TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Bereich Ingenieurwissenschaften, Institut für Angewandte Informatik

Professur für Prozessmodellierung für vernetzte technische Systeme

Praktikumsprotokoll

Komplexpraktikum Networked Systems Modeling Wintersemester 2021/22

Florian Minnecker

24.01.2022

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Bereich Ingenieurwissenschaften, Institut für Angewandte Informatik

Professur für Prozessmodellierung für vernetzte technische Systeme

Zusammenfassung

Die ständig steigende Anzahl von Automobilen sowie deren damit einhergehende immer höhere Technologisierung bringen sowohl neue Probleme als auch verkehrsregelungstechnische Möglichkeiten mit sich. Gerade schwer einsehbare Kreuzungen, die trotzdem der Grundregel rechts vor links folgen, bergen ein Kollisionsrisiko. Zudem müssen Autos oft anhalten, um anderen Vorfahrt gewähren zu können. Das stört nicht nur den Verkehrsfluss, sondern birgt auch Risiken. Die vorliegende Arbeit beschäftigt sich mit der Erstellung und Implementierung eines Modells, um eine T-Kreuzung und sich auf dieser begegnende Autos simulieren zu können. Positionen und Geschwindigkeiten werden hierbei mittels Car-to-Car Kommunikation periodisch von jedem an andere Fahrzeuge übermittelt. Mithilfe dieser Daten sollen eigens implementierte Assistenzsysteme vor Kollisionen mit anderen Autos warnen und Unfälle auf verschiedene Weise verhindern. Final sollen die entstandenen Lösungen verglichen werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Vorbereitungen

Das gesamte Projekt wird mittels Instant Veins (vgl. Sommer/German/Dressler 2011: 3-15) - welches die Software OMNeT++, SUMO und TraCI sowie das Framework Veins enthält - umgesetzt. Es umfasst u.a. eine virtuelle Maschine auf Basis von Linux, Entwicklungsumgebungen für die Frameworks OMNeT++ und Veins sowie das Tool SUMO (vgl. Sommer 2021).
Während SUMO mithilfe von XML Dokumenten konfiguriert wird (vlg. Alvarez Lopez et al. 2018), ist das OMNet++ Modell in C++ implementiert, kann mit einer INI Datei angepasst konfiguriert und mittels NED Dateien (vgl. Varga 2010: 36) parametrisiert werden. Damit nehmen Instant Veins sowie der vorgegebene skeleton code die grundlegendsten, vor allem auf Betriebssystemebene administrativen, Konfigurationsmaßnahmen vorweg. Es entsteht ein untereinander verbundenes System zur Lösung der Aufgabe.

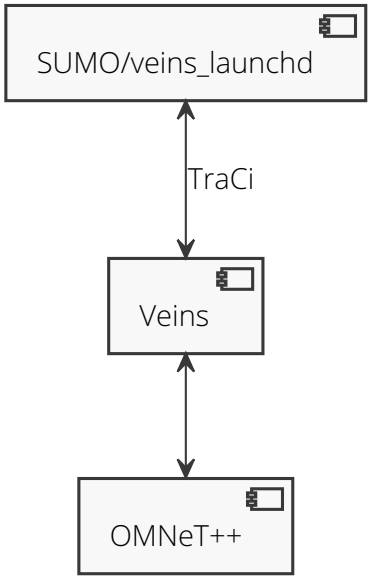

Abbildung 1.1.: Kopplung der Komponenten

1.1. Grundkonfiguration

Um Instant Veins mittels des freien Hypervisors QEMU als virtuelle Maschine auszuführen, musste das Festplattenabbild in ein unterstütztes Format konvertiert werden:

```
qemu-img convert instant-veins*{.vmdk,.qcow2} -O qcow2
```

Die Erstellung der Straßenkreuzung sowie die Grundkonfiguration des Netzwerkes wurde anschließend mithilfe von SUMO vorgenommen. Der zeitliche Abstand zwischen den losfahrenden Autos folgt einer stochastischen Verteilung. Durch das Importieren der entstandenen Konfigurationsdateien in das Projekt konnte mit der Implementierung begonnen werden.

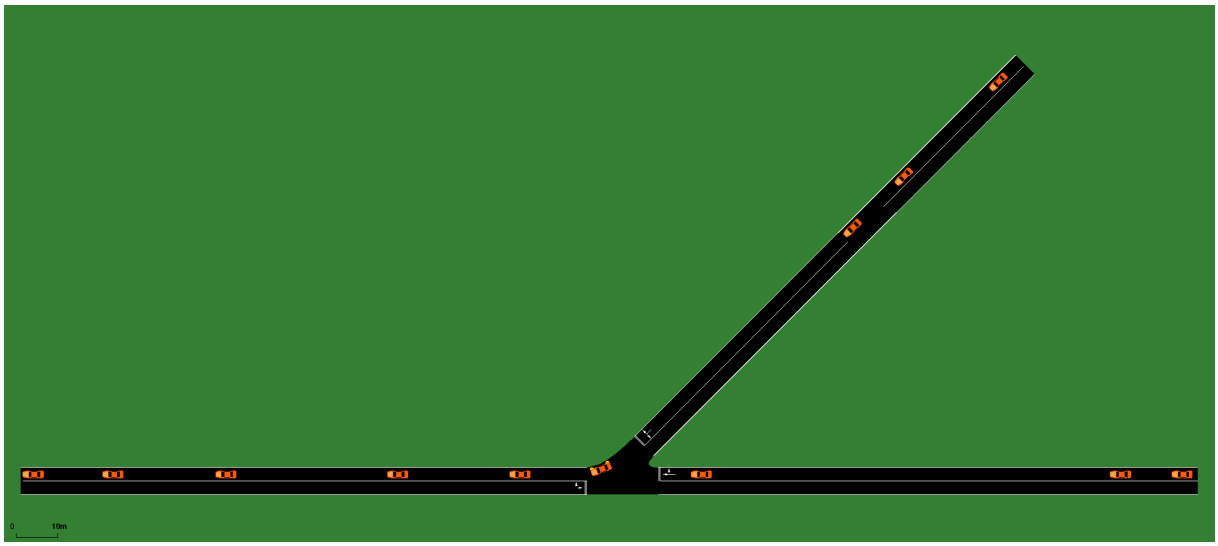

Abbildung 1.2.: SUMO Simulation der Kreuzung

1.2. Modellentscheidungen

Es wurde ein dynamisches Modell mit stochastischen Teilen entsprechend der Aufgabe, eine Kreuzung mit über die Zeit zufällig ankommenden Autos zu simulieren, geschaffen. Das Modell wurde vereinfacht, indem genau ein Szenario simuliert und stark abstrahiert worden ist (genaueres in Kapitel 4.2).

