

FAKULTÄT FÜR INFORMATIK

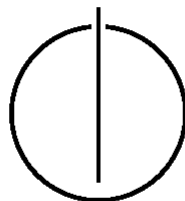
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

**Konzeption und Implementierung einer
C2X-basierten dynamischen Routenführung
für die mikroskopische Verkehrssimulation VISSIM**

**Design and implementation of a
V2X-based dynamic routing assistance
for the microscopic traffic simulation VISSIM**

Bearbeiter: Michael Müller
Aufgabensteller: Univ.-Prof. Dr. Hans-Joachim Bungartz
Betreuer: Dipl.-Inf. Dirk Pflüger
Dipl.-Inf. Mathias Baur
Abgabedatum: 15. September 2010



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. September 2010

Michael Müller

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Mein besonderer Dank gilt meinem externen Betreuer vom Lehrstuhl für Verkehrstechnik der Technischen Universität München, Herr Mathias Baur, der mir die Einarbeitung in das Thema erleichterte und mir mit hilfreichen Anregungen zur Seite stand.

Darüber hinaus bedanke ich mich bei Herrn Dirk Pflüger, der mich als Fachbetreuer mit wertvollen Hinweisen und konstruktiver Kritik unterstützte.

Weiterhin sei Herrn Matthew Fullerton für seine Hilfeleistungen im technischen Bereich sowie bei der Erstellung von Auswertungsdiagrammen gedankt.

Kurzfassung

Die Bundesrepublik Deutschland verfügt über ein 231.000 Kilometer langes, gut ausgebautes überörtliches Straßennetz (Stand 01.01.2010). Der zunehmende Individualverkehr infolge der voranschreitenden Globalisierung und Mobilität der Gesellschaft führt dazu, dass dieses Netz immer häufiger überlastet ist. Folglich bilden sich Staus, die nicht nur ein großes volkswirtschaftliches Problem darstellen, sondern auch das Risiko für Unfälle deutlich erhöhen.

Das Forschungsprojekt „Sichere Intelligente Mobilität – Testfeld Deutschland“ (sim^{TD}), welches von den Bundesministerien für Bildung und Forschung, Wirtschaft und Technologie sowie Verkehr, Bau und Stadtentwicklung durchgeführt wird, hat das Ziel die Effizienz und Sicherheit des Individualverkehrs in Deutschland zu verbessern. Grundlage hierfür ist der Einsatz von Car-to-X-Kommunikation (C2X-Kommunikation). Diese ermöglicht es unter anderem, Verkehrsteilnehmer frühzeitig auf mögliche Gefahren oder Staus hinzuweisen, damit diese entsprechend reagieren können. Die sim^{TD} -Funktion der C2X-basierten dynamischen Routenführung stellt dabei eine wesentliche Rolle dar. Sie ermöglicht eine umfassende Analyse der aktuellen Verkehrslage und bietet den Verkehrsteilnehmern auf deren Basis eine optimale Ziel-führung.

Mit der vorliegenden Arbeit wurde ein Beitrag für sim^{TD} geleistet, indem ein Programm entwickelt wurde, welches die dynamische Routenführung auf Basis der C2X-Kommunikation in die mikroskopische Verkehrssimulation VISSIM der Firma PTV integriert. Der Lehrstuhl für Verkehrstechnik der Technischen Universität München hat als Forschungspartner von sim^{TD} unter anderem die Aufgabe, die Auswirkungen der Nutzung der C2X-Technologie zu erfassen. Hierfür werden unter Verwendung von PTV VISSIM umfangreiche Verkehrssimulationen durchgeführt. Mithilfe des im Rahmen dieser Arbeit erstellten Programms kann der großflächige Einsatz der C2X-basierten dynamischen Routenführung als eine der sim^{TD} -Funktionen simuliert werden.

Inhaltsverzeichnis

Danksagung	3
Kurzfassung	4
Inhaltsverzeichnis	5
1 Einleitung	7
2 Grundlagen und Forschungsbezug	9
2.1 Sichere Intelligente Mobilität – Testfeld Deutschland (sim ^{TD})	9
2.1.1 Allgemeiner Überblick	9
2.1.2 Auswahl und Ausrüstung des Testgebiets	9
2.1.3 Vision	10
2.2 Verkehrssimulation	12
2.2.1 Anwendung von Verkehrssimulationen	12
2.2.2 Simulationsmodellansätze	12
2.2.3 Mikroskopische Verkehrssimulation	13
2.3 PTV VISSIM	14
2.3.1 Allgemeiner Überblick	14
2.3.2 VISSIM-Schnittstellen	14
3 Zielsetzung und Aufgabenstellungen	16
4 Grundlegende Konzepte und Implementierungsansätze	18
4.1 Wahl eines geeigneten VISSIM-Netzes	18
4.2 Erzeugung eines gerichteten Graphen auf Basis des VISSIM-Netzes	19
4.2.1 Allgemeines Verfahren zur Grapherzeugung	19
4.2.2 Optimierung des Netzgraphen	21
4.3 Verkehrslage- und Reisezeiterfassung	22
4.3.1 Allgemeine Vorgehensweise	22
4.3.2 Ermittlung der Verkehrslage	23
4.3.3 Ermittlung der Reisezeiten und deren Überführung in Kantengewichte des Netzgraphen	23
4.4 Ermittlung von Alternativrouten	24
4.5 Übertragen von Alternativrouten auf das VISSIM-Netz als statische Routen ..	25
4.6 Beeinflussung des Routingverhaltens einzelner Fahrzeuge	25
4.6.1 Individuelles Routing	25
4.6.2 Kollektives Routing	26
5 Dokumentation	27
5.1 Zusatzprogramme	27
5.1.1 Erzeugen des Netzgraphen	27
5.1.2 Erzeugen eines gewichteten Netzgraphen	28
5.1.3 Ermitteln und Speichern aller Alternativrouten	28
5.2 Hauptprogramm	29
5.2.1 TrafficAnalysis.py – Basisklasse für das dynamische Routing	29
5.2.2 NetGraph.py – Graphrepräsentation	30
5.2.3 AvgTravelTime.py – Bestimmung der mittleren Reisezeit	30
5.2.4 AvgQueue.py – Ermittlung der mittleren Geschwindigkeit und Fahrzeuganzahl	31

5.2.5	DynamicRouter.py – Umsetzung der dynamischen Routenführung	31
5.2.6	Route.py – Routenrepräsentation im Netzgraph.....	32
6	Durchführung der Verkehrssimulation mit C2X-basierter dynamischer Routenführung	33
6.1	Szenario-abhängige Konfiguration der Simulationsparameter.....	33
6.1.1	Räumliche und zeitliche Eingrenzung des Untersuchungsgebiets	33
6.1.2	Wahl der Szenarien.....	33
6.2	Wahl der Messgrößen	35
6.2.1	Mikroskopische Messgrößen	35
6.2.2	Makroskopische Messgrößen	35
7	Ergebnisse der Simulationsstudie	39
7.1	Vergleich der Szenarien „dichter Verkehr“ und „freies Fahren“	39
7.2	Vergleich der Szenarien mit und ohne dynamischer Routenführung.....	41
8	Zusammenfassung und Ausblick	42
	Abbildungsverzeichnis.....	43
	Tabellenverzeichnis	44
	Literaturverzeichnis	45
	Anhang.....	46
I	Detaillierte Dokumentation der Implementierung.....	46
II	Zusätzliche Abbildungen	57
III	CD mit Programmquelltexten.....	57

1 Einleitung

Mit zunehmender Globalisierung und Mobilität der Gesellschaft geht unweigerlich auch ein Wachstum des Verkehrsaufkommens einher. Selbst das gut ausgebaute Verkehrsnetz der Bundesrepublik Deutschland ist ohne aufwändige Neubaustrecken mit dieser Situation immer häufiger überfordert. So lassen sich Staubildung und damit verbundene Verzögerungen nicht verhindern. Staus sind allerdings nicht nur lästige Begleiterscheinungen des täglichen Individualverkehrs, sondern verursachen auch enorme volkswirtschaftliche Kosten. Nach Schätzungen der EU-Kommission liegen diese in Deutschland bei jährlich rund 17 Milliarden Euro. Weiterhin steigt mit zunehmender Verkehrsdichte auch das Unfallrisiko. So liegt die Zahl der bei Verkehrsunfällen verletzten bzw. getöteten Personen laut Statistischem Bundesamt trotz neuer Sicherheits- und Assistenzsysteme immer noch bei jährlich rund 420.000 bzw. 5.000.

Eine vielversprechende Technologie zur Verbesserung der Verkehrseffizienz und der Sicherheit stellt die Car-to-X-Kommunikation (C2X-Kommunikation) dar. Mithilfe von C2X-Kommunikation können umfangreiche Verkehrslagedaten zeitnah an die Verkehrszentralen übermittelt werden, sodass sich ein realitätsnahes Bild der aktuellen Verkehrssituation erstellen lässt. Davon ausgehend können Informationen gezielt an Verkehrsteilnehmer weitergegeben werden, um diese beispielsweise auf einen bevorstehenden Stau hinzuweisen.

Um die C2X-Technologie voranzutreiben, führt die Bundesrepublik Deutschland, vertreten durch die Bundesministerien für Bildung und Forschung sowie Wirtschaft und Technologie und unterstützt durch das Bundesministerium für Verkehr, Bau und Stadtentwicklung, das Forschungsprojekt „Sichere Intelligente Mobilität - Testfeld Deutschland“ (sim^{TD}, vgl. 2.1) durch. Vordergründiges Ziel von sim^{TD} ist es, das Fahren schneller und gleichzeitig sicherer und angenehmer zu gestalten. Das heißt, Verkehrshindernisse und Gefahren sollen schneller erkannt und das Verkehrsmanagement optimiert werden. In Kooperation mit vielen Partnern aus Industrie und Forschung werden dazu sämtliche Funktionen aus den verschiedenen Anwendungsgebieten von C2X-Kommunikation vereint und in einem breit angelegten Feldversuch unter realen Verkehrsbedingungen getestet.

Die Technische Universität München ist Forschungspartner von sim^{TD}. Im Rahmen dieses Projekts ist der Lehrstuhl für Verkehrstechnik unter anderem für die Erfassung von verkehrlichen Auswirkungen verantwortlich, die sich aus der Nutzung von C2X-Kommunikation ergeben. Mithilfe der mikroskopischen Verkehrsflusssimulation VISSIM (vgl. 2.3) von dem Beratungs- und Softwareunternehmen PTV AG sollen hierfür umfangreiche Simulationsstudien durchgeführt werden. VISSIM bietet diesbezüglich eine spezielle C2X-Schnittstelle an, die es ermöglicht, mithilfe eines Python-Scripts auch C2X-Kommunikation in die Simulation einzubeziehen.

Ziel der vorliegenden Bachelorarbeit ist es, eine VISSIM-Erweiterung zu entwickeln, welche Verkehrssimulationen mit dynamischer Routenführung ermöglicht. Insbesondere sollen durch die Verwendung von C2X-Kommunikation hochaktuelle Verkehrslageinformationen die Grundlage für die Berechnung einer optimalen Route bilden. Für die Erfassung der nötigen Daten ist die VISSIM C2X-Schnittstelle zu verwenden. Damit einher geht die Verwendung von Python als Programmiersprache.

Zunächst werden das Projekt sim^{TD} und nach einer kurzen Einführung in die Theorie der Verkehrssimulation das Simulationstool VISSIM vorgestellt. Dabei wird das Hauptaugenmerk auf dem Bezug zu dieser Arbeit liegen. Anschließend werden im Kapitel 3 die Aufgaben und Ziele der Arbeit genauer spezifiziert. Abschnitt 4 gibt einen Überblick über die grundlegenden Konzepte und Implementierungsansätze, die zur Lösung der Aufgaben dienen. Die Dokumentation der umgesetzten Python-

Implementierung folgt im Kapitel 5. Den zweiten Teil der Arbeit wird eine kurze Simulationsstudie bilden. Sie soll zeigen welchen Einfluss die C2X-basierte dynamische Routenführung auf den Verkehrsfluss und damit auf die Verkehrseffizienz hat. Die Versuchsdurchführung wird im Abschnitt 6 erläutert, während im darauffolgenden Kapitel die Ergebnisse vorgestellt werden. Abschließend werden eine kurze Zusammenfassung sowie ein kurzer Ausblick gegeben.

2 Grundlagen und Forschungsbezug

2.1 Sichere Intelligente Mobilität – Testfeld Deutschland (sim^{TD})

2.1.1 Allgemeiner Überblick

„Das Forschungsprojekt sim^{TD} gestaltet durch die Erforschung und Erprobung der Car-to-X-Kommunikation und ihrer Anwendungen die sichere und intelligente Mobilität von morgen.“[1] Mit diesem ehrgeizigen Ziel haben sich führende deutsche Automobilhersteller, Automobilzulieferer, Kommunikationsunternehmen und Forschungsinstitute unter Leitung der Daimler AG zu einem Konsortium zusammengeschlossen. Der Name sim^{TD} steht für „Sichere Intelligente Mobilität – Testfeld Deutschland“. Unterstützt und gefördert wird dieses Forschungsprojekt durch die Bundesministerien für Wirtschaft und Technologie, Bildung und Forschung, Verkehr, Bau und Stadtentwicklung sowie das Land Hessen und die Stadt Frankfurt. Die Fördersumme beträgt ca. 30 Millionen Euro, sodass sich das Gesamtbudget auf ca. 53 Millionen Euro beläuft. Das seit 01. September 2008 laufende Projekt soll nach vierjähriger Laufzeit am 31. August 2012 abgeschlossen werden.

In der Vergangenheit wurden in vielen nationalen und internationalen C2X-Projekten (z.B. AKTIV, COM2REACT, PRE-DRIVE C2X) einzelne Funktionen der C2X-Kommunikation erforscht und zum Teil unter vereinfachten Bedingungen getestet. Darum gilt es nun, mit sim^{TD} die Technologie der Fahrzeug-zu-Fahrzeug- bzw. Fahrzeug-zu-Infrastruktur-Kommunikation in einem großflächigen Feldtest rund um die hessische Metropole Frankfurt zur Anwendung zu bringen und eine Markteinführung vorzubereiten. Dazu gehört auch die Integration von Mehrwertdiensten, welche die spätere Marktdurchdringung beschleunigen soll.

2.1.2 Auswahl und Ausrüstung des Testgebiets

Das Testgebiet (vgl. Abb. 1) erstreckt sich auf einer Länge von etwa 40 Kilometer und einer Breite von etwa 15 Kilometer rund um Frankfurt am Main und beinhaltet sowohl Autobahn- und Landstraßen, als auch Strecken im städtischen Bereich. Es wurde aufgrund seiner Rolle als wichtige Verkehrsdrehscheibe in Deutschland und seiner gut ausgebauten Infrastruktur für eine praxisnahe Erprobung als besonders geeignet befunden.

Für den Versuch kommen 100 ITS¹ Roadside Stations zum Einsatz, welche die Verbindung zwischen den bis zu 400 Testfahrzeugen und der Verkehrszentrale Hessen bzw. der Integrierten Gesamtverkehrsleitzentrale Frankfurt herstellen sollen. Dabei kommt der IEEE² WLAN-Standard 802.11p zum Einsatz. Dieser ermöglicht auch das sogenannte Multihopping, bei dem Daten indirekt, also über andere Fahrzeuge, an den Kommunikationspartner übermittelt werden. Zusätzlich sollen Mobilfunktechnologien wie GSM und UMTS eventuell auftretende Versorgungslücken schließen.

¹ ITS – Intelligent Transport Systems

² IEEE – Institute of Electrical and Electronics Engineers

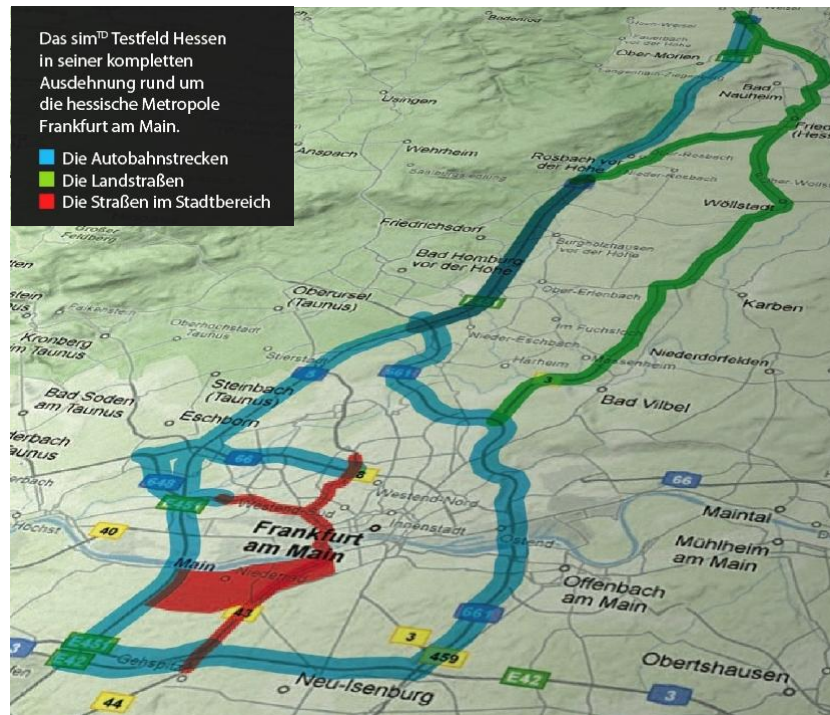


Abb. 1: Testgebiet sim^{TD} [2]

2.1.3 Vision

Das Ziel von sim^{TD}, die sichere und intelligente Mobilität von morgen zu gestalten, lässt sich in die folgenden drei Visionen gliedern:

1. Unfallfreies Fahren
2. Effizienteres Fahren
3. Vernetztes mobiles Leben

Trotz vieler aktiver und passiver Sicherheitssysteme kommt es immer wieder zu Unfällen, die mit vorausschauender Fahrweise oder besserem Informationsmanagement vermieden werden könnten. Beispielsweise bilden verdeckte Stauenden oder liegengebliebene Fahrzeuge ein erhöhtes Sicherheitsrisiko. Zwar können aktuelle Fahrerassistenzsysteme das nähere Umfeld des Fahrzeuges schon relativ gut erfassen und den Fahrer auf mögliche Kollisionen aufmerksam machen, doch gibt es Situationen, bei denen diese Systeme nicht ausreichen. So soll die Vernetzung von Fahrzeugen durch C2X-Kommunikation dazu führen, dass Gefahren noch früher erkannt und rechtzeitig entsprechende Gegenmaßnahmen ergriffen werden können. In der Praxis bedeutet dies, dass beispielsweise stark bremsende oder stehende Fahrzeuge ein entsprechendes Warnsignal an alle nachfolgenden Fahrzeuge schicken, welche die Information per Multihopping weiterreichen können. Über die Roadside Stations gelangt diese Information auch in die Verkehrszentrale, die zum Beispiel dynamische Verkehrszeichen entsprechend der Situation steuern und gegebenenfalls auch Rettungskräfte alarmieren kann. So können die direkten und indirekten Folgen eines Unfalls durch schnelle Hilfe und Vermeidung von Auffahrunfällen vermindert werden. Abb. 2 verdeutlicht dieses Szenario.

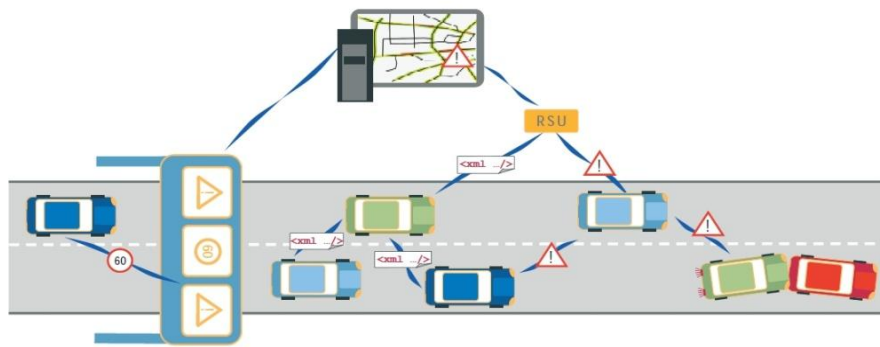


Abb. 2: Einsatz von C2X-Kommunikation bei einem Verkehrsunfall [3]

Die zweite Vision befasst sich mit der Verkehrseffizienz. Schon heute ist es eine große Herausforderung, die vorhandenen Verkehrswege optimal zu nutzen und dabei Staus zu verhindern. In Zukunft wird dies aufgrund des zunehmenden Verkehrsaufkommens immer schwieriger werden. Dabei spielen nicht nur ökonomische, sondern auch ökologische Aspekte eine Rolle. Um diesen Aufgaben zu begegnen, ist ein umfangreiches Verkehrsmanagement nötig.

Eine bereits existierende Möglichkeit zur Verbesserung des Verkehrsflusses ist die dynamische Routenführung, also das Anpassen der Streckenführung an die aktuelle Verkehrssituation, um möglichst schnell an das gewünschte Reiseziel zu gelangen. Dabei soll nach den sim^{TD} -Spezifikationen F_1.2.3 [4] und F_1.3.1 [5] zwischen kollektivem und individuellem Routing unterschieden werden. Bei ersterem ist die Verkehrszentrale dafür zuständig, alle Fahrzeuge auf Basis der Verkehrssituation und der Streckenkapazitäten zu routen und im Falle eines Staus den Verkehrsteilnehmern eine Auswahl möglicher Alternativrouten vorzuschlagen. Die Fahrer können sich dann für eine Option entscheiden oder auf der ursprünglichen Route verbleiben. Beim individuellen Routing hingegen ist die Verkehrszentrale nur für die Weitergabe der Verkehrslagedaten verantwortlich. Das eigentliche Routing findet in den Fahrzeugen statt. So können auch die einzelnen Vorlieben der Fahrer, wie zum Beispiel das Meiden von Autobahnen, bei der Navigation berücksichtigt werden.

Zu den Zielen von sim^{TD} zählt unter anderem die Verbesserung dieser Funktionen mithilfe von C2X-Kommunikation. Wichtigste Voraussetzung dafür sind umfangreiche und aktuelle Datenbestände. Heute werden meist noch stationäre Systeme oder visuelle Beobachtungen zur Datenerfassung verwendet. Aber auch die Verkehrsbeeinflussung ist auf schnelle Kommunikationswege angewiesen. Neben den dynamischen Verkehrszeichen ist der Verkehrsfunk derzeit das Haupthilfsmittel für die Informationsweitergabe an die Fahrzeuge. Desweiteren sind die meisten Navigationssysteme bereits mit dynamischer Routenführung ausgestattet, jedoch werden die nötigen Daten nur über den schmalbandigen und damit langsamen Traffic Message Channel (TMC) übermittelt.

