E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Development of a Microscopic Driver-Centric Simulator with Unity3D and SUMO

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Carlos Biurrun Quel

Luis Javier Serrano Arriezu (Universidad Pública de Navarra)

Cristina Olaverri Monreal (FH Technikum Wien)

Pamplona, 27 de Junio de 2016

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Declaration of Authenticity

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (for example see §§ 21, 42f and 57 UrhG (Austrian copyright law) as amended as well as § 14 of the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

In particular I declare that I have made use of third-party content correctly, regardless what form it may have, and I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see § 14 para. 1 Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.”

Place, Date	Signature

Resumen

Diversas herramientas han sido desarrolladas en el campo de la simulación de tráfico. Estas herramientas permiten recrear situaciones de tráfico complejas de una forma realista, que pueden ser empleadas en la planificación de sistemas de transporte o para evaluar comunicaciones V2V (Vehículo a vehículo) en situaciones reales de tráfico.

A pesar de que ciertos trabajos han sido llevados a cabo en el campo de la representación 3D de las simulaciones de tráfico, no existen demasiados ejemplos de plataformas Driver-Centric (con perspectiva del conductor) en los que el comportamiento del resto de vehículos es modelado por un simulador de tráfico.

En este Trabajo Fin de Grado se presenta la elaboración de un simulador de tráfico Driver-Centric, capaz de emplear redes de carreteras reales junto con modelos de tráfico realistas, en un escenario 3D. Una librería existente en Java ha sido traducida a .NET y usada en Unity3D para implementar el protocolo TraCI y lograr la comunicación con el programa de simulación microscópica de tráfico, SUMO. Finalmente, se ha evaluado el funcionamiento del simulador y se han presentado futuras líneas de trabajo

Palabras clave: Driver-centric, SUMO, Unity3D, TraCI, Simulación microscópica

Kurzfassung

Es wurde eine Vielzahl von Tools fuer Verkehrs-Simulationen entwickelt. Diese Verkehrs-Simulations-Tools erlauben die Simulation von speziellen Verkehrs Situationen und eine Auswertung wichtiger Parameter fuer Verkehrsplanungen. Weiters koennen effekte von V2V Kommunikation getestet werden.

Es wurden bereits eine Vielzahl von fahrer zentrierten 3d simulatoren erstellt, jedoch ohne das Einbeziehen von Fahrverhalten der anderen simulierten Verkehrsteilnehmer.

In dieser Bachelorrbeit werden Farher zentrierte 3D Simulationen erarbeitet. Diese 3d Simulation basiert auf echt Zeit-Daten, reale Strassen und Verkehrssituationen. Dadurch wir ein 3D-Modell erstellt. Eine bereits existierende Java bibliothek wurde in eine .Net bibliothek convertiert. Weiters wurde eine Unity3D fuer die Implimentierung eines TraCI Protokoll erstellt. Grund hierfuer ist Schnittstelle zum mikroscopischem Verkers simulator SUMO. Abgeschlossen wird die Bachelorabeit mit einer Evaluation der Performence des simulators und moeglichen neuen Arbeitsfeldern.

Schlagwörter: Driver-centric, SUMO, Unity3D, TraCI, Mikroscopischem Simulation

Abstract

Many different tools have been developed for traffic simulation. These tools allow to represent complex, realistic traffic situations that can be useful for transport planners in order to evaluate specific traffic situation or in order to test Vehicle-to-Vehicle or communications with real traffic situations.

Although some work has been carried out in the field of 3D visualization of traffic simulation, there are not many examples of Driver-Centric driving platforms in which the mobility behaviour of the rest of the vehicles is modelled by a traffic simulator.

In this bachelor thesis, the elaboration of a 3D Driver-Centric Simulator is proposed, capable of using real-world road networks together with realistic traffic models, in a 3D scenario. An already existing Java library has been translated to .NET and used in Unity3D for implementing the TraCI protocol to communicate with the microscopic traffic simulator SUMO. At the end, evaluation of the simulator's performance is presented and future lines of work are defined.

Keywords: Driver-centric, SUMO, Unity3D, TraCI, Microscopic Simulation

Acknowledgements

First of all, I would like to thank my supervisor in Vienna, Dr. Cristina Olaverri, for her patience, assistance and attention, especially in the last month of the semester, and also for introducing me to some colleagues in the department.

I would also like to thank my supervisor in Pamplona, Dr. Luis Serrano, for the attention provided since many months before arriving to Vienna and the efforts made to find a supervisor like Dr. Olaverri, as well as solving the different doubts prompted during this stay.

I would also like to thank my department colleagues, Aso, Roman and Florian, for their help, kindness and attention. Every time I had a doubt or a problem, they had made the effort of meeting with we and trying to provide a solution. Special mention to Florian for his support with Unity3D. Without him, this work would have not been possible.

Gracias a mis amigos, los de toda la vida, también los que me han acompañado durante los 3 años y medio de carrera realizados en la UPNA, y los que me han acompañado este último semestre, bien sea desde la distancia, o los conocidos en Viena.

Gracias por todos esos “anims” en catalán, “egurres” en euskera y esos “venga que tú puedes” en castellano durante esta recta final.

Mención especial para los “chés”, Lucas y Gonzalo, por esas tardes interminables revisando dónde estaba el error entre las líneas de un script.

La palabra agradecer se queda corta en este apartado, donde me refiero, como no podría ser de otra forma, a mis padres. Gracias. Gracias por haberme criado como lo habéis hecho, inculcándome los valores del respeto y el esfuerzo personal como máximos en esta vida. Gracias por ofrecerme la oportunidad de vivir una experiencia en el extranjero que, sin duda, va a marcar un antes y un después en mi forma de ser y mi forma de concebir el mundo. Dedicaros este Trabajo Final de Grado es lo menos que podría hacer.

Gracias también al resto de mi familia, a mis hermanos, Javier e Isaac, que aunque estemos cada uno en una punta del mundo, nos veremos pronto. A mi abuela Alicia, a mis tías Mari Santos y Carmen, a mi tía Rosi y mi Ramón, a mis primos Miguel, Manuel y Walter, y por último pero no menos importante, a prima Sandra y a su “chiqui” cuyos videos tantas sonrisas me han sacado este semestre.

Finalmente, mención especial a los que ya no están. A mi tío Fausto, mi padrino. Por todas esas “zapatillas” en la calle del Laurel. Porque siempre me transmitías una confianza increíble y me hacías creer que podía con todo. Ni siquiera me viste empezar la carrera, pero has estado presente durante todo el trayecto. Y como no, a mi abuelo Segundo, que tomaste el último tren de repente, sin llegar a poder despedirnos, el 23 de marzo. Por todas esas historias que nos contabas, ese carácter incansable, que te impulsaba a ir día tras día a la huerta para dar de comer a tu familia, así como tu humor incansable, y todos esos chistes que te sacabas de la manga. Porque sois los que me habéis empujado durante el camino, y sobre todo en esta recta final, gracias.

Table of Contents

1 Introduction

The current section of introduction is divided in 2 subsections. In first place, the motivation on which this work is based, is presented. In the next one, the statement of the research problem and objectives is exposed.

1.1 Motivation

The motivation of this work is based on the potential features of a Driver-Centric simulator whose movement is based on a microscopic traffic simulator could have, and the potential areas of research in which this simulator could be used, being able to evaluate the influence of the traffic on the driver and vice versa.

Although traffic modelling and simulation started over many decades ago, the constant increase of the amount of vehicles in the road, and the need for developing new transport infrastructure, together recent increasing relevance of new technologies applied on vehicles, such as VANET, which consists on vehicles communicating between them while driving, makes traffic simulators an important are of research.

Together with this VANET development, several In-Vehicle Information Systems (IVIS) and ADAS – Advanced Driver Assistance Systems - have been developed [1][2]. These applications have many different potentials, from improving driver's safety, at the same time as enhancing the use of the transport networks, increasing their efficiency improving driver's comfort during the driving task or just as a way of entertainment for the passengers.

However, this systems can be a potential cause of distraction for the driver. The deployment and testing of these applications in a real vehicular environment entail some potential risks, endangering the testing driver, at the same time it requires a considerable investment in terms of money and time.

On the other hand, simulation states as a reliable, low-cost solution, which, in addition, provides a quick feedback at the same time it offers a lot of flexibility, offering the possibility to create many different, realistic, scenarios that could take place in a real environment.

Although there any several projects on traffic simulators working together with network simulators, as well as projects regarding driving simulators, little work has been done in development of realistic, real-time 3D visualization of traffic simulators together with a driver-centric perspective or driving simulators.

For this reasons, in this bachelor thesis the development of a Driver-Centric simulator coupled with a powerful, realistic traffic simulator for generating the boundary conditions of the traffic network and the movement of the vehicles contained in it.

1.2 Research problem and objectives

The development of a Driver-Centric 3D driving simulation platform based on a realistic traffic simulator states as a useful tool that could be employed in different situations:

- Research in In-Vehicle Information and Advance Driver Assistance Systems, such as HUDs (Heads-Up Displays) – which for example may display important information regarding the network status – allowing to test and evaluate their performance as well as their effects on drivers behaviour.

- As a training tool for evaluating and/or improving driving skills in a realistic traffic situation generated by a traffic simulator.

In order to achieve this, a powerful game engine such as Unity3D, is used in order to represent a realistic 3D traffic scenario which is modelled by a traffic simulator which usage has increased in the past years due to its open-source and high portability, SUMO. [3]

As it is presented in the related work section, there are plenty of works and studies which show that TCP communication and remote control of SUMO is possible through TraCI protocol. Therefore, there are plenty of evidences that connection between this tool and a 3D engine environment such as Unity 3D would be possible as well.
Achieving this, an in-vehicle perspective – as well as a 3D-full-scenario perspective – of the traffic network will be provided. Besides, the interaction between the driver and the network is such that the driver can also influence the rest of the vehicles.

The rest of the current document is organised as follows:

In Section 2, an introduction about simulation fundamentals and traffic flow theory was explained. Then, an overview of the state of the art of traffic simulation and driving simulators is presented.

Section 3 presents the implementation process. First, the generation process of the traffic scenarios (defining the road network, assigning the traffic flows…) is explained. Then, the TraCI protocol, interface through which the communication with SUMO is possible, is presented. The use of an existing library for implementing this protocol comes next. Finally,

the generation of the 3D scenario is detailed, with special emphasis in explaining the algorithm for retrieving and representing the objects of the network.

Some comments regarding the evaluation of the developed simulator are made in Section 4, while the last section includes the conclusions of the current document and introduces some lines for future work.

2 State of the art and related work

Simulation platforms are remarkable and very effective tools for analysing different scenarios which could not be studied by other means or which cost would not be affordable. [4]

Simulation platforms base their calculations on one or more mathematical approaches and algorithms called simulation models.

2.1 Clasification of simulation models

Simulation models can be categorised in several ways, however, the 3 fundamental classifications are the distinction between deterministic and stochastic, static and dynamic and discrete-event and continuous, as stated in [5] and furtherer explained in [6]:

Deterministic vs Stochastic:

- A **deterministic model** is that which consists essentially of equations related between them. In this type of simulations there are one or more known – this is, not random – inputs that will lead to one or more known and unique outputs, without any random component.
  One example of deterministic modelling is the “Four step demand model”, used in macroscopic modelling, in which the amount of trips generated in a network can be determined from a series of equations and some defined parameters. [7]

- A **stochastic model** contains a set of random variables or inputs that, which implies that the system outputs will be also random. These outputs are calculated several times for the different input values, obtaining a statistic distribution for their values.

In Figure 1, a distinction between deterministic and stochastic input/output diagram can be observed.

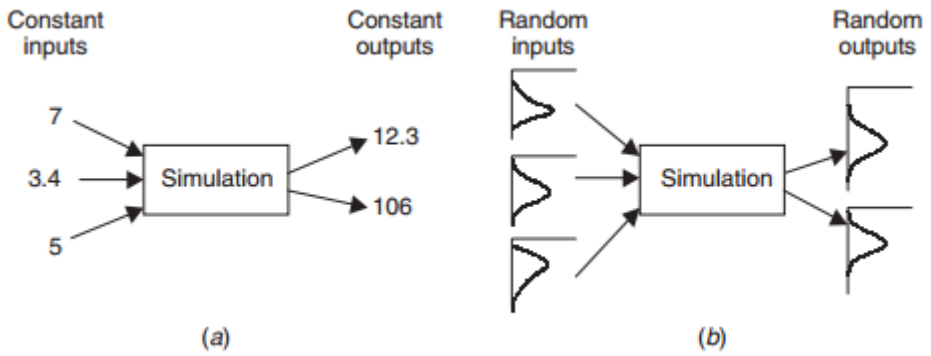

Figure 1: Examples of (a) deterministic simulation and (b) stochastic simulation [6]

Static vs Dynamic:

- A **static model** doesn't involve change in time. It analyses a system in a specific point of time, with some specific state of the variable values. One static model would analyse the current status of the variables, classes and objects contained in the simulation.

- A **dynamic model** analyses a system within the passage of time. A clock controls the progression of the simulation and the modification of time-dependent variables' values. Variable values are updated on each time step.

Discrete vs Continuous:

- A **discrete model** is that in which variables are changed in specific, discrete points of time.

- A **continuous model** will change state variables continuously over time. Differential equations are often used to describe the change rate of the variables.

In this case, transport simulator SUMO, which is presented in next sections in further detail, states as an example of both cases [8]. Here, vehicles are modelled in a continuous-space and discrete-time way, where time is divided in different simulation steps, but position (space) is a continuous magnitude that may contain any value.

In Figure 2, a comparison between the evolution of discrete and continuous variables over time is shown.

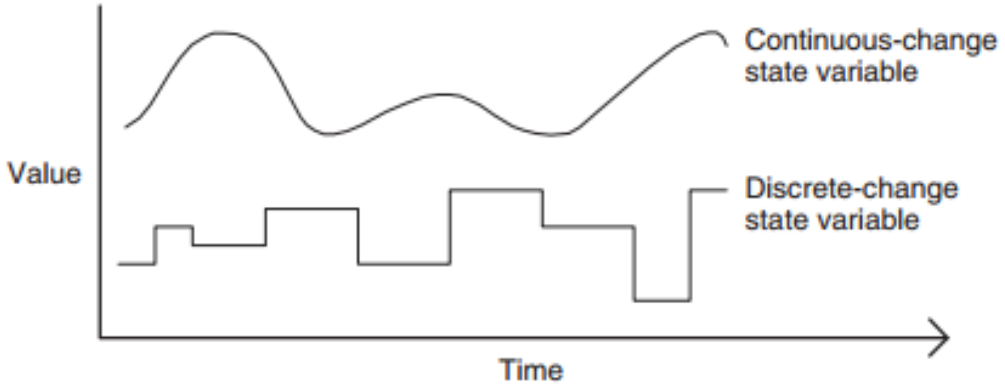

Figure 2: Comparison of a discrete-change state variable and a continuous-change state variable.[6]

In the following subsections, an overall approach towards traffic and driving simulators is presented. Explanations regarding their fundamental concepts and the main related projects carried out in the recent years concerning the integration of traffic simulators and 3D driving environments, as well as existing driving simulators, are given.

2.2 Traffic Simulators

Traffic simulation models can be arranged in three different groups, regarding the level of detail of analysis that is carried out. These three categories are: macroscopic, microscopic and mesoscopic. Special interest on microscopic models is being deposited, since they seem more relevant for this thesis purpose. For further details, reference to [9][10] or related literature presented in this section should be made.

2.2.1 Macroscopic traffic simulation models

Macroscopic simulation models describe the evolution of traffic flows over time and space, based on the relationships between traffic volume, speed, and density, where density is defined as number of vehicles per kilometre of road. Traffic flow is represented as an aggregate (average) of the mentioned measures, instead of a group of single vehicles.

As stated in [10], the most common macroscopic model is the LWR (named after its authors, Lighthill-William-Richards), which describes this process using two basic concepts: Kinematic waves [11] and Shock waves [12], which are defined by a set of differential equations.

Some newer models, like Daganzo's Cell Transmission Model are also used [13], [14]. In this model, the road is discretised into cells. The traffic volume on each cell is considered, and the

number of vehicles crossing the boundaries between two adjacent cells after each simulation step is calculated. Therefore, this model can be seen as a discretization of the previous, LWR. Some years later, in 1999, an optimization to this model was introduced by Dangazo himself, where relation with density values of the cell in previous time is incorporated. [15]

The main advantage of macroscopic models is that, since they have fewer parameters than mesoscopic or microscopic models, they achieve a higher computational efficiency. However, this computational efficiency is counteracted by the low level of detail.

Currently, one of the software which is ruling the macroscopic transport planning market is PTV VISUM [16]. However, this software is not available for free and a license must be paid.

2.2.2 Mesoscopic traffic simulation models

A third class called mesoscopic models arises as a balanced solution between the aggregated macroscopic models and the individual microscopic approach. This is explained in a higher level of detail in [10].