2. Implementierung

Die Implementierung wurde in C++ geschrieben. Es handelt sich um eine object-oriented and process oriented discrete-event simulation (DES). Es wurden die beiden grundlegenden Assistenzsysteme sowie Methoden zur Ermittlung und Verarbeitung der dafür benötigten Daten implementiert. Den Kern der Implementierung stellt die Klasse des InteractingVehicles dar, dessen Instanzobjekt jedes Auto ist (siehe Abbildung 2.1).

Abbildung 2.1.: Klassendiagramm des InteractingVehicles

Im Anschluss an die Implementierung des Modells wurde für die Auswertung der Simulation ein seperates Script erstellt (Kapitel 2.5).

2.1. Datenhaltung

Die Daten werden in den Instanzvariablen eines jeden InteractingVehicles gespeichert. Die grundlegende Datenstruktur kann der Abbildung 2.1 entnommen werden. Während die meisten Attribute dazu dienen, Historien zu speichern, die später für Berechnungen herangezogen werden können, bezwecken andere die Ermöglichung parametrisierter Simulationen. Hierbei werden die Werte per NED Konfigurationsdatei dynamisch gesetzt.

- psInterval gibt den zeitlichen Abstand zwischen zwei Broadcastnachrichten der Car2Car Kommunikation an
- breakDuration ist die Zeitspanne für kompletten Stillstand nach einer Gefahrenbremsung
- meetTimeWarnBefore, meetCollissionWarnBefore, meetBreakBefore sind zeitliche Abstände für die korrespondierenden Aktionen, wie das zeitliche Vorwarnen vor einem Unfall
- criticalMeetingDuration ist der zeitliche Abstand, den die Autos an demselben Ort haben müssen, um nicht zu kollidieren

Es werden außerdem weitere Werte gespeichert (u.a. durch Vererbung), welche der Auswertung dienen (siehe Kapitel 2.5).

2.2. Event Queue

Der in Veins implementierte Eventloop dient dazu, Eventobjekte, cMessages, zu verarbeiten. Diese können lokal oder über einen BUS (bspw. für Car2Car Kommunikation), mit und ohne Verzögerung der Simulationszeit, emittiert werden. Mithilfe des *kind* Attributes können verschiedene Instanzen untereinander unterschieden werden. Da DemoSafetyMessage eine solche Unterscheidung für einstellige Zahlen schon implementiert hat, wurde die Entscheidung getroffen, in dieser Implementierung von 999 für jeden Eintrag zu dekrementieren, sodass beide Implementierungen noch genügend Reserve behalten.

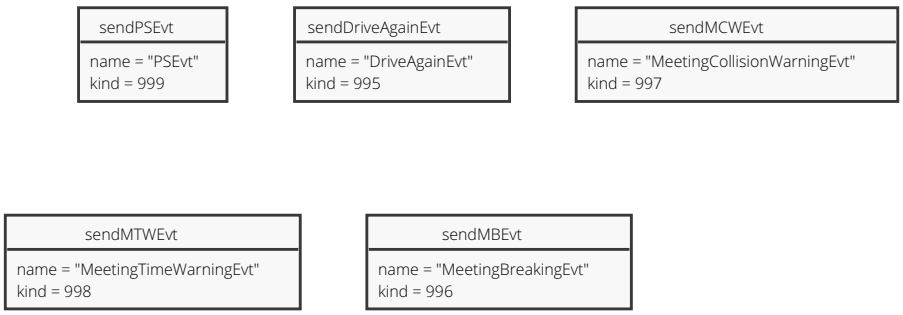

Abbildung 2.3.: Messages im Überblick

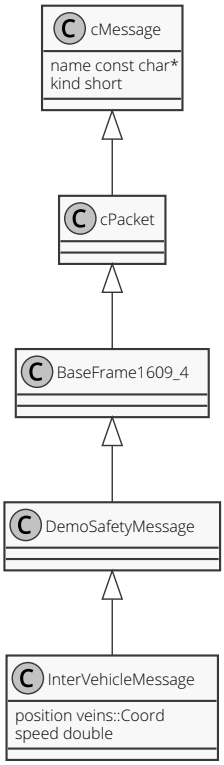

Abbildung 2.2.: Vererbung

Um über Nachrichten zusätzlichen payload versenden zu können - welches bspw. für Datenaustausch in Car2Car Netzwerken essentiell ist - können eigene Messageklassenobjekte definiert werden, die von cMessage erben. Veins beinhaltet einige solcher Klassen. Für die Implementierung der Car2Car Nachrichten wurde die Klasse InterVehicleMessage generiert (vgl. Varga 2010: 45).

Auch diese erbt, unter anderem, von cMessage (siehe Abbildung 2.4).