Die C2X-Kommunikation ermöglicht es hingegen, die hochaktuellen Daten einzelner Fahrzeuge direkt in die Verkehrslageerfassung mit einzubeziehen. So können beispielsweise verunglückte Fahrzeuge über C2X-Kommunikation auf sich aufmerksam machen und so frühzeitig einen aufkommenden Stau ankündigen. Auch die häufigen Staus durch hohes Verkehrsaufkommen lassen sich zum Beispiel durch Übermittlung von Geschwindigkeiten und Reisezeiten der Fahrzeuge leichter identifizieren. Mithilfe dieser Daten liefert die dynamische Routenführung – sowohl kollektiv, als auch individuell – deutlich bessere Ergebnisse. Durch C2X-Kommunikation lässt sich jedes Fahr-

zeug zeitnah und individuell mit den relevanten Verkehrslageinformationen bzw. Alternativroutenvorschlägen versorgen.

Nicht zuletzt soll auch das mobile Leben mithilfe der C2X-Kommunikation vereinfacht und verbessert werden. Durch die Vernetzung von Fahrzeugen mit ortsfester Infrastruktur lassen sich beispielsweise Mehrwertdienste, wie die Lokalisierung der nächsten Tankstelle oder Parkmöglichkeit, integrieren. Aber neben den relativ statischen Informationen zu Sehenswürdigkeiten oder Veranstaltungen können nun auch Multimediatdaten vom Fahrzeug aus abgerufen werden. Somit stellt Mobilität in Zukunft im Bezug auf Kommunikation kaum noch eine Einschränkung dar.

2.2 Verkehrssimulation

2.2.1 Anwendung von Verkehrssimulationen

Verkehrsprojekte wie sim^{TD} haben großen Einfluss auf die Abläufe in einem Verkehrsnetz. Um die Auswirkungen solcher Vorhaben im Vorfeld abklären zu können und einzelne Komponenten zu optimieren, hat sich in der Vergangenheit die rechnerbasierte Verkehrssimulation als ein sehr gutes Hilfsmittel herausgestellt. Weiterhin lassen sich so in kurzer Zeit und mit relativ geringem personellen und finanziellen Aufwand viele verschiedene Szenarien simulieren und auswerten. Für sim^{TD} stellt der Feldtest das zentrale Element im Bezug auf die Erprobung der C2X-Technologie dar. Jedoch kann solch ein breit angelegter Versuch nicht ohne sorgfältige Planung durchgeführt werden. Auch hierfür stellt die Verkehrssimulation ein mächtiges Werkzeug dar. So lassen sich Szenarien erstellen, die dann im praktischen Feldversuch genauer getestet werden können. Bei sim^{TD} wurde hierfür ein Simulationslabor eingerichtet, welches aus einem Fahrsimulator an der Julius-Maximilians-Universität Würzburg und der Verkehrssimulation an der Technischen Universität München besteht. Die Hauptaufgabe des Labors ist die Ermittlung möglicher Wirkungen der sim^{TD} -Funktionen auf Nutzerakzeptanz, Fahr- und Verkehrssicherheit sowie Fahr- und Verkehrseffizienz.

2.2.2 Simulationsmodellansätze

Aufgrund der vielen unbekannt Parameter in einem dynamischen Verkehrssystem und der damit verbundenen Komplexität, ist eine realistische Abbildung des Straßenverkehrs nicht denkbar. Vielmehr ist es sinnvoll, abstrahierende Modelle zu entwickeln, welche Verkehrsabläufe möglichst realitätsnah nachbilden. Es kommen dabei im Wesentlichen die nachfolgenden drei verschiedenen Ansätze zum Einsatz:

- Makroskopische Simulation
- Mikroskopische Simulation
- Mesoskopische Simulation

Die makroskopische Verkehrssimulation basiert auf der Annahme, dass sich die Fahrzeuge in einem Verkehrsnetz ähnlich verhalten, wie eine Flüssigkeit in einem Kanalsystem. Das heißt, es stehen nicht die einzelnen (diskreten) Fahrzeuge im Blickpunkt der Betrachtung, sondern die (kontinuierlichen) Eigenschaften des Fahrzeugkollektivs. Wichtige Größen zur Beschreibung eines solchen Systems sind die Geschwindigkeit, die Dichte und der Fluss des Verkehrs. Schon das Vorhandensein von statistischen Verteilungen dieser Größen ist bereits hinreichende Bedingung für die Modellierung des Systems.

In dieser Arbeit steht die mikroskopische Verkehrssimulation im Vordergrund (vgl. 2.2.3). Das Interesse gilt hierbei den einzelnen Verkehrsteilnehmer und deren genaue

Verhaltensweisen. Wie bei der makroskopischen Simulation bilden auch hier Geschwindigkeit, Fluss und Dichte des Verkehrs die Grundlage für die Darstellung des Modells. Allerdings werden diese nun für jedes Fahrzeug bzw. jeden Streckenabschnitt genau aufgelöst. Damit einher gehen Nachbildungen fahrzeugspezifischer Beschleunigungs- und Bremsvorgänge in Abhängigkeit der jeweiligen Verkehrssituation. Außerdem lassen sich Dichte und Fluss des Verkehrs mittels Detektoren durch Zählen der passierenden Fahrzeuge pro Zeitintervall bestimmen.

Eine Kombination aus mikroskopischer und makroskopischer Simulation bildet die mesoskopische Verkehrssimulation. Hierbei wird das Verhalten der Fahrzeuge mithilfe von Warteschlangen modelliert. Jedes Fahrzeug reiht sich entsprechend der individuellen Routenführung an Knotenpunkten in eine entsprechende Warteschlange ein. Nach einer von der Anzahl der Fahrzeuge und der Straßenkapazität abhängigen Wartezeit wird das Fahrzeug zum nächsten Knotenpunkt weitergeleitet. Anwendung findet dieser Ansatz zum Beispiel bei der Simulation von Ampelschaltungen.

2.2.3 Mikroskopische Verkehrssimulation

Die mikroskopische Verkehrssimulation ermöglicht eine detaillierte Nachbildung des Straßenverkehrs unter Berücksichtigung der Verhaltensweisen einzelner Verkehrsteilnehmer. Zur Abbildung der realen Verkehrsbewegungen gibt es verschiedene Modellansätze. Grundsätzlich gilt, dass Fahrzeugführer nur nach Wahrnehmungen handeln, die einen relativ kleinräumigen Bereich um das eigene Fahrzeug betreffen. Daher stellen sogenannte Fahrzeug-Folge-Modelle eine gute Möglichkeit dar, um dieses Verhalten nachzubilden. Mittels partieller Differentialgleichungen lassen sich dabei Aktionen und Reaktionen der Verkehrsteilnehmer modellieren.

Ein weitverbreitetes Fahrzeug-Folge-Modell ist das 1974 von Rainer Wiedemann entwickelte und nach ihm benannte WIEDEMANN-Modell³. In Abb. 3 lässt sich erkennen, dass in diesem psycho-physischen Modell das Fahrverhalten von Änderungen im Abstand und der Relativgeschwindigkeit zu vorausfahrenden Fahrzeugen abhängig ist. So beschleunigt ein Verkehrsteilnehmer, wenn der Abstand größer wird oder seine Geschwindigkeit unter die des vorausfahrenden Fahrzeugs sinkt. Im Gegenzug bremst er, sobald der Abstand unter einen gewissen persönlichen Schwellwert fällt. Ohne weitere externe Einflüsse wird sich so im Laufe der Zeit ein lokales Gleichgewicht einstellen, das dem Folgebereich entspricht. Das WIEDEMANN-Modell bildet die Grundlage für die Verkehrssimulation VISSIM.

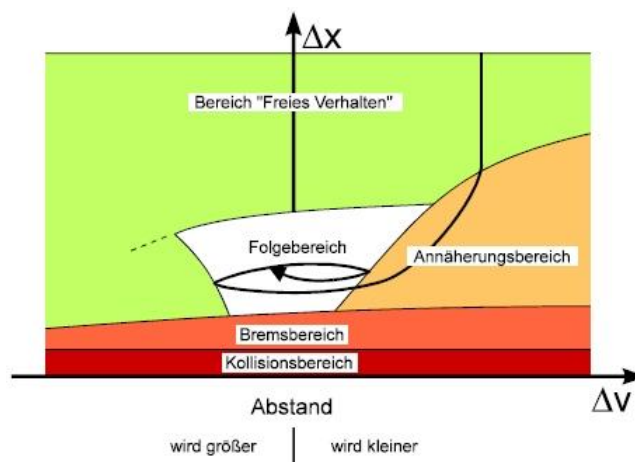


Abb. 3: WIEDEMANN-Modell [6]

³ WIEDEMANN-Modell – Psycho-physisches Abstandsmodell, das auf der Nachbildung der menschlichen Wahrnehmungs- und Entscheidungsprozesse basiert

2.3 PTV VISSIM

2.3.1 Allgemeiner Überblick

Die mikroskopische Verkehrssimulation VISSIM der PTV Planung Transport Verkehr AG ist das weltweit am weitesten verbreitete multimodale mikroskopische Simulationsprogramm für Verkehrsabläufe. Es bietet neben dem realitätsnahen Fahrzeug-Folge-Modell nach Wiedemann sowie leistungsfähigen Fahrstreifenwechselmodellen auch detaillierte Analysemöglichkeiten. So sind verschiedene Kenngrößen für die Auswertung wählbar. Dazu gehören unter anderen Verkehrsdichten, Reise- und Verlustzeiten sowie Durchschnittsgeschwindigkeiten. Diese Daten lassen sich mittels Querschnittsmessungen auf beliebigen Streckenabschnitten bestimmen.

Das Verkehrsnetz in VISSIM besteht aus Strecken und sogenannten Verbindern, welche die einzelnen Fahrspuren verschiedener Strecken miteinander verknüpfen. Die in diesem Netz verkehrenden Fahrzeuge werden entweder entlang statischer Routen oder mithilfe dynamischer Umlegung auf veränderlichen Routen geführt. Die dynamische Umlegung wird vor allem auf komplexen Verkehrsnetzen verwendet, wenn alle möglichen Quelle-Ziel-Beziehungen abgebildet werden sollen. Damit entfällt die manuelle Festlegung aller nötigen Routen. Allerdings lassen sich mit diesem Verfahren keine zusätzlichen Zielführungsalgorithmen einbinden, sodass für die in der vorliegenden Arbeit zu implementierenden Funktionen nur die Streckenführung mithilfe statischer Routen relevant ist.

Dabei wird jedem Fahrzeug beim Überfahren eines bestimmten Streckenquerschnitts eine sogenannte Routenentscheidung zugewiesen. Diese kann mehrere Routen mit gleichem Startpunkt enthalten. Gemäß einer vorgeschriebenen Verteilung wird das Fahrzeug entlang einer dieser Routen geführt. Für die dynamische Routenführung ist es essentiell, dass ein Fahrzeug diese Zuordnung während der Fahrt noch ändern kann. Dies wird über den Wechsel auf eine Alternativroute realisiert, die zur gleichen Routenentscheidung gehört. Weiterhin muss diese auch durch den aktuellen Aufenthaltsort des Fahrzeugs führen, denn das Fahrzeug kann während der Simulation nicht an einer Stelle im Netz verschwinden und an einer anderen wieder auftauchen. Das heißt, vor Simulationsbeginn müssen bereits alle möglichen bzw. sinnvollen Alternativrouten im Netz integriert sein, sodass später Fahrzeuge auf die Alternativrouten navigiert werden können, die sich aus der C2X-basierten dynamischen Routenführung ergeben.

VISSIM berücksichtigt bei der Simulation sämtliche Eigenschaften der Fahrzeuge. Darunter zählen Typ, Größe, Gewicht, Motorisierung und vor allem auch die technische Ausstattung. Diese ist im Bezug auf die C2X-Kommunikation von besonderer Bedeutung. So lassen sich spezielle C2X-Fahrzeuge einbinden, über welche sich die Funktionalität der Fahrzeug-zu-Fahrzeug-Kommunikation simulieren lässt. Hierzu wird die C2X-Schnittstelle (vgl. 2.3.2) verwendet.

Auch bezüglich der Darstellung bietet VISSIM vielfältige Möglichkeiten. Neben der 2D-Ansicht können auch fahrzeugspezifische 3D-Modelle realitätsnah abgebildet werden. Unter verschiedenen Kamerapositionen lassen sich beispielsweise Mitfahrten in einem Fahrzeug als Video festhalten. Diese sehr rechenaufwendigen zusätzlichen Features werden jedoch in dieser Arbeit nicht verwendet.

2.3.2 VISSIM-Schnittstellen

Um einen externen Zugriff auf VISSIM-Simulationen zu erhalten und diese somit zu erweitern, steht die COM⁴-Schnittstelle zur Verfügung. Mit deren Hilfe lassen sich

⁴ COM – Component Object Model

VISSIM-Objekte durch andere Programme teilweise beeinflussen. Zu den VISSIM-Objekten zählen sämtliche Elemente, die zu einem VISSIM-Netz gehören, also beispielsweise Strecken, Zuflüsse, Routen oder auch die Fahrzeuge. Diese sehr umfangreiche Schnittstelle hat neben fehlender Möglichkeit zur C2X-Kommunikation den Nachteil, dass sie relativ langsam ist. Sie eignet sich entsprechend nur für kleinere Funktionen, die von der im Folgenden erläuterten C2X-Schnittstelle nicht abgedeckt werden können.

Grundlage für die Simulation von C2X-basierter dynamischer Routenführung bildet die C2X-Schnittstelle, die seit der aktuellen VISSIM Version 5.20 zur Verfügung steht. Mithilfe eines Python-Scripts, dessen allgemeine Form in Abb. 4 dargestellt ist, ist es möglich auf C2X-Daten zuzugreifen. Unter Verwendung des Kommunikationsmoduls VCOM und des pickle-Packages⁵ lassen sich durch Serialisierung auch Fahrzeug-zu-Fahrzeug-Nachrichten austauschen. Das VCOM-Modul emuliert dabei die Nachrichtenübermittlung unter Berücksichtigung von Sendereichweiten und damit verbundenen Empfangswahrscheinlichkeiten. Für die vorliegende Arbeit wird allerdings angenommen, dass alle C2X-Fahrzeuge immer im direkten Kontakt mit der Verkehrszentrale stehen und somit keine Kommunikation zwischen Fahrzeugen also insbesondere auch kein Multihopping nötig ist. Die Verkehrszentrale wird dabei nicht durch ein spezielles Objekt in VISSIM dargestellt, sondern mithilfe des angesprochenen Python-Scripts realisiert. So kann über die C2X-Schnittstelle auf alle C2X-Fahrzeuge und deren Fahrdaten zugegriffen werden. Zusätzlich lässt sich auch die COM-Schnittstelle in diesem Script verwenden. Hierfür muss jedoch VISSIM mit dem Parameter „-automation“ gestartet werden, damit bei der COM-Initialisierung keine neue VISSIM-Instanz erzeugt wird, sondern sich die Anwendung mit der bereits gestarteten Instanz verbindet. Dies verlangsamt die Simulation allerdings noch weiter, sodass der Zugriff auf COM-Objekte nur selten erfolgen sollte und unveränderliche Objekte zwischengespeichert werden sollten.

```
import c2x
class C2XSimpleExample (c2x.ApplicationBase):
    def __init__ (self):
        c2x.ApplicationBase.__init__ (self)
    def processTimestep (self):
        pass

if (__name__ == "__main__"):
    app = C2XSimpleExample ()
    app.run ()
```

Abb. 4: Allgemeine Form des C2X-Pythonscripts

Die C2X-Schnittstelle bietet auch die Möglichkeit, die auf statischen Routen verkehrenden Fahrzeuge während der Simulation auf eine andere Route zu führen. Somit lässt sich die dynamische Routenführung auf statische Routen abbilden.

⁵ Pickle – Package zur Serialisierung und Deserialisierung von Python-Objekten

3 Zielsetzung und Aufgabenstellungen

Übergeordnetes Ziel dieser Arbeit ist es, im Rahmen des Forschungsprojektes sim^{TD} (vgl. 2.1) Konzepte und Funktionalitäten dynamischer Routenführung auf Basis von C2X-Kommunikation in die mikroskopische Verkehrssimulation VISSIM (vgl. 2.3) einzubetten. Grundlage dafür bilden die sim^{TD} -Spezifikationen F_1.2.3 [4] und F_1.3.1 [5]. Diese spezifizieren zwei Anwendungsfälle der C2X-basierten dynamischen Routenführung. Zum einen das kollektive und zum anderen das individuelle Routing. Für beide Szenarien gilt es im Rahmen dieser Arbeit, eine Python-Implementierung zu entwickeln, die es unter Verwendung der C2X-Schnittstelle von VISSIM ermöglicht, die dynamische Routenführung zu simulieren. Damit wird ein wesentlicher Beitrag geleistet, die sim^{TD} -Funktion der erweiterten Navigation und des Umleitungsmanagements zu verbessern. So lassen sich einerseits mithilfe der Verkehrssimulationen vor dem Praxis-einsatz im Feldversuch grundlegende Parameter bestimmen, die den Verkehrsfluss maximieren. Andererseits findet eine Wirkungsermittlung mithilfe eines Modells statt, das anhand der Ergebnisse aus dem sim^{TD} -Feldversuch kalibriert wird. Im Folgenden wird die Zielstellung genauer spezifiziert.

Zunächst ist ein geeignetes VISSIM-Netz zu finden, welches ausreichend groß ist und über eine Vielzahl an Verzweigungen verfügt, damit die dynamische Routenführung sinnvoll gesetzt werden kann. Weiterhin sollte es verschiedene Straßen mit unterschiedlichen Streckenkapazitäten enthalten. So lassen sich die Auswirkungen von Stauumfahrungen realistisch untersuchen.

Die feingliedrige Darstellung des VISSIM-Netzes (vgl. 2.3.1) mit sämtlichen Streckenparametern ist für das angestrebte Ziel des dynamischen Routings zu aufwendig. Daher muss zunächst eine Abstrahierung erfolgen. Eine geeignete Repräsentation für ein Verkehrsnetz ist ein gerichteter Graph, der Strecken auf Kanten und Verbinder auf Knoten abbildet. Dabei sollte die Darstellung das Netz bestmöglich vereinfachen, damit die darauf laufenden Routingalgorithmen zeitnahe Ergebnisse liefern. Die genaue Lokalisation eines Staus auf einem Streckenabschnitt ist beispielsweise nicht nötig, denn die dynamische Routenführung würde in jedem Fall eine Alternativroute wählen, die den gesamten Streckenabschnitt umfährt. In Kapitel 4.2 wird das grundlegende Konzept der Graphrepräsentation vorgestellt.

Die Grundlage für die dynamische Routenführung bildet die aktuelle Verkehrslage, die mithilfe der C2X-Daten ermittelt werden kann. Daher besteht eine wesentliche Aufgabe darin, die Verkehrslage anhand von Reisezeiten und Durchschnittsgeschwindigkeiten zu ermitteln und Verkehrseignisse zu identifizieren. Die gesammelten Verkehrsinformationen gilt es, als Kantengewichte in den Netzgraphen zu übertragen, um anschließend auf deren Basis Alternativrouten im Netzgraph bestimmen zu können. Auch müssen die Verkehrsinformationen bezüglich ihrer Relevanz für die Route eines jeden Fahrzeugs geprüft werden.

Zur Berechnung der Alternativrouten sind effiziente Graphalgorithmen einzusetzen, die für kollektives und individuelles Routing stets optimale Lösungen liefern. Die ermittelten Alternativrouten gilt es im Anschluss wieder zurück auf das VISSIM-Netz zu übertragen, um die Fahrzeuge auf diese leiten zu können. Dabei sind verschiedene Akzeptanzraten zu berücksichtigen, welche das Verhalten der Fahrer vereinfacht widerspiegeln sollen.

Außerdem soll in einer kurzen Simulationsstudie (vgl. Kapitel 6 und 7) die allgemeine verkehrliche Wirkung der C2X-basierten dynamischen Routenführung auf einem ausgewählten Verkehrsnetz ermittelt werden, indem Aussagen bezüglich eventueller Fahrzeitverbesserungen oder Stauverminderungen getroffen werden. Zur wirklichkeitsnahen Nachbildung des Verkehrsaufkommens sind dabei reale Daten von Verkehrsde-

tektoren der wichtigsten Streckenzüge im Verkehrsnetz zu verwenden. In der Simulation ist stets zwischen kollektiver und individueller Routenführung zu unterscheiden.

Abschließend soll im speziellen Fall des individuellen Routings ein Schwellwert für die Verlustzeit ermittelt werden, ab der eine Routenneuberechnung sinnvoller Weise erfolgt. Eine Neuberechnung wäre dann erforderlich, wenn die Verwendung einer günstigeren Route für das einzelne Fahrzeug eine Verbesserung darstellen würde.

4 Grundlegende Konzepte und Implementierungsansätze

4.1 Wahl eines geeigneten VISSIM-Netzes

Das Verkehrsnetz bildet die Basis für die Verkehrssimulation in VISSIM. Daher ist es im Bezug auf das Testen von Systemen zur Verkehrsbeeinflussung von besonderer Bedeutung. Die dynamische Routenführung kann nur sinnvolle Ergebnisse liefern, wenn auch das Netz genügend Alternativstrecken zur Verfügung stellt, über die Fahrzeuge geroutet werden können. Desweiteren sollte das Netz möglichst realistische Verkehrssituationen abbilden können. Dazu dient die Verwendung verschiedener Streckentypen, die sich in der Kapazität, der Länge, der Breite und auch in der Anzahl der Spuren unterscheiden. Um diesen Anforderungen zu genügen, ist es naheliegend, real existierende Verkehrsnetze manuell nachzubilden. Da hier das Erreichen einer hohen Detailtreue mit erheblichem Aufwand verbunden ist, bietet VISSIM die Möglichkeit, Netze aus anderen Simulationsanwendungen wie zum Beispiel VISUM⁶ zu importieren.

Für die vorliegende Arbeit wurde das in Abb. 5 gezeigte VISSIM-Netz zur Verfügung gestellt, welches im Rahmen von sim^{TD} verwendet wird. Es stellt das sim^{TD} -Testgebiet mit hoher Genauigkeit dar und bietet somit die optimale Grundlage, die aus der Simulation gewonnenen Ergebnisse später in der Planung des Feldversuchs einsetzen zu können. Das Netz umfasst die wesentlichen Trassen aller Verkehrsmittel. Für jede davon sind Geschwindigkeitsrichtwerte als sogenannte Wunschgeschwindigkeiten vorgegeben. Somit können in der späteren Simulationsstudie (vgl. Kapitel 6 und 7) die mithilfe von Detektoren gewonnenen realen Verkehrsflussdaten verwendet werden, um möglichst wirklichkeitsnahe Ergebnisse zu erhalten.