There are several mesoscopic models. Some of them make groups of vehicles and treats them as individual entities, providing high levels of detail for this entities, but low accuracy regarding their interactions and behaviour. [17]

Another approach discretise the road links into cells, which can be empty or can contain a vehicle. Some set of simple behavioural rules determine how many cells will each vehicle move on each simulation step. [18]

Some other models implement an approach where the road is modelled as two parts: a running part, and a queueing part. Vehicles in the running part are modelled individually with their own speed while the ones at the queueing part are waiting. However, their behaviour is not detailed further. This individual speed is determined though a macroscopic speed-density function, thus combining microscopic technics (individual vehicle modelling) with aggregated macroscopic measures. [19], [20]

Mesoscopic models combine the properties of both microscopic and macroscopic simulation models. As such, mesoscopic models provide less fidelity than microsimulation tools, but are superior to the typical planning analysis techniques.

2.2.3 Microscopic traffic simulation models

Microscopic models analyse the movement of each individual vehicle contained in the network. In contrast to macroscopic models – where traffic is seen as an aggregate of vehicles, and where some differential equations are used to model them as a whole – in microscopic models, each vehicle is modelled by its own equation, a function of its position, speed and acceleration [9].

When trying to simulate driver's behaviour, two main models come to scene: car-following model, lane-changing model.

2.2.3.1 Car-following model

The interaction of the driver with the network (vehicles, obstacles, signals…) is described as a car-following model, in which every vehicle has a "lead vehicle" that influences its speed and acceleration. This can be seen in Figure 3.

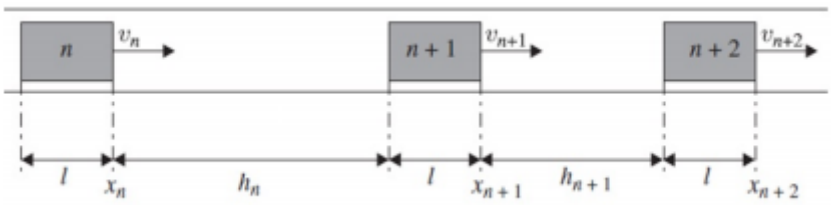

Figure 3: Numbering vehicles in car-following model [9]

Depending on the situation and the way the lead vehicle is influencing the vehicle, three different states can be distinguished, according to [9] :

- **Free Traffic State**: Vehicle density is low enough that the current vehicle can drive at its desire speed without any influence of a lead vehicle.

- **Following State**: current vehicle's speed is influenced by its lead vehicle. It is the most common situation in everyday traffic. Vehicle density is high enough to be considered. Headways (or gaps) between vehicles are shortened. Therefore, the lead vehicle has an influence on the current vehicle speed and acceleration.

- **Braking (also called Emergency) State**: the lead vehicle is stopped or its speed is lower than the speed of the current vehicle. Therefore, the current vehicle would start to decelerate in order to avoid collision.

Several car-following models have been developed over time. One of the most classic models was developed by Gipps in the 70's. [21]

One disadvantage of the first models was that some parameters were not properly calibrated and had unrealistic values. New models, like the Intelligent Driver Model (IDM), by Treiber, Hennecke and Helbing, try to solve this. [22]

2.2.3.2 Lane-changing model

Modelling roads with a unique lane on each direction can be quite straightforward. However, the majority of the roads have two or more lanes. When a vehicle changes from one lane to another, it has a relevant impact on the traffic flow.

Talking about the purpose of which the lane change is done, there are two basic types of lane changes, as explained in [23]:

- **Mandatory Lane Change (MLC)**: drivers are determined to change lane because it is necessary to follow their path. Example: the driver wants to turn left in next intersection and there is an existing left-turn pocket for this purpose. The lane change is compulsory.

- **Discretionary Lane Change (DLC)**: drivers consider that changing lane would offer a higher utility – this is, better traffic conditions – such as not following the truck driving ahead because it is driving under driver's desired speed.

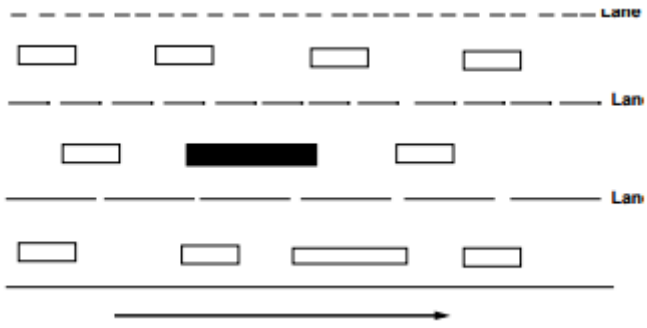

Figure 4: Process of lane change [23]

When performing a discretionary lane change (DLC), the driver believes the adjacent lane offers a higher utility. Gipps proposed in [24] the following equation, in order to determine if the current vehicle desired speed is achievable in the current lane:

$$V_n(t+T) = b_n T + \sqrt{b_n^2 T^2 - b_n(2D_x(t) - V_n(t)T - \frac{V_{n-1}^2}{b})} \qquad (1)$$

Where:

- **$V_n(t+T)$** is the maximum safe speed for vehicle n respecting its preceding vehicle at time t+T.
- **T** is the time between before and after the lane change.
- **$V_n(t)$** is the speed of this vehicle, and $V_{n-1}(t)$ is the preceding's.
- **$D_x(t)$** is the distance between vehicles.
- **b_n** is the braking that driver is willing to undertake.

Therefore, the driver will change lane if D_x is higher than the existing gap between his vehicle and the preceding one.

2.2.3.3 Microscopic simulators

Each type of transport simulation has its own characteristics which may be suitable for a specific case of study or not.

The **main advantage of microscopic** simulation is its capability to model and study each individual vehicle, providing a vast level of detail of the study network. However, this individual modelling implies the resolution of differential equations for each of the vehicles, requiring a huge computational power. However, while this problem could be an important issue when traffic simulation practices started some decades ago, it has been decreasing its relevance over time due to technological improvements and more powerful computers. Although nowadays this could still be a problem when talking about very large networks, the importance of this issue has decreased.

While microscopic simulators offers a higher level of detail and data collection from the network, requiring more powerful computational systems, macroscopic and mesoscopic simulators offer lower levels of detail, in compensation to lower computational requirements. Mesoscopic simulators stand as a half-way solution between micro- and macroscopic simulators, which can also be interesting in situations where individual data is not required but some more accuracy is requested.

All in all, each transport planner should determine which type of simulator suits his study purpose and resources available in a more complete way. In order to develop a Driver-Centric simulator where non controlled vehicles perform realistic movements, a **microscopic simulator must be chosen.**

In the past decades, several microscopic simulation tools have been developed. Some of them are the result of research projects and are currently available as open-source tools, while others are exclusively commercial and a license should be paid in order to use them.
In this section, some of the more relevant are mentioned:

2.2.3.3.1 TSS – AIMSUN

AIMSUN [25] is a hybrid traffic modelling software which allows to simultaneously apply multimodal microscopic and mesoscopic analysis to large networks, being able to model each area with a higher or lower level of detailed as desired. [26] With almost 4,000 licenses worldwide, is managed by the company TSS – Traffic Simulation Systems. It has a detailed GUI with also allows a 3D representation of the simulation.

2.2.3.3.2 PTV – VISSIM

VISSIM [27] is one of the microscopic simulators that is ruling the market nowadays, with more than 7,000 licenses all over the world. It allows to model multi-modal traffic flows in urban areas as well as motorways and rails. It also includes a complete GUI with a toggle button enabling 3D perspective for the simulation process.

2.2.3.3.3 Quadstone – PARAMICS

PARAMICS [28] is another commercial microscopic multimodal simulation tool. It allows traffic and pedestrian simulation and allows to model large networks and offers detailed reports of Measures of Effectiveness. Similarly to the above mentioned, it also includes the possibility to enable a 3D representation of the simulation in the GUI.

2.2.3.3.4 MOTUS

MOTUS [29] is an **open source** Java-based package for microscopic simulation. It does not include a GUI for network building but includes a GUI for simulation visualization. The simulation can be easily integrated and run in Matlab. Its Java-class structure allows to include new algorithms, technologies or models just by class inheritance, which makes it a suitable tool for researchers. As an interesting feature, it includes the possibility to include Road-Side

Units and On-Board Units (RSU and OBU, respectively), which can be configure to affect drivers behaviour.

2.2.3.3.5 SUMO

SUMO – acronym for **S**imulation of **U**rban **Mo**bility – is a microscopic, multi-modal (capable of simulating different transport modes), open source simulator that has been developed for more than 15 years, especially by the Institute of Transportation Research (IV) at the German Aerospace Centre (DLR), although it keeps adding contributors to its development.

It allows to simulate more than 100,000 vehicles in a network of more than 10,000 multi-lane edges (streets, roads), distinguishing between several vehicle types and being able to define different priority-ruled or traffic-signal-controlled junctions.

Since it is a microscopic simulator, each vehicle is considered individually, with their own driver behaviour, speed and route. Therefore, it bases its simulation in two main microscopic models, introduced previously in this section: Car-Following and Lane-Changing models. The implemented Lane-Changing model is explained in [30], while the different Car-Following models that can be used by SUMO are presented in [3], [31], [32].

One remarkable feature is the implementation of the TraCI protocol [33], which allows the data retrieval and remote control simulation through a TCP connection, It offers two versions of execution, with or without GUI.

2.3 Driver centric simulators

When talking about driving simulators, most of them are included in Driver-Centric simulators, which offer a perspective from the driver inside de vehicle, with the dashboard, steering wheel, the windshield, and some other items such as mirrors.

Driving simulators started to be applied in the research field of driver behaviour studies in the 1960's. However, the vast development and progress of computer technology soon contributed to a fast increase in the number of studies in this field and the development of more advanced and realistic graphic environments. [34]

Driving simulators nowadays are used in different areas of Transport Planning and Intelligent Transport Systems, in order to assess driver's behaviour in different situations. These tools are often called by many authors as "**Serious Games**" (SG).

A **SG** is a simulation tool that employs the videogame technologies in order to achieve some training or researching purpose, not necessarily related to the idea of having fun. These main purposes, in the context of driving simulators, are categorized in [35], where deeper definition of the concept of SG is also provided:

- **Test and evaluation of acceptability of new network elements**. They could be used to test the design of a new road. However, this should not be oriented in a way of the functionality of the road for a specific traffic flow, but a perspective from the driver's point of view.

- **Training:** As mentioned in [34], first simulators were used for training in war machinery. This can also be applied to civil purposes. Some examples are [36], where an improvement of driving ability was achieved in a group of neurologically impaired persons, or [37], where improvement in performance while driving was also achieved in a group of elderly people.

- **Evaluation of new ADAS** (Advanced Driving Assistance Systems) and **IVIS** (In-Vehicle Information Systems)**:** These systems can operate at the same time that the driving task, and therefore, it is necessary to measure their impact on driver's behaviour. Authors in [2] developed an evaluation platform for testing different factors that can jeopardize the driver's performance while interacting with specific IVIS such as a Head-Up Display.

- **Impaired driving behaviour:** It has already seen, when talking about training purposes, that impaired people could benefit from this tools in a way where their driving skills where improved. However, this tools can also modelled situations in which the driver is physically or mentally impaired, such as being under the effects of drugs [38], alcohol [39], or just fatigue.

- **Other external distractions**: Some other factors that can arise as a potential source of distraction are the situations in which the vehicle is not only carrying the driver, but other passengers. One example can be seen in [40], where the authors studied the driver's performance when carrying a baby aboard, in different situations such as "crying baby", "asleep baby" or "awake baby".

In conclusion, **Safety** could be seen as an overall purpose. Driving simulations, either used for training, studying behavioural aspects or testing new technologies, aim to find solutions in order to improve driver's safety and driving performance.

Driver-Centric Traffic Simulators

In previous section 2.3., some works in which driver centric simulators were used in order to evaluate different driving situations and behaviours were presented. However, they lacked tools for generating a realistic environment where **NPC** (Non-Player Controlled) vehicles follow microscopic traffic models.

Previous works have been done in coupling traffic simulators together with other tools. One of the most common approaches in the field of ITS (Intelligent Transportation Systems) in the recent years was the development of the so-called “VANET simulator”. This simulation tools consist on the coupling of one traffic simulator together with a network simulation. Many examples can be found, with different traffic and network simulators involved [41], [42], [43].

Authors in [44] state the lack of a VANET simulator with 3D visualization features, and present a first approach for their own 3D Visualization system, although they state the need for future improvements in the models used. However, any attempt in developing a driver-centric perspective is shown.

Recent work from the Royal Institute of Technology of Stockholm [45] developed a way of communicating between the traffic simulator SUMO and the graphic 3D engine Unity3D in order to satisfy the need to a realistic visualization of traffic in urban environments. For this purpose, the TraCI protocol was used in order to communicate SUMO and Unity3D. Nonetheless, any approach to driver centric perspective was carried out.

A Driver-Centric perspective with a windshield HUD implementation is made in [46]. In this work, the SUMO simulator is used to generate traces of the vehicle movements, which are used by Unity3D to generate the traffic. However, this implies that any direct communication between SUMO and Unity3D is being done, since traces are processed offline. Therefore, driver’s behaviour cannot influence the rest of the vehicles.

On the other hand, there are some works in which our goal of synchronising the driver centric simulator with a microscopic simulator is achieved. Furthermore, in this tools, a coupling with a network simulator is also achieved. Therefore, complete interaction between the driver and the traffic network is reached, including the possibility to test applications based on vehicular communication.

This Driver-Centric simulator is presented in [47], and is developed using OpenSceneGraph as the 3D Engine, DIVERT as the traffic simulator, and NS-3 as a network simulator. In this work, authors used this tool in order to implement and evaluate Virtual Traffic Lights and See-

Through System [48] applications inside the vehicle. A diagram of the architecture of the simulator can be found in Figure 5.

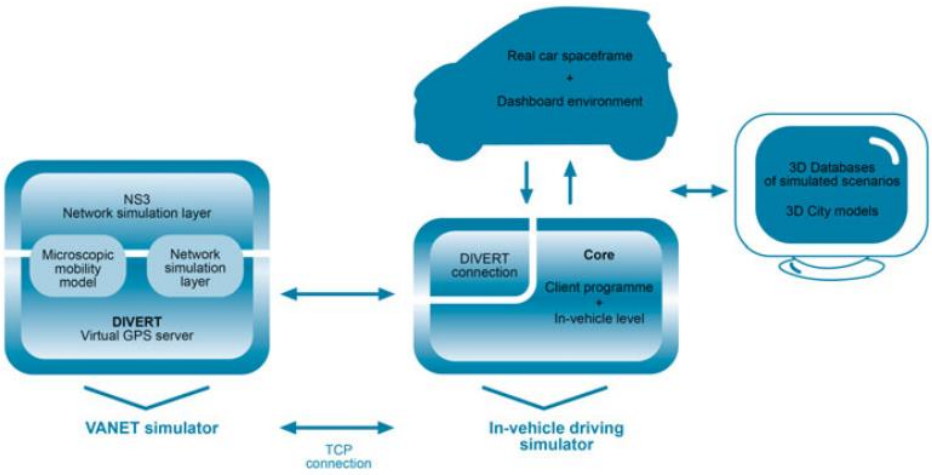

Figure 5:Architecture of the driving simulator[47]

Finally, a very recent work, which was published during the development of the current bachelor thesis, presents ITDNS [49], an integrated traffic-driving-network simulator. Here, the commercial traffic simulator PARAMICS is used together with the network simulator NS-2 and a non-specified driving simulator for testing and evaluating Cyber-Physical – also called Cooperative – Transportation systems. An overview of the ITDNS structure can be seen in Figure 6.

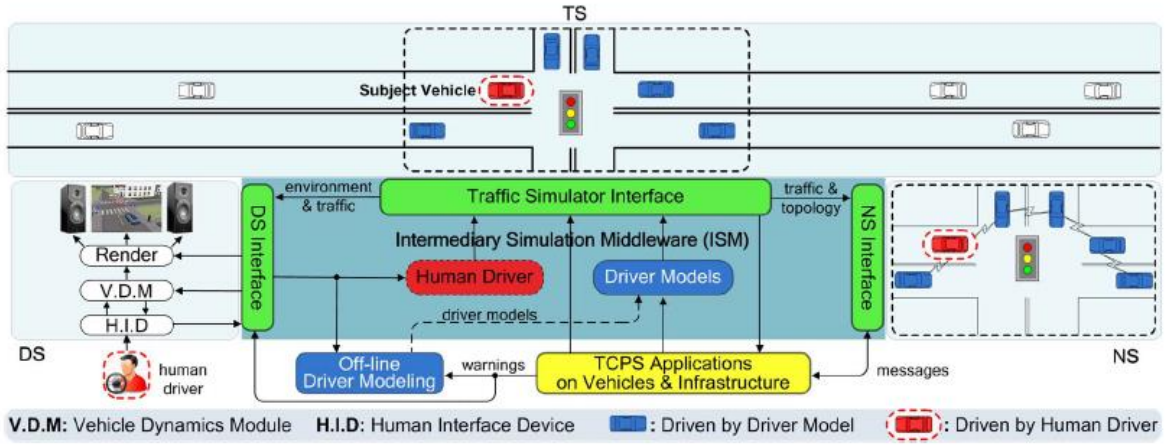

Figure 6: Overview and structure of the ITDNS Simulator [49]

3 Implementation

In this section the implementation of a **Microscopic Driver-Centric Simulator** is described. First, some conclusions regarding the state of the art and related literature presented in previous section are made. Then a series of different tools which are used for the development of this Driver-Centric simulator is detailed.

