Diese Punkte können ebenfalls durch einen Vektor ausgedrückt werden. Nach Definition handelt es sich um einen Punkt des unendlichen Raumes , wenn die W-Koordinate 0 ist. Dabei ergibt sich auch der Bezug zur euklidischen Ebene. Ein Vektor mit dieser Eigenschaft verläuft parallel zur Ebene des euklidischen Raumes. Deshalb existiert kein Schnittpunkt und

somit auch keine Möglichkeit diesen unendlichen Punkt innerhalb der Ebene darzustellen (Abbildung 2.4).

Eine Gerade lässt sich durch zwei Punkte eindeutig bestimmen. Anschaulich betrachtet spannen die beiden Geraden, die durch den Nullpunkt und je einen der beiden Punkte verlaufen, eine Ebene auf, die die euklidische Ebene in genau dieser Gerade schneidet. Das Kreuzprodukt der zu dieser Ebene zugehörigen Vektoren beschreibt den Normalenvektor der Ebene. Er stellt also damit gleichzeitig innerhalb der euklidischen Ebene die Gerade zwischen den beiden Punkten dar.

Im projektiven Raum besteht also kein Unterschied in der Darstellung eines Punktes und einer Geraden. Eine Ausnahme bildet dabei der Nullpunkt, der nicht definiert ist, weil er gegen das Axiom der uniformen Skalierung verstößt[1]. Konsequenterweise gelten für Punkt und Gerade die gleichen Axiome (*Dualitätsprinzip*).

2.4.2 Bestimmung des Schnittpunktes zweier Geraden

Um den Schnittpunkt zweier Geraden mit den Methoden der projektiven Geometrie zu bestimmen, benötigt man also im ersten Schritt die beiden zugehörigen Normalenvektoren n und m. Der Schnittpunkt s dieser beiden Geraden erfüllt folgende Eigenschaften:

1. $s \perp n$
2. $s \perp m$

Einen Vektor, der diese Bedingungen erfüllt, erhält man durch das Kreuzprodukt zwischen den beiden Vektoren n und m. Der sich daraus ergebene Vektor beschreibt den Schnittpunkt der beiden Geraden, der anschließend leicht wieder in den euklidischen Raum zurückgerechnet werden kann.

[1] Der Nullpunkt ist durch uniforme Skalierung nicht in einen anderen Punkt transformierbar.

Dafür werden die X- und Y-Koordinate durch die W-Koordinate dividiert: $(\frac{X}{W}, \frac{Y}{W})$.

Aus diesen Erkenntnissen abgeleitet ergibt sich nun die Formel, die in der Bachelorarbeit verwendet wurde:

$$k = \underbrace{(e_1 \times e_2)}_{n} \times \underbrace{(s \times z)}_{m}$$

Dabei repräsentieren e_1 und e_2 bzw. s und z jeweils die Koordinaten einer Geraden. Nach dem obigen Verfahren ist k der Schnittpunkt der beiden Geraden. Sofern k ein Punkt im Unendlichen ist, und deshalb in der euklidischen Ebene nicht existiert[2], weiß man, dass die beiden Geraden parallel zueinander verlaufen.

2.4.3 Vorteile der projektiven Geometrie

Auf den ersten Blick fragt man sich nun, ob es sich überhaupt lohnt, den Umweg zur Bestimmung des Schnittpunktes zwischen zwei Geraden über die projektive Geometrie zu gehen. Der Algorithmus muss zwischen jedem Kommunikationsschritt zwischen zwei Fahrzeugen durchgeführt werden. Es sollte also das Ziel sein, den rechnerischen Aufwand für seine Durchführung möglichst gering zu halten.

Im Gegensatz zu den gewöhnlichen Methoden der euklidischen Geometrie ist es mit unserer Version nicht notwendig, aufwändige Rechenoperationen wie etwa das Ziehen einer Wurzel[3] durchzuführen. Dadurch lässt sich die Rechenzeit des Algorithmus auf weniger Rechentakte reduzieren, was die Geschwindigkeit der Berechnung steigert.

[2] Der Versuch des Rückrechnens in die Koordinaten der euklidischen Geometrie ergibt eine Divison durch Null.

[3] Notwendig zur Überprüfung der Parallelität zwischen zwei Geraden.

Kapitel 3

Implementierung

Zur Beschreibung eines Verkehrsszenarios verwendet VISSIM ASCII-Textdateien mit der Dateiendung INP. In einer solchen meist recht umfangreichen Datei befinden sich sämtliche Informationen, die in der Simulation eine Rolle spielen. Hier werden unter anderem verschiedene Fahrzeugtypen mit unterschiedlichen Eigenschaften, mögliche Verhaltensweisen der Fahrer im Straßenverkehr, oder die Koordinaten der Straßen in Form von einzelnen Stützstellen definiert, um sowohl gerade Straßen als auch Kurven ausdrücken zu können. Der Beschreibung liegt dabei eine eindeutige Syntax zugrunde. Alle Werte stehen in Zusammenhang mit Schlüsselwörtern.

Der Algorithmus bestimmt, ob sich zwischen zwei Fahrzeugen Hindernisse befinden, oder direkter Sichtkontakt auf einer Sichtlinie besteht. Unter einem Hindernis versteht man in diesem Zusammenhang ein Objekt, das außerhalb des Straßennetzes liegt, und in der Lage ist, die Funkkommunikation zu stören. Wenn sich die Sichtlinie zwischen beiden Fahrzeugen also nicht auf Straßen des Straßennetzes befindet, heißt das, dass sie an irgendeiner Stelle auf ein Hindernis stößt. In diesem Fall ist Kommunikation unmöglich. Der Algorithmus wird als Methode innerhalb des ns-2 aufgerufen. Als Eingabewerte benötigt der „Sichtlinie-Algorithmus“ die Positionen der beiden zu überprüfenden Fahrzeuge und vollständigen Zugriff auf das Straßennetz des simulierten Szenarios. Die Koordinaten der Fahrzeuge sind bereits bekannt, da der ns-2 über das dll-Interface von VISSIM diese Information erfragen kann. Die Straßendaten hingegen sind nicht verfügbar. Deshalb müssen diese vor Simulationsbeginn aus der INP-Datei ausgelesen, und in

einer geeigneten Speicherstruktur, die für den Algorithmus zugänglich ist, abgespeichert werden.

3.1 Einlesen von Straßendaten

3.1.1 Der herkömmliche Weg

Natürlich ist es möglich die relevanten Informationen – also die Straßendaten – mit einer C-Methode, die den gesamten Text „parst“, direkt einzulesen. Der dafür notwendige Aufwand ist jedoch, wenn man den Umfang der VISSIM-Szenariodatei[1] betrachtet, vergleichsweise hoch. Bei dieser Variante darf auch nicht vernachlässigt werden, dass die zugrunde liegende Methode völlig statisch in Bezug auf die Einlesewerte aufgebaut ist. Ein im Text versehentlich gesetztes Tabulator- oder Returnzeichen löst hier direkt einen Fehler bei den einzulesenden Dateien aus. Dadurch kann die Datei evtl. unbrauchbar werden, weil sie falsche Informationen enthält.

3.1.2 Der Parsergenerator

Diese Probleme kann man elegant vermeiden, wenn man einen Parsergenerator zum Einlesen der Daten verwendet. Der Generator führt über eine Datei zuerst eine lexikalische und anschließend eine syntaktische Analyse durch. Strukturprobleme im Aufbau der Datei sind hier irrelevant, weil beispielsweise sämtliche Textsetzungszeichen wie die Leertaste oder das Return beim Einlesen ignoriert werden können. Als Ausgabe erzeugt der Generator C-Quellcode, der in der Lage ist, gleichartige Dateien zu parsen. Im Rahmen dieser Bachelorabeit kamen für diesen Zweck die verbesserten GNU-Werkzeuge [GNU] Flex [Fre05b] und Bison [Fre05a] zum Einsatz. Diese basieren auf den Tools lexx und yacc, die in jeder Unix-Standarddistribution enthalten sind.

[1] Aufwändige Dateien können schnell mehrere Megabyte groß sein.

Die lexikalische Analyse

Flex ist im ersten Schritt für die lexikalische Analyse zuständig. Es dient im Wesentlichen der Vorverarbeitung der Daten. Beim Durchlaufen der Datei wird nach zuvor in einer Datei definierten Schlüsselwörtern, den sogenannten Token, gesucht. Der komplette Rest hingegen wird verworfen. Alle Token setzen sich aus einem regulären Ausdruck zusammen. Sie werden zunächst in einer Variable vom Datentyp *string* (yytext) gespeichert, und müssen unter Umständen, wenn man mit diesen Token weitere Operationen plant, in einen anderen Datentyp konvertiert werden. So muß yytext etwa in einen *double*-Wert konvertiert werden, wenn eine ZAHL vorliegt. Dieser Teil geschieht in einem durch geschweifte Klammern abgekapselten C-Block.