Abb. 5: VISSIM-Netz des sim^{TD} -Testgebiets

⁶ PTV VISUM – Makroskopische Verkehrssimulation der PTV AG

4.2 Erzeugung eines gerichteten Graphen auf Basis des VISSIM-Netzes

Basierend auf den Zielstellungen aus Kapitel 3 soll das VISSIM-Netz geeignet als gerichteter Graph repräsentiert werden. Dazu bezeichnen $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ zwei gerichtete Graphen mit Knotenmengen $V_1 = \{v_1, \dots, v_{n_1}\}$ und $V_2 = \{w_1, \dots, w_{n_2}\}$ sowie Kantenmengen $E_1 = \{e_1, \dots, e_{m_1}\}$ und $E_2 = \{f_1, \dots, f_{m_2}\}$, $n_1, n_2, m_1, m_2 \in \mathbb{N}$.

4.2.1 Allgemeines Verfahren zur Grapherzeugung

Aufgrund der inneren Struktur einer VISSIM-Netz-Datei ist es einfacher zunächst Strecken in Knoten und Verbinder in Kanten umzuwandeln. Anschließend wird aus dem entstandenen gerichteten Graphen G_1 ein Graph G_2 erzeugt, bei dem Strecken Kanten entsprechen und Verbinder durch Knoten dargestellt werden. Im Folgenden wird das Vorgehen zur Umwandlung von G_1 in G_2 erläutert.

Zunächst sei angemerkt, dass in VISSIM-Netzen eine Strecke durchaus über mehr als zwei Verbinder (je einer am Anfang und einer am Ende der Strecke) mit verschiedenen anderen Strecken verbunden sein kann. So existieren beispielsweise bei einer Einmündung (vgl. Abb. 6) immer mindestens zwei Verbinder pro Fahrtrichtung. Der grüne Verbinder verknüpft die rote Strecke mit dem rechten Abzweig und der blaue Verbinder führt von der roten Strecke gerade aus zur nächsten Strecke. Da der Anfang der Strecke in der Regel ebenfalls über mindestens einen Verbinder mit anderen Strecken verbunden ist, sind also mehr als zwei Verbinder mit der roten Strecke verknüpft.

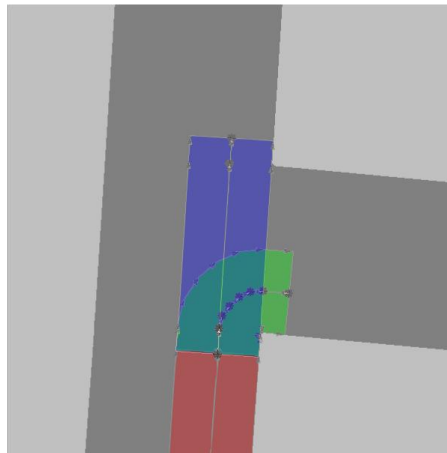


Abb. 6: Einmündung in VISSIM-Netz

Bezogen auf den Graph G_1 entsprechen diese Verbinder einfach mehreren Kanten, die von einem Knoten v_x abgehen bzw. zu v_x hinführen. Bei der Umwandlung zu G_2 werden diese (mehr als zwei) Kanten jedoch in einzelne Knoten überführt, die dann alle mit der (einen) Kante f_x verbunden werden müssten, die sich aus der Umwandlung von v_x ergibt. Dies steht im Widerspruch zur Eigenschaft einer Kante, genau zwei Knoten miteinander zu verbinden. Um dieses Problem zu lösen, wäre es sinnvoll alle eingehenden bzw. alle ausgehenden Kanten des Knoten v_x aus G_1 zu je einem Knoten w_i bzw. w_o in G_2 zusammenzufassen. Die Abb. 7 und Abb. 8 verdeutlichen dieses Vorgehen in zwei Schritten an einem einfachen Beispiel. Im Graph G_1 besitzt der Knoten v_x die zwei eingehenden Kanten e_4 und e_5 sowie die beiden ausgehenden Kanten e_6 und e_7 . Im ersten Schritt zur Umwandlung in G_2 werden die ausgehenden Kanten zu dem Knoten w_{67} vereinigt. Dabei gehen bezüglich der allgemeinen Netzstruktur keine für das Rou-

ting relevanten Informationen verloren. Die Kante f_x ist nun jedoch doppelt vorhanden. Ebenso wird der Knoten v_z in G_1 auf zwei Kanten f_z in G_2 abgebildet. Daher scheint es naheliegend, die eingehenden Kanten aller Knoten in G_1 , die nun Startknoten der mehrfach vorhandenen Kanten sind, ebenfalls zu je einem Knoten zusammenzufassen. Dies erfolgt im zweiten Schritt der Umwandlung. Es werden somit w_4 und w_5 zu w_{45} sowie w_{23} und w_{67} zu w_{2367} vereinigt. Nun ist zwar das Problem der mehrfach vorkommenden Kanten gelöst, jedoch ergibt sich dadurch ein größeres Problem. Über den neuen Knoten w_{2367} existiert nun eine Verbindung zwischen Kante f_x und f_y . Im Graph G_1 besteht jedoch zwischen den entsprechenden Knoten v_x und v_y keine Verbindung. Das heißt, dass durch die Graphmodellierung neue Routingmöglichkeiten hinzugefügt werden, die im VISSIM-Netz nicht vorhanden sind.

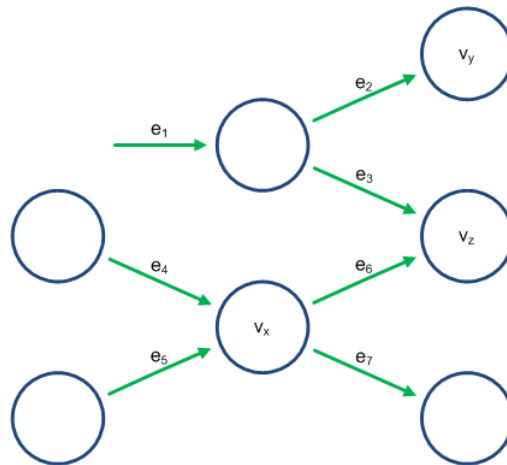


Abb. 7: Graph G_1

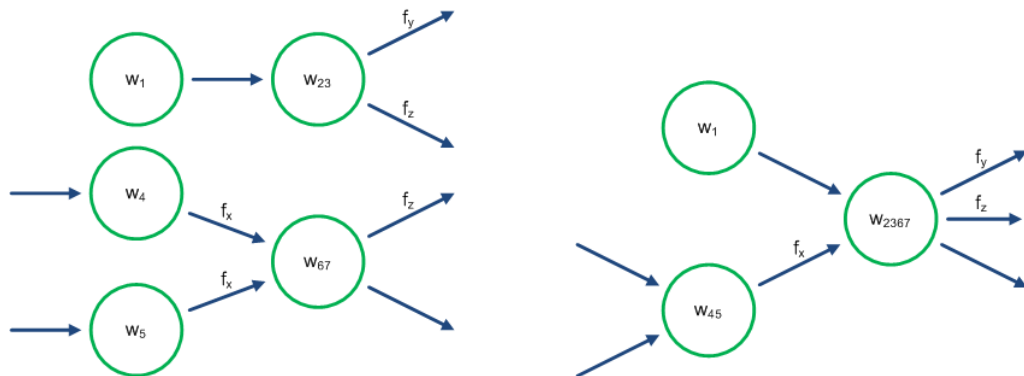


Abb. 8: Graph G_2 ,
links: nach erstem Umwandlungsschritt,
rechts: nach zweitem Umwandlungsschritt

Da der Graph nach dem ersten Schritt der Umwandlung diese Fehlinformationen noch nicht enthält, sollte dieses Zwischenergebnis als Graphrepräsentation gewählt werden. Die mehrfach existierenden Kanten müssen dabei durch eine entsprechende Abbildungsvorschrift gesondert behandelt werden (vgl. Kapitel 5.1.1).

4.2.2 Optimierung des Netzgraphen

Der nach dem im vorigen Abschnitt erläuterten Prinzip erzeugte Graph G_2 gibt zwar das VISSIM-Netz richtig wider, allerdings ist er für die Routensuche noch sehr ineffizient. So besitzt er sehr viele Knoten, die nur genau eine ausgehende und eine eingehende Kante besitzen, also keine Alternativen für das Routing bieten. Diese Knoten werden im Folgenden als Zwischenknoten bezeichnet. Knoten mit mehr als einer eingehenden oder mehr als einer ausgehenden Kante werden Verzweigungsknoten genannt.

Außerdem befinden sich im VISSIM-Netz zum Teil kleine Strecken, die nur langsam befahren werden können oder nur für ein geringes Verkehrsaufkommen konzipiert sind. Bei der dynamischen Routenführung ist es jedoch wichtig, dass die Alternativrouten über Strecken führen, deren Kapazität ausreichend groß ist. Daher ist es prinzipiell sinnvoll nur solche Strecken im Netzgraphen darzustellen.

Im Folgenden werden diese beiden Optimierungsansätze genauer vorgestellt:

Eliminierung von Zwischenknoten

Wie im übergeordneten Abschnitt erläutert, sind für das Routing nur Verzweigungsknoten relevant, denn nur dort müssen Routenentscheidungen getroffen werden. Daher ist es sinnvoll Zwischenknoten aus dem Graph zu eliminieren und die betreffenden Kanten zu einer einzigen zusammenzufassen. Abb. 9 zeigt einen Graph, der über drei Zwischen- und drei Verzweigungsknoten verfügt. Durch die Optimierung werden die Zwischenknoten entfernt, sodass anschließend die Verzweigungsknoten direkt miteinander verbunden sind (vgl. Abb. 10).

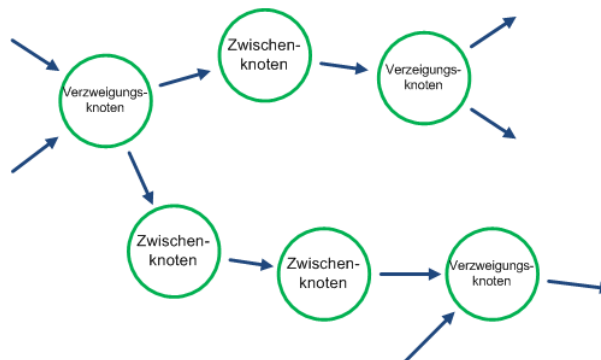


Abb. 9: Graph mit Zwischen- und Verzweigungsknoten

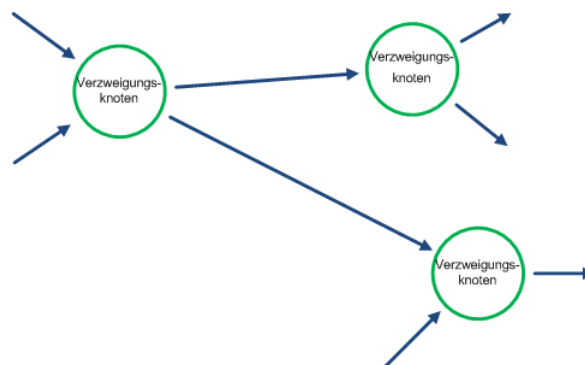


Abb. 10: Graph ohne Zwischenknoten

Um die beschriebene Graphreduzierung im gesamten Graph strukturiert umzusetzen, wird der folgende Algorithmus verwendet:

Im ersten Schritt werden alle Verzweigungsknoten und alle die Knoten gesucht, welche keine eingehende Kante besitzen. Diese bilden den Ausgangspunkt für die anschließende Löschung der Zwischenknoten. Dabei wird beginnend von einem dieser Ausgangsknoten rekursiv jeder verbundene Zwischenknoten zusammen mit den entsprechenden Kanten eliminiert bis wieder ein Verzweigungsknoten erreicht wird. Dann wird eine Kante zwischen Ausgangs- und Endknoten erzeugt, die repräsentativ für alle gelöschten Kanten steht.

Beschränkung auf nützliche Strecken

Die Strecken in VISSIM verfügen über das Attribut „Verhaltenstyp“. Der Verhaltenstyp kann angeben, welche Fahrzeugklassen den Streckenabschnitt befahren dürfen oder welche Geschwindigkeit angemessen für den Bereich ist. Um die unbrauchbaren Strecken bei der Graphgenerierung auszuschließen, werden nur die Strecken geparkt, deren Verhaltenstyp für die dynamische Routenführung geeignet ist. In dem in dieser Arbeit verwendeten VISSIM-Netz werden die Verhaltenstypen berücksichtigt, die für Geschwindigkeiten über 30 Kilometer pro Stunde stehen. Bei Anwendung dieser Funktion auf andere Netze sind die relevanten Verhaltenstypen manuell zu bestimmen. Dabei ist stets darauf zu achten, dass eine eindeutige Zuordnung von Fahrtgeschwindigkeiten zu Verhaltenstypen existiert. Sollte dies nicht der Fall sein, kann dieser Optimierungsschritt auch abgeschaltet werden.

Aufgrund dieser Beschränkung kann es zu einer Teilung des Graphen in mehrere Zusammenhangskomponenten kommen. Im Falle des verwendeten Netzes entstehen neben einer sehr großen Komponente noch sehr viele kleinere und damit ebenfalls unbrauchbare Komponenten. Daher werden alle kleinen Zusammenhangskomponenten gelöscht. So kann die Größe des Graphen nochmals reduziert werden.

Nach der Durchführung der beschriebenen Optimierungsverfahren besteht der Netzgraph nur noch aus Verzweigungsknoten. Zudem sind nur Streckenabschnitte repräsentiert, deren Kapazitäten eine gewisse Mindestanforderung erfüllen. Damit ist der Netzgraph im Bezug auf die Anforderung der dynamischen Routenführung optimiert.

4.3 Verkehrslage- und Reisezeiterfassung

4.3.1 Allgemeine Vorgehensweise

Die Verkehrslage- und Reisezeiterfassung bilden die Grundlage für die dynamische Routenführung. Die Funktionsspezifikation der realen sim^{TD} -Funktion sieht vor, dass diese Daten über die Roadside Stations ermittelt werden. Für jeden Streckenabschnitt existiert eine solche Roadside Station, welche die Fahrdaten der Fahrzeuge des zugehörigen Streckenabschnitts aggregiert. Zu den Fahrdaten gehören unter anderem Geschwindigkeit und Position. Mithilfe dieser Größen lassen sich sowohl die Verkehrslage, als auch die benötigte Reisezeit für jeden Streckenabschnitt bestimmen. Die Werte der verschiedenen Fahrzeuge werden dabei über einen festgelegten Zeitraum hinweg gemittelt. Alle veralteten Daten, die nicht mehr in diesem Intervall liegen, werden verworfen, falls ausreichend neuere Daten vorliegen. Der Zeitraum lässt sich während der Datenerfassung ändern und an die örtlichen Gegebenheiten anpassen. So ist es beispielsweise bei einem Netz, das im Wesentlichen aus Autobahnstrecken besteht, sinnvoll einen kürzeren Zeitabschnitt zu betrachten als bei Netzen in ländlichen Bereichen.

Durch die, im Gegensatz zu einer wenig befahrenen Landstraße, größere Fahrzeuganzahl auf Autobahnen reichen schon wenige Minuten aus, um ausreichende Daten für die Verkehrslageerfassung zu erhalten. Für die vorliegende Arbeit wird das Intervall auf 10 Minuten gesetzt.

In dieser Arbeit soll die beschriebene Vorgehensweise möglichst realistisch in VISSIM nachgebildet werden. Hierfür kommen verschiedene Datenstrukturen zum Einsatz. Für jede Roadside Station wird eine Queue angelegt, in der die Fahrzeugdaten chronologisch gespeichert werden. So können neue Daten vorn zu der Queue hinzugefügt werden und ältere Daten nach dem „First In – First Out“-Prinzip wieder entnommen und gelöscht werden. Alle diese Queues werden in einer Liste gespeichert. Übertragen auf die Realität würde diese Liste den Daten entsprechen, die in einer Verkehrszentrale vorliegen.

In den folgenden Abschnitten wird die Anpassung dieses Verfahrens an die einzelnen Funktionen zur Verkehrslage- und Reisezeitermittlung genauer erläutert.

4.3.2 Ermittlung der Verkehrslage

Bei der Verkehrslageerfassung spielen die Größen Durchschnittsgeschwindigkeit und die durchschnittliche Anzahl der Fahrzeuge auf einem Streckenabschnitt eine wesentliche Rolle. So lässt sich beispielsweise aus der Durchschnittsgeschwindigkeit ermitteln, ob es zu Stauungen kommt. Die Anzahl der Fahrzeuge in Kombination mit der Kapazität der Strecke lässt auf die Auslastung der Straße schließen.

Zur Ermittlung dieser Größen werden zusätzlich zu den im Punkt 4.3.1 vorgestellten Datenstrukturen für jeden Streckenabschnitt noch jeweils eine Variable für die kumulierte Geschwindigkeit aller Fahrzeuge und für die Anzahl der erfassten Zeitschritte verwendet. Die Durchschnittsgeschwindigkeit lässt sich dann durch Teilen der kumulierten Geschwindigkeit durch die Anzahl der Einträge in der Daten-Queue bestimmen. Analog erhält man die mittlere Fahrzeuganzahl als Quotient aus der Queuelänge und der Anzahl der erfassten Zeitschritte.

4.3.3 Ermittlung der Reisezeiten und deren Überführung in Kantengewichte des Netzgraphen

Da die Reisezeit im Gegensatz zur Geschwindigkeit kein Fahrzeugattribut ist, muss diese Größe zunächst für jede Strecke bestimmt werden. Dazu wird für jedes Fahrzeug gespeichert, auf welchem Streckenabschnitt es sich im vergangenen Zeitschritt befunden hat und wann es diesen befahren hat. Über die aktuelle Position des Fahrzeugs lässt sich ermitteln, ob es im aktuellen Zeitschritt auf einen neuen Streckenabschnitt gefahren ist. In einem solchen Fall ergibt sich die Reisezeit des vorherigen Streckenabschnitts als Intervall zwischen dem Zeitpunkt des Befahrens und dem aktuellem Zeitpunkt.

Die Bestimmung der mittleren Reisezeit wird analog zur Verkehrslageerfassung umgesetzt. Auch hier wird neben den im Abschnitt 4.3.1 dargestellten Datenstrukturen noch eine weitere Variable benötigt. Diese speichert kumuliert die Reisezeiten aller Fahrzeuge, die den entsprechenden Streckenabschnitt befahren haben. Zur Bestimmung der mittleren Reisezeit wird die Summe der Reisezeiten durch die Anzahl der Einträge in der Queue geteilt.

Da das dynamische Routing immer die aktuell schnellste Route bevorzugt, stellen die für jeden Streckenabschnitt ermittelten Reisezeiten die Grundlage dafür dar. Das heißt, der Netzgraph muss entsprechend der Fahrzeiten mit Kantengewichten versehen werden. Aufgrund der im Abschnitt 4.2.2 erläuterten Optimierung des Graphen existieren im VISSIM-Netz mehr Strecken als Kanten im dazugehörigen Graphen vorhanden

sind. Dabei stellt der Optimierungsalgorithmus jedoch sicher, dass immer eine surjektive Abbildung zwischen Strecken und Kanten existiert. Somit können die Reisezeiten aller zu einer Kante gehörenden Strecken (Urbilder) aufsummiert werden und als Gewicht auf die entsprechende Kante (Bild) übertragen werden.

Mit diesem Verfahren lassen sich die Reisezeiten für jeden Streckenabschnitt ermitteln und als Kantengewichte auf den Netzgraph übertragen. Aufbauend darauf kann dann die im folgenden Abschnitt beschriebene Berechnung der Alternativrouten durchgeführt werden.

4.4 Ermittlung von Alternativrouten

Das dynamische Routing auf Basis von statischen Routen kann in VISSIM nur erfolgen, wenn vor der Simulation bereits alle möglichen Alternativrouten im verwendeten Verkehrsnetz integriert sind. Das heißt, während eines Simulationslaufes lassen sich keine neuen Routen zum Netz hinzufügen. Der Grund für diese Einschränkung ist, dass Routen in VISSIM nur einmal geladen werden und Fahrzeuge während einer Simulation nur auf Routen geleitet werden, die zuvor eingelesen wurden. Daher ist es nötig eine Funktion zu entwickeln, die zu den vorhandenen statischen Routen alle geeigneten Alternativen ermittelt und zum Netz hinzufügt, bevor die eigentliche Simulation stattfinden kann. Wie im Abschnitt 2.3.1 beschrieben, ist jede statische Route in VISSIM einer Routenentscheidung zugeordnet, die durch einen Startquerschnitt definiert ist. Überfährt ein Fahrzeug einen solchen Fahrbahnquerschnitt, wird es nach einer festgelegten Verteilung auf eine der zugehörigen Routen geführt. Die Routen verfügen über Zielquerschnitte, die das Ende der Route definieren. Weiterhin sind alle Strecken und Verbinder in der Route enthalten, über welche die Route führt.

Die bereits vorhandenen Routen gilt es zunächst auf den Netzgraph zu übertragen, bevor die Ermittlung der möglichen Alternativrouten im Graph erfolgen kann. Es ist jedoch nicht sinnvoll für jede Route alle möglichen Wege zu ermitteln, die den gleichen Start- und Zielquerschnitt besitzen. Ab einer bestimmten Größenordnung kann eine Alternativroute auch bei sehr starken Verkehrsstörungen nicht mehr zu einer Fahrzeitverbesserung gegenüber der Standardroute führen. Um eine Grundlage für die entsprechende Beschränkung der Routensuche zu erhalten, sollten zunächst die Fahrzeiten bestimmt werden, die bei freier Fahrt auf den vorhandenen Routen benötigt werden. Dafür wird eine Vorsimulation durchgeführt, bei der die Verkehrsdichte um ein vielfaches kleiner als in der Realität ist, jedoch möglichst viele Streckenabschnitte befahren werden. Die hierdurch ermittelten Standardreisezeiten bilden die Datenbasis für die anschließende Alternativroutensuche. Es werden nur die Alternativen verwendet, für welche die Reisezeit maximal um einen bestimmten Faktor größer ist als die der schnellsten Route. Aufgrund der empirisch ermittelten großen Anzahl von Alternativen und des damit verbundenen hohen Rechenaufwands während eines Simulationslaufes, wird der Faktor im Rahmen dieser Arbeit auf 1,5 gesetzt. Der Algorithmus zur Suche aller dementsprechenden Start-Ziel-Wege basiert auf dem Prinzip der Tiefensuche. Die so gefundenen Alternativrouten werden in einer Datei gespeichert, die später bei jeder Simulation geladen werden kann. Außerdem müssen die Routen in das VISSIM-Netz zurück überführt werden, damit eine Routenbeeinflussung der Fahrzeuge während der eigentlichen Simulation durchgeführt werden kann. Im Folgenden Abschnitt wird dieser Schritt genauer erläutert.