First, taking into account the different literature and related work in the area presented in previous section, in this section some conclusions were extracted about the advantages and disadvantages of simulation tools to define the requirements for the development process.

The simulation process presents a variety of **advantages** respect to field-test analysis and implementation, which are detailed as follows:

- **Flexibility and fast feedback**: Different parameters and/or models can be configured and reconfigured easily without any "in-field" modification – One would avoid manufacturing a prototype every time a modification is made, for example.

- **Safety**: The case of transport simulation states as a perfect example for this property. Testing a traffic scenario in real life – for example, testing the performance of a new ADAS – endangers the testing driver as well as the rest of drivers. Real testing may take place when simulation results prompt safe values.

- **Different levels of detail**: The system can be seen with different levels of detail, modules containing smaller modules or components containing more simple components. This allows the user to understand the system of study easily. In the case of traffic simulators, the network can be seen and modelled with different levels of detail (macroscopic, mesoscopic or microscopic).

  One simulation can be run with different speeds. One may need to slow down the simulation in order to understand the behaviour of the system with a higher depth, while it could also be speed up in order to study a long period of time.

- **Illustrative**: Simulation allow to obtain and show results in a quick, illustrative way. Good results of a simulation study could awake the interest of a potential customer or investor.

- **Cost-saving**: As a consequence of the points above mentioned. The flexibility to change the current scenario without having to modify any real implementation or

prototype, in a fast way and without any personal risk, constitute a cost-saving tool in terms of time and money.

However, simulation also has its **disadvantages**, which should also be borne in mind. These drawbacks are mostly related with **computational** and **processing capabilities**.

Big simulation scenarios can demand vast amounts of computational resources, causing delay in the simulation. This would constitute an important problem in case the data which is being simulated is expected to be used by other program running at the same time. For the Driver-

When talking about a Driver-Centric simulator, two main parts can be distinguished. First, one **microscopic traffic simulation tool** which can generate realistic traffic flows based on well-known, reliable, complex microscopic traffic models.

Because of its open-source availability and the vast amount of different works that have been carried out with it, the simulation tool **SUMO was chosen**. Data from this tool is extracted through TraCI Protocol and processed by a 3D Graphic Engine.

Also due to its availability without payment (two versions are available, the personal – free – and the professional edition – under paid license – but the features of the graphical engine are not restricted in the first mentioned), its powerful engine, the enormous amount of documentation that can be found on the Internet – including a hole community for support - and the numerous works made with it, **Unity 3D stands as a suitable tool** for developing our driving simulator and therefore was also chosen.

3.1 Generating the microscopic simulation scenario with SUMO

In this section, the process followed to create the different microscopic simulation scenarios is described. This process is detailed and at the same time explanations about basic concepts regarding the different elements that conform the simulation are given. Finally, the TraCI communication protocol is explained.

Installing SUMO

Latest developed version of SUMO is 0.26. However, this version was launched during the elaboration of this work, thus it was not used in the current project. Instead, **version 0.25 was**

chosen. All different versions can be found and downloaded from SUMO page in SourceForge [50].

The latest version can also be downloaded from the SUMO webpage (http://sumo.dlr.de/wiki/Installing), where instructions for installing in different Operative Systems can be found as well.

SUMO simulation package includes all the different tools needed to prepare a traffic network – roads, vehicles, traffic lights… – and simulate it. All the included applications can be seen in Table 1.

NAME	DESCRIPTION
SUMO	Command line application for simulation execution and visualization.
SUMO-GUI	Graphic User Interface for simulation execution and visualization.
NETCONVERT	Network importer and generator; reads road networks from different formats and converts them into the SUMO-format
NETEDIT	A graphical network editor.
NETGENERATE	Generates abstract networks (grid, spider-net, random…)for the SUMO-simulation
DUAROUTER	Computes routes through the network.
JTRROUTER	Computes routes using junction turning percentages
DFROUTER	Computes routes from induction loop measurements
OD2TRIPS	Generates vehicles trips from O/D Matrices
POLYCONVERT	Imports points of interest (POIs) and polygons from different formats allowing to be visualized by SUMO-GUI
ACTIVITYGEN	Generates a demand based on mobility wishes of a modelled population
MESO	Mesoscopic simulation tool implementing queue-model, being up to 100 times faster than the pure microscopic simulation. Not relevant for our project.
MESO-GUI	The above mentioned tool for mesoscopic simulation with a graphical user interface.

Table 1: Applications contained in the SUMO package, extracted from [8]

In order to work with SUMO command line in a more comfortable way, **the PATH variable** inside the computer's environment variables was configured. Configuring this, SUMO can be called from the command line just writing "sumo", or "sumo-gui" for the graphic version, without needing to reference the folder in which the program is contained.

For this purpose, the following steps were made in a Windows system:

1. From the desktop, right-click **My Computer** and click **Properties**.

2. In the System Properties window, click on the **Advanced** tab.

3. In the Advanced section, click the **Environment Variables** button.

4. Finally, in the Environment Variables window, select the **Path** variable in the Systems Variable section and click the **Edit** button.

5. **Append the directory** where sumo.exe is located (e.g. C:\Program Files\sumo-0.25.0\bin), preceded by a semi-colon (;).

3.1.1 Defining and implementing the road network

The traffic network in SUMO is defined by the SUMO network file. This file has an extension .net.xml specific of SUMO. An **X-Y coordinate system** is used to reference the position of all the elements of the network. The origin of this system is the (0,0) point, and positive Y axe is oriented to the north (therefore, the other 3 cardinal points are oriented subsequently). This must be taken into account when manually defining a network or when trying to analyse one existing network.

Two different networks were created in order to be used as scenarios for the Microscopic Driver-Centric Simulator. These **scenarios** can be found in the attached folder "Scenarios", and are the following:

- **Technikum**: A real representation of the area nearby FH Technikum Wien. Network was automatically generated from OSM data using the NETCONVERT tool included in SUMO package.

- **Grid**: A simple grid-like network consisting of perpendicular streets creating intersections controlled by Traffic Lights.

The process followed for creating the networks corresponding to both scenarios presented above is detailed in the current section. However, in order to provide some fundamental explanations about the way a network is defined in SUMO, its main elements have been also explained in order to reinforce the understanding of the files that are being used.

3.1.1.1 The elements of the network

Junctions: also called nodes in Transport Planning, represent intersections of different edges. They can work under priority rules or under traffic lights control. This is specified in the attribute "**type**".
An example XML definition of a junction is shown in Figure 7. Here, the sample junction works under right-of-way priority rules. This junction corresponds to one of over two thousand junctions contained in the generated network Technikum.net.xml.
The **request** elements describe for each link, denoted by "index", which traffic streams have higher priority (therefore, which streams will make a vehicle on link "index" stop).

```
<junction id="1170707016" type="priority" x="1017.47" y="1502.84" incLanes="
30283125#1_0 30283125#1_1" intLanes=":1170707016_0_0 :1170707016_0_1" shape="
1017.57,1506.89 1021.33,1501.59 1020.97,1499.44 1019.31,1498.32 1017.24,1498.
70 1013.50,1504.01">
    <request index="0" response="00" foes="00" cont="0"/>
    <request index="1" response="00" foes="00" cont="0"/>
</junction>
```

Figure 7: Example XML definition of a junction in SUMO for the city of Vienna – extracted from Technikum.net.xml

Edges: connection between two nodes. They represent the streets (in urban scenarios) or roads (general scenario). They are **unidirectional**, therefore, 2 edges may connect the same pair of nodes, making the distinction in the "from" and "to" attributes of the edge. Each edge can contain one or more lanes.

Lanes: Each lane can specify the vehicles that are not allowed to use it, the maximum speed (in metres/second), its length and its width. Shape attribute is a collection of (X,Y) pairs that define the central line of the geometry of the lane. An example XML definition of an edge with 2 lanes is shown in Figure 8. This edge is part of the generated network Technikum.net.xml, and corresponds to one segment of Dresdner Straße, in Vienna.

```
<edge id="387105838#0" from="3904409040" to="3904409038" name="Dresdner Straße"
 priority="7" type="highway.secondary" spreadType="center" shape="1285.50,828.
20 1297.53,813.08 1357.04,737.18">
    <lane id="387105838#0_0" index="0" disallow="tram rail_urban rail
    rail_electric ship" speed="13.89" length="105.71" shape="1287.51,823.02
    1296.24,812.06 1352.80,739.91">
        <param key="origId" value="387105838"/>
    </lane>
    <lane id="387105838#0_1" index="1" disallow="tram rail_urban rail
    rail_electric ship" speed="13.89" length="105.71" shape="1290.09,825.08
    1298.83,814.10 1355.40,741.94">
        <param key="origId" value="387105838"/>
    </lane>
</edge>
```

Figure 8: Example definition of an Edge with two lanes in SUMO for the city of Vienna – extracted from Technikum.net.xml

Connections: Describe the allowed connections within two lanes of two different edges. **From/to** attributes refer to the incoming and outgoing edge, while **fromLane/toLane** refer to the specific lanes of the mentioned edges.
This connection might be also ruled by a traffic light, in which case a **"t1"** attribute would indicate the ID of the traffic program. Attribute "**dir**" indicates the direction of the movement which is made in this connection, this is going straight (s), or turning (left: l; right: r; partially left or right: L/R), while the **state attribute** presents the status of the connection:

- If ruled by right-of-way, it indicates if it is a minor road or a mayor road (m/M),
- If controlled by traffic lights, it will indicate if its red (r), yellow (y) or green (g if there is another movement that has priority over this or G if this is the mayor movement).

An example XML definition a connection is presented in Figure 9, also from the network Technikum.net.xml.

```
<connection from="-10361262#1" to="9402267#3" fromLane="0" toLane="0" via="
:370947306_10_0" dir="l" state="m"/>
```

Figure 9: Example definition of a Connection in SUMO for the city of Vienna – extracted from Technikum.net.xml

Traffic light logics: It defines the different phases and phase sequence of a traffic controller specified with id attribute. This signal program can be static or actuated (attribute type). An example XML definition of a traffic controller for one intersection is shown in Figure 10. Each phase has its **duration**, and specifies the **state** for each of the signal heads controlled by this logic unit in the intersection, starting from the north link and continuing in a clockwise way.

For the example, an intersection from the Grid-like scenario has been chosen due to its simplicity. The graphic representation of the intersection in Figure 10 is presented in Figure

11. Incoming movements from north and south (movements 0, 1, 2 for north and 6, 7, 8 for south) have a red light, while incoming movements from east (3, 4, 5) and west (9, 10, 11) have a green light.

Through movements have priority over turning movements, therefore, movements 5 and 11 have a “g” instead of a “G”, because they need to wait for movements 10 and 4, respectively. The signal program is depicted in Figure 12, where the different phases can be observed. The screenshots were made in the second 74 of the simulation, therefore, the fifth phase is shown in the pictures.

```
<tlLogic id="2/2" type="static" programID="0" offset="0">
    <phase duration="31" state="GGgrrrGGgrrr"/>
    <phase duration="4" state="yygrrryygrrr"/>
    <phase duration="6" state="rrGrrrrrGrrr"/>
    <phase duration="4" state="rryrrrrryrrr"/>
    <phase duration="31" state="rrrGGgrrrGGg"/>
    <phase duration="4" state="rrryygrrryyg"/>
    <phase duration="6" state="rrrrrGrrrrrG"/>
    <phase duration="4" state="rrrrryrrrrry"/>
</tlLogic>
```

Figure 10: Example definition of the phase sequence for traffic light <id> in SUMO – extracted from grid.net.xml

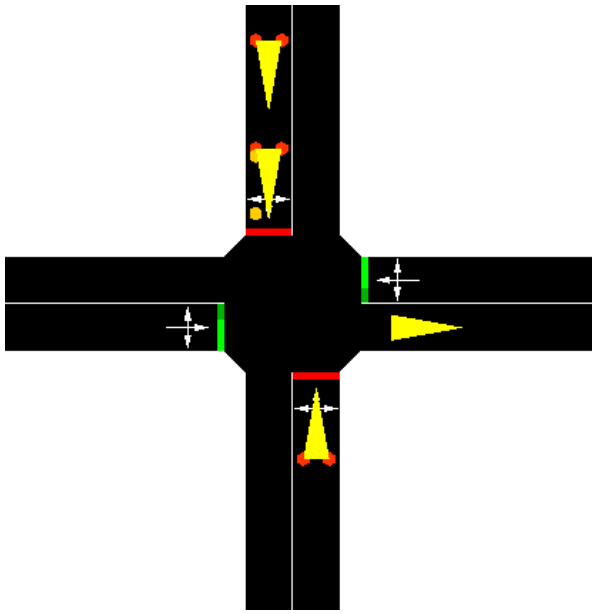

Figure 11: Example Signal-Controlled intersection. Current phase "rrrGGgrrrGGg". Screenshot made at second 74 of simulation. – extracted from GUI representation of grid.sumo.cfg

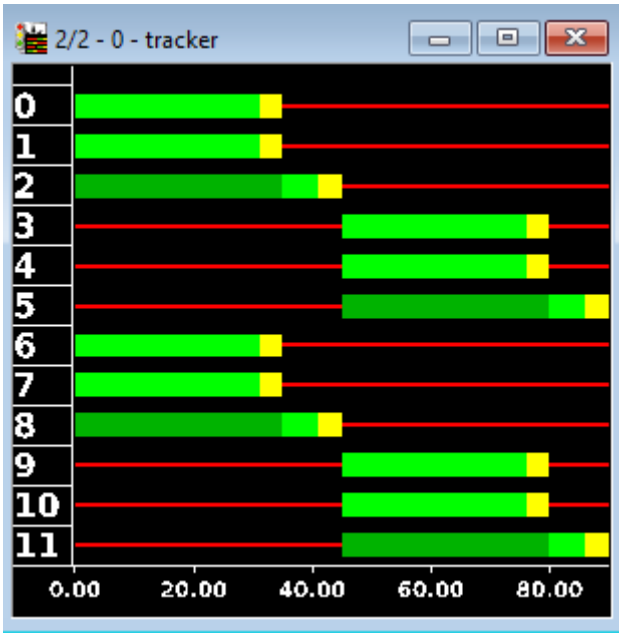

Figure 12: Signal program for the sample intersection. Diagram created with SUMO-GUI.

3.1.1.2 Generation of the network

SUMO offers a variety of ways for generating the network. The simplest, but also most tedious and not scalable way, is to manually create the XML files for structuring the network. However, as soon as the network has some tens of edges and nodes, this solution becomes a tedious, time-demanding, not-worthy and non-sense waste of time. For this reason some of the applications mentioned in Table 1 were used in the process of creating the networks.

To be specific, 4 tools have been used for creating both scenarios: NETGENERATE, NETCONVERT, NETEDIT and POLYCONVERT.

***Note**: some of the command lines presented in the SUMO Wiki contain attribute definitions which are deprecated. For this reason, sample commands are provided in this section in order to illustrate a proper way of generating these abstract networks. A summary of the commands is presented in Appendix B as well.*

3.1.1.2.1 Generation of the Grid-like scenario

This tool is used to generate abstract networks. These abstract networks are classified in three groups: grid, spider and random.

Grid networks are rectangular networks in which all the intersections are formed by perpendicular incoming and outgoing edges.

NETGENERATE allows to create grid-networks, specifying a variety of parameters such us the number of junctions per axe and the length of the edges. The mentioned **Grid network used** for testing the Driver-Centric Simulator **is shown in Figure 13**, together with the command used for its creation and a zoom into one of the intersections contained.

A grid of 6x6 edges was created, including 16 T-intersections and 16 4-street-intersections, all of them controlled by similar signal program generated automatically. The corners of the network (2-street intersections) also included traffic lights programs. However, they were unnecessary due to the lack of conflict between incoming movements. Therefore, they were deleted. The deleting process is detailed in section 3.1.2.2.3 “Editing the network”.

