THM | CAMPUS GIESSEN | MNI
TECHNISCHE HOCHSCHULE MITTELHESSEN
Mathematik, Naturwissenschaften und Informatik

Bachelorarbeit

Dynamische Geschwindigkeitsempfehlung auf Basis von „vehicle-to-infrastructure“ Kommunikation unter Einfluss von kollektiver Intelligenz zwischen Fahrzeugen

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt dem

Fachbereich Mathematik, Naturwissenschaften und Informatik

der Technischen Hochschule Mittelhessen

Samir Faycal Tahar M’Sallem

27. Februar 2023

Referent: Prof. Dr.-Ing. Seyed Eghbal Ghobadi

Korreferent: Moritz Schauer, M.Sc.

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Hanau, den 27. Februar 2023 Samir Faycal Tahar M'Sallem

Zusammenfassung

Viele Metropolregionen sind von einem erhöhten Verkehrsaufkommen und dem damit verbundenen Ausstoß von CO_2-Emissionen betroffen. Eine vorausschauende Fahrweise und weniger Beschleunigung können eine Einsparung von Kraftstoff und eine Reduzierung der Emissionen bewirken. Diese Bachelorarbeit geht der Frage nach, inwiefern ein Informationsaustausch zwischen Fahrzeugen und der Infrastruktur genutzt werden kann, um eine optimale Geschwindigkeitsempfehlung für ein Fahrzeug zu bestimmen. Als Grundlage hierfür dienen vorausgegangene Ansätze zur Umsetzung einer Kommunikation zwischen Fahrzeugen und Ampelsystemen. Potentielle Grenzen und Schwächen bisheriger Ansätze werden aufgezeigt und schlussendlich in einem verbesserten Verfahren unter Verwendung von kollektiver Intelligenz umgesetzt. Zu diesem Zweck wird das System um die Kommunikation zwischen Fahrzeugen erweitert, sodass eine ganzheitliche Lösung in Bezug auf das Verkehrsaufkommen erzielt werden kann. Um die Leitfrage der Bachelorarbeit zu beantworten, erfolgt die Umsetzung innerhalb einer Simulationslösung, die die Ergebnisse durch die Aufzeichnung der Geschwindigkeiten simulierter Fahrzeuge hervorbringt. Die Ergebnisse zeigen, dass das umgesetzte Verfahren eine Verbesserung in Bezug auf den Verkehrsfluss der Fahrzeuge bewirkt. Deutlich wird dies, da die Fahrzeuge im Sinne einer Organisation in Richtung der Ampel fahren und somit verzichtbare Geschwindigkeitsanpassungen vermieden werden können. Zusätzlich werden nachfolgende Verkehrsteilnehmer nicht ineffektiv von anderen Verkehrsteilnehmern beeinflusst. Da dieses Verfahren lediglich im Rahmen einer Simulation erprobt wurde, lässt die Forschungsarbeit Raum offen, um zukünftige Forschungen entsprechend der Ergebnisse zu orientieren. Denkbar wäre die Erprobung des Verfahrens in der Realität unter Existenz einer adäquaten Infrastruktur zur Kommunikation von Fahrzeugen.

Inhaltsverzeichnis

1 Einleitung

Die Energiewende und der Umstieg auf die Elektromobilität sind ein Ansatz um Emissionen innerhalb von Metropolregionen zu verringern. Neben Veränderungen in der Technologie sind neue Denkweisen und eine Offenheit für Neues ein maßgeblicher Erfolgsfaktor für dieses Problem. Kraftstoffersparnisse und Einsparungen von Emissionen können durch eine bessere und vorausschauendere Fahrweise hervorgerufen werden.

Insbesondere in Bezug auf das autonome Fahren ist eine vorausschauende Fahrweise erstrebenswert, da Fahrzeuge qualitativ hochwertige Daten benötigen, um in die Lage versetzt zu werden korrekt zu agieren. Um eine zuverlässige Informationsquelle für Fahrzeuge zu ermöglichen, werden Fahrzeuge mit ihrer Umwelt vernetzt um somit auf Daten ihrer Umgebung und des Internets zugreifen zu können. Die Kommunikation zwischen Fahrzeugen einerseits, sowie Fahrzeugen und der Infrastruktur andererseits bilden die grundsätzlichen Pfeiler des vernetzten Fahrens.

Mit Hilfe der Kommunikation zwischen einem Fahrzeug und der Infrastruktur kann eine Geschwindigkeitsempfehlung berechnet werden, durch die ein Fahrzeug befähigt wird die grüne Welle zu befahren und vorausschauender zu agieren.

1.1 Motivation

Die Grundlage der Forschung dieser Arbeit bildet die Studie der City of Ottawa [1], in der ein solches System zum Liefern einer Geschwindigkeitsempfehlung umgesetzt wurde. Die Entwicklung basiert auf einem Informationsaustausch zwischen einem Ampelsystem und einem Fahrzeug. Neben der Implementierung wurde dieses System im Anschluss an echten Fahrern von Kraftfahrzeugen getestet. Die daraus gesammelten Daten wurden hinsichtlich der Einsparung von Emissionen und Kraftstoff analysiert, weshalb die Studie als passende Grundlage für diese Forschungsarbeit gewählt wurde. Ausgehend der Ergebnisse und vorausgegangenen Entscheidungen hinsichtlich der Implementierung in der Studie, kann ein Verbesserungspotential entdeckt werden, das im Nachgang tiefer behandelt wird.

1.2 Zielsetzung

Ziel dieser Forschungsarbeit ist es, eine verbesserte Implementierung auf Basis der Studie anzubringen, indem eine Anreicherung mittels kollektiver Intelligenz unter Verwendung von vehicle-to-vehicle (V2V) Kommunikation erfolgt. Zu diesem Zweck sollen theoretische Hintergründe geeigneter Methoden zum Austausch von Informationen angebracht werden und ebenfalls in die Implementierung einfließen. Weiterhin soll ein Konzept der technischen Umsetzung aufgestellt und begründet werden. Die Realisierung soll durch Erklärung der Verfahren dargelegt werden. Zu erwarten ist eine Aufstellung der Ergebnisse der Simulation unter Vergleich der Resultate der Studie. Diese Bachelorarbeit soll in einem Fazit unter Heranziehen des gesammelten Wissens aus der Studie und der Implementierung münden und Ansätze für zukünftige Forschungen bereithalten.

1.3 Struktur der Arbeit

Da die Grundlage dieser Arbeit die Studie der City of Ottawa bildet, wird diese zunächst als Forschungsstand herangezogen. Nachdem auf die Studie eingegangen wurde, sollen vorausgegangene Implementierungen, die die spätere Realisierung unterstützen, erklärt werden. Im Anschluss werden Verbesserungsvorschläge aus der Studie gezogen, die den weiteren Verlauf dieser Arbeit bestimmen.

Aus diesem Anlass werden anschließend erweiterte Verfahren in Bezug auf die Kommunikation von Fahrzeugen erklärt, da diese für die nachfolgende Implementierung notwendig sind. Dem zugrunde liegen theoretische Konzepte, unter die insbesondere die kollektive Intelligenz fällt, die in die Konzeption und Umsetzung einfließen sollen. Durch Vorstellung dieser theoretischen Hintergründe soll ein vertieftes fachliches Ziel unter Verwendung der erlangten theoretischen Hintergründe formuliert werden.

Nachdem dies geschehen ist, wird ein technisches Konzept zur Umsetzung der Implementierung aufgestellt. Daran angegliedert folgt die Realisierung des Konzepts durch zuvor ausgewählte Technologien.

Zuletzt wird abschließend eine Zusammenfassung der Ergebnisse aufgestellt, die in drei Abschnitte geteilt wird. Zunächst werden die Resultate der Simulation analysiert und vor dem Hintergrund der Umsetzung evaluiert und bewertet. Davon ausgehend wird im Anschluss ein Fazit in Hinblick auf die Forschungsfrage und die eingangs erwähnte Studie gezogen. Schlussendlich soll ein Ausblick auf zukünftige Ansätze in diesem Themenbereich gegeben werden, der an die Ergebnisse dieser Bachelorarbeit geknüpft ist.

2 Forschungsstand

Im Folgenden werden die Studie der City of Ottawa, sowie die vorausgegangene Entwicklung eines vergleichbaren Dienstes beschrieben. Im Anschluss sollen Schwächen aufgedeckt und in einer Reihe von Verbesserungsvorschlägen notiert werden.

2.1 Studie der City of Ottawa

Der Technische Bericht in [1] enthält die Beschreibung und Evaluierung des Projekts „EcoDrive II“. Inhaltlich befasst sich dieser mit einer Studie zu Vorteilen der Nutzung von Green Light Optimal Speed Advisory (GLOSA) innerhalb der Stadt Ottawa in Kanada. Die GLOSA ist eine optimale Geschwindigkeitsempfehlung, die an die Grünphasen einer Ampel angepasst ist, sodass eine Fahrt ohne Stillstand möglich ist [1]. Generiert werden diese Daten mittels einer Kommunikation zwischen Fahrzeugen und der Infrastruktur, welche bezeichnet wird als vehicle-to-infrastructure (V2I) Kommunikation. Ein Feldversuch im Rahmen des Projekts sammelte Daten über einen Zeitraum von acht Wochen, in dem Fahrer mit einer Android App ausgestattet wurden, die relevante Informationen zum Befahren der grünen Welle lieferte. Die App visualisiert die momentane Signalfarbe einer Ampel vor einem Fahrzeug basierend auf den Global Positioning System (GPS) Koordinaten des Mobiltelefons. Weiterhin wird die GLOSA angezeigt, die von den Fahrern befolgt werden soll. Die Evaluierung erfolgt, indem Fahrzeugparameter des Kraftstoffverbrauchs und der Geschwindigkeit über die On-Board-Diagnose (OBD) Schnittstelle ausgelesen werden. Dies ist eine einheitliche Schnittstelle, zum Auslesen relevanter Parameter aus den Steuergeräten eines Fahrzeugs [1]. Die Ziele der GLOSA sind Abweichungen in der Fahrzeuggeschwindigkeit zu minimieren und eine Reduzierung des Kraftstoffverbrauchs, sowie das Einsparen von Emissionen hervorzurufen. Um dies zu ermöglich wird über das Mobiltelefon auf eine Schnittstelle, bezeichnet als Application Programming Interface (API) über das Internet zugegriffen. Dieser Dienst wird von der Firma Traffic Technologies Services (TTS) als information-as-a-service (IaaS) betrieben und liefert Topologien, Schaltzyklen und Position einer vorausliegenden Ampel auf Basis der Position des Mobiltelefons [1].

Die Kommunikation wird nachfolgend durch die Visualisierung der sogenannten High-Level Architektur beschrieben:

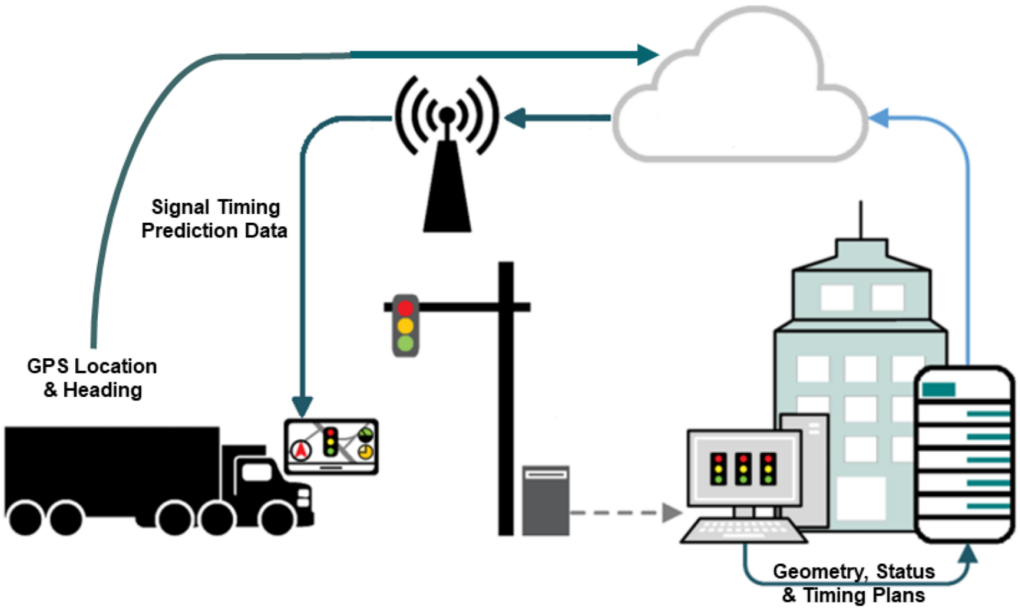


Abbildung 2.1: High-Level Architektur des GLOSA Systemaufbaus [1, S. 21]

Zu erkennen ist die V2I Kommunikation. Aus technischer Sicht wurden für die Realisierung folgende Daten des Mobiltelefons herangezogen:

- Geographische Breite („latitude“)
- Geographische Länge („longitude“)
- Ausrichtung
- Genauigkeit
- GPS Geschwindigkeit

Mittels dieser Daten wird in [1, S. 42] der Prozess zur Datenverarbeitung aufgezeigt. Zunächst werden die Positionsdaten an die API von TTS übermittelt. Sofern die Position mit einer passenden Kreuzungsanfahrt übereinstimmt werden die Daten der Kreuzung zurückgeliefert. Aus den Signalphasen und einer Datei, in der die Höchstgeschwindigkeiten aller Straßen bekannt sind, wird eine Geschwindigkeitsempfehlung berechnet [1, S. 42]. Im Weiteren werden Regelungen beschrieben, die ebenfalls in die Berechnung der GLOSA einfließen:

- Sofern die Geschwindigkeit $95\,\mathrm{km\,h^{-1}}$ überschreitet, wird keine Geschwindigkeitsempfelung angezeigt, da der Fahrer mit hoher Warscheinlichkeit auf einer Autobahn unterwegs ist.

- Zu Testzwecken innerhalb der Studie ist es möglich eine Geschwindigkeitsempfehlung auszusprechen, die $10\,\mathrm{km\,h^{-1}}$ über der erlaubten Höchstgeschwindigkeit liegt. Dies hat den Hintergrund, dass die Empfehlungen nicht zu restrikiv sein sollen.

- Da typischerweise andere Verkehrsteilnehmer vor der Ampel warten, wird bei der Berechnung ein Zeitpuffer berücksichtigt, das von der Zeit der Grünphase subtrahiert wird. Dazu wird festgelegt, dass die Verzögerung bei Fahrt nach links fünf Sekunden beträgt, anderenfalls werden zehn Sekunden abgezogen [1, S. 43].

Nachdem sieben Fahrzeuge mit diesem System ausgestattet wurden begann die Testphase. Zunächst fuhren die Fahrzeuge sechs Wochen ohne Geschwindigkeitsempfehlung, um einen Richtwert für die anschließende Analyse zu erhalten. Im Anschluss wurde den Fahrern zwei Wochen die Geschwindigkeitsempfehlung angezeigt.

Die Daten wurden wöchentlich erhoben, um eine Vergleichbarkeit zu schaffen. Dies hat den Hintergrund, da die Sorge bestand, dass die Fahrer durch Nutzung der Klimaanlage an wärmeren Tagen die Ergebnisse verfälschen könnten [1].

Mit Hinblick auf die Ergebnisse ist zu erkennen, dass eine Einsparung des Kraftstoffs und eine Reduzierung der CO_2-Emissionen hervorgerufen werden konnte. Bezogen auf alle Testfahrzeuge konnte eine Einsparung um 2,5% erzielt werden [1, S. 6]. Da dies sowohl alle Fahrzeuge, als auch alle Straßentypen beinhaltet konnte ferner untersucht werden, dass Fahrten die lediglich auf Straßen stattfanden, die eine GLOSA liefern können eine durchschnittliche Einsparung von 7% ermöglicht wurde.

Die Forscher kamen zum Ergebnis, dass es relevanter sei die verschiedenen Straßenarten, Wartezeiten eines Fahrzeugs und mögliche Straßen auf denen keine GLOSA verfügbar ist zu betrachten, da diese Routen dem realen Fahrverhalten eines Fahrzeugs am nächsten kommen [1]. Weiterhin können die Ergebnisse hinsichlich der Motivation der Fahrer und dem Verkehrsaufkommen variieren. So erzielten junge motivierte Fahrer maximal 14,4% Einsparung des Kraftstoffverbrauchs, wohingegen ältere Fahrer bekanntgaben, dass sie die Empfehlungen wenig befolgten. Weiterhin konnte Fahren bei geringem Verkehrsaufkommen überdies den Verbrauch um 18% reduzieren. Das Fahren auf Durchfahrtsstraßen erzielte 6,8% Ersparnis in Bezug auf den Kraftstoffverbrauch, wogegen Nebenstraßen mit weniger Verkehrsaufkommen bis zu 16,5% ermöglichten [1, S. 9].

Insbesondere aus dem Ergebnis des motivierten Fahrers ist erkennbar, dass die Einsparung durch die tatsächliche Umsetzung des Fahrers und der damit verbundenen Umstellung des Fahrverhaltens zusammenhängt [1, S. 9].

Dies wird ebenfalls in der Analyse der Ergebnisse deutlich, da der Erfolg an drei wesentlichen Faktoren festgemacht wird. Zunächst muss der Fahrer eine Akzeptanz gegenüber der Technologie und der Änderung des Fahrstils mitbringen.

Weiterhin ist das vorhandene Verkehrsaufkommen entscheidend. Zuletzt ist die Straßenart, wie zuvor an den Ergebnissen aufgezeigt, ein relevanter Faktor [1, S. 118].

Die Evaluierung der Verhaltensweisen und die Funktion der GLOSA offenbart Ergebnisse, die im Nachgang von der Carleton University gedeutet werden.

Beginnend wird grundsätzlich das Verfahren bewertet. Die Verwendung des Internetzugangs über das Mobiltelefon wird als kosteneffizient bezeichnet und die Visualisierung der Daten als wenig anfällig um Ablenkungen zu verursachen. [1, S. 129]. Darüber hinaus wird als positiv erachtet, dass die Fahrer in den meisten Fällen in der Lage waren, die Informationen über aufkommende Ampelphasen zu nutzen. Dem gegenüber steht die Tatsache, dass die voreingestellten Zeitpuffer - repräsentativ für den vorausfahrenden Verkehr - unter Existenz eines hohen Verkehrsaufkommens zu einer Nichteinhaltung der Empfehlung führten [1, S. 129]. Dieser Punkt wird dadurch belegt, dass Fahrten bei einem hohen Verkehrsaufkommen zu weniger Einsparungen führten. In Bezug auf die Möglichkeit eine GLOSA auszusprechen die das Tempolimit überschreitet ist erkennbar, dass dies die Fahrer tendenziell verlangsamt, da sie nicht bereit sind die Geschwindigkeit zu erhöhen, wodurch keine Verbesserung erzielt werden kann. Weiterhin können Fahrer per se die Geschwindigkeit nicht erhöhen, sofern ein vorausfahrendes Fahrzeug langsamer fährt. Schließlich wird der Fahrer die Geschwindigkeit ebenfalls nicht anpassen, sofern die Empfehlung deutlich von der momentanen Fahrgeschwindigkeit abweicht [1, S. 129].

Schlussendlich werden zukünftige Forschungsansätze angebracht. Unter anderem könnten Mechanismen der V2V Kommunikation genutzt werden, um den Verkehrsfluss innerhalb einer Grünphase zu erhöhen [1, S. 78]. Weiterhin geben autonom fahrende Fahrzeuge gegenüber den herausgearbeiteten Unterschieden hinsichtlich junger und älterer Fahrer die Sicherheit, dass diese die Geschwindigkeitsempfehlung einhalten würden. Aus diesem Grund könnte die Geschwindigkeitsempfehlung für das autonome Fahren einen größeren Nutzen haben [1, S. 76].