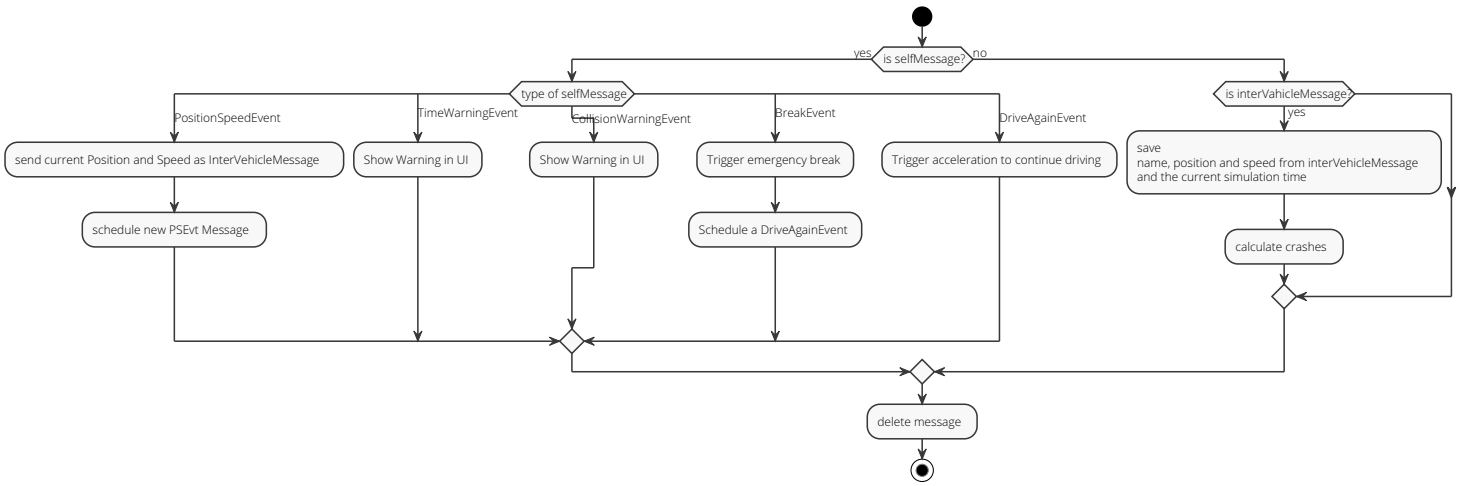

Abbildung 2.4.: Aktivitätsdiagramm Event Queue

Zusätzlich zu dieser Verarbeitungsvariante bietet Veins das Callback onPositionUpdate an. Bei jedem Update der Position des aktuellen Vehikels der Fahrzeugsimulation kann damit die aktuelle Position und die Simulationszeit ausgelesen und gespeichert werden.

2.3. Warn- und Bremssystem

Grundlegend muss bekannt sein, zu welcher Simulationszeit und an welchem Ort sich eine Kollision ereignen wird. Die Gleichungen hierfür wurden wie in Kapitel 3 beschrieben implementiert und werden immer berechnet, sobald eine Variable hinzukommt (per InterVehicleMessage oder onPositionUpdate, wie in Kapitel 2.2 beschrieben). Sobald eine Kollision ausgemacht wird, können die Events zum Warnen und Bremsen mit der errechneten Eintrittszeit - abzüglich der jeweilig konfigurierten Pufferzeiten - scheduled werden. Da mehrere Kollisionen nacheinander berechnet werden können, ist es wichtig, die früheste am höchsten zu priorisieren. Es werden jedoch alle berechneten Ereignisse seriell abgearbeitet und nach Eintritt gelöscht.

Das Auto kann dabei zwischen verschiedenen Zuständen wechseln, wie in Abbildung 2.5 dargestellt.

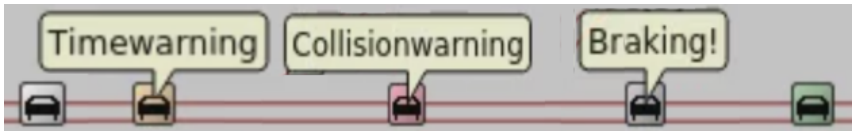

Abbildung 2.6.: Zustandsvisualisierung

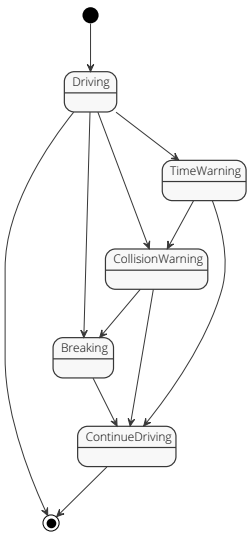

Abbildung 2.5.: Zustandsdiagramm

Zum Zwecke der Visualisierung verändern die Autos je nach Zustand die Farbe. Zusätzlich wird eine Sprechblase angezeigt, wenn Warnungen für Menschen in den Autos ausgegeben werden sollen.

2.4. Verzögerungssystem

Mithilfe von Anpassungen des Warn- und Bremssystems kann ein vollständiges Abbremsen verhindert werden. Hierzu wird nicht mit unveränderter Geschwindigkeit an die Kreuzung heran gefahren. Die Geschwindigkeit wird schon vorher vermindert, sodass die Kreuzung erst nach der errechneten Kollision erreicht wird. Bei jeder Neuberechnung des Kollisionspunktes wird die Geschwindigkeit zum Vermeiden der Kollision berechnet und somit die maximale Geschwindigkeit angepasst (vgl. hierzu auch Kapitel 3).
Die Visualisierung besteht aus einer Information über dem jeweiligen Icon.

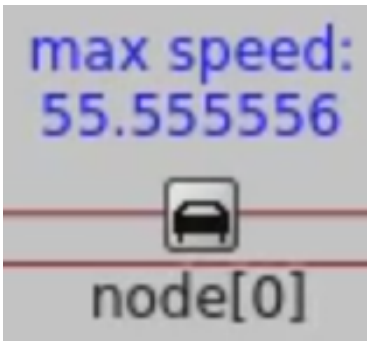

Abbildung 2.7.: Geschwindigkeitsvisualisierung

2.5. Auswertung

Für die Auswertung mussten keine weiteren Implementierungen im Modell vorgenommen werden (vlg. Varga 2010: 58 f.). Die für die Auswertung wichtigen Daten werden von der Basisklasse VeinsMobility bereitgestellt und von OMNeT++ - nach Setzen eines Konfigurationseintrags in der INI Konfiguration - in einer Datei gespeichert.

```
**.vector-recording = True
```

Diese können anschließend in das gängige Format CSV exportiert werden. Um einen Vergleich herzustellen, wurden mehrere Versuche unternommen. Diese untergliedern sich zum Einen in die Regelungen für die Kreuzung als Präfix des Versuchsnamens:

- right_before_left: Es gilt die Regel rechts vor links.
- braking: Das Bremsassistenzsystem ist als einziges aktiv.
- braking: Die Assistenzsysteme zum Bremsen und Geschwindigkeitsanpassen sind aktiv.

Als Suffix dient eine jeweilig parametrisierte Anpassung an die Menge der Autos, welche die Kreuzung passieren.