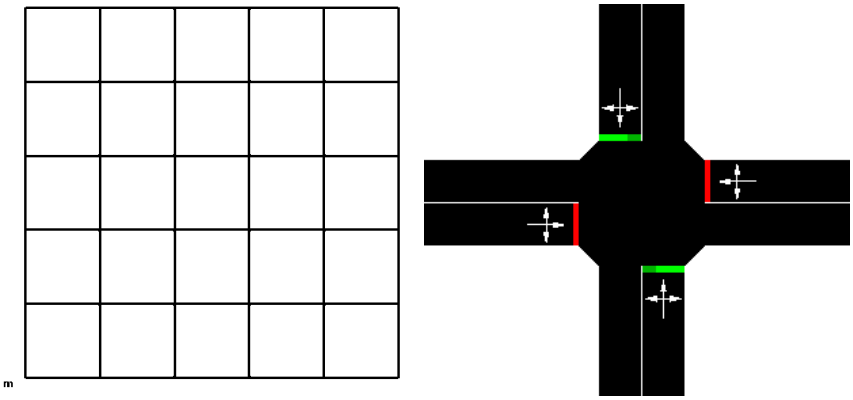

Figure 13: Example representation of a grid network generated by NETGENERATE (top left). Zoom into an intersection of the sample network (top right). Command line used for generating the grid netwotk (bottom)

```
netgenerate -g true --grid.number 6 --grid.length 80 --no-turnarounds.tls true -j traffic_light -o Grid.net.xml
```

.

3.1.1.2.2 Generation of a realistic network from OSM Data – Technikum.net.xml

Abstract networks can be useful to analyse simple scenarios, but in many cases where simulation needs the usage of realistic scenarios, they are insufficient.

Importing existing networks from different sources, accepting many different input formats, is possible thanks to the **NETCONVERT** tool.

This tool can be useful when a traffic network is available from other simulation packages such as PTV VISSIM and VISUM, software mentioned in previous Related Work section, or from other sources like **OSM (OpenStreetMap)** data.

First, OSM data has been extracted. For achieving this purpose, different options were considered:

a) Extracting the data manually from OSM webpage [51], clicking the tab “Export” and selecting the area of interest.

b) Using osmWebWizard.py. This python script is provided inside the tools folder, and opens a web browser that allows selecting the area of interest and automatically generates a .net file.

In the current work, option a) was chosen because a higher control over the converting process was desired. The desired area was defined manually as shown in Figure 14. The corresponding boundary values for latitude and longitude have been also presented.

Figure 14: (Top) Manual definition of the boundaries for the desired area from OpenStreetMap. (Bottom) Corresponding latitude and longitude boundary values.

	48.2454	
16.3741		16.3894
	48.2365	

In order to create the .net.xml file for SUMO, the NETCONVERT tool was called generating the corresponding .net.xml file. The used command line was:

```
netconvert --osm-files Technikum.osm -o Technikum.net.xml
```

It was observed and it must be bared in mind for new OSM importations, that **conversion from OSM to SUMO network is not perfect**. Sometimes, because of lack of information in the OSM data. Some other times, because the NETCONVERT tool makes an invalid interpretation of this data. Special attention must be paid to the warnings prompted by the conversion process. The received warnings for the conversion of the Technikum network are shown in Figure 15.

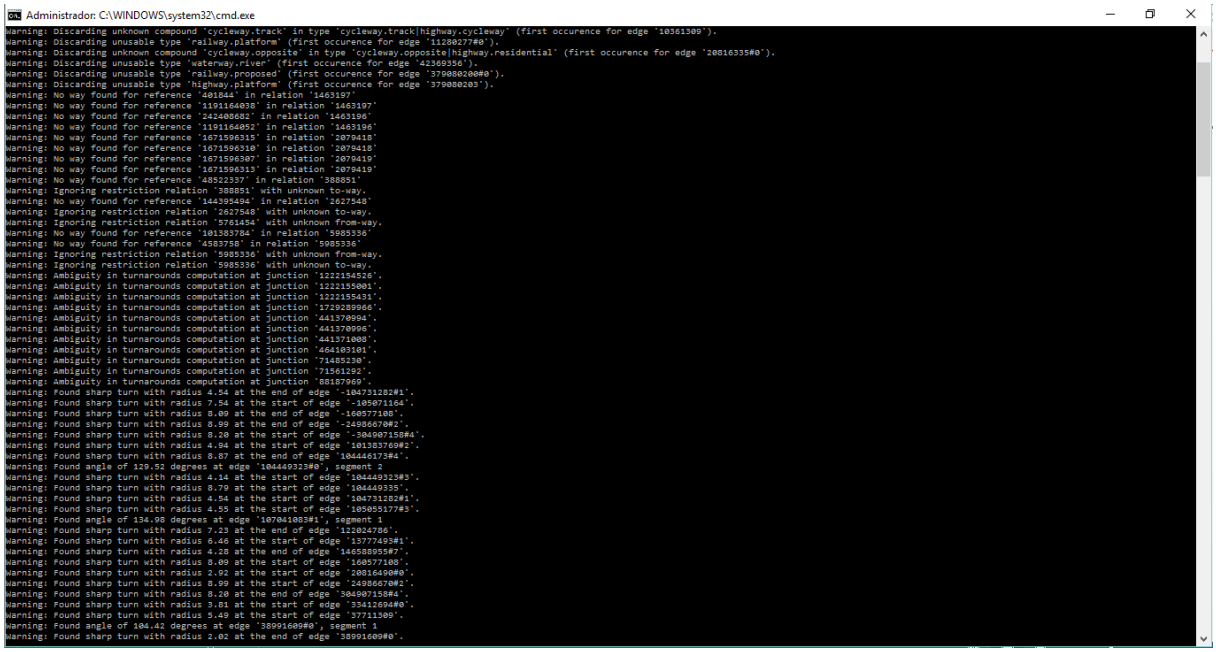

Figure 15: Warnings prompted during OSM to NET conversion

This warnings, although they are not errors, can refer to a misinterpretation of the OSM data, and are frequently due to junctions/nodes which are very close to each other, as well as edges for different modes (for example, one tram railway in the middle of two unidirectional edges). Errors of this kind were experienced during the development of the mentioned scenario. An example of problematic intersection is shown in Figure 16. The TLS configuration was checked and it was observed that the traffic lights had different unsynchronised programs.

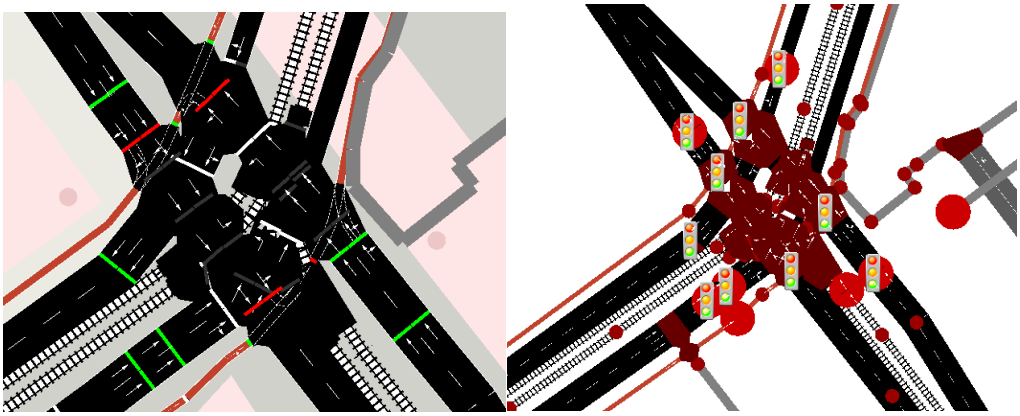

Figure 16: (Left) Sample intersection in which OSM conversion presents mistakes, corresponding to Dresdner Straße, Stromstraße and Marchfelstraße in Vienna, included in first version of Technikum.net.xml (Right) Sample intersection opened with NETEDIT, showing the traffic lights.

In order to prevent problems of this type, some extra arguments considered to improve the network conversion were included in the command line:

> ***--junctions.join***: This is used to solve those cases where, for example, a railway divides the street (Dresdner Straße). Instead of merging into a single junction, when crossing with another street, each edges creates an individual junction, resulting in several junctions close to each other which in reality are just one single junction. Whenctivating this option, the NETCONVERT tries to merge them.
>
> ***--tls.join***: This is used in relation to the previous case. Junctions which appear really close to each other, but separated, may be controlled by the same signal control in reality, but after the conversion they are assigned separate signal controllers. If unsynchronised, the result is an inoperative junction, as the example shown in Figure 16. When activating this option, the same signal controller is assigned for the mentioned junctions.

After the reconversion, the visual representation of the network was the same. However, it could be observed that the traffic lights in the mentioned junction and other junctions presented the same synchronised signal program.
OSM includes much more data rather than roads and junctions. It also includes information regarding rivers, forests, parks or buildings. Although this information was not especially relevant for the presented Driver-Centric simulator, a polygon file was generated from this information and added in the SUMO simulation in order to have a more detailed view.

This could be achieved with the tool **POLYCONVERT**. This tool requires the .osm file, the .net file generated from the first one, and one .xml file called “typemap” which is provided by the

SUMO package and includes polygon definitions for the different types of terrain. A copy from the original file was created and renamed as “typeTechnikum.xml”. The following command line was employed:

```
polyconvert --net-file Technikum.net.xml --osm-files Technikum.osm --type-file typeTechnikum.xml -o Technikum.poly.xml
```

This file was appended in the configuration file (which is explained in next section) in order to apply its effect. In Figure 17, a screenshot from the network using the generated polygons is shown. This kind of files are useful to improve the graphical appearance of the network in the GUI version, reason why they were implemented during the network elaboration process, although they were not used in the implementation of the Driver-Centric Simulator.

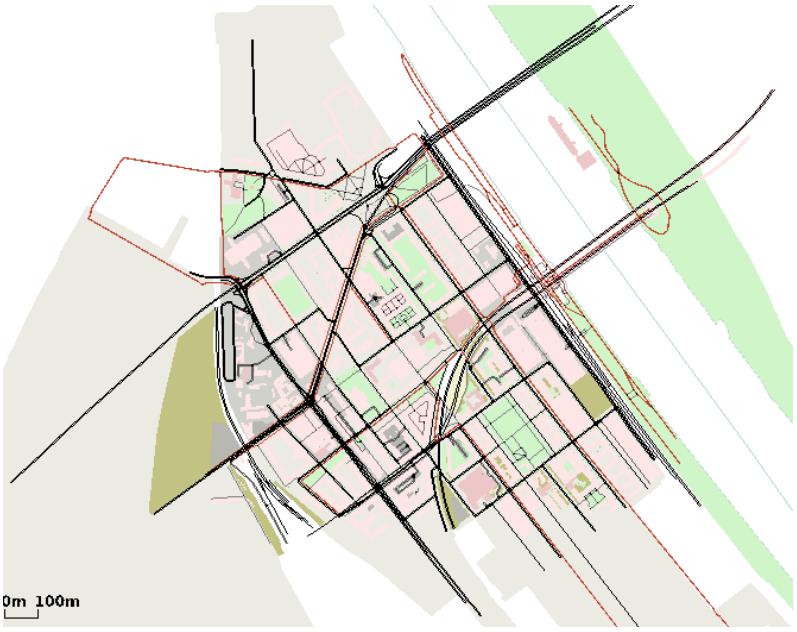

Figure 17: Screenshot from Technikum.net scenario, after applying polygons and before editing.

3.1.1.2.3 Edition of the autogenerated networks

Due to the fact that tools for auto-generating the networks have been used, the obtained networks might not have the desired characteristics. This was observed for both scenarios implemented in this work.

Extra edges, junctions, uncoordinated signal programs are some of the problems that were found in the generated networks. In order to correct them, the NETEDIT tool was used.

NETEDIT is a network editor which allows to create networks from scratch or to import one existing network and modify any of the elements. This editor includes a GUI, making the

creation and edition of networks quite user-friendly. It allows from generating simple nodes and edges, being able to configure all their different attributes, to include bus stops and design traffic signal programs for intersections.

Edition of Grid-like network

First, the simpler network, Grid.net.xml, was edited. The only corrections made were the addition of 2 single lane edges on each of the 4 corner intersections of the roads - one single lane edge in every corner in order to create some reference points for starting and another one to ending the routings, avoiding that vehicles might appear/disappear in the middle of the road – and 2 single lane edges in the centre of the network for starting the driving simulation there.

The early mentioned were assigned IDs named after their relative position, start_CP for the incoming lane and end_CP for the outgoing lane, where CP stands for cardinal point (in this case, NE, NW, SW and SE). The 2 edges in the middle of the network where assigned the IDs start_sim and end_sim. This IDs are very important in order to create the 3D scenario.
In Figure 18, a comparison of the network before and after editing is presented.

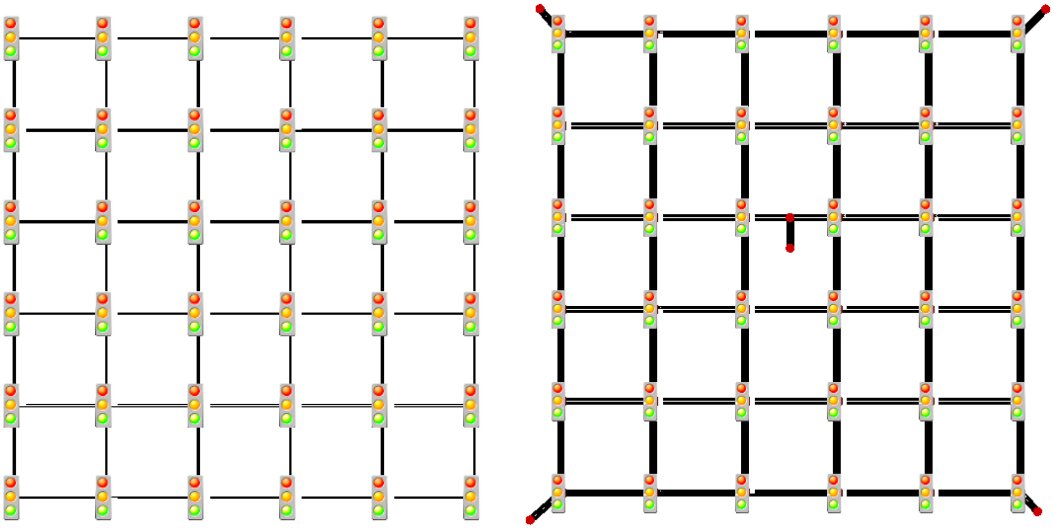

Figure 18: Grid.net before (Left) and after editing (Right)

Edition of OSM imported network – Technikum.net

In Figure 19, the Technikum.net network generated from OSM data is shown inside the NETEDIT application before any modification. It was observed that several modifications should be made, such as deleting irrelevant edges in order to use less resources while simulating.

Figure 19: Road network generated from OSM data of the surroundings of FH Technikum Wien, before being edited.

After the edition, a cleaner version of the network was obtained. It is presented in Figure 20. The modifications made to the network were the following: deleting most of pedestrian and bicycle lanes, since no pedestrian or bicycle simulation is done in the current work; delete the railways corresponding to the “Schnell Bahn”; deleting external roads in order to give a more rectangular shape to the network. Some railways corresponding to the tram that travels through Dresdner Straße were left in order to maintain the original signal program logics for some of the intersections (since deleting one incoming lane will delete the signal state that was assigned to that lane in the signal program).

Similarly to the previous scenario, two edges, called start_sim and end_sim, have been created in order to be the origin point of for the controlled vehicle in the Driver Centric Simulator.

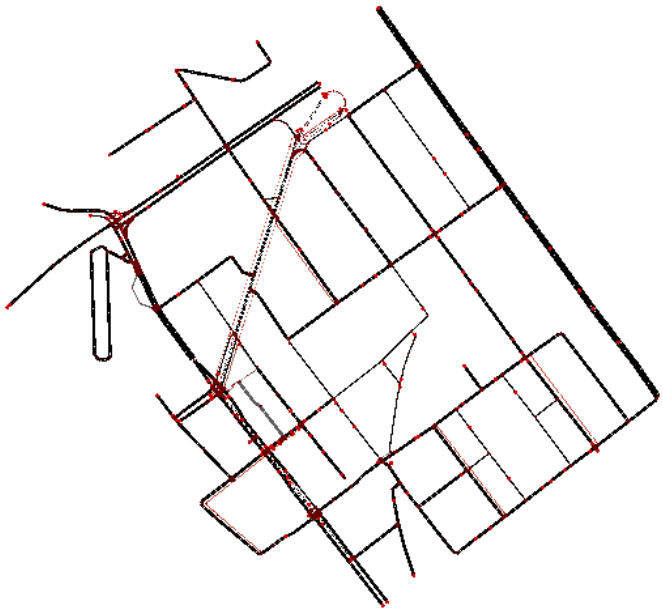

Figure 20: Road network generated from OSM data of the surroundings of FH Technikum Wien, after edition

3.1.2 Demand Modelling in SUMO

Up to this point, generation of the road network has been presented. However, any reference to vehicles, trips or routes has been done yet. Together with the .net.xml file, at least another file is needed in order to model one scenario. This is the .rou.xml file, which contains information about vehicles, their types, and their routes.

Distinction between two essential concepts must be remarked.

- A **trip** is a vehicle change of location, defined by vehicle's starting and ending edge, as well as the departure time.
- A **route** is a detailed description of a trip, which contains the internal edges that the vehicle will pass.