Das vorstehende „<STRECKEN>“ bedeutet, dass diese Regel nur wirksam ist, wenn sich der Parser gerade im Zustand „<STRECKEN>“ aufhält. Dies ist vorteilhaft, denn in der VISSIM Szenariodatei sind natürlich nicht nur die für uns interessanten Straßendaten gespeichert. So ist es möglich, direkt zu dem für uns relevanten Datenteil – den Straßendaten – zu springen und sämtliche sonstige Vorkommen von Zahlen zu ignorieren.

```
<STRECKEN>-?[0-9]+(\.[0-9]+)? {yylval.Double = strtod(yytext,NULL); return (ZAHL);};
```

Diese Regel zeigt die Erkennung einer Zahl vom Datentyp *double*. Das Token ZAHL wird als solches erkannt, wenn eine Zahl vorliegt, die sich wie folgt zusammensetzt:

1. Optional ein negatives Vorzeichen
2. Beliebige Folge von ganzzahligen Ziffern als Vorkommateil
3. Optional ein Trennpunkt $\backslash\cdot$ mit anschließender Folge von ganzzahligen Ziffern als Nachkommateil

Syntax einer Straße in VISSIM

Eine Strecke hat in einer VISSIM Datei stets die gleichbleibende Syntax: VON x y (UEBER x y)+ NACH x y. Die Werte x und y sind dabei jeweils *double*-Werte, die zusammen eine Koordinate im zweidimensionalen Raum darstellen. Sie werden umgrenzt von den Token VON, UEBER und NACH. Die Token VON und NACH definieren den eindeutigen Start- bzw. Endpunkt der Straße. Das Token UEBER ist optional und beschreibt einen Zwischenpunkt auf der Straße. Um auch eine längere Kurve als Straße abbilden zu können, ist die Anzahl an Vorkommen von „UEBER x y“ innerhalb der Straßendefinition beliebig lang. Um eine solche Erkennung zu ermöglichen, muss Flex natürlich auch in der Lage sein, die Token VON, UEBER und NACH zu erkennen.

3.1.3 Die syntaktische Analyse

Im zweiten Schritt ist es die Aufgabe von Bison, syntaktische Zusammenhänge zwischen den einzelnen Token zu erkennen. Dafür wird hier die von Flex erzeugte Methode yylex() aufgerufen. Diese gibt nach und nach in der korrekten Reihenfolge die erkannten Token zurück. Der jeweils aktuelle Wert wird im entsprechenden Datentyp in der Variable yylval abgelegt.

Um die semantischen Zusammensetzung korrekt bestimmen zu können, benötigt Bison eine eigene Beschreibungsdatei. Diese beinhaltet Regeln in Form einer kontextfreien Grammatik in Backus-Naur-Form. Diesen Regeln kann nun – ähnlich wie den Token bei Flex – eine entsprechende Aktion in C-Code zugeordnet werden. Innerhalb einer Regel ist es möglich, verschiedene Fälle für eine Regel zu unterscheiden.

Die Regeln setzen sich aus Terminal- und Nichtterminalsymbolen zusammen. Um beide voneinander unterscheiden zu können, werden Terminalsymbole in Großbuchstaben und Nichtterminalsymbole in Kleinbuchstaben dargestellt. Terminalsymbole sind alle Token der lexikalischen Analyse. Um aus einer einfachen, sinnfreien Ansammlung von Token eine komplexe Regel bilden zu können, sind Nichtterminalsymbole notwendig. Sie kön-

nen nach Belieben definiert werden, und ermöglichen rekursive Strukturen innerhalb der Regeln.

Ein gewöhnlicher Compiler arbeitet nach dem Prinzip eines *Up-Down Parsers*. Bison hingegen ist ein *Bottom-Up Parser* [Ins04], bei dem das Prinzip an den Aufbau eines Baumes angelehnt ist. Bei diesem Verfahren wird ausgehend von allen Terminalsymbolen – den Blättern (bottom) – versucht, den Ausdruck Schritt für Schritt zu reduzieren. Dabei werden nach und nach Terminal- bzw. Nichtterminalsymbole durch entsprechend komplexere Nichtterminale ersetzt. Die einzige Bedingung dabei ist, dass diese dasselbe ausdrücken. So werden beim Einlesen der Straßendaten die Werte x und y, sofern ein Token VON, UEBER oder NACH davor steht, zum Nichtterminal *punkt*, welches in einem *struct Punkt* die beiden *double*-Werte abspeichert, zusammengefaßt.

Das Ziel ist die Umschreibung der kompletten Folge von Terminalsymbolen durch nur ein Nichtterminal. Wenn man sich den Ausdruck als Baum vorstellt, entspricht dies der Wurzel des Baumes(up). Die eigentliche Abarbeitung erfolgt dann beginnend mit dem Nichtterminal *strecken* in entgegengesetzter Richtung. Wie man sich die Anwendung dieses Verfahres für das Einlesen von Straßendaten vorstellen kann, zeigt folgende Abbildung 3.1:

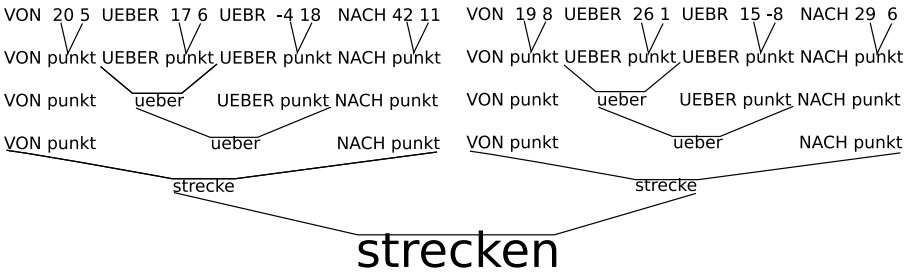


Abbildung 3.1: Bison Parserbaum für das Einlesen von Strecken

Einlesen von Straßen

Die obige Abbildung ist eine starke Vereinfachung dessen, was tatsächlich im Programm geschieht. Ausgehend von einer Ansammlung von Token, die in keinem Zusammenhang stehen, wird versucht, die Startregel aus Bison *strecken* – also die Wurzel des Baumes – zu „matchen“. Da die dafür benötigten Token nicht vorliegen, wird nun versucht, die ent-

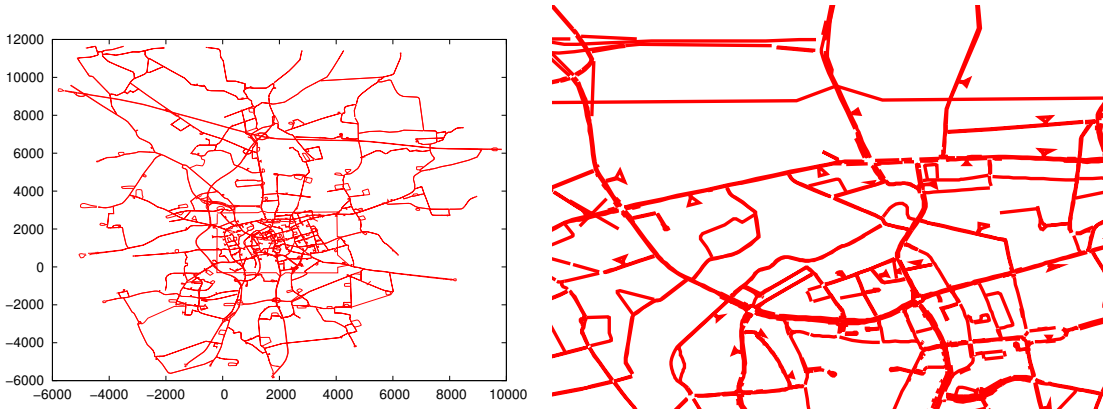
sprechend niederrangigen Regeln zu erfüllen. Nachdem die Werte x und y zum Nichtterminal *punkt* zusammengefasst wurden, kann die *ueber*-Regel angewendet werden. Diese erstellt bei ihrem ersten Aufruf im C-Segement eine doppelt-verkettete Liste vom Datentyp *list*. Für den benötigten Aufgabenbereich ist diese Speicherstruktur gut geeignet, da das Anhängen eines Elementes nur konstanten Aufwand hat.

Die Struktur muss zwingend dynamisch sein, da beim ersten Aufruf zur Laufzeit bei der Erstellung der Liste unklar ist, wieviele Elemente die Liste letztendlich enthalten wird. Jedes Element der Liste enthält ein Koordinatenpaar (x,y), das in einem *struct* mit der Bezeichnung Punkt abgelegt wird. Nach dem zweiten Teil der *ueber*-Regel wird nun diese Liste Schritt für Schritt um einen weiteren Punkt erweitert. Dadurch erhält man schließlich eine vollständige Liste mit allen Stützpunkten bezüglich der betrachteten Straße. Diese Liste ist im Nichtterminal *ueber* gespeichert.

Die *strecke*-Regel benötigt eine Folge vom Typ „VON *punkt ueber* NACH *punkt*“. Dabei können die beiden neu hinzugekommenen Punkte einfach an die Liste, die aus der *ueber*-Regel übergeben wurde, vorne bzw. hinten angehängt werden. Aus der daraus entstandenen Liste werden anschließend einzelne Streckenabschnitte erstellt.

Diese setzen sich jeweils aus den vier Koordinaten eines Streckenabschnittes zusammen. Der nächste Streckenabschnitt verwendet nun als Anfangspunkt den Punkt, den der letzte Streckenabschnitt als Endpunkt gesetzt hatte. Dabei ergibt sich eine gewisse Redundanz in der Speicherung.