2.2 Vorausgegangene Implementierungen

Als technische Grundlage für diese Forschungsarbeit dient außerdem die Umsetzung equivalenter technischer Komponenten in [2], wie sie in der Studie der City of Ottawa aufgezeigt wurden. Darunter fällt eine mobile Applikation, sowie eine vergleichbare API. Theoretische Hintergründe bilden die technischen Anforderungen der Studie, sowie das Produkt „Ampelinformation Online“ [3] der Audi AG. Die Daten der Infrastruktur entspringen aus der gleichen Bezugsquelle wie in der Studie. Diese kommen von der Firma TTS und beinhalten alle Topologien und Schaltzyklen für die Stadt Ingolstadt. Konzipiert wurde eine native App für IOS und Android Endgeräte, die im Stande ist die Signalphasen und die GLOSA wie in der Studie der City of Ottawa zu visualisieren. Darüber hinaus wurde eine API entwickelt, die basierend auf den Standortinformationen des Mobiltelefons die Ampelschaltungen und die GLOSA der voraus befindlichen Ampel zurückliefert [2, S. 2]. Innerhalb dieses Berichts wurden Verfahren und Umsetzungen beschrieben, die auf Überschneidungen mit den Ansätzen aus dem „EcoDrive II“ Projekt treffen. Dennoch können einige Unterschiede festgehalten werden, die die zuvor beschriebenen Grenzen der Umsetzbarkeit der GLOSA teilweise beseitigen. Die API kann exemplarisch mittels einer Hypertext Transfer Protocol (HTTP) Anfrage aufgerufen werden. Eine exemplarische Anfrage mit den benötigten Parametern ist in Anhang A abgebilet. Der Algorithmus zur Berechnung der GLOSA ist als Flussdiagramm in Anhang C abgebildet. Die Implementierung wird in Anhang D ersichtlich. Zunächst orientiert sich dieser an der Momentangeschwindigkeit des Fahrzeugs [2, S. 9]. Daraus resultiert eine Geschwindigkeitsempfehlung, die niemals eine höhere Abweichung als $15\,\mathrm{km\,h^{-1}}$ gegenüber der momentanen Geschwindigkeit produziert. Darüber hinaus wird eine Beschleunigungskraft festgelegt, die einem üblichen Fahrzeug entspricht. Sofern die erforderliche Beschleunigung für eine Geschwindigkeitsänderung diese Kraft übersteigt, ist die Empfehlung ungültig, da sie nicht umsetzbar wäre [2, S. 10]. Schlussendlich werden Versuche unternommen die Informationen des gesamten Verkehrsgeschehens besser einfließen zu lassen. Zu diesem Zweck wird über eine API von Google Verkehrsinformationen und Tempolimits abgefragt. Die Tempolimits dienen dem Zweck eine dynamische Höchstgeschwindigkeit zu definieren, die auf der Straße auf der das Fahrzeug befindlich ist gefahren werden darf. Die Verkehrsinformationen sollen, ähnlich wie eine Ankunftszeit bei einem Navigationssystem, eine Auskunft darüber geben, ob die GLOSA für den vorliegenden Fall angebracht ist. Sofern ein Fahrzeug aufgrund der Verkehrsinformationen eine längere Fahrtdauer zur Ampel hat als Informationen über die Schaltzyklen existieren, so verliert die GLOSA an Aussagekraft [2, S. 8].

2.3 Verbesserungsvorschläge

Die Studie greift Grenzen der Umsetzbarkeit der Geschwindigkeitsempfehlung, sowie mögliche Ansätze zu Verbesserungen auf. In Hinblick auf die Zielsetzung dieser Bachelorarbeit gilt es diese Punkte in Form eines verbesserten Konzepts zu beseitigen. Ein besonderer Schwerpunkt wird dabei auf die Berechenbarkeit des Verkehrs gelegt, da diese innerhalb der Studie lediglich in schwacher Ausprägung erfolgte. Das vordefinierte Zeitpuffer ermöglichte unter einem geringen Verkehrsaufkommen eine funktionierende Geschwindigkeitsempfehlung. Die Grenzen dieser statischen Definition wurden bereits innerhalb der Studie bei einem hohen Verkehrsaufkommen ersichtlich. Zu diesem Zweck wurde in [2] der Versuch gestartet, durch Nutzung von Verkehrsinformationen die Geschwindigkeitsempfehlung positiv anzureichern. Da dieses Verfahren dennoch keine vollständige und zuverlässige Lösung darstellt, gilt es weitere Mechanismen herauszuarbeiten. Darüber hinaus beschränkt sich die Geschwindigkeitsempfehlung der Studie zu jedem Zeitpunkt lediglich auf das Fahrzeug, welches unmittelbar in Kommunikation mit der Infrastruktur steht. Für den Fall, dass der nachfolgende Verkehr aufgrund dieses Fahrzeugs ausgebremst wird, weil es der Geschwindigkeitsempfehlung folgt, existiert keine Möglichkeit dem nachfolgenden Verkehr zu erlauben der grünen Welle ebenfalls zu folgen. Dieses Problem differiert von der Kernthese der Studie, da diese die individuelle Kraftstoffersparniss eines Fahrzeugs beleuchtet. Die Konsequenzen für das Kollektiv - hier der Menge an Fahrzeugen auf der Straße - bleiben unberücksichtigt.

In [4] und [5] werden weitere Implementierungen einer GLOSA durch V2I Kommunikation diskutiert. Dort wird ebenfalls das Hauptaugenmerk auf die Einsparung von Kraftstoff eines Fahrzeugs gelegt. Insbesondere vor dem Hintergrund der Möglichkeiten, die sich durch die V2I Kommunikation eröffnen, sollten weitere vehicle-to-everything (V2X) Ansätze[1] in Betracht gezogen werden und mit in die Implementierung eingebracht werden.

Zuletzt kann in Hinblick auf [6] festgehalten werden, dass die aufgezeigten Grenzen der GLOSA Empfehlungen durch die Studie in [1] allgemeingültig sind. Eckhoff, Halmos und German zeigen auf, dass bei einem geringen Verkehrsaufkommen dem GLOSA System eine hohe Performanz und Zuverlässigkeit zukommt. Sofern mehr Verkehr vorhanden ist verschlechtert sich dies drastisch. Aus ihrer Sicht führt dies dazu, dass Fahrzeuge ohne Informationen über die Ampel einen höheren CO_2-Ausstoß verursachen und alle Verkehrsteilnehmer eine längere Wartezeit haben [6].

1 V2X bezeichnet die grundsätzliche Möglichkeit in der das Fahrzeug mit der Umwelt kommunizieren kann

3 Forschungsthema

Nachdem die Grundlagen bisheriger Arbeiten im Bereich des vernetzten Fahrens zwischen Fahrzeugen und Ampelsystemen erläutert wurden, sollen nun geeignete Verfahren aufgegriffen werden, die im Nachgang implementiert werden.

3.1 Vehicle-to-vehicle Kommunikation

Die Koordination und Kooperation im Straßenverkehr nach Kraft [7] bildet eine wesentliche Grundlage um eine Vielzahl an Szenarien innerhalb des Verkehrsgeschehens zu entschärfen [7, S. 3]. Diese Kooperation bedingt eine Kommunikation von Fahrzeugen. Zu diesem Zweck wird die V2V Kommunikation herangezogen um eine Kooperation von Verkehrsteilnehmern zu ermöglichen. Die V2V Kommunikation beschreibt dabei den Austausch von Daten zwischen Fahrzeugen die sich in unmittelbarer Nähe voneinander befinden [8]. Die V2V Kommunikation ermöglicht das omnidirektionale Aussenden von Nachrichten in die Umgebung des Fahrzeugs [8]. Eine Möglichkeit zum Aussenden von Nachrichten wird dabei durch die Dedicated Short-range Communication (DSRC) ermöglicht. Diese stellt in Kombination mit dem GPS eine kosteneffiziente Lösung in Blick auf die Kommunikationsformen dar [8]. Ferner wird geschlussfolgert, dass die V2V Kommunikation eine entscheidende Technik in Blick auf das autonome Fahren ist [8, S. 45].

3.2 Floating Car Data

Die Floating Car Data (FCD) beschreibt das Sammeln von Zustandsinformationen einer Menge von Fahrzeugen. Die Grundlage bildet das GPS, da aus diesem die wesentliche Datengrundlage entspringt. Fahrzeuge senden ihre Positionsinformationen, Ausrichtung und Geschwindigkeit über das Internet an einen dezentralen Dienst, der diese Daten in einem breit aufgestellten Spektrum an Anwendungsgebieten einsetzen kann [9].

Hauptsächlich wird FCD verwendet, um Verkehrsstatistiken anzufertigen oder eine Verkehrskontrolle in Echtzeit vorzunehmen. Weiterhin können die Informationen auch genutzt werden um Navigationssysteme mit Informationen über mögliche Verkehrsmeldungen zu informieren. In letzterem Fall kann ein solcher Dienst als Intelligent Transport System (ITS) bezeichnet werden. Zu diesem Zweck liegt die Grundidee der FCD in einer anonymen Übermittlung von Daten, sodass keine Datenschutzverletzung zustande kommen kann. Resultierend aus den Einsatzgebieten der FCD können Fahrzeuge als sich bewegende Sensoren oder präziser als Software-Agenten im Sinne der Sammlung von Daten bezeichnet werden [9]. Es existieren in [9] Ansätze um traditionelle FCD durch Computer Vision anzureichern, um so einen Mehrwert in Blick auf die Bestimmung des Verkehrsaufkommens zu erhalten. Aufgrund der Komplexität dieser Fusionierung ist lediglich die traditionelle FCD ein Teil dieser Bachelorarbeit.

3.3 Kollektive Intelligenz

Die kollektive Intelligenz, im Weiteren als Schwarmintelligenz bezeichnet, ist nach Hamann eine

> „konzeptionelle Sichtweise, die auf bestimmte ideale Grundprinzipien fokussiert ist.“ [10, S. 14].

Die Schwarmintelligenz schreibt einem Kollektiv, bestehend aus mehreren Individuen, feste Verhaltensmuster zu, nach denen sich die Individuen innerhalb des Schwarms richten. Ein interessanter Punkt beinhaltet, dass die Individuen keinen Blick auf die vollständige Situation werfen und dennoch eine Kooperation zwischen allen Teilhabern erfolgt [10, S. 19]. Eine Bewegung als Kollektiv stellt stets eine geordnete Bewegung dar [10, S. 36].

Hamann führt die Schwarmintelligenz auf Entdeckungen aus der Tierwelt zurück. Unter anderem verortet er diese bei Ameisen und Bienen. Weiterhin wird der Begriff der Arbeitsteilung und Arbeitsverteilung angesprochen, da diese notwendige Bestandteile des Zusammenlebens der Kolonien sind.

Hamann begründet, dass die Ameisen auf bestimmte Reize reagieren und entsprechend dieser Reize ihr Verhalten an die notwendigen Handlungen anpassen. Über dieses Verfahren sind die Tiere in der Lage ihr Leben innerhalb des Schwarms zu sichern [10, S. 20].

Im weiteren Verlauf wirft er die Frage auf, ob die kollektive Intelligenz auch auf Menschen übertragbar sei. Nach seiner Auffassung haben Menschen ein starkes Sozialverhalten, allerdings könne man sie nicht mit Tieren vergleichen, da das Verhalten von Tieren als Reaktion auf eine bestimmte Handlung erfolge, wobei Menschen eine Kognition besitzen. Dennoch begründet er, dass in Situation in denen Menschen ausschließlich reaktiv reagieren können ein Schwarmverhalten gemäß der Schwarmintelligenz hervorgebracht wird. Das angebrachte Beispiel beschreibt eine Paniksituation in der Menschen in Richtung eines Notausgangs strömen um vor einem Feuer zu fliehen. Dabei wählen die Menschen bewusst den schnellsten Weg zum Ausgang, unter Berücksichtigung anderer Menschen vor ihnen. Die daraus folgende Struktur und Ordnung zur Problemlösung entspricht den Grundprinzipien der Schwarmintelligenz [10, S. 31].

Um nun einen Nutzen aus der Schwarmintelligenz für diese Arbeit zu ziehen, ist die nachfolgende Aussage von Hamann als Grundlage gegeben:

> „Kurzgefasst kann man sagen, dass die Biologie bestehende Schwarmverhalten verstehen möchte, während man im Ingenieurwesen Schwarmverhalten erzeugen möchte.“ [10, S. 93].

Das Schwarmverhalten kann aus der Perspektive von Ingenieuren durch zwei Strategien erzeugt werden:

- automatische Entwurfsstrategie

  Die automatische Entwurfsstrategie beschreibt die Entwicklung von Verhaltensmustern unter Heranziehung von maschinellem Lernen oder Verfahren zur Optimierung, wie zum Beispiel das Reinforcement Learning.

- verhaltensbasierte Entwurfsstrategie

  Die verhaltensbasierte Entwurfsstrategie beschreibt die Entwicklung von Verhaltensmustern unter Aufstellung von manuell gewählten Parametern und Bedingungen [10, S. 96].

Inwiefern diese Impulse konkret in die Entwicklung einfließen, wird in der anschließenden Zielsetzung festgelegt.

3.4 Thematische Überleitung

Für diese Bachelorarbeit wird unterstellt, dass auf ein Verkehrsszenario in dem Fahrzeuge existieren die Schwarmintelligenz angewendet werden kann. Da die Fahrer nach den Regeln der Verkehrsordnung agieren, sind sie in ihrem Handeln an die bestehenden Konventionen gebunden. Daraus ist es möglich zu schließen, dass die Fahrer auf eine festgelegte Aktion eine Reaktion zeigen werden, die aufgrund der vorherrschenden Gesetze des Autofahrens berechenbar ist. Unter dieser Annahme ist es naheliegend, dass die Fahrer auf Signale ihrer Umwelt vertrauen werden, sofern diese für den Fahrer vertretbar sind. Die Signale sollen von anderen Fahrzeugen des Kollektivs ausgesendet werden, weshalb die V2V Kommunikation im Weiteren verwendet werden soll. Das übergeordnete Ziel der Präsentation einer Lösung um den Verkehrsfluss zu optimieren bleibt dabei unverändert.

Ein sogenannter Shared Information Space (SIS) dient als Gedächtnis für das Kollektiv, sodass auf vorherige Impulse zurückgegriffen werden kann und mögliche Aktionen in Abhängigkeit dieser erfolgen können. Ein SIS beschreibt nach Methoden des Informationsmanagements einen geteilten virtuellen Informationsraum, in dem Individuen eine Informationsbasis schaffen, auf deren Basis das Wissen unter allen Individuen aufgeteilt werden kann [11]. Repräsentativ für die hier vorliegende Aufgabe soll dieser SIS als Plattform dienen um Kommunikation im Kontext der Fahrzeuge zu ermöglichen.

Das Ziel der Verbesserung des Verkehrsflusses unter Existenz der V2I Kommunikation, soll im Folgenden durch die Erkenntnisse über die kollektive Intelligenz und der V2V Kommunikation verbessert werden. Die FCD soll hierfür als Datenbasis fungieren.

In einem technischen Bericht in [12] wird eine neuartige Systemarchitektur beschrieben, die als hybride Architektur bestehend aus V2I und V2V Kommunikation bezeichnet werden kann. Der Anreiz ist es ein ITS noch besser mit Daten zu versorgen. Konkret existieren dafür sogenannte Super-Fahrzeuge, die sowohl mit der Infrastruktur, als auch mit anderen Fahrzeugen kommunizieren können. Die Daten die diese Fahrzeuge erhalten werden aggregiert und an den intelligenten Dienst gesendet [12]. Dieser Ansatz einer hybriden Kommunikation soll als Grundlage für die nachfolgende Implementierung genutzt werden, da im vorliegenden Fall die Daten der Infrastruktur, unter Berücksichtigung der Verkehrssituation, genutzt werden sollen.

Folglich gilt es im Anschluss eine geeignete Software zu wählen, um eine Simulation zu erstellen, die die Problematik des Verkehrsflusses simuliert.

4 Konzept

4.1 Simulation

Um aus dem Forschungsthema nun ein realisierbares Konzept zu bestimmen, gilt es passende Technologien und Software auszuwählen. Im Mittelpunkt der Entwicklung steht deshalb die Simulationssoftware Simulation of Urban Mobility (SUMO), die es ermöglicht Verkehrsszenarien abzubilden. SUMO ist eine frei verfügbare, hoch skalierbare Simulationssoftware, die vom „Institute of Transportation Systems“ entwickelt wird [13]. Die Rolle von SUMO ist dabei die Simulation eines Verkehrsgeschehens durch Erzeugung und Simulierung von Aktoren. Im Sinne der Forschungsfrage sind Fahrzeuge und ein Ampelsystem die wesentlichen Aktoren. Durch Heranziehen des Traffic Control Interface (TraCI) kann das Szenario gezielt kontrolliert und mit Logik ergänzt werden. TraCI dockt über eine TCP-Schnittstelle an die laufende Simulation an. Die Simulation fungiert dabei als Server, wobei mehrere Clients mittels der TraCI Bibliothek auf die Simulation Einfluss nehmen können [13]. Ein wichtiger Aspekt ist die Fähigkeit Variablen und Parameter aus der laufenden Simulation auslesen und verändern zu können. Dies wird im Fortgang mit Hilfe der Programmiersprache Python erfolgen unter Verwendung von TraCI erfolgen. Die Entscheidung die Implementierung in Python umzusetzen ist damit zu begründen, dass die Simulationssoftware selbst in C++ geschrieben ist und eine Bibliothek für Python bereitstellt. Weiterhin bietet Python die Möglichkeit performanten und umfangreichen Code effizient abzubilden.

Bevor TraCI grundsätzlich zum Einsatz kommen kann, muss ein Szenario modelliert werden. Da die Forschungsfrage neben V2V Kommunikation auch die V2I Kommunikation beinhaltet, soll das Wissen über die Schaltzyklen einer ausgewählten Ampel aus der API, die in Kapitel 2.2 beschrieben wurde, mit in die Simulation einfließen. Der davon ausgehende Wunsch einer Echtzeit-Simulation setzt klare Voraussetzungen hinsichtlich der Modellierung des Szenarios. Einerseits bedingt dies, dass das abgebildete Szenario einem realen Verkehrsszenario gleicht. Weiterhin muss dieses durch den Content-Provider in Bezug auf die Datenverfügbarkeit abgedeckt sein. Auch wird der Aspekt der Echtzeitfähigkeit einen großen Stellenwert einnehmen, da die Echtzeit-Simulation ohne synchronisierte Zeitwerte nicht umsetzbar wäre.

Der Content-Provider stellt Informationen über Ampelsysteme in Ingolstadt zur Verfügung. Innerhalb dieser Stadt wird eine Ampelkreuzung ausgewählt, die besonders gut für die Simulation geeignet ist.

Dabei wird nach den folgenden Kriterien entschieden:

1. Verfügbarkeit

   Die Zeitpunkte, zu denen die Informationen über die Ampelschaltungen verfügbar sind, sind entscheidend, da die Ampel mitunter bei geringem Verkehrsaufkommen oder zu späten Uhrzeiten ausgeschaltet wird.

2. Zuverlässigkeit

   Der Content-Provider liefert zu jeder Ampelphase eine Zuverlässigkeit in Prozent (engl. „confidence level“), die angibt mit welcher Wahrscheinlichkeit die Phase zum Zeitpunkt eintreten wird. Im Kontext von Echtzeit-Simulationen bietet eine hohe Zuverlässigkeit eine bessere Robustheit für die gesamte Simulation.

3. Komplexität

   Die Komplexität des Ampelsystems ist nicht allein bezogen auf die Performanz der Simulation mit Bedacht zu wählen, sondern auch betreffend auf die Forschungsaufgabe abzuwägen. Für ein erstes Konzept genügt ein Ampelsystem mit mittlerer Komplexität, wobei mehr als zwei Anfahrtsrichtungen wünschenswert sind.