Trips are not enough for simulating a network in SUMO. Routes and vehicles can be generated manually by XML definition or automatically, using one or more of the several tools included in SUMO for this purpose.

Vehicle flows were assigned manually for the grid-like network scenario and then the routes were automatically generated with the DUAROUTER tool. On the other hand, the vehicle

inputs and their routing for the OSM urban scenario were generated in a completely automatic process.

3.1.2.1 Generation of vehicle flows and routing for Grid scenario

3.1.2.1.1 Manual definition of trips and flows

Vehicles and their routing are included in the simulation inside the .rou.xml file. In this file, routes, vehicle types and vehicles can be defined. The example .rou.xml file provided by SUMO located in the tutorial folder is shown in Figure 21. The file is structured as a series of xml tags with a parent-children relationship between them. All tags are children from the <routes> tag, and can have children at the same time.

- **<vType>** stands for Vehicle Type and allows to predefine certain vehicle types with common characteristics such as length, acceleration, speed or passenger capacity (among many others).

  Although in the example there is not any **vClass** attribute, it is also worth mentioning. This attribute represents an Abstract Vehicle Class and is used in lane definitions in order to allow or prohibit the usage of that lane for certain vehicle types. Some examples are “passenger” (default value), “tram”, or “bicycle”. Therefore, a railway for trams will only allow trams using it.

- **<route>** represents a series of edges that will be passed by one or more vehicles. They can be defined separately as in the example, with an id, so they can be used by different vehicles, or they can be defined as children of a vehicle tag, in which case this route would be specific for that vehicle.

- **<vehicle>** defines a specific vehicle that will drive in the network since the simulation step “depart” and that will follow the route which has been assigned to it. The type attribute allows to assign a predefined vType to configure specific parameters of the vehicle behaviour.

```
<routes>
    <vType accel="1.0" decel="5.0" id="Car" length="2.0" maxSpeed="100.0" sigma="0.0" />
    <route id="route0" edges="1to2 out"/>
    <vehicle depart="1" id="veh0" route="route0" type="Car" />
</routes>
```

Figure 21: Example .rou.xml file included in SUMO package

As explained before, SUMO needs routes to carry out the simulation. However, some "incomplete routes" definitions can be made: trips and flows. An example definition of each of them can be seen in Figure 22.

```
<trip id="t" depart="0" from="beg" to="end"/>
<flow id="f" begin="0" end="100" number="23" from="beg" to="end"/>
<flow id="f2" begin="0" end="100" number="23" from="beg" to="end" via"e1 e23 e7"/>
```

Figure 22: Example definition of trips and flows – extracted from SUMO Wiki [8]

- The **trips** were explained previously. Exclusively the departing time, the origin edge and the destination edge is provided.

- One **flow** represents a series of vehicles that will be generated periodically during the specified period between "begin" and "end" attributes. The attribute "number" indicates the amount of vehicles that will be generated during this period. Also a "type" attribute could be incorporated in order to specify the type of the vehicles that will be generated.

Both elements have the possibility to include a "via" attribute, in which specific edges which must be passed can be determined.

A .flows file was manually written, where the vehicle flows were defined, as it can be seen in Figure 23.

```
<flow id="0" from="start_SW" to="end_NE" via="2/2to3/2 3/3to2/3" begin="0" end="1000" number="20"/>
<flow id="1" from="start_SE" to="end_NW" via="3/2to2/2 2/3to3/3" begin="0" end="1000" number="20"/>
<flow id="2" from="start_NW" to="end_SE" via="2/3to3/3 3/2to2/2" begin="0" end="1000" number="20"/>
<flow id="3" from="start_NE" to="end_SW" via="3/3to2/3 2/2to3/2" begin="0" end="1000" number="20"/>
<flow id="4" from="start_SW" to="end_NE" via="2/2to3/2 3/3to2/3" begin="1001" end="4500" number="50"/>
<flow id="5" from="start_SE" to="end_NW" via="3/2to2/2 2/3to3/3" begin="1001" end="4500" number="50"/>
<flow id="6" from="start_NW" to="end_SE" via="2/3to3/3 3/2to2/2" begin="1001" end="4500" number="50"/>
<flow id="7" from="start_NE" to="end_SW" via="3/3to2/3 2/2to3/2" begin="1001" end="4500" number="50"/>
```

Figure 23: Flows of vehicles defined for the Grid scenario – extracted from Grid.flows.xml

The edges "start_xy" and "end_zv", where x, y, z, and v stand for the cardinal points, are the extra edges mentioned in section 3.1.2.2.3. Some additional edges where defined in the "via" attribute, meaning that passing through this edges is compulsory when generating the routes.

The selected flows were thought to be a good assignment for testing, since vehicles use the inner streets of the network in order to reach their destination. This way, the effect of the traffic lights control was observable. In case the "via" attribute was not included, vehicles would travel through the external edges, and the driver-centric experience would not be enough challenging.

Two intervals were distinguished. The first 1000 seconds of simulation, a higher level of traffic was included in the network. 20 vehicles are generated from each starting point, making a total of 80 vehicles generated in the first 1000 seconds. Each starting point generates a vehicle every 50 seconds. The rest of the simulation period, 3500 seconds, generates a total of 200 vehicles. Each starting point generates one vehicle every 70 seconds.

This was made in order to test the performance of the graphic engine when representing a higher number of elements.

3.1.2.1.2 Automatic generation of routes

The tool DUAROUTER, included in SUMO package, was used to compute the input flows and automatically generate the routing of all the vehicles. The obtained file received the name of "Grid.rou.xml". Also a rou.alt.xml file was created, which contains a distribution probability for each vehicle route, but was deleted because it was not considered useful.

3.1.2.2 Generation of vehicle flows and routing for real scenario - Technikum

This scenario was considered too big and the XML syntax generated from OSM conversion was too complex to manually generate the flows of vehicles. For this reason, they were created using one of the tools included in the SUMO package, randomTrips.py.

This python script generates a series of trips for an input road network .net.xml file. The –r option was activated, allowing the script to automatically call DUAROUTER, obtaining directly a .rou.xml file with all the computed routes. If not activated, a .trip.xml file would be obtained and DUAROUTER should have been called manually.

A time interval of 4500 seconds during which this trips are generated was specified with the options –b (begin) 0, and –e (end) 4500. Also, a minimum length for the trips of 500 metres was specified with the option –min-distance. After finishing the process, the file Technikum.rou.xml was obtained.

As mentioned in the previous subsection, another .rou.alt.xml file was generated by the process and was also discarded since it was not considered useful.

3.1.3 Configuration and execution of simulation in SUMO

Execution of SUMO is available from the command line, calling sumo or sumo-gui in case of wanting to visualize a graphic representation of the simulation. Different options can be added

to the command in order to configure the simulation. In order to avoid very long option lists, configuration files allow to set up different parameters and input files.

This configuration files can have the extension sumo.cfg or .sumocfg, and are loaded in sumo from the command line with the option “–c”.

An example definition of a configuration file is shown in Figure 24. The children of the tag input are the different files that are used in the simulation. In the tag time, the simulation interval is specified. Another options can be configured, either in the config file or in the command line, such as a random simulation seed (--seed <int>) or the configuration of the simulation step length, which by default is 1s but can be made even smaller (--step-length 0.1 for example, would configure the step length of 100 ms).

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
noNamespaceSchemaLocation="http://sumo.sf.net/xsd/sumoConfiguration.xsd">

    <input>
        <net-file value="Test2.net.xml"/>
        <route-files value="Test2.rou.xml"/>
        <gui-settings-file value="Tes2.settings.xml"/>
        <additional-files value="Test2.poly.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="1000"/>
    </time>

</configuration>
```

Figure 24: Example SUMO config file

Once the simulation has been launched, in case of sumo.exe, execution information is shown in the command console, while in the case of sumo-gui.exe, the graphical interface is opened with a 2d visualization of the network and the vehicles performing their trips.

Figure 25 shows the example of sumo console when being run with and without the GUI, while Figure 26 presents the graphic interface of the simulator. Current simulation step can be observed in the display placed in the top. Simulation can be stopped and resumed with the buttons next to it. The delay parameter allows to slow down the simulation (increasing the time between simulation steps).
In this section, the TraCI Protocol is presented. Some explanations are given in order to justify the utility of TraaS, the library used for communicating with SUMO from Unity3D, which is also presented at the end of the section and which is one of the main blocks of the Driver Simulator.

```
C:\Users\Biurrun\Tesis\Examples\Test2>sumo -c Test2.sumo.cfg
Loading configuration... done.
Step #143.00 (1ms ~= 1000.00*RT, ~45000.00UPS, vehicles TOT 55 ACT 45)
```