Zu guter letzt werden die Streckenabschnitte zusammmen als eine Straße mit Hilfe einer map-Speicherstruktur abgespeichert. Deren Einträge setzen sich zusammen aus dem Index der Straße, sowie einem Zeiger auf die entsprechende Liste. Somit ist der Zugriff von außen auf eine bestimmte Straße (anhand des Index) durch den strukturierten Aufbau der Daten mit geringem Aufwand möglich.



(a) Komplettansicht

(b) Vergrößerter Ausschnitt

Abbildung 3.2: Unvollständige Darstellung des Straßennetzes von Braunschweig

Einlesen von Verbindungsstraßen

Das Ergebnis eines solchen Einleseprozesses zeigt Abbildung 3.2. Dabei fällt auf, dass das Straßennetz Lücken aufweist. Diese Löcher im System treten an den Straßenkreuzungen und zwischen einzelnen Straßen – also genau den Verbindungsstellen zwischen einzelnen Straßen – auf.

Um dieses Problem zu lösen, wurde die Einlesemethode um die Fähigkeit erweitert, Verbindungsstellen auch als Straßen einzulesen. Die Syntax einer solchen Verbindungsstrecke innerhalb der INP-Datei ist dabei anders als bei einer normalen Straße (Abbildung 3.3).

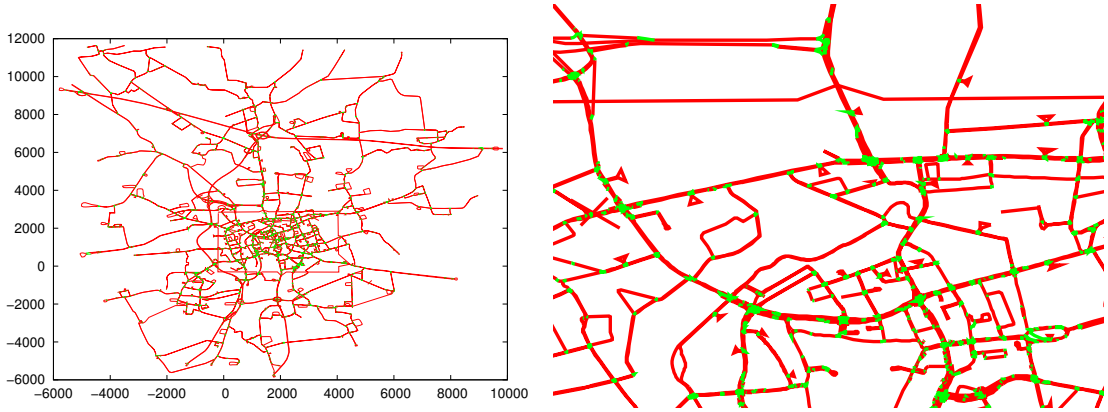
Die Stützpunkte können nach dem gleichen Verfahren wie bei normalen Straßen abgearbeitet werden. Die *strecke*-Regel muss jedoch erweitert werden, da die Notation für den Start- bzw. Endpunkt hier anders ist. Anstatt direkt die Koordinaten anzugeben, wird hier nur der Index einer Straße sowie ein Distanzwert angegeben. Dieser Wert beschreibt die Entfernung zu dem Punkt, den man erreicht, wenn man auf der Straße mit dem genannten Index vom Startpunkt aus entlang die Streckenabschnitte bis zu diesem Wert „abläuft“.

```
STRECKE     1 NAME "" BESCHRIFTUNG  0.00 0.00   TYP       1 LAENGE  114.960 SPUREN  2 SPURBREITE  3.50
3.50 STEIGUNG 0.00000 KOSTEN 0.00000 ZUSCHLAG 0.00000 ZUSCHLAG 0.00000 SEGMENT LAENGE
10.000    VON       -349.761     -236.000   NACH       -268.575     -154.608

VERBINDUNG 10000 NAME "" BESCHRIFTUNG  0.00 0.00   VON STRECKE 6 SPUREN 1 BEI 83.461   UEBER
-1087.085 -547.074 UEBER -1087.132 -547.090 UEBER -1093.028 -550.930 UEBER -1098.863 -554.940
UEBER -1098.910 -554.958    NACH STRECKE 80 SPUREN 1 BEI 0.603  ALLE   DXNOTHALT 5.000
DXEINORDNEN 200.000   STEIGUNG 0.00000  KOSTEN 0.00000  ZUSCHLAG 0.00000   ZUSCHLAG 0.00000
SEGMENT LAENGE   10.000
```

Abbildung 3.3: oben: normale Strecken-Definition, unten: Verbindungs-Definition

Wie das Straßenbild von Braunschweig aussieht, nachdem die Einlesemethode um eine Regel für Verbindungsstrecken und eine Methode zur Umrechnung der Abhängigkeit (Straßenindex,Distanzwert $\Longrightarrow$ X- und Y-Koordinaten des zugehörigen Punktes) erweitert wurde, zeigt Abbildung 3.4. Hierbei sind die Verbindungsstraßen durch grüne, die normalen Straßen durch rote Linien dargestellt.



(a) Komplettansicht

(b) Vergrößerter Ausschnitt

Abbildung 3.4: Darstellung des vollständigen Straßennetzes von Braunschweig

Notation einer Bison-Regel

Um einen kleinen Einblick in die genaue Funktionsweise einer Bison-Regel zu geben, wird im Folgenden die *ueber*-Regel in Abbildung 3.5 näher erläutert:

```
ueber : /* leer */ {$$ = (new list<struct Punkt>);} //Erstellung der Liste für die Stützpunkte
      | ueber UEBER punkt nutzlosezahl {$1->push_back($3);$$=$1;}   //Anhänges eines Punktes an
                                                                      die Liste
      ;
```

Abbildung 3.5: Auszug aus den verwendeten Bison-Regeln, hier: die *ueber*-Regel

Auf der linken Seite steht immer das Nichtterminal, hier: *ueber*, auf der rechten Seite sämtliche semantischen Konstellationen, in denen diese Regel auftreten kann. In unserem Fall tritt die Regel also in zwei verschiedenen Situationen in Kraft. Kommt ein *ueber* alleine vor, wird der erste Teil der Regel aufgerufen. Im hier nicht dargestellten Definitionsteil der Bisondatei wurde vorangehend der Datentyp der *ueber*-Regel festgesetzt. Da der Regel ein Variablenwert vom Typ einer Liste aus *struct Punkt* zugeordnet wurde, kann eine solche Liste erstellt werden.

Der erste Regelteil wird für jede Strecke nur einmal aufgerufen. Danach kann er nicht mehr „gematched“ werden. $$ entspricht dem Rückgabewert der Regel. Dieser Wert – ein Pointer auf eine Liste – ist wichtig, damit geklärt ist, wie der aktuelle Zustand innerhalb der Regel ist. Wird die Regel erneut aufgerufen, so kann mit diesem Zustand fortgefahren werden. Im zweiten Teil der Regel wird bei Aufruf an die zuvor im ersten Teil erstellte Liste ($1) ein Punkt angehängt ($3). Abschließend wird widerum ein Pointer auf die um ein Element verlängerte Liste zurückgegeben.

Aufgrund der Tatsache, dass VISSIM Szenariodateien aus regulären Ausdrücken aufgebaut sind, muss die Notation für eine Straße nicht immer vollständig identisch sein. So kommt es zum Beispiel vor, dass in einer Datei Stützstellen durch zwei Koordinaten, in der anderen Datei jedoch durch zwei Koordinaten gefolgt von einer Null ausgedrückt werden. Um diese Besonderheiten korrekt behandeln zu können, wurde das Nichtterminal *nutzlosezahl* erstellt. Diese Regel repräsentiert entweder die leere Menge oder ein erkanntes Token ZAHL, das im Vorgang der Streckenerkennung jedoch ohne Bedeutung ist.

Nachdem die Straßendaten erfolgreich eingelesen wurden, sind sie für den Algorithmus für den gesamten Ablauf der Simulation dadurch, dass sie in einer globalen Variable abgelegt wurden, zugänglich.

3.2 Idee des Sichtlinien-Algorithmus

Das Ziel, das der Algorithmus erreichen soll, wurde bereits in den vorangehenden Kapiteln erwähnt. Ausgehend von den zweidimensionalen Koordinaten von zwei Fahrzeugen soll ermittelt werden, ob eine freie Sichtline, evtl. über mehrere Straßen hinweg, besteht. Genau in diesem Fall ist es für eines der Fahrzeuge möglich, dem anderen Fahrzeug Informationen zu übermitteln. Die Sichtlinie ist genau dann frei, wenn jeder Punkt der Verbindungsstrecke zwischen den beiden Fahrzeugen nicht weiter als eine Konstante d von einer Straße entfernt ist.

3.2.1 Die Konstante d

Diese Konstante beschreibt den Bereich um jede Straße, in der eine Kommunikation zwischen Fahrzeugen funktioniert. Sie kann vor Beginn der Simulation frei gewählt werden. Es sollte jedoch vorsichtig mit ihr umgegangen werden, weil ein zu groß gewähltes d den eigentlichen Effekt des Algorithmus wieder zu Nichte macht. Wählt man nämlich $d = \infty$, können alle Fahrzeuge wieder nur begrenzt durch ihre Sendereichweite miteinander kommunizieren, was der Ausgangssituation vor Implementierung des Algorithmus entspricht. Man sollte es daher anstreben, dass d einen Wert beschreibt, der einerseits eine Kommunikation für einen Grünstreifenbereich einer großen Straße gestattet, aber andererseits die Kommunikation bei zu großen Hindernissen am Straßenrand untersagt.

