Creation of a modifiable Traffic Simulator

Daniel Hirsch, Jiekai Jia, Mengnan Jiang, Yiran Shu

***Abstract*— Traffic simulators are a widely used utility technology for research. It can be used in various fields such as road design and traffic dispatching. It is often necessary to evaluate the behaviour of different car motion models in different traffic scenarios. For researchers, it is therefore practical to have a tool at hand to freely modify (1.) the street network they are working with and (2.) the car motion models. A traffic simulation framework is proposed to allow users to freely implement different car motion models and street networks with little to no limitations. The simulation framework is implemented for evaluation in Python.**

I. Introduction

For solving traffic problems, traffic simulator tools are developed for planning streets and dispatching traffic flow. In this project, we are proposing a simulation framework that is able to flexibly simulate different motion models together with different street network scenarios. The goal is to have a framework that allows users and developers to freely extend its functionality. The street networks shall be designed in a way that is non-restrictive and lays as few assumptions as possible. The same applies to the traffic motion models, where this project aims to find a method to implement and simulate different models flexibly.

To verify the functionality of our simulator, we use a common motion model, called the Intelligent Driver Model (abbreviated as IDM). IDM is an intelligent microscopic traffic flow model, that each vehicle can change its state of motion according to its conditions and the state of the environment.

Here, we have to state again, that this simulator is not limited to just IDM motion models. We aim to derive a simulation approach that allows any arbitrary motion model to implement.

Also, we need to emphasize again, that this simulator is not limited to just the street network that will be introduced in this project report. The flexible simulation framework we propose allows a user to set up his own, independent street network which can represent all relevant traffic scenarios for simulation.

- merge scenarios
- T-crossing
- X-crossing
- car-following scenarios

The above stated individual scenarios can, of course, also be combined into more complex scenarios.

The result of this project will be a useful simulator with a user-interface and several simulation options and graphical visualization support.

This project was supported by Karsten Kreutz, M.Sc. .

To showcase an example simulation situation, the main interface of the traffic simulator is illustrated in figure 1. Cars enter the main streets from two entrances (black arrows on the left side), and the entrance below has a merge street. And a tunnel section (dark gray street), that cars can pass through, is added to the main streets. Eventually, cars will leave by the exit (red arrow on the right side). Besides, as a test platform, the scenario of the simulator is also highly extensible and can be easily modified, such that users can quickly modify the streets as needed.

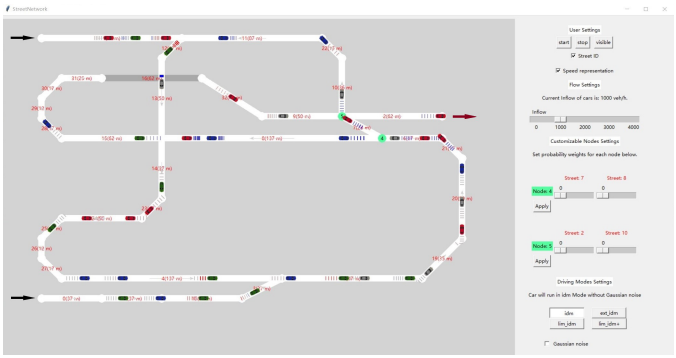


Fig. 1. Main interface of traffic simulator

The simulator should record and store the real-time movement information of each car, that users can analyze the behaviours of different models quantitatively.

With real-time simulation the simulator can control the cars of the street in three aspects: controlling the current flow (Flow settings), changing the possibility of cars entering different streets (Customizable Nodes Settings) and changing different motion models (Driving Modes Settings). By controlling the current traffic flow users can test the operation of different motion models under different traffic conditions intuitively. With different possibilities of cars enter different streets users can test the effectiveness of different motion models on road in complex scenarios (e.g., a road with high traffic flow). The function of switching the car dynamics helps the user to compare different models easily and quickly.

The simulator is based on Python and achieved all of the proposed functions. The operating interface is implemented through Tkinter, a module for window design of python. The program is written, running and debugged through Pycharm.

This article will further cover these topics:
II) State of the Art
III) Our Simulation Framework
IV) Traffic Dynamics Models
V) Results and Evaluation
VI) Further Ideas and Outlook

"State of the Art" informs about the status of the existing simulators and explain why a new simulator is needed. The section "Our Simulation Framework" will cover Simulation Principles, Software Architecture, and Additional Features. The detailed explanations of how motion models work are in the part of the "Traffic Dynamics Models" section. The part "Results/Evaluation" section shows the performance of the simulator and will analyze the movement status of the car. However, there are still some limitations and some points to be improved, which can be found in "Further Ideas and Outlook".

II. State of the art

In this section, the advantages and disadvantages of existing simulators are analyzed and the necessity of developing new simulators is explained.

A. Previously existing traffic simulators

1) PTV VISSIM: PTV VISSIM [1] is a microscopic traffic flow simulation software package developed by PTV AG in Karlsruhe, Germany. It is an effective tool for evaluating traffic design and urban planning, users are able to analyze traffic conditions under various conditions. An example of its 3-dimensional simulation perspective is showcased in figure 2.

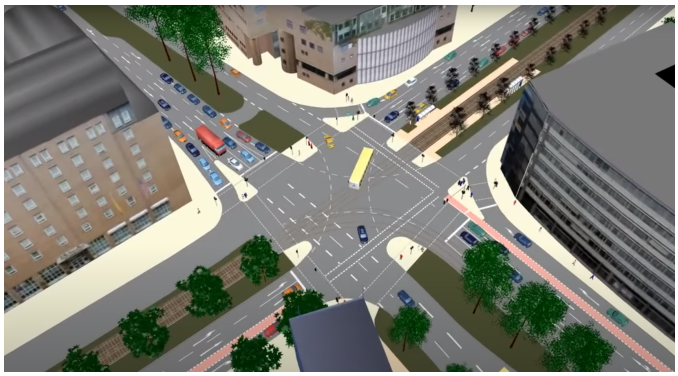


Fig. 2. PTV VISSIM provides a 3D simulation view [2]

But PTV VISSIM cannot save the complete movement data. For the information about the state of car movement, it can only be captured through detectors. So the requirement of recording the movement state in real-time cannot be fulfilled. There is no doubt that the subsequent analysis operations become very limited. However, the biggest limitation is that this software is currently unable to use different IDM versions and motion models in general. Therefore it's not suitable to be a test platform for traffic dynamics.

Additionally, it is closed-source commercial software which hinders research progress and makes implementing very custom needs for simulation very difficult and costly.

2) SUMO: Simulation of Urban MObility (SUMO) [3] is a traffic simulation package based on Eclipse technology, developed by the German Aerospace Center (DLR). It allows for intermodal simulation with a large set of tools for scenario creation. SUMO is used to evaluate traffic control measures, such as new traffic light systems, and to make short-term traffic forecasts under heavy traffic. SUMO is able to use the TraCI (Traffic Control Interface) interface to implement IDM development in Python language, while PTV VISSIM cannot develop IDMs.