- normal: Eine normale Menge an Autos passiert die Kreuzung.
- double: Die doppelte Anzahl der normalen Menge an Autos passiert die Kreuzung.

- half: Die halbe Anzahl der normalen Menge an Autos passiert die Kreuzung

Anschließend konnte mithilfe von Python und den Frameworks panda, numpy und matplotlib die Auswertung innerhalb eines Jupyter Notebooks vorgenommen werden (siehe A.1).

3. Gleichungen

Die, für die bearbeiteten Aufgaben, wichtigsten Gleichungen:
Warn- und Bremssystem Kapitel 3.1 und 3.2
Verzögerungssystem Kapitel 3.1, 3.2 und 3.3

3.1. Schnittpunkt

Der Schnittpunkt zweier Vehikel liegt in der Ebene genau da, wo sich die Strecken der beiden schneiden. Repräsentiert werden Punkte durch die Koordinaten in einem Dreier Tupel (x, y, z), da der Simulationsraum durch ein kartesisches Koordinatensystem dargestellt wird.

$$\forall \vec{r} = P_1 - P_0, \vec{r} = Q_1 - Q_0 : P_1 = P_0 + \vec{r},\ Q_1 = Q_0 + \vec{s} \tag{3.1}$$

$$\Rightarrow \forall t, u \in [0, 1] :\ P_0 + t \cdot \vec{r} = S,\ Q + u \cdot \vec{s} = S \tag{3.2}$$

$$S = P_x = Q_y \Rightarrow I = Q + u \cdot \vec{s} = P + t \cdot \vec{r} \tag{3.3}$$

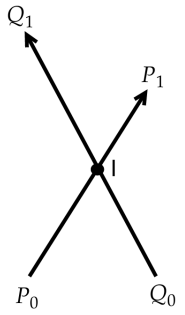

Abbildung 3.1.: Schnittpunkt

Gleichung (3.1) gibt die grundlegenden Voraussetzungen an - P_0 zu P_1 für die erste, Q_0 zu Q_1 für die zweite Strecke. Mit der Gleichung (3.1) kann jeder beliebige Punkt auf der jeweiligen Strecke repräsentiert werden, dementsprechend auch der gesuchte Schnittpunkt S. Da S auf beiden Geraden liegen muss, gilt Gleichung (3.3). Es wird konstatiert, dass sobald u oder t bekannt sind, der Schnittpunkt berechnet werden kann.
Für diese Berechnungen wird $\times$ als Kreuzprodukt definiert.

$$P + t \cdot \vec{r} = Q + u \cdot \vec{s} \qquad | \times s \tag{3.4}$$

$$\Leftrightarrow (P + t \cdot \vec{r}) \times s = (Q + u \cdot \vec{s}) \times s \tag{3.5}$$

$$\Leftrightarrow (P \times s) + (t \cdot \vec{r}) \times s = (Q \times s) + (u \times s \cdot \vec{s} \times s) \tag{3.6}$$

$$\Rightarrow (P \times s) + (t \cdot \vec{r}) \times s = Q \times s \qquad | - (P \times s) \tag{3.7}$$

$$\Leftrightarrow (t \cdot \vec{r}) \times s = Q \times s - P \times s \tag{3.8}$$

$$\Leftrightarrow t = (Q - P) \times \frac{s}{r \times s} \tag{3.9}$$

Dass $s \times s = 0$ gilt, kann in Gleichung (3.4) genutzt werden, sodass das Auflösen (Gleichungen (3.5) und (3.6)) zum vollständigem Wegfall von $u \cdot \vec{s}$ führt. Durch Äquivalenzumformungen (Gleichungen (3.7), (3.8), (3.9)) kann t berechnet werden. Analog könnte für u mit $\times r$ verfahren werden.
Der Schnittpunkt ist durch ein Einsetzen von t oder u in die Gleichung (3.3) gefunden (vgl. Rees 2009).

3.2. Zeitdifferenz bis zum Erreichen eines Punktes

Die Zeitdifferenz wird unter der Annahme einer konstanten Beschleunigung errechnet. Mithilfe von Umformungen der physikalischen Formel $s(t) = \frac{a}{2}t^2 + v_0 t + s_0$ kann anstatt des Weges die Zeit $t(s)$ repräsentiert werden (vgl. Strommer 2021):

$$t(s) = -\frac{v}{a} + \sqrt{\frac{v^2}{a^2} + \frac{2s}{a}} \tag{3.10}$$

Für die Beschleunigung a können wir, mithilfe des Vektors der gefahrenen Strecke und der Zeitabstände zwischen den Messwerten der Punkte, die Formel (3.11) anwenden:

$$a(t) = \frac{s}{t^2} \tag{3.11}$$

3.3. Berechnung der Verzögerungsgeschwindigkeit

Mithilfe des Schnittpunktes S und der Zeitdifferenz t aus den Kapiteln 3.1 und 3.2 kann die notwendige maximale Geschwindigkeit v berechnet werden, um die Begegnung zu vermeiden. s ist die Distanz zum Schnittpunkt S, welche mithilfe der Vektorlänge berechnet werden kann, bspw. mit dem Vektor $\vec{r} = P_1 - P_0$ aus Kapitel 3.1:

$$s = |(r)| \tag{3.12}$$

$$v = \frac{s}{t} \tag{3.13}$$

4. Diskussion

Das umgesetzte Modell bildet eine T-Kreuzung mit geraden Strecken ab. Die Nutzung von sowohl validierten als auch verifizierten Modellen bspw. für die Car2Car Kommunikation bietet eine gute, vertrauenswürdige Basis (vgl. Sommer 2020). Hierdurch wird die Auftrittsmöglichkeit der Fehler nahezu vollständig auf den in dieser Arbeit entwickelten Code beschränkt.