In unserem Verfahren haben Straßen stets dieselbe Breite. Dies stimmt jedoch nur im idealistischen Sinn, denn VISSIM simuliert natürlich auch verschiedene Straßentypen wie Autobahnen, Landstraßen oder normale Straßen, die sich in der Spurbreite unterscheiden können. Durch geeignete Wahl von d kann dieses Problem jedoch vernachlässigt werden. Das Komplement des Bereiches – also alles, was nicht innerhalb dieser Rechtecke liegt – entspricht nach unserer Definition den umliegenden *obstacles* (Abbildung 3.7).

Abbildung 3.6 zeigt ein VANET. Die gestrichelte Linie stellt die Sichtlinie zwischen den einzelnen Fahrzeugen A bis D dar. Nach obiger Definition können A und B miteinander kommunizieren und B kann die Information von A auch an C weiterleiten. Zwischen C

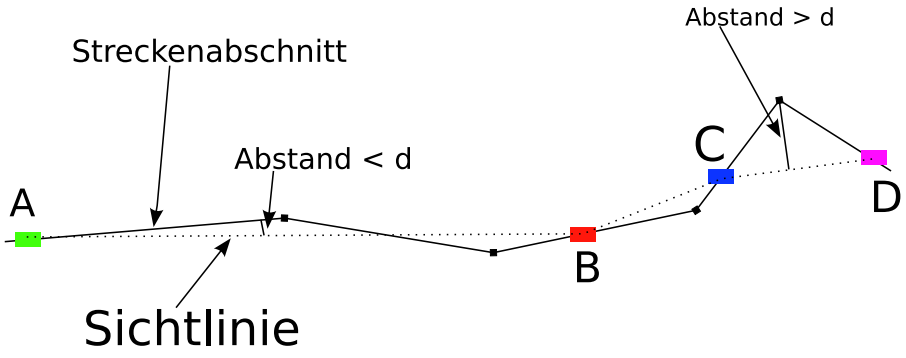


Abbildung 3.6: VANET, das aus vier Fahrzeugen besteht

und D hingehen befindet sich ein Gebäude, weshalb ein Datenaustausch nicht möglich ist.

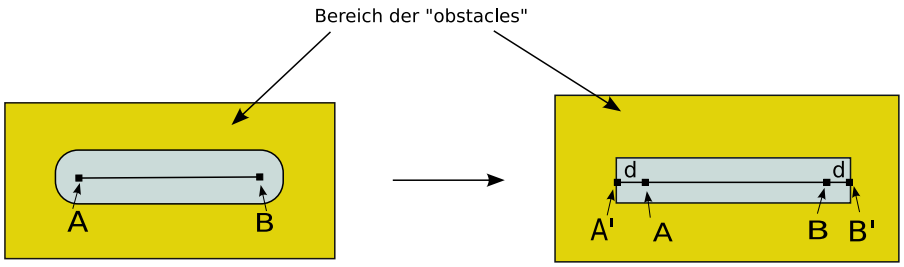


Abbildung 3.7: Übergang von der idealer zur vereinfachten Darstellung

3.2.2 Vereinfachung des Kommunikationsbereichs

Um den Aufwand der geometrischen Berechnung zu senken, haben wir die Umgebung, die durch d definiert wird, in ihrer Modellierung etwas vereinfacht. An den Enden des Streckenabschnitts haben wir dazu den Halbkreis um den Endpunkt der Strecke so ersetzt, dass nun um jeden Abschnitt ein vollständiges Rechteck zu finden ist (Abbildung 3.7). Zur weiteren Vereinfachung werden an dieser Stelle A' und B' eingeführt. Für gegebene Endpunkte A und B einer Strecke gilt:

$$A' := A - d \cdot \frac{B-A}{|B-A|} \tag{3.1}$$

$$B' := B + d \cdot \frac{B-A}{|B-A|} \tag{3.2}$$

Die Strecke wird also in ihrer Länge um den Teil, in dem eine Kommunikation möglich ist, erweitert (Abbildung 3.7). Beim Einlesen der Straßendaten werden, um dies umzusetzen, für jeden Streckenabschnitt nicht die Koordinaten A und B, sondern stets die Koordinaten A' und B' abgespeichert. Dies bietet einen Vorteil in der weiteren Berechnung. Das Rechteck lässt sich nun mit weniger Aufwand bestimmen.

3.2.3 Prinzipielle Vorgehensweise

Möchte nun ein Fahrzeug A überprüfen, ob es Sichtkontakt zu Fahrzeug B hat, geht es wie folgt vor: Es sucht alle Straßen, die innerhalb eines Kreises mit Radius d um seine aktuelle Position liegen. Für alle Straßen, die dort aufgefunden werden, erfolgt anschließend eine Überprüfung, ob diese geeignet sind, das Fahrzeug A näher an das Fahrzeug B zu bringen. Dazu wird wie in der Abbildung 3.7 jeweils ein Rechteck um die aktuell betrachtete Strecke gezogen. Alle vier Seiten des Rechtecks werden daraufhin mit der Sichtgeraden durch A und B geschnitten. Unter der Vorraussetzung, dass sich die aktuell betrachtete Koordinate innerhalb und nicht etwa auf dem Rand oder außerhalb des Rechtecks befindet, ergibt der Schnitt von der Geraden mit dem Rechteck stets zwei Schnittpunkte. Durch geschickte Wahl der Rechtecksgröße kann also ausgeschlossen werden, dass die Sichtgerade das Rechteck in einem Punkt – einem Eckpunkt – bzw. in unendlich vielen Punkten – einem Schnitt durch eine komplette Rechtecksseite – schneidet.

Aus diesen Schnittpunkten kann mit einer geometrischen Berechnung ermittelt werden, welcher der beiden Punkte besser als neuer Startpunkt geeignet ist. Für den Fall, dass das Zielfahrzeug innerhalb des Rechtecks liegt, bricht der Algorithmus mit einer Erfolgsmeldung ab. Anderenfalls wird erneut nach neuen Straßen gesucht, und entsprechend eine neue Überprüfung durchgeführt. Diese Vorgehensweise wiederholt sich so lange, bis der Algorithmus entweder erfolgreich ist, oder keine neuen Straßen in der Umgebung findet.

Der Algorithmus endet also niemals in einer Endlosschleife. Im vergangenen Schritt bereits betrachtete Straßen werden vom Suchvorgang ausgeschlossen. Abbildung 3.8 zeigt ein Beispiel für dieses Verfahren. Die einzelnen Schnittpunkte werden mit f_i bezeichnet, wobei i die Anzahl der bisherigen Schnittpunkte seit Beginn des Durchlaufs des Algorithmus' darstellt. Das Fahrzeug A findet also über die Zwischenpunkte f_1 und f_2 heraus, dass Sichtkontakt zu Fahrzeug B besteht.

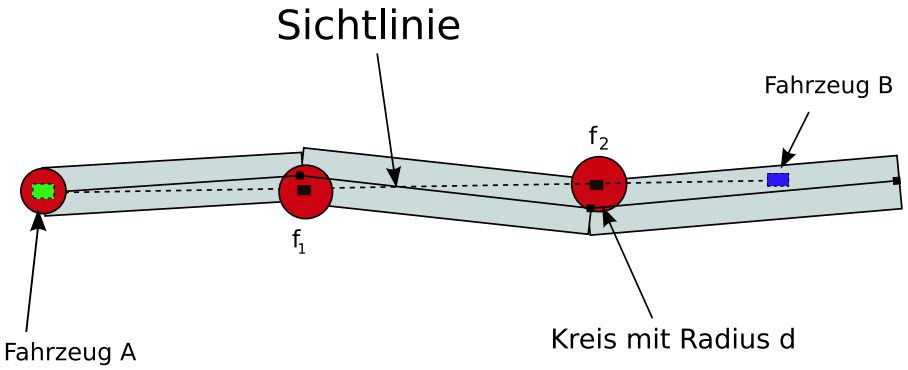


Abbildung 3.8: Beispiel für die erfolgreiche Durchführung des Algorithmus'

Um die genaue Implementierung dieser Algorithmusidee besser erklären zu können, zeigt Algorithmus 1 das Verfahren im Pseudocode. Wie im obigen Beispiel möchte der ns-2 testen, ob Fahrzeug A Daten an Fahrzeug B senden darf. R steht für die Menge aller Straßenabschnitte. Entsprechend steht r für einen Abschnitt einer Straße aus unserer zuvor eingelesenen dynamischen Speicherstruktur der Straßen des VISSIM-Szenarios. X ist die Menge der Strecken, die sich in der Nähe des aktuellen f_i befinden, x beschreibt entsprechend die einzelnen Elemente daraus. a_1 bis a_4 sind die vier Ecken des Rechtecks, k_1 und k_2 die beiden Schnittpunkte, die entstehen, wenn das Rechteck mit der Sichtlinie zwischen dem Startpunkt s und dem Zielpunkt z – den aktuellen Koordinaten der beiden Fahrzeuge – geschnitten wird. Diese Koordinaten kann der ns-2 durch seine Schnittstelle zu VISSIM selbst erfragen.