The SUMO simulation user interface is showcased in figure 3.

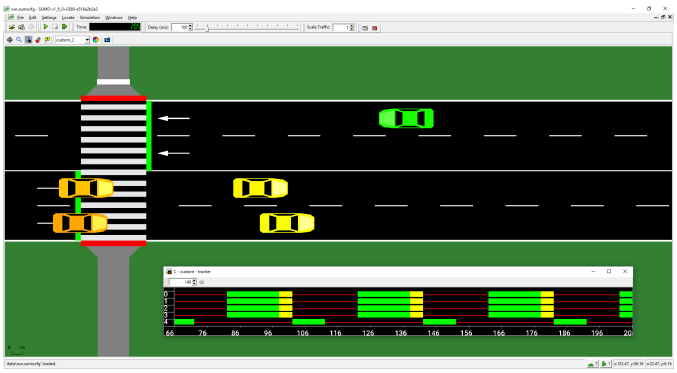


Fig. 3. SUMO is an opensource simulator developed by DLR - image from SUMO homepage [3]

Advantages of SUMO include its open-source nature [4] which allows users to obtain full access to the project implementation code. It conclusively also has a large user base in the research domain and currently has 47 active contributors continuously developing the code.

However, like VISSIM, users can only obtain incomplete car movement information through the detector. It is very inconvenient to analyze and evaluate IDM operation results quantitatively. Also, it is not (conveniently) possible to implement arbitrary motion models, which is why it is not suitable for our purposes.

To implement changes in SUMO, one has to grasp the concepts of a very big project, fork off the project online and implement the changes on his own without knowing the side effects. It is desirable to have a simulation framework that allows users to make changes in enclosed scopes. This means that we aim to develop a simulator where the motion model is implemented in just one place and the entire street network is modelled by just one script with clear boundaries.

This is not guaranteed by third-party projects.

3) traffic-simulation.de: Another popular traffic simulator is the client-side web software [5]. Understanding traffic phenomena by means of simple simulation scenarios is the main purpose of this simulator. Users are allowed to control traffic and create a traffic jam, to find out how to trigger and resolve a traffic breakdown. The code is open-source [6] and therefore available for anyone to modify. A demonstration is depicted in figure 4.

This simulator uses IDM, but users are only allowed to set the initial speed and acceleration parameters of IDM. There are only several simple fixed scenarios for simulation available. It is not possible to insert an arbitrary (general) motion model that is not IDM. The user cannot set the desired scenarios and it is not possible to propose a custom street scenario from the users perspective. Besides, it cannot store the car's movement data. Therefore, this project is not suitable for testing motion models in general.

Fig. 4. The client-side simulation performed on traffic-simulation.de

B. Development of the new Simulator

The above various projects are mainly used to simulate various traffic systems and evaluate the traffic system, while what we need is a testing platform for different motion models without any restrictions for the user. Any street network scenario and any motion model shall be possible to implement. Additionally, the basic requirements for recording complete car movement data cannot be fulfilled. In response to the above problems, the new simulator has been designed and many additional features have been added.

This project aims to propose a fully modifiable traffic simulator, where the street network and the motion models can be set arbitrarily complex.

1) Driving modes settings: An important feature of the new simulator is Driving modes settings. Users are allowed to select different motion models and Gaussian noise in the user interface, more instructions are in Appendix I-F. In addition, more motion models and different parameters can also be easily added and changed in the code.

2) Qualitative analysis tool: In order to show the movement state of the car more intuitively, the new simulator has added a new feature called speed representation(speed bar). It reflects not only the speed of the car but also shows the acceleration through the color of the bar. This feature is designed to make a qualitative analysis tool, which allows users to observe the movement of the car intuitively. It will be explained in part III-G.3.

3) Quantitative analysis tool: The logger feature can record the position, speed and acceleration of each car at each time step, and record it in '.log' file and '.txt' file. After the simulation, the logger provides users with accurate motion data for analysis. This feature makes up for the disadvantage of the previous simulators that cannot record complete data. More detailed instructions are in Appendix IV.

III. Our Simulation Framework

In this section, the main concepts of our approach to traffic simulation are explained.

A. Problem of street network modelling

One of the main problems of creating a flexible traffic simulator is to design the underlying street network. The simulator shall be modifiable. In a best-case scenario, any real street network constellation should be possible to simulate.

However, one can not cover all cases of street network situations by hard coding them. Therefore, the idea arises to create reusable components of street constellations. Possible components could be

- X-crossing
- T-crossing
- merge
- straight street
- curved street

Connecting the components to each other, one can design various street networks. However, this approach does not cover all situations. How, for example, is it possible to simulate an X-crossing where one of the streets has two lanes?

This illuminates the fact that a modularized simulation approach is versatile, but not universal. Therefore, we have investigated a more basic approach.

B. The street-node concept

Instead of following a modularized approach where one has ready-to-use building blocks, one could also use the building blocks of the building blocks. Specifically: Which fundamental building blocks does every street scenario consist of?

Inspired by graph theory, the hypothesis can be established that any street network is a directed graph and can accordingly be modelled from undirectional streets (edges) and nodes.

A node is an abstract entity that connects two streets. Real-world streets do not consist of nodes. They are merely a tool for building connections between streets.

A node can have multiple incoming and outgoing streets.

C. Modelling approaches

An example of this is illustrated in figure 5. Here, a two-lane T-crossing is modelled using only unidirectional streets (direction indicated by the arrows) and nodes (highlighted in green).

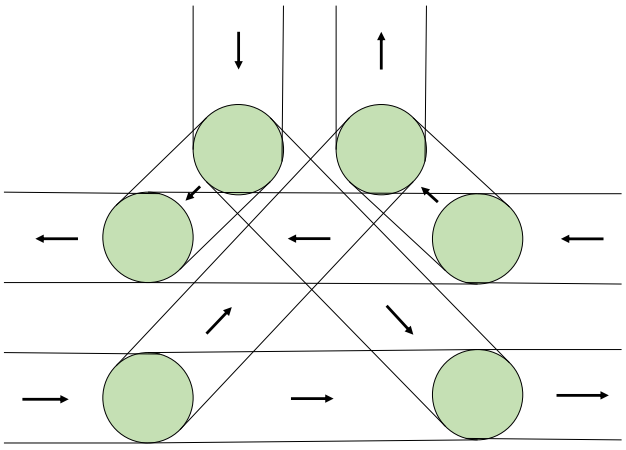


Fig. 5. Modelling a two-lane T-crossing with the street-node concept

This T-crossing is not a fully functional model of the real world. A car crossing from the lower-left street into the upper T-branch will enter the (diagonal) street that overlaps the opposing street coming from the right. Collisions will not be detected in this setting, because cars only consider other cars on streets that are connected by nodes. Overlapping streets ignore each other.