4.1. Vergleich zwischen den Straßenregelungssystemen

Es wurden Parameterstudien für eine Simulationsdauer von 3600 Sekunden durchgeführt und aus den gesammelten Daten Graphen generiert (vgl. hierzu Kapitel 2.5).
Die Graphen 4.1 und 4.2 zeigen die durchschnittlichen Geschwindigkeiten der Autos für jeden Versuch. Daran lässt sich klar erkennen, dass die höchsten Geschwindigkeiten mithilfe der automatisierten Gefahrenbremsmethode erreicht wurden. Anders als bei der Geschwindigkeitsanpassung müssen hier nur die von rechts kommenden Autos bremsen, statt dass jeweils der als letztes an der Kreuzung ankommende die Geschwindigkeit verzögern muss. Anhand der Grafik 4.3 zeigt sich, dass auch für die absolute Anzahl an abgefertigten Autos das Bremsassistenzsystem das am besten geeignetste ist. Gleichzeitig wird deutlich, dass bei normalem Verkehrsaufkommen ein Stillstand der Autos am besten durch das Assistenzsystem für Geschwindigkeitsanpassung vermieden werden kann. Man kann anhand des Graphen 4.4 feststellen, dass es vor allem beim Bremsassistenzsystem für einzelne Autos zu erheblich langen Wartezeiten kommen kann, während die meisten die Kreuzungen schnell passieren können. Anhand 4.5 erkennt man außerdem eine leichte Verbesserung der Gesamtzeit für alle Autos, eine größere für einzelne und eine insgesamt kompaktere Formation.
Bei einem Vergleich der Grafiken 4.1 und 4.4 fällt auf, dass einzelne Autos bis zum Ende stehen bleiben mussten und die Kreuzung so gar nicht passieren konnten. Die in Abbildung 4.3 gezeigten Werte für avoidance_double sind somit verfälscht und dienen nur als Anhaltspunkt.

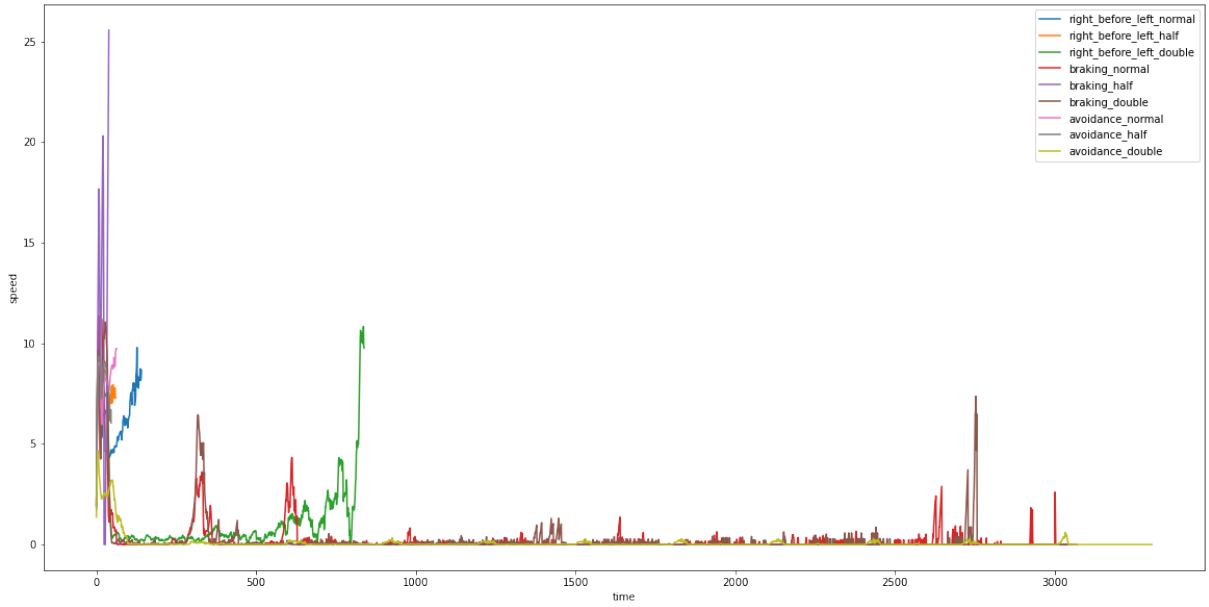

Abbildung 4.1.: Durchschnittsgeschwindigkeit (Lineplot)

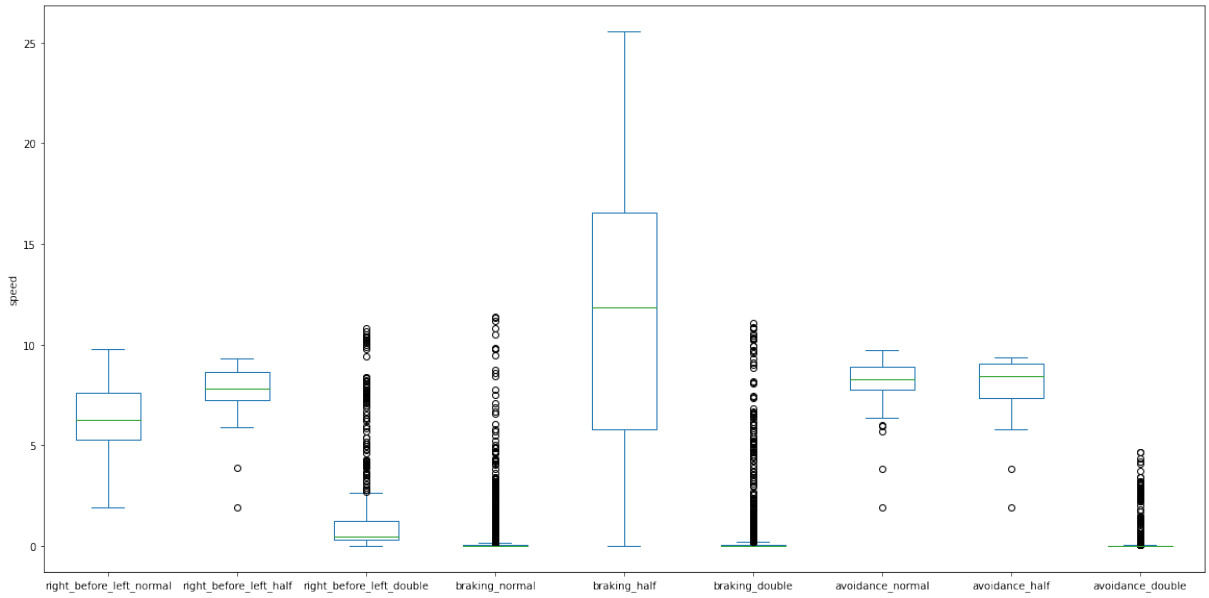

Abbildung 4.2.: Durchschnittsgeschwindigkeit (Barplot)