```
C:\Users\Biurrun\Tesis\Examples\Test2>sumo-gui -c Test2.sumo.cfg
Loading configuration... done.
```

Figure 25: Command Line of SUMO with and without GUI

In Table 2, the input files and other parameter configuration made for both scenarios is shown.

	GRID SCENARIO	TECHNIKUM SCENARIO
Network file	Grid.net.xml	Technikum.net.xml
Route file	Grid.rou.xml	Technikum.rou.xml
Additional files	-	Technikum.poly.xml
Begin time	0	0
End time	4500	4500

Table 2: Configuration file definitions for each scenario

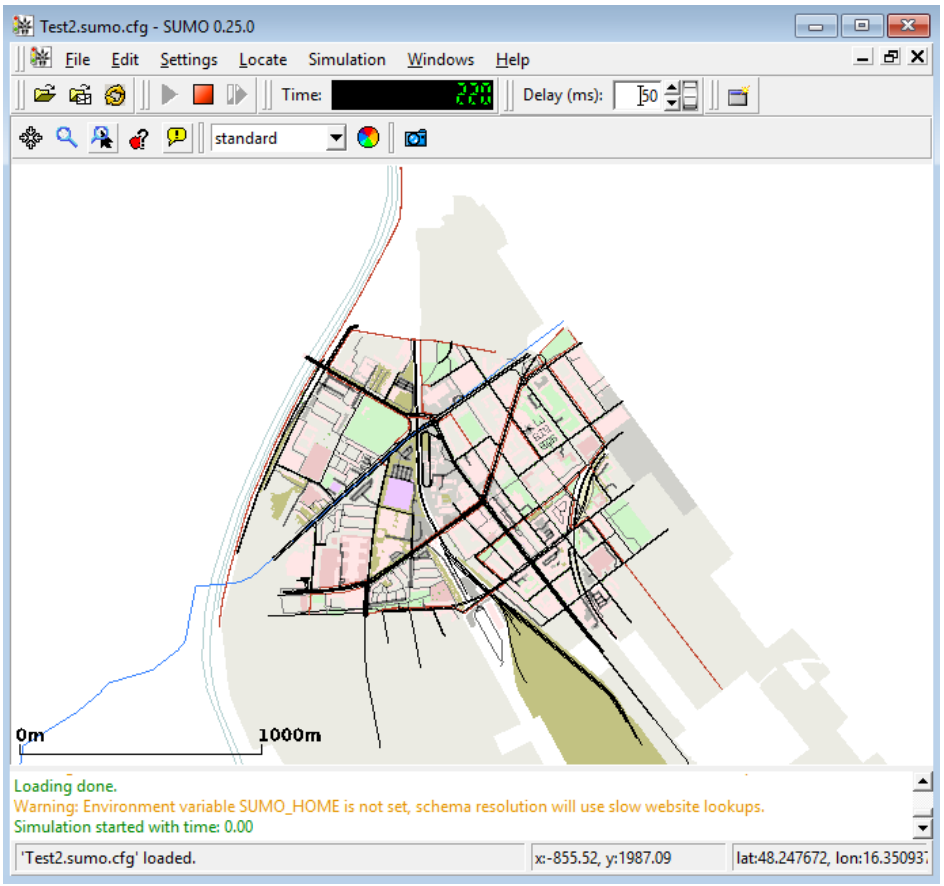

Figure 26: Execution example of SUMO with GUI

3.2 Communication between Unity3D and SUMO - TraCI Protocol and TraaS library

TraCI – short for **Tra**ffic **C**ontrol **I**nterface – is the communication protocol that allows to control and modify a simulation process in SUMO while this is being performed.

It also allows to retrieve that from any of the elements of the network that is being used in the simulation, such as vehicles, edges, junctions, or traffic lights. Each of this objects have a series of variables which can be retrieved or modify online.

This protocol **is what many of the works** where SUMO was combined with other software such as network or driving simulators, presented in section 2, **were based on**.

3.2.1 Analysis of the protocol

In Figure 27, a diagram of how the communication between SUMO and a TraCI client takes place is shown. First, SUMO must be run with the option --remote-port <PortNumber>. Activating this option, SUMO will be run and will wait until one client connects to its listening port, before performing the simulation.

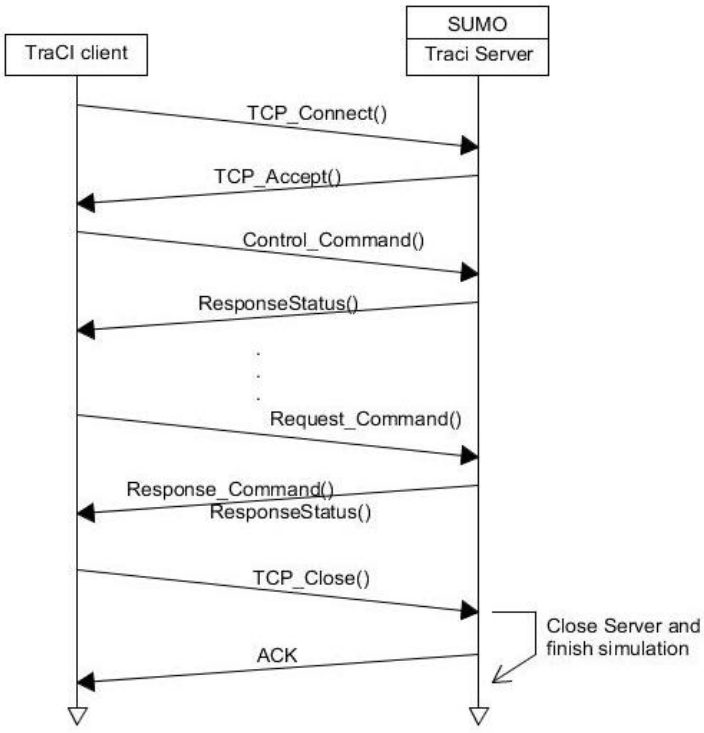

Figure 27: Implemented communication between SUMO and TraCI client, where control and request commands are sent to the TraCI server.

Once the client requests the connection, SUMO accepts it and the simulation **control duties are transferred to the client**. From this moment, the client can determine when to perform the simulation steps, and whether to apply changes of vehicle variables, changes of traffic signal logics, reassign the vehicle routing, or retrieving data regarding the elements of the network and the measures made by different induction loops and detectors.

Besides, the **TraCI commands have a specific common structure**, which is shown in the Figure 28. A 32 bit-length field indicates the total number of bytes that are sent, including itself (therefore, the number in this field will be at least 4). Then, all different commands follows. Each command has their own Identifier, which also allows to recognise the response to that command. The length field also includes itself in the counting of bytes. The command length is dependent on each command. Also in Figure 28, on the right side, a special structure for commands which are larger than 255 bytes (maximum number represented by a length field of 8 bits) is presented.

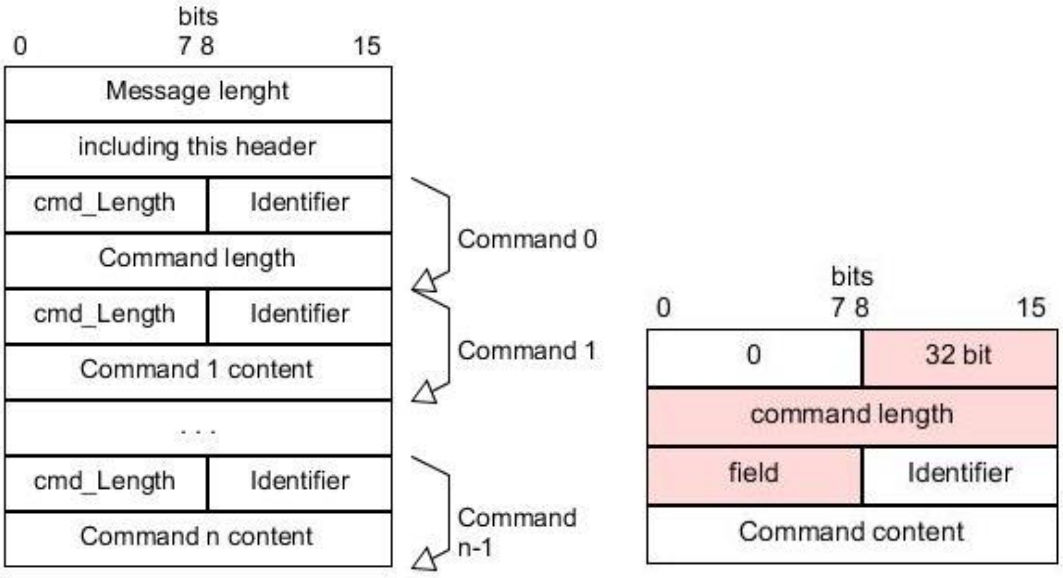

Figure 28: (Left) Structure of a TraCI Message. (Right) Extended structure for command length

A more detailed explanation regarding the different types of commands that can be used in the TraCI protocol, as well as their identifier and contents, is provided in **Appendix A**.

3.2.2 Implementing the TraCI protocol

In order to communicate with SUMO through the TraCI protocol, the need of some library which could be implemented in Unity3D was established as the main need. For this purpose, either developing a library from scratch or using one already existing stood as the only possibilities.

At first instance, efforts were made in developing a new C# library from scratch, since no existing C# library was found. For implementing the protocol, some existing libraries, referenced in the SUMO webpage were taken as reference. However, during the development of this library, a very useful tool was found which allowed to skip the development of this library.

3.2.2.1 IKVM.NET

IKVM.NET [52],[53] is an implementation of Java for the Microsoft .NET Framework. Among other uses, this tool is able to generate .NET libraries (.dll files) from Java bytecode (.class or .jar files) by using the tool ikvmc, which is a Java to .NET translator. This meant that one of the already existing libraries that implements the TraCI protocol could be used without spending extra time in the development of a whole C# library.

3.2.2.2 TraaS – TraCI as a Service

The chosen library is called **TraaS** [54], which stands for “TraCI as a Service”. This is an open source Web Service which allows to perform a remote control of SUMO from any programming language. A diagram representing its implementation is shown in Figure 29.

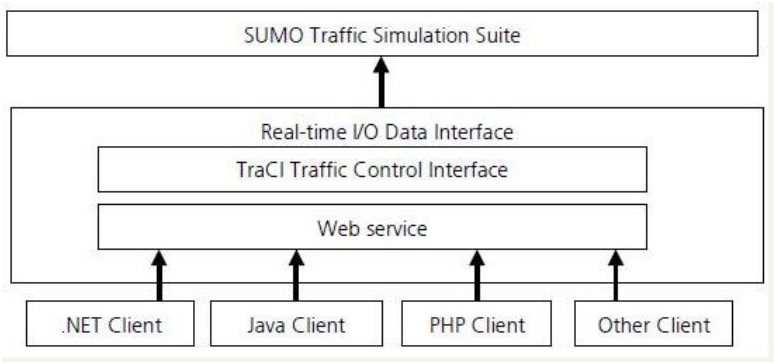

Figure 29: TraaS Web Service implementation [54]

However, this feature was not the interesting point about it. Making use of a webserver would introduce a considerable amount of delay in the proposed Driver-Centric Simulator, and some synchronization problems might occur. Therefore, the implementation of this library in the current project was something more alike to the diagram shown in Figure 30, were the .NET Client, Unity3D, implements the TraCI communication.

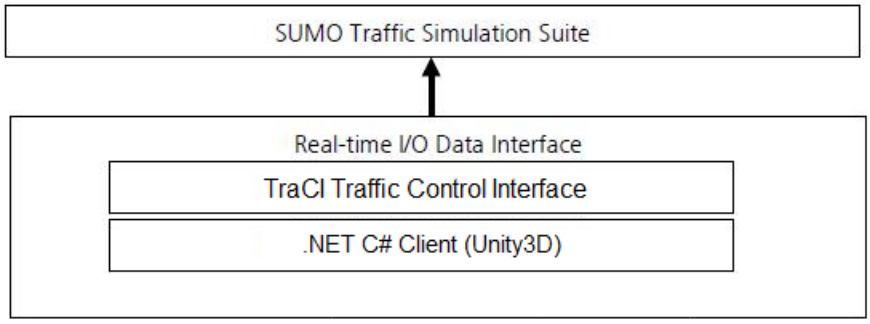

Figure 30: Schema of the TraaS library implemented in Unity3D

What was considered interesting about this library, was first, that it is a library written in Java code. Secondly, that it is a complete library which includes an enormous variety of commands which can be sent to SUMO in order to retrieve data regarding any of the objects mentioned in Appendix A, as well as changing their state.

In order to adapt the TraaS java library to Unity 3D, the following steps have been made:

1. Download of the TraaS library .jar from its webpage, mentioned above.

2. It was observed that the version of Java with which the .jar file was not suitable for the IKVM. Therefore, the source files were needed to be extracted and rebuilt.

3. Given that the source files were available, the deletion of the classes corresponding to the Web Service implementation was also carried out. The following classes were considered to be not useful and were deleted in order to make the library lighter. All of them were contained in the de/tudresden/ws package.

    - Service.class
    - Service_Impl.class
    - ShutdownHook.class
    - Traci.class
    - WebService.class: This class also included a main method, which was the cause of several errors when trying to convert the .jar file into a .dll. If a main method is included, the IKVMC tool generates an executable file .exe, instead of a .jar. Once it was determined that this method was not useful, this class and the above mentioned were also deleted.

    Although some other classes were not useful as well, they were not deleted because some classes constructors or methods (which have not been used in the current project) needed them. In order not to edit each of the classes individually – because it

was considered the effort would be similar compared to developing a library from scratch – these classes were left in the package. Future modifications are considered to be required, in order to optimise it, but the available time did not make this feasible.

4. Rebuild of the library in a .jar file called “TraaS_Optimized”, using Java SE 1.7, in order to present a suitable version for making the conversion with the IKVM tool.

5. Conversion of the “TraaS_Optimized.jar” file into “TraaS_Optimized.dll” file with IKVMC, included in the IKVM tool.

6. Importing the generated .dll file together with a series of other .dll files provided by IKVM – and which importation was determined to be essential in order to make the transformed java code work – into Unity3D.

3.3 Implementation of the 3D scenario – Unity3D

3.3.1 Setting up the scenario

The Unity project consists on a series of Assets: prefabs – also called models – for the vehicles and the roads and railways, scripts, which are explained late in this section, materials for giving more realistic aspect to the roads and vehicles and the scenes. It also includes the converted library for implementing the TraCI Protocol.

There are 2 scenes on which the simulator is divided: Start and Simulation.

Scene 0 - Start

This scene acts as the presentation scene, in which some vehicles and buildings have been placed. The main object of this scene is the Canvas placed in front of the Main Camera. This Canvas includes a text field where the path of the SUMO configuration file must be entered, and a button. When pressing the button, the script onPress creates a new object of type persistant_data, containing the mentioned path, which is kept and sent to the next scene.

A screenshot from the Start scene is observable in Figure 31. The buildings included in this scene are part of the free asset package European Town Corner Building by FlamingSands available in the Asset Store of Unity.

Scene 1 - Simulation

This scene hosts the Driver-Centric simulation itself. Ruled by the script SumoBehaviour, the connection with SUMO is performed and the generation of the 3D scenario is carried out. This script takes the path of the SUMO configuration file from the persistant_data object stored in the first scene. The location of the sumo executable file is also provided and must be configured (choosing either the GUI-version, sumo-gui.exe or the console version, sumo.exe).

Once the scenario has been initialized, two main task are performed by the Unity Engine. First, updating the position and the number of vehicles in the road following the algorithm presented in next section **3.3.2. Algorithm**. Secondly, allowing the driver's vehicle movement, performed by the script drive, which is explained in section **3.3.3. Driver-Centric perspective**.

A screenshot from the Simulation scene is provided in Figure 32. The Driver-Centric perspective can be observed in the middle of the simulation where 3 cars are arriving to an intersection.

Figure 31: Scene 0 - Start as viewed in Unity3D Game mode

No buildings were included in the scenario, since no dynamically building generation process was found. Nevertheless, CityEngine [55] was found to be a suitable tool for generating building models which could be inserted into the simulator.

Figure 32: Scene 1 - Simulation. The scenario has been generated and the vehicles are updating their positions. Driver-Centric perspective can be observed.

3.3.2 Algorithm

The algorithm presented in this section is the corresponding algorithm implemented in the class **SumoBehaviour**, which controls the data retrieving, the generation of the 3D scenario, and the updating of vehicles' positions.

Two different parts of the algorithm can be distinguished: the initialization of the scenario and the update of the vehicle objects.

3.3.2.1 Initialization

The initialization is based in an object of type **SumoTraciConnection**, which handles the execution of SUMO listening as a server in a free TCP port, creating a runtime process, connects to this server as a TCP client and sends the different commands for retrieving the desired data. The first two tasks are handled by the method runServer() of the **SumoTraciConnection** object, while each command for have their own specific method.

SUMO has been launched and Unity is connected to it, a first retrieval of data must be carried out. The data retrieval uses the method **do_job_get()** from the SumoTraciConnection object. In this method, a parameter of type **SumoCommand** is received and processed in order to send the proper request message to the Traci Server.

First, the road network must be created in the 3D scenario. This is achieved through methods **readLanes()** and **printLanes()**, implemented in the class SumoBehaviour. A schematic representation of the code in this first stage of initialization is shown in Table 3.

SumoBehaviour.initialize()

```
conn = new SumoTraciConnection(sumo_dir,config_file)
conn.runServer()
       launchSumoInRemotePort()
       tcp_connectToSUMO()
conn.do_timestep()
readLanes()
       printLanes()
```

Table 3: Initialize algorithm for setting up the scenario in Unity3D

First, a reference to every lane existing in the SUMO network should be obtained. This is achieved with the **readLanes()** method. A schematic representation of this method is shown in Table 4. For this purpose. a SumoCommand Lane.getIDList() is set as the parameter requested by do_job_get(). A list with the IDs of every lane object existing in the SUMO network is obtained.

This list is iterated, requesting to the TraCI Server different variables of the specified lanes: the vertices that conform its shape, the lane width, and the allowed vehicles that can use this lane, creating one object of type Link – a class created for storing a reference to each variable of each lane objects retrieved from SUMO – object which is also added to a list of links.

readLanes()

```
conn.do_job_get(LaneIDList)
linkList = new List<Link>()
foreach(id in IDlist)
      vertices = conn.do_job_get(laneID.Vertices)
      width = conn.do_job_get(laneID.Width)
      allowed = conn.do_job_get(laneID.AllowedVehicles)
            type = switch(allowed)
      link = new Link(vertices,width,id,type)
      linkList.add(link)
printLanes()
```

Table 4: readLanes() method algorithm for retrieving all the lanes existing in the SUMO network

Finally, the **printLanes()** method is called in order to generate the corresponding 3D models of the different links, as well as the **car controlled by the user**. In this method, the list of links created in the readLanes() method iterated. A schematic representation of this method is shown in Table 5.

Each link/lane has a series of (X,Y) points called vertices which define the shape of the lane. One segment can be defined every 2 X-Y tuples, with its length and angle respect to the coordinated system Thus, **one GameObject is created per each segment** of the lane.

The first vertex is extracted in order to create the first segment and then create the rest in an iterative way. Applying basic geometry theory – trigonometry identities for the angle and Pythagoras' Theorem for the length – the angle of the segment respect to the coordinate system and its length are calculated. Then, depending on the **type** of the lane, stored in the Link object, a railway segment or a road segment is instantiated. The second point of the tuple is then taken as the new reference and the rest of the segments are created iteratively.

Before iterating to the next lane, the ID of the current is checked. If its ID corresponds to **"start_sim"**, then this ID is the selected start point for the driving simulation. The vehicle which can be controlled by the user (therefore, the most important object of the scenario), is instantiated in the middle point of the last segment of the lane. It was observed that the vehicle was instantiated on the right side of the road, almost out of it, reason why an **offset** was included.

printLanes()

```
foreach(link in linkList)
       x0 = link.vertices[0].x
       y0 = link.vertices[0].y
       for(j=1…lane.vertices.length-1)
            x1 = link.vertices[j].x
            y1 = link.vertices[j].y
            angle = arctg((y1-y0)//x1-x0))
            length = sqrt((y1-y0)^2 + (x1-x0)^2)
            center.x = (x1+x0)/ 2
            center.y = (y1+y0) / 2
            if(link.type == tram)
                  Instantiate new Railway (center, width, length)
            else
                  Instantiate new Road (center, width, length)
            x0 = x1;
            y0 = y1;
      If(link.ID == start_sim)
           Instantiate new DriverVehicle(segment.center + offset,
           segment.angle)
```

Table 5: printLanes() method algorithm for instantiating the road and driving vehicle GameObjects

Once the road network has been initialized, as well as the driving vehicle, the existence of vehicles in the road must be checked. This is the purpose of the method **readCars().** A schematic representation of this method is shown in Table 6.

First, a SumoCommand Vehicle.getIDList() request is sent to the TraCI Server. A List with all the vehicle IDs existing in the network at the first simulation step is provided. This list might be empty or not, but this is not a problem. A list of Auto – which, similarly to Lane, stores the variables regarding one vehicle extracted from SUMO – is initialized. This list is updated (adding and removing elements) during the update stage.

In case some vehicle already exists in the first simulation step, its position, angle respect to the coordinate system and speed are retrieved. A new Auto object is created and added to the mentioned vehicle list.

readCars()

```
conn.do_job_get(VehicleIDlist)
vehicleList = new List<Auto>();
foreach(id in vehicleIDlist)
      position = conn.do_job_get(vehicleID.Position)
      angle = conn.do_job_get(vehicleID.Angle)
      speed = conn.do_job_get(vehicleID.Speed)
      vehicle = new Auto(position,id,speed,angle)
      vehicleList.add(vehicle)
printCars()
```

Table 6: readCars() method for initializing the vehicleList and retrieving the first vehicles

Finally, the **printCars()** method is called. A schematic representation of this method is presented in Table 7. This method iterates the vehicle list and generates a new Car object using the variables regarding position and angle. Depending on the configured departing times in the .rou.xml file, this method might not create any vehicle.

printCars()

```
foreach(vehicle in vehicleList)
      Instantiate new Car (position,angle)
```

Table 7: printCars() method for instantiating the first vehicles

3.3.2.2 Update

Once the scenario has been initialized, the simulation must be run. During this execution, synchronization stands as a critical feature which must be satisfied.

Unity3D has a specific updating system, which is the method **Update()**, inherited from the class **MonoBehaviour**. This method is called in every frame of the application. However, the number of FPS (Frames Per Second) might not the same during the execution of the application. For this reason, the method **FixedUpdate()** was implemented. This method ensures the number of FPS is constant. Therefore, allows that the time gap between the SimulationStep requests from the TraCI Client is also constant.

It was observed that performing a request of the vehicles in the network on every frame was quite computationally demanding, which also resulted in a very unrealistic movement. Since the simulation step in sumo corresponds to 100 milliseconds, the vehicles seem to teleport from one location to another on every frame.

Since the usual number of FPS is 50, the time between frames is 20 milliseconds. Attending to this value, it was proposed to perform a simulation step every 5 frames. However, the result was even worse than the previous one. The teleporting effect was increased by the fact that the position update was perform within a higher time gap.

For this reason, due to the availability of each vehicle's speed, the **move()** method was developed. This method calculates the distance that each car would advance on each frame from its speed and the time between frames, **Time.deltaTime**. The **rotation** of the vehicle is not modified, but it is used in order to calculate the corresponding components of the speed vector for the coordinate axis X and Z. It is presented in a schematic way in Table 8.

Every 2 frames
move()
foreach(vehicle in vehicleList) vehicle.getSpeed() vehicle.getAngle() vehicle.x = vehicle.x + speed*sin(angle)*deltaTime vehicle.z = vehicle.z + speed*cos(angle)*deltaTime

Table 8: The move() method for inter-step movement of vehicles

Therefore, between simulation steps, the vehicle is able to advance a distance equivalent to the distance that it would perform in continuous time.

SumoBehaviour.FixedUpdate

```
int count++
if (count == 10)
      readCars()
            printCars()
            do_timestep()
else if (count % 2 == 0)
      move()
```

Table 9: Algorithm for updating vehicles' position and number

This approach was tested, performing a new Simulation Step and retrieving data about new vehicles, as well as vehicles which finished their trips, every 5 frames and performing the artificial movement in the frames in the middle.

Every 10 frames

readCars()

```
added = conn.do_job_get(
Simulation.getAddedVehicleIDList)
vehicleIDList.Add(added)
removed = conn.do_job_get(
Simulation.getRemovedVehicleIdList)
vehicleIDList.Remove(removed)
 foreach(id in vehicleIDlist)
       position = conn.do_job_get(vehicleID.Position)
       angle = conn.do_job_get(vehicleID.Angle)
       speed = conn.do_job_get(vehicleID.Speed)
       vehicle = new Auto(position,id,speed,angle)
       vehicleList.add(vehicle)
 printCars()
```

printCars()