This problem is inherent to the street-node concept and can be fixed by cleverly setting the nodes. While the T-crossing model in figure 5 more accurately resembles the movement of the car, the model illustrated in figure 6 is better suited for simulation. Each node is placed in a way that allows the detection of opposing cars.

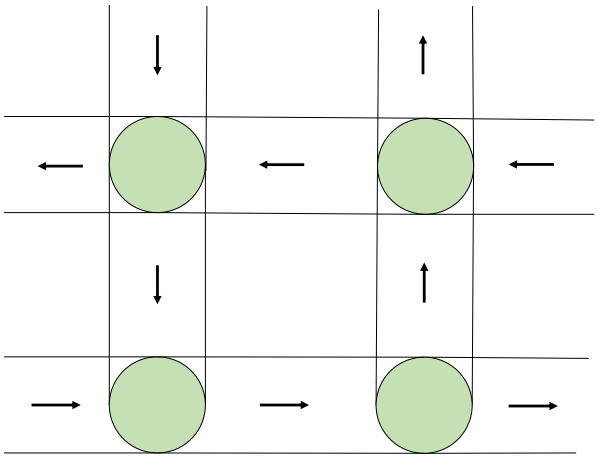


Fig. 6. A collision-free T-crossing

This modelling approach is, however, also not perfect since cars will be able to randomly keep driving on the center loop in 6.

It seems that there is always a payoff in modelling real-world scenarios. While one approach causes undetected collisions, the other approach causes unexpected car paths. The two-lane T-crossing illustrates this problem very well and shows that a simulator aiming for more complex simulations will have to add additional restrictions (or rules) for car paths.

For the purpose and complexity of the street network, we are aiming to implement in this project, the unrestricted street-node concept is sufficient.

By establishing a few nodes and the street connections between the nodes, a versatile approach to modelling street networks is established.

D. Traffic Simulation

So far, the modelling of the street network scenario was discussed. The next step is to look at how actual vehicles are simulated to drive the street network. The technical implementation details are avoided in this section, but some general solutions for common problems are explained here. The following few paragraphs will cover the occurring questions for traffic simulation and our solutions.

1) Q: Which path does a car take?: In this simulator, we assume that each car drives on a fixed path which is generated at random once the car enters the street network. The path always starts at a source street and ends at a sink street. A source street generates new cars, a sink street destroys cars.

In between the source and the sink, at every node, turns are taken at random at every node that has more than one outgoing street. By default, the next street chosen by the car is equally-distributed random.

2) Q: How does the simulator work to handle car dynamics?: Each car object carries the its system states

- longitudinal position x
- longitudinal velocity v
- longitudinal acceleration a

as well as knowledge about other agents

- long agents
- merge agents
- intersect agents

Section IV-B explains the definitions of agent types in detail. This approach is non-specific to any model and allows us to easily exchange car dynamics models.

3) Q: Why are the system states of the car object longitudinal?: We can observe that the movement of vehicles is two-dimensional by looking down on a realistic traffic network. It includes the motion in both longitudinal and lateral directions, which makes the problem complex. In the simulator, we control the movement of the vehicle by the motion model, for a detailed description of the motion model see section IV. For simplicity we use the longitudinal motion model that only involves the vehicle response in the longitudinal direction, and does not include lateral motions such as lane changes. In other words, this longitudinal motion model is one-dimensional. This model has practical physical significance because it describes the behavior of vehicles in the longitudinal direction and the effect from other agents, such as merging and intersection.

E. The street-node concept in other software

We do not claim ownership or invention of the street-node concept. In fact, the concept is so simple that it would not be realistic to claim any ingenuity on it. Connecting different streets by nodes is also intuitive and we can think of such use cases also outside the realm of traffic simulation. Upon a short look, a very popular software seems to use a similar concept. It is not employed for traffic simulation, but rather for navigational purposes.

Here, in figure 7 it is visible that streets in the Google Maps [7] software for navigation seem to indicate a direction and they seem to be connected to each other. We can already see that the underlying concept is very similar to ours. Upon closer examination by zooming in and switching to satellite view, we can see how the real street scenario, a complex multiple lane X-crossing is modelled.

In the close-up satellite view in figure 8 the similarity of the concept is striking. The individual streets of the modelled crossing are connected by slightly visible nodes. It is a street-node network. Using the satellite image underlying the street

Fig. 7. Unidirectional streets in Google Maps - Landgraf-Georg street crossing in Darmstadt city

Fig. 8. Unidirectional streets in Google Maps - Satellite view - Landgraf-Georg street crossing in Darmstadt city

network, it becomes clear that the difference between our approach and Google's approach is that Google does not attempt to closely model the reality. As one can see in the satellite image, multiple streets in the reality are modelled by one unidirectional street in the model.

This is acceptable because Google Maps uses this concept for navigation and does not aim to provide navigational instructions on individual-lane precision. For traffic simulation, however, we can not approximate multiple lanes with just one street. The use case of the street-node concept is different.

F. Visualization

The visualization and animation of the simulator are held simple. Cars are a random choice of four colors (blue, gray, green, red). Their position updates according to the specified movement model after each simulation step. A car illustration rotates according to the direction it is moving. The direction is determined by calculating $\Delta y = x_{new} - x_{old}$ and $\Delta y = y_{old} - y_{new}$. The rotational angle can then be calculated with trigonometry.

Also, the simulator attempts to visualize the acceleration of a car using multiple color-graded bars appended to each car. This helps the user to inspect motion models. The result will be shown in the evaluation section.

G. Additional Features

When designing the simulator, three additional features are added, which may be of great interest to subsequent applications. Additional features include editable node probabilities, agent-specific parameter storage for general motion models, and multiple color-graded bars.

1) Editable node probabilities: Editable node probabilities allow users to adjust the car's turn probabilities through the user interface if a node has more than one exiting street. A major advantage of this feature is that it makes cars more controllable and is easier to obtain specific traffic situations, such as traffic jams, which provides significant convenience for further investigation.

Node probabilities are distributed by default, which means that the probabilities are $50\% - 50\%$ if a node has two exiting streets. The probabilities will be recalculated when the probability bars on the user interface are dragged.

The implementation of editable node probabilities allows the simulation framework user to model real street networks more precisely. In the real world, not every turn at every street is chosen by equally-distributed probabilities. The real-world is more complex and therefore, this additional feature is needed. Even though we put it here under the "Additional Features" section, it is a rather fundamental concept of street network modelling without which the modelling of realistic scenarios would be only vague.

A simple example of this is a big, multiple-lane main road with a small branch-off onto a small alley. It would be very unrealistic to assume that half of all cars choose to drive onto the little alley, which does not happen in the real world.