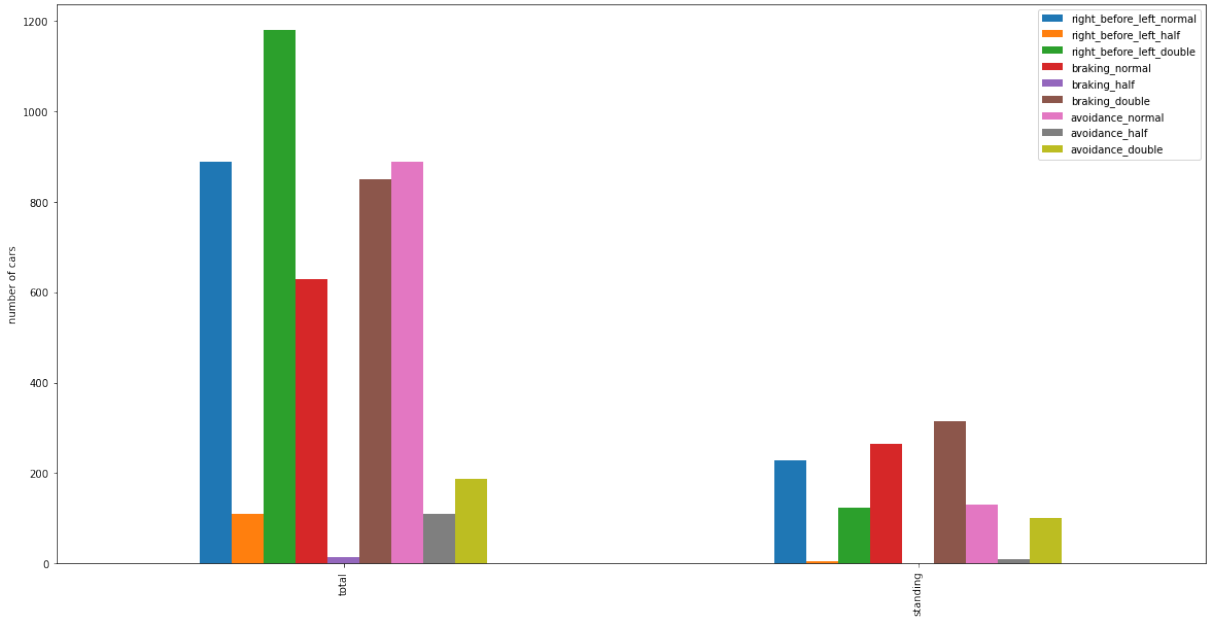

Abbildung 4.3.: Gesamtanzahl vs. stillstehende Autos

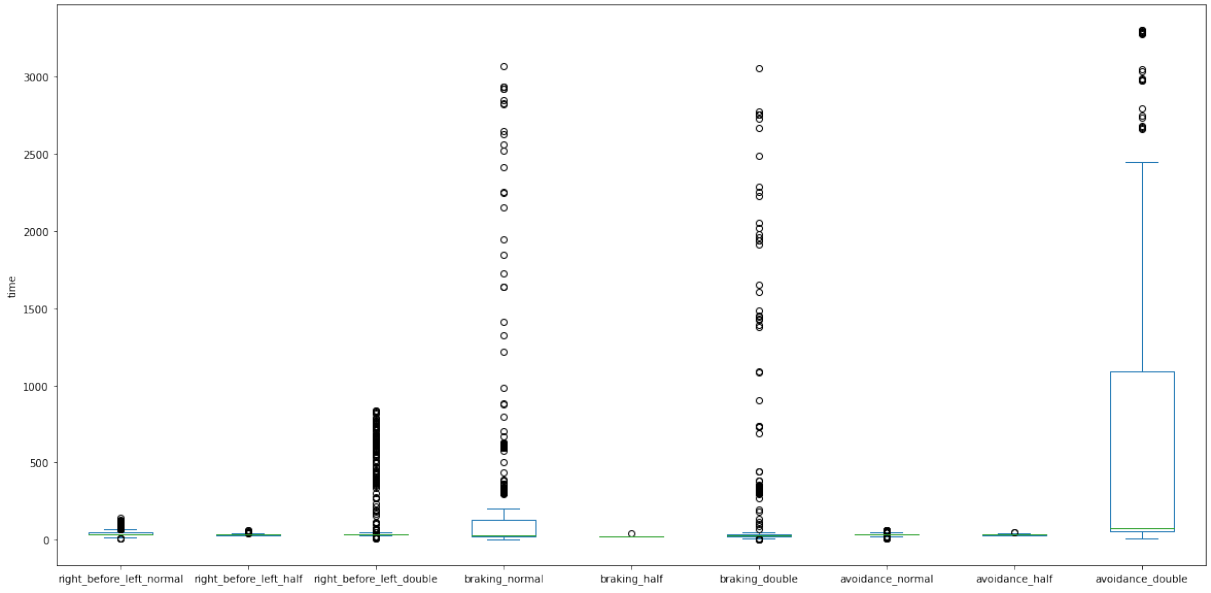

Abbildung 4.4.: Gesamtzeit der Autos für die Strecke

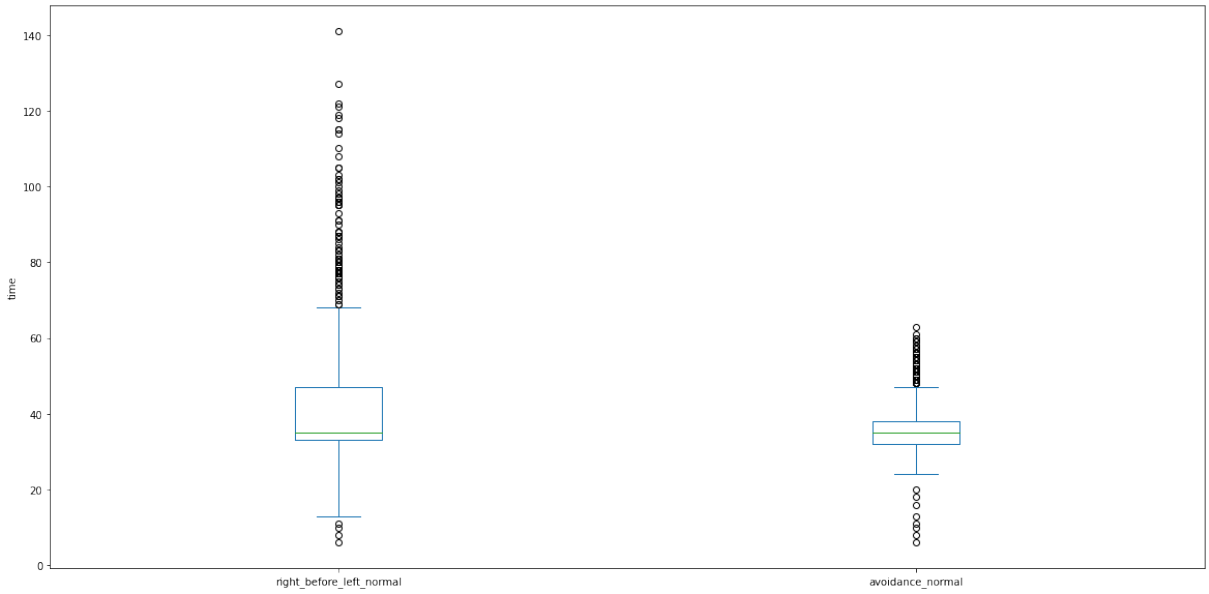

Abbildung 4.5.: Gesamtzeit der Autos für die Strecke (Auswahl)

4.2. Validierung

Da im Vorfeld keine Daten für das Modell gesammelt werden mussten, entfällt die Validierung von Eingangsdaten (data validation). Es wurde ein Modell nach Aufgabenstellung implementiert, sodass die notwendigen Abläufe anhand von typischen Merkmalen modelliert werden konnten. Die konzeptionelle Validierung (conceptual model validation) ist daher nur in begrenztem Umfang möglich. So wurden für das Modell Simplifizierungen vorgenommen, welche auf der Aufgabenstellung beruhen und nicht dem realen System entsprechen:

- Es wurden ausschließlich Autos betrachtet. Öffentliche Verkehrsmittel, Fahrradfahrer, Fußgänger, u.a. wurden nicht einbezogen.
- Alle Autos kommunizieren miteinander (Car2Car Kommunikation), Autos ohne diese Funktion (bspw. ältere Autos) sind nicht Teil des Modells.
- Die Straßen sind exakt linear, es gibt weder Kurven noch Berge oder Täler.
- Es wurden keinerlei Umwelteinflüsse betrachtet wie bspw. verschiedene Wetterlagen oder Hindernisse auf den Fahrbahnen.
- Es gibt keine Störfaktoren für die Kommunikation (Häuser, Umwelteinflüsse, ...).
- Alle Autos haben dieselben Eigenschaften, z.B. bei Beschleunigung, der Kurvengeschwindigkeit und des Bremsweges.
- Die Art der Kreuzung (T-Kreuzung) wurde absolut festgelegt, abweichende Kreuzungsmodelle sind nicht Teil des Modells.
- Fehlfunktionen von Systemen (auch mechanischer Systeme wie Bremsen oder menschlicher wie ärztliche Notfälle) wurden nicht betrachtet.
- Die Simulation beinhaltet nur Rechtsverkehr auf einspurigen Straßen.
- Der Einfluss von Gegenverkehr auf die Simulation wurde nicht berücksichtigt.