```
foreach(vehicle in vehicleList)
      vehicle =  = GameObject.find(vehicleID)
      if (vehicle.exists)
            vehicle.UpdatePosition
      else
            Instantiate new Car(position,angle)
conn.do_timestep()
```

Table 10: vehicleList updating, instantiating of new vehicles and position updating of existing ones

However, it was observed that, although the movement of the vehicles was effectively smoother and more realistic, it was too fast. For this reason, the interval was increased as twice as proposed, **performing a simulation step every 10 frames and an artificial movement every 2 frames**. The visual perception after implementing this approach was slightly improved, reason why it was finally implemented. Some schematic representation of the code of this method is presented in Table 10.

The readCars() method updates the vehicleIDList with the added and deleted vehicles on each simulation step. Once this list is updated with the new vehicles IDs, the same iteration process as the one detailed in Table 6 is carried out, retrieving the position, speed and angle of each vehicle of the network. This was represented in Table 9.

Finally, the printCars() method is called. In this stage, it iterates the vehicleList and looks for an already instantiated vehicle. If this vehicle is found, its position is updated. If not, a new GameObject is instantiated with the corresponding position and angle.

3.3.3 Driver-Centric perspective

The Driver-Centric perspective bases its performance in an object called Player_Vehicle. This vehicle is very similar to the NPC_Vehicle, but incorporates the Main Camera of the scene, which is located inside the vehicle in a way that the perspective from the driver's eyes is achieved. Figure 32, in section 3.3.1., presents a screenshot from the driver's perspective during simulation in which the vehicle's speed is 10 km/h (because it was approaching to an intersection).

The movement of the vehicle is performed with the script drive. This script checks on every frame the inputs (throttle/brake, left/right, either implemented with the keyboard arrows or with a gaming driving set) and computes the equivalent acceleration/braking power in order to update the speed. It also calculates the turning power, changing the rotation of the vehicle, as well as the rotation of the steering wheel, which provides a more realistic driving sense.

3.3.4 Setting-up the simulation room

A room has been prepared to host the simulation platform. A commercial gaming driving set consisting of a seat, pedals and steering wheel has been connected to the computer in order to be used for providing a more realistic driving experience. In the following figures, the appearance of the set-up is presented.

Figure 33: Driver-Centric perspective in the simulation room.

Figure 34: Driving set placed on a platform connected to the computer

Figure 35: Simulator set-up general perspective

4 Evaluation

In this section, some comments regarding the simulator performance are provided. These comments concern the **three main parts** of the simulation: generation of the roads, the generation of the vehicles and the driving performance.

Generation of the roads

In terms of “lag” or delay, it has been observed that the generation of the road was considerably fast for both scenarios, being slightly slower for the case of the OSM scenario, due to its higher number of elements.

It has been observed that the implemented algorithm for creating the road network works considerably well. In the case of the Grid scenario, the simplest one, the conversion is performed without any mistake. In Figure 36, and also in Figure 37 with a higher zoom, a bird perspective of the mentioned scenario is provided.

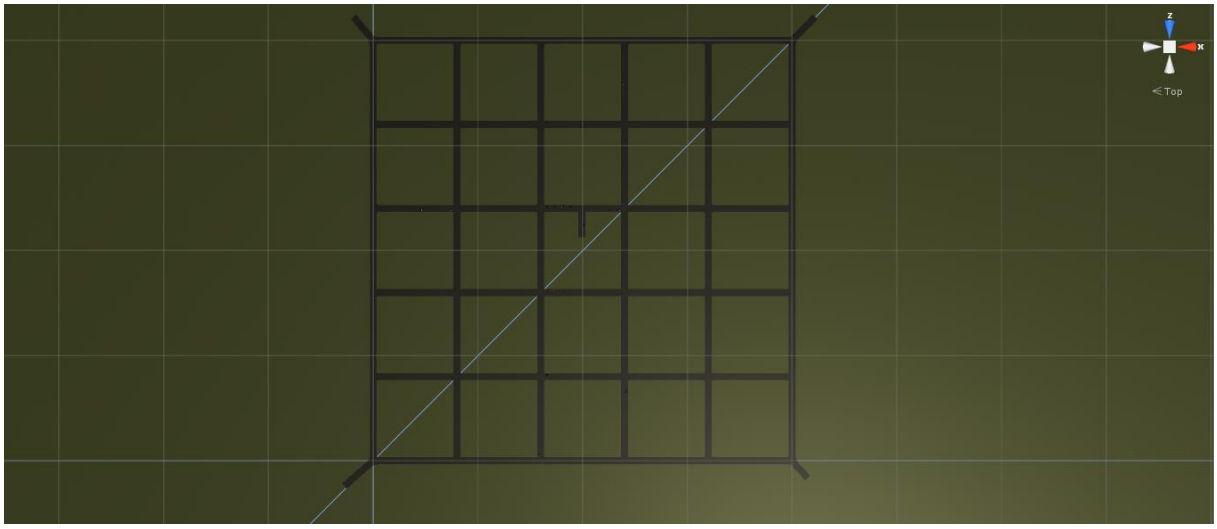

Figure 36: Bird perspective from the Grid scenario. Screenshot from Unity 3D Project.

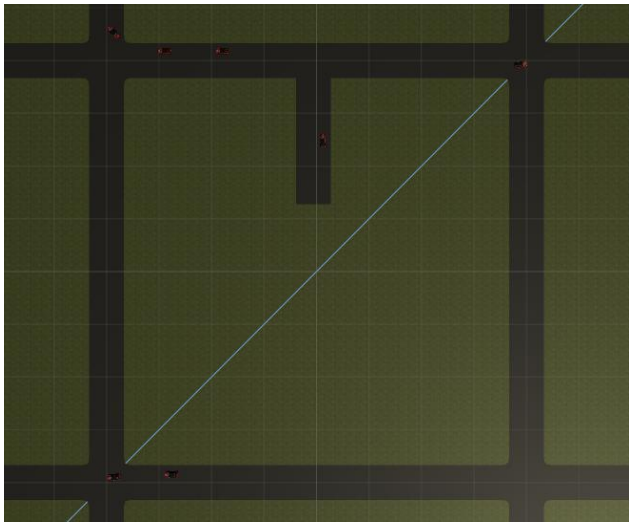

Figure 37: Zoom into the center of the bird perspective of the Grid scenario. Different cars performing their movements are shown, as well as the driver-controlled vehicle, which stands in the lane in the middle. Screenshot from Unity 3D Project.

In the case of the OSM generated scenario, a first impression provides the same feelings as the Grid scenario. A bird perspective from the generated scenario is provided in Figure 38.

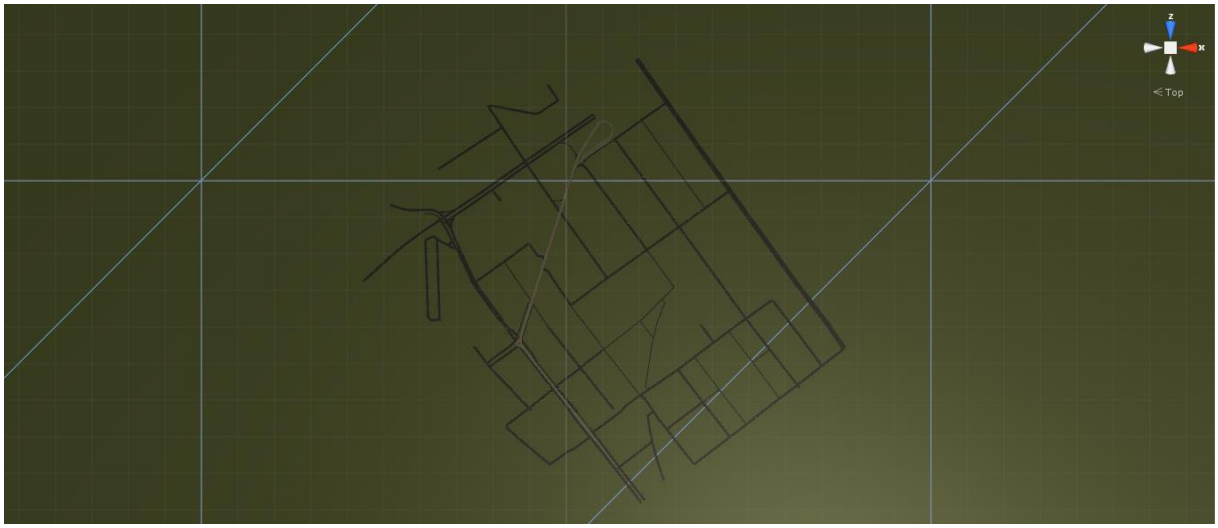

Figure 38: Bird perspective from the Technikum scenario. Screenshot from Unity 3D Project.

However, when deeper examination was performed, some minor mistakes were denoted. An example of these situations is provided in Figure 39. Here, it can be seen that the tram railways, which are overlapped with the roads in SUMO, completely cover them in the Unity scenario. It is also observed than some lanes present irregular boundaries.

After some investigation, it was determined that this was due to the fact that in the original network, an intersection with a bicycle lane was there. Although the bicycle lane was erased, the intersection was not. Therefore, an imperfect network editing process might be the cause for this irregular boundaries.

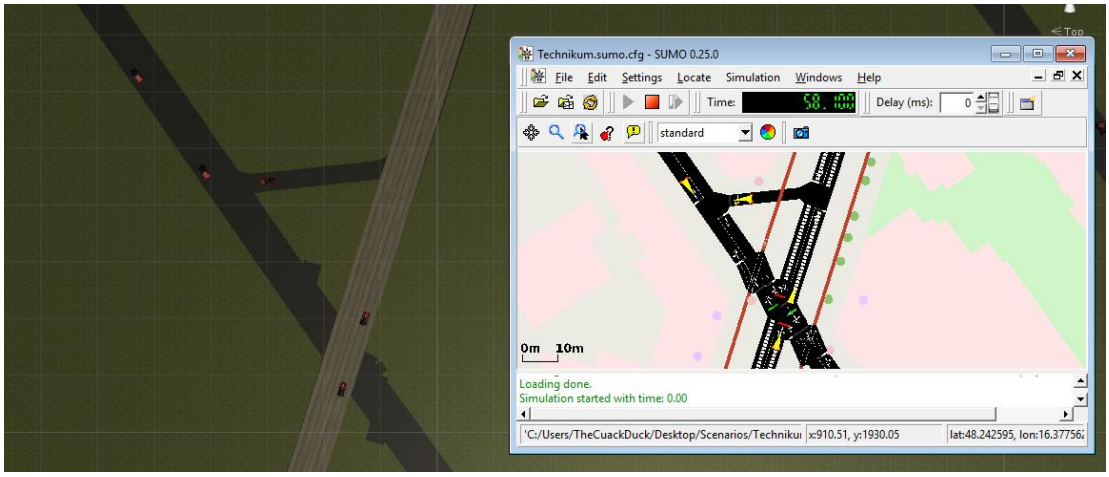

Figure 39: Example of an error in generating the road for OSM imported scenario. Unity3D representation is provided on the left side, while the SUMO-GUI representation is seen on the right side.

Generation of the vehicles

Since the vehicles are generated on each simulation step (if new vehicles' information has been retrieved via TraCI), the generation process runs smoothly. It has been observed that in case of generating tens of vehicles at once, the simulation suffers a considerable delay due to all the TCP messages sent, received and processed, and the driving task becomes impossible to perform. However, any empirical measure of this delay have been performed.

It is proposed for further improvements, although not yet implemented, the instantiation of only the vehicles located within a radius from the driver's vehicle. This measure is thought to be able to decrease the delay.

Driving performance

The driving task in terms on delay did not show any remarkable problem in the Grid scenario, since the number of vehicles contained in the network was not big enough to imply a demanding computational charge. For the Technikum scenario, however, it was observed that after approximately 10 minutes of simulation, the driving performance was diminishing due to the increasing number of vehicles contained in the network. The delay also reduced the quality of the performance in general, with some of the NPC vehicles making paused movements.

It is also denoted that the sensitivity of the steering wheel was too high, and the turning movements were performed too fast to be considered realistic.

5 Conclusions and future work

In this bachelor thesis, **the development of a Driver-Centric Simulator** whose NPC traffic is based on **microscopic simulation** has been developed. This simulator is based on the connection of the microscopic simulator SUMO and the graphic 3D engine Unity3D, through the TraCI protocol, which was implemented by using an existing Java library, TraaS, available online, and translating it into .NET C# in order to make it available to use in Unity3D.

Previous work on this area achieved the development of Driver-Centric simulators using Unity 3D and a microscopic traffic simulator, but none of them used the microscopic simulator SUMO to generate the vehicle movements online – although offline generation of a 3D scenario with Unity3D using SUMO traces has been already achieved. Therefore, the current work stands as the first approach to connect Unity3D and SUMO.

Some basic concepts in order to understand the structure of SUMO and its networks were provided. Besides, the TraCI protocol, key interface for allowing the data exchange between SUMO and Unity3D, was also presented.

Two different scenarios were developed in order to test the simulator. One of them constitutes a grid of perpendicular streets, while the other one stands as a real urban scenario generated from OpenStreeMap Data.

During the **evaluation process**, it was determined that the errors in representation observed in the 3D scenario where due to an imperfect editing process of the sumo network file, instead of a problem from the implemented algorithm. For this reason, in further developments, more attention should be paid during the network edition process.
At the same time it was also observed that vehicle-crowded networks introduce a considerable delay in the simulation. A modification in the algorithm which consists in only instantiating the vehicles located within a radius from the driver's vehicle is also proposed for further developments.
Finally, it was denoted the need of reducing the sensitivity of the steering wheel, since it was producing very unrealistic turning movements.

As **future lines of work**, more features could be included in the simulator. The implemented library allows the retrieval of data regarding many different object types from the SUMO network, being able in theory to retrieve information regarding the traffic lights phases, for example (and therefore being able to implement traffic lights in the simulator).

It also allows to modify the state of the simulation being performed in SUMO, being able to add new vehicles and modifying routes. These commands result interesting from the point of view of **including the driver's vehicle in the SUMO simulation**, and updating its position on every step, in order to allow, bidirectional influence between the simulators. NPC vehicles, for example, would consider the existence of the controlled vehicle, and therefore they should be able to maintain a security gap with it, and brake in case of collision risk.

Also, the fact of using a **third-party open-source library** made it possible to implement the TraCI protocol in a fast way, which also allows a fast updating in case a new version of the library is created (IKVM tool would be used to generate the new .dll file and this would be imported to the Unity project). On the other hand, it also implied the following **disadvantages**:

- **Code efficiency might not be achieved**. Since it was developed by other person/people, trust was deposited on them when using their library. At least, it was proven that the communication was taking place successfully when using this library.

- **JavaDoc information is lost** in the conversion process. Due to the conversion from java to .NET, the different classes and methods can be used, but the original documentation is needed to be checked, since the .NET editor does not include this information.

In spite of these disadvantages, **the adopted solution stands as a balanced** one, in which a great amount of time was saved in order to implement the TraCI protocol, and therefore, the first trials with the graphic engine, Unity3D, could be done in an early stage.

For this reason, as a possible line of future work, the development of a C# library for implementing the TraCI protocol could be considered. Not only would it be an advantage to use an own-made library, but also a huge contribution to the SUMO community.

Finally, when fixed the mentioned performance problems of the graphic scenario, creating a testing scenario for VANETs or different ADAS and/or IVIS, as carried out in many of the works presented in the related work section, would **leverage the potentials of this simulation tool.** For these applications, the sumo network and traffic demand should be defined consciously, in order to generate the proper boundary conditions for the study.

Bibliography

[1] J. S. V. . Gonçalves, R. J. F. . Rossetti, J. . B. Jacob, J. . Gonçalves, C. . Olaverri-Monreal, A. . B. Coelho, and R. . B. Rodrigues, “Testing Advanced Driver Assistance Systems with a serious-game-based human factors analysis suite,” *IEEE Intell. Veh. Symp. Proc.*, no. Iv, pp. 13–18, 2014.

[2] J. Gonçalves, R. J. F. Rossetti, and C. Olaverri-Monreal, “IC-DEEP: A serious games based application to assess the ergonomics of in-vehicle information systems,” *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, pp. 1809–1814, 2012.

[3] D. Krajzewicz and G. Hertkorn, “SUMO (Simulation of Urban MObility) An open-source traffic simulation,” *… Symp. Simul. …*, pp. 63–68, 2002.

[4] E. B. Lieberman and A. K. Rathi, “Traffic Simulation,” 1999.

[5] A. Gibb, M. St-Jacques, G. Nourry, and M. Johnson, “A Comparison of Deterministic vs Stochastic Simulation Models for Assessing Adaptive Information Management Techniques over Disadvantaged Tactical Communication Networks,” *7th ICCRTS*, no. April 2002, pp. 1–16, 2002.

[6] C. Harrell, “Simulation Basics,” *Simul. Using ProModel*, pp. 57–85, 2011.

[7] M. G. McNally, “The Four Step Model.,” *UC Irvine Cent. Act. Syst. Anal.*, 2008.

[8] “Sumo at a Glance,” 2016. [Online]. Available: http://sumo.dlr.de/wiki/Sumo_at_a_Glance. [Accessed: 04-Jun-2016].

[9] S. Miller, “Traffic Modeling,” pp. 1–15, 2011.

[10] W. Burghout, “Mesoscopic simulation models for short-term prediction,” *Cent. Traffic Res.*, pp. 10–20, 2005.

[11] M. J. Lighthill and G. B. Whitham, “On Kinematic Waves. II. A Theory of Traffic Flow on Long Crowded Roads,” *Proc. R. Soc. London A Math. Phys. Eng. Sci.*, vol. 229, no. 1178, pp. 317–345, May 1955.

[12] P. I. Richards, “Shock Waves on the Highway,” *Oper. Res.*, vol. 4, no. 1, pp. 42–51, Feb. 1956.

[13] C. F. Daganzo, “The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory,” *Transp. Res. Part B Methodol.*, vol. 28, no. 4, pp. 269–287, 1994.

[14] C. F. Daganzo, “The cell transmission model, part II: Network traffic,” *Transp. Res. Part B Methodol.*, vol. 29, no. 2, pp. 79–93, 1995.

[15] C. F. Daganzo, “The Lagged Cell-Transmission Model.” .

[16] “Transport strategy and transport modelling with PTV Visum,” *Vision-traffic.ptvgroup.com*, 2016. [Online]. Available: http://vision-traffic.ptvgroup.com/en-uk/products/ptv-visum. [Accessed: 31-May-2016].

[17] D. R. LEONARD, P. GOWER, and N. B. TAYLOR, “CONTRAM: structure of the model,” *Res. Rep. - Transp. Road Res. Lab.*, no. 178.

[18] L. SMITH, R. Beckman, and K. Baggerly, “TRANSIMS: TRANSPORTATION ANALYSIS AND SIMULATION.” 1995.

[19] R. Jayakrishnan, H. S. Mahmassani, and T.-Y. Hu, “An evaluation tool for advanced traffic information and management systems in urban networks,” *Transp. Res. Part C Emerg. Technol.*, vol. 2, no. 3, pp. 129–147, 1994.

[20] C. Gawron, R. Schrader, and E. Dahlhaus, “Simulation-Based Traffic Assignment,” *der Math. Fak.*, vol. zur Erlang, p. 93, 1998.

[21] P. G. Gipps, “A behavioural car following model for computer simulation,” *Transportation Research part B*, vol. 15. pp. 101–115, 1981.
[22] M. Treiber, A. Hennecke, and D. Helbing, “Congested Traffic States in Empirical Observations and Microscopic Simulations,” *Phys. Rev.*, vol. 62, no. 2, pp. 1805–1824, 2000.
[23] “NPTEL :: Civil Engineering - Traffic Engineering & Management,” *Nptel.ac.in*, 2016. [Online]. Available: http://nptel.ac.in/courses/105101008/15. [Accessed: 31-May-2016].
[24] P. G. Gipps, “A model for the structure of lane-changing decisions,” *Transp. Res. Part B Methodol.*, vol. 20, no. 5, pp. 403–414, 1986.
[25] “Top Features in Aimsun,” 2015. [Online]. Available: https://www.aimsun.com/aimsun/top-features/. [Accessed: 31-May-2016].
[26] J. Barceló and J. Casas, “Simulation Approaches in Transportation Analysis: Recent Advances and Challenges,” R. Kitamura and M. Kuwahara, Eds. Boston, MA: Springer US, 2005, pp. 57–98.
[27] J. Barcelo, M. Fellendorf, and P. Vortisch, *Fundamentals of Traffic Simulation*, vol. 145. 2010.
[28] “Quadstone Paramics | Traffic and Pedestrian Simulation, Analysis and Design Software,” 2016. [Online]. Available: http://www.paramics-online.com/. [Accessed: 30-Apr-2016].
[29] W. Schakel, B. van Arem, H. van Lint, and G. Tamminga, “A modular approach for exchangeable driving task models in a microscopic simulation framework,” in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, 2013, pp. 565–571.
[30] J. Erdmann, “Lane-changing model in SUMO,” 2014.
[31] S. Kraus, P. Wagner, and C. Gawron, “Metastable States in a Microscopic Model of Traffic Flow,” *Phys. Rev. E*, vol. 55, no. 304, pp. 55–97, 1997.
[32] S. Krau??, “Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics,” *D L R - Forschungsberichte*, no. 8, 1998.
[33] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux, “TraCI: An Interface for Coupling Road Traffic and Network Simulators AxelWegener1,” *Proc. 11th Commun. Netw. Simul. Symp. - CNS ’08*, pp. 155–163, 2008.
[34] E. Blana, “Driving Simulator Studies: A Literature Review,” 1996.
[35] D. Djaouti, J. Alvarez, and J.-P. Jessel, “Classifying serious games: The G/P/S model,” *Handb. Res. Improv. Learn. Motiv. through Educ. games Multidiscip. approaches*, no. 2005, pp. 118–136, 2011.
[36] A. E. Akinwuntan, W. De Weerdt, H. Feys, J. Pauwels, G. Baten, P. Arno, and C. Kiekens, “Effect of simulator training on driving after stroke: A randomized controlled trial,” *Neurology*, vol. 65, no. 6, pp. 843–850, Sep. 2005.
[37] D. L. Roenker, G. M. Cissell, K. K. Ball, V. G. Wadley, and J. D. Edwards, “Speed-of-Processing and Driving Simulator Training Result in Improved Driving Performance,” *Hum. Factors J. Hum. Factors Ergon. Soc.*, vol. 45, no. 2, pp. 218–233, 2003.
[38] H. RL, B. TL, G. Milavetz, A. Spurgin, P. RS, G. DA, G. Gaffney, and H. MA, “Cannabis effects on driving lateral control with and without alcohol.,” vol. 154. 2015.
[39] Y. C. Li, N. N. Sze, S. C. Wong, W. Yan, K. L. Tsui, and F. L. So, “A simulation

study of the effects of alcohol on driving performance in a Chinese population." 2016.
[40] C. Olaverri-Monreal, J. Gonçalves, and K. J. Bengler, "Studying the driving performance of drivers with children aboard by means of a framework for flexible experiment configuration," *IEEE Intell. Veh. Symp. Proc.*, no. Iv, pp. 7–12, 2014.
[41] M. Piórkowski, M. Raya, a. L. Lugo, P. Papadimitratos, M. Grossglauser, and J.-P. Hubaux, "TraNS: Realistic Joint Traffic and Network Simulator for VANETs," *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 12, no. 1, p. 31, 2008.
[42] J. Härri, F. Filali, C. Bonnet, and M. Fiore, "VanetMobiSim: generating realistic mobility patterns for VANETs," *Proc. 3rd Int. Work. Veh. ad hoc networks*, pp. 4–5, 2006.
[43] C. Sommer, S. Member, and R. German, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," vol. 10, no. 1, pp. 3–15, 2011.
[44] S. Guan, R. E. De Grande, and A. Boukerche, "Real-time 3D visualization for distributed simulations of vanets," *Proc. - IEEE Int. Symp. Distrib. Simul. Real-Time Appl. DS-RT*, pp. 138–146, 2014.
[45] J. Raghothama, M. Azhari, M. R. Carretero, and S. Meijer, "Architectures for distributed, interactive and integrated traffic simulations," *2015 Int. Conf. Model. Technol. Intell. Transp. Syst. MT-ITS 2015*, no. June, pp. 387–394, 2015.
[46] V. Charissis, S. Papanastasiou, W. Chan, and E. Peytchev, "Evolution of a full-windshield HUD designed for current VANET communication standards," *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, no. Itsc, pp. 1637–1643, 2013.
[47] P. Gomes, C. Olaverri-monreal, and M. Ferreira, "Driver-Centric VANET Simulation ," pp. 143–154, 2011.
[48] C. Olaverri-Monreal, P. Gomes, R. Fernandes, F. Vieira, and M. Ferreira, "The See-Through System: A VANET-enabled assistant for overtaking maneuvers," in *2010 IEEE Intelligent Vehicles Symposium*, 2010, pp. 123–128.
[49] Y. Hou, S. Member, Y. Zhao, A. Wagh, L. Zhang, C. Qiao, K. F. Hulme, C. Wu, A. W. Sadek, and X. Liu, "Simulation Based Testing and Evaluation Tools for Transportation Cyber - Physical Systems," *IEEE Trans. onVehicular Technol.*, vol. PP, no. 99, pp. 1098–1108, 2015.
[50] SUMO, "SourceForge - Simulation of Urban MObility." 2012.
[51] "OSM - Open Street Map," 2016. [Online]. Available: https://www.openstreetmap.org.
[52] J. Frijters, "IKVM.NET Home Page," 2014. [Online]. Available: https://www.ikvm.net/#Introduction.
[53] "IKVM.NET Sourceforge Wiki," 2016. [Online]. Available: https://sourceforge.net/p/ikvm/wiki/Main_Page/.
[54] "TraaS . TraCI as a Service." [Online]. Available: http://traas.sourceforge.net/cms/.
[55] E. CityEngine, "3D Modeling Software for Urban Environments.," 2013. .

List of Figures

List of Tables

List of Abbreviations

ADAS	Advanced Driving Assistance Systems
DLC	Discretionary Lane Change
FPS	Frames Per Second
GPL	General Public License
GUI	Graphic User Interface
HUD	Heads-Up Display
IVIS	In-Vehicle Information Systems
MLC	Mandatory Lane Change
MOE	Measure of Effectiveness
NPC	Non Player Controlled
OSM	OpenStreetMap
SUMO	Simulation of Urban MObility
TraaS	TraCI as a Service
TraCI	Traffic Control Interface
VANET	Vehicle Ad-hoc NETwork

Appendix A: TraCI Protocol: Commands, Variables and Identifiers

This purpose of this appendix is to illustrate the structure of the messages that are exchanged between the TraCI server and its client, on which the communication is based. This brief explanation was developed while trying to understand the TraaS library and the messages involved.

1. CONTROL COMMANDS

SimulationStep: When received, SUMO is forced to run the simulation until the desired timestep. If the timestep sent is 0, only one simulation step is performed. A diagram was created in order to illustrate the structure of this command. It is presented in Figure 40.

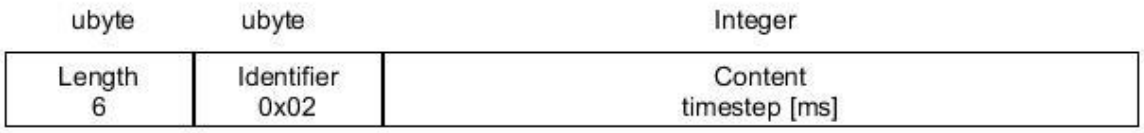

Figure 40: SimulationStep command structure

Close: Closes the connection. The identifier is 0x7F and no content is included.

2. DATA RETRIEVAL COMMANDS

There are 15 different object types that can be extracted. The request commands for each of the object types have a unique Identifier. These identifiers are presented in Table 11. Each object of each type has a unique ID that allows to identify it, and a series of variables can be retrieved from them. The general structure a Get Variable Command is shown in Figure 41.

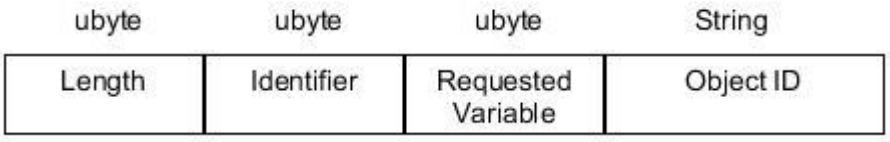

Figure 41: General structure of a Get Object Variable command

The response from SUMO to these commands follows the structure shown in Figure 35, where return type identifier indicates the type of the variable that has been requested.

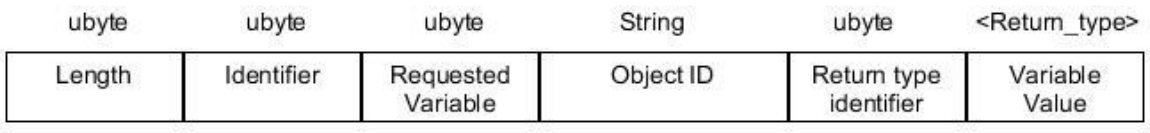

Figure 42: General structure of a Get Object Variable response from SUMO

OBJECT TYPE	COMMAND ID
Induction loop	0xA0
Multi-Entry/Multi-Exit Detector	0xA1
Traffic Lights	0xA2
Lane	0xA3
Vehicle	0xA4
Vehicle Type	0xA5
Route	0xA6
Point of interest	0xA7
Polygon	0xA8
Junction	0xA9
Edge	0xAA
Simulation	0xAB
GUI	0xAC
Lane Area Detector	0xAD
Person	0xAE

Table 11: Identifiers for Data Retrieving Commands

3. STATE CHANGE COMMANDS

These commands allow to modify the status of the simulation while this is being performed. Adding a new vehicle, modifying its route or changing a traffic light program are some examples of what is possible to do.

There are 11 different types of commands that allow to perform this modifications. Each type together with its identifier is shown in Table 12.

OBJECT TYPE	COMMAND ID
Traffic Lights	0xC2
Lane	0xC3
Vehicle	0xC4
Vehicle Type	0xC5
Route	0xC6
Point of interest	0xC7
Polygon	0xC8
Junction	0xC9
Edge	0xCA
Simulation	0xCB
GUI	0xCC

Table 12: Identifiers for State Change Commands

Similarly to the data retrieval commands, the state change commands, as well as their responses from SUMO, have also defined structures, which are presented in Figures 43 and 44, respectively.

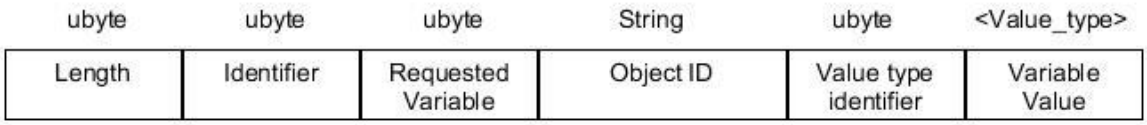

Figure 43: General structure of a State Change Command.

The ubyte “Value type identifier” indicates the TraCI Server the variable type of the value that is being tried to be set. These identifiers are the same as the identifiers seen in Figure 31 and are listed in Table 13.

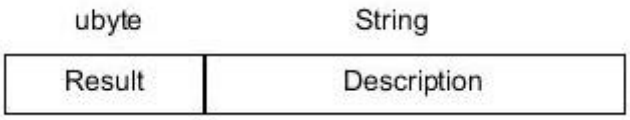

Figure 44: Structure of a State Change Response from SUMO.

DATA TYPE	IDENTIFIER	SIZE and description
Ubyte	0x07	8 bit
Byte	0x08	8 bit
Integer	0x09	32 bit
Float	0x0A	32 bit
Double	0x0B	64 bit
String	0x0C	32 bit length field + 8bit/char
stringList	0x0E	32 bit number of strings + n strings
2DPosition	0x01	2 doubles: X,Y
3DPosition	0x03	3 doubles :X,Y,X
Road Map Position	0x04	Variable
Lon-Lat-Position	0x00	2 doubles: long and latitude
Lon-Lat-Alt position	0x02	3 doubles: longitude, latitude and altitude
BoundaryBox	0x05	4 doubles: xmin, ymin, xmax,ymax
Polygon	0x06	8 bit length + 2 double (x,y) per point of the shape
Color	0x11	4 ubytes: R,G,B,A
Traffic Light Phase List	0x0D	8 bit phase count + 2 strings + 1ubyte per phase

Table 13: Variable types identifiers

Appendix B: Summary of SUMO Command line

In this appendix, a summary of the commands used in the command line for generating the different files requested for both scenarios is presented.

Generation of the Grid network