2) Agent specific parameter storing for general motion models: Agent specific parameter storing for general motion models explicitly defines from several aspects how a car drives during a simulation. Each car behaves differently and has a corresponding set of parameters, which provides a more realistic simulation environment. For details on the types and values of the agent specific parameters, see Table I.

The storing of agent parameter is for general parameter set and is not specific to any motion models. For our testing Model IDM, the corresponding parameters will be detailed listed in section V. Any heterogeneous traffic scenario needs a motion model that allows for cars with different driving parameters. Therefore, the storing of agent-specific parameters is very important for realistic real-world traffic representation through simulation.

3) Multiple color-graded bars: Multiple color-graded bars are a visualization of speed and acceleration. Each car has 5 color-graded bars behind it during a simulation. The user can recognize the speed and acceleration respectively from the distance between two adjacent bars and the color of bars, as shown in Figure 9.

Red means acceleration is larger than 2 $\mathrm{m/s^2}$, blue means deceleration is less than -5 $\mathrm{m/s}$ and gray means acceleration is 0. The color is determined by RGB values that are linear with acceleration.

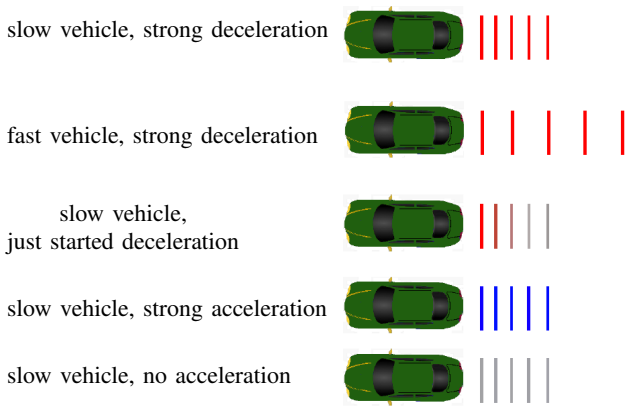


Fig. 9. Multiple color-graded bars

The purpose of this is (1.) to improve the look and feel of the simulation and aid the simulation user in gaining better insights into the motion models at hand and how car dynamics behave when they are implemented, and (2.) to help developers of a motion model debug and fix issues in their motion model. In case the acceleration and speed visualization does not correspond to what is expected of the model, developers can see in which situation the issue occurs. Additionally, (3.) the visualization of motion models is very useful because the simulation speed will not be constant over time. The more load (street and car models and their graphics) the simulation framework has to compute simultaneously, the slower the simulation will perform. To not confuse decelerating/slower cars with the decrease of simulation speed, it is essential to have an indicator of the actual acceleration that is used in the simulation.

IV. Traffic Dynamics Models

Now we are going to dive into the basics of traffic simulation, namely the motion models. Motion models are a class of models that can describe the movement patterns of vehicles moving on a traffic network. The more complex the model is, the closer it is to the realistic vehicle's movement patterns. People are always looking for simple and effective motion models to analyze many real traffic problems. A general motion model receives any car states as an input and returns the newly calculated car states as an output. This can really be anything from the common Intelligent Driver Model, to very extravagant, experimental models such as random acceleration and deceleration.

In this project, we aim to keep the possibility to simulate any motion model. Therefore, this section explains how the motion models take effect on individual agents in the simulator. Based on an understanding of how it works, anyone can add different motion models to the simulator according to their requirements.

A. Agent specific parameter for general motion models

The agent specific parameters, are applicable in all motion models. Because they describe the characteristics of the agent itself and how it drives on the traffic network. Regardless of the motion model used, these characteristics are objectively present and have physical significance, so these parameters play an extremely important role in the motion model. In the simulator, the parameters were chosen, as shown in the table below.

Parameter	Value	unit
Maximal velocity $v_{\max}$	50.0	$\mathrm{m/s}$
Minimal velocity $v_{\min}$	0.0	$\mathrm{m/s}$
Maximal acceleration $a_{\max}$	4.0	$\mathrm{m/s^2}$
Maximal deceleration $b_{\max}$	-8.0	$\mathrm{m/s^2}$
Reaction time t_{react}	0.5	s
Maximal agent gap $d_{\max}$	200.0	m
Agent length $\mathrm{car}_{\mathrm{length}}$	7.0	m
Duration of acceleration period a_{duration}	1.0	s

TABLE I
GENARAL MOTION MODEL PARAMETERS

The agent specific parameter set consists of maximal velocity $v_{\max}$, minimal velocity $v_{\min}$, maximal acceleration $a_{\max}$, maximal deceleration $b_{\max}$, reaction time t_{react}, maximal agent gap $d_{\max}$, agent length $\mathrm{car}_{\mathrm{length}}$ and duration of acceleration period a_{duration}. The maximal and minimal velocity limit the speed of a car within a reasonable range to describe the type of highway used. Maximal acceleration, maximal deceleration, and car length determine the type of car. Reaction time and duration of the acceleration period are a reflection of the type of driver. A reasonable selection of parameters will be important for simulation results that have a high practical value. For example, if maximal deceleration is too large, the braking distance will be noticeably short. In other words, the high-speed car will stop immediately if there is a car ahead of it, which is impossible.

B. Effect between agents in the traffic flow

In reality, a driver driving on a highway must always observe the movement of the surrounding vehicles to determine the subsequent driving strategy, i.e., accelerate, decelerate or maintain the current speed. Therefore, the essence of the motion model is the effect between vehicles.

Then the question arises: which vehicles can affect the car when it's driving on the road? In fact, different motion models have different views on this. For example, in the IDM, each vehicle is influenced by one leading car "in front" of it. However, in other models, each vehicle may be influenced by multiple vehicles at the same time.

We use the concept of environment sets to ensure that the simulator can apply different motion models. Each agent

has its own environment set, which stores all other agents associated with the current agent at the current time step and is updated in real-time during the simulation. Different motion models can select different other agents in environment set for calculating the next states of the current agent.

According to the position relationship between agents, each environment set is divided into four parts, namely long agents, merge agents, intersect agents, and parallel agents. Parallel agents store the agents driving in the parallel lane of the current agent's lane, and are only considered if there are multi-lanes on the traffic network. Since we are using the longitudinal motion model in the simulator (see section III-D.3 for details), this part will not be discussed here.

1) Long agents: This part stores all agents nearby the current agent. Here we define the scope of "nearby" as the agent's current lane, the previous lane it has driven through, and the next lane it is about to enter.

2) Merge and intersect agents: Merge agents represent all agents about to be merged with the current agent, while intersect agents store all agents about to be intersected with.

An intersection and a merge both lead multiple cars into the same node. The difference is, that merge agents are cars that will pick the same outgoing street, whereas intersect agents are cars that will pick different outgoing streets.