Für die operationale Validität des Modells (operation validity) wurde multistate validation angewendet. Durchgängige Visualisierung, sowie die face validity Methode (umgesetzt durch regelmäßige Präsentationen vor Kursleitern und anderen Studierenden), wurden während des Projektes durchgeführt. Durch die Parametrisierung der Simulation konnte außerdem parameter variability/sensivity analysis durchgeführt werden.

4.3. Verifikation

Die Verifikation wurde hauptsächlich durch vorhergehende Erfahrungen in der Programmierung und der Programmiersprache durchgeführt. Periodisch wurde der Code Third Partys zur Verfügung gestellt, jedoch ohne explizites Ziel der Verifikation. Weitergehend wurde Debuggen genutzt, um auftretende Fehler zu finden und zu beheben. Tests und Korrektheitsbeweise waren nicht Teil dieser Arbeit, entsprechend gibt es hier Verbesserungspotential.

4.4. Fazit

Die Arbeit zeigt grundlegend zwei Verwertungsmöglichkeiten der Car2Car Kommunikation auf. Das Bremssystem funktioniert sehr gut. False Positives sind hierbei hinnehmbar und könnten mithilfe von menschlichem Eingreifen eliminiert werden. Dass das Geschwindigkeitsanpassungssysteme für größere Verkehrsaufkommen versagt und eine Zwangsbremsung notwendig ist, kann Anreize geben, diese Methodik weiter zu verbessern und sie mit anderen Systemen, wie bspw. mit intelligenten Ampel- oder Verkehrsleitsystemen, zu koppeln. Die größte Schwachstelle der Simulation stellt der Fakt dar, dass die Daten des realen Systems nur mithilfe einer theoretischen Distribution in das Modell eingeflossen sind sowie die in Kapitel 4.2 aufgezeigten Simplifizierungen. Hierdurch, und durch die geringe sicherheitsrelevante Eignung durch die im Kapitel 4.3 genannten Implikationen, kann das Modell nur sehr bedingt auf reale Systeme übertragen werden. Abschließend lässt sich sagen, dass die implementierten Hilfssysteme den Straßenverkehr in Zukunft sicherer und gleichzeitig reibungsloser gestalten könnten.