4.5 Übertragen von Alternativrouten auf das VISSIM-Netz als statische Routen

Die statischen Routen in VISSIM sind in der Netzdatei angegeben und liegen in folgender Formatierung vor:

```
ROUTENENTSCHEIDUNG <ID> NAME <Name (optional)> BESCHRIFTUNG 0.00 0.00
  STRECKE <ID der Startstrecke> BEI <Streckenkilometer des Startquerschnitts>
  ZEIT VON <Beginn der Gültigkeit> BIS <Ende der Gültigkeit>
  FAHRZEUGKLASSEN <IDs der betreffenden Fahrzeugklassen>
  ROUTE <ID> ZIEL STRECKE <ID> BEI <Streckenkilometer des Zielquerschnitts>
    ANTEIL <Anteil der Fahrzeuge, die auf diese Route geleitet werden>
    UEBER <Liste aller Verbinder und Strecken, über welche die Route führt>
  ROUTE <ID> ZIEL STRECKE <ID> BEI <Streckenkilometer des Zielquerschnitts>
    ANTEIL <Anteil der Fahrzeuge, die auf diese Route geleitet werden>
    UEBER <Liste aller Verbinder und Strecken, über welche die Route führt>
```

Dabei können beliebig viele Routen zu einer Routenentscheidung zugordnet werden. Der Anteil der Fahrzeuge, die über eine jede Route geführt werden sollen, kann relativ oder auch als prozentualer Belastungswert angegeben werden. VISSIM rechnet die relativen Anteile automatisch für die absolute Fahrzeugzahl um.

Für die Überführung der ermittelten Alternativrouten in das beschriebene Format müssen zunächst alle zugehörigen Knoten und Kanten des Graphen wieder in die entsprechenden Strecken und Verbinder des VISSIM-Netzes umgewandelt werden. Hierzu wird die im Kapitel 4.2 erläuterte Abbildung von Strecken auf Kanten umgekehrt. Die Ermittlung der entsprechenden Verbinder zwischen diesen Strecken ist nicht nötig, da VISSIM diese beim Start des nächsten Simulationslaufes automatisch hinzufügt. Nachdem diese Umwandlung erfolgt ist, können alle Alternativrouten zu den existierenden Routenentscheidungen in die Netzdatei geschrieben werden.

4.6 Beeinflussung des Routingverhaltens einzelner Fahrzeuge

Bei der dynamischen Routenführung in VISSIM können Fahrzeuge nur auf eine der ermittelten Alternativrouten (vgl. Abschnitt 4.4) geführt werden. Daher müssen auch während der Simulation nur die Reisezeiten dieser Routen miteinander verglichen werden. Die konkrete algorithmische Umsetzung dessen wird in den beiden folgenden Unterkapiteln beschrieben.

4.6.1 Individuelles Routing

Beim individuellen Routing wird für jedes Fahrzeug einzeln eine Alternativroute bestimmt, die zum aktuellen Zeitpunkt die schnellste Route vom Aufenthaltsort zum Zielquerschnitt ist. Gestartet wird die Routensuche jedoch nur, wenn die Verlustzeit einen gewissen Schwellwert übersteigt. Daher gilt es zunächst die Differenz zwischen der aktuell benötigten Reisezeit und der Standardreisezeit zu ermitteln. Dazu werden für die Route eines Fahrzeugs die aktuellen Gewichte aller vorausliegenden Kanten aufsummiert und mit den Kantengewichten des Standardgraphs verglichen. Die Ermittlung eines optimalen Schwellwertes ist eine der im Kapitel 3 erläuterten Ziele der vorliegenden Arbeit.

Falls die Reisezeitdifferenz für ein Fahrzeug den Schwellwert überschreitet, wird das Routing aktiviert. Das heißt, über die aktuelle Route des betrachteten Fahrzeugs wird die zugehörige Routenentscheidung und damit auch alle zuvor gesammelten Alternativrouten bestimmt. Aus diesen Alternativen werden genau die herausgefiltert, die auch über die den aktuellen Aufenthaltsort des Fahrzeugs führen, denn nur dann kann

auch ein eventueller Routenwechsel stattfinden. Von allen relevanten Routen wird dann die schnellste gesucht. In Abhängigkeit einer festgelegten Akzeptanzrate kann dann das Fahrzeug umgeleitet werden.

4.6.2 Kollektives Routing

Im Gegensatz zum individuellen Routing wird das von der (virtuellen) Verkehrszentrale gesteuerte Umleitungsmanagement nur dann aktiviert, wenn die Durchschnittsgeschwindigkeit auf einem Streckenabschnitt einen festgelegten Schwellwert unterschreitet. In dieser Arbeit wird dieser Schwellwert auf 30 Kilometer pro Stunde gesetzt, da ab dieser Geschwindigkeit im Allgemeinen von stockendem Verkehr also einer Verkehrsstörung gesprochen wird.

Lassen sich solche gestörte Streckenabschnitte identifizieren, werden alle Routen ermittelt, die über diese Strecken verlaufen. Zu diesen werden anschließend alle ungestörten Alternativen gesucht. Diese müssen nun an die Fahrzeuge, welche sich auf den gestörten Routen befinden, als Umleitungsempfehlungen übermittelt werden. In den Fahrzeugen kann dann ermittelt werden, welche der Empfehlungen sinnvolle Alternativen zur aktuellen Route darstellen. Auch hier gilt wie beim individuellen Routing, dass nur Streckenführungen relevant sind, die auch den aktuellen Aufenthaltsort des Fahrzeugs beinhalten. Von den so ermittelten Routen werden die drei aktuell schnellsten bestimmt. Es kann eine Funktion erstellt werden, die unter Berücksichtigung verschiedener Fahrerprofile die Routenwahl eines Fahrers simuliert. In der Zielstellung zur vorliegenden Arbeit (vgl. Kapitel 3) ist aber eine derartig detaillierte Fahrerverhaltenssimulation nicht vorgesehen. Stattdessen wird ähnlich wie beim individuellen Routing eine Funktion implementiert, die in Abhängigkeit einer festgelegten Akzeptanzrate die Fahrzeuge auf die schnellste ungestörte Alternativroute führt.

5 Dokumentation

In diesem Kapitel werden die umgesetzten Implementierungen zur C2X-basierten dynamischen Routenführung in VISSIM dokumentiert. Dabei sind die Namen aller Klassen, Methoden, Attribute und Variablen kursiv dargestellt. Der zugehörige Quelltext befindet sich auf der beigelegten CD.

Für eine Simulation unter Verwendung der entwickelten Funktionalität müssen die `c2x.ini`, in der vermerkt ist, welches Pythonscript von VISSIM aufgerufen wird, und alle implementierten Python-Klassen im Verzeichnis der VISSIM-Netz-Datei liegen. Weiterhin müssen folgende serialisierte Dateien vorhanden sein:

- `<Netzdateiname>Graph.txt`
- `<Netzdateiname>DefaultGraph.txt`
- `<Netzdateiname>.sr`

Die ersten beiden Dateien enthalten jeweils den Netzgraphen sowie alle zugehörigen Referenztabellen der Klasse *NetGraph* (vgl. Kapitel 5.2.2). Bei ersterer sind alle Kantengewichte des Graphen auf den Standardwert eins gesetzt. Die zweite Datei hingegen beinhaltet den Graphen mit den in einer Vorsimulation ermittelten Kantengewichten auf Basis der bei freier Fahrt benötigten Reisezeiten. Die dritte Datei enthält sowohl die manuell ins Netz eingetragenen statischen Routen, als auch sämtliche ermittelten Alternativrouten. Die drei Dateien müssen in einem ersten Schritt vor der eigentlichen Simulation erstellt werden. Dazu dienen die im Folgenden als „Zusatzprogramme“ bezeichneten Implementierungen. Das Programm, das während eines Simulationslaufes zur Ausführung kommt, wird im weiteren Verlauf der Arbeit „Hauptprogramm“ genannt. Nachfolgend werden diese beiden Programmklassen genauer vorgestellt.

5.1 Zusatzprogramme

5.1.1 Erzeugen des Netzgraphen

Basierend auf der Notation aus Kapitel 4.2 wird in der Methode *parseVerbinderFromFile* der Klasse *NetGraph* ein Graph G_1 erzeugt, bei dem die Knotenmenge der Menge aller Strecken in VISSIM entsprechen und Verbinder auf Kanten abgebildet werden. Dabei werden nur die Strecken und Verbinder berücksichtigt, deren Verhaltenstyp in der Liste *Verhaltenstypen* enthalten ist. In *swapNodesEdges* wird dann aus G_1 der Graph G_2 erzeugt. Die entsprechenden Datenstrukturen für die Graphrepräsentation werden aus der Python-Library „python-graph“ importiert.

Um die zu einem Knoten in G_2 gruppierten ausgehenden Kanten aus G_1 später wieder auseinanderhalten zu können, wird das Dictionary⁷ *VerbinderSourceTab* erzeugt, das für jeden Knoten in G_1 dessen ausgehenden Kanten enthält und mit dessen Namen indiziert wird.

Nach dem Zusammenfassen der ausgehenden Kanten aus G_1 zu einem Knoten in G_2 , können die Knoten aus G_1 als Kanten in G_2 eingefügt werden. Dafür wird ein Dictionary *SourceDestTab* angelegt, welches für jeden Knoten in G_1 die Zielknoten der ausgehenden Kanten sammelt. Diese Zielknoten werden somit im Anschluss als Kanten in G_2 eingefügt. Der entstehende Graph G_2 enthält, wie im Kapitel 4.2.1 erläutert, Kanten, deren Label mehrfach im Graph vorkommt. Um den Zugriff auf diese Kanten zu vereinfachen, wird das Dictionary *edgeLabelTab* erstellt, welches jedem Kantenlabel die entsprechenden Kanten zuordnet.

⁷ Dictionary: Python-Datenstruktur ähnlich einer Hashmap in Java

Wie im Kapitel 4.2.2 erläutert, ist es sinnvoll unbrauchbare Strecken bei der Graphgenerierung auszuschließen und den Graph somit zu optimieren. Dies wird erreicht, indem nur die Strecken geparkt werden, deren Verhaltenstyp in der Liste *Verhaltenstypen* angegeben ist. Diese Liste ist nur für das in dieser Arbeit verwendete Netz gültig und muss daher für jedes andere Netz neu bestimmt werden. Um die Verwendung anderer Netze so nicht völlig auszuschließen, wird vor dem Parsen geprüft, ob es sich um das hier verwendete Netz handelt. Nur dann wird diese Optimierungsmethode angewendet.

Die in Kapitel 4.2.2 beschriebene Vorgehensweise zur weiteren Reduzierung des Graphen durch Eliminierung von Zwischenknoten wird mithilfe der Funktionen *mergeTracks* und *reduceGraph* umgesetzt. Erstere dient der Ermittlung von Ausgangsknoten für die Reduzierung. Letztere wird von *mergeTracks* aufgerufen und eliminiert dann, beginnend bei dem Ausgangsknoten, rekursiv alle Zwischenknoten bis wieder ein Verzweigungsknoten erreicht wird.

Durch Aufruf der Funktion *saveGraph* der Klasse *NetGraph* wird der Netzgraph zusammen mit allen für die spätere Rückübertragung der Graphinformationen auf das VISSIM-Netz nötigen Dictionaries als „<Netzdateiname>Graph.txt“ gespeichert.

5.1.2 Erzeugen eines gewichteten Netzgraphen

Nachdem der Netzgraph erzeugt und gespeichert wurde, kann eine Vorsimulation durchgeführt werden (vgl. Kapitel 4.4). Diese wird mithilfe der Klasse *CalcTravelTimes* durchgeführt, die bis auf die Routingfunktionalität mit der Klasse *TrafficAnalysis* (vgl. 5.2.1) übereinstimmt. Dabei wird der gespeicherte Netzgraph mithilfe der *NetGraph*-Methode *readFile* geladen und mit den aus der Simulation gewonnenen Reisezeiten gewichtet. Alle zehn Minuten wird durch Aufruf von *saveGraphWithWeights* ein Zwischenergebnis gespeichert. Im verwendeten VISSIM-Netz sollte nach zwei bis drei Stunden ein ausreichendes Ergebnis vorliegen.

5.1.3 Ermitteln und Speichern aller Alternativrouten

Zur Bestimmung aller Alternativrouten dienen die Methoden *parseAllStaticRoutes* und *findAllStaticRoutes* der Klasse *NetGraph*. In ersterer wird die Netz-Datei eingelesen und alle Routenentscheidungen mit zugehörigen Routen in der Datenstruktur *existingStaticRoutes* als Objekte der Klasse *Route* (vgl. Abschnitt 5.2.6) übertragen. Die einem zweidimensionalen Array ähnliche Datenstruktur *existingStaticRoutes* besteht aus zwei ineinander geschachtelten Dictionaries, die mit der Startstrecke bzw. der Zielstrecke indiziert sind. Es werden in *parseAllStaticRoutes* nur die Strecken aus der VISSIM-Route geparkt. Die Verbinder sind für das Routing nicht relevant und werden daher ignoriert.

Durch Aufruf von *findAllStaticRoutes* werden zu jeder Route in *existingStaticRoutes* alle relevanten Alternativrouten gesucht. Dazu dienen die nach dem Prinzip aus Kapitel 4.4 aufgebauten Methoden *findAllRoutesFromTo* und *search*. Erstgenannte bestimmt mithilfe des in python-graph integrierten Dijkstra-Algorithmus die kürzeste Reisezeit von der Start- zur Zielstrecke einer Route. Anschließend wird die durch Aufruf der rekursiven Funktion *search* die beschränkte Tiefensuche nach den Alternativrouten gestartet. Alle gefundenen Routen werden in die Datenstruktur *StaticRoutes* übertragen. Diese ist nach dem wie *existingStaticRoutes* aus ineinander verschachtelten Dictionaries aufgebaut. Jedoch wird nicht nur eine Route pro Start-Ziel-Strecken-Paar gespeichert, sondern eine Liste aus allen Alternativrouten.

Mithilfe der Methode *saveAllStaticRoutes* werden in die Datei „<Netzdateiname>.sr“ *existingStaticRoutes*, *StaticRoutes* und der Inhalt der Netzdatei serialisiert ge-

speichert. Weiterhin werden die Routen aus *StaticRoutes* in die Netzdatei geschrieben und unter dem Namen „<Dateiname der Netzdatei> WithAllRoutes.inp“ gespeichert.

Damit sind die nötigen Dateien für die eigentliche Simulation erstellt und die Vorbereitung somit abgeschlossen.

5.2 Hauptprogramm

Die grundlegende Struktur des Hauptprogramms ist im in der nachfolgenden Abbildung (Abb. 11) dargestellten UML-Diagramm zu erkennen. Zentrales Element bildet die Klasse *TrafficAnalysis*, von der aus alle Funktionen für die dynamische Routenführung aufgerufen werden. In den folgenden Abschnitten wird auf die wichtigsten Klassen und deren Methoden eingegangen. Die detaillierte Dokumentation des gesamten Quelltextes befindet sich im Anhang (vgl. Anhang IDetaillierte Dokumentation der Implementierung).

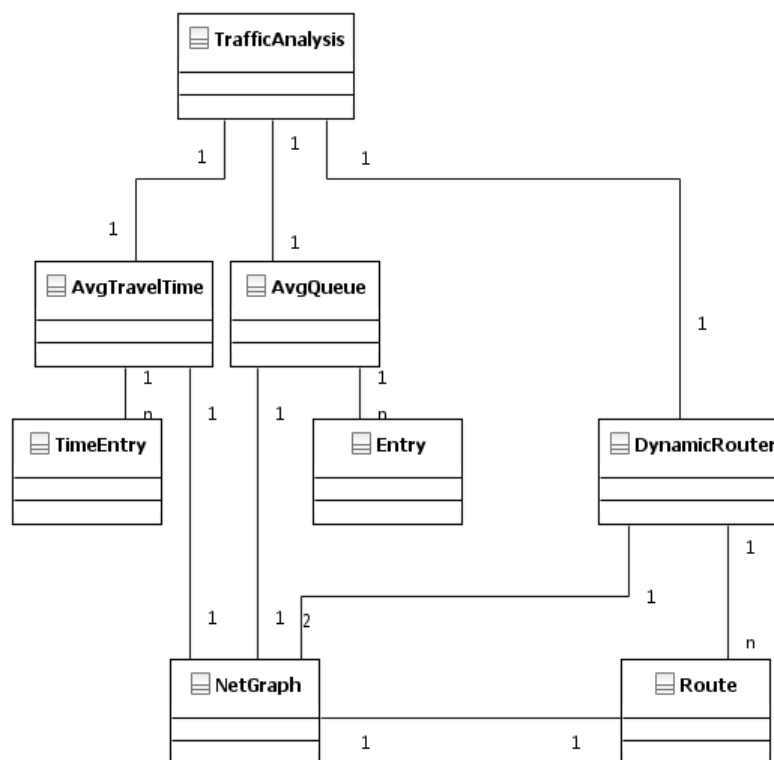


Abb. 11: UML-Diagramm für das Hauptprogramm

5.2.1 TrafficAnalysis.py – Basisklasse für das dynamische Routing

Die Klasse *TrafficAnalysis* bildet die Basis für das gesamte Dynamische Routing. Sie entspricht dem im Kapitel 2.3.2 erwähnten Pythonscript, das über die C2X-Schnittstelle von VISSIM aus aufgerufen wird. Dabei wird in jedem Simulationszeitschritt die Methode *processTimestep* ausgeführt. Für die Ermittlung aller Strecken des VISSIM-Netzes ist es nötig, zusätzlich zur C2X-Schnittstelle auch noch die COM-Schnittstelle zu verwenden. Diese wird neben den Klassen *TravelTimeQueue* und *AvgQueue* im ersten Zeitschritt initialisiert. Voraussetzung für die Verwendung der COM-Schnittstelle ist, dass VISSIM mit dem Parameter „-automation“ gestartet wird.

In *processTimestep* wird die Reisezeitermittlung angestoßen, in dem alle aktuellen Fahrzeugdaten an die *TravelTimeQueue* und *AvgQueue* weitergegeben und alle alten

Daten mithilfe von *deleteAllOldEntries* entfernt werden. Bevor in regelmäßigen Abständen (standardmäßig einmal in der Minute) das dynamische Routing aktiviert wird, gibt es ein Update des Netzgraphen, damit die zuvor ermittelten Reisezeiten auch beim dynamischen Routing verfügbar sind.

Eine Instanz der Klasse *DynamicRouter* ist für das dynamische Routing zuständig. Durch Aufruf von *routeIndividual* wird jedes einzelne Fahrzeug individuell und durch Aufruf von *routeCollective* alle Fahrzeuge im Kollektiv geroutet. Es ist stets nur eines dieser Verfahren zu verwenden, um Inkonsistenzen zu vermeiden.

5.2.2 NetGraph.py – Graphrepräsentation

Die Klasse *NetGraph* beinhaltet sämtliche Graphfunktionen, die beim dynamischen Routing sowie bei allen Zusatzprogrammen (vgl. Kapitel 5.1) verwendet werden. Die Funktionsweisen der bisher nicht erwähnten Methoden werden im Anhang genauer dokumentiert (vgl. Anhang I).

5.2.3 AvgTravelTime.py – Bestimmung der mittleren Reisezeit

Die Reisezeiten der einzelnen Fahrzeuge werden immer im Bezug auf eine Kante im Netzgraphen ermittelt. Das heißt, es wird für jede Strecke nur die Reisezeit des gesamten zugehörigen Streckenabschnitts angegeben. Eine Reisezeit ergibt sich aus den Zeitpunkten des Befahrens und des Verlassens eines Streckenabschnitts durch ein C2X-Fahrzeug. Diese Reisezeiten werden zusammen mit dem aktuellen Zeitstempel und der Fahrzeug-ID als *TimeEntry* (vgl. Anhang I.g) in einer zu dem entsprechenden Streckenabschnitt gehörenden Queue *TravelTimeQueue* für die in *Duration* angegebene Dauer (in Sekunden) gespeichert. Zusätzlich werden alle zu den *TimeEntries* gehörigen Reisezeiten kumuliert in *TimeSum* gespeichert. Gibt es veraltete Daten in der Queue, werden diese wieder aus der Queue entnommen und aus *TimeSum* subtrahiert, falls noch ausreichend aktuelle Daten vorhanden sind. Die Durchschnittsreisezeit wird in der Methode *getAvgTravelTime* ermittelt, indem *TimeSum* durch die Anzahl der Einträge in der Queue geteilt wird.

Zum Einfügen neuer Daten in die Queue dient die Methode *addTravelTime*. Diese muss in jedem Zeitschritt parametrisiert mit den aktuellen C2X-Daten Fahrzeug-ID, Strecken-ID und Zeitstempel aufgerufen werden. Mithilfe der globalen Variable *VehLinkTimeList* wird dann geprüft, ob das Fahrzeug im Bezug auf den letzten Zeitschritt einen Streckenabschnittswechsel vollzogen hat. Ist dies der Fall, wird ebenfalls mithilfe von *VehLinkTimeList* die Reisezeit für den verlassenen Streckenabschnitt ermittelt. Diese Daten werden dann wie zuvor beschrieben in die *TravelTimeQueue* und *TimeSum* eingefügt. Anschließend wird *VehLinkTimeList* aktualisiert.

Gelöscht werden Daten durch Aufruf der Methode *deleteAllOldEntries*. Auch diese Funktion sollte in jedem Zeitschritt aufgerufen werden, um sicherzustellen, dass keine veralteten Daten in der Queue bleiben. Es werden dabei alle *TimeEntries* aus der *TravelTimeQueue* entnommen, deren Zeitstempel älter als *Duration* ist. Hierbei wird die Zeitlinearität ausgenutzt. So werden nur solange Einträge (angefangen bei dem ältesten) auf Aktualität geprüft und gelöscht bis ein Eintrag gefunden wurde, der noch ausreichende Aktualität besitzt, weil die nachfolgenden Einträge später in die Queue eingetragen wurden und somit jünger sind. Mithilfe der entnommenen *TimeEntries* lassen sich die veralteten Reisezeiten ermitteln und aus *TimeSum* abziehen.