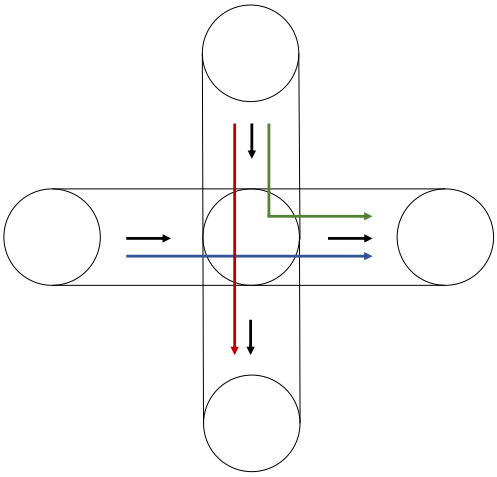


Fig. 10. merge agent (green) and intersect agent (red) from a given car's perspective (blue)

The difference is illustrated in figure 10 from a given car agent's perspective, highlighted in blue. A car that will enter the same node and continue its path on the same outgoing street is considered a merge agent. A car that will enter the same node but continue its path on a different street is considered an intersect agent.

C. Currently applied motion models

The motion model of choice for the first implementation and evaluation are the IDM and the adapted models based on IDM. Although we covered several different models in the simulator, only the basic IDM model will be explained here.

1) The IDM: The IDM [8] is one of the simplest, most comprehensive, and accident-free car-following motion models. It describes the change in the condition of the vehicle between free flow and congested flow on a single lane.

First, the IDM comes with a variety of adjustable parameters, see Table II for details. Different from the agent-specific parameters, they are only applicable to all IDM-based motion models for describing the different driving styles of drivers and different vehicle types. For example, a cautious driver is characterized by low v_0, $|b|$ and high T. Compared to heavy vehicles, light-duty vehicles tend to have higher v_0 and $|b|$. For a detailed description of IDM parameters, we refer to [9].

Parameter	unit
Desired velocity v_0	$\mathrm{m/s}$
Comfortable deceleration b	$\mathrm{m/s^2}$
Minimal net distance in traffic jam s_0	m
Safe time gap T	s
Acceleration exponent δ	–

TABLE II
IDM PARAMETERS

Now let's focus on how this model works. In the IDM, the current vehicle only considers itself and its leading vehicle. Depending on the road conditions, the leading vehicle is the front, the merging, or the intersecting vehicle of the current vehicle. Any driver's decision to accelerate or brake depends only on his or her speed v, the net distance to the leading car s (bumper to bumper), and the speed difference with the leading car Δv.

The acceleration equation for each vehicle

$$a = \frac{dv}{dt} = f\left(v, \Delta v, s\right) = a_{free} + a_{brake} \tag{1}$$

consists of two parts, the acceleration of the vehicle in the free flow (2) and the deceleration in the congested flow (3):

$$a_{free} = a_{max} \cdot \left[1 - \left(\frac{v}{v_0}\right)^{\delta}\right] \tag{2}$$

$$a_{brake} = -a_{max} \cdot \left(\frac{s^*\left(v, \Delta v\right)}{s}\right)^2 \tag{3}$$

where $s^*\left(v, \Delta v\right)$ is the expected spacing of the driver in the current situation, defined as follows:

$$s^*\left(v, \Delta v\right) = s_0 + max\left(0, vT + \frac{v\Delta v}{2\sqrt{a_{max}b}}\right) \tag{4}$$

The vehicle under the IDM follows the following laws of motion:

- When the vehicle is in a free-flowing state, its velocity always converges gradually to the desired velocity v_0 until they are equal.
- When the vehicle is in a crowded flow state will try to maintain a distance of s^* from the leading vehicle.

2) The IDM Plus: The Intelligent Driving Model Plus is an adapted version of the IDM proposed by Wouter J. Schakel et al. [10] to focus on the stability of traffic flow, here referred to as IDM+. As shown in (5), the difference between IDM and IDM+ is that the free flow and congested flow terms are clearly separated in the IDM+.

$$\frac{dv}{dt} = a_{max} \cdot min\left[1 - \left(\frac{v}{v_0}\right)^{\delta}, 1 - \left(\frac{s^*(v, \Delta v)}{s}\right)^2\right] \quad (5)$$

D. The models with interference

It is impossible for any driver to always make the most accurate judgments about current road conditions. Therefore, we need a generic noise model to simulate some errors in the driver's judgment in each time step.

We assume that the noise to a moving vehicle is influenced by two factors. One is the state of that vehicle itself, and the other is the state of the vehicles it is currently focused on. This raises a problem: for the second factor, different motion models will pick different vehicles.

To ensure that the noise model can be applied to any motion model, we build the environment value set on the base of the environment set. It stores the noise values of all agents in the environment set for the current agent, and it is also updated in real-time during the simulation. Each noise value follows the equation

$$a_{noise} = \left(1 - \frac{\Delta v}{v}\right) \cdot min\left[0.2, p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right] \quad (6)$$

where $p(x)$ is Gaussian noise. It refers to the noise whose probability density function follows Gaussian distribution, where x represents the gray level [11]. Figure 11 is a graph of Gaussian noise with normal distribution.

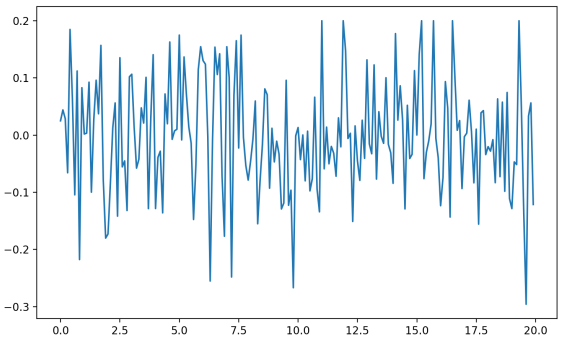


Fig. 11. Gaussian noise with its mean gray value μ of 0, standard deviation σ of 0.1, and upper limit of 0.2

When an agent is affected by other agents, the corresponding noise values can be selected from the current agent's environment value set. If there are more than 1 other agent, the interference can be calculated as the sum of other agents' noise. This is the basic mathematical description of noise between agents.

V. Results and Evaluation

In the previous sections, the main traffic simulator concepts and the principles of how different motion models work in the simulator were explained in detail. Now a specific street network with an IDM is applied and the performance of the modifiable traffic simulator is discussed.

A. Establishment of street network

The street network is an essential part of a successful simulation. An appropriate traffic simulator should be applied to all types of street networks, which means that a hardcoded traffic simulator with a single street network is not acceptable. Therefore, the modifiable street network is the first criterion during the development of our traffic simulator.

The user can easily draw a defined street network on his own by entering the coordinates of the nodes. Two nodes define a street and if two streets are connected, they should have the same node. A street network with two X-crossings, one T-crossing, and two merge scenarios were drawn for our traffic simulator, as shown in figure 12.

Fig. 12. an overview of street network

B. Selection of Motion models

A reliable motion model is a key aspect of a simulation, which has relevance for practical applications. Therefore, a convenient switch between motion models, when comparisons between motion models are required, should be considered. According to this requirement, a selection area has been added to the user interface, where users can choose directly the corresponding template. This area is also extensible for new motion models if a project team in the future designs the models. Our traffic simulator has implemented 3 IDMs, as set out in Table III. This simulation chose lim_idm with a noise model.

Motion model	noise
idm	with/without
lim_idm	with/without
lim_idm+	with/without

TABLE III
MOTION MODELS WITH/WITHOUT NOISE

C. Car dynamics for different traffic scenarios

The simulation aims to verify the validity of our traffic simulator, which can be achieved by observing car behaviors in different traffic scenarios. 15 sets of the IDM parameters were chosen as shown in Table IV.

Parameter	Value	unit
v_0	$16.0 + 0.4 * i, i = 0, \ldots, 14$	m/s
b	$-4.0 - 0.4 * i, i = 0, \ldots, 14$	m/s^2
s_0	2.0	m
T	$1.5 - 0.1 * i, i = 0, \ldots, 14$	s
δ	4.0	–

TABLE IV
INTELLIGENT DRIVER MODEL PARAMETERS

1) Car-following scenario: A car-following scenario is a scenario without merge agents and intersect agents. The cars should follow the front car and keep a safe distance to it. In other words, the behavior of the car should be affected by the front car. The simulator captured 5 successive cars' dynamics(speed and acceleration), which means a quantitative representation of car behaviors, to verify whether the IDM was correctly implemented.

As shown in figure 13, 4 cars behaved similarly, with the exception of car 0. The reason for the unique behavior of car 0 is that car 0 was the first car in the simulation and there was no car in front. Because of the other 4 cars' similar behavior car 1 was taken as an example to analyze, it was at first faster than the front car (car 0) and therefore decelerated to avoid a crash with car 0. When a safe distance was maintained, car 1 was moving uniformly. The small disturbance of the dynamic curve was caused by the noise of the front car.

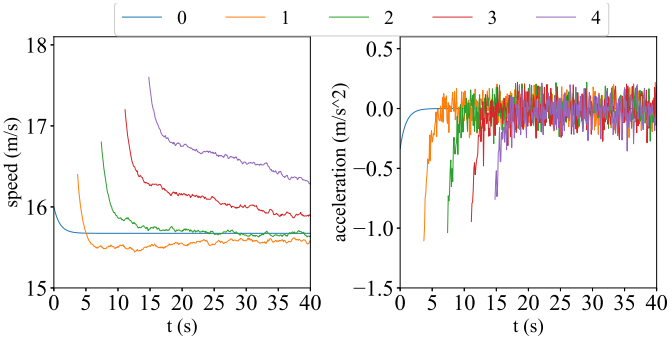


Fig. 13. Car dynamics on straight street

2) Merge scenario: If a node has more than one incoming street, this node is called a merge point and the incoming streets are the merge street. A scenario with a merge point is a merge scenario. When a car drives to a merge point, it will detect whether there are cars on the other merge streets. If the cars exist, they should follow the 'first come first go' principle, which means the car closer to the merge point should go first and the other cars should decelerate even stop.

Figure 14 illustrates the dynamic of 5 cars. All 5 cars behaved similarly. Taking car 2 as an example, it firstly decelerated slightly until approximately 11 secs, sharp decelerated from 11 secs to 12 secs, and then accelerated. The reason for the first deceleration is that car 0, as a front car, was slower than car 2, which means car 2 should decelerate to avoid a car crash. The second sharp deceleration is due to the existence of merge cars. After the merge car drove through the merge point, car 2 accelerated. The car behavior on T-crossing and X-crossing is similar to the merge scenario and will not be discussed in this section.

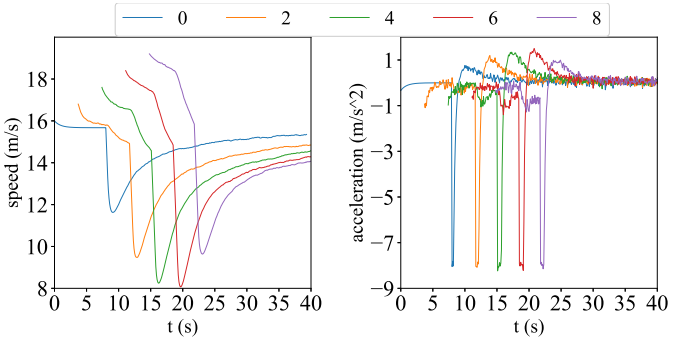


Fig. 14. Car dynamics in merge scenario

D. Evaluation of simulator

Flexibility and extensibility are always two basic requirements during our simulator design. Therefore, the simulator framework consists of different independent modules to ensure these two requirements. An independent module means this module can be debugged, maintained, and even replaced without influencing other modules. Based on this point, the simulator framework is flexible and extensible.

The simulation test has 3 steps in all, and every step is accomplished in the corresponding module. The street network is drawn by `setupStreetNetwork_CN.py`, in which street nodes' coordinates are defined. It certainly could draw a new street network by altering coordinates. However, the point is that the developer in the future can easily combine this module with his module. For example, a developer can write a module that can convert pictures of a street network to corresponding coordinates, which can be applied in our module. The motion models are implemented in `car.py`, in which the developer can design new motion models in the future. The measurement takes place in `logger.py`, in which all cars' dynamics are by default recorded. The developer can record the car on a specific street by altering a few lines of codes. Based on the recorded data, an analysis module, if needed in the future, can also be developed. This kind of extension is quite convenient in our simulator framework.

The Test progress above has shown that this traffic simulator was flexible with arbitrary street networks and extensible with arbitrary motion models. The simulator is modifiable that any motion model in the future could be adapted easily to test the validness of different motion models and any street network could be applied conveniently to investigate the car behaviour of different traffic scenarios.

Additionally, the simulator framework has a good design in the aspect of visualization. For example, the multiple colour-graded bars provide users an intuitional representation of car

dynamics. The users observe car acceleration through the colour of bars with which it's more convenient to test the correctness of motion models.

A limitation of this simulator is that the simulator framework is based on one lane street and has no other traffic objects like a traffic light, which means the simulator is only suitable for simple traffic scenarios. Another limitation is the animation lag of our simulator framework. As calculation increases, the animation lag will appear, which may cause visual discomfort to the user. The reason is that the increasing calculation time becomes larger than the animation update time per step. A multi-progressing method, which can speed up computer calculation, maybe the answer to it. Even so, this simulator framework could be a basic framework for further work.

VI. Further ideas

Currently, the simulator can implement relatively single traffic scenarios (roads, bridges and tunnels). Objects such as traffic lights, roadblocks and speed limitation signs can be added in the future work to simulate more complex traffic situations. Then we can also use the simulator to find the optimal design for the road by observing the change of traffic flow. In the aspect of dynamic models the simulator is not limited to the motion models that are available today, new motion models can be added for simulating. It is also possible to extend the single lane to multiple lanes to simulate the Lane-Changing Model. In addition, the current representation of turning lanes in the simulator is somewhat "rigid": we use multiple straight roads to stitch together an approximate rounded lane. This can also be improved so that the vehicle's performance during the turn is closer to "reality".

As explained in section III-E "The street-node concept in other software", it is possible to derive from satellite images traffic networks for navigation as Google LLC already does. We do not know whether these street networks are generated fully automatically or whether human review and intervention is necessary. Still, this is a good opportunity for traffic simulation of existing traffic scenarios. In case, researchers want to be able to evaluate the dynamics of traffic in a certain locale in a city, they should be able to derive from satellite data an actual representation of the street network that is compatible with our simulator. All it needs is a software that reads either satellite imagery directly and performs street network mapping, or a software that is able to access Google Maps API [12].

Alternatively, for unrestricted use, which is more common and useful for research, this same approach can be taken with OpenStreetMap [13], which is an open software that is free to use for everyone. From this, future works could attempt to remodel real-work street scenarios and allow the possibility to simulate with any traffic motion model any street network anywhere in the world.

APPENDIX I
USER INTERFACE

The user interface provides on the left side the simulation animation and on the right side simulation controls.

A. Start, Stop, Visible

The Start and Stop buttons start and stop the simulation.
The Visible button hides the animation interface from view.

B. Streets and Nodes in the animation

Each street and each node has its own ID. The street ID plus the direction of the street are indicated in the street network animation. Controllable nodes are highlighted in green and also display their id.

By setting the "street ID" hook, you can make the text on the streets (text of node ID, street ID and street length) disappear or be displayed.

C. Speed representation

In the animation, the state of cars (speed, acceleration) is visible. Every time step (except for the first four time steps of the agent of course) behind the cars there would be 5 bars.

The distance between the bars gives a good visualisation of the vehicles speed and the color of the bars is used to visualize the acceleration. Full red means strong braking, full blue strong accelerating while gray means no acceleration (driving at constant speed) and acceleration values are interpolated.

By setting the "Speed representation" hook, you can make bars disappear or be displayed.

D. Inflow control

The inflow control sets the number of cars entering the scenario per hour.

E. Controllable nodes

Nodes can have custom node probabilities. For each outgoing street of a customizable node, a probability can be set. The probability determines the routes cars take.

Here the probabilities used are weighted probabilities and what you set are the weights. For example, if you set the weights to 3 and 50, then the first street will be picked 3 out of 53 times, and the second street will be picked 50 out of 53 times. Here the point of weights is that it is more flexible when you have more complex scenarios.

To set the newly configured probabilities, press the apply button.

F. Driving modes

All car objects in the street network should be driven with the same movement mode. The driving mode should be selected before the simulation. Please try not to change the mode after you have clicked the start button. Frequent mode changes may cause the simulation to crash.

The following three different movement models are currently available: idm, lim_idm and lim_idm+.

To select the motion model by clicking on the different buttons. By setting the "Gaussian noise" hook, you can choose whether to add noise to the current motion model or not.

APPENDIX II
CODE STRUCTURE

The project consists of the files:

A. Images

- `car_blue2.png` - blue car image
- `car_gray2.png` - grey car image
- `car_green2.png` - green car image
- `car_red2.py` - red car image

B. Network setup

- `setupStreetNetwork_CN.py` - sets up a complex street network
- `setupStreetNetwork_SN.py` - sets up a simple street network

C. Only animation

- `carDrawer.py` - manages all car illustration objects in the animation pane
- `carIllustration.py` - represents an individual car illustration
- `drawStreetNetwork.py` - draws the street network
- `userInterfaceDrawer.py` - creates the user interface

D. Dynamics calculation

- `ground_structure.py` - provided basic dynamics structure

E. Parameter setting

- `param_manager.py` - manages parameters of the simulation
- `Parameter.py` - represents an individual parameter

F. Logger

- `logger.py` - logs the simulation car states into a .log file
- `logger_dict.py` - logs the simulation car states in the form of a dictionary into a .txt file

G. Mixed

- `mainStreetNetworkTest.py` - the **main** file functions
- `Live_Simulation.py` - performs the realtime simulation (moves vehicles and updates the environment)
- `car.py` - represents a car, calculates the state of the car based on the motion model
- `lane.py` - represents a single lane
- `street.py` - represents an individual street
- `streetConnectionPoint.py` - represents a node that connects two streets
- `streetNetwork.py` - represents a full street network consisting of many streets
- `mathUtil.py` - provides space to add math utility

APPENDIX III
STARTING THE SIMULATOR

The simulator is started using the `mainStreetNetwork Test.py` file.

Aside from that, relevant to the user are only `car.py` and the `setupStreetNetwork_*.py` files.

A. car.py

This file contains the car class. It represents the logic and dynamics of an individual car object. The behavior should, in general, not be modified. Only the car movement model can be safely modified.

During the simulation, function `__calculateCarStateAccordingToMovementModel()` calculates the state (position, velocity and acceleration) of the car object at the next time step based on the currently selected motion model.

The leading double underscore indicates that this method is `private` and should never be called outside of the `Car` class.

Then the state of the car object is updated by the function `updateState()`.

B. setupStreetNetwork_*.py

The main advantage of this simulator is, that the street network can be freely modified. This can be done in the `setupStreetNetwork_*.py` file.

It is easiest to see how it works by looking at the code directly:

```
# Build a simple street network
def buildStreetNetwork():

    ### Define all of the street connection nodes
    # Entrance nodes
    node0 = StreetConnectionPoint(100, 650)

    # Exit nodes
    node13 = StreetConnectionPoint(950, 100)

    # Merge and intersection nodes
    node4 = StreetConnectionPoint(650, 600)
    node7 = StreetConnectionPoint(950, 200,
                isProbabilitySettable=True)

    # Rest of the street nodes
    node1 = StreetConnectionPoint(200, 650)
    ...
    node12 = StreetConnectionPoint(150, 600)


    ### Define all of the streets
    # Entrance Streets
    street1 = Street(node0, node1,
                canSpawnCars=True)

    # Exit Streets
    street14 = Street(node7, node13)

    # Merge and intersection streets
    street4 = Street(node3, node4)
    street13 = Street(node12, node4)
    ...

    # Rest of the streets
    street2 = Street(node1, node2)
    ...
    street12 = Street(node11, node12)


    # All streets to be drawn in the simulation
    streets = [street1,
               ...
               street14,
              ]
    streetNetwork = StreetNetwork(streets)

    return streetNetwork
```

First, nodes are placed at the desired coordinates. Nodes, where the turn probabilities shall be modified simply receive a `isProbabilitySettable = True` parameter. The simulator then adapts the user interface controls automatically for this node.