Mittels dieser Aufstellung von Kriterien kann schließlich eine passende Kreuzung für die Umsetzung ausgewählt werden:

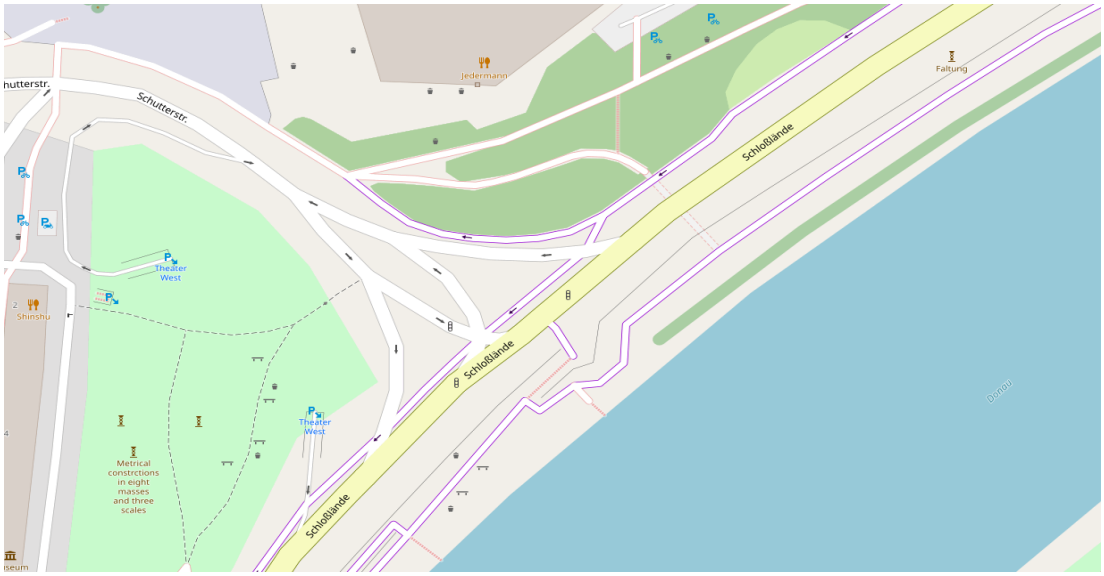


Abbildung 4.1: Ausgewählte Ampelkreuzung für das Simulationsszenario
© OpenStreetMap

Die in Abbildung 4.1 aufgezeigte Ampelkreuzung besteht aus drei Anfahrtswegen (engl. „approaches“), erkennbar an den Ampelsymbolen. Jede Anfahrt enthält mindestens einen Signalkopf (engl. „signal head“), der von der Ampelanlage kontrolliert wird. Im vorliegenden Fall existiert für jede Anfahrt genau ein Signalkopf, was zur Folge hat, dass jede Fahrtrichtung, die von der Anfahrt aus getätigt werden kann, stets der selben Ampelschaltung folgt.

Um diese Ampelkreuzung in der Simulation abbilden zu können, wird im Folgenden das Python Skript zugehörig zum „OSM Web Wizard“ herangezogen. Dabei handelt sich es sich um eine Web-Anwendung, die über die frei verfügbaren Kartendaten von OpenStreetMap ein analoges Szenario existierender Topologien in der Simulation erzeugen kann. Durch Angabe von Positionsdaten kann so ein genaues Abbild der Ampelkreuzung aus Abbildung 4.1 erzeugt werden. Dies ist insofern wichtig, da die Positionsdaten der Fahrzeuge innerhalb des Szenarios mit denen der Realität übereinstimmen müssen. Indem ein genaues Abbild der Realität entnommen wird, werden die Positionsdaten der Kreuzung, sowie der Anfahrten, entsprechend der Realität modelliert.

Das Skript erzeugt eine lauffähige Simulation, die bereits einige Fahrzeuge und Ampelsysteme an der korrekten Stelle enthält. Die generierten Konfigurationsdateien beschreiben das Szenario hinsichtlich des Aufbaus der Anfahrtswege, der Definition des Ampelsystems, sowie der Strukturen und Gebäude in der Umgebung. Da die Konfiguration der Schaltzyklen der Ampeln, sowie die Routen der Fahrzeuge willkürlich gewählt wurden, müssen diese in der Realisierung zunächst grundlegend angepasst werden.

4.2 Softwarepakete

Im Anschluss an die Modellierung des Szenarios gilt es Softwarepakete zu definieren, die im Anschluss implementiert werden sollen. Softwarepakete beschreiben im Sinne der Programmiersprache Python einzelne Module, die hinsichtlich ihres Kontexts in logisch abgetrennte Segmente aufgeteilt werden. Dies fördert sowohl die Struktur, als auch die Nachvollziehbarkeit der Implementierung. Im weiteren Verlauf sollen die Module dann auf notwendige Aspekte anderer Module zugreifen können, sodass die Logik entsprechend des im Folgenden erwähnten Konzepts realisiert werden kann.

Zunächst soll ein Paket definiert werden, welches die Aufgabe hat die Simulation auszuführen und zu beenden. Dies erfolgt über die vorher genannte TraCI Bibliothek. Als nächstes wird ein Modul angelegt, welches es ermöglicht Daten aus der Simulation auszugeben. Dies umfasst im engeren Sinne nicht nur die Kommandozeile, sondern auch eine Bibliothek, die es ermöglicht Graphen und Diagramme basierend auf den extrahierten Daten zu erstellen. Dieser Punkt wird besonders für die Evaluierung und Auswertung der Ergebnisse an Relevanz gewinnen. Hinsichtlich des logischen Ablaufs der Simulation müssen die Ampeldaten aus der Realität in die Simulation übernommen werden. Um die Funktionalität zu ermöglich, die Daten aus der API des V2I Service zu übernehmen, ist es notwendig ein Paket zu definieren, welches mittels HTTP Aufruf die Daten der API empfangen und prozessieren kann. Als Grundlage hierfür dient der exemplarische Aufruf der API in Anhang A, sowie eine exemplarische Antwort in Anhang B. Weiterhin werden diese Daten dann in einem seperaten Modul für die Simulation vorbereitet und übernommen. TraCI bildet folglich eine Vermittlungsebene zwischen der Simulation und den Echtzeitdaten aus diesem Backend-Dienst, da nur so die Echtzeitdaten in die Simulation übertragen werden können. Durch die beschriebenen Pakete können bereits die grundlegenden Funktionen abgedeckt werden, um die Ampeln und Fahrzeuge entsprechend der Daten aus der Realität zu beeinflussen. Da das Ziel dieser Arbeit in der Erweiterung dieses Ansatzes um V2V Kommunikation innerhalb eines Kollektivs liegt, wird ein weiteres Modul entworfen, welches diese Kommunikation ermöglicht. Dazu sollen Funktionen definiert werden die es ermöglichen ein Fahrzeugnetzwerk aufzubauen, um Sender und Adressaten ausfindig zu machen. Darüber hinaus muss eine Datenstruktur gemäß des SIS existieren, in dem Signale und Nachrichten versendet und gespeichert werden können. Schließlich muss eine einheitliche Regelung hinsichtlich der zu kommunizierenden Signale existieren. Im Folgenden sollen diese zu definierenden Signale genauer erklärt werden.

4.3 Kommunikationsstrategie

Neben dem aufgebauten Netzwerk, in dem potentielle Sender und Empfänger von Signalen bestimmt werden, ist der Kommunikationskanal entscheidend. Für die Simulation wird vereinfacht eine Datenstruktur angelegt, welche es ermöglicht Nachrichten abzulegen und auszulesen. Dabei kann jeder Empfänger einer Nachricht nur die für die eigene Identität bestimmten Nachrichten auslesen. SUMO stellt die Möglichkeit zur Verfügung Objekte in das Geschehen einzuzeichnen. Durch die TraCI Bibliothek können Polylinien gezeichnet werden, die den Kommunikationsweg zwischen Fahrzeugen oder Fahrzeugen und der Ampel visualisieren. Dieser Schritt ist erforderlich, da für die Implementierung auf eine zusätzliche Netzwerksimulation verzichtet wird. Dies ist damit zu begründen, dass TraCI die Kontrolle an die Netzwerksimulation abgeben müsste, sofern eine Kommunikation zwischen Knoten stattfinden würde. Knoten repräsentieren im vorliegen Fall die Fahrzeuge und die Ampel. Die Simulation würde also zum Zwecke der Visualisierung von Kommunikationswegen pausiert werden, um die Kommunikation nachvollziehbar zu gestalten. Allerdings hätte dies schwerwiegende Folgen auf die Echtzeitfähigkeit der Simulation, da die gewonnenen Informationen der Ampelschaltzyklen fortlaufend nicht mehr mit denen der Realität übereinstimmen würden. Aus diesem Grund wird in der Umsetzung auf eine zusätzliche Netzwerkkomponente verzichtet, sodass die Kommunikation über einfache Datenstrukturen erfolgt.

Eine Netzwerkbildung gemäß der V2V Kommunikation kommt aber dennoch zustande, indem Fahrzeuge und ihre Umgebung durch die Simulation analysiert werden, sodass potentielle Sender und Adressaten von Nachrichten bestimmt werden können. Dieser Analyse liegt die FCD zugrunde, da die Simulation die Zustandsinformationen der Fahrzeuge überprüft und entsprechende Handlungen einleitet.

Dementsprechend bilden Fahrzeuge die Hauptakteure der Simulation. Diese lassen sich in zwei Kategorien unterteilen: Traditionelle Fahrzeuge bewegen sich gesteuert durch die Simulation über verschiedene Routen innerhalb des Szenarios. Weiterhin existieren Super-Fahrzeuge, die in der Lage sind mit der Infrastruktur zu kommunizieren und die GLOSA zu erhalten. Die Idee zur Umsetzung dieser Super-Fahrzeuge ist an die konzeptionelle Umsetzung aus [12] angelehnt, da die Super-Fahrzeuge sowohl V2I Kommunikation, als auch V2V Kommunikation durchführen.

Die GLOSA wird auf Basis der Position und Fahrgeschwindigkeit eines Super-Fahrzeugs berechnet. Dies lässt wie zuvor bemängelt den davor befindlichen Verkehr außen vor. Um in der Lage zu sein die GLOSA positiv anreichern zu können, werden andere Fahrzeuge befähigt Nachrichten an die Super-Fahrzeuge zu senden. Im Sinne der Erkenntnisse aus dem Forschungsthema kommt also eine vergleichbare hybride Architektur wie in [12] zum Einsatz.

Jede Nachricht enthält den Zeitstempel der Simulation zu dem diese gesendet wurde. Weiterhin ist in ihr der Sender und Empfänger der Nachricht hinterlegt. Die Nachricht wird beschrieben durch eines der nachfolgenden Signale:

- Der Hinweis auf eine rote Ampel: RED

  Das Signal wird von Fahrzeugen verwendet, die vor einer roten Ampel warten. Über das Kommunikationsnetzwerk werden alle nachfolgenden Super-Fahrzeuge alamiert, sofern sie ebenfalls auf die Ampel zufahren und nicht bereits an der Ampel warten.

- Die verbleibende Zeit bis zur Grün-Phase: TTG

  Das Signal dient als Rückantwort zum ersten Signal. Ausgesendet wird es von Super-Fahrzeugen, da diese von dem ersten Signal profitieren und somit im Sinne der kollektiven Intelligenz die wartenden Fahrzeuge ebenfalls mit Informationen versorgen. Aus der Überschrift lässt sich damit schließen, dass die Fahrzeuge informiert werden, wann die Ampel in die grüne Phase wechselt. Vorstellbar ist, dass dieses Signal als Implikation des Bereitmachens genutzt werden kann.

- Der Wunsch, sich der grünen Welle anzuschließen: $MOVE$

  Das Signal wird von Fahrzeugen verwendet, die sich hinter einem Super-Fahrzeug in Richtung der Ampel bewegen. Durch das Signal soll impliziert werden, dass die Fahrzeuge über die grüne Ampel fahren möchten und nicht durch das Super-Fahrzeug ausgebremst werden wollen. Hintergrund ist die zugrundeliegende Geschwindigkeitsempfehlung aus der GLOSA, welche sich aus einer Mindestgeschwindigkeit und Maximalgeschwindigkeit zusammensetzt. Die daraus resultierende Geschwindigkeit die das Fahrzeug fährt, orientiert sich an diesen Geschwindigkeitsgrenzen. Das daraus entstehende Problem könnte eine Fahrgeschwindigkeit sein, die deutlich unter der zugelassenen Höchstgeschwindigkeit auf der Straße liegt. Für das Super-Fahrzeug reicht diese Geschwindigkeit aus, um über die Kreuzung zu fahren, für den nachfolgenden Verkehr könnte es durch diesen Umstand zu einer Verlangsamung kommen. Aus diesem Grund versucht das Super-Fahrzeug bei Erhalt dieses Signals die Geschwindigkeit zu erhöhen, sofern die Maximalgeschwindigkeit der GLOSA noch nicht erreicht wurde.

- Der Wunsch das Vorfahrtsrecht zu erhalten: $TURN$

  Ein weiteres interessantes Verhalten von Fahrern ist das Abtreten des Vorfahrtrechts an andere Verkehrsteilnehmer. Das Signal $TURN$ überträgt dies auf ein spezielles Szenario an der Abbiegespur der Kreuzung.

Wenn ein Fahrzeug nach links abbiegen möchte, muss es dem geradeausfahrenden Gegenverkehr Vorfahrt gewähren. Sofern ein hohes Verkehrsaufkommen vorhanden ist, kann dies zu langen Wartezeiten an der Haltelinie führen. Um das Verkehrsaufkommen dahingehend zu kontrollieren, kann das wartende Fahrzeug dieses Signal an ein Super-Fahrzeug im Gegenverkehr senden. Wenn das Super-Fahrzeug sicherstellen kann, dass es selbst die grüne Ampel passieren kann, bevor diese auf rot wechselt, reduziert es die Geschwindigkeit um dem wartenden Fahrzeug Vorrang zu gewähren.

Die Signale sind in einer Enumeration mittels Code festgehalten und werden gebündelt mit einem Datensatz versendet. Dieser Datensatz enthält in Abhängigkeit von dem gegebenen Signal die Positionsdaten des Senders oder die Restzeit der Grünphase. Wie bereits in Kapitel 3.2 beschrieben, sollen die Fahrzeuge als Software-Agenten agieren und durch ihren Beitrag zum Informationsnetzwerk einen Mehrwert im Sinne der Forschungsfrage erzielen.

Die Super-Fahrzeuge haben einen Wissensvorsprung, da sie über Informationen der zukünftigen Ampelschaltungen verfügen. Dies ist gewünscht, da dieser Umstand in der Realität ebenfalls vorzufinden ist. Fahrzeuge - beispielsweise unmittelbar vor einem Verkehrsunfall - haben die Fähigkeit das Verkehrsgeschehen anders wahrzunehmen als der dahinterliegenden Verkehr, der unwissend über die vorherrschende Situation ist. Dies führt dazu, dass das Fahrzeug mit Kenntnis über die Situation eine Reaktion entsprechend des Geschehens ausführt. Das Ziel ist eine Lösung zu finden, sodass Fahrzeuge kontrolliert als Kollektiv, unabhängig von ihrem ursprünglichen Wissensstand, besser gegen das Verkehrsaufkommen gerüstet sind.

Unterstützend dazu soll gemäß der Schwarmintelligenz eine verhaltensbasierte Entwurfsstrategie [10, S. 96] nach Hamann definiert werden, sodass die Fahrzeuge klaren Regeln folgen und in passenden Situationen die zuvor definierten Signale aussenden. Die automatische Entwurfsstrategie unter Heranziehung von maschinellem Lernen oder Reinforcement Learning wird in dieser Entwicklungsarbeit nicht herangezogen, da sie die Komplexität übersteigen würden.

5 Realisierung

5.1 Steuerung der Ampel gemäß der Realität

Führt man die generierte Simulation mittels der graphischen Oberfläche von SUMO aus, so erkennt man Fahrzeuge, die sich im Szenario bewegen. Die Ampelschaltung ist auf einen Schaltzyklus von 45 Sekunden zwischen Grün- und Rot-Phase eingestellt. Im ersten Schritt soll nun die Möglichkeit geschaffen werden einen dynamischen Eingriff in das Schaltverhalten der Signalköpfe der Ampelanlage vorzunehmen, um die Ampelschaltungen aus der Realität, mittels der Daten des Content-Providers, zu realisieren. Dazu wird zunächst die Simulation analog zum Client-Server Paradigma mit Hilfe von TraCI gestartet. Dies ist durch folgenden Quellcode realisierbar:

```
1 import traci
2
3 traci.start([sumo_binary, "-c", sumo_config_file, "--start"])
```

Die erste Zeile importiert die TraCI Biblitohek und erlaubt den Zugriff auf dessen Funktionalitäten. Im nächsten Schritt wird die Simulation durch die Übergabe der Konfigurationsdatei ausgeführt. Eine beispielhafte Konfigurationsdatei muss dabei mindestens folgende Einträge enthalten:

```
1 <input>
2     <net-file value="../osm.net.xml.gz"/>
3     <route-files value="../routes.rou.xml"/>
4     <additional-files value="../osm.poly.xml.gz"/>
5 </input>
6
7 <traci_server>
8     <remote-port value="52244"/>
9 </traci_server>
10
11 <gui_only>
12     <gui-settings-file value="osm.view.xml"/>
13 </gui_only>
```

Zunächst werden alle weiteren benötigten Konfigurationsdateien eingebunden. Damit sind genauer die Beschreibung des Straßennetzwerks, die Routen über die Kanten (Straßen) der Simulation, sowie sonstige Texturen innerhalb der Simulation gemeint.

Weiterhin muss ein Server-Port definiert werden, unter dem die Simulation über die TraCI Schnittstelle gestartet wird. Zuletzt wird außerdem eine Konfigurationsdatei für die graphische Darstellung der Simulation eingebunden. Diese enthält den Einstiegspunkt auf der Karte, sowie die Festlegung eines Zeitabstands zwischen jedem Schritt in der Simulation. Da die Echtzeit-Simulation basierend auf den gewonnenen Daten aus der API stattfinden soll, ist es zwingend notwendig, dass jeder Schritt in der Simulation eine Sekunde benötigt. Dazu wird festgelegt, dass zwischen jedem Simulationsschritt eine Verzögerung von 1000 Millisekunden liegt. Durch Hinzufügen der folgenden Schleife kann die Simulation durch TraCI ausgeführt werden:

```
1 while traci.simulation.getMinExpectedNumber() > 0:
2     traci.simulationStep()
3
4 traci.close()
```

Im Code-Abschnitt ist erkennbar, dass geprüft wird, ob bereits Fahrzeuge innerhalb der Simulation sind, oder folgen werden. Sofern diese Bedingung wahr ist wird ein Schritt in der Simulation vollzogen. Sollten keine Fahrzeuge mehr innerhalb der Simulation sein, so wird die Schleife verlassen und die Simulation terminiert.

Die nachfolgende Logik wird daher innerhalb dieser Schleife ergänzt werden, da diese zur Laufzeit wiederholt aufgerufen wird.

Um die Daten der Ampelanlage aus der Realität in die Simulation einfließen zu lassen, müssen diese über die API abgefragt werden. Die API Schnittstelle bildet dabei die im Voraus entwickelte Anwendung, welche die Daten des Content-Providers beinhaltet und aufarbeiten kann. Im Anschluss muss der entsprechende Signalkopf gemäß der Schaltung aus der Realität die Signalfarbe für den ermittelten Zeitraum annehmen. Zunächst wird ein Modul definiert, welches die Python Bibliothek „requests“ importiert. Mit dieser kann eine HTTP-Request ausgeführt werden, um die API-Schnittstelle anzusprechen.

Der Endpunkt innerhalb der API muss mit den in Kapitel 2.2 beschriebenen Parametern aufgerufen werden. Dies bringt ein Hindernis in der Umsetzung auf, da die benötigten Positionsdaten ad hoc nicht zur Verfügung stehen. Der gewählte Lösungsansatz hierfür ist das Bestimmen von Positionsdaten vor jeder Kreuzungsanfahrt. Die verwendete Kreuzung besteht aus drei Anfahrten, für die jeweils ein Punkt gewählt werden muss, der innerhalb der Anfahrt liegt. Mittels der freien Kartensoftware OpenStreetMap können dafür drei Punkte bestehend aus Breitengrad und Längengrad bestimmt werden.

Weiterhin wird eine Kompassrichtung benötigt, zu der das Fahrzeug unter Normalbedingungen auf dem Punkt in Richtung der Ampel ausgerichtet wäre. Dies dient dem Zweck der Validierung, dass das Fahrzeug tatsächlich in Richtung der Kreuzung fährt. Die Berechnung erfolgt durch Angabe eines weiteren Punktes, der jeweils vor dem zuerst ausgewählten Punkt in Richtung der Kreuzung liegt. Aus diesen Informationen kann man mit Hilfe der Formel gemäß [14] die Kompassrichtung aus diesen zwei Punkten bestimmen. Eine Auflistung der Punkte, sowie den Kompassrichtungen zu jeder Anfahrt ist in Anhang E gegeben.

Nachdem die Punkte und Kompassrichtungen bekannt sind, kann der API Endpunkt angesprochen werden. Die ebenfalls zu übergebenen Parameter der Eigengeschwindigkeit des Fahrzeugs, sowie ob das Verkehrsaufkommen für die GLOSA-Bestimmung mit einfließen soll, können willkürlich gewählt werden, da diese keinen Einfluss auf die Ampelsignale haben. Die Rückmeldung der API (siehe Anhang B) bildet eine Antwort im JSON Format, aus der die Ampelsignale wie folgt extrahiert werden können:

```
1  def extract_tli(response):
2      ...
3      signals = response["signals"]
4      phases = []
5      count = len(signals[0]["predictions"])
6      predictions = signals[0]["predictions"]
7
8      if(count == 0):
9          return None
10     else:
11         phases.append([signals[0]["bulbColor"],
               predictions[0]["timeToChange"],
               predictions[0]["confidence"]])
12         if(count == 2):
13             phases.append([predictions[0]["bulbColor"],
                   predictions[1]["timeToChange"] -
                   predictions[0]["timeToChange"],
                   predictions[1]["confidence"]])
14         return phases
15 # [["Green", 11, 97], ["Red", 45, 95]]
```

Die obige Funktion beinhaltet den Mechanismus um die momentane und zukünftige Ampelphase aus der Antwort zu extrahieren. Das Schema der Antwort beinhaltet eine Liste von Signalen, die schrittweise bearbeitet wird. Zunächst wird die momentane Signalfarbe der Phase extrahiert. Im Anschluss wird die Startzeit der nächsten Phase analysiert und als Restzeit der momentanen Phase gesetzt. Sofern die übernächste Phase bekannt ist, kann die Dauer der nächsten Phase ermittelt werden, da diese endet, sobald die übernächste Phase beginnt.

Der Rückgabewert ist eine Liste bestehend aus Tupel, die die Signalfarbe („bulbColor“), die Dauer der Phase („timeToChange“), sowie die vorher angesprochene Wahrscheinlichkeit („confidence level“) beinhalten. Ein beispielhafter Rückgabewert der Funktion ist in Zeile 16 des obigen Abschnitts abgebildet. Das erste Tupel beschreibt die Phase, die zum Zeitpunkt der Antwort aktiv ist. Folglich wäre nach diesem Rückgabewert der Signalkopf für die nächsten 11 Sekunden grün, bevor im Anschluss eine Rot-Phase für 45 Sekunden eintritt.

Nachdem man dieses Verfahren für alle drei Anfahrtsrichtungen durchgeführt hat, sollen die Signalköpfe der Simulation entsprechend dieser Informationen gesetzt werden. Aus der Dokumentation der TraCI Bibliothek geht für diese Operation folgende Funktion hervor [13]:

```
1 setRedYellowGreenState(string, string) -> None
2
3 "Sets the named tl's state as a tuple of light definitions from
      rugGyYuoO, for red, red-yellow, green, yellow, off, where lower case
      letters mean that the stream has to decelerate."
```

Die Funktion erwartet zwei Übergabeparameter. Der erste Parameter bezieht sich auf die Identifikation der Ampel. Genauer meint das den eindeutigen Namen der Ampel innerhalb der Simulation.

Für diesen Zweck wurde die Ampel innerhalb der Konfigurationsdatei wie folgt definiert:

```
1 <tlLogic id="tli" type="actuated" programID="0" offset="0">
2     <phase duration="3"  state="rrr"/>
3 </tlLogic>
```

Das Element „tlLogic“ bezieht sich dabei auf die Logik der Ampelanlage. Abdeckt darin sind alle Signalköpfe der drei Anfahrten. Erkennbar ist dies anhand des „state“ Attributs, das fortan als Zeichenkette Z beschrieben wird. Jedes Zeichen $z \in Z$ beschreibt die aktuelle Signalfarbe für eine Anfahrt. Es ist möglich, dass die Zeichenfolge mehr Elemente enthält, als Anfahrten. Dies würde zur Folge haben, dass manche Anfahrten mitunter mehrere Signalköpfe enthalten, die unterschiedliche Schaltzeiten haben.

Für den vorliegenden Fall gilt für die Zeichenfolge Z:

$$\sum z \in Z = Anzahl\ Anfahrten$$

da jede Anfahrt für die ausgewählte Kreuzung genau einen Signalkopf enthält.

Mittels der Dauer (engl. „duration“) kann die Länge der Phase bestimmt werden.

Nach dem obigen Abschnitt wurden alle Signalköpfe der Ampel initial für drei Sekunden auf rot gesetzt, da nach der Dokumentation der Funktion *setRedYellowGreenState* das Zeichen r eine Rot-Phase signalisiert [13]. Um nun die Ampel entsprechend der Echtzeitdaten schalten zu lassen, müssen die vorher gewonnenen Informationen in das passende Schema gemäß der Zeichenkette Z umgewandelt werden. Zu diesem Zweck wird die nachfolgende Funktion definiert:

```
1  def get_current_phases():
2      if(approach2 == None or approach3 == None or approach1 == None):
3          return get_current_phases()
4
5      ttc = min([approach2[0][1], approach3[0][1], approach1[0][1]])
6
7      currentphase = ''
8
9      for phase in [approach2, approach3, approach1]:
10         if(phase[0][0] == 'Green'):
11             currentphase += 'g'
12         else:
13             currentphase += 'r'
14
15     return [currentphase, ttc]
```

Seien *approach*1, *approach*2 und *approach*3 gegeben als Listen von Tupel der aufkommenden Schaltzyklen der jeweiligen Anfahrt. Zunächst wird die Dauer der nächsten Phase berechnet. Da alle Signalköpfe durch einen Befehl kontrolliert werden, entspricht die Dauer der zu definierenden Phase der kleinsten Restdauer aller aktuellen Phasen. Abzubilden ist dies durch folgende Formel:

$$\min(\{(c,t,p) \mid (c,t,p) \in L \wedge (c,t,p)\, hat\, index\, =\, 0, L \in P\}, (c,t,p) \rightarrow t),$$

wobei P die Liste aller Anfahrten ist:

$$P = [L_1, L_2, L_3].$$

Ein Anfahrt ist definiert durch maximal zwei Phasen:

$$L = [(c,t,p),(c,t,p)]$$

und eine Phase ist festgelegt als:

$$(c,t,p) := (Signalfarbe, Restzeit, Wahrscheinlichkeit).$$

Dabei ist zu berücksichtigen, dass L eine geordnete Liste ist und die aktuelle Phase in Hinblick auf ihre Restdauer ausgewertet wird.

Das Resultat der Berechnung liefert das Minimum der Zeitwerte aller aktuellen Phasen. Dieser spiegelt die Zeit zum nächsten Phasenwechsel eines Signalkopfes innerhalb der Kreuzung wider.

Im weiteren Hinblick auf die Implementierung bedarf es der Untersuchung der aktuellen Phasen hinsichtlich ihrer Signalfarbe. Als Grundlage hierfür dient die Umwandlung der Signalfarben aus der API analog zu den Zeichen der Simulation:

- „Green“ → ’g’
- „Red“ → ’r’

Die Signalfarbe gelb wird in diesem Kontext nicht berücksichtigt, da der Content-Provider keine Daten für diese Phase zur Verfügung stellt. Es existieren grundlegende Regeln für Ampelsysteme, die den Zeitpunkt und die Dauer einer Gelb-Phase vorgeben [4]. Zwar ist es grundsätzlich möglich diese Ampelphase basierend auf den vorherrschenden Phasen zu ermitteln, allerdings wird darauf verzichtet, da der Wechsel zwischen den Ampelphasen somit häufiger auftreten würde.

Um dieses Verfahren mittels eines Beispiels zu demonstrieren sei die Menge P gegeben als:

P = [

$$[[\text{„}Red\text{“}, 13, 98], [\text{„}Green\text{“}, 45, 95]],$$
$$[[\text{„}Green\text{“}, 11, 97], [\text{„}Red\text{“}, 45, 95]],$$
$$[[\text{„}Green\text{“}, 12, 93], [\text{„}Red\text{“}, 44, 95]]$$

]

Nach dem Vorgehen oben würde nun für jedes $p \in P$ das erste Element herangezogen werden und hinsichtlich der Restdauer verglichen werden. Dabei wird die geringste Restdauer zurückgegeben und als Zeitpunkt zum nächsten Phasenwechsel festgelegt. Im hier aufgezeigten Beispiel würde $ttc = 11$ gesetzt werden, wobei die Einheit Sekunden - oder präziser: Simulationsschritten - entspricht. Die Ampelphasen würden gemäß der Umwandlung in „rgg“ übersetzt werden, da das erste Element von P eine Rot-Phase enthält und die anderen beiden Elemente eine grüne Phase liefern.

Um die aus der Funktion gewonnenen Erkenntnisse in der Simulation darzustellen kann der folgende Quellcode in den Rumpf der eingangs definierten Schleife hinzugefügt werden:

```
1 ...
2 if(ttc <= 0):
3     state, ttc = traffic_light_manager.get_current_phases()
4     traci.trafficlight.setRedYellowGreenState('tli', state) # setze die
          Signalfarbe der Ampel 'tli' entsprechend der gewonnenen
          Kenntnisse
5
6 ttc -= 1
7
8 ...
9
10 traci.simulationStep() # inkrementiert die Schrittweite der Simulation
```

Die aktuelle Phase aus dem Rückgabewert kann, wie in Zeile vier ersichtlich, auf die Ampel übertragen werden. Die Restzeit bis zum nächsten Phasenwechsel wird einer globalen Variable zugewiesen und nach jeder Iteration dekrementiert. Sollte die Restzeit den Wert null erreicht haben wird die Prozedur erneut ausgeführt. Dieses Verfahren ist in soweit schlüssig, da zuvor festgelegt wurde, dass eine Schrittdauer innerhalb der Simulation per se eine Sekunde benötigt.

Mit diesem Verfahren ist die Simulation befähigt, die Signalköpfe der Ampel entsprechend der Signale der realen Ampel umzusetzen.

5.2 Realisierung der vehicle-to-infrastructure Kommunikation

Nachdem hiermit die Infrastruktur entsprechend der Realität umgesetzt wurde, ist es notwendig den Fahrzeugen zu ermöglichen die GLOSA zu befolgen. Für diesen Anlass können bereits umgesetzte Verfahren wiederverwendet werden. Der Aufruf der API-Schnittstelle unterscheidet sich dahingehend, dass nun die echten Positionsdaten eines Fahrzeugs versendet werden müssen. Dies geschieht vor dem Hintergrund, dass das Fahrzeug eine Geschwindigkeitsempfehlung ausgehend von dessen Positionsdaten erhalten möchte. Für diese Anforderung stellt TraCI zwei Funktionen bereit, die es ermöglichen die Position eines Fahrzeugs zu ermitteln. Zunächst kann die Fahrzeugposition relativ zum Koordinatensystem der Simulation ausgegeben werden. Durch Aufruf der Funktion *convertGeo* der TraCI Bibliothek [13] kann diese Position in ein reales Paar aus Breitengrad und Längengrad umgewandelt werden. Die Möglichkeit diese Daten zu ermitteln resultiert aus dem in Kapitel 4.1 gewählten Ansatz, das Szenario ausgehend von dem realen Kartenabbild zu generieren.

Mittels der vorher implementierten Funktion um die API mittels HTTP-Request anzusprechen, kann die GLOSA für die aktuelle Fahrzeugposition ermittelt werden. Die daraus resultierende Antwort enthält neben den Signalen der Ampel, auf die das Fahrzeug zufährt, auch die GLOSA berechnet nach dem Ansatz in Anhang C. Mittels einer neuen Funktion, können alle Super-Fahrzeuge innerhalb der Simulation entsprechend der GLOSA bewegt werden:

```
1  def move_according_to_glosa(vehicle):
2      if helper.vehicle_did_not_cross_intersection(vehicle): # pruefen, ob
           das fahrzeug noch nicht ueber die kreuzung gefahren ist
3          ...
4          visualizer.create_glosa_polyline(vehicle) # zeichne eine linie
               zwischen fahrzeug und ampel
5          x, y = traci.vehicle.getPosition(vehicle) # fahrzeugposition
6          long, lat = traci.simulation.convertGeo(x, y) # fahrzeugposition
               in breiten- und laengengrad umwandeln
7          angle = traci.vehicle.getAngle(vehicle) # fahrtwinkel
8
9          current_speed = traci.vehicle.getSpeed(vehicle) * 3.6
10
11         # erhalte glosa von api:
12         glosa, distance, signals = glosa_for_position(lat, long, angle,
               current_speed if current_speed > 15 else 30)
13
14         if glosa == None or distance == None or signals == None:
15             return
16         # auslesen der parameter aus der antwort:
17         speed = get_recommended_speed(glosa)
18         decision = get_justification(glosa)
19         min_speed = get_minimum_speed(glosa)
```

```
20          max_speed = get_maximum_speed(glosa)
21          tli_store.write(traci.simulation.getTime(), vehicle, distance,
                glosa, signals) # speichern der erhaltenen werte
22          ...
23
24          traci.vehicle.setSpeed(vehicle, speed / 3.6) # erhaltene
                geschwindigkeitsempfehlung fuer das fahrzeug setzen
25
26      else:
27          traci.vehicle.setSpeed(vehicle, -1) # fuer den fall, dass das
                fahrzeug bereits ueber die kreuzung gefahren ist
```

Der vorliegende Ausschnitt aus dem Quellcode bildet die Logik für das Setzen der Fahrzeuggeschwindigkeit ab. Dazu notwendig wird die momentane Geschwindigkeit des Fahrzeugs, sowie der Fahrtwinkel des Fahrzeugs und die Position übergeben. Daraus resultierend können die Geschwindigkeitsempfehlung, sowie die Minimalgeschwindigkeit und Maximalgeschwindigkeit gelesen werden. Schlussendlich wird die Geschwindigkeit des Fahrzeugs an die Empfehlung angepasst. Dabei ist erwähnenswert, dass die Geschwindigkeit unmittelbar vom Fahrzeug angenommen wird. Diese Umsetzung wird innerhalb des Fazits evaluiert werden.

Sofern das Fahrzeug die Kreuzung noch nicht überfahren hat, wird die Geschwindigkeit entsprechend der oben beschriebenen Logik angepasst. Sollte das Fahrzeug die Kreuzung bereits überfahren haben, wird die Kontrolle des Fahrzeugs zurück an die Simulation übergeben. Dies hat den Hintergrund, dass für die Untersuchung im folgenden Fall nur Fahrzeuge vor der Kreuzung relevant sind. Die neu definierte Funktion wird ebenfalls in den Rumpf der Schleife aufgenommen und folglich bei jeder Iteration ausgeführt. Eine Einschränkung die dabei getroffen werden muss, sind die Fahrzeuge, die von dieser Funktion beeinflusst werden.

Aus der Konfiguration für die Routen und Fahrzeuge geht folgendes hervor:

```
1 ...
2 <flow id="f_4" begin="0.00" route="approach1_left" end="3600.00"
      vehsPerHour="300.00"/>
3 <flow id="v2v2i" begin="10.00" color="red" route="approach2_straight"
      end="3600.00" vehsPerHour="50.00"/>
4 ...
```

Erkennbar sind zwei Elemente, des Typs $flow$. Ein „Flow“ gemäß SUMO Dokumentation, beschreibt ein wiederholtes Aufkommen von Fahrzeugen, die die gleichen Parameter erhalten [13]. Dabei kann wie im obigen Beispiel der Startpunkt innerhalb der Simulation, die Farbe der Fahrzeuge, der Endpunkt, sowie die Häufigkeit eingestellt werden. Im hier aufgezeigten Ausschnitt wird ein Flow f_4 erstellt, der normale Fahrzeuge widerspiegelt.

Ein weiteres Element mit der Identifikationsnummer $v2v2i$ repräsentiert einen Fluss von Super-Fahrzeugen.

Für die vorher genannte Implementierung bedeutet das, dass nur Fahrzeuge innerhalb der Funktion manipuliert werden sollen, die aus dem Flow bestehend aus Super-Fahrzeugen generiert wurden. Zu diesem Zweck kann mittels der Funktion $getIDList()$ aus der TraCI Bibliothek eine Liste aller aktiven Fahrzeuge innerhalb der Simulation angefordert werden. Die Liste beinhaltet die Identifikationsnummer nach dem eben vorgestellten Schema, wobei jedes Fahrzeug durch eine zusätzlich angehängte Laufnummer identifiziert wird.

Um nun die Super-Fahrzeuge innerhalb der Simulation erkennen zu können, wird folgender Code verwendet:

```
1  def get_super_vehicles(vehicles):
2      super_vehicles = []
3
4      for vehicle in vehicles:
5          if vehicle.startswith("v2v2i"):
6              super_vehicles.append(vehicle)
7
8      return super_vehicles
9
10 ...
11 for vehicle in get_super_vehicles(vehicles):
12     move_according_to_glosa(vehicle)
13 ...
```

Mittels einer Schleife kann jedes in der Liste enthaltene Fahrzeug hinsichtlich des Namens untersucht werden. Für den Fall, dass das Fahrzeug die Bedingung erfüllt, wird es in einer seperaten Liste mit allen weiteren Super-Fahrzeugen zurückgegeben. Im Schleifenrumpf, über die die Simulation kontrolliert wird, kann dann der Funktionsaufruf in Zeile 11-12 hinzugefügt werden. Dies ermöglicht die Steuerung aller existierenden Super-Fahrzeuge innerhalb der Simulation, unter der Prämisse, dass diese noch nicht über die Kreuzung gefahren sind.

Im Laufe dieses Realisierungsprozesses bedarf es einigen Erweiterungen dieser Funktionalität, da die Fahrzeuge nicht nur durch die Ampel, sondern gemäß des Forschungsziels gleichermaßen durch andere Fahrzeuge beeinflusst werden sollen. Die Umsetzung der V2I Kommunikation, hinsichtlich der vorgeführten Verfahren und Mechanismen, kann als abgeschlossen und gemäß den Vorgaben umgesetzt betrachtet werden.

5.3 Erweiterung um vehicle-to-vehicle Kommunikation

In den vorherigen Kapiteln, insbesondere Kapitel 3.1 und Kapitel 3.3 ging hervor, welche Grundlagen und Prinzipien für die Umsetzung der bidirektionalen Kommunikation zwischen Fahrzeugen als wichtig erscheinen. Die theoretischen Hintergründe wurden schließlich in ein Konzept umgewandelt, das eine einheitliche Kommunikationsstrategie verlangt und die Fähigkeit ein Netzwerk zwischen Fahrzeugen aufzubauen, um Sender und Empfänger von Nachrichten auszumachen. TraCI stellt für diesen Anwedungszweck keine Lösung bereit, allerdings können die vorher genutzten Mechanismen der Datenerhebung verschiedener Fahrzeuge als Grundlage genutzt werden, um davon ausgehend ein Fahrzeugnetzwerk aufzubauen.

Da TraCI im vorliegenden Anwendungsfall die Schlüsselkomponente zur Kommunikation mit der Simulation ist, muss ein weiteres Paket definiert werden, welches an die FCD der Fahrzeuge mittels TraCI andocken kann. Aus diesen Daten muss ein Netzwerk zwischen Fahrzeugen innerhalb der Simulation aufgebaut werden.