Ein Problem, das sich ergibt, wenn Fahrzeuge das Netz verlassen, muss gesondert behandelt werden. Da die Funktion *addTravelTime* nur für alle aktuell im Netz befindlichen Fahrzeuge aufgerufen wird, bleiben die Fahrzeuge unberücksichtigt, die das VISSIM-Netz im aktuellen Simulationsschritt verlassen haben. In *VehLinkTimeList* sind

sie jedoch weiterhin vertreten, können jedoch nicht mehr für die Reisezeitbestimmung verwendet werden. Das führt dazu, dass alle Streckenabschnitte am Ende des Netzes nicht mit Reisezeiten versehen werden. Um diesen Fall zu berücksichtigen, muss in jedem Zeitschritt die Methode *checkAllGoneVehicles* aufgerufen werden. Hier werden alle Fahrzeuge in *VehLinkTimeList*, die nicht mehr im Netz vorhanden sind, herausgenommen und jeweils die Methode *addTravelTime* aufgerufen.

5.2.4 AvgQueue.py – Ermittlung der mittleren Geschwindigkeit und Fahrzeuganzahl

Die Klasse *AvgQueue* ist im Aufbau sehr ähnlich zu *AvgTravelTime* aus dem vorigen Kapitel. Sie dient der Erfassung der durchschnittlichen Geschwindigkeit und Anzahl der Fahrzeuge auf den einzelnen Streckenabschnitten im VISSIM-Netz. Das Funktionsprinzip ist im Kapitel 4.3.2 erläutert. Mit der Methode *enqueue* werden die Fahrzeugdaten als *Entry* (vgl. Anhang I.h) zur Queue *EntryQueue* hinzugefügt. Zu den Fahrzeugdaten gehören der aktuelle Zeitstempel, die Geschwindigkeit und die Nummer der aktuellen Strecke des Fahrzeugs. In der Variable *VelocitySum* werden alle Geschwindigkeiten aufsummiert, die sich aktuell in der Queue befinden. In *TimestampCount* wird die Anzahl der verschiedenen in der *EntryQueue* vorhandenen Zeitstempel gezählt. Die Funktion *deleteAllOldEntries* dient dem Löschen von veralteten Einträgen *EntryQueue*. Die durchschnittliche Geschwindigkeit und Anzahl der Fahrzeuge wird in den Methoden *getAvgVelocityOfLink* und *getAvgVelocityOfEdge* bzw. *getAvgVehicleCountOfLink* und *getAvgVehicleCountOfEdge* ermittelt. Diese sind jeweils entweder mit der Nummer einer Strecke im VISSIM-Netz parametrisiert oder mit dem Label einer Kante im Netzgraphen. Die Unterscheidung in die beiden Varianten dient der Optimierung von internen Aufrufen.

5.2.5 DynamicRouter.py – Umsetzung der dynamischen Routenführung

In der Klasse *DynamicRouter* wird sowohl das individuelle (vgl. Kapitel 4.6.1), als auch das kollektive (vgl. Kapitel 4.6.2) Routing realisiert. Gestartet wird das dynamische Routing entsprechend durch Aufruf der Methoden *routeIndividual* bzw. *routeCollective*.

In der mit einem C2X-Fahrzeug parametrisierten Funktion *routeIndividual* wird zunächst geprüft, ob der Verlustzeitschwellwert für die Route des Fahrzeugs überschritten ist. Dies erfolgt mithilfe der Methode *getDelayBiggerThanThreshold*. Darin werden die Fahrzeiten im aktuellen Netzgraphen mit denen im bei der Initialisierung eingelesenen Standardgraphen verglichen. Wichtig ist hier, dass die Kantengewichte des Standardgraphen wirklich den Reisezeiten bei freier Fahrt entsprechen (vgl. Kapitel 5.1.2). Wird der Schwellwert überschritten, kann das eigentliche Routing durchgeführt werden. Dazu werden die Reisezeiten aller Alternativrouten, die sich in *RouteTab* befinden, ermittelt. Die Datenstruktur *RouteTab* ist dabei ähnlich aufgebaut wie *existingStaticRoutes* (vgl. 5.1.3) der Klasse *Netgraph*. Allerdings sind die Dictionaries mit den Identifikationsnummern der jeweiligen Routenentscheidung und Route indiziert, um schnelleren Zugriff auf die Alternativrouten zu erhalten. Wenn die schnellste gefundene Alternativroute eine kürzere Reisezeit besitzt als die aktuelle Route des Fahrzeugs, wird die Methode *getDriverAcceptance* aufgerufen. Mithilfe dieser kann das Verhalten des Fahrers bezüglich der Routenwahl simuliert werden. Die Funktion liefert einen Boolean-Wert, der angibt, ob der Fahrer die gefundene Alternativroute akzeptiert. Die genaue Umsetzung dieser Methode kann variiert werden. Es ist beispielsweise denkbar, die Akzeptanzrate von der Fahrzeitverbesserung abhängig zu machen. Für die

vorliegende Arbeit wird eine einfache Zufallsfunktion mit einer Akzeptanzwahrscheinlichkeit von 50 Prozent verwendet.

Die Methode *routeCollective* ruft zunächst die Funktion *getJammedEdges* auf, um die Kanten im Netzgraph zu ermitteln, bei denen die Reisezeit kleiner als der Wert der Variable *TRAFFIC_JAM_THRESHOLD* ist. Da die Geschwindigkeiten der Fahrzeuge beim Zugriff über die C2X-Schnittstelle in Meter pro Sekunde angegeben sind, wird die Variable *TRAFFIC_JAM_THRESHOLD* ebenfalls in dieser Einheit angegeben und auf den in Kapitel 4.6.2 definierten Schwellwert für Verkehrsstörungen von $\frac{30 \frac{\text{km}}{\text{h}}}{3,6} = 8,3 \frac{\text{m}}{\text{s}}$ gesetzt. Anschließend werden mithilfe der Funktion *getJammedRoutes* alle gestörten Routen ermittelt. Darauf aufbauend werden im Anschluss die entsprechenden Alternativrouten aus *StaticRoutes* ermittelt und in die ähnlich aufgebaute Datenstruktur *alternativeRoutes* übertragen. Dabei werden zu jedem Start-Ziel-Strecken-Paar nur einmal alle ungestörten Alternativrouten bestimmt. Bevor die Daten an die Fahrzeuge weitergegeben werden, wird noch ein Dictionary *jammedRouteNum* angelegt. Dieses enthält für jede Identifikationsnummer der Routenentscheidungen alle Routennummern, die gestört sind. Damit lassen sich die von der Störung betroffenen Fahrzeuge schneller identifizieren, da über die C2X-Schnittstelle nur auf die Identifikationsnummern der aktuellen Routenentscheidung bzw. Route des Fahrzeugs zugegriffen werden kann. Die Suche nach den betroffenen Fahrzeugen erfolgt mithilfe der Methode *FindAffectedVehicles*. Falls die aktuelle Route des Fahrzeugs in *jammedRouteNum* zu finden ist, wird die Funktion *sendAlternativesToVehicle* aufgerufen. Diese ermittelt die Reisezeiten der Alternativrouten, in denen der Streckenabschnitt, auf dem sich das Fahrzeug aktuell befindet, enthalten ist. Damit lassen sich dann alle anwendbaren Alternativen ermitteln. In der Realität wäre jetzt der Fahrer des Fahrzeuges für die Routenwahl verantwortlich. In der Implementierung gibt es zwei Möglichkeiten zur Realisierung. Entweder wird die Routenwahl über die Funktion *getDriverChoice* oder durch die Suche nach der schnellsten Alternativroute und Verwendung der bereits für das individuelle Routing genutzten Funktion *getDriverAcceptance* festgelegt. Erstere Methode ist hier nicht genauer spezifiziert. Es kann aber dabei nach einer beliebigen Heuristik zwischen der aktuellen Route und den möglichen Alternativrouten entschieden werden. Die in dieser Arbeit verwendete Funktion *getDriverAcceptance* simuliert hingegen die Fahrerentscheidung durch eine zufällig generierte Akzeptanzrate.

5.2.6 Route.py – Routenrepräsentation im Netzgraph

Die Repräsentation der VISSIM-Routen im Netzgraph wird mithilfe der Klasse *Route* definiert. Die Klasse besitzt alle wesentlichen Attribute einer VISSIM-Route. Dazu zählen *Start* und *Dest*, welche die Nummer der Start- und Zielstrecke angeben, sowie die Identifikationsnummern der zugehörigen Routenentscheidung und Route *routeDecisionNum* und *routeNum*. Weiterhin sind die zur Route gehörenden Kanten in der Liste *EdgeList* vermerkt. Die Funktion *getViaTracks* gibt einen String zurück, der alle zu den Kanten der Route gehörenden VISSIM-Strecken beinhaltet. Dieser liegt in dem im Kapitel 4.5 beschriebenen Format vor und kann somit in der Methode *saveAllStaticRoutes* (vgl. Kapitel 5.1.3) zum Speichern der VISSIM-Route verwendet werden.

6 Durchführung der Verkehrssimulation mit C2X-basierter dynamischer Routenführung

Um die Auswirkungen der C2X-basierten dynamischen Routenführung auf die Verkehrseffizienz zu untersuchen, wird im Folgenden eine kurze Simulationsstudie vorgenommen. Diese liefert qualitative Aussagen über den Nutzen der C2X-basierten dynamischen Routenführung gegenüber einer Routenführung ohne dieses System. Um die statistische Sicherheit der Aussagen zu erhöhen, wären aufwändige Mehrfachsimulationen mit einer Vielzahl an Szenarien nötig. Dies würde jedoch den zeitlichen Rahmen dieser Arbeit sprengen. Daher wurden die im folgenden Abschnitt erläuterten Annahmen getroffen und die entsprechenden Konfigurationen vorgenommen.

6.1 Szenario-abhängige Konfiguration der Simulationsparameter

6.1.1 Räumliche und zeitliche Eingrenzung des Untersuchungsgebiets

Für diese Simulationsstudie wird das in Kapitel 4.1 vorgestellte VISSIM-Netz als Untersuchungsgebiet gewählt. Es ist eine sehr gute Abbildung des realen Verkehrsnetzes im Großraum Frankfurt. Damit lassen sich auch die im Rahmen von sim^{TD} zur Verfügung stehenden Detektordaten der Hauptverkehrswege im Untersuchungsgebiet für die Definition von Zuflüssen und Routenentscheidungen nutzen.

Die verwendeten Detektordaten umfassen das gesamte Jahr 2009 der im Testgebiet liegenden Bundesautobahnen A3, A5, A66 und A661. Sie liegen jedoch zunächst in einem ungeeigneten Format vor und müssen entsprechend aufbereitet werden. So enthalten sie für jede Fahrspur, über 15 Minuten gemittelte Werte der Geschwindigkeit und der Anzahl der Fahrzeuge. Dabei wurde außerdem zwischen Personen- und Lastkraftwagen unterschieden. Für die Definition von Zuflüssen und Routenentscheidungen sind jedoch die Größen Verkehrsstärke, Lastkraftwagenanteil und Abbiegerate relevant. Daher werden die Werte der zu einer Strecke gehörende Fahrstreifen zusammengefasst und für alle Fahrzeuge über vier Zeitintervalle hinweg zu einer Stunde aggregiert. Weiterhin werden die Anteile der Lastkraftwagen bzw. der abbiegenden Fahrzeuge im Bezug auf die Gesamtfahrzeugzahl bestimmt.

Auf Basis der aufbereiteten Daten werden die Zuflüsse definiert. Da Fahrzeuge nur am Rand in das Netz einfahren können, werden auch nur die Daten der entsprechenden Detektoren benötigt. Für jeden Zufluss wird eine Routenentscheidung erstellt, welche die Fahrzeuge entsprechend der ermittelten Abbiegeraten auf verschiedene Routen führt. Die Zuflüsse auf den Strecken, für die keine Detektordaten vorliegen, werden auf die gleichen Werte gesetzt, die in der Vorsimulation (vgl. 4.4) verwendet wurden, um eine grundlegende Netzbelastung festzulegen.

Zeitlich muss die Studie begrenzt werden, da der Rechenaufwand in einem derart großen Netz mit entsprechend vielen Fahrzeugen sehr hoch ist. Um trotzdem aussagekräftige Ergebnisse zu erhalten, wird die Simulationszeit auf 90 Minuten gesetzt. Je nach Szenario werden die Parameter der Zuflüsse entsprechend der ermittelten Werte aus den Detektordaten für einen bestimmten Zeitraum festgelegt. Die Wahl der Szenarien wird im folgenden Abschnitt erläutert.

6.1.2 Wahl der Szenarien

Die Wahl der Szenarien hängt von den zu ermittelnden Aussagen ab. Die wichtigste Fragestellung ist, welchen Nutzen der Einsatz von C2X-basierter dynamischer Routenführung für die Reisezeit des einzelnen Fahrzeugs und für den Gesamtverkehrsfluss hat.

Durchführung der Verkehrssimulation mit C2X-basierter dynamischer Routenführung

Diese sehr grobe Fragestellung sollte weiter differenziert werden. So gibt es bezüglich des Nutzens verschiedene mögliche Abhängigkeiten.

Ein wesentlicher Faktor ist die Verkehrsdichte. Je mehr Fahrzeuge das Verkehrsnetz befahren, desto größer ist vermutlich der Zeitgewinn, wenn eine weniger befahrene Route gewählt wird. Sind dagegen nur wenige Fahrzeuge im Netz, wird das dynamische Routing voraussichtlich nur geringe Vorteile bieten. Um diesen Sachverhalt genauer zu untersuchen, wird die Simulationsstudie die Szenarien „dichter Verkehr“ und „freies Fahren“ umfassen. Für ersteres Szenario werden die Zuflüsse entsprechend der Verkehrsstärke zu einer Hauptverkehrszeit definiert. Hierfür werden die Detektordaten nach einem Zeitraum durchsucht, in dem alle Streckenabschnitte mindestens etwa 90 Prozent der maximalen Verkehrsstärke aufweisen. Ein solcher Zeitraum wurde am 09.06.2009 zwischen 7:00 Uhr und 8:30 Uhr ermittelt. Für das Szenario „freies Fahren“ wird ein Zeitraum gewählt, bei dem die Verkehrsstärke auf allen Strecken im Bereich von 30 bis 50 Prozent der maximalen Verkehrsstärke liegt. Hierfür wurde der Zeitraum am 04.06.2009 von 19:00 Uhr bis 20:30 Uhr gewählt.

Einen großen Einfluss auf den Nutzen der C2X-basierten dynamischen Routenführung hat auch die Ausstattungsrate, also der Anteil der Fahrzeuge mit C2X-Funktionalität an der Gesamtfahrzeuganzahl. Darum sollten die Auswirkungen verschiedener Ausstattungsraten untersucht werden. Am wichtigsten ist dabei der Vergleich zwischen 0 und 20 Prozent. Bei einer Ausstattungsrate von 20 Prozent sollten sich bereits erste Verbesserungen hinsichtlich der Reisezeiten ergeben. Eine höhere Rate lässt sich derzeit aufgrund der starken Rechenabhängigkeit von der Anzahl der C2X-Fahrzeuge nicht mit ausreichender Genauigkeit in überschaubarer Zeit simulieren.

Ferner gilt es bei der Wahl der Szenarien stets zwischen kollektivem und individuellem Routing zu unterscheiden, um die Unterschiede der beiden Funktionen im Bezug auf die Veränderungen des Verkehrsflusses zu beurteilen. Der Verlustzeitschwellwert für die Routenneuberechnung beim individuellen Routing soll dabei mit verschiedenen Werten im Bereich von 30 bis 600 Sekunden belegt werden, um einen Richtwert für das entsprechende verkehrnetzspezifische Optimum zu ermitteln.

Die folgende Tabelle zeigt die verschiedenen Kombinationen der Parameter und die für die Auswertung gewählten Szenarien.

Verkehrsstärke Ausstattungsrate	90% – 100% 09.06.2009 7:00-8:30	30% – 50% 04.06.2009 19:00-20:30
0%	Szenario 1	Szenario 3
10%		
20%	Szenario 2	Szenario 4
50%		
100%		

Tabelle 1: Kombinationen der Simulationsparameter und Szenariowahl

Durchführung der Verkehrssimulation mit C2X-basierter dynamischer Routenführung

Die sonstigen relevanten Simulationsparameter werden wie folgt festgelegt:

Funktion	Parameter	Wert
VISSIM	Simulationsdauer	5400 Sekunden
	Berechnungsfrequenz	5 Zeitschritte/Simulationssekunde
	Startzufallszahl	42
Verkehrslageerfassung	Frequenz der Verkehrslageerfassung (TrafficAnalysis.py)	1 Schritt/Simulationssekunde
	Frequenz der Routingvorgänge (TrafficAnalysis.py)	1 Routing/Simulationsminute
	Zeitintervall für die Reisezeitermittlung (AvgTravelTime)	600 Sekunden
	Zeitintervall für die Erfassung der Geschwindigkeit und Fahrzeuganzahl (AvgQueue)	600 Sekunden
Alternativroutensuche (NetGraph.py)	Zeitfaktor für die Wahl der relevanten Alternativrouten	1,5
Dynamisches Routing (DynamicRouter.py)	Routenakzeptanzrate für Simulation der Routenwahl des Fahrers	50 Prozent
Kollektives Routing (DynamicRouter.py)	Grenzwert für die Definition einer gestörten Strecke	30/3,6 \approx 8,3 Meter/Sekunde

Tabelle 2: Festlegung der Szenario-unabhängigen Simulationsparameter

6.2 Wahl der Messgrößen

Die Auswertung der Simulationsstudie wird sich auf den Vergleich der im vorigen Abschnitt beschriebenen Szenarien beschränken. Dabei sollen die Unterschiede für die einzelnen Fahrzeuge im mikroskopischen Bereich sowie makroskopisch, für den gesamten Verkehr herausgestellt werden.

6.2.1 Mikroskopische Messgrößen

Im mikroskopischen Bereich stellen vor allem Reisezeit und Geschwindigkeit der einzelnen Fahrzeuge eine wesentliche Rolle dar. Die Durchschnittsgeschwindigkeiten werden jedoch ebenso makroskopisch betrachtet (vgl. 6.2.2). Da die mikroskopische Betrachtung der Geschwindigkeiten im Allgemeinen nur für die Auswertung einzelner Situationen geeignet ist und das dynamische Routing im Rahmen dieser Arbeit global zum Einsatz kommt, wird sich bei der mikroskopischen Untersuchung auf die Reisezeit als Messgröße beschränkt. Hierfür werden in VISSIM zehn verschiedene Messquerschnitte gesetzt. Diese dienen der Reisezeitbestimmung für beide Richtungen auf den Autobahnen A3, A5, A66 und A661 sowie auf der Bundesstraße B3. Damit werden die wichtigsten Routen abgedeckt. Die B3 bildet dabei eine Ausnahme, da über sie keine der definierten statischen Routen führt. Jedoch werden voraussichtlich die C2X-Fahrzeuge bei einem Störfall auf der A5 oder der A661 über diese umgeleitet, sodass eine Betrachtung der dortigen Reisezeiten durchaus Aussagen über die Auswirkungen der dynamischen Routenführung auf Nebenstraßen liefert.

6.2.2 Makroskopische Messgrößen

Als makroskopische Messgrößen werden die wichtigsten Kenngrößen für die Analyse des Verkehrsflusses gewählt. Diese sind die Durchschnittsgeschwindigkeit der Fahrzeuge v , die Verkehrsstärke q und die Dichte ρ des Verkehrs. In sogenannten Fun-

damentaldiagrammen lassen sich die Beziehungen zwischen diesen Größen darstellen und damit Aussagen bezüglich der Verkehrseffizienz auf einem bestimmten Streckenabschnitt treffen. Hierfür werden in VISSIM für 9 verschiedene Streckenabschnitte jeweils in Hin- und Rückrichtung Streckenauswertungen aktiviert. Die Streckenabschnitte befinden sich auf den im vorigen Abschnitt erwähnten Autobahnen und Bundesstraßen.

Um die drei Arten von Fundamentaldiagrammen geeignet interpretieren zu können, sei zunächst deren allgemeines Aussehen erläutert:

Verhältnis von Geschwindigkeit zur Verkehrsdichte, $v(\rho)$:

Für die folgenden Aussagen wird angenommen, dass alle Verkehrsteilnehmer auf die maximal erlaubte oder mögliche Geschwindigkeit v_{max} beschleunigen, wenn die Verkehrsdichte gegen Null geht. Im Gegenzug wird die maximale Dichte ρ_{max} genau dann angenommen, wenn die Fahrzeuge im Stau stehen, da bei höheren Geschwindigkeiten ein entsprechender Sicherheitsabstand eingehalten wird. Unter der sinnvollen Annahme, dass die Geschwindigkeit der Fahrzeuge mit zunehmender Dichte monoton fällt, lässt sich folgender linearer Zusammenhang erkennen:

$$v(\rho) = v_{max} \left(1 - \frac{\rho}{\rho_{max}}\right)$$

Das entsprechende Fundamentaldiagramm ist in folgender Abbildung dargestellt:

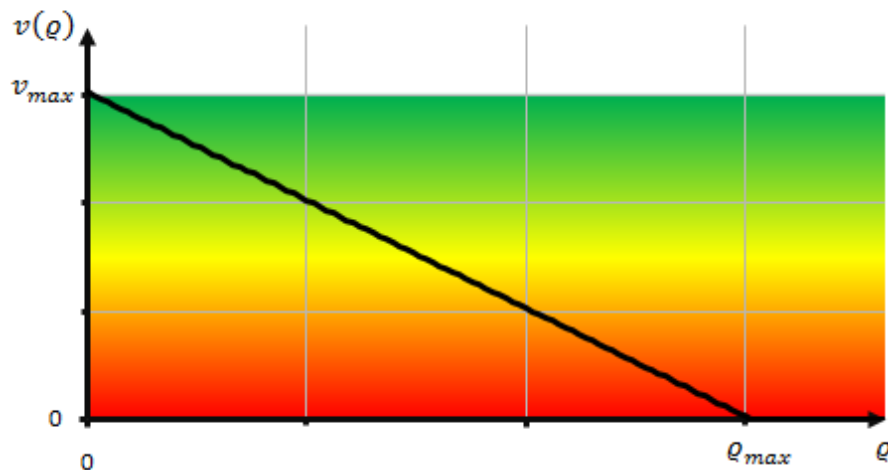


Abb. 12: Fundamentaldiagramm Geschwindigkeit zur Verkehrsdichte

Das Diagramm wird in die drei farblich hervorgehobenen Bereiche unterteilt. Der grüne Bereich steht für freien Verkehrsfluss, der gelbe Bereich für den sogenannten gebundenen Verkehrsfluss und der rote Bereich steht für Stau. Je größer der Anteil des Graphen im grünen Bereich ist, desto besser stellt sich die Verkehrssituation dar, denn dann sind sowohl die Streckenauslastung als auch die Geschwindigkeit der Fahrzeuge relativ groß.