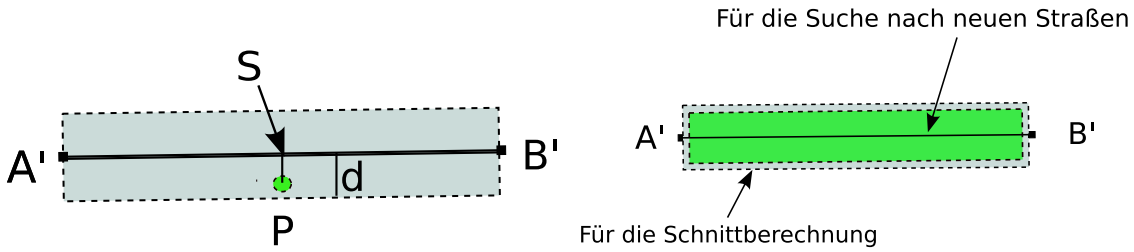
Eine gefundene Strecke kann akzeptiert werden, wenn zwei Bedingungen für den aktuell betrachteten Zwischenpunkt f_i erfüllt sind:

1. Der Abstand von f_i zum Streckenabschnitt ist kleiner als $(1-\varepsilon)\cdot d$. Dabei muß gelten: $1 \geq \varepsilon \geq 0$. Eine gute Wahl für ε ist etwa 0,01.

1: $i \leftarrow 0$
2: $f_0 \leftarrow s$
3: **while** true **do**
4: $X \leftarrow \{r \in R \mid f_i$ liegt im Bereich der Strecke $r\}$
5: **if** $X = \emptyset$ **then**
6: return false
7: **end if**
8: Wähle $x \in X$ beliebig
9: Erstelle Rechteck um x mit Eckpunkten: a_1, a_2, a_3, a_4
10: **for all** Seiten des Rechtecks $\overline{e_1 e_2} \in \{\overline{a_1 a_2}, \overline{a_2 a_3}, \overline{a_3 a_4}, \overline{a_4 a_1}\}$ **do**
11: $k \leftarrow (e_1 \times e_2) \times (s \times z)$
12: **if** k liegt im Unendlichen **then**
13: next
14: **end if**
15: **if** $k \notin \overline{e_1 e_2}$ **then**
16: next
17: **end if**
18: $a \leftarrow \left(\overrightarrow{k_1 f_i}\right)^2 < \overrightarrow{k_1 f_i} \bullet \overrightarrow{z f_i}$
19: **if** a **then**
20: $f_{i+1} = k_1$
21: **end if**
22: $b \leftarrow \left(\overrightarrow{k_2 f_i}\right)^2 < \overrightarrow{k_2 f_i} \bullet \overrightarrow{z f_i}$
23: **if** b **then**
24: $f_{i+1} = k_2$
25: **end if**
26: **if** $(\text{not } a) \wedge (\text{not } b)$ **then**
27: return true
28: **end if**
29: **end for**
30: $i \leftarrow i + 1$
31: **end while**

Algorithmus 1: Algorithmus für die Überprüfung der Sichtlinie

2. Der Schnittpunkt *s*, der sich ergibt, wenn man eine zur Strecke orthogonale Hilfsgerade, die durch f_i läuft, mit der Strecke schneidet, befindet sich auf der Strecke zwischen A' und B' und hat zu diesen Punkten einen Abstand $\geq \varepsilon$.



(a) Verdeutlichung der beiden Bedingungen (b) Darstellung der verschiedenen Rechtecke

Abbildung 3.9: Erklärungen zum Kommunikations-Rechteck

Beide Bedingungen zusammen stellen sicher, dass sich f_i sowohl in horizontaler (Punkt 1) als auch in vertikaler Lage (Punkt 2) innerhalb des Rechtecks um die Strecke befindet (Abbildung 3.9).

3.2.4 Ignorieren von schon betrachteten Strecken

Ignoriert man die Bedingungen aus Abschnitt 3.2.3 und verwendet für die Straßensuche ein Rechteck mit Abstand *d*, hat dies einen entscheidenen Nachteil. Falls man Straßen mit mehreren Streckenabschnitten betrachtet, wird nun ab dem zweiten Streckenabschnitt jeweils in der Suche auch die alte Strecke, die im vorherigen Schritt betrachtet wurde, ebenfalls als gültige Strecke gefunden. Diese Strecke liegt auf jeden Fall näher am Startpunkt als am Zielpunkt. Sie kann also niemals einen Forschritt in Richtung Ziel bringen, wird aber trotzdem jedes Mal betrachtet.

Wählt man den Abstand wie in den Bedingungen, kann man dieses Problem elegant umschiffen. Hier ist dann das Rechteck, in dem Straßen gesucht werden, stets kleiner als das Rechteck, das betrachtet wird, wenn die Schnittpunkte bestimmt werden sollen. Wird keine Straße gefunden, die diese Vorraussetzung erfüllt, d.h. $X = \emptyset$, kann der Algorithmus logischerweise abgebrochen werden.

Die gefundenen Strecken x, die diese Bedingungen erfüllen, werden nun genauer betrachtet. Sobald ein Schnittpunkt mit dem Rechteck gefunden wird, der näher an z liegt als das aktuelle f_i, wird dieser als neues f_i gewählt.

3.2.5 Erstellung des Rechtecks um eine Strecke

Bis dies stattgefunden hat, wird mit jeder Strecke wie folgt verfahren: Es wird das Kommunikations-Rechteck um die Strecke gelegt. Dazu wird der normierte Vektor $\overrightarrow{v}$ benötigt. Dieser steht senkrecht auf der Strecke $\overline{A'B'}$. Mit seiner Hilfe lassen sich die Koordinaten der Ecken des Rechtecks – also a_1, a_2, a_3 und a_4 – bestimmen. Für $A' = (x_{A'}, y_{A'})$ und $B' = (x_{B'}, y_{B'})$ gilt:

$$\begin{aligned}
\overrightarrow{v} &= \frac{1}{|\,B' - A'\,|} \cdot \begin{pmatrix} y_{B'} - y_{A'} \\ -x_{B'} + x_{A'} \end{pmatrix} \\
a_1 &= A' + d \cdot \overrightarrow{v} \\
a_2 &= B' + d \cdot \overrightarrow{v} \\
a_3 &= A' - d \cdot \overrightarrow{v} \\
a_4 &= B' - d \cdot \overrightarrow{v}
\end{aligned}$$

Die vier Seiten des Rechtecks, die durch diese vier Punkte eindeutig definiert sind, werden anschließend jede für sich betrachtet. Durch die dreimalige Anwendung des Kreuzproduktes zwischen einer Rechtecksseite $\overline{e_1 e_2}$ und der Sichtlinie $\overline{sz}$ lässt sich k wie folgt berechnen (Abschnitt 2.4.2):

$$k = (e_1 \times e_2) \times (s \times z)$$

k beschreibt den Schnittpunkt zwischen der Rechtecksseite und der Sichtlinie. Sofern k einen Schnittpunkt ergibt, der im euklidischen Raum nicht existiert, weiß man, dass die entsprechende Seite parallel zur Sichtlinie verläuft. Diese Seite muss nicht näher betrachtet werden, und kann verworfen werden. Anderenfalls gilt es zu überprüfen, ob

sich der Schnittpunkt überhaupt auf der Strecke der Rechtecksseite befindet, oder die Seite außerhalb des Rechtecks schneidet (Abbildung 3.10).

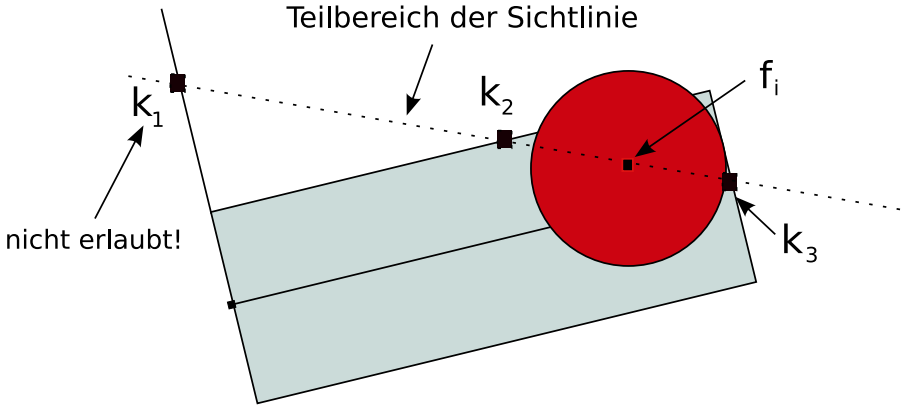


Abbildung 3.10: k_1 wird als ungültiger Schnittpunkt verworfen

3.2.6 Die beiden Schnittpunkte

Nach den Regeln der Geometrie ergibt der Schnitt zwischen einem Rechteck und der Sichtgeraden mit den von uns gewählten Beschränkungen der Rechtecksgröße stets zwei Schnittpunkte, nämlich k_1 und k_2. Mit diesen wird eine geometrische Berechnung durchgeführt. Diese basiert auf einer Gesetzmäßigkeit, die nur bei Vektoren, die parallel zu einander verlaufen, gilt:

$$\vec{a} \parallel \vec{b} \Longrightarrow \vec{a} \bullet \vec{b} = \begin{cases} |\vec{a}| \cdot |\vec{b}| & \text{wenn } \vec{a} \text{ und } \vec{b} \text{ in die selbe Richtung zeigen} \\ -\left(|\vec{a}| \cdot |\vec{b}|\right) & \text{sonst} \end{cases}$$

Wie Abbildung 3.11 zeigt gibt es vier mögliche Positionen, an denen z liegen kann. Entweder irgendwo außerhalb des Rechtecks auf einem anderen Streckenabschnitt oder innerhalb des Rechtecks vor bzw. hinter dem aktuellen f_i. Alle Möglichkeiten, die sich daraus für k_1 und k_2 ergeben, zeigt Tabelle 3.1.

Im ersten Fall für Bedingung a ergibt das Skalarprodukt auf der rechten Seite einn größeren Wert als auf der linken Seite, weil $\overrightarrow{zf_i}$ länger ist als $\overrightarrow{k_1 f_i}$. Deshalb ergibt diese

Fall	Bed. a: $(\overrightarrow{k_1 f_i})^2 < \overrightarrow{k_1 f_i} \bullet \overrightarrow{z f_i}$	Bed. b: $(\overrightarrow{k_2 f_i})^2 < \overrightarrow{k_2 f_i} \bullet \overrightarrow{z f_i}$
1	true	false
2	false	false
3	false	false
4	false	true

Tabelle 3.1: Fallunterscheidung für die Lage des Punktes z

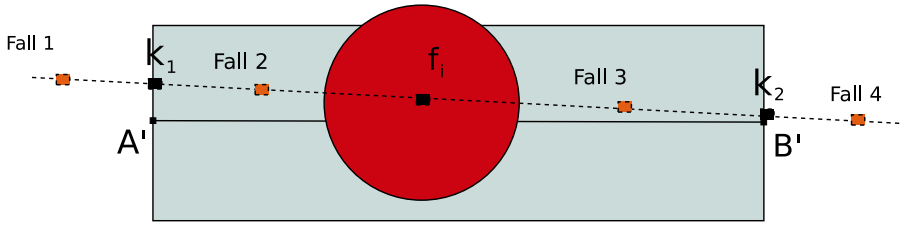


Abbildung 3.11: Vier Bereiche, in denen ein möglicher Zielpunkt liegen kann

Bedingung den Wahrheitswert true. Im zweiten Fall sieht es anders aus. Weil sich der Zielpunkt innerhalb des Rechtecks befindet, ist $\overrightarrow{z f_i}$ kürzer als $\overrightarrow{k_1 f_i}$. Somit ist die Aussage falsch. Im dritten und vierten Fall verlaufen die betrachteten Vektoren entgegengesetzt. Nach der obigen Formel ist klar, dass in beiden Fällen Bedingung a false ist. Für Bedingung b ergeben sich die zu Bedingung a gespiegelten Ergebnisse.

Trifft Bedingung a zu, so muss also das Ziel außerhalb des Rechtecks bezüglich der Seite mit Schnittpunkt k_1 liegen. Ähnliches gilt für Bedingung b mit der anderen Seite, auf der sich Schnittpunkt k_2 befindet. Für diese beiden Fälle ist es klug, entsprechend k_1 bzw. k_2 als neues f_i auszuwählen. Treffen die Aussagen a und b jedoch beide nicht zu, kann man sofort nach Fall 2 oder 3 daraus folgern, dass sich z irgendwo innerhalb des Rechtecks befinden muss. Deshalb kann der Algorithmus dann erfolgreich beendet werden, weil sich nach Definition des Rechtecks zwischen f_i und z keine *obstacles* befinden können.

Kapitel 4

Simulationsergebnisse

4.1 Vorbereitung: Integration in den ns-2

Da der Quellcode des Netzwerksimulators ns-2 offen liegt, ist es möglich, eigene Codeabschnitte einzufügen. Nach einer Neukompilierung des Quellcodes können anschließend die dadurch entstandenen erweiternden Funktionalitäten genutzt werden. Für die Verwendung im ns-2 wurde der Algorithmus in eine boole'sche Methode verpackt, der die Koordinatenpaare von zwei Fahrzeugen und der Straßenabstandsparameter *d* übergeben werden. Als Rückgabewert liefert diese Methode einen Wahrheitswert, auf den der ns-2 entsprechend reagieren kann.

4.2 Die Szenarien

Szenarien werden meist in extrem mühevoller Detailarbeit in eine INP-Datei übertragen. Aus diesem Grund sind leider noch nicht viele Szenarien verfügbar, die eine Umgebung mit vielen *obstacles* – wie etwa eine komplette Stadt – beschreiben. Deshalb fiel die Wahl des Szenarios recht schnell auf die Stadt Braunschweig – einer Stadt in Deutschland, die in großen Teilen erfasst wurde. Simuliert werden dabei etwa 800 km Straßen und 3000 Fahrzeuge, die jeweils einer bestimmten Fahrzeuggruppe mit spezifischen Eigenschaften zugeordnet sind.

4.3 Die Simulation

Grundvorraussetzung für einen Vergleich verschiedener Simulationsvarianten ist es, dass bei allen dasselbe Szenario – bei uns Braunschweig – und die gleichen Simulationsbedingungen gewählt werden. In unserer Simulation soll untersucht werden, wie gut sich die Information über ein stattgefundenes Ereignis – zum Beispiel einen Verkehrsunfall – über alle Fahrzeuge hinweg verbreitet. Dafür wird ein fester Punkt P im Szenario gewählt, der kontinuierlich neue Informationen über das Ereignis in seiner Sendereichweite broadcastet. In der Realität repräsentiert dieser Punkt also beispielsweise das verunglückte Fahrzeug, dass die in der Nähe befindlichen Fahrzeuge über den aktuellen Stand informieren möchte. Erhält ein solches Fahrzeug die Information, versucht es widerum, sie an seine Nachbarn weiterzuleiten. Dadurch entsteht ein Informationsfluss, der die Information nach und nach über Teile des Szenarios verbreitet. Dabei speichern die Fahrzeuge immer die aktuell neueste Information bezüglich des Ereignisses. Um dabei realistische Ergebnisse zu erhalten, wird davon ausgegangen, dass die Austattungsrate – also die Anzahl der Fahrzeuge, die über die notwendige Hardware zum Senden verfügen – auf 20 Prozent beschränkt ist.

Der Lehrstuhl für Rechnernetze setzt momentan ein anderes Verfahren zur Simulation von Szenarien mit *obstacles* ein. Vor Simulationsbeginn werden hier mit dem Cursor Koordinaten von Rechtecken, die Hindernisse repräsentieren sollen, festgesetzt und in einer Speicherstruktur abgelegt. Vor jedem Kommunikationsschritt wird dann überprüft, ob die Sichtlinie zwischen den beiden Fahrzeugen irgendein *obstacle* schneidet, was per Definition die Kommunikation unmöglich macht.

In unserer Simulation werden unter den oben genannten Bedingungen drei Varianten mit einander verglichen: der Sichtlinien-Algorithmus bei unterschiedlicher Wahl des Straßenabstandsparameters d, das manuelle Verfahren und eine Variante, in der *obstacles* nicht verwendet werden.

Abbildung 4.1 zeigt, wie sich generell die Informationen im Durchschnitt aller Fahrzeuge, die sich im Szenario bewegt haben, in Abhängigkeit zur Distanz vom Ort P ,