Der technische Entwurf sieht dabei folgende Kommunikationsstrategie vor, die aus den in Kapitel 4.3 aufgestellten theoretischen Anforderungen resultieren:

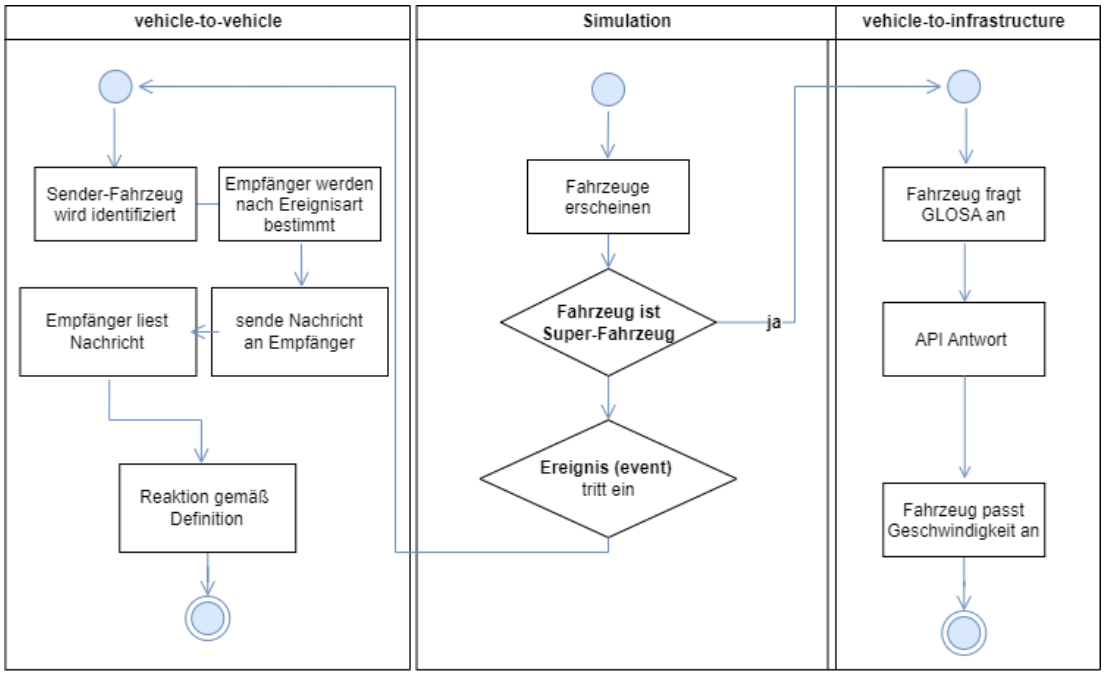


Abbildung 5.1: Kommunikationsstrategie innerhalb der Simulation

Die Abbildung unterscheidet drei Komponenten innerhalb der Simulation. Die V2I Kommunikation wurde bereits in Kapitel 5.2 implementiert. Weiterhin wurde das Erscheinen neuer Fahrzeuge mittels einem Flow umgesetzt.

In diesem Abschnitt sollen Ereignisse definiert werden und entsprechende Reaktionen basierend auf ausgewählten Sender-Fahrzeugen durchgeführt werden.

Der SIS als geteilter Informationsraum, den die Fahrzeuge nutzen können um einen gemeinsamen Wissensvorsprung zu erzielen, kann im ersten Schritt durch eine Interface Beschreibung abgebildet werden:

```
1 class SharedInformationSpace:
2     def __init__(self)
3     def write(self, time, sender, receivers, signal, data)
4     def read(self, receiver)
```

Der SIS ist dabei ein Singleton, das das Aufnehmen und Herausgeben von Daten ermöglicht. Dies meint im Sinne der V2V Kommunikation, dass alle Fahrzeuge über diese Instanz in der Lage sind einen Informationsaustausch zu betreiben. Diese globale Zusammenführung des gesamten Informationsaustauschs könnte in der Realität durch einen dezentralen Dienst, wie zum Beispiel ein ITS zusammengeführt werden, da die gespeicherten Informationen zumeist aus FCD bestehen.

Die Funktion *write* repräsentiert das Aussenden einer Nachricht an potentielle Empfänger. Die Implementierung der Funktion wird wie folgt gewählt:

```
1         for receiver in receivers:
2             if not any(message[1] == sender and message[2] == receiver
                  and message[3] == signal for message in self.messages):
3                 self.messages.append((time, sender, receiver, signal,
                      data))
4                 visualizer.create_vehicle_polyline(sender, receiver)
```

Eine Nachricht enthält den Zeitpunkt der Simulation, zu dem die Nachricht versendet wurde. Weiterhin ist in ihr der Sender und Empfänger enthalten, um eine Zuordnung zu ermöglichen. Ein Signal gemäß der in Kapitel 4.3 festgelegten Signale wird zusammen mit einem Datensatz innerhalb des SIS abgelegt. Darüber hinaus wird sichergestellt, dass kein Fahrzeug mehrfach das gleiche Signal an den gleichen Empfänger senden kann. Dies hat den Hintergrund, dass die Informationen, die ohnehin bereits empfangen und verarbeitet wurden, nicht mehrfach verarbeitet werden müssen. Als Grundlage dient das Szenario in dem ein Fahrzeug vor einer Kreuzung wartet und nachfolgende Fahrzeuge alarmiert. Es ist ausreichend, dass das Fahrzeug diese Information einmalig sendet, da sich die Position des Fahrzeugs bis zur nächsten grünen Ampelphase nicht ändern wird da es in der Warteschlange verbleibt. Nachdem eine Nachricht abgelegt wurde, wird eine Polylinie zwischen Sender und Empfänger gezeichnet, um die Kommunikation zwischen diesen beiden Knoten zu visualisieren.

Im Anschluss können die Empfänger die Nachrichten auslesen. Dies geschieht über die Funktion *read*. Diese wird für jedes Fahrzeug innerhalb der Simulation aufgerufen. Das Resultat ist eine Liste von Nachrichten, die ein Fahrzeug erhalten hat. Mittels den zwei Funktionen kann der Kommunikationsfluss zwischen Fahrzeugen ermöglicht werden.

Im Weiteren muss ein Netzwerk aufgebaut werden, das Fahrzeuge bestimmt die potentielle Sender und Empfänger von Nachrichten werden. Dem zugrunde liegt die TraCI Bibliothek, da mit ihr die relevanten Zustandsinformationen der Fahrzeuge - die FCD - ausgelesen werden können. Die ausgangs definierten Signale in Kapitel 4.3 dienen als Motivation für die Umsetzung dieses Netzwerks.

Zunächst sollen Fahrzeuge bestimmt werden, die an einer roten Ampel auf die nächste grüne Ampelphase warten. Die Simulation stellt dabei folgende Grundprinzipien bereit:

1. Neben der Ampel existiert kein Hindernis, das die Fahrzeuge vor oder nach der Kreuzung zum Stillstand bringen könnte.

2. Jedes Fahrzeug setzt die Fahrt nach dem Warten unverzüglich fort.

3. Die Ampel ist gemäß der Daten aus der Realität geschaltet.

Zunächst ist es notwendig sicherzustellen, dass neben der Ampel kein weiteres Hindernis existiert, das einen Stillstand verursachen könnte. Dies hat den Hintergrund, dass die Fahrzeuge für diesen Fall eine Warnung an den nachfolgenden Verkehr senden. Weiterhin ist erwähnenswert, dass die Fahrzeuge ihre Fahrt unmittelbar weiterführen, sobald die Ampel ihre Signalfarbe ändert oder das vorausfahrende Fahrzeug anfährt. Dies ist insofern wichtig, da es möglich sein muss auf Basis des wartenden Fahrzeugs Rückschlüsse für den nachfolgenden Verkehr zu ziehen. Im Falle der Simulation fährt das Fahrzeug unmittelbar los, für die Realität würde dieses Verhalten unterstellt werden, sodass eine zeitliche Einordnung überhaupt möglich gemacht wird. Andernfalls wäre es nicht möglich einen Zeitpunkt auszumachen an dem der Folgeverkehr an der Kreuzung ankommen muss, unter der Prämisse, dass der vorrausfahrende Verkehr bereits losgefahren ist. Zuletzt ist es unabdingbar, dass die Ampel wie in der bisherigen Implementierung an die Daten der Realität gekoppelt ist und demnach gleich schaltet. Hintergrund ist, dass die Daten zwischen der Ampel innerhalb der Simulation und den Daten der Super-Fahrzeugen synchronisiert sein muss, sodass Folgerungen und Geschwindigkeitsanpassungen stets auf einer korrekten und einheitlichen Datenbasis erfolgen. Sofern die Implementierung wie bisher fortgeführt wird, ist dieser Punkt erfüllt.

Nachdem die grundlegenden Voraussetzungen erfüllt sind, kann begonnen werden Fahrzeuge zu identifizieren, die momentan vor einer roten Ampel warten. Dem zugrunde liegt folgender Quellcode:

```
1 def detect_waiting_vehicles():
2     vehicle_ids = traci.vehicle.getIDList()
3     waiting_vehicles = []
4 
5     for vehicle in vehicle_ids:
6         if traci.vehicle.getWaitingTime(vehicle) > 0 and
              helper.vehicle_did_not_cross_intersection(vehicle):
7             waiting_vehicles.append(vehicle)
8 
9     return waiting_vehicles
```

Zunächst wird die Liste aller Fahrzeuge innerhalb der Simulation angefordert. Nachdem dies erfolgt ist, wird durch diese Liste iteriert und mittels der Funktion $getWaitingTime$ überprüft, ob das Fahrzeug an einer Ampel wartet. Dies wird festgestellt, indem untersucht wird, ob die Geschwindigkeit des Fahrzeugs $\leq 0.1 m/s$ beträgt und das Fahrzeug bereits mindestens einen Simulationsschritt in diesem Zustand ist. Weiterhin wird sichergestellt, dass das Fahrzeug noch nicht über die Kreuzung gefahren ist, folglich also nur vor der Ampel warten kann. Sofern die Bedingungen wahr sind wird das Fahrzeug zu einer weiteren Liste bestehend aus allen wartenden Fahrzeugen hinzugefügt.

Diese Liste repräsentiert potentielle Sender des warnenden Signals, das Aufschluss darüber erteilt, dass das Fahrzeug an der Ampel wartet. Um die Empfänger dieser Nachrichten zu bestimmen müssen folgende Voraussetzungen festgelegt werden:

1. Die Fahrzeuge die angesprochen werden fahren die gleiche Route wie das Sender-Fahrzeug.

2. Die Fahrzeuge befinden sich im nachfolgenden Verkehr und haben folglich weniger Strecke als das Sender-Fahrzeug zurückgelegt.

3. Die Fahrzeuge sind nicht bereits Teil der wartenden Fahrzeugschlange an der Ampel, da eine Kommunikation andernfalls keinen Nutzen hätte.

4. Die anzusprechenden Fahrzeuge müssen Super-Fahrzeuge sein, da nur diese ihre Geschwindigkeit basierend auf den Erkenntnissen der GLOSA und der Fahrzeug-Kommunikation anpassen können.

Diese Punkte spiegeln das Gedankenkonstrukt für die Ermittlung der Adressaten dieser Nachricht wider. Um nun zu evaluieren, welche Fahrzeuge betreffend der Anforderungen als Empfänger dieser Nachricht in Frage kommen, wird die FCD des Sender-Fahrzeugs aus der Simulation extrahiert:

```
1 roadId = traci.vehicle.getRoadID(vehicle) # road the vehicle is
      currently on
2 route = traci.vehicle.getRoute(vehicle) # route of the vehicle
3 vehiclePos = traci.vehicle.getLanePosition(vehicle) # distance the
      vehicle passed on the current road
```

Gegeben sei ein aus der vorherigen Funktion ermitteltes Fahrzeug *vehicle*, das den nachfolgenden Verkehr durch Senden einer Nachricht alarmieren soll. Dazu wird zunächst die momentane Straße auf der das Fahrzeug positioniert ist, sowie die Route die das Fahrzeug fährt und die Position innerhalb der Straße herangezogen. Die Route des Fahrzeugs sei gegeben als geordnete Liste, bestehend aus Straßen, die das Fahrzeug innerhalb der Route befährt. Dabei ist $s \in route$ die aktuelle Straße, auf der sich das Fahrzeug befindet. Die aktuelle Straße s hat den Index p innerhalb der geordneten Liste, wobei die Straße mit dem Index $p-1$ die vom Fahrzeug zuvor befahrene Straße ist und die Position $p+1$ die Straße ist, die das Fahrzeug im Anschluss befahren wird.

Daraus lässt sich schließen, dass die für das Signal relevanten Fahrzeuge hinter dem Sender-Fahrzeug, also insbesondere auf dahinterliegenden Straßen verortet sein müssen. Aus diesem Grund muss der zuvor definierte Index p ermittelt werden. Aus der bekannten Route, sowie der momentane Straße kann durch die Funktion $route.index(roadId)$ der Index der aktuellen Straße innerhalb der Route bestimmt werden.

Um fortlaufend alle hinter dem Sender-Fahrzeug befindlichen Super-Fahrzeuge ausfindig zu machen, kann mittels des bekannten Index der aktuellen Straße, durch die Liste der Straßen auf der Route iteriert werden.

Mittels der Funktion $getLastStepVehicleIDs(route[i])$ können alle Fahrzeuge auf einer Straße ermittelt werden. Die Straße wird dabei durch eine Laufnummer i bestimmt, die solange inkrementiert wird, bis i dem momentanen Index p entspricht. Für die Liste an Fahrzeugen wird überprüft, ob es sich um ein Super-Fahrzeug handelt. Sofern dies der Fall ist, wird das Fahrzeug zu einer Liste von Empfängern hinzugefügt. Entspricht die Laufnummer i in der letzten Iteration dem Wert von p, wird die Straße analysiert auf der das Sender-Fahrzeug momentan befindlich ist. Für diesen Fall müssen tiefgehendere Bedingungen definiert werden, welche im Folgenden aufgezeigt werden:

```
1 for vid in vehiclesOnEdge:
2     if vid == vehicle:
3         continue # ignoriere die eigene identitaet
4     approaching_vehicle_pos = traci.vehicle.getLanePosition(vid)
5     if helper.is_super_vehicle(vid) and traci.vehicle.getSpeed(vid) > 0
          and approaching_vehicle_pos < sender_vehicle_pos and
          traci.vehicle.getRoute(vid) == route:
6         approaching_behind_vehicles.append(vid)
```

Sei *vehiclesOnEdge* die Liste aller Fahrzeuge, die sich auf der gleichen Straße wie das Sender-Fahrzeug befinden. Um nun zu ermitteln welche Fahrzeuge potentielle Empfänger der alarmierenden Nachricht sind, müssen die zuvor aufgestellten Prämissen hinsichtlich der Detektierung von Adressaten konkretisiert werden. Für alle Straßen die das Sender-Fahrzeug bereits abgefahren hat ist es ausreichend zu überprüfen, dass die Empfänger Super-Fahrzeuge sind. Für die Straße auf der sich das Fahrzeug zum Zeitpunkt der Ermittlung befindet bedarf es der Beurteilung, dass ein potentieller Empfänger zweifelsfrei ein Nachfolger des Sender-Fahrzeugs ist und nicht bereits selbst ein Teil der Fahrzeugkolonne. Dem zugrunde liegt der Quellcode, der festlegt, dass die zurückgelegte Distanz des zu adressierenden Fahrzeugs geringer sein muss als die des Senders. Weiterhin muss sichergestellt werden, dass das Fahrzeug in Bewegung ist, indem die Geschwindigkeit durch die Funktion *getSpeed* analysiert wird. Sofern die Bedingungen zutreffen, kann das Fahrzeug zweifelsfrei als dahinterliegendes Fahrzeug eingeordnet werden.

Die aus diesem Vorgehen gewonnene Liste an Super-Fahrzeugen, die zum Zeitpunkt des Stillstands eines Fahrzeugs an der Kreuzung im dahinterliegenden Verkehr fahren, müssen über den Stillstand des Fahrzeugs alarmiert werden. Die Alarmierung geschieht dabei durch das Senden einer Nachricht an die betreffenden Fahrzeuge.

Mittels der vom SIS bereitgestellten Funktionen *write* und *read* wurden zuvor bereits die theoretischen Hintergründe erläutert. Eine konkrete Implementierung im Falle des Signals *RED* würde folgendermaßen realisiert werden:

```
1 sis.write(traci.simulation.getTime(), sender, receivers,
      signals.Signal.RED, traci.vehicle.getPosition(sender))
```

Der Aufruf der Funktion *write* beinhaltet das Signal, sowie die Position des Senders. Diese Informationen können im Anschluss von einem Super-Fahrzeug genutzt werden um die GLOSA entsprechend der bereits wartenden Fahrzeuge anzupassen.

Sofern ein Super-Fahrzeug nun auf die Nachricht zugreift, muss es prüfen ob eine Verminderung der Geschwindigkeit aufgrund der wartenden Fahrzeuge erforderlich ist.

Die Implementierung der GLOSA gemäß Kapitel 2.2 sieht dabei das Heranziehen des Weg-Zeit Gesetzes vor. Die Berechnung für die angepasste dynamische Geschwindigkeitsempfehlung erfolgt nun auf Basis der Nachrichten wartender Fahrzeuge. Die übersendeten Positionsdaten der Fahrzeuge können für die Berechnung der angepassten Geschwindigkeitsempfehlung herangezogen werden.

Gegeben sei für ein Super-Fahrzeug das Wissen über die Ampel gemäß der Antwort der API (siehe Anhang B). Darin enthalten ist neben dem Abstand zur Ampel auch die Signalphasen und die GLOSA bestehend aus der Minimalgeschwindigkeit, der Maximalgeschwindigkeit und der daraus resultierenden Empfehlung. Analog zu den Berechnungen der GLOSA kann nun, gemäß des Quellcodes in Anhang F, eine angepasste Geschwindigkeitsempfehlung bestimmt werden.

Durch das Aufkommen weiterer Verkehrsteilnehmer wird der Abstand zwischen dem Super-Fahrzeug und dem Sender der Warnnachricht herangezogen. Für die Berechnung sind weiterhin die Signalphasen, der Zeitpunkt innerhalb der Simulation, sowie die Momentangeschwindigkeit von großer Relevanz.

Zunächst werden die erhobenen Signalphasen hinsichtlich ihrer Aktualität geprüft. Dies geschieht mit Hilfe der Zeitdifferenz zwischen dem Zeitpunkt der Speicherung und dem aktuellen Zeitpunkt in der Simulation. Sei die Zeit bis eine Grünphase eintritt zusätzlich als Time-to-Green (TTG) benannt. Für den vorliegenden Fall in dem die Ampel rot ist und eine grüne Phase folgt, ist die Minimalgeschwindigkeit definiert als:

$$v_{min} := \frac{Distanz\ zur\ Kreuzung}{\mathrm{TTG} + Dauer\ Grünphase}$$

Die Maximalgeschwindigkeit wird durch die wartenden Fahrzeuge maßgeblich beeinflusst:

$$v_{max} := \frac{Distanz\ zum\ Senderfahrzeug}{\mathrm{TTG} + Anzahl\ wartende\ Fahrzeuge * 2}$$

Die im Nenner enthaltene Zeit der Formel zur Bestimmung von v_{max} ist eine Annäherung für die Dauer, die die Fahrzeuge benötigen, um bei Beginn der Grün-Phase zu beschleunigen. Der Hintergrund ist, dass vermieden werden soll, dass das Super-Fahrzeug durch die Fahrzeugkolonne die Geschwindigkeit weiter reduzieren muss.

Nachdem der Bereich für eine valide Geschwindigkeit basierend auf den Informationen des wartenden Fahrzeugs erneut berechnet wurde, kann mittels eines Entscheidungsbaumes die optimale Geschwindigkeit für das Fahrzeug definiert werden:

Sofern die aktuelle Geschwindigkeit innerhalb des neuen Geschwindigkeitsbereiches liegt - zwischen v_{min} und v_{max} - kann das Fahrzeug die Geschwindigkeit beibehalten. Für den Fall, dass das Fahrzeug eine zu hohe oder zu niedrige Geschwindigkeit aufweist, muss es diese entsprechend der berechneten Ergebnisse anpassen.

Als Resultat sollte das Fahrzeug aufgrund der geänderten Geschwindigkeit erst an der Fahrzeugkolonne ankommen, sobald diese sich bereits in Bewegung gesetzt hat.

Um ebenfalls auf die Frage einzugehen, wie der Ablauf zu betrachten ist, sofern mehrere Fahrzeuge nacheinander an der Kreuzung zum Stillstand kommen, sei folgendes Kommunikationsdiagramm gegeben:

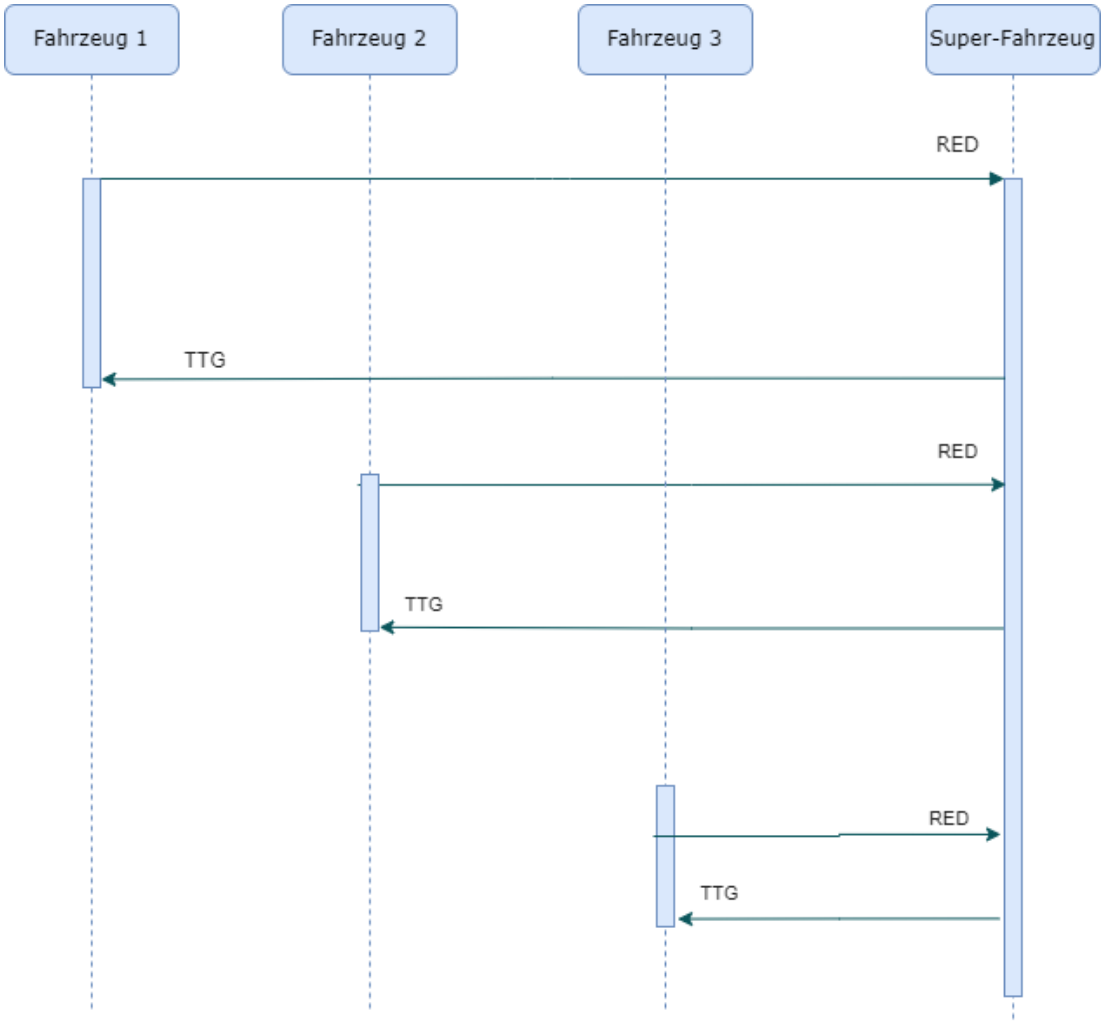


Abbildung 5.2: Kommunikation zwischen wartenden Fahrzeugen und einem Super-Fahrzeug

Zu erkennen sind drei Fahrzeuge, die nacheinander an der Kreuzung zum Stillstand kommen und das Signal *RED* aussenden. Das Super-Fahrzeug wird daraufhin die Geschwindigkeit entsprechend des Sender-Fahrzeugs anpassen und die verbleibende Dauer bis zur Grün-Phase an den Sender zurückschicken. Da die Fahrzeuge wie in der Realität nacheinander an der Kreuzung ankommen, wird das Fahrzeug seine Geschwindigkeit schrittweise reduzieren. Die zurückgesendete TTG findet dabei keine Anwendung bei den Fahrzeugen, da diese ohnehin unmittelbar nach Beginn der Grün-Phase anfahren.

In der Realität könnte diese Information besonders für autonome Fahrzeuge von Nutzen sein, da keine Beobachtung der Ampelanlage mittels eines Sensors erforderlich wäre.

Mittels diesem Verfahren sind die Super-Fahrzeuge befähigt, die Geschwindigkeitsempfehlung resultierend aus der V2I Kommunikation mit Hilfe von V2V Kommunikation an den realen Verkehr anzupassen, um so eine optimalen Fahrgeschwindigkeit zu ermöglichen. Dabei muss angemerkt werden, dass sich diese auf das Super-Fahrzeug beschränkt. Durch das Reduzieren der Geschwindigkeit des Super-Fahrzeugs besteht das potentielle Risiko den nachfolgenden Verkehr zu verlangsamen und so den Verkehrsfluss zu stören.

Da die Forschungsfrage der Verbesserung des Verkehrsflusses unter Berücksichtigung der Schwarmintelligenz nachgeht, genügt dieser bisherige Ansatz nicht vollständig, da nur ein einzelnes Individuum berücksichtigt wird.

Aus diesem Grund müssen die restlichen Signale gemäß Kapitel 4.3 implementiert werden.

Gegeben sei ein Fahrzeug, welches hinter einem Super-Fahrzeug fährt und durch dieses ausgebremst wird. Gemäß des Signals $MOVE$ soll das Fahrzeug befähigt werden eine Aufforderung an das Super-Fahrzeug zu schicken, um dem nachfolgenden Verkehr zu ermöglichen ebenfalls die grüne Welle zu befolgen. Die Ermittlung zielt dabei auf Fahrzeuge ab die eine Geschwindigkeit aufweisen, die unter der Höchstgeschwindigkeit auf der Straße liegt. Diese sei als $50\,\mathrm{km\,h^{-1}}$ gegeben. Fahrzeuge die mindestens $10\,\mathrm{km\,h^{-1}}$ unter der Höchstgeschwindigkeit liegen, werden als Fahrzeuge betrachtet, die ausgebremst werden. Diese Fahrzeuge senden nun ein Signal an das vorausfahrende Super-Fahrzeug aus, welches analog zur Ermittlung der nachfolgenden Super-Fahrzeuge ermittelt werden kann. Lediglich ein Unterschied in der Implementierung besteht darin, dass die Fahrzeuge nach einem Fahrzeug im Netzwerk suchen, welches mehr Strecke als das Sender-Fahrzeug zurückgelegt hat und noch nicht über die Kreuzung gefahren ist.

Nachdem das Super-Fahrzeug das Signal innerhalb einer Nachricht erhalten hat, wird die Momentangeschwindigkeit und die GLOSA analysiert. Sofern die Geschwindigkeit des Fahrzeugs unterhalb der Maximalgeschwindigkeit liegt, welche sowohl aus der reinen V2I Kommunikation oder unter Einfluss von V2V Kommunikation von wartenden Fahrzeugen berechnet worden sein kann, wird das Super-Fahrzeug die Geschwindigkeit erhöhen. Das erhoffte Resultat dieser Geschwindigkeitsanpassung ist, dass das Super-Fahrzeug weiterhin ohne Abbremsen über die Kreuzung fahren kann, wobei nun der nachfolgende Verkehr ebenfalls die Möglichkeit erhält über die Kreuzung zu fahren bevor die nächste Rot-Phase beginnt.

Mittels dieser Ergänzung kann der Verkehrsfluss auf einer Anfahrtsrichtung begünstigt werden.

Für das Signal $TURN$ wird nun das Wissen aus den Implementierungen der Signale RED und $MOVE$ kombiniert. Wenn ein Fahrzeug an der Haltelinie wartet wird ein Super-Fahrzeug auf der gegenüberliegenden Fahrspur gesucht und entsprechend des Kommunikationsablaufs adressiert. Das angesprochene Super-Fahrzeug wird überprüfen, ob seine momentane Geschwindigkeit höher als die Minimalgeschwindigkeit ist, um über die Kreuzung zu fahren. Sofern dies der Fall ist, reduziert das Fahrzeug seine Geschwindigkeit entsprechend der Minimalgeschwindigkeit. Daraus resultierend entsteht eine Lücke in der Fahrzeugkolonne, wodurch das Sender-Fahrzeug den Abbiegevorgang einleiten kann.

Durch die Realisierung dieses Signals ist es ebenfalls möglich die Optimierung des Verkehrsflusses auf das gesamte Verkehrsgeschehen auszubauen und nicht lediglich auf ein einzelnes Individuum oder Individuen innerhalb einer Kreuzungsanfahrt. Ob und inwiefern sich die Realisierung innerhalb der Simulation tatsächlich bewährt, soll im Anschluss durch Analyse von Simulationsparametern, insbesondere den Fahrgeschwindigkeiten, überprüft werden.

5.4 Qualitätssicherung

Ein Softwareprodukt wird stets durch Qualitätssicherung begleitet. Die hinzugefügte Logik zur Simulation lässt sich in diesem Sinne als Softwareprodukt betiteln. Im Laufe des Implementierungsprozesses ist ersichtlich geworden, dass die Umsetzungen im Code zu Anfang Fehler aufweisten oder ein Verbesserungspotential enthielten. Um die Robustheit der Simulation zu stärken und Validierungsprozesse innerhalb der Entwicklung zu ermöglichen wurde eine Anforderung zum vorausgegangenen Konzept ergänzt: Die Möglichkeit ein Feature mittels gleicher Bedingungen mehrfach zu testen.

Um ein Szenario später wiederholt ausführen zu können dienen folgende Grundüberlegungen:

- Die Simulation startet stets mit den selben Fahrzeugen zur gleichen Zeit.
- Die Fahrzeuge werden abhängig von der momentanen Ampelschaltung und der Geschwindigkeitsempfehlung beeinflusst.
- Die Aufrufe des Moduls zur Kommunikation mit der API folgt zu jeder Zeit dem gleichen Schema.

Aus diesen Überlegungen lässt sich schließen, dass der kritische Faktor die Daten der API-Schnittstelle sind, da diese über den Fortgang der Simulation maßgeblich entscheiden. Damit die Simulation dahingehend verbessert werden kann, müssen die Daten der API-Schnittstelle im Kontext der Simulation gespeichert werden, um diese später wiederholt aufrufen zu können. Damit würde die Simulation einerseits eine Echtzeit-Simulation bleiben, da die Daten lediglich gespeichert werden, andererseits besteht die Möglichkeit ein Feature unter Heranziehen von erprobten Daten zu verbessern. Dies meint ferner, dass Szenarien die als besonders anschaulich oder wertbringend klassifiziert werden zu einem späteren Zeitpunkt erneut aufgegriffen werden können.

Um diese Funktionalität zu realisieren, wurde das Modul erweitert, das die Kommunikation mit der API enthält. Insofern können nun die erhaltenen Antworten innerhalb einer JSON-Datei abgelegt werden.

Dazu dient die nachfolgende Erweiterung des bestehenden Programmcodes:

```
1 offlinemode = True
2 store = False
3 scenario = "scenario2"
4 response_msg = 0
5     ...
6     if offlinemode:
7             return previously_stored_response()
8
```

```
9       if store:
10          with open(f"api/rest/{scenario}//{response_msg}.json", "w") as
                outfile:
11              json.dump(response, outfile)
12
13          response_msg += 1
14
15      return response_json
```

Gegeben sei ein erfolgreicher Aufruf der API, der die Antwort innerhalb der Variable *response* abgelegt hat. Sofern der boolesche Wert *store* wahr ist wird die Antwort innerhalb eines Ordners benannt nach dem Wert der Variable *scenario* gespeichert. Der Dateiname entspricht dabei einer Laufnummer, welche im Anschluss an das Verfahren inkrementiert wird. Für den Fall, dass die boolesche Variable $offlinemode$ wahr ist, wird die nachfolgende Funktion aufgerufen:

```
1 def previously_stored_response():
2
3       with open(f"api/rest/{scenario}/{response_msg}.json", "r") as infile:
4           loaded_response = json.load(infile)
5
6       response_msg += 1
7
8       return loaded_response
```

Der Aufruf dieser Funktion impliziert, dass auf ein zuvor gespeichertes Szenario zugegriffen werden soll. Durch die Übergabe der Variable in der der Speicherort festgelegt ist, sowie der Laufnummer, kann auf die entsprechende Antwort der API zugegriffen werden. Im Anschluss an das Verfahren wird diese Nachricht zurückgegeben und die Laufnummer erhöht, sodass das unter dem Verzeichnis gespeicherte Szenario rekonstruiert werden kann. Da wie ausgangs erwähnt die Abfolge der Anfragen an die API stets nach dem gleichen Schema verläuft, wird der Programmcode nun in der selben Weise funktionieren wie zum Zeitpunkt des Speicherns des ausgewählten Szenarios. Mittels dieses Verfahrens ist es möglich das entsprechende Szenario erneut nachzustellen, da anstelle des erneuten Aufrufs der API auf die bereits gespeicherten Daten zurückgegriffen wird. Die Antwort liefert dabei die notwendigen Informationen über die Schaltungen und Geschwindigkeitsempfehlungen, da diese ohne weitere Modifikation gespeichert wurden. Unter Zuhilfenahme von diesem Verfahren ist es möglich den Algorithmus für die Geschwindigkeitsempfehlung eines Fahrzeugs oder die Kommunikation zwischen Individuen des Kollektivs schrittweise anhand eines bestimmten Szenarios zu verbessern. Mittels dieses Verfahrens konnte die Modellierung der verhaltensbasierten Entwurfsstrategie nach Hamann [10] etabliert werden.

6 Zusammenfassung

Auf Basis der vorgestellten Implementierungen aus [1], [2], [4] und [12] wurde ein Konzept einer verbesserten GLOSA in der Simulationsumgebung SUMO umgesetzt.

Die aufgezeigte Implementierung greift auf Daten des IaaS TTS zu. Anschließend werden die erhaltenen Topologien und Schaltzyklen in einer zusätzlichen API, beschrieben in Kapitel 2.2, aufbereitet und eine GLOSA basierend auf der GPS Position, der Geschwindigkeit und den Schaltzyklen berechnet. Im Anschluss werden die Daten für die Simulation bereitgestellt, sodass die simulierte Ampel entsprechend der Ampel der Realität schaltet. Zuletzt werden simulierte Fahrzeuge durch den Aufruf der API befähigt eine valide Geschwindigkeitsempfehlung zu erhalten. Auf dieser Basis wurde ein Konzept einer hybriden V2I Kommunikation unter Einfluss von V2V Kommunikation angereichert. Die Prinzipien der Schwarmintelligenz wurden durch Analyse der FCD der Fahrzeuge angewendet, wodurch Fahrzeuge befähigt wurden einen situationsangepassten, intelligenten Eingriff in das Verkehrsgeschehen vorzunehmen. Dazu wurde eine angepasste Geschwindigkeitsempfehlung innerhalb der Simulation definiert, die bereits ausgesprochene GLOSA Informationen mit den Erkenntnissen der V2V Kommunikation fusioniert. Aufgrund dieser Berechnungen sind die Fahrzeuge in der Lage bei einem beliebigen Verkehrsaufkommen der grüne Welle zu folgen.

Im Folgenden sollen Resultate der Simulation herausgearbeitet werden und in Hinblick auf die Forschungsfrage evaluiert werden. Darüber hinaus wird ein Fazit mit Rückblick auf die eingangs bearbeitete Studie gezogen. Zuletzt sollen weitere Ansätze zur Implementierung geklärt und ein Ausblick im Sinne zukünftiger Forschungsfragen und Forschungsansätze gegeben werden.

6.1 Resultate

Als geeignetes Mittel um die Simulation zu evaluieren dienen Liniendiagramme, die den Verlauf der Fahrgeschwindigkeit der Super-Fahrzeuge aufzeigen. Zu erwarten ist eine gleichbleibende Geschwindigkeit über den gesamten Zeitraum, in der die Fahrzeuge vor der Kreuzung verortet sind.

Das Liniendiagramm besteht aus einem Koordinatensystem, das auf der x-Achse den Zeitpunkt der Simulation enthält. Auf der y-Achse befindet sich die Geschwindigkeit des Fahrzeugs in $\mathrm{km\,h^{-1}}$.

Im Folgenden sollen nun die Diagramme aufgezeigt werden, die eine stufenweise Verbesserung der Fahrgeschwindigkeit anhand der implementierten Verfahren verdeutlichen.

Beginnend wird der Verlauf der Fahrgeschwindigkeit eines Fahrzeugs aufgezeigt, welches ohne Kommunikation mit der Umwelt auf die Kreuzung zufährt:

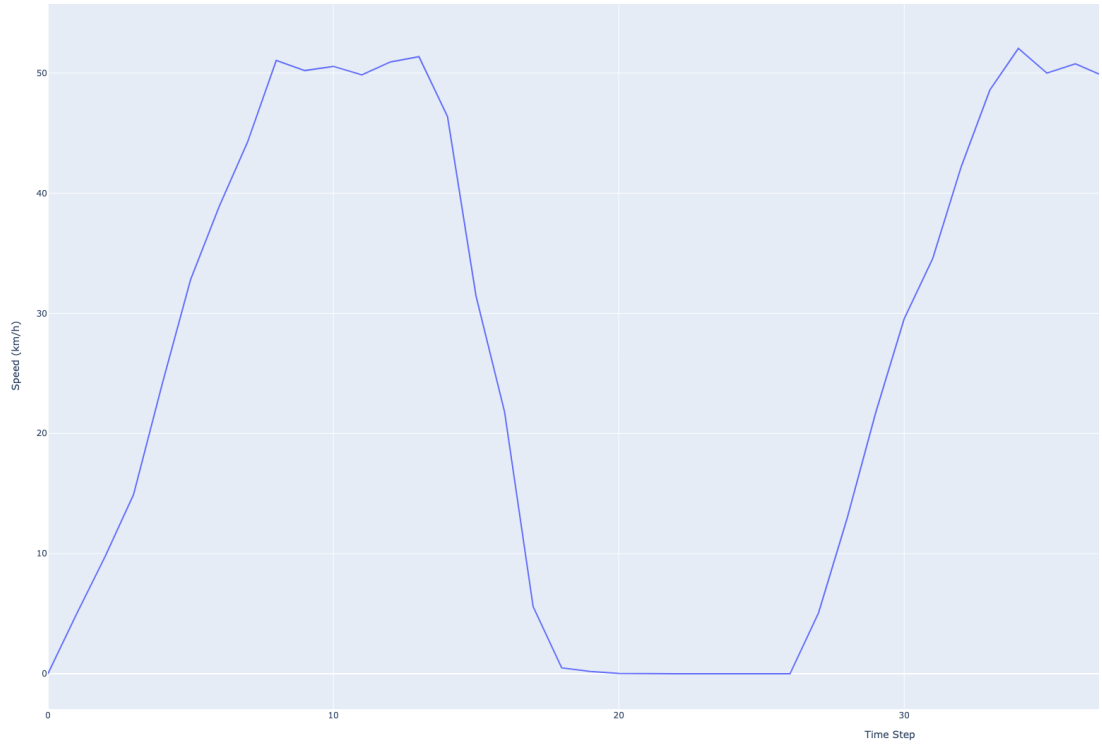


Abbildung 6.1: Verlauf der Fahrgeschwindigkeit eines Fahrzeugs ohne Kommunikation

Erkennbar ist, dass das Fahrzeug zu Beginn auf die zugelassene Höchstgeschwindigkeit der Straße beschleunigt. Im Anschluss wird es aufgrund der roten Ampel vollständig zum Stillstand gebracht. In Anbetracht dieses Umstandes differiert die Minimalgeschwindigkeit und Maximalgeschwindigkeit des Fahrzeugs stark. Bereits daraus lässt sich ein deutliches Verbesserungspotential erkennen, welches das zuvor definierte Forschungsziel stützt.