Verhältnis von Verkehrsstärke zur Verkehrsdichte, $q(\rho)$:

Um die Verkehrsstärke, also die Anzahl der Fahrzeuge an einem Ort über einen bestimmten Zeitraum gesehen, in Abhängigkeit der Dichte zu bestimmen, wird die im vorigen Abschnitt ermittelte Dichte-abhängige Geschwindigkeit mit der Verkehrsdichte multipliziert. Nach einsetzen ergibt sich somit folgender quadratischer Zusammenhang:

$$q(\rho) = v_{max} \left(1 - \frac{\rho}{\rho_{max}}\right) \rho$$

Das entsprechende Fundamentaldiagramm ist in folgender Abbildung dargestellt:

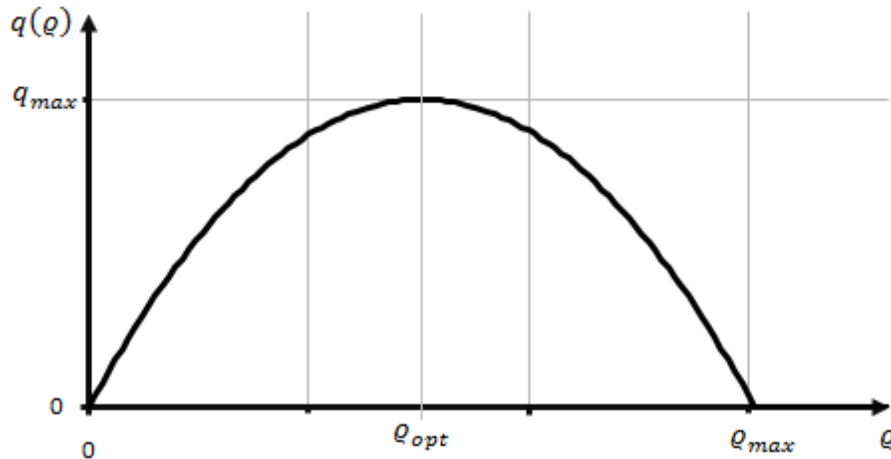


Abb. 13: Fundamentaldiagramm Verkehrsstärke zur Verkehrsdichte

In diesem Diagramm lässt sich erkennen, dass die Verkehrsstärke bei einer gewissen optimalen Dichte ρ_{opt} maximal wird. Sowohl bei geringerer Verkehrsdichte, als auch bei einer größeren Dichte ist die Verkehrsstärke kleiner als q_{max} . Da die Geschwindigkeit bei weniger dichtem Verkehr höher ist, wird der Bereich links des Optimums in der Verkehrsplanung besser bewertet als der Bereich rechts der optimalen Dichte. Jedoch ist die Verkehrssituation im Bereich um ρ_{opt} am besten. Das heißt, die Breite des Bereiches nahe der optimalen Verkehrsstärke ist ein Indikator für die Güte der Verkehrslage.

Verhältnis von Verkehrsstärke zur Geschwindigkeit, $q(v)$:

Im dritten Fundamentaldiagramm wird die Verkehrsstärke in Abhängigkeit der Geschwindigkeit dargestellt. Ausgehend von einem linearen Zusammenhang zwischen Geschwindigkeit und Verkehrsdichte zeigt sich ein ähnliches Diagramm wie beim Verhältnis zwischen Verkehrsstärke und Dichte (vgl. Abb. 14).

Es zeigt sich, dass die Verkehrsstärke für eine bestimmte Geschwindigkeit v_{opt} maximal ist, während sie für alle kleineren Werte monoton steigt und für $v > v_{opt}$ monoton fällt. Auch hier gilt, dass die breite des Bereiches, in dem die Verkehrsstärke nahe an q_{max} ist, die Güte der Verkehrssituation angibt. Dabei ist diese außerdem umso besser, je weiter rechts q_{max} liegt, denn dann ist die Geschwindigkeit bei maximaler Verkehrsstärke größer als weiter links.

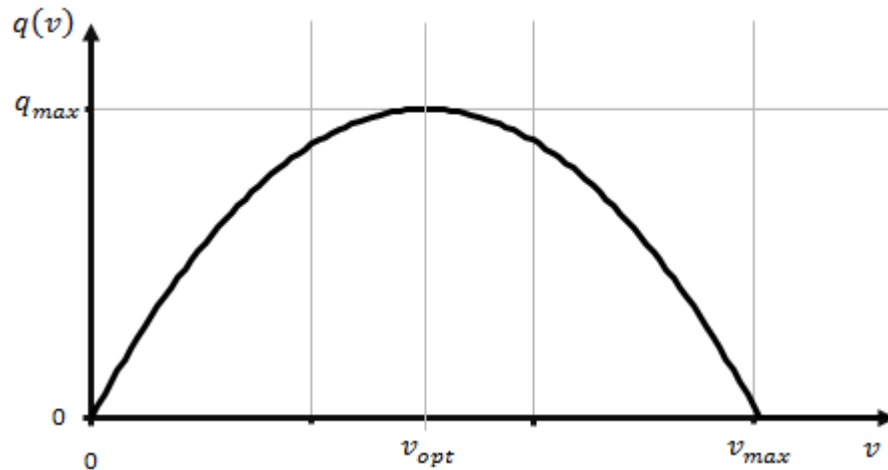


Abb. 14: Fundamentaldiagramm Verkehrsstärke zur Geschwindigkeit

Neben den Fundamentaldiagrammen gibt es noch weitere Möglichkeiten für die Auswertung der makroskopischen Messgrößen. Eine davon sei im Folgenden kurz vorgestellt:

Die Geschwindigkeit lässt sich sowohl räumlich (zu einer bestimmten Zeit), als auch zeitlich (an einem bestimmten Ort) als Graph in einem Diagramm darstellen. Eine gute Methode, um die Bildung von Staus zu beobachten, ist die Farbdarstellung der Verkehrsgeschwindigkeit in einem Weg-Zeit-Diagramm. Jeder gemessenen Geschwindigkeit wird dabei eine Farbe zugeordnet, die entsprechend des Orts und des Messzeitpunktes im Diagramm abgebildet wird. Im Vergleich zweier solcher Diagramme auf Basis unterschiedlicher C2X-Ausstattungsraten können Aussagen bezüglich einer Verkürzung bzw. Verlängerung von Staus durch die Nutzung der C2X-basierten dynamischen Routenführung getroffen werden.

7 Ergebnisse der Simulationsstudie

7.1 Vergleich der Szenarien „dichter Verkehr“ und „freies Fahren“

Um einen grundlegenden Eindruck über die unterschiedlichen Verkehrssituationen bei dichtem und bei freiem Verkehr zu erhalten, wird im Folgenden ein qualitativer Vergleich der beiden Szenarien 1 und 3 aus Kapitel 6.1.2 durchgeführt.

Zunächst werden die Reisezeiten der beiden Szenarien für einen ausgewählten Streckenabschnitt gegenübergestellt. Im folgenden Diagramm (vgl. Abb. 15) wurden die beiden Reisezeitmessreihen für die Autobahn A5 Richtung Süden zeitlich aufgetragen. Dabei wurde als Zeitpunkt der Messung die Simulationssekunde gewählt, bei der das jeweilige Fahrzeug den Zielquerschnitt überfährt.

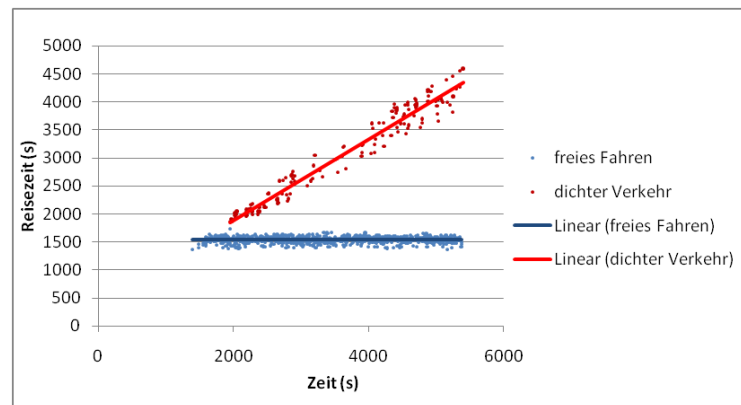


Abb. 15: Reisezeiten für A5 Richtung Süden mit Regressionsgeraden
rot: dichter Verkehr, blau: freies Fahren

Im abgebildeten Diagramm lassen sich deutliche Unterschiede in den Reisezeiten erkennen. Während die Reisezeiten bei freier Fahrt (blau) relativ konstant bleiben, steigt die Fahrtdauer bei dichtem Verkehr (rot) sehr stark an.

Die Fahrzeuge werden in der Simulation zunächst immer auf die schnellste Route geführt. Daher lässt sich im Bezug auf die Anwendung der C2X-basierten dynamischen Routenführung vermuten, dass der Nutzen bei freier Fahrt nur minimal ist. Auch die geringe Streuung spricht dafür, dass sich bei dieser Verkehrslage durch die Verwendung der dynamischen Routenführung kaum Vorteile erzielen lassen. Bei dichtem Verkehr hingegen zeigt sich ein größeres Einsparpotenzial, denn je länger die Reisedauer ist, desto eher könnten auch deutlich längere Alternativrouten als Ausweichmöglichkeiten dienen und somit die Reisezeit verringern.

Für die nachfolgenden Analysen wird der Teilabschnitt der Autobahn A5 zwischen den Anschlussstellen „Ober-Mörlen“ und „Friedberg“ (vgl. Abb. 20 im Anhang, Seite 57) als Untersuchungsgrundlage ausgewählt.

Die räumliche und zeitliche Darstellung der Durchschnittsgeschwindigkeit auf diesem Abschnitt zeigt ebenfalls einen deutlichen Unterschied zwischen den beiden Szenarien „freies Fahren“ und „dichter Verkehr“ (vgl. Abb. 16). Bei freier Fahrt sind große Bereiche des Diagramms in Grüntönen gefärbt. Dies lässt erkennen, dass die Geschwindigkeiten sehr hoch sind. Lediglich am Ende des Streckenabschnitts (im Diagramm oben), etwa bei Streckenkilometer zehn, fällt die Geschwindigkeit in den Bereich von 30 bis 90 Kilometer pro Stunde. Aufgrund der Ortsfestigkeit des Anfangspunktes lässt sich dies jedoch eher auf eine lokale Gegebenheit zurückführen als auf zu hohes Verkehrsaufkommen. Bei dichtem Verkehr hingegen deuten die mit zunehmender Dauer immer großflächigeren roten Bereiche auf die Entstehung eines größeren Staus hin. Au-

ßerdem lassen sich anhand der hellroten Bereiche innerhalb der dunkelroten Gebiete die typischen Stauwellen erkennen, die sich stromaufwärts bewegen.

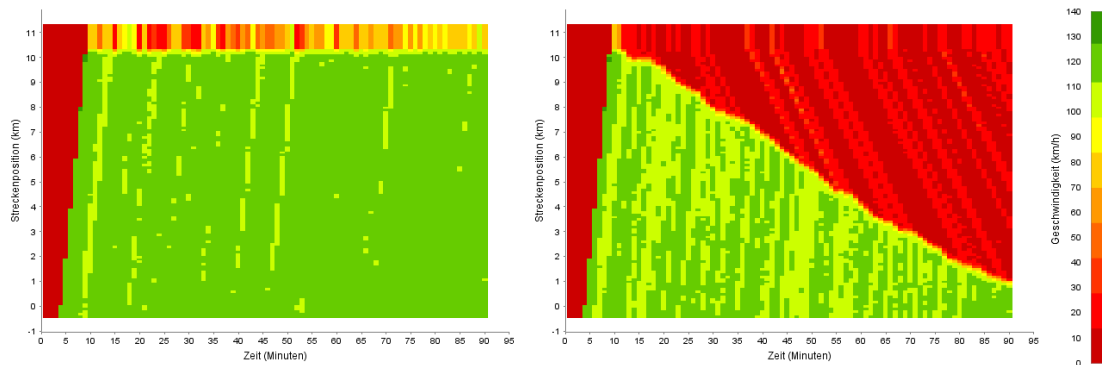


Abb. 16: Weg - Zeit - Geschwindigkeit - Diagramme für gewählten Abschnitt der A5 links: „freies Fahren“, rechts: „dichter Verkehr“

Im Folgenden sollen die unterschiedlichen Verkehrssituationen auf dem gewählten Autobahnabschnitt anhand von Fundamentaldiagrammen (vgl. 6.2.2) makroskopisch analysiert werden.

Die Punktmengen in den beiden Geschwindigkeit-Verkehrsdichte-Diagrammen (vgl. Abb. 17) sind sich bis zu einer Dichte von etwa 130 Fahrzeugen pro Kilometer sehr ähnlich. Damit lässt sich erkennen, dass voraussichtlich ein für den Streckenabschnitt charakteristischer Zusammenhang zwischen Geschwindigkeit und Verkehrsdichte existiert. Im Gegensatz zum Szenario „dichter Verkehr“ ist die Verkehrsdichte bei freier Fahrt nicht über diesen Wert gestiegen. Dies weist darauf hin, dass die Kapazität der Strecke bei freier Fahrt nicht ausgelastet wird.

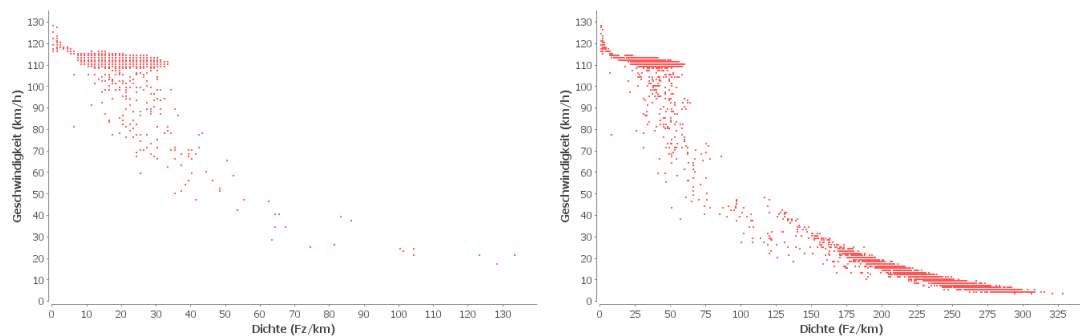


Abb. 17: Geschwindigkeit - Dichte - Diagramme für gewählten Abschnitt der A5 links: „freies Fahren“, rechts: „dichter Verkehr“

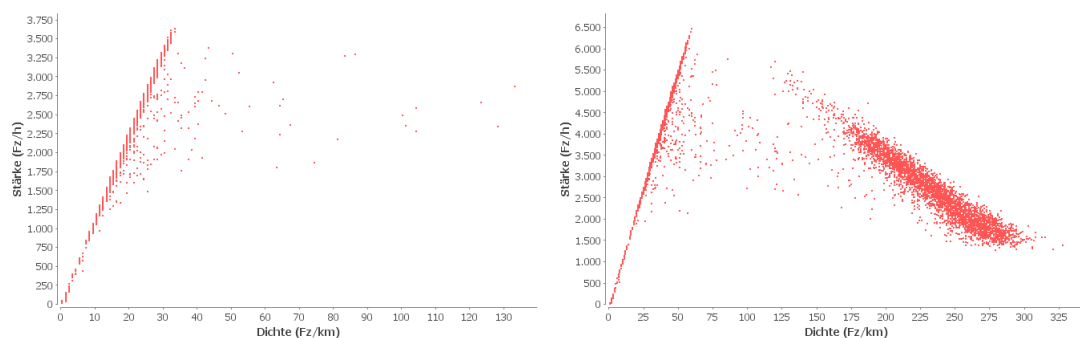


Abb. 18: Verkehrsstärke - Dichte - Diagramme für gewählten Abschnitt der A5 links: „freies Fahren“, rechts: „dichter Verkehr“

An den Verkehrsstärke-Verkehrsdichte-Diagrammen (vgl. Abb. 18) der beiden betrachteten Szenarien lässt sich sehr gut erkennen, wie gut die Verkehrslage bei freier Fahrt im Gegensatz zum dichten Verkehr ist. Zwar liegt der maximale Fluss im rechten Diagramm höher als im linken, jedoch befindet sich ein Großteil der aufgezeichneten Werte im Bereich von 175 bis 300 Fahrzeugen pro Kilometer. Wie im Kapitel 6.2.2 erläutert, ist dieser Bereich schlechter zu bewerten als der Bereich gleicher Verkehrsstärke bei niedrigerer Dichte. Im linken Diagramm befinden sich fast alle Werte im besseren Bereich.

Zuletzt seien die Verkehrsstärke-Geschwindigkeit-Diagramme (vgl. Abb. 19) betrachtet. Auch hier ist zu erkennen, dass bei freier Fahrt der Anteil der Werte im Bereich größerer Geschwindigkeit und Verkehrsstärke höher ist als bei dichtem Verkehr. Das heißt, im Szenario „dichter Verkehr“ ist ein wesentlicher Anteil der Werte bei gleicher Verkehrsstärke im (schlechteren) Bereich niedrigerer Geschwindigkeit. Dementsprechend wird damit bestätigt, dass die Verkehrssituation im Szenario „freies Fahren“ besser ist als im Szenario „dichter Verkehr“.

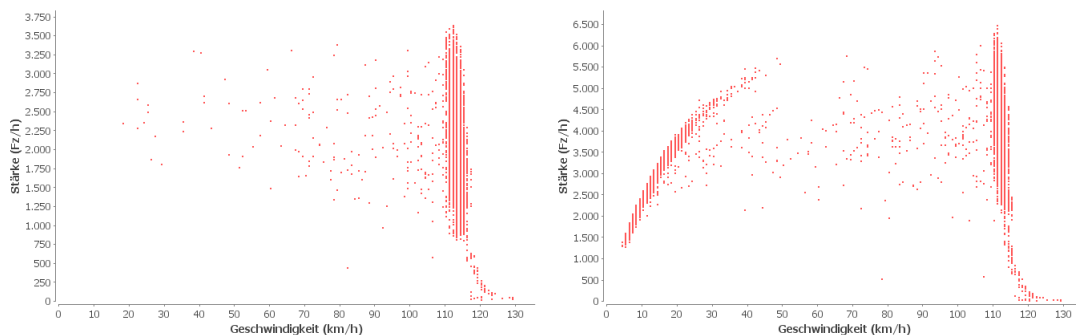


Abb. 19: Verkehrsstärke - Geschwindigkeit - Diagramme für gewählten Abschnitt der A5 links: „freies Fahren“, rechts: „dichter Verkehr“

7.2 Vergleich der Szenarien mit und ohne dynamischer Routenführung

Im Folgenden sollen die im Kapitel 6.1.2 vorgestellten Szenarien 1 und 2 bzw. 3 und 4 miteinander verglichen werden. Bei den Szenarien mit C2X-Unterstützung muss dabei zwischen kollektiver und individueller Routenführung unterschieden werden und die Auswirkungen des unterschiedlichen Routings gegenübergestellt werden. Für das individuelle Routing sind weiterhin Szenarien mit verschiedenen Verlustzeitschwellwerten hinsichtlich der Verkehrseffizienz miteinander zu vergleichen.

Leider musste kurzfristig eine nicht absehbare Fehlfunktion bei der Routenzuweisung über die C2X-Schnittstelle in VISSIM festgestellt werden. Da diese Funktion die Grundlage dafür bildet, Fahrzeuge auf eine Alternativroute zu führen, ist der sinnvolle Einsatz der C2X-basierten dynamischen Routenführung derzeit nicht möglich. Das in dieser Arbeit entwickelte Programm wurde im Rahmen der zeitlichen Möglichkeiten ausführlich getestet, sodass ein Fehler innerhalb des Programms auszuschließen ist. Auch nach längerer Suche konnte jedoch die Ursache dieses sehr speziellen Fehlers nicht gefunden werden, sodass von einem VISSIM-internen Problem auszugehen ist.

Die C2X-Schnittstelle ist erst seit der aktuellen Version 5.20 in VISSIM verfügbar. Für die Arbeit im Rahmen von sim^{TD} wird diese am Lehrstuhl für Verkehrstechnik sehr stark eingesetzt. Dabei ist es essentiell, Fehler in der neu entwickelten Schnittstelle zu erkennen und in Absprache mit der Firma PTV zu beseitigen. Diese Arbeit trägt somit als Test der C2X-Schnittstellenfunktionalität zum Gelingen von sim^{TD} bei. Da eine Korrespondenz mit PTV zur Fehleranalyse im zeitlichen Rahmen dieser Arbeit nicht realisierbar ist, muss gegenwärtig auf eine ausführliche Simulationsstudie verzichtet werden.

8 Zusammenfassung und Ausblick

Im Rahmen der vorliegenden Arbeit gelang es, ein Python-Programm zu entwickeln, welches die C2X-basierte dynamische Routenführung in die mikroskopische Verkehrssimulation VISSIM integriert. Dabei wurden zwei Funktionen implementiert, welche jeweils einen der beiden Anwendungsfälle – individuelle und kollektive Routenführung – simulieren.

Die vorliegende Arbeit ist in das Forschungsprojekt sim^{TD} eingebettet, welches die Technische Universität München als Forschungspartner begleitet. Zu den Aufgaben des Lehrstuhls für Verkehrstechnik zählt dabei die Erfassung der Auswirkungen einer großflächigen Nutzung von C2X-Kommunikation. Die Simulation von entwickelten sim^{TD} -Funktionen bildet dabei die Grundlage für die Nutzenanalyse. Somit konnte mit der Integration der sim^{TD} -Funktion des C2X-basierten dynamischen Routings in VISSIM ein wesentlicher Beitrag für dieses Projekt erbracht werden.

Es wurde ein Verkehrsnetz für VISSIM identifiziert, welches für den Einsatz dynamischer Routenführung geeignet ist und gleichzeitig direkt für sim^{TD} eingesetzt werden kann, da es das Testgebiet sehr gut abbildet. Durch die Entwicklung einer generischen Funktion zur Generierung einer geeigneten Graphrepräsentation des gewählten Netzes konnte die Basis für die weiteren Programmteile gelegt werden.

Es gelang Algorithmen und Datenstrukturen zu erstellen, die eine zielgerichtete Analyse der Verkehrslage in VISSIM ermöglichen und daraus Kantengewichte im Netzgraphen erzeugen. Mithilfe einer Vorsimulation konnte der Graph mit Standardgewichten versehen werden. Es wurden Traversierungsalgorithmen implementiert, die auf Basis dieser Standardgewichte für alle vorhandenen statischen Routen in VISSIM sinnvolle Alternativrouten bestimmen.

Gemäß den sim^{TD} -Spezifikationsdokumenten F_1.2.3 [4] und F_1.3.1 [5] wurden schließlich Routingalgorithmen entwickelt, die es erlauben, Verkehrssimulationen unter Verwendung individueller und kollektiver C2X-basierter dynamischer Routenführung in VISSIM durchzuführen.

Abschließend konnte eine kurze Simulationsstudie durchgeführt werden, die einen grundlegenden Vergleich der Verkehrslage zweier ausgewählter Szenarien beinhaltet. Hierfür wurden reale Detektordaten aus dem Untersuchungsgebiet verwendet, um Zuflüsse und Routenentscheidungen in VISSIM festzulegen. Durch die Auswahl der passenden Tageszeiten konnten die Szenarien „freies Fahren“ und „dichter Verkehr“ generiert werden. Dabei wird der typische Werktagsverkehr zum einen in den Abendstunden und zum anderen zur morgendlichen Hauptverkehrszeit simuliert. Die Ergebnisse zeigen die deutlichen Unterschiede der beiden Szenarien auf. Während sich bei freier Fahrt nur wenige Verbesserungsmöglichkeiten durch die Verwendung von C2X-basierter dynamischer Routenführung bieten, ist es durchaus möglich, durch Nutzung der C2X-Technologie bei dichtem Verkehr einen deutlichen Reisezeitgewinn zu erzielen.

Das in dieser Arbeit entwickelte Programm wird am Lehrstuhl für Verkehrstechnik für Verkehrssimulationen in verschiedenen Bereichen eingesetzt. Dazu zählen die Bestimmung von Richtwerten für den praktischen sim^{TD} -Feldtest und die Simulation eines großflächigen Einsatzes der C2X-Technologie. Dabei lassen sich verschiedene Programmparameter entsprechend der Testergebnisse kalibrieren. Darüber hinaus sind Möglichkeiten zur Erweiterung des Programms gegeben. So ist beispielsweise unter Verwendung unterschiedlicher Fahrermodelle eine realistische Simulation von Fahrerentscheidungen bezüglich der Routenwahl möglich.

Ferner wurde durch diese Arbeit die neu entwickelte C2X-Schnittstelle in VISSIM getestet und so erste Erfahrungswerte für die Entwicklung weiterer Programme am Lehrstuhl für Verkehrstechnik gesammelt.

Abbildungsverzeichnis

Abb. 1: Testgebiet sim^{TD} [2].....	10
Abb. 2: Einsatz von C2X-Kommunikation bei einem Verkehrsunfall [3]	11
Abb. 3: WIEDEMANN-Modell [6].....	13
Abb. 4: Allgemeine Form des C2X-Pythonscripts	15
Abb. 5: VISSIM-Netz des sim^{TD} -Testgebiets	18
Abb. 6: Einmündung in VISSIM-Netz	19
Abb. 7: Graph G_1	20
Abb. 8: Graph G_2 ,.....	20
Abb. 9: Graph mit Zwischen- und Verzweigungsknoten	21
Abb. 10: Graph ohne Zwischenknoten	21
Abb. 11: UML-Diagramm für das Hauptprogramm.....	29
Abb. 12: Fundamentaldiagramm Geschwindigkeit zur Verkehrsdichte.....	36
Abb. 13: Fundamentaldiagramm Verkehrsstärke zur Verkehrsdichte.....	37
Abb. 14: Fundamentaldiagramm Verkehrsstärke zur Geschwindigkeit	38
Abb. 15: Reisezeiten für A5 Richtung Süden mit Regressionsgeraden	39
Abb. 16: Weg - Zeit - Geschwindigkeit - Diagramme für gewählten Abschnitt der A5	40
Abb. 17: Geschwindigkeit - Dichte - Diagramme für gewählten Abschnitt der A5	40
Abb. 18: Verkehrsstärke - Dichte - Diagramme für gewählten Abschnitt der A5	40
Abb. 19: Verkehrsstärke - Geschwindigkeit - Diagramme für gewählten Abschnitt der A5	41
Abb. 20: Streckenabschnitt zwischen Ober-Mörlen und Friedberg [7].....	57

Tabellenverzeichnis

Tabelle 1: Kombinationen der Simulationsparameter und Szenariowahl 34
Tabelle 2: Festlegung der Szenario-unabhängigen Simulationsparameter 35

Literaturverzeichnis

- [1] simTD-Konsortium., simTD. [Online] [Zitat vom: 27. Juli 2010.] <http://www.simtd.de>.
- [2] T-Systems., *simTD-Überblick*. Bonn : s.n., 09. Oktober 2009.
- [3] Busch, Univ.-Prof. Dr.-Ing. Fritz., Lehrstuhl für Verkehrstechnik. *Technische Universität München*. [Online] [Zitat vom: 11. Mai 2010.] http://www.vt.bv.tum.de/uploads/misc/poster/TUMVT_Poster_C2X.pdf.
- [4] Gansen, Tobias, et al., *Spezifikationsdokument simTD Funktion F_1.2.3: Erweiterte Navigation Version 0.33*.
- [5] Geistefeldt, Justin, et al., *Spezifikationsdokument simTD Funktion F_1.3.1 Version 1.0*.
- [6] PTV Planung Transport Verkehr AG., *PTV Vision VISSIM©, Version 5.20, Benutzerhandbuch*. Karlsruhe : s.n., November 2009.
- [7] Google., Google Maps. [Online] 2010. [Zitat vom: 13. September 2010.] <http://maps.google.de>.
- [8] Harding, Jochen., Modellierung und mikroskopische Simulation. [Online] [Zitat vom: 29. Juli 2010.] http://deposit.ddb.de/cgi-bin/dokserv?idn=986729477&dok_var=d1&dok_ext=pdf&filename=986729477.pdf.
- [9] Pavlidis, Panos., "Karlsruhe Institute of Technology." [Online] 3. Juli 2007. [Zitat vom: 3. August 2010.] http://www.ipd.uni-karlsruhe.de/~damast/seminar/FAS2007/pdfs/FAS2007_Pavlidis_Fahrerassistenzsysteme_Vortrag.pdf.
- [10] Bungartz, Hans-Joachim, et al., *Modellbildung und Simulation*. Berlin Heidelberg : Springer-Verlag, 2009.
- [11] Reimann, Michael., "Universität Karlsruhe." [Online] 05. Juli 2007. [Zitat vom: 28. August 2010.] http://www.ipd.uni-karlsruhe.de/~damast/seminar/FAS2007/pdfs/FAS2007_Reimann_Simulationsmodelle_Ausarbeitung.pdf.
- [12] Matiello, Pedro, et al., python-graph - Project Hosting on Google Code. [Online] 20. März 2010. [Zitat vom: 28. Juni 2010.] <http://code.google.com/p/python-graph/>.
- [13] Wikimedia Foundation Inc., Wikipedia. *Verkehrsstau*. [Online] [Zitat vom: 28. Juli 2010.] <http://de.wikipedia.org/wiki/Verkehrsstau>.
- [14] PTV Planung Transport Verkehr AG., PTV Vision VISSIM©. [Online] [Zitat vom: 28. Juli 2010.] <http://www.ptv.de/software/verkehrsplanung-verkehrstechnik/software-und-system-solutions/vissim/>.
- [15] Forschungsgesellschaft für Straßen- und Verkehrswesen e.V., *Hinweise zur mikroskopischen Verkehrsflusssimulation - Grundlagen und Anwendungen*. Köln : s.n., 2006.
- [16] Statistisches Bundesamt Deutschland., Verkehrsmittelbestand und Infrastruktur. [Online] 2010. [Zitat vom: 10. September 2010.] <http://www.destatis.de/jetspeed/portal/cms/Sites/destatis/Internet/DE/Content/Statistiken/Verkehr/VerkehrsmittelbestandInfrastruktur/Tabellen/Content75/Verkehrsinfrastruktur,templateId=renderPrint.psml>.

Anhang

I Detaillierte Dokumentation der Implementierung

I.a TrafficAnalysis.py

Attribute:

COMInitialized	<ul style="list-style-type: none"> • Boolean • gibt an, ob die COM-Schnittstelle erfolgreich initialisiert wurde
TravelTimeQueueInitialized	<ul style="list-style-type: none"> • Boolean • gibt an, ob die <i>TravelTimeQueue</i> erfolgreich initialisiert wurde
QueueInitialized	<ul style="list-style-type: none"> • Boolean • gibt an, ob die <i>AvgVelocityQueue</i> erfolgreich initialisiert wurde
DynamicRouterInitialized	<ul style="list-style-type: none"> • Boolean • gibt an, ob die Klasse <i>DynamicRouter</i> erfolgreich initialisiert wurde
COMVissim	<ul style="list-style-type: none"> • COM-VISSIM-Objekt
COMNet	<ul style="list-style-type: none"> • COM-VISSIM-Netz • Enthält alle Strecken und Verbinder
COMVehicles	<ul style="list-style-type: none"> • Liste aller Fahrzeuge des VISSIM-Netzes über COM-Schnittstelle
TravelTimeQueue	<ul style="list-style-type: none"> • Instanz der Klasse <i>AvgTravelTime</i> – für Bestimmung der Reisezeiten
AvgQueue	<ul style="list-style-type: none"> • Instanz der Klasse <i>AvgQueue</i> – zur Bestimmung der durchschnittlichen Geschwindigkeiten und Fahrzeuganzahlen pro Streckenabschnitt
DynamicRouter	<ul style="list-style-type: none"> • Instanz der Klasse <i>DynamicRouter</i> – Zur Anwendung der dynamischen Routenführung

Methoden:

<code>__init__(self)</code>	<ul style="list-style-type: none"> • Konstruktor • initialisiert C2X-Schnittstelle
<code>InitializeCOMValues(self)</code>	Initialisiert die COM-Schnittstelle und alle COM-Variablen
<code>initializeTravelTimeQueue(self)</code>	Initialisiert die <i>TravelTimeQueue</i>
<code>processTimestep(self)</code>	<ul style="list-style-type: none"> • Wird in jedem Zeitschritt von VISSIM über die C2X-Schnittstelle aufgerufen. • Initialisiert im ersten Zeitschritt alle Variablen durch die Aufrufe der Methoden <i>InitializeCOMValues</i>, <i>initializeTravelTimeQueue</i> und <i>initializeDynamicRouter</i> • Aktualisiert in jedem Zeitschritt die <i>TravelTimeQueue</i> und die <i>AvgQueue</i> durch Aufruf der Methoden <i>addTravelTime</i> und <i>deleteAllOldEntries</i> bzw. <i>enqueue</i> und <i>deleteAllOldEntries</i>. Das heißt, es werden immer die aktuellen Fahrzeugdaten hinzugefügt und alle veralteten Daten gelöscht. • In regelmäßigen Abständen wird der aktuelle Zustand von <i>TravelTimeQueue</i> und <i>AvgQueue</i> ausgegeben. • Bevor ebenfalls in regelmäßigen Abständen (default 1 Min.) das dynamische Routing aktiviert wird, gibt es ein Update des Netzgraphen • Das dynamische Routing wird durch Aufruf von <i>DynamicRouter.routeIndividual</i> für jedes einzelne Fahrzeug individuell oder durch Aufruf von <i>DynamicRouter.routeCollective</i> für alle Fahrzeuge im Kollektiv gestartet.

I.b CalcTravelTime.py

Attribute:

TravelTimeQueueInitialized	<ul style="list-style-type: none"> • Boolean • gibt an, ob die <i>TravelTimeQueue</i> erfolgreich initialisiert wurde
TravelTimeQueue	<ul style="list-style-type: none"> • Instanz der Klasse <i>AvgTravelTime</i> – für Bestimmung der Reisezeiten

Methoden:

<code>__init__(self)</code>	<ul style="list-style-type: none"> • Konstruktor • initialisiert C2X-Schnittstelle
<code>initializeTravelTimeQueue(self)</code>	<ul style="list-style-type: none"> • Initialisiert die <i>TravelTimeQueue</i>
<code>processTimestep(self)</code>	<ul style="list-style-type: none"> • Wird in jedem Zeitschritt von VISSIM über die C2X-Schnittstelle aufgerufen. • Initialisiert im ersten Zeitschritt die <i>TravelTimeQueue</i> durch Aufruf der Methode <i>initializeTravelTimeQueue</i> • Aktualisiert in jedem Zeitschritt die <i>TravelTimeQueue</i> durch Aufruf der Methoden <i>addTravelTime</i> und <i>deleteAllOldEntries</i>. Das heißt, es werden immer die aktuellen Fahrzeugdaten hinzugefügt und alle veralteten Daten gelöscht. • In regelmäßigen Abständen wird der aktuelle Zustand von <i>TravelTimeQueue</i> ausgegeben. (default alle 100 Sekunden) • Ebenfalls in regelmäßigen Abständen (alle 10 Min.) wird der <i>NetGraph</i> aktualisiert und gespeichert

I.c DynamicRouter.py

Attribute:

DELAY_THRESHOLD	<ul style="list-style-type: none"> Verlustzeit-Schwellwert für das individuelle Routing in Sekunden (default 60)
TRAF-FIC_JAM_THRESHOLD	<ul style="list-style-type: none"> Grenzwert für die Geschwindigkeit in m/s (default 8,3) Alle Kanten mit geringeren Geschwindigkeiten gelten als gestört
StaticRoutes	<ul style="list-style-type: none"> Dict, bestehend aus Dicts, die alle statischen Routen vom Typ <i>Route</i> in einer Liste enthalten Index basiert auf Start- bzw. in 2. Ebene auf Zielstrecke der Routen
DefaultGraph	<ul style="list-style-type: none"> Netzgraph mit Standardreisezeiten
CurrentGraph	<ul style="list-style-type: none"> Aktueller Netzgraph
AvgVelocityQueue	<ul style="list-style-type: none"> Instanz der Klasse <i>AvgQueue</i> – zur Bestimmung der durchschnittlichen Geschwindigkeiten und Fahrzeuganzahlen pro Streckenabschnitt
RouteTab	<ul style="list-style-type: none"> Dict, bestehend aus Dicts, die jeweils eine statische Route als Typ <i>Route</i> enthalten Index basiert auf Routenentscheidungs- bzw. in 2. Ebene auf der Routennummer der Routen
MyNetGraph	<ul style="list-style-type: none"> Instanz der Klasse <i>NetGraph</i>
debugFile	<ul style="list-style-type: none"> File-Objekt zum Loggen von Routinginformationen

Methoden:

<code>__init__(self)</code>	<ul style="list-style-type: none"> Konstruktor Initialisierung der globalen Variablen
<code>createRouteTab(self)</code>	<ul style="list-style-type: none"> Initialisiert die <i>RouteTab</i>
<code>routeIndividual(self, vehicle)</code>	<ul style="list-style-type: none"> Startet das individuelle dynamische Routing <i>vehicle</i> (c2x.vehicle) – zu routendes Fahrzeug ruft <i>getDelayBiggerThanThreshold</i> für <i>vehicle</i> auf Falls True, wird Routing ausgeführt <ul style="list-style-type: none"> Übernimmt Reisezeit und aktuelle Kante von Rückgabewert aus <i>getDelayBiggerThanThreshold</i> Sucht schnellerste Alternativroute Prüft, ob Route schneller als aktuelle Route Falls ja, Aufruf von <i>getDriverAcceptance</i> Falls <i>getDriverAcceptance</i>, wird aktuelle Route von <i>vehicle</i> auf die neue Route gesetzt
<code>getDelayBiggerThanThreshold(self, vehicle)</code>	<ul style="list-style-type: none"> prüft, ob aktuelle Verlustzeit auf der Route von <i>vehicle</i> (c2x.vehicle) größer als <i>DELAY_THRESHOLD</i> ist gibt 3-Tupel <i>x</i> folgender Form zurück (Boolean, Integer, 2-Tupel) <ul style="list-style-type: none"> dient der Optimierung von <i>routeIndividual</i> <i>x</i> = (<i>x1</i>, <i>x2</i>, <i>x3</i>) <ul style="list-style-type: none"> <i>x1</i> = Prüfergebnis <i>x2</i> = Reisezeit für Rest der aktuellen Route des <i>vehicle</i>, falls <i>x1</i> <i>x2</i> = 0, sonst <i>x3</i> = Aktuelle Kante des <i>vehicle</i>, falls <i>x1</i> <i>x3</i> = None, sonst
<code>routeCollective(self, vehicles)</code>	<ul style="list-style-type: none"> Startet das kollektive Routing <i>vehicles</i> (c2x.vehicles) – zu routendes Fahrzeug ruft <i>getJammedEdges</i> und <i>getJammedRoutes</i> auf, um gestörte Routen zu finden Sucht alle Alternativen für gestörte Routen (nur einmal pro Start-Ziel-Paar) -> in <i>alternativeRoutes</i> einfügen Lege für die Alternativrouten ein dict <i>jammedRouteNum</i> von

	<p>dicts an, die jeweils mit Routenentscheidungen bzw. Routennummern indiziert sind</p> <ul style="list-style-type: none"> • Rufe <i>FindAffectedVehicles</i> mit <i>vehicles</i>, <i>jammedRouteNum</i>, <i>alternativeRoutes</i> auf, um betroffene Fahrzeuge zu ermitteln <ul style="list-style-type: none"> • Betroffene Fahrzeuge werden ggf. umgeleitet
getJammedEdges (self)	<ul style="list-style-type: none"> • Ermittelt alle gestörten Kanten in <i>DefaultGraph</i> • D.h. Kantengewicht in m/s < <i>TRAFFIC_JAM_THRESHOLD</i> • Gibt Liste aus Kanten (2-Tupel) zurück
getJammedRoutes (self, jammedEdges)	<ul style="list-style-type: none"> • Ermittelt alle Routen in <i>StaticRoutes</i>, die Kanten aus <i>jammedEdges</i> enthalten • Gibt Liste aus Routen vom Typ <i>Route</i> zurück
FindAffectedVehicles (self, vehicles, jammedRouteNum, alternatives)	<ul style="list-style-type: none"> • Sucht alle Fahrzeuge in <i>vehicles</i> (<i>c2x.vehicles</i>), die auf eine Route aus <i>jammedRouteNum</i> fahren • Für alle die, wird <i>sendAlternativesToVehicle</i> mit dem betroffenen Fahrzeug <i>vehicle</i>, der aktuellen Route und der Liste von Alternativrouten der gleichen Routenentscheidung aufgerufen
sendAlternativesToVehicle (self, vehicle, currentRoute, alternativeRoutes)	<ul style="list-style-type: none"> • Prüft, welche Alternativrouten aus <i>alternativeRoutes</i> für das Fahrzeug in Frage kommen und setzt Fahrzeug <i>vehicle</i> (<i>c2x.vehicle</i>) auf die schnellste Route • Bestimmt Restreisezeit der aktuellen Route <ul style="list-style-type: none"> ➔ 2 Spezialfälle, wenn <i>vehicle</i> auf Start- oder Zielstrecke der <i>currentRoute</i> ist ➔ 1.Fall (Startstrecke): Restreisezeit = komplette Reisezeit ➔ 2.Fall (Zielstrecke): kein dyn. Routing mehr nötig/möglich • Suche alle schnelleren Routen, als die <i>currentRoute</i> • 2 Möglichkeiten für Fahrerentscheidungssimulation: <ol style="list-style-type: none"> 1. Aufruf von <i>getDriverChoice</i> 2. Aufruf von <i>getDriverAcceptance</i> für schnellste Route • Setze Route von <i>vehicle</i> auf gewählte/akzeptierte Route
getDriverChoice (self, currentRoute, alternativeRoutes)	<ul style="list-style-type: none"> • Simuliert Fahrerentscheidung • Kann zwischen aktueller (<i>currentRoute</i>) und allen Alternativrouten (<i>alternativeRoutes</i>) entscheiden • Gibt die ausgewählte Route zurück • <u>Ist nicht implementiert und wird nicht aufgerufen!!!</u> (s. <i>sendAlternativesToVehicle</i>)
getDriverAcceptance (self, currentTravelTime, betterTravelTime)	<ul style="list-style-type: none"> • Simuliert Akzeptanz des Fahrers bezüglich der vorgeschlagenen Route mit Reisezeit <i>betterTravelTime</i> • Aufruf der Random-Funktion <i>randint</i> mit Intervallgrenzen 0 und 1 • Gibt mit 50% Wahrscheinlichkeit True zurück • Kann auch in Abhängigkeit von <i>currentTravelTime</i> und <i>betterTravelTime</i> implementiert werden

I.d NetGraph.py

Attribute:

INPUT_FILE	Serialisierte txt-Inputdatei des Netzgraphen
VISSIM_INPUT_FILE	VISSIM-Netz-Datei (.inp)
OUTPUT_GRAPHIC_FILE	Grafische Ausgabedatei
OUTPUT_FILE	Serialisierte txt-Zieldatei des Netzgraphen
Graph	StreckenVerbinderGraph
SourceDestTab	Tabelle aller Zielknoten, der von einem Knoten ausgehenden Kanten
VerbinderSourceTab	<ul style="list-style-type: none"> • Tabelle aller Verbinder, die von der gleichen Strecke ausgehen • Index basiert auf dem Namen der VON-Strecke des Verbinders
VerbinderDestTab	<ul style="list-style-type: none"> • Tabelle aller Verbinder, die zur gleichen Strecke führen • Index basiert auf dem Namen der NACH-Strecke des Verbinders
StreckenTab	<ul style="list-style-type: none"> • Tabelle aller Strecken, die einer Kante im Netzgraphen zugeordnet werden • Index basiert auf der VISSIM-Strecke
RepStreckenTab	<ul style="list-style-type: none"> • Gegenstück zu <i>StreckenTab</i> – enthält alle VISSIM-Strecken zu einer Kante • Index basiert auf Label der Kante im Netzgraphen
edgeLabelTab	<ul style="list-style-type: none"> • Tabelle aller Kanten, die zu einem Kantenlabel gehören • Index basiert auf Kantenlabel
outboundEdgesTab	<ul style="list-style-type: none"> • Tabelle aller Kanten, die von einem Knoten ausgehen • Index basiert auf dem Knoten • Dient der Verbesserung von <i>getOutboundEdges</i>
Verhaltenstypen	<ul style="list-style-type: none"> • Liste aller benötigten Verhaltenstypen (als Nummer), die für das dynamische Routing relevant sind • Dient der Reduzierung des Netzes auf die wesentlichen Streckenabschnitte
StaticRoutes	<ul style="list-style-type: none"> • Dict, bestehend aus Dicts, die alle statischen Routen vom Typ <i>Route</i> in einer Liste enthalten • Index basiert auf Start- bzw. in 2. Ebene auf Zielstrecke der Routen
data	<ul style="list-style-type: none"> • Daten der VISSIM-Netz-Datei
existingStaticRoutes	<ul style="list-style-type: none"> • Dict, bestehend aus Dicts, die jeweils eine bereits vorhandenen statische Route als Typ <i>Route</i> enthalten • Index basiert auf Start- bzw. in 2. Ebene auf Zielstrecke der Routen