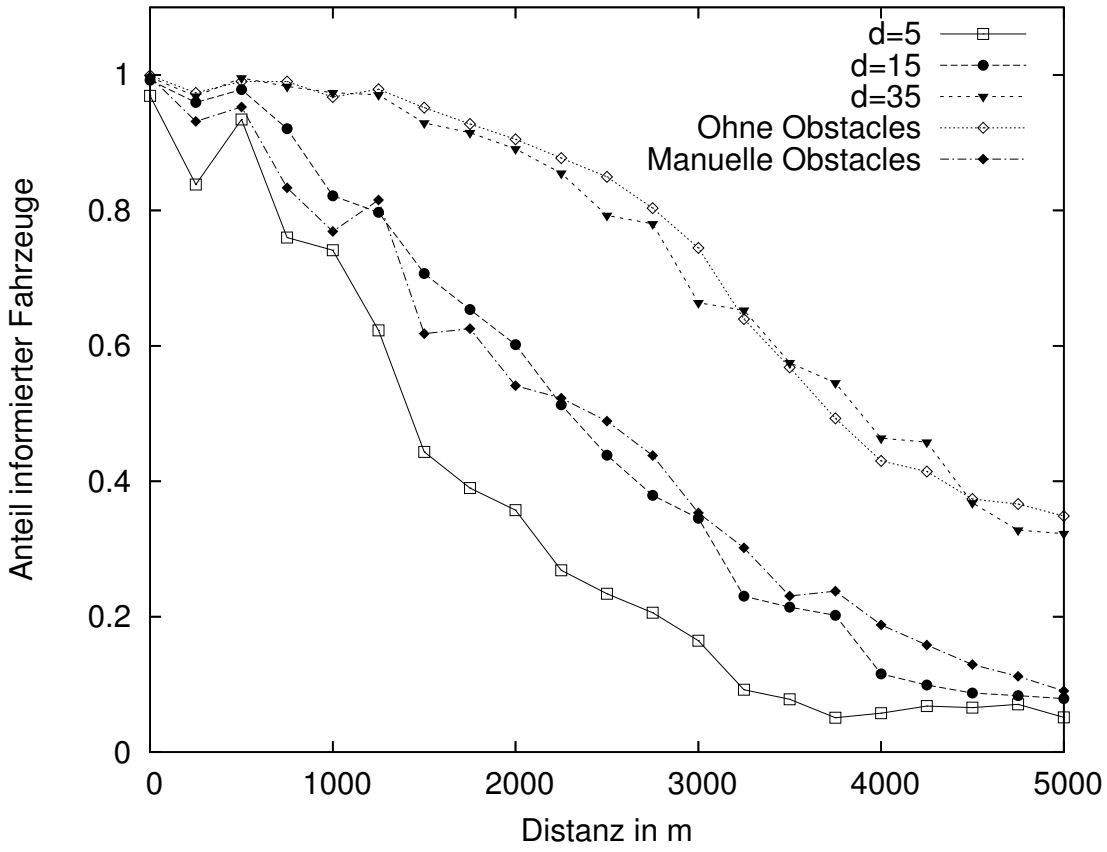


Abbildung 4.1: Verteilung der Information in Abhängigkeit zur Entfernung zu *P*

an dem das Ereignis stattgefunden hat, verhalten. Es ist klar, dass durch die begrenzte Ausstattungsrate bei größerer Entfernung der Verteilungsgrad abnimmt. Die Variante ohne *obstacles* stellt den idealen Fall dar. Jede zu einer Simulation zugehörige Kurve, die stärker gestaucht ist, beschreibt also eine Simulation, bei der Informationen schneller verloren gehen. Bei Verwendung von Hindernissen ist dies möglicherweise ein Hinweis dafür, dass hier das *obstacle modelling* funktioniert.

Es fällt auf, dass die Ergebnisse des Sichtlinien-Algorithmus bei unterschiedlich gewählten Werten für die Konstante *d* stark von einander abweichen. Dies war allerdings auch zu erwarten, denn *d* definiert schließlich den Bereich der *obstacles*. Bei niedrig gewählten *d* verhält sich der Algorithmus demzufolge mindestens genauso gut wie die manuelle Variante.

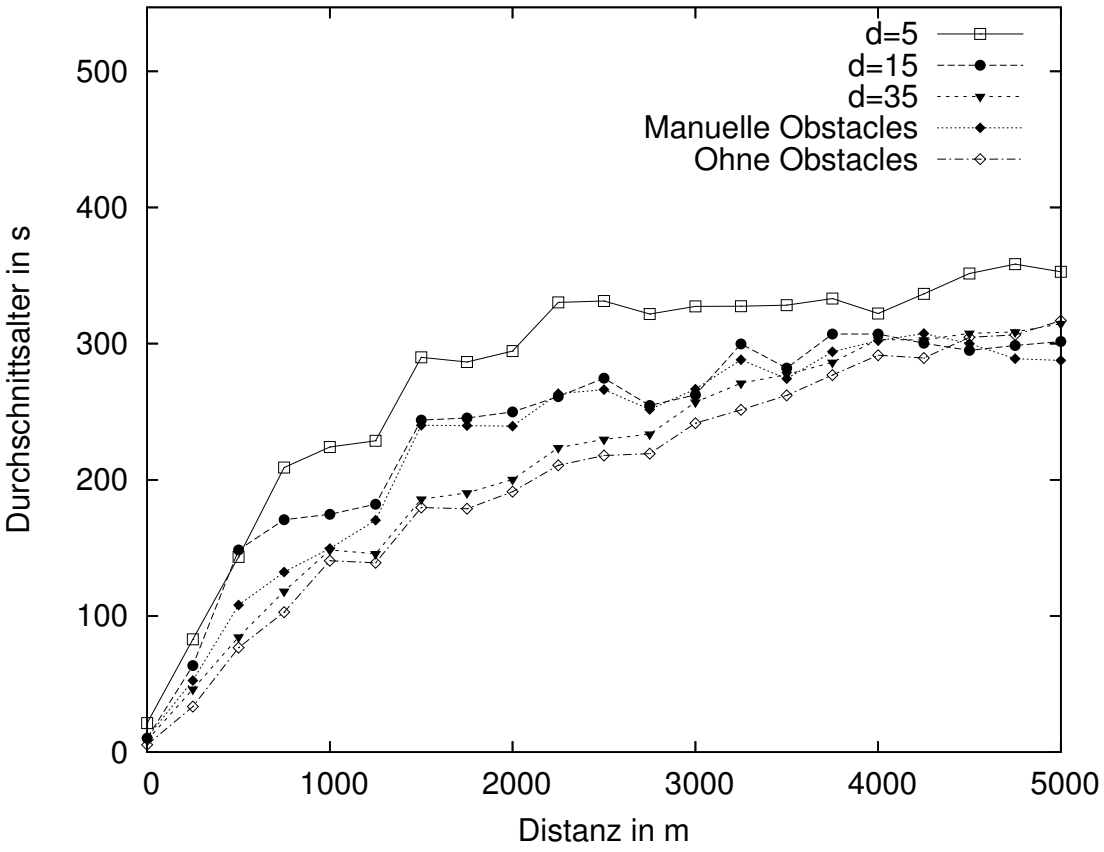


Abbildung 4.2: Alter der Information in Abhängigkeit zur Entfernung zu *P*

Abbildung 4.2 stellt graphisch dar, wie sich die Informationen im Durchschnitt über alle Fahrzeuge im Szenario in Abhängigkeit zu ihrem Alter verbreiten. Hierbei ist es einsichtig, dass bei zunehmender Entfernung zu *P* das Alter der zuletzt erhaltenen Information zunimmt. Dadurch ergibt sich stets eine steigende Kurve, wie anhand der idealen Variante ersichtlich. Je steiler die Kurve verläuft, desto älter sind also die Informationen über das Ereignis. Dies ist widerum eine Möglichkeit zu erkennen, ob das *obstacle modelling* im Szenario funktioniert.

Auch hier ist erkennbar, dass sich der Sichtlinien-Algorithmus gut verhält – auch wenn er teilweise nur marginal von der Variante ohne Hindernisse abweicht.

Kapitel 5

Zusammenfassung und Ausblick

Innerhalb dieser Bachelorarbeit wurde ein Algorithmus entwickelt, der in der Lage ist, den Realitätsgrad der Simulation eines Fahrzeug-zu-Fahrzeug Szenarios zu verbessern. Mit Hilfe von Straßendaten ist er nährungsweise in der Lage *obstacles* zu modellieren. Dazu wird das Szenario in zwei Teilbereiche unterteilt. Rund um alle Strecken des Straßenetzes befindet sich der Kommunikationsbereich, in dem es möglich ist, Funksignale untereinander auszutauschen. Der zweiten Bereich – das Komplement des Kommunikationsbereichs – ist nach Definition die Fläche, in der sich Hindernisse befinden, die einen Datenaustausch zwischen Fahrzeugen unmöglich machen.

Ausgehend von dieser Vorraussetzung dürfen innerhalb der Simulation zwei Fahrzeugen miteinander kommunizieren, wenn sich die Sichtlinie vollständig im Kommunikationsbereich befindet. Anderenfalls kann davon ausgegangen werden, dass sich ein Hindernis zwischen den Fahrzeugen befindet, das die Kommunikation stören bzw. das Signal abschwächen kann. Für diesen Fall wird dem Simulator der Kommunikationsaustausch untersagt.

Da der Algorithmus während der Laufzeit nicht nur einmal, sondern für jeden Kommunikationsschritt erneut ausgeführt wird, ist es ratsam, den rechnerischen Aufwand so gering wie möglich zu halten. Das Ziel nachfolgender Arbeiten kann es sein, diesen Aufwand

weiter herabzusenken.

In der aktuellen Implementierung des Algorithmus wird für jede Überprüfung eines Abschnittes innerhalb des Algorithmus die komplette Liste der Straßen mit all ihren Abschnitten durchlaufen. Dabei reicht es eigentlich, dieses Verfahren auf alle Straßen zu beschränken, die im direkten Umfeld der aktuell betrachteten Strecke – also innerhalb eines Kreises mit Radius *d* um den aktuell betrachteten Fortschrittspunkt – liegen.

Um dies zu erreichen, kann man eine Speicherstruktur für die Straßen verwenden, bei der Informationen einzelnen Quadranten zugeteilt werden können. Dadurch ergibt sich eine hierarchische Struktur, in der man zweidimensionale Koordinaten effizient suchen kann. Diese Bedingungen erfüllt beispielsweise ein Quadtree [Sam84]. Hierbei wird ein Gebiet in vier gleich große Abschnitte unterteilt. Diese können auf dieselbe Weise rekursiv weiter unterteilt werden. Der Grad der Genauigkeit, den ein einzelnes Element ausdrücken kann, wird dabei durch die Feinheit der Auflösung der Gebiete bestimmt.