In Hinblick auf die Realisierung wurde zunächst die V2I Kommunikation implementiert. Für diesen Zweck wird ein Fahrzeug ohne Einbeziehung weiterer Verkehrsteilnehmer in Richtung der Kreuzung bewegt. Die erhaltenen Informationen der Kommunikation zwischen Fahrzeug und Ampel werden gemäß der Realisierung in Kapitel 5.2 dem Fahrzeug zugeführt und auf die Geschwindigkeit angewendet. Ein exemplarisches Diagramm der Fahrgeschwindigkeit zeigt folgendes Ergebnis auf:

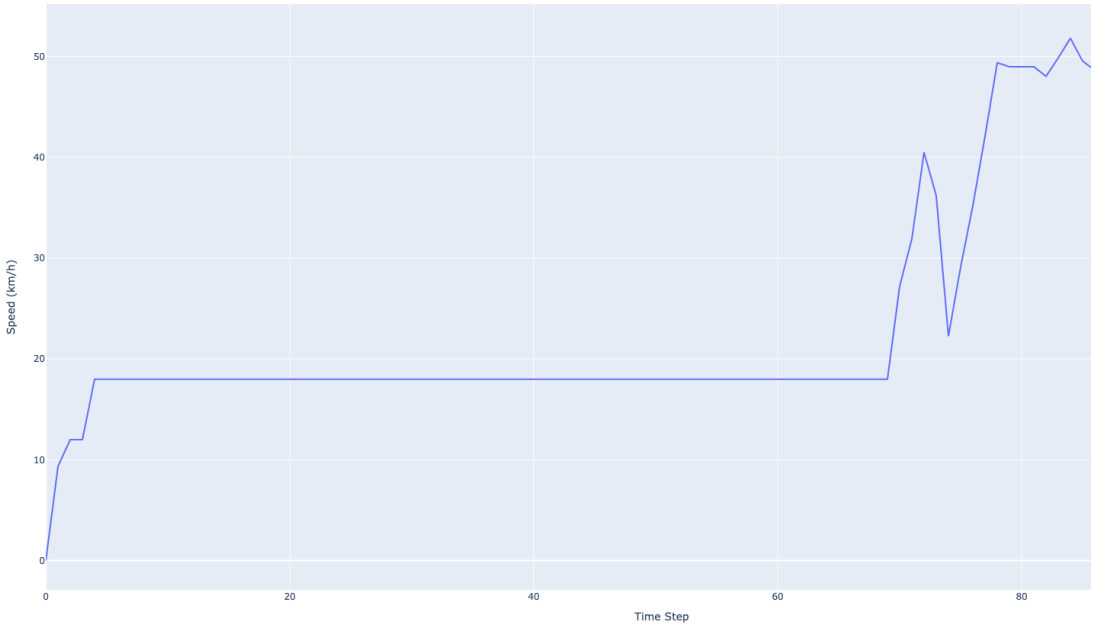


Abbildung 6.2: Verlauf der Fahrgeschwindigkeit eines Fahrzeugs mit GLOSA Informationen

Wie zu erkennen ist, wird die Fahrgeschwindigkeit auf die von der V2I Kommunikation resultierende Geschwindigkeitsempfehlung gesetzt. Mit dieser Geschwindigkeit nähert sich das Fahrzeug an die Ampel an. Durch die Geschwindigkeitsempfehlung ist das Fahrzeug in der Lage, die Kreuzung zu passieren ohne zum Stillstand kommen zu müssen. Das Verhalten des Fahrzeugs hinsichtlich seiner Geschwindigkeit im Zeitraum zwischen 70 und 80 ist der Tatsache geschuldet, dass die Fahrzeuge per se langsamer über die Kreuzung fahren, um Kollisionen zu vermeiden. In der Realität würden Fahrzeuge mit gleichbleibender Geschwindigkeit über die Kreuzung fahren, weshalb diese Veränderung der Geschwindigkeit unerheblich ist.

Vergleichbar ist diese Umsetzung mit den Erkenntnissen der Studie aus Kapitel 2.1, da dort bei schwachem Aufkommen anderer Fahrzeuge ein solches Resultat erzielt werden konnte.

Das Verfahren stößt an seine Grenzen sofern nun weitere Verkehrsteilnehmer eingebunden werden, da diese das Fahrzeug ausbremsen könnten und die Empfehlung somit belanglos werden würde. Diesem Zweck dient die V2V Kommunikation, da durch sie die ursprüngliche Geschwindigkeitsempfehlung positiv beeinflusst werden kann. Zunächst wird dafür der Einfluss des Signals *RED* analysiert:

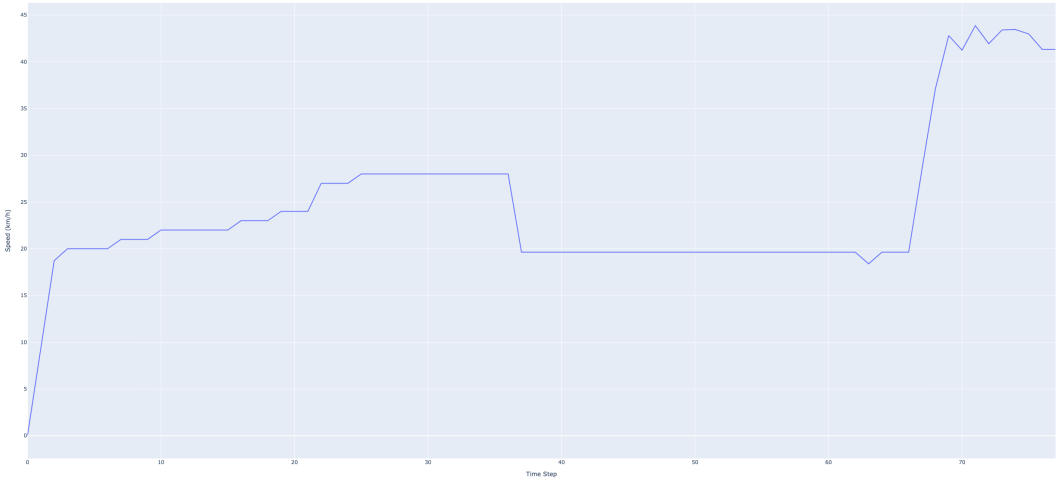


Abbildung 6.3: Einfluss des Signals *RED* auf die Fahrgeschwindigkeit

Erkennbar ist, dass das Fahrzeug einem Eingriff mit mehr Dynamik ausgesetzt ist. Zunächst wird das Fahrzeug mittels der GLOSA auf $27\,\mathrm{km\,h^{-1}}$ beschleunigt. Zum Zeitpunkt 37 trifft ein Signal eines haltenden Fahrzeugs ein, worauf das Fahrzeug seine Geschwindigkeit auf $20\,\mathrm{km\,h^{-1}}$ reduziert. Wie erkennbar ist, kann das Fahrzeug die Kreuzung durch diesen Umstand ohne weitere Anpassung der Fahrgeschwindigkeit passieren.

Für den vorliegenden Fall entspricht die Geschwindigkeit von $20\,\mathrm{km\,h^{-1}}$ der Maximalgeschwindigkeit, die das Fahrzeug fahren kann, ohne bremsen zu müssen. Sollten keine wartenden Fahrzeuge an der Ampel stehen, würde das Fahrzeug dennoch nicht die Höchstgeschwindigkeit fahren, da dies nicht im Sinne der Ziele der GLOSA nach Kapitel 2.1 wäre. Für diesen Fall besteht die Möglichkeit das Fahrzeug zu beschleunigen um dem dahinter liegenden Verkehr zu ermöglichen ebenfalls die grüne Welle zu befahren.

Analog zum Entschleunigen des Fahrzeugs mittels des Signals RED, kann das Fahrzeug durch Übersenden des Signals $MOVE$ beschleunigt werden, sofern dies möglich ist. Abzubilden ist dabei die Veränderung der Geschwindigkeit wie folgt:

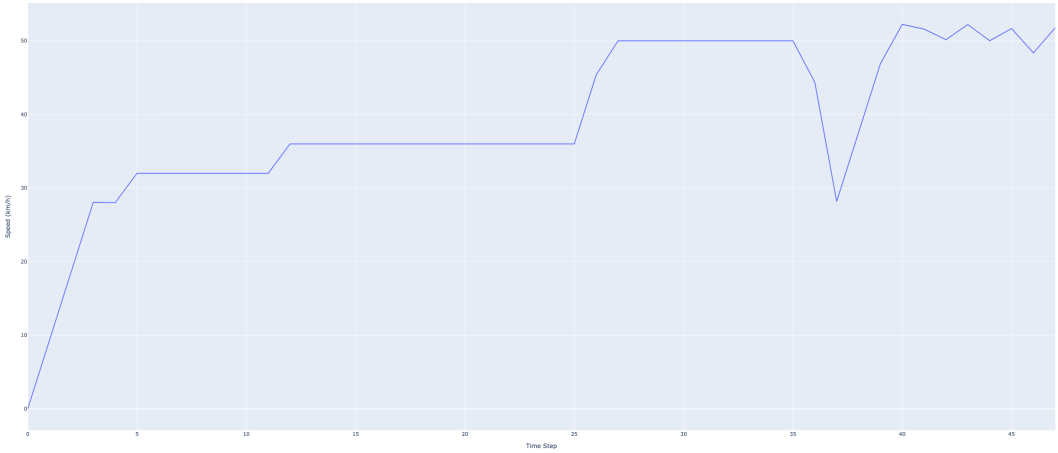


Abbildung 6.4: Einfluss des Signals $MOVE$ auf die Fahrgeschwindigkeit

Wie zu erkennen ist, enthält das Fahrzeug durch die V2I Kommunikation eine Geschwindigkeitsempfehlung, durch die es befähigt ist, die Kreuzung vor der Rot-Phase zu überqueren. Die übersendeten Signale führen zu einer Erhöhung der Geschwindigkeit, wodurch der dahinter liegende Verkehr ebenfalls in der Lage ist, die Kreuzung zu passieren. Evident ist dies an der Erhöhung der Fahrgeschwindigkeit in den Zeitbereichen drei bis fünf, zehn bis 12 und 25 bis 27. An diesen Stellen ist erkennbar, wie das Fahrzeug die Geschwindigkeit aufgrund der eintreffenden Signale anpasst, sodass die nachfolgenden Fahrzeuge ebenfalls über die grüne Ampel fahren können. Aus der Sicht der API hätte für die Überquerung der Ampel eine Geschwindigkeit von $27\,\mathrm{km\,h^{-1}}$ ausgereicht, aufgrund der Fahrzeuge hinter dem Super-Fahrzeug hat dieses seine Fahrgeschwindigkeit ungefähr verdoppelt. Der Einbruch des Graphen zwischen den Zeitpunkten 35 und 40 ist dabei erneut auf das zurückhaltende Verhalten der simulierten Fahrzeuge zurückzuführen. Lässt man dieses Verhalten außen vor, so kann ein optimaler Verkehrsfluss und eine positive Anreicherung der Fahrgeschwindigkeit durch das Zusammenspiel der V2I und V2V Kommunikation festgestellt werden. Ein Punkt der kritisiert werden muss ist, dass aufgrund der Erhöhung der Fahrgeschwindigkeit keine signifikante Einsparung von Kraftstoff oder CO_2-Emissionen möglich ist.

Ein Sonderfall kann entdeckt werden, wenn das Super-Fahrzeug bereits die maximale Geschwindigkeit - in Hinblick auf die Geschwindigkeitsempfehlung - einhält und vom nachfolgenden Verkehr die Aufforderung erhält die Geschwindigkeit zu erhöhen:

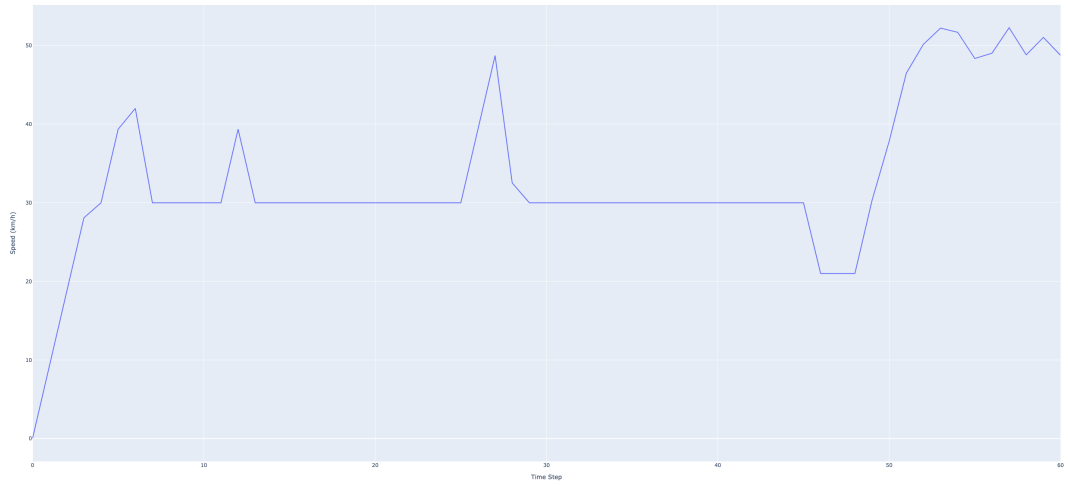


Abbildung 6.5: Grenzen der kollektiven Intelligenz

Aus diesem Graphen wird ersichtlich, wie das Fahrzeug aufgrund der Signale nachfolgender Verkehrsteilnehmer versucht, die Geschwindigkeit zu erhöhen. Gleichzeitig wird es aufgrund der Maximalgeschwindigkeit limitiert, wodurch die eintreffenden Signale kollidieren. Innerhalb der Simulation ist das Umsetzen der Geschwindigkeit trivial, weshalb dieses Verhalten eine gewinnbringende Visualisierung für diese Problematik liefert. Für die Realität sollte eine geeignete Lösung gefunden werden, welche im nachfolgenden Fazit aufgegriffen wird. Dennoch wird das Fahrzeug ungeachtet der Ausschläge in die Lage versetzt die Kreuzung innerhalb des Zeitabschnitts zu überqueren, ohne zum Stillstand zu kommen. Der Graph zeigt daher die Grenzen der Schwarmintelligenz auf, da trotz der Kommunikation keine höhere Fahrgeschwindigkeit erzielt werden kann, da das Fahrzeug andernfalls an der Kreuzung ankommen würde bevor die wartenden Fahrzeuge losgefahren sind.

6.2 Fazit

In Hinblick auf die Resultate aus der Simulation lässt sich ein Resümee hinsichtlich der Umsetzung ziehen. Die zugrundeliegenden Ergebnisse zeigen auf, wie eine schrittweise Verbesserung der Geschwindigkeit eines Fahrzeugs erzielt wurde. Ermöglicht wurde dies, indem zu jedem Zeitpunkt eine hohe Berechenbarkeit des Verkehrsaufkommens geschaffen wurde. Dem zugrunde liegt die V2V Kommunikation, die innerhalb der präsentierten Studie nicht zum tragen kam. Folglich kann eine Verbesserung im Verfahren zur Abschätzung von wartenden oder voraus fahrenden Fahrzeugen festgehalten werden.

Das Richten der Geschwindigkeitsempfehlung nach der aktuellen Fahrgeschwindigkeit des Fahrzeugs ermöglicht es außerdem, die zuvor geübte Kritik zu beseitigen die sich mit der Problematik befasste, dass Fahrer nicht bereit sind ihre Geschwindigkeit anzupassen, sofern eine drastische Änderung erforderlich ist. Durch die Beseitigung dieses erfolgskritischen Faktors kann die Funktionalität eines solchen Systems in der Realität grundsätzlich gewährleistet werden.

Weiterhin beschränkt sich das Verfahren nicht auf einzelne Fahrzeuge innerhalb des großen und komplexen Kontrukts des Verkehrs, vielmehr wurde versucht eine ganzheitliche Lösung für das Verkehrsaufkommen zu finden. Hieraus lässt sich schließen, dass ein Problem behandelt wird, welches auf einer höheren Abstraktionsebene verortet ist.

Durch die V2V Kommunikation sind die Fahrzeuge in die Lage versetzt worden ihre Intentionen und Anforderungen innerhalb des Kollektivs geltend zu machen. Evident hierfür sind die Signale RED und $MOVE$, da diese aufzeigen welchen positiven Einfluss eine V2V Kommunikation unter Verwendung einer V2I Kommunikation erzielen kann.

Die Fähigkeit Validierungen anhand der Ausführungen in 5.4 durchzuführen ist ein weiterer Punkt, der die Testbarkeit und Wartbarkeit eines Systems sicherstellt. Im vorliegenden Fall könnte dieses Verfahren genutzt werden, um Fehler einer realen Entwicklung für ein Fahrerassistenzsystem aufzudecken und anhand stets gleicher Bedingungen zu beseitigen. Ferner könnten daran auch neue Verfahren erprobt werden, wodurch dies eine sinnige Ergänzung im Sinne der Forschung darstellt.

Im Gegensatz zu den Erprobungen der Studie der City of Ottawa wurden die umgesetzten Verfahren lediglich in einer Simulation modelliert und dargestellt. Zwar könnte die Implementierung auf jedes Szenario in der ein Ampelsystem und entsprechende Fahrzeuge existieren angewendet werden, allerdings kann nicht sichergestellt werden, wie das System in der Realität überzeugen würde. Dies ist insbesondere dem Umstand geschuldet, dass die Simulation auf Fahrzeugen beruht, die stets ein optimales Fahrerprofil abbilden, somit also unverzüglich nach dem Schalten auf grün beschleunigen.

Darüber hinaus enthält die umgesetzte Geschwindigkeitsempfehlung keinen Faktor der die Dauer betitelt, die ein Fahrer benötigt ein spezielles Kommando umzusetzen. Auch die Tatsache, dass die Fahrzeuge die erhaltene Geschwindigkeitsempfehlung unmittelbar nach dem Erhalt umsetzen, bildet keinen realitätsnahen Ansatz.

In Hinblick auf die Ergänzung um V2V Kommunikation ist anzumerken, dass die ursprüngliche Berechnung der Geschwindigkeitsempfehlung der V2I Kommunikation dezentral innerhalb der API geschieht, wogegen die angereicherte Geschwindigkeitsempfehlung basierend auf den Daten anderer Verkehrsteilnehmer zentral innerhalb des Quellcodes der Steuerung der Simulation geschieht. Einzubringen ist daher, dass die Berechnung auf unterschiedlichen Ebenen mit potentiellen Risiken hinsichtlich inkonsistenter Daten verbunden sein kann. Eine Lösung könnte an dieser Stelle das Übertragen der gewonnenen Daten anderer Fahrzeuge an den V2I Dienst sein, wodurch eine Empfehlung basierend auf allen Erkenntnissen dezentral formuliert werden kann. Der V2I Dienst würde dann die Rolle eines ITS einnehmen, da die Fahrzeuge unmittelbare Befehle erhalten würden, die zusätzlich reflektiert auf Basis der gesamten FCD berechnet wurden. Eine Überlappung von Signalen, wie sie in Abbildung 6.5 aufgezeigt wurde, könnte über dieses Verfahren beseitigt werden.

6.3 Ausblick

Die Studie beweist, dass eine Geschwindigkeitsempfehlung basierend auf Daten der Infrastruktur formuliert werden kann. Die Kombination von V2V und V2I Kommunikation für diesen Anwendungsfall stellt eine neuartige Implementierung dar, die Raum in Hinblick auf weitere Forschungen bereithält. Denkbar wären insbesondere das zuvor angesprochene Auslagern der Kommunikation mittels eines dezentralen Dienstes.

Die Ursachen hierfür liegen neben einer geordneten Kommunikationsstrategie, auch in sicherheitsrelevanten Aspekten, die erforscht werden müssen. Die Möglichkeit performante Algorithmen zur Berechnung einer idealen Geschwindigkeitsempfehlung anzuwenden ist ebenfalls ein relevanter Faktor. Die allgegenwärtige Technologie des maschinellen Lernens könnte maßgeblich dazu beitragen anhand der Menge existierender FCD präzise Abschätzungen hinsichtlich des Verkehrsaufkommens zu treffen. Weiterhin wäre auch eine automatische Entwurfsstrategie nach Hamann [10] denkbar, wie sie in Kapitel 3.3 angerissen wurde. Ansätze zur Verwendung von Machine Learning innerhalb von Fahrzeugnetzwerken lassen sich bereits in [15, 16] finden.

Zuletzt muss evaluiert werden, ob diese Technologie auf traditionelle Kraftfahrzeuge angewendet werden kann, oder doch eine unterstützende Technologie für die immer größer werdende Debatte rund um das autonome Fahren sein kann.

Literaturverzeichnis

[1] „Final Report to Tranport Canada: I2V Connected Vehicle Pilot Project - City Fleet - Signalized Intersection Approach and Departure Optimization Application“, City of Ottawa, Tech. Bericht, Jan. 2020. Online verfügbar: https://tcdocs.ingeniumcanada.org/sites/default/files/2020-05/City%20of%20Ottawa%20-%20I2V%20Connected%20Vehicle%20Pilot%20Project%20-%20City%20Fleet%20-%20Signalized%20Intersection%20Approach%20and%20Departure%20Optimization%20Application.pdf (abgerufen: 04.12.2022).

[2] S. F. T. M'Sallem, „Implementierung eines Proof of Concepts zur Visualisierung von vehicle-to-infrastructure Kommunikation im Kontext von Ampelinformationen“, Technische Hochschule Mittelhessen, Gießen, Projektbericht, Dez. 2022.

[3] „Ampelinformationen online | Audi connect.“, Online verfügbar: https://www.audi.de/de/brand/de/service-zubehoer/connect/ampelinformation-online.html (abgerufen: 04.12.2022).