```
netgenerate –g true –grid.number 6 – grid.length 80 –no-turnarounds.tls true –j traffic_light –o Grid.net.xml
```

Generation of the network from imported OSM data

```
netconvert –osm-files Technium.osm –o Technikum.net.xml
```

Generation of polygons for GUI visualization of Technikum.net

```
polyconvert –net-file Technikum.net.xml –osm-files Technkum.osm –type-file typeTechnikum.xml –o Technikum.poly.xml
```

Generation of routes from the predefined trips for the Grid scenario

```
duarouter -n Grid.net.xml -f Grid.flows.xml -o Grid.rou.xml
```

Generation of routes from randomly generated trips for the Technikum scenario

```
python randomTrips.py –n Technikum.net.xml –r Technikum.rou.xml –b 0 –e 4500 –min-distance 500
```

Grid.sumo.cfg

```
<configuration>
    <input>
        <net-file value="Grid.net.xml"/>
        <route-files value="Grid.rou.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <end value="4500"/>
    </time>
</configuration>
```

Technikum.sumo.cfg

```
<configuration>
    <input>
        <net-file value="Technikum.net.xml"/>
        <route-files value="Technikum.rou.xml"/>
        <additional-files value="Technikum.poly.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <end value="4500"/>
    </time>
</configuration>
```