Then streets are setup using the nodes. A street gets two nodes as input parameter and then is automatically created connecting the specified nodes.

If a street shall be able to be a spawn (source) for new cars, the additional parameter `canSpawnCars = True` is passed to it.

A street can also be turned into a tunnel by setting the `isTunnel = True` flag. The difference between a regular street and a tunnel street is only the representation in the animation. Cars become invisible when entering the tunnel and reappear after leaving it.

Then, all streets are put into an array and used to initialize and return the street network.

Thus, the process of creating any desired street network is made as simple as possible for the user.

The user can configure as many street networks as desired using this approach. He only has to set the import statement `from setupStreetNetwork_CN import buildStreetNetwork` in the `mainStreetNetwork Text.py` file accordingly to match the filename. In this case, the complex (CN) streetNetwork is loaded, which is built-in `setupStreetNetwork_CN.py`.

APPENDIX IV
LOGGER

The logger configured by the `logger.py` and `logger_dict.py` files runs automatically with every simulation. A .log file and a .txt file are created in the project folder containing the car states at each simulation step.

A. *.log file

```
[{0: 'x=1.532; v=15.639; a=5.76'}, {1: 'x=1.533; v=15.668; a=6.07'}
                                  ]
[{0: 'x=3.125; v=16.215; a=5.125'}, {1: 'x=3.131; v=16.275; a=5.449
                                   '}]
...
[{0: 'x=71.635; v=19.51; a=0.003'}, {1: 'x=72.884; v=19.925; a=0.
                                    004'}, {2: 'x=1.533; v=15.659; a=
                                    5.973'}, {3: 'x=1.535; v=15.698;
                                    a=6.394'}]
...
```

Fig. 15. Sample *.log file

The *.log file contains multiple lists. Each list corresponds to all cars on the street network at a certain time point and their status (longitudinal Coordinate, speed, acceleration).

Figure 15 is a sample *.log file. The scenery starts with two cars (car0 and car1), while at the first time step the coordinate of car0 is $1.532m$, the velocity is $15.639m/s$ and the applied acceleration is $5.76m/s^2$. In the second time step the position of car1 is $3.131m$, the speed is $16.275m/s$, and the applied acceleration is $5.449m/s^2$. In the * time step, car2 and car3 enter the scenery with an initial speed of $15.7m/s$.

B. *.txt file

The *.txt file contains the data constructed in two embedded dictionaries. The key of the outer dictionary is describing the time step, the keys of the inner dictionaries describe the car ID. Each car key leads to a List $[x, v, a]$ of this car at the current time step. An example of *.txt file is illustrated in figure 16.

So in the following example, the scenery starts with two cars (0 and 1) and the simulation ended after 347 time steps with 4 cars, with car0 with $[x, v, a] = [12.695, 1.9, -3.0]$.

```
{
0: {0: [0.0, 10.0, 1.0], 1: [5.5, 8.0, 0.0]},
1: {0: [1.005, 10.1, 1.0], 1: [6.3, 8.0, 0.0]},
...
345: {0: [12.5, 2.0, -1.0], 1: [13.0, 3.0, 0.0], 2: [0.0, 10.0, 1.0
                                ], 3: [6.5 , 5.0, 0.0]},
346: {0: [12.695 , 1.9, -3.0], 1: [13.3, 1.0, 0.0], 2: [1.005, 10.1
                                , 1.0], 3: [7.0, 8.0, 0.0]}
}
```

Fig. 16. Sample *.txt file

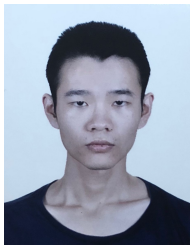
REFERENCES

[1] "Traffic simulation software - ptv vissim - ptv group," https://www.ptvgroup.com/en/solutions/products/ptv-vissim/, accessed: 2021-01-28.
[2] "Ptv vissim: Simulation of a complex intersection," https://youtu.be/OtYby7QnyAE, accessed: 2021-01-28.
[3] "Sumo - simulation of urban mobility," https://www.eclipse.org/sumo/, accessed: 2021-01-28.
[4] "Eclipse sumo github repository," https://github.com/eclipse/sumo, accessed: 2021-01-28.
[5] M. Treiber, "Traffic simulation," https://traffic-simulation.de/.
[6] "traffic-simulation.de github repository," https://github.com/movsim/traffic-simulation-de, accessed: 2021-01-28.
[7] "Google maps - a mapping service developed by google," https://www.google.de/maps/place/Darmstadt/, accessed: 2021-01-28.
[8] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E62*, pp. 1805–1824, 2000.
[9] M. Treiber and A. Kesting, "Interactive Traffic Simulation - Microscopic Open-Source Simulation Software in Javascript," 2017.
[10] W. J. Schakel, B. van Arem, and B. D. Netten, "Effects of Cooperative Adaptive Cruise Control on traffic flow stability," 2010.
[11] P. Cattin, "Image Restoration: Introduction to Signal and Image Processing," 2012.
[12] "Google maps platform documentation," https://developers.google.com/maps/documentation, accessed: 2021-01-28.
[13] "Openstreetmap - open maps free of license," https://www.openstreetmap.org/, accessed: 2021-01-28.

Yiran Shu Yiran Shu (B.Sc.) studies electrical engineering and information technology at TU Darmstadt, specializing in automation and control engineering. She wrote her bachelor's thesis in the School of Optical-Electrical and Computer Engineering at the University of Shanghai For Science and Technology, China, with a focus on substation and distribution system design.

Mengnan Jiang Mengnan Jiang(B.Sc.) studies electrical engineering and information technology at TU Darmstadt, specializing in automation and control engineering. He wrote his bachelor's thesis in college of Electrical Engineering and Automation, Fuzhou University, China, focusing on Large-capacity AC Contactor System.

Daniel Hirsch Daniel Hirsch (B.Sc.) studies electrical engineering and information technology at TU Darmstadt, specializing in automation and control engineering. He wrote his bachelor's thesis with the communication technology department, focusing on Networked Control Systems. Aside from his studies, he works as a Junior Software Engineer at Telespazio VEGA, where he develops operational mission planning software for European Space Agency (ESA/ESOC).



Jiekai Jia Jiekai Jia (B.Sc.) studies electrical engineering and information technology at TU Darmstadt, specializing in automation and control engineering. He wrote his bachelor's thesis in the School of Electrical Engineering at the Hebei University of Technology, China, with a focus on the reliable design of capacitors.