Methoden:

<code>__init__(self)</code>	<ul style="list-style-type: none"> • Konstruktor
<code>printGraphInPicture(self)</code>	<ul style="list-style-type: none"> • Wandelt den <i>Graph</i> in eine dot-Datei, die in <i>OUTPUT_GRAPHIC_FILE</i> gespeichert wird
<code>saveGraph(self)</code>	<ul style="list-style-type: none"> • Serialisiert <i>Graph</i> und alle relevanten globalen Variablen in <i>OUTPUT_FILE</i>
<code>createEdgeLabelTab(self)</code>	<ul style="list-style-type: none"> • Ergänzende Funktion für <i>readfile</i> zur Erzeugung von <i>edgeLabelTab</i>
<code>readFile(self, filename)</code>	<ul style="list-style-type: none"> • Liest serialisierten <i>Graph</i> und alle relevanten globalen Variablen aus Datei mit Dateinamen <i>filename</i> oder per default <i>INPUT_FILE</i>
<code>parseVerbinderFromFile(self, filename)</code>	<ul style="list-style-type: none"> • Liest Datei mit Dateinamen <i>filename</i> ein • erzeugt einen Netzgraphen mit den Strecken repräsentiert als Knoten und Verbinder als Kanten • verwendet nur Strecken bzw. Verbinder, deren Verhaltenstypen einem aus der Liste <i>Verhaltenstypen</i> entsprechen
<code>find_node_in_other_graph(self, graph, name)</code>	<ul style="list-style-type: none"> • Sucht einen Knoten mit Namen <i>name</i> im gegebenen Graphen <i>graph</i> und gibt diesen zurück

getOutboundEdges (<i>self</i> , <i>node</i> , optimized)	<ul style="list-style-type: none"> Gibt alle ausgehenden Kanten des Knoten <i>node</i> in <i>Graph</i> als Liste zurück Parameter <i>optimized</i> gibt an, ob die laufzeitoptimierte Methode verwendet werden soll. Diese sollte nur bei einem danach unveränderten <i>Graph</i> verwendet werden, weil es sonst zu Inkonsistenzen kommt. Über <i>createOutboundEdgesTab</i> können diese jedoch behoben werden
createOutboundEdgesTab (<i>self</i>)	<ul style="list-style-type: none"> Erstellt neue <i>outboundEdgesTab</i> basierend auf aktuellem <i>Graph</i>
getOutboundEdgesInOtherGraph (<i>self</i> , <i>graph</i> , <i>node</i>)	<ul style="list-style-type: none"> Gibt alle ausgehenden Kanten des Knoten <i>node</i> im gegebenen Graphen <i>Graph</i> als Liste zurück
getIncomingEdges (<i>self</i> , <i>node</i>)	<ul style="list-style-type: none"> Gibt alle eingehenden Kanten des Knoten <i>node</i> in <i>Graph</i> zurück
getIncomingEdgesInOtherGraph (<i>self</i> , <i>graph</i> , <i>node</i>)	<ul style="list-style-type: none"> Gibt alle eingehenden Kanten des Knoten <i>node</i> im gegebenen Graphen <i>Graph</i> zurück
swapNodesEdges (<i>self</i>)	<ul style="list-style-type: none"> Überschreibt <i>Graph</i> mit neuem Graphen, der aus dem alten durch Tauschen von Knoten und Kanten hervorgeht Zusammenfassung aller Knoten zu Kanten und aller ausgehenden Kanten zu einem Knoten
getConnectedComponent (<i>self</i> , <i>graph</i>)	<ul style="list-style-type: none"> Gibt dict zurück, das für jeden Knoten die Nummer der zugehörigen Zusammenhangskomponente enthält Basiert auf <i>Package pygraph :: Package algorithms :: Module accessibility :: connected_components(graph)</i> Angepasst für gerichtete Graphen
colorComponent (<i>self</i> , <i>graph</i> , <i>visited</i> , <i>color</i> , <i>node</i>)	<ul style="list-style-type: none"> Recursive Hilfsfunktion für <i>getConnectedComponent</i>
mergeTracks (<i>self</i>)	<ul style="list-style-type: none"> Sucht alle Knoten, die nicht (nur) eine eingehende und eine ausgehende Kante besitzen → sind Ausgangspunkte für mögliche Zusammenfassungen von Kanten Von hieraus wird die Methode <i>reduceGraph</i> mit den jeweiligen Ausgangsknoten als Parameter aufgerufen
reduceGraph (<i>self</i> , <i>node</i> , <i>mergedEdge</i>)	<ul style="list-style-type: none"> Rekursive Funktion (Tiefensuche) Fasst beginnend bei einem Ausgangsknoten alle Kanten zusammen, die nur eine eingehende und eine ausgehende Kante besitzen
dummyGraph (<i>self</i>)	<ul style="list-style-type: none"> Erzeugt einen Testgraphen
setEdgeWeight (<i>self</i> , <i>edgeID</i> , <i>weight</i>)	<ul style="list-style-type: none"> Sucht in <i>edgeLabelTab</i> nach der Kante mit dem Label <i>edgeID</i> und setzt deren Kantengewicht auf <i>weight</i>
getRepresentativeEdgeOfLinkID (<i>self</i> , <i>linkID</i>)	<ul style="list-style-type: none"> Ermittelt zu einer Strecke (wenn vorhanden) eine Liste der Kanten, die diese Strecke im Netzgraphen repräsentieren
getRepresentativeEdgeLabelOfLinkID (<i>self</i> , <i>linkID</i>)	<ul style="list-style-type: none"> Ermittelt zu einer Strecke (wenn vorhanden) den Namen der Kante, die diese Strecke im Netzgraphen repräsentiert
shortestPath (<i>self</i> , <i>startNode</i> =None)	<ul style="list-style-type: none"> Sucht kürzesten Pfad vom Startknoten <i>startNode</i> zu allen anderen Knoten
saveGraphWithWeights (<i>self</i> , <i>filename</i>)	<ul style="list-style-type: none"> Setzt alle Kantengewicht == 1 auf <i>sys.maxint</i> Speichert <i>Graph</i> unter <i>filename</i> oder per default unter <ul style="list-style-type: none"> "DynamischesRoutingDefaultGraph.txt" durch Aufruf von <i>saveGraph</i>
loadGraphWithWeights (<i>self</i> , <i>filename</i>)	<ul style="list-style-type: none"> Lädt <i>Graph</i> aus Datei <i>filename</i> oder per default aus <ul style="list-style-type: none"> "DynamischesRoutingDefaultGraph.txt" durch Aufruf von <i>readFile</i>
findAllRoutesFromTo (<i>self</i> , <i>startTrack</i> , <i>destTrack</i> , <i>routeDecisionNum</i> , <i>startIn-</i>	<ul style="list-style-type: none"> Sucht alle Routen von <i>startTrack</i> nach <i>destTrack</i> Generiert neue <i>Route</i>-Objekte

dex)	<ul style="list-style-type: none"> • Fügt Routen zu <i>StaticRoutes</i> hinzu • <i>routeDecisionNum</i> – Nummer der zugehörigen Routenentscheidung • <i>startIndex</i> – Nummer der ersten gefundenen Route
search (self, startTrackGlob, startNodeLoc, destTrack, destNode, nodeList, routeDecisionNum, routeNumStartIndex, limit)	<ul style="list-style-type: none"> • Rekursive Hilfsfunktion für <i>findAllRoutesFromTo</i> • Basiert auf Tiefensuche • <i>startTrackGlob</i> – Startstrecke der zu findenden Routen • <i>startNodeLoc</i> – aktueller Knoten in der Rekursion • <i>destTrack</i> – Zielstrecke der zu findenden Routen • <i>destNode</i> - Zielknoten • <i>nodeList</i> – Liste aller Knoten, die auf der Route liegen • <i>routeDecisionNum</i> – Nummer der Routenentscheidung • <i>routeNumStartIndex</i> – Nummer der ersten gefundenen Route • <i>limit</i> – Rekursionslimit in Sekunden (Sucht nur Strecken der Länge <i>limit</i>)
loadAllStaticRoutes (self, filenameWithoutExtension)	<ul style="list-style-type: none"> • Lädt serialisierte Routendatei mit Namen <i><filenameWithoutExtension>.sr</i> in <i>StaticRoutes</i> und <i>existingStaticRoutes</i> • Zusätzlich wird <i>data</i> geladen, damit ein schnelles Übertragen von statischen Routen auf ein VISSIM-Netz erfolgen kann
parseAllStaticRoutes (self, filename)	<ul style="list-style-type: none"> • Lädt Datei <i>filename</i> in <i>data</i> ein • Parst alle statischen routen aus der Datei • Speichert gefundene Routen in <i>existingStaticRoutes</i>
findAllStaticRoutes (self)	<ul style="list-style-type: none"> • Ruft <i>parseAllStaticRoutes</i> auf • Sucht alle Routen von jeder Start- zu jeder Zielstrecke in <i>existingStaticRoutes</i> • Ruft jeweils <i>findAllRoutesFromTo</i> auf • Speichert auch alle „existing“ Routen in <i>StaticRoutes</i>
saveAllStaticRoutes (self, filenameWithoutExtension)	<ul style="list-style-type: none"> • Speichert alle <i>StaticRoutes</i> serialisiert unter <i><filenameWithoutExtension>.sr</i> ab • Fügt alle <i>StaticRoutes</i> zur VISSIM-Netz-Datei hinzu • Ruft dazu jeweils <i>insertRouteInOutputData</i> auf
insertRouteInOutputData (self, route)	<ul style="list-style-type: none"> • Fügt <i>route</i> zu <i>data</i> hinzu • Verwendet VISSIM-Format für Routen
getNextNode (self, track)	<ul style="list-style-type: none"> • Gibt zu einer VISSIM-Strecke <i>track</i> den darauffolgenden Knoten im <i>Graph</i> an

I.e AvgTravelTime.py

Attribute:

Graph	<ul style="list-style-type: none"> Graphdarstellung als <i>Netgraph</i>
Duration	<ul style="list-style-type: none"> Vorhaltezeit der ermittelten Reisezeiten in Sekunden – Standard 600 (10 Minuten)
TravelTimeQueue	<ul style="list-style-type: none"> Tabelle aller Reisezeitwarteschlangen für alle Kanten Index basiert auf dem Label der betreffenden Kante Reisezeitwarteschlangen enthalten Einträge vom Typ <i>TimeEntry</i>
LinkList	<ul style="list-style-type: none"> Liste aller auch im Graph als Kanten vorkommenden Strecken
TimeSum	<ul style="list-style-type: none"> Tabelle enthält für jede Kante die entsprechende Summe aller Reisezeiten, die im relevanten Zeitraum (<i>Duration</i>) liegen
VehLinkTimeList	<ul style="list-style-type: none"> Tabelle enthält für jedes C2X-Fahrzeug den zuletzt befahrenen Streckenabschnitt (entspricht einer Kante im Graph) und den Zeitpunkt des ersten Befahrens dieses Streckenabschnitts, sodass sich damit die Reisezeiten ermitteln lassen
MINIMUM_ENTRIES	<ul style="list-style-type: none"> Anzahl der Einträge, die in den Reisezeitwarteschlangen mindestens enthalten bleiben, auch wenn die Aktualität der Daten nicht mehr ausreichend ist

Methoden:

<code>__init__(self)</code>	<ul style="list-style-type: none"> Konstruktor, initialisiert die Reisezeitwarteschlangen in <i>TraveltimeQueue</i> und legt die Liste <i>LinkList</i> an
<code>addTravelTime(self, vehID, linkID, timestamp)</code>	<ul style="list-style-type: none"> Fügt einen neuen <i>TimeEntry</i> in die Reisezeitwarteschlange für die repräsentierende Kante von <i>linkID</i> ein, wenn das C2X-Fahrzeug <i>vehID</i> einen Streckenabschnitt verlässt
<code>checkAllGoneVehicles(self, timestamp, vehicles)</code>	<ul style="list-style-type: none"> Durchläuft <i>VehLinkTimeList</i> Sucht nach allen Fahrzeugen, die nicht mehr in <i>vehicles</i> enthalten sind Ruft <i>addTravelTime</i> für alle gefundenen Fahrzeuge auf
<code>deleteAllOldEntries(self, timestamp)</code>	<ul style="list-style-type: none"> Löscht alle Einträge aus den <i>TimeTravelQueues</i>, die älter als <i>Duration</i> sind, falls noch mehr als <i>MINIMUM_ENTRIES</i> vorhanden sind Gleichzeitig werden die entsprechenden Reisezeiten aus <i>TimeSum</i> entfernt
<code>updateNetGraph(self)</code>	<ul style="list-style-type: none"> Überträgt die aktuellen Durchschnittsreisezeiten als Kantengewichte auf den Netzgraphen
<code>getAvgTravelTime(self, linkID)</code>	<ul style="list-style-type: none"> Ermittelt die Durchschnittsreisezeit für den gesamten repräsentierenden Streckenabschnitt einer gegebenen Strecke <i>linkID</i>
<code>printAllAvgTravelTimes(self)</code>	<ul style="list-style-type: none"> Gibt alle Durchschnittsreisezeiten der repräsentierenden Streckenabschnitte auf der Konsole aus
<code>setTimeInterval(self, duration, timestamp = -1)</code>	<ul style="list-style-type: none"> Setzt die globale Variable <i>Duration</i> Wenn ein korrekter Zeitstempel übergeben wird, wird im Anschluss auch <i>deleteAllOldEntries</i> ausgeführt, sodass alle Reisezeitwarteschlangen und Attribute konsistent gehalten werden
<code>test()</code>	<ul style="list-style-type: none"> Statische Methode zum Test der Klasse

I.f AvgQueue.py

Attribute:

Graph	<ul style="list-style-type: none"> Graphdarstellung als <i>Netgraph</i>
Duration	<ul style="list-style-type: none"> Vorhaltezeit der ermittelten Geschwindigkeiten bzw. Fahrzeuganzahlen in Sekunden – Standard 600 (10 Minuten)
EntryQueue	<ul style="list-style-type: none"> Tabelle aller Warteschlangen für alle Kanten mit Einträgen vom Typ <i>Entry</i> Index basiert auf dem Label der betreffenden Kanten
VelocitySum	<ul style="list-style-type: none"> Tabelle enthält für jede Kante die entsprechende Summe aller Geschwindigkeiten, die im relevanten Zeitraum (<i>Duration</i>) liegen
LatestTimestamp	<ul style="list-style-type: none"> Tabelle enthält für jede Kante den Zeitstempel des neusten Eintrages in der Warteschlange
TimestampCount	<ul style="list-style-type: none"> Tabelle enthält für jede Kante die Anzahl der Einträge in der Warteschlange
MINIMUM_ENTRIES	<ul style="list-style-type: none"> Anzahl der Einträge, die in den Warteschlangen mindestens enthalten bleiben, auch wenn die Aktualität der Daten nicht mehr ausreichend ist

Methoden:

<code>__init__(self)</code>	<ul style="list-style-type: none"> Konstruktor, initialisiert die Warteschlangen in <i>EntryQueue</i> und initialisiert die Attribute mit Standardwerten
<code>getAvgVelocityOfLink(self, linkID)</code>	<ul style="list-style-type: none"> Ermittelt die Durchschnittsgeschwindigkeit für die repräsentierende Kante einer gegebenen Strecke <i>linkID</i>
<code>getAvgVelocityOfEdge(self, edgeLabel)</code>	<ul style="list-style-type: none"> Ermittelt die Durchschnittsgeschwindigkeit für die Kante mit Label <i>edgeLabel</i>
<code>printAllSignificantValues(self)</code>	<ul style="list-style-type: none"> Gibt für alle Kanten die Durchschnittsgeschwindigkeiten aus, falls mindestens ein Wert vorhanden ist
<code>enqueue(self, timestamp, velocity, linkID)</code>	<ul style="list-style-type: none"> Fügt einen neuen <i>Entry</i> in die Warteschlange für die repräsentierende Kante von <i>linkID</i> mit Geschwindigkeit <i>velocity</i> ein
<code>deleteAllOldEntries(self, timestamp)</code>	<ul style="list-style-type: none"> Löscht alle Einträge aus den <i>EntryQueues</i>, die älter als <i>Duration</i> sind, falls noch mehr als <i>MINIMUM_ENTRIES</i> vorhanden sind Gleichzeitig werden die entsprechenden Geschwindigkeiten bzw. Fahrzeuganzahlen aus <i>VelocitySum</i> und <i>TimestampCount</i> entfernt
<code>setTimeInterval(self, duration, timestamp = -1)</code>	<ul style="list-style-type: none"> Setzt die globale Variable <i>Duration</i> Wenn ein korrekter Zeitstempel übergeben wird, wird im Anschluss auch <i>deleteAllOldEntries</i> ausgeführt, sodass alle Warteschlangen und Attribute konsistent gehalten werden
<code>getAvgVehicleCountOfLink(self, linkID)</code>	<ul style="list-style-type: none"> Ermittelt die Fahrzeuganzahl für die repräsentierende Kante einer gegebenen Strecke <i>linkID</i>
<code>getAvgVehicleCountOfEdge(self, edgeLabel)</code>	<ul style="list-style-type: none"> Ermittelt die Fahrzeuganzahl für die Kante mit Label <i>edgeLabel</i>

I.g TimeEntry.py

Attribute:

Timestamp	• Aktueller Zeitstempel
Time	• Reisezeit
VehID	• Fahrzeug-ID

Methoden:

<code>__init__(self)</code>	<ul style="list-style-type: none"> • Konstruktor • Setzen der globalen Variablen
<code>getTimestamp(self)</code>	• Gibt den Zeitstempel zurück
<code>getTime(self)</code>	• Gibt die Reisezeit zurück
<code>getVehID(self)</code>	• Gibt die Fahrzeug-ID zurück

I.h Entry.py

Attribute:

Timestamp	• Aktueller Zeitstempel
Velocity	• Fahrzeuggeschwindigkeit
Link	• Aktuelle Kante

Methoden:

<code>__init__(self)</code>	<ul style="list-style-type: none"> • Konstruktor • Setzen der globalen Variablen
<code>getTimestamp(self)</code>	• Gibt den Zeitstempel zurück
<code>getVelocity(self)</code>	• Gibt die Geschwindigkeit zurück
<code>getLink(self)</code>	• Gibt die aktuellen Kante zurück

I.i Route.py

Attribute:

Graph	<ul style="list-style-type: none"> • Graphdarstellung als <i>Netgraph</i>
RouteList	<ul style="list-style-type: none"> • Liste von Knoten und Kanten, über die die Route verläuft • Reihenfolge vom <i>Start</i> zum <i>Dest</i>
EdgeList	<ul style="list-style-type: none"> • Liste der Kanten über die die Route verläuft • Reihenfolge vom <i>Start</i> zum <i>Dest</i>
Start	<ul style="list-style-type: none"> • Startstrecke der Route
Dest	<ul style="list-style-type: none"> • Zielstrecke der Route
routeDecisionNum	<ul style="list-style-type: none"> • Routenentscheidungsnummer
routeNum	<ul style="list-style-type: none"> • Routennummer

Methoden:

__init__(self)	<ul style="list-style-type: none"> • Konstruktor • Setzen der globalen Variablen
__eq__(self, other)	<ul style="list-style-type: none"> • Komparator für Routen • Vergleicht die Attribute <ul style="list-style-type: none"> ◦ <i>Graph</i> ◦ <i>Start</i> ◦ <i>Dest</i> ◦ <i>EdgeList</i> • Sollte nicht im größeren Maßstab verwendet werden • Als Vergleich können auch <i>routeDecisionNum</i> und <i>routeNum</i> verwendet werden
setStartTrack(self, trackID)	<ul style="list-style-type: none"> • Setzt die Startstrecke der Route auf <i>trackID</i>
getStartTrack(self)	<ul style="list-style-type: none"> • Liefert die Startstrecke der Route
setDestTrack(self, trackID)	<ul style="list-style-type: none"> • Setzt die Zielstrecke der Route auf <i>trackID</i>
getDestTrack(self)	<ul style="list-style-type: none"> • Liefert die Zielstrecke der Route
setRouteDecisionNum(self, routeDecisionNum)	<ul style="list-style-type: none"> • Setzt die Routenentscheidungsnummer der Route auf <i>routeDecisionNum</i>
getRouteDecisionNum(self)	<ul style="list-style-type: none"> • Liefert die Routenentscheidungsnummer der Route
setRouteNum(self, routeNum)	<ul style="list-style-type: none"> • Setzt die Routennummer der Route auf <i>routeNum</i>
getRouteNum(self)	<ul style="list-style-type: none"> • Liefert die Routennummer der Route
addNodeToRoute(self, node)	<ul style="list-style-type: none"> • Fügt Knoten <i>node</i> hinten in die <i>RouteList</i> ein
addEdgeToRoute(self, edge)	<ul style="list-style-type: none"> • Fügt Kante <i>edge</i> hinten in <i>RouteList</i> und <i>EdgeList</i> ein
getRoute(self)	<ul style="list-style-type: none"> • Liefert <i>RouteList</i>
getEdges(self)	<ul style="list-style-type: none"> • Liefert <i>EdgeList</i>
getViaTracks(self, trackTab)	<ul style="list-style-type: none"> • Liefert alle Zwischenstrecken der Route auf Basis der <i>EdgeList</i> • Für die Umwandlung in Strecken wird eine Repräsentationstabelle benötigt, die für jede repräsentierende Kante alle Strecken enthält

II Zusätzliche Abbildungen



Abb. 20: Streckenabschnitt zwischen Ober-Mörlen und Friedberg [7]

III CD mit Programmquelltexten

Auf der beigelegten CD befinden sich die Python-Quelltexte sämtlicher im Anhang I dokumentierten Klassen für das dynamische Routing. Weiterhin wurden diese Ausarbeitung, eine Simulationsumgebung für VISSIM und die nötigen Python-Bibliotheken für die C2X-Schnittstelle von VISSIM auf der CD gespeichert.