[4] R. Bodenheimer, D. Eckhoff, und R. German, „GLOSA for adaptive traffic lights: Methods and evaluation“, in *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, Okt. 2015, S. 320–328.

[5] T. Tielert, M. Killat, H. Hartenstein, R. Luz, S. Hausberger, und T. Benz, „The impact of traffic-light-to-vehicle communication on fuel consumption and emissions“, in *2010 Internet of Things (IOT)*, Nov. 2010, S. 1–8.

[6] D. Eckhoff, B. Halmos, und R. German, „Potentials and limitations of Green Light Optimal Speed Advisory systems“, in *2013 IEEE Vehicular Networking Conference*, Dez. 2013, S. 103–110, iSSN: 2157-9865.

[7] A.-K. Kraft, „Kooperation zwischen Verkehrsteilnehmern : Entwicklung und Evaluation von HMI-Konzepten zur Unterstützung kooperativen Fahrens“, Dissertation, Universität Ulm, Mai 2021, accepted: 2021-05-12T15:23:26Z ISBN: 9781757899840. Online verfügbar: https://oparu.uni-ulm.de/xmlui/handle/123456789/37391 (abgerufen: 12.01.2023).

[8] L. C. Ezenwa, „Drahtlose V2V-Kommunikation mit DSRC-Technik im Vergleich zu C-V2X“, *ATZ - Automobiltechnische Zeitschrift*, Vol. 124, Nr. 6, S. 42–45, Juni 2022. Online verfügbar: https://doi.org/10.1007/s35148-022-0837-0 (abgerufen: 23.02.2023).

[9] D. F. Llorca, M. A. Sotelo, S. Sánchez, M. Ocaña, J. M. Rodríguez-Ascariz, und M. A. García-Garrido, „Traffic Data Collection for Floating Car Data Enhancement in V2I Networks“, *EURASIP Journal on Advances in Signal Processing*, Vol. 2010, Nr. 1, S. 1–13, Dez. 2010, number: 1 Publisher: SpringerOpen. Online verfügbar: https://asp-eurasipjournals.springeropen.com/articles/10.1155/2010/719294 (abgerufen: 15.01.2023).

[10] H. Hamann, *Schwarmintelligenz*. Berlin, Heidelberg: Springer, 2019. Online verfügbar: http://link.springer.com/10.1007/978-3-662-58961-8 (abgerufen: 18.12.2022).

[11] L. J. Heinrich, R. Riedl, und D. Stelzer, *Informationsmanagement: Grundlagen, Aufgaben, Methoden*, 11. Aufl. Berlin [u.a.]: De Gruyter Oldenbourg, 2014.

[12] J. Miller, „Vehicle-to-vehicle-to-infrastructure (V2V2I) intelligent transportation system architecture“, in *2008 IEEE Intelligent Vehicles Symposium*, Juni 2008, S. 715–720, iSSN: 1931-0587.

[13] P. A. Lopez *et al.*, „Microscopic Traffic Simulation using SUMO“, in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, S. 2575–2582, iSSN: 2153-0017.

[14] A. Upadhyay, „Formula to Find Bearing or Heading angle between two points: Latitude Longitude -“, Apr. 2015. Online verfügbar: https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/ (abgerufen: 05.01.2023).

[15] M. J. Sataraddi und M. S. Kakkasageri, „Machine Learning based Vehicle-to-Infrastructure Communication in VANETs“, in *2021 IEEE 18th India Council International Conference (INDICON)*, Dez. 2021, S. 1–6, iSSN: 2325-9418.

[16] C.-C. Ho, B.-H. Huang, M.-T. Wu, und T.-Y. Wu, „Optimized Base Station Allocation for Platooning Vehicles Underway by Using Deep Learning Algorithm Based on 5G-V2X“, in *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, Okt. 2019, S. 1–2, iSSN: 2378-8143.

Abkürzungsverzeichnis

API	Application Programming Interface
DSRC	Dedicated Short-range Communication
FCD	Floating Car Data
GLOSA	Green Light Optimal Speed Advisory
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IaaS	information-as-a-service
ITS	Intelligent Transport System
OBD	On-Board-Diagnose
SIS	Shared Information Space
SUMO	Simulation of Urban Mobility
TraCI	Traffic Control Interface
TTG	Time-to-Green
TTS	Traffic Technologies Services
V2I	vehicle-to-infrastructure
V2V	vehicle-to-vehicle
V2X	vehicle-to-everything

Abbildungsverzeichnis

A Exemplarischer Aufruf der API

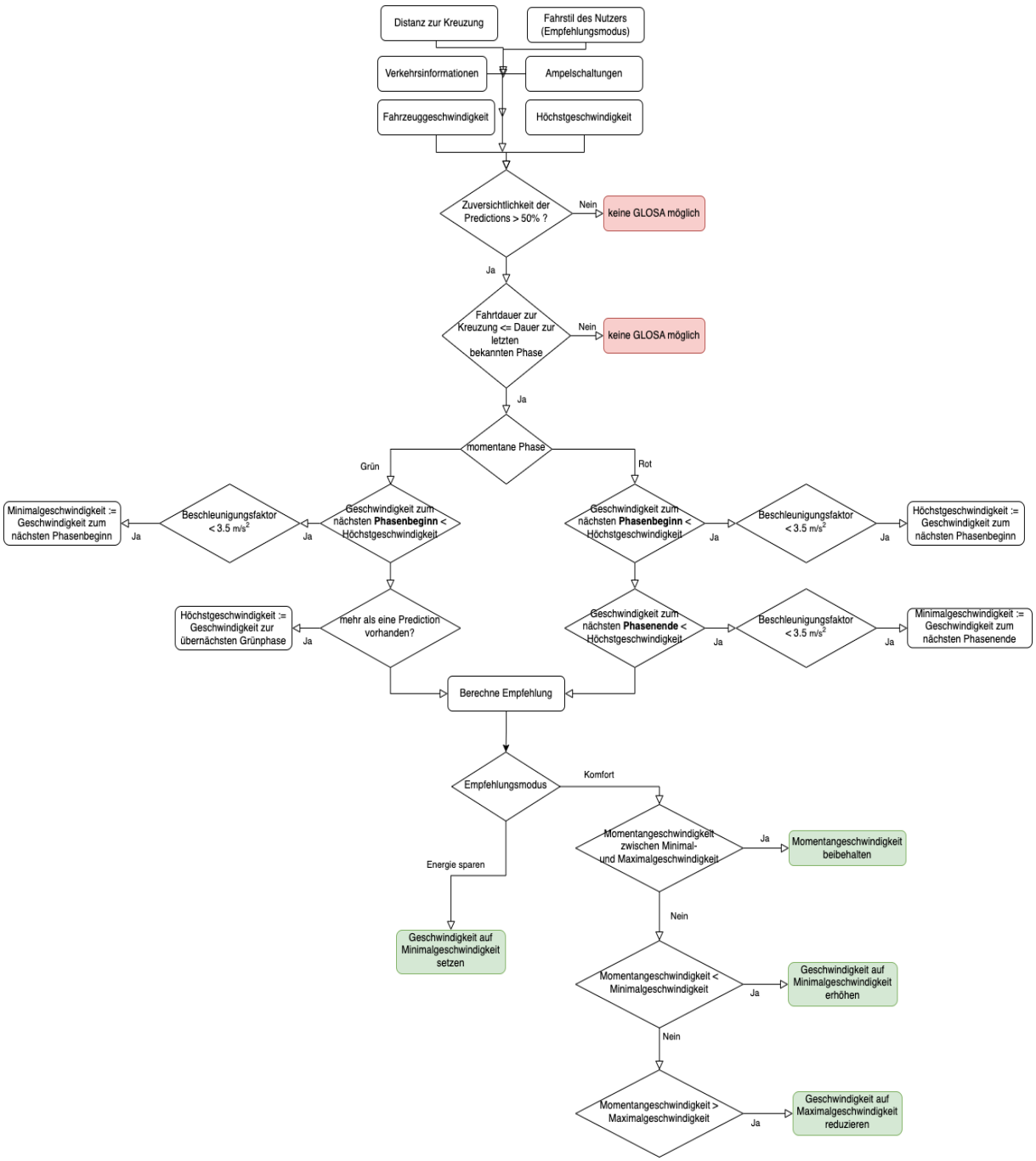
```
curl --location --request POST
    'http://tli-backend.azurewebsites.net/api/predictions/' \
--header 'Content-Type: application/json' \
--data-raw '{
    "latitude":48.726405691398355,
    "longitude":11.441749699789504,
    "direction":67,
    "currentSpeed": 30,
    "considerTraffic": false
}'
```

B Exemplarische Antwort der API

```
{
    "intersectionId": 1080, # id of the intersection
    "approachId": 3, # id of the approach at the intersection
    "name": "Schlosslaende @ Schutterstrasse",
    "distanceToIntersection": 0.0, # distance to the intersection approach (since the vehicle is inside it will return 0)
    "distanceToStopLine": 440.536200289113, # distance to the stop line (m)
    "signals": [ # array of signal heads present at an intersection approach
        {
            "signalId": 1, # signal head id, important if multiple signal heads exist for one approach
            "turnType": "Left_Straight", # turntype that indicates which turns the signal head controls
            "bulbColor": "Green", # current bulb color of the signal head
            "glosa": { # green light optimal speed advisory
                "justification": "[Green] -> Red",  # justification for the glosa algorithm. []: indicates the aimed phase at which the algorithm predicts the vehicle to cross the intersection
                "minimumSpeed": 41, # minimum speed to cross intersection before red phase
                "maximumSpeed": 50, # maximum speed to cross intersection (will be determined either by speed limit of the street and/or vehicles in front of the vehicle)
                "recommendation": 41, # recommended speed that the car should actually drive (between min- and max-speed, while always trying to be as near as possible to the vehicles current speed to avoid heavy breaking or accelerating)
                "speedLimit": 50 # speed limit on the street, delivered from google roads api
            },
            "predictions": [ # next signal phases on the approach
                {
                    "bulbColor": "Red", # next phase will be red
                    "timeToChange": 38, # time to change -> phase will start in 38s
```

```
23                      "confidence": 93 # confidence level (between 0-100%)
                            determines the assurance that this event will
                            take place at this timestep
24                  },
25                  {
26                      "bulbColor": "Green", # after next phase will be
                            green again
27                      "timeToChange": 76, # will start in 76s -> from this
                            one can ascertain that red phase will have a
                            duration of 38s, because it will start in 38s
                            and after another 38s (38 + 38 = 76) the green
                            phase will start again
28                      "confidence": 90 # confidence level is slightly
                            smaller than the previous one as the conditional
                            probability determines that also the first event
                            must take place
29                  }
30              ]
31          }
32      ]
33  }
```

C Beschreibung des GLOSA Algorithmus [2]



D Implementierung der GLOSA Berechnung [2]

```
1  package de.valtech.mobility.tli.service;
2
3  import de.valtech.mobility.tli.constants.DrivingMode;
4  import de.valtech.mobility.tli.helper.CollectionCheck;
5  import de.valtech.mobility.tli.model.Prediction;
6  import de.valtech.mobility.tli.model.SpeedAdvisory;
7  import de.valtech.mobility.tli.model.TrafficSignal;
8  import de.valtech.mobility.tli.model.pojo.maps.DistanceMatrixPojo;
9  import java.util.List;
10 import org.springframework.stereotype.Service;
11
12 @Service
13 public class GlosaService {
14
15     public SpeedAdvisory calcSpeedAdvisory(final TrafficSignal
           trafficSignal, final int distanceToStopLine, final int
           currentSpeed, final int speedLimit, final DistanceMatrixPojo
           trafficInformation, final DrivingMode drivingMode) {
16         final var predictions = trafficSignal.getPredictions();
17         final var currentBulbColor = trafficSignal.getBulbColor();
18
19         if (CollectionCheck.invalidData(predictions) ||
               predictions.stream()
20             .anyMatch(p -> p.getTimeToChange() == 0 ||
                   p.getBulbColor().isBlank() || p.getConfidence() == 0)) {
21             return null;
22         } else {
23             return computeBehaviour(currentSpeed, speedLimit,
                   predictions, currentBulbColor, distanceToStopLine,
                   trafficInformation, drivingMode);
24         }
25     }
26
27     private SpeedAdvisory computeBehaviour(final int currentSpeed, final
           int speedLimit, final List<Prediction> predictions, final String
           currentBulbColor, final double distanceToStopLine, final
           DistanceMatrixPojo trafficInformation, final DrivingMode
           drivingMode) {
28         final boolean hasMultiplePredictions = predictions.size() > 1;
29
30         final int timeToNextPhase = predictions.get(0).getTimeToChange();
```

```
31          final int speedToNextPhaseStart =
                speedByDistanceTime(distanceToStopLine, timeToNextPhase);
32          final boolean accelerationRecommendedToNextPhase =
                isAccelerationAchievable(accelerationFactor(speedToNextPhaseStart,
                currentSpeed, timeToNextPhase));
33
34          int timeToAfterNextPhase = 0;
35          int speedToAfterNextPhaseStart = 0;
36          boolean accelerationRecommendedToAfterNextPhase = false;
37
38          if (hasMultiplePredictions) {
39              if
                    (!predictionAdequate(conditionalProbability(predictions.get(0).getConfi
                    predictions.get(1).getConfidence()))) {
40                  return null;
41              }
42              timeToAfterNextPhase = predictions.get(1).getTimeToChange();
43              speedToAfterNextPhaseStart =
                    speedByDistanceTime(distanceToStopLine,
                    timeToAfterNextPhase);
44              accelerationRecommendedToAfterNextPhase =
                    isAccelerationAchievable(accelerationFactor(speedToAfterNextPhaseStart,
                    currentSpeed,
45                  timeToAfterNextPhase));
46          }
47
48          if (intersectionUnreachableInTime(trafficInformation,
                timeToNextPhase, timeToAfterNextPhase,
                hasMultiplePredictions)) {
49              return null;
50          }
51
52          String justification = "";
53          int minimumSpeed = 0;
54          int maximumSpeed = speedLimit;
55
56          if (currentBulbColor.equalsIgnoreCase("Red")) {
57              if (speedToNextPhaseStart <= speedLimit &&
                    accelerationRecommendedToNextPhase) {
58                  maximumSpeed = speedToNextPhaseStart;
59              }
60              if (hasMultiplePredictions && speedToAfterNextPhaseStart <=
                    speedLimit && accelerationRecommendedToAfterNextPhase) {
61                  minimumSpeed = speedToAfterNextPhaseStart;
62              }
63              justification = "Red -> [Green]";
64          } else {
65              if (speedToNextPhaseStart <= speedLimit &&
                    accelerationRecommendedToNextPhase) {
66                  minimumSpeed = speedToNextPhaseStart;
```

```
67                  justification = "[Green] -> Red";
68              } else if (hasMultiplePredictions) {
69                  maximumSpeed = speedToAfterNextPhaseStart;
70                  justification = "Green -> Red -> [Green]";
71              }
72          }
73
74          return new SpeedAdvisory(justification, minimumSpeed,
                maximumSpeed, speedRecommendation(maximumSpeed,
                minimumSpeed, currentSpeed, drivingMode),
75              speedLimit);
76      }
77
78      private boolean intersectionUnreachableInTime(final
            DistanceMatrixPojo trafficInformation, final int
            timeToNextPhase, final int timeToAfterNextPhase, final boolean
            hasMultiplePredictions) {
79          if (trafficInformation == null) {
80              return false;
81          }
82          final int durationInTraffic =
                trafficInformation.getRows().get(0).getElements().get(0).getDurationInTraff
83
84          return hasMultiplePredictions ? durationInTraffic >
                timeToAfterNextPhase : durationInTraffic > timeToNextPhase;
85      }
86
87      private int speedRecommendation(final int maxSpeed, final int
            minSpeed, final int currentSpeed, final DrivingMode drivingMode)
            {
88          if (drivingMode == DrivingMode.FUEL_SAVE) {
89              return minSpeed > 5 ? minSpeed : maxSpeed;
90          } else {
91              if (maxSpeed >= currentSpeed && minSpeed <= currentSpeed) {
                    // between min and max, so could remain same speed
92                  return currentSpeed;
93              } else if (minSpeed > currentSpeed) { // drive faster
94                  return minSpeed;
95              } else { // drive slower
96                  return maxSpeed;
97              }
98          }
99      }
100
101     private int speedByDistanceTime(final double distance, final double
            time) {
102         return (int) ((distance / time) * 3.6);
103     }
104
```

```
105     private int accelerationFactor(final int predictedSpeed, final int
            currentSpeed, final int time) {
106         return (predictedSpeed - currentSpeed) / time;
107     }
108
109     private boolean isAccelerationAchievable(final int acceleration) {
110         return Math.abs(acceleration) <= 3.5;
111     }
112
113     private double conditionalProbability(final double pA, final double
            pB) {
114         return pA * pB / 10000;
115     }
116
117     private boolean predictionAdequate(final double
            conditionalProbability) {
118         return conditionalProbability >= 0.5;
119     }
120
121 }
```

E Erlangung der Schaltzyklen aller Anfahrtsrichtungen

```
1  def get_approach_signal(approachId):
2      # Returns signals for the given approach
3
4      # Background of this function is that in order to control the
           traffic signal heads in the simulation like in real-life through
           the API,
5      # i need to obtain these signals and then pass them to the
           simulation handler
6      # Since i cannot just ask the API to provide signals for a given
           approach at an intersection, i have to request the signals
           through a point
7      # The points specified in the function calls below stand for a point
           at each approach which will then return signals for the
           requested approach
8
9      if approachId == 1:
10         logger.printlog("###### Approach 1 ######")
11         return extract_tli(client.perform_request(48.76280618764156,
               11.427623411273599, 105.4492514593))
12     elif approachId == 2:
13         logger.printlog("###### Approach 2 ######")
14         return extract_tli(client.perform_request(48.763513657462475,
               11.431514833553978, 235.909123032))
15     else:
16         logger.printlog("###### Approach 3 ######")
17         return extract_tli(client.perform_request(48.758733700993886,
               11.425519395220782, 40.2685517938))
```

F Implementierung der angepassten Geschwindigkeitsempfehlung

```
1  def improve_vehicle_speed(receiver, sender, distance_to_last_vehicle):
2      # Improves the vehicle speed of a glosa influenced vehicle based on
           received vehicle positions (based on v2v communication)
3      time, receiver, distance_to_intersection, glosa, signals =
           tli_store.read(receiver)
4      age = traci.simulation.getTime() - time
5
6      if get_minimum_speed(glosa) < get_recommended_speed(glosa):
7          if signals[0][0] == 'Red' and (signals[0][1] - age) >= 0:
8              ttg = signals[0][1] - age
9              if ttg == 0:
10                 print(f"Vehicle speed of {receiver} can remain same!")
11                 return ttg
12
13             vehicle_route = traci.vehicle.getRoute(receiver)
14
15             distance_to_vehicle = traci.vehicle.getLeader(receiver)
16             distance_to_last_vehicle = distance_to_vehicle[1] if
                   distance_to_vehicle != None and distance_to_vehicle[0]
                   == sender and distance_to_vehicle[1] <
                   distance_to_intersection else distance_to_last_vehicle
17             vehicle_delay = ((distance_to_intersection -
                   distance_to_last_vehicle) /
                   traci.vehicle.getLength(receiver))
18             green_end = signals[1][1] if len(signals) > 1 else 10
19             min_speed = distance_to_last_vehicle / (ttg + green_end)
20             max_speed = distance_to_last_vehicle / (ttg +
                   traci.edge.getLastStepVehicleNumber(vehicle_route[helper.get_approach_r
                   * 2)
21             if max_speed * 3.6 > 50:
22                 max_speed = 45 / 3.6
23             current_speed = traci.vehicle.getSpeed(receiver)
24             new_speed = 0
25
26             if min_speed < current_speed and current_speed < max_speed:
27                 new_speed = current_speed
28             elif min_speed > current_speed:
29                 new_speed = min_speed
```

```
30              elif current_speed > max_speed:
31                  new_speed = max_speed
32
33              traci.vehicle.setSpeed(receiver, new_speed)
34              if current_speed != new_speed:
35                  print(f"Changed vehicle speed of {receiver} from
                        {current_speed * 3.6} km/h to {new_speed * 3.6}
                        km/h")
36              else:
37                  print(f"Vehicle speed of {receiver} can remain same!")
38
39              influenced_vehicles.append([receiver, min_speed, max_speed])
40
41              return ttg
```