In einem solchen Quadtree könnte man dann im ersten Anlauf festlegen, in welchem Stadtteil sich eine Straße befindet, und anschließend im Algorithmus nur Straßen des entsprechenden Stadtteils durchlaufen. Für den Fall, dass man einen Quadtree mit wesentlich feineren Abschnitten vorliegen hat, kann man im zweiten Anlauf den Aufwand weiter senken, indem man nur Straßen verwendet, die innerhalb derselben Zelle oder den benachbarten Zellen liegen. Alternativ kann man auch auf eine Vielzahl anderer Speicherstrukturen zurückgreifen. Andere Speicherstrukturen, die ähnliches erfüllen können, sind ein Hash-Baum [FNPS79] oder ein R-Baum [BKSS90].

Eine weitere mögliche Verbesserung ist bei der Definition des Kommunikationsbereiches möglich. Momentan sind alle Straßen für den Algorithmus gleich breit. Ihre Breite wird dabei nur durch den Straßenabstandsparameter *d* definiert. In einer gewöhnlichen Umgebung kommen jedoch viele verschiedene Straßentypen zum Einsatz. So kann beispielsweise eine kleine Seitenstraße einspurig, eine Stadtautobahn hingegen vierspurig in jeder Richtung sein. Um die Ergebnisse der Simulation noch weiter verbessern zu können, sollten deshalb die unterschiedlichen Spurbreiten berücksichtigt werden. Die In-

formationen könnten ebenfalls aus den Szenariodateien vor der Laufzeit des Programms ausgelesen werden und Einfluss auf die Erstellung der Rechtecke um die jeweiligen Streckenabschnitte der Straßen haben.

Es ist klar, dass durch umfangreiche und vor allem realistische Simulationen die Technik der Fahrzeug-zu-Fahrzeug-Kommunikation stetig verbessert werden kann. Je genauer dabei die Ergebnisse sind, desto bessere Schlüsse lassen sich daraus ziehen. Der im Rahmen dieser Bachelorarbeit entwickelte Algorithmus trägt einen Schritt in diese Richtung bei, wie die Simulationsergebnisse verdeutlicht haben. Der Algorithmus verhält sich beim *obstacle modelling* mindestens genauso gut wie die Variante, bei der *obstacles* manuell erstellt wurden. Er bietet jedoch im Vergleich zu diesem zwei Vorteile. Er ermöglicht durch den Einsatz der frei wählbaren Konstante *d* eine größere Dynamik bei der *obstacle*-Modellierung. Desweiteren ist er im Gegensatz zur manuellen Variante, bei der die *obstacles* erst von Hand definiert werden müssen, mit wenig Aufwand direkt in völlig neuen Szenarien einsetzbar.

Es ist zu erwarten, dass die Technik in einigen Jahren so weit entwickelt sein wird, dass Fahrerassistenzsysteme helfen können, die Sicherheit im Straßenverkehr stark zu verbessern. Bis dahin ist es jedoch noch ein weiter, aber sicherlich spannender Weg.

Literaturverzeichnis

[ASSC02] AKYILDIZ, I., W. SU, Y. SANKARASUBRAMANIAM und E. CAYIRCI: *Wireless sensor networks: a survey*. Elsevier Computer Networks, 38:393–422, March 2002.

[BKSS90] BECKMANN, NORBERT, HANS-PETER KRIEGEL, RALF SCHNEIDER und BERNHARD SEEGER: *The R*-tree: an efficient and robust access method for points and rectangles*. In: *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, Seiten 322–331, New York, NY, USA, 1990. ACM Press.

[BMW03] BMW GROUP PRESSEABTEILUNG: *Weniger Stau durch sprechende Autos*. http://www.presseportal.de/story.htx?nr=484485, September 2003.

[Car] *Car2Car Communication Consortium*. http://www.car-2-car.org/.

[Dai] DAIMLERCHRYSLER AG. http://www.et2.tu-harburg.de/fleetnet/.

[Fes05] FESTAG, ANDREAS: *Wifi fürs Auto*. Funkschau, Seite 45, 2005.

[FNPS79] FAGIN, RONALD, JURG NIEVERGELT, NICHOLAS PIPPENGER und H. RAYMOND STRONG: *Extendible hashinga fast access method for dynamic files*. ACM Trans. Database Syst., 4(3):315–344, 1979.

[Fre05a] FREE SOFTWARE FOUNDATION (FSF): *Bison*. http://www.gnu.org/software/bison/, 2005.

[Fre05b] FREE SOFTWARE FOUNDATION (FSF): *Flex*. http://www.gnu.org/software/flex/, 2005.

[GNU] *The GNU Project*. http://www.gnu.org/gnu/thegnuproject.html.

[IEE97] IEEE COMPUTER SOCIETY LAN MAN STANDARDS COMMITTEE: *Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*. The Institute of Electrical and Electronic Engineers, New York, NY, 1997.

[IET04] IETF MANET WORKING GROUP: *The Dynamic Source Routing Protocol (DSR)*. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt, Jule 2004.

[Ins04] INSTITUT FÜR INFORMATIK DER UNIVERSITÄT ZÜRICH: *Bottom-Up Parsing*. www.ifi.unizh.ch/stff/siclemat/lehre/ws0405/pcl1/bu_shift.pdf, 2004.

[Kar] KARWEG, BASTIAN: *Playstation Portable Magazin*. http://www.playstationportable.de/.

[KS01] KOSCH, TIMO und CHRISTAN SCHWINGENSCHLÖGL: *Mobile Ad-Hoc Networking Testbed for Vehicle-to-Vehicle Communication*. In: *5.th International Conference on Systemics, Cybernetics and Informatics*, 2001.

[LCS+05] LOCHERT, CHRISTIAN, MURAT CALISKAN, BJÖRN SCHEUERMANN, ANDREAS BARTHELS, ALFONSO CERVANTES und MARTIN MAUVE: *Multiple Simulator Interlinking Environment for Inter Vehicle Communication*. In: *The Second ACM International Workshop on Vehicular Ad Hoc Networks (VANET 2005)*, Seiten 87–88, Cologne, Germany, September 2005.

[LHT+03] LOCHERT, CHRISTIAN, HANNES HARTENSTEIN, JING TIAN, HOLGER FÜSSLER, DAGMAR HERRMANN und MARTIN MAUVE: *A Routing Strategy for Vehicular Ad Hoc Networks in City Environments*. In: *Proc. of IEEE Intelligent Vehicles Symposium (IV2003)*, Seiten 156–161, Columbus, OH, June 2003.

[MFCO00] MOEN, BRIAN, JOEL FITTS, DAVE CARTER und YANG OUYANG: *A Comparison of the VISSIM and CORSIM Traffic Simulation Models*. Technischer Bericht, Institute of Transportation Engineers, Dallas, TX, USA, 2000. http://www.english.ptv.de/download/traffic/library/2000VISSIM-CORSIM.pdf.

[NOW] *Network on wheels (NOW)*. http://www.network-on-wheels.de/.

[ns2] *The Network Simulator - ns-2*. http://www.isi.edu/nsnam/ns.

[OPN] OPNET TECHNOLOGIES: *OPNET*. http://www.opnet.com.

[PB94] PERKINS, CHARLES E. und PRAVIN BHAGWAT: *Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers*. In: *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, Seiten 234–244, New York, NY, USA, 1994. ACM Press.

[PBR03] PERKINS, C. und E. BELDING-ROYER: *Ad hoc On-Demand Distance Vector (AODV) Routing*. July 2003. RFC 3561.

[PC97] PARK, VINCENT D. und M. SCOTT CORSON: *A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*. In: *IEEE Conference on Computer Communications, INFOCOM'97, April 7-11, 1997, Kobe, Japan*, Band 3, Seiten 1405–1413. IEEE, IEEE, April 1997.

[PTV04] PTV PLANUNG TRANPORT VERKEHR AG: *Benutzerhandbuch VISSIM 4.0*, June 2004.

[PTV05] PTV PLANUNG TRANPORT VERKEHR AG: *ptv simulation - VISSIM*. http://www.ptv.de/, 2005.

[Ril03] RILEY, GEORGE F.: *The Georgia Tech Network Simulator*. In: *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, Seiten 5–12, New York, NY, USA, 2003. ACM Press.

[RK98] ROME, ERICH und MARINA KOLESNIK: *3D-Szenenrekonstruktion aus Bilddaten*. Jule 1998. http://www.ais.fraunhofer.de/projects/Makro/reports/makro-report-VI-1.pdf.

[Rou] ROUTERSIM: *CCNA Network Visualizer 5.0*. http://www.routersim.com/.

[Sam84] SAMET, HANAN: *The Quadtree and Related Hierarchical Data Structures*. ACM Comput. Surv., 16(2):187–260, 1984.

[Sim02] SIMREAL INC. http://nsl10.csie.nctu.edu.tw/, March 2002.

[Wie74] WIEDEMANN, R.: *Simulation des Straßenverkehrsflußes*. Schriftenreihe des Instituts für Verkehrswesen der Universität Karlsruhe, 8, 1974.

[Wik] WIKIPEDIA: *Mehrwegeausbreitung*. http://de.wikipedia.org/wiki/Mehrwegeausbreitung. Artikelrevision vom 2. Februar 2006.

[ZBG98] ZENG, XIANG, RAJIVE BAGRODIA und MARIO GERLA: *GloMoSim: a library for parallel simulation of large-scale wireless networks*. In: *PADS '98: Proceedings of the twelfth workshop on Parallel and distributed simulation*, Seiten 154–161, Washington, DC, USA, 1998. IEEE Computer Society.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, den 6. Februar 2006 Markus Michel