5. Literaturverzeichnis

- Alvarez Lopez, Pablo/Behrisch, Michael/Bieker-Walz, Laura/ Erdmann,Jakob/Flötteröd, Yun-Pang/Hilbrich, Robert/Lücken, Leonhard/Rummel, Johannes/Wagner, Peter/Wießner, Evamarie (2018): Microscopic Traffic Simulation using SUMO. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), S. 2575-2582.
- Rees, Gareth (2009): How do you detect where two line segments intersect?, URL: https://stackoverflow.com/a/565282, eingesehen am: 18.01.2022.
- Sommer, Christoph/German, Reihard/Dressler, Falko (2011): Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis, IEEE Transactions on Mibile Computing (TMC), vol. 10 (1), Seite 3-15, URL: https://veins.car2x.org/, eingesehen am: 18.01.2022.
- Sommer, Christoph (2021): Instant Veins Virtual Machine. Components, URL: https://veins.car2x.org/documentation/instant-veins/#: :text=Components, eingesehen am: 18.01.2022.
- Sommer, Christoph (2020): Instant Veins Virtual Machine. Features, URL: https://veins.car2x.org/features/, eingesehen am: 18.01.2022.
- Strommer, Johannes (2021): Formeln für Geschwindigkeit, Beschleunigung, Weg & Zeit. Formeln bei gleichmäßiger Beschleunigung – Anfangsgeschwindigkeit $\neq 0$, URL: https://www.johannes-strommer.com/formeln/weg-geschwindigkeit-beschleunigung-zeit, eingesehen am: 18.01.2022.
- Varga Andras (2010): OMNeT++. In: Wehrle, Klaus/Güneş, Mesut/Gross, James (Hrsg.): Modeling and Tools for Network Simulation. Springer: Berlin, Heidelberg, S. 35-59, URL: https://doi.org/10.1007/978-3-642-12331-3_3, eingesehen am: 10.01.2022.

A. Anhang

A.1. IPython Notebook

0.1 Preconditions:

Install requirements, set up functions

[ ]:
```
from decimal import Decimal
try:
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
except:
    !pip install pandas numpy matplotlib scipy pivottablejs
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

%matplotlib inline
plt.rcParams["figure.figsize"] = [20, 10]

# some code from https://docs.omnetpp.org/tutorials/pandas/#2-setting-up
def parse_if_number(s):
    try: return float(s)
    except: return True if s=="true" else False if s=="false" else s if s else␣
 ↪None

def parse_ndarray(s):
    return np.fromstring(s, sep=' ') if s else None

# code of my own:

# returns a tuple with speed dataframe and time array
def getSpeedArrAndMaxTime(file):
    csv = pd.read_csv(file, converters = {
        'attrvalue': parse_if_number,
        'binedges': parse_ndarray,
        'binvalues': parse_ndarray,
        'vectime': parse_ndarray,
        'vecvalue': parse_ndarray})

    scalars = csv[(csv.name == 'speed') & (csv.module.astype(str).str.
 ↪startswith('Scenario.node[')) & (csv.module.astype(str).str.endswith('].
 ↪veinsmobility'))]

    speedarr = []
    timearr = []
    for index, row in scalars.iterrows():
        time = row['vectime']
        value = row['vecvalue']
```

```
        data = {}
        r = time[0] # let all graphs start on 0
        if(len(time) != len(value)):
            print("nomething is wrong, we time and value len don't match!");

        for s in range(min(len(time), len(value))): # normally, no difference␣
 ↪should occour
            data[time[s] - r] = value[s];

        timearr.append(time[ len(time)-1 ] - r)
        speedarr.append(pd.Series(data));

    return (speedarr, timearr)

### Takes a Series array and returns a concated series
def getConcatedDS(arr):
    series = None
    for s in arr:
        series = pd.concat((series, s))
    return series


### Takes a Series array and returns a dict with min and max
def getMinMax(arr):
    min_speed = Decimal('Infinity') # todo: how can i set this to inf?
    max_speed = -1

    for s in arr:
        nmin = s.agg('min')
        nmax = s.agg('max')
        if(nmin < min_speed):
            min_speed = nmin
        if(max_speed < nmax):
            max_speed = nmax

    return {"min": min_speed, "max": max_speed}


### takes a concated Series and return a goruped series
def getMeanSeries(concatedSeries):
    by_row_index = concatedSeries.groupby(concatedSeries.index)
    df_means = by_row_index.mean()
    return df_means

# returns num of cars wich totally stopped inside a array of dataframes
def getStandingCars(arr):
    standing_cars = 0
```

```
    for s in arr:
        if(0 == s.agg('min')):
            standing_cars += 1
    return standing_cars

speeds = {}
times = {}
for variant in ["right_before_left", "braking", "avoidance"]:
    for style in ["normal", "half", "double"]:
        (speedArr, timeArr) = getSpeedArrAndMaxTime("input/"+variant+"/"+style+".
 ↪csv")
        speeds[variant + "_" + style] = speedArr
        times[variant + "_" + style] = timeArr
```

0.1.1 All in one:

The following graph shall give an overview over the different speeds:

[ ]:
```
dfCompArr = {}
extremeCompArr = {}
standStillCompArr = {}
maxTimeComptArr = {}

for v in speeds:
    extremeCompArr[v] = {"min": getMinMax(speeds[v])['min'], "max":␣
 ↪{getMinMax(speeds[v])['max']}, "mean": getConcatedDS(speeds[v]).mean()}
    dfCompArr[v] = getMeanSeries(getConcatedDS(speeds[v]))
    standStillCompArr[v] = {"total": len(speeds[v]), "standing":␣
 ↪getStandingCars(speeds[v])}

df = pd.DataFrame(dfCompArr)
df.plot.line(xlabel="time", ylabel="speed")
df.plot.box(xlabel="time", ylabel="speed")

pd.DataFrame(standStillCompArr).plot.bar(ylabel="number of cars")
```

0.1.2 Maximum amount of time a car needs to cross the intersection

[ ]:
```
df = pd.DataFrame(dict([ (k,pd.Series(v)) for k,v in times.items() ]))
display(df.plot.box(ylabel="time"))

# only list right_before_left_normal, avoidance_normal
atimes = {}
for i in ['right_before_left_normal', 'avoidance_normal']:
    atimes[i] = times[i]

df = pd.DataFrame(dict([ (k,pd.Series(v)) for k,v in atimes.items() ]))
```

```
display(df.plot.box(ylabel="time"))
```

[ ]: