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

A simulation environment for vehicle-pedestrian interaction

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Leire Artal Villa
Cristina Olaverri Monreal (FH Technikum Wien)
Luis Serrano Arriezu (UPNA)
Viena, 8 de junio de 2018

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 (see Urheberrechtsgesetz/ Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. 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 Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

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

Kurzfassung

Der Schutz gefährdeter Straßenbenutzer (VRU) war in den letzten Jahren ein aktives Forschungsthema. In diesem Zusammenhang sind P2V *(Pedestrian-to-Vehicle)* und V2P *(Vehicle-to-Pedestrian)* Technologien aufgrund der hohen Erkennungsraten und der hohen Benutzerzufriedenheit, die sie erreichen, zu den entscheidenden Technologien geworden, um mögliche Gefahren zu reduzieren. Die letzten Versionen von SUMO *(Simulation of Urban Mobility)* und TraaS *(TraCI as a Service)* zeigen die Möglichkeit, sowohl P2V- als auch V2P-Kommunikation in einer simulierten 3D-Umgebung mit mikroskopischer Modellierung von Fahrzeugen und Fußgängern zu entwickeln und die Möglichkeit, die Effizienz des vorgeschlagenen Systems zu testen.

In dieser Bachelorarbeit wurde eine 3D-Simulation mit Fußgängern unter Verwendung einer leistungsstarken Spiel-Engine (Unity 3D) durchgeführt, die Daten aus einem Open-Source-Verkehrssimulator über eine TPC-Verbindung zur Simulation von Fahrzeug-Fußgänger-Interaktionen nutzt. Zur Validierung der Simulation wurde ein Anwendungsfall vorgeschlagen.

Schlagwörter: VRU, P2V, V2P, SUMO, Fußgänger

Abstract

Protecting Vulnerable Road Users (VRUs) has been an active research topic in recent years. In this context, P2V (Pedestrian-to-Vehicle) and V2P (Vehicle-to-Pedestrian) technologies have become the crucial technologies to minimize potential dangers, due to the high detection rates and the high user-satisfaction levels they achieve. The recent release of the last version of SUMO (Simulation of Urban Mobility) and TraaS (TraCI as a Service) reveal the possibility of developing both P2V and a V2P communications in a simulated 3D environment with microscopic modeling of vehicles and pedestrians, as well as the possibility of testing the efficiency of the proposed system.

In this Bachelor-Thesis a 3D simulation with pedestrians has been performed, using a powerful game engine (Unity 3D) which takes data from an open source traffic simulator, capable of simulating vehicle-pedestrian interaction, through a TPC connection. For the purpose of validating the simulation a use-case has been proposed.

Keywords: VRU, P2V, V2P, SUMO, pedestrian

Resumen

Proteger a los usuarios más vulnerables de la carretera (VRU) ha sido un tema de investigación activo en los últimos años. En este contexto, las tecnologías P2V (*Pedestrian-to-Vehicle*) y V2P (*Vehicle-to-Pedestrian*), conocidas por las altas tasas de detección que consiguen, así como la gran satisfacción de usuario lograda, constituyen un instrumento importante para minimizar el peligro de accidentes. El reciente lanzamiento de las últimas versiones de SUMO (*Simulation of Urban Mobility*) y TraaS (*TraCI as a Service*), revelan la posibilidad de llevar a cabo tanto comunicaciones P2V como V2P en un entorno simulado y tridimensional con modelado microscópico de vehículos y peatones, así como la opción de testear la eficiencia del sistema propuesto.

Con este Trabajo de Fin de Grado se pretende realizar una simulación en 3D que incluye peatones, utilizando un motor de video-juegos potente (Unity 3D), que mediante una comunicación TCP es capaz de tomar los datos obtenidos de un simulador microscópico (de código abierto) en 2D. El trabajo concluye proponiendo un caso práctico en el que se aplica el sistema propuesto.

Palabras clave: VRU, P2V, V2P, SUMO, Peatón

Laburpena

Errepidearen erabiltzaile ahuleenak (VRU) babestea ikerketa gai aktiboa izan da azkenaldian. Horren harira, hala P2V (Pedestrian-to-Vehicle) nola V2P (Vehicle-to-pedestrian) teknologiak tresna garrantzitsu bilakatu dira trafiko-istripuak murrizteko nahian, detekzio-tasa altuez gain erabiltzaileen artean ere arrakasta handia lortu dutelako. SUMOn (*Simulation of Urban Mobility*) eta TraaSen (*TraCI as a Service*) egindako azken eguneraketei esker, posible da P2V eta V2P teknologiak hiru dimentsiotako inguru simulatu batean ezartzea.

Lan honen helburua, oinezkoak dituen hiru dimentsiotako trafiko-simulazio bat burutzea da, bi programen (SUMO, Unity 3D) arteko TCP komunikazioan oinarrituz, P2V eta V2P teknologiak frogatzea ahalbidetzen duena. Halaber, aipatu komunikazioaren bitartez, bi dimentsiotako simuladore zehatz batek lortutako datuak inguru errealago batean irudika daitezke. Azkenik, lanak izan dezakeen erabilera praktiko bat aurkezten da.

Hitz gakoak: VRU, P2V, V2P, SUMO, oinezko

Acknowledgements

First of all, I would like to thank my supervisor in Vienna, Dr. Cristina Olaverri, for her patience, continuous support and enthusiasm. I would also like to show gratitude to her for all the time she spent reviewing my writing, which I really appreciate.

Thanks also to Dr. Luis Serrano and Mag. Florian Ellinger for checking my subjects many times without complaining, making this stay possible.

I must express my gratitude to all the people who have previously worked in the simulator. Special thanks to Ahmed Hussein who taught me many strategies to deal with Unity 3D in the beginning of this work and really helped me to improve my programming style. Without him my pedestrians would still be jumping. I also offer my gratitude to Alberto Díaz for his funny and all-inclusive e-mails and also for his advices with SUMO and the orientation provided. I am also indebted to Javier Errea, who offered me his help before coming to Vienna, encouraged me to take part in this project and shared his experiences with me.

Besides, thanks to my office-friends. Special thanks to Vicente and Liu, for our long conversations in the evenings when nothing was working. Thanks also to Khurram for helping me to bring people for the experiment and also to Daniel for his availability while performing the experiment.

Beyond programming, thanks to all the people I met in Vienna, who contributed to the driving experiment: Alberto, Giuseppe, Pol, Juan, Néstor, Lars, Leandro, Rodrigo, Melissa, Hana, Matías and Denis. Thanks also for all the trips, parties and great moments we have spent together. Danke an Tim, für seine Hilfe bei der Kurzfassung.

Last but not least, I acknowledge the support of my family. Specially the support of my mother, Isabel, who has always been an example of personal strength to face difficulties, has made an effort and has encouraged me to go abroad. Special thanks also to Elena who has always supported me.

Table of Contents

1 Introduction ... 7
1.1 Motivation and state of the art ... 7
1.2 Research aim ... 9
1.2.1 Research question and hypothesis ... 9
1.3 Structure of the Thesis ... 10

2. Literature review ... 10
2.1 Pedestrian detection ... 10
2.2 Vehicle blind spot and driver visibility ... 12
2.3 Pedestrian modeling in SUMO ... 13

3 Background ... 14
3.1 3DCoAutoSim ... 14
3.2 SUMO-environment ... 15
3.3 Data-Logging and device selection ... 16

4 Methodology ... 17
4.1 The simulation in SUMO: 2D scenario ... 17
4.1.1 Generation of a realistic network from OSM data ... 17
4.1.2 Edition of the network with *netedit* and *netconvert* ... 18
4.1.3 Generation of vehicle routes and pedestrian demand ... 20
4.1.4 Executing a simulation with pedestrians in SUMO-GUI ... 22
4.2 Connecting SUMO with Unity 3D: TraCI Protocol and TraaS library ... 22
4.2.1 Translating approach ... 24
4.3 The simulation in Unity: 3D scenario ... 24
4.3.1 *Environment* Game-Object created by CityEngine with data from the city of Vienna 27
4.3.2 Classication of roads retrieved from SUMO: instantiating sidewalks and pedestrian crossings ... 28
4.3.3 Instantiating and moving pedestrians retrieved from SUMO ... 31
4.3.4 Manually placed and coordinated traffic lights ... 39

4.3.5 Road markings placed manually 45
4.3.6 Bidirectional communication system between cars and pedestrians using P2V and V2P. 46
4.3.7 Logging the position of every pedestrian using GPS devices 57
4.4 Experimental setup 60
4.4.1 Instructions 60
4.4.2 Familiarization phase and driving tests 60
4.4.3Post-task questionnaire 61
4.5 Data collection and parameters of interest 62
4.5.1 Driver performance metrics 62
4.5.2 Driver performance calculations and parameters 62
4.5.3 Methodology for analyzing simulation data 63
5 Results 67
5.1 Objective results 67
5.2 Subjective results 69
6 Conclusions 70
Bibliography 71

1 Introduction

1.1 Motivation and state of the art

Even if the number of accidents in the European Union has decreased a lot in recent years, traffic accidents are still one of the major causes of death [2].

In 2013 both passenger cars and pedestrians road accident fatalities were the major causes of death in European Union, as it can be seen in Figure 1. Recently, some steps have been taken in the field of ITS to reduce the aforementioned accidents.

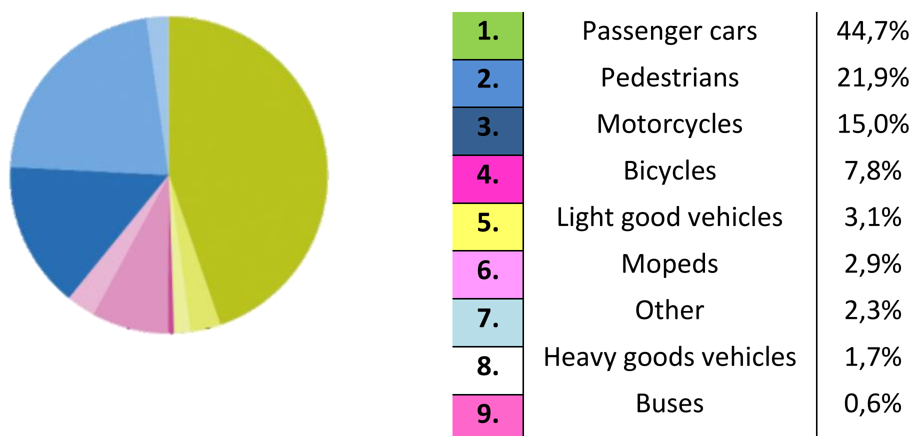

Figure 1: Road accident fatalities by category of vehicle in European Union in 2013

For example, authors in [13] tried to select the most effective ITS approach that can improve safety in certain intersections using the Vehicle-to-Pedestrian (V2P) and the Pedestrian-to Infrastructure (P2I) communications and technologies as traffic signal adjustments, Automated Pedestrian Detection (APD) and Countdown Pedestrian Signal (CPS). Furthermore, contributors in [14] analyzed some ITS applications for pedestrian safety and they tried to find the most suitable one for a particular city.

There are also some works that are focused on pedestrian detection for ITS [12]. In particular, authors in [15] develop an application which is based on a collision-prediction algorithm. By using both P2V and V2P communications they achieve good results in trying to identify if there is a possibility of collision as well as in increasing the acceptance of autonomous vehicles on the roads. Authors of the work in [16] present a method of detecting and tracking pedestrians from a moving vehicle using both Histogram of Oriented Gradients (HOG) and the Kalman filter. Besides, a work in [17] is focused on detecting pedestrians with the back-camera of a mobile device using image-processing techniques while the front-camera of the mobile device is used to locate the driver's face position and see if a TOR-Take Over Request- is needed. That is, combining the outputs of both cameras they present a cost effective mobile application capable to detect pedestrians in a range from 5 to 40 m.

Anyway, still about 2000 people die due to the movement of large human crowds every year [3]. The association for safe international road travel states that road crashes account for 2,2 percent of all deaths globally and predicts that road injuries will become the fifth leading cause of death by 2030 [17][18].

In fact, pedestrians play an important and indefinite role in the traffic system because of their complex and unpredictable interactions with vehicles. Due to the high rate of accidents caused by pedestrians, some works have also been carried out in the field of pedestrian simulation, since it stands a valuable tool for analyzing different scenarios without having to risk people's life.

Although simulation of transportation systems started over forty years ago, it still is an important area of discipline nowadays. Various national and local transportation agencies, academic institutions and consulting firms use simulation to aid in their management of transportation networks. Nevertheless safety could be the overall purpose of traffic simulation [1].

Authors in [7] estimate models for pedestrian and bicycle crashes and contributors in [8] review several approaches to macroscopic pedestrian modeling. The work done in [9] would be an example of a 2D simulation using microscopic models for pedestrians. In fact, there are plenty of examples of works that use microscopic modeling for pedestrians [11].

As far as simulation tools are concerned, PARAMICS VISSIM, AIMSUM and SUMO stand among the most recognized simulators which use microscopic models and allow to include a pedestrian flow in the simulation [4]. For example, PARAMICS has its own software to simulate pedestrian behavior in real word environments. VISSIM allows to make a 3D simulation with pedestrians, but it fails in the calibration process of certain parameters and it is considered to be a difficult program to handle [5]. AIMSUM enables pedestrian-vehicle interactions at uncontrolled, actuated-controlled or fixed-controlled intersections, but it is not convenient for navigating between different time periods and it does not provide background maps [6]. Finally, The recent release (2017-12-19) of the last version of SUMO (0.32.0) together with the last version of TraaS (2017-08-16) reveal the possibility of performing a remote-controlled realistic 3D traffic scenario which includes pedestrians. The aforementioned version of SUMO implements TraCI-Traffic Control Interface-, which provides the necessary commands for both remotely retrieving and changing the state of pedestrian objects. By means of TraaS library and the above-named methods, authors in [1] are able to retrieve the information (vertices, length, width, type) of lanes and cars (speed, position, angle) from SUMO and insatiate them as game objects in Unity 3D. Authors in [10] reuse the translated library by retrieving information of Traffic Lights from SUMO and taking it from Unity 3D.

However, little work has been done for developing a real-time 3D visualization of a traffic simulation which includes both pedestrians and a driver centric perspective at the same time. Works in [1] and [10] stand as an evidence that a driver-centric simulator which includes a pedestrian flow could be achieved by combining SUMO and Unity 3D. By integrating pedestrians in the currently implemented simulators, more realistic and optimized data could be obtained.

1.2 Research aim

The aim of this Bachelor Project is to **develop a pedestrian detection system in a simulated vehicle.** The proposed system will be based on a bidirectional communication between cars and pedestrians, that is, the system will mimic real-life P2V and V2P communications. In fact, the connection between SUMO and Unity can reproduce a real-life connection between:

- Any pedestrian who has a smartphone equipped with an application running on the background that uses the GPS localization sensor to access the pedestrian's current location and a magnetometer sensor to access the pedestrian's current orientation angle in a similar manner as authors in [15] do.
- Any car which is nearby and it is broadcasting its location information.

In fact section 4.3.3. proves that what Unity retrieves from SUMO is the position of any simulated pedestrian and car every time-step. Therefore P2V and a V2P will be developed in the environment, since both simulated cars and pedestrians can be aware of each other's position due to the established connection between SUMO and Unity.

Note that only the P2V technologie will be tested in this bachelor thesis. A future version of the simulator will include the possibility of controlling both a simulated car and a simulated pedestrian through two computers and therefore it will be possible to test also the implemented V2P technologie .

1.2.1Research question and hypothesis

The following research question was defined to be tested within the described framework:

RQ: does the proposed pedestrian detection system assist the driver to drive in a better way, improving the driving performance?

Within this work the following hypothesis will be tested:

H1: having a P2V system which is in charge of estimating a collision point between every pedestrian and the simulated car, when the distance between them is less than 40m will help to reduce the amount of accidents, by:

- displaying a green square in the HUD when the time to reach the collision point is less than 5s.
- warning the driver with an acoustic alarm

1.3 Structure of the Thesis

The thesis contains 6 chapters. Chapter 1 includes the introduction, the motivation, the state of the art and the research aim. Chapter 2 is in charge of reviewing the basic literature related to the thesis. The aim of chapter 3 is to delineate where the personal contribution begins, since this bachelor thesis is a part of a larger project. Chapter 4 shows the step-by-step methodology. Finally, chapter 5 contains the results and its discussion and chapter 6 concludes the thesis.

2. Literature review

Protecting vulnerable road users (VRUs) has been an active research topic in recent years. The European Commission defines VRUs as “non-motorized road users, such as pedestrians and cyclists and persons with disabilities or reduced mobility and orientation” [21].

2.1 Pedestrian detection

The detection of VRU stands as a solution to enhance road safety. Among the different methods to detect VRU, two could be highlighted.

- Perception based VRU protection: perception is the organization, identification, and interpretation of sensory information in order to represent and understand the presented information, or the environment. Therefore, perception-based methods consist of detecting pedestrians by using different types of active sensors, such as radars and ultrasonic sensors. Looking back, a wide variety of sensors have been shown to be efficient to detect and locate obstacles situated in their FOV-Field Of View- [19]. In some works video cameras are used as sensors in order to recognize VRU-s [26]. Nevertheless, in 2010 the number of false alarms was still too high to allow the deployment of this kind of systems in commercial vehicles and more reliable systems were developed [19]. Nowadays pedestrian tracking stands as one of the most reliable tools to detect pedestrians. A variety of methods have been developed for tracking single or multiple pedestrians in static or moving cameras by exploiting different types of image information:

- Background subtraction: it consists of measuring the differences between video frames and a learned background model. This kind of systems are known to be fast and often have the additional advantage of recovering foreground object silhouettes. For example, the University of Cranfield uses a combination of background separation and optical flow for tracking pedestrians [22].

- Appearance and motion: it consists of identifying tracks of image regions that look like people. These methods can be advantageous in dense crowds where occlusions can cause other methods to fail, but are usually more computationally expensive than background separation based methods. Authors used this kind of systems for identifying pedestrians carrying objects [22].

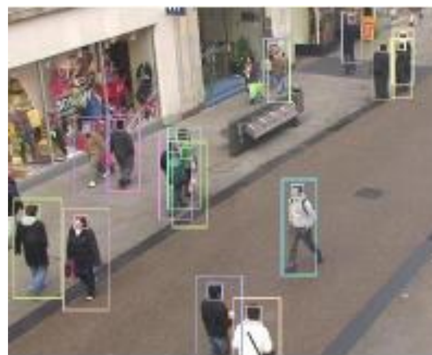

Figure 2: example of an appearance and motion-based pedestrian detection system

On the other hand, Haar-like features have also become popular in this area. Yet, it should be noted that corresponding features are either determined by means of exhaustive searches over all possible variations [23] or by means of less exhaustive random sampling [24]. Related to this, contributors in [25] found out that although Haar-like templates perform well for face detection are not necessarily suited for pedestrian detection as they may fail to capture visual characteristics of human body. As a remedy, they propose to design particularly tailored templates for up-right body shapes.

However, the arguably most popular features for visual pedestrian detection are based on Histogram of Oriented Gradients (HOGs) as introduced in [16]. HOG features brought about significant improvements and therefore establish an important baseline.

The main disadvantage of perception-based methods is that they are not able to detect obstructed pedestrians with the anticipation of communication based systems.

As shown in [15], the GPRS-based method proposed is able to detect a pedestrian that crosses the road behind another vehicle obstructing the view of an oncoming vehicle while usual perception-based method is not.

- Communication based VRU protection: it is based on the use of smartphones or other personal devices to send Personal Safety Messages (PSM), over a wireless communication channel [20]. In some works they are called GPRS-based methods, since they usually use the smartphone of the VRU-s uses the latitude and longitude coordinates, which are then transformed into local coordinates to estimate the relative position of the communicating pedestrians and vehicles. Some examples of this protection methods would be Vehicle-to-Pedestrian (V2P) and Pedestrian-to-Vehicle (P2V) communications or a combination of them.

However, communication-based methods also represent many technical challenges. One of the problems is the high channel usage which is caused by an overwhelming amount of VRU devices transmitting PSMs through a channel of limited bandwidth as stated in [20]. The overloaded channel results in significant transmission errors for PSMs and may degrade the networking performance for all other types of messages sharing the same wireless medium. This kind of methods are also known for having strong requirement on communication delay and energy consumption at the VRUs side [19]. Besides, Global Positioning System (GPS) readings are frequently inaccurate in urban canyons and the error provided can be more than tens of meters. Thus, some recent works show that better results can be obtained by fusing both perception and communication techniques [20].

Finally, some authors propose to use estimation filters [15] as an alternative of their systems to ensure a more accurate estimation of the localization through a higher calculation frequency.

2.2 Vehicle blind spot and driver visibility

In transport, driver vision has been identified as a significant factor. Driver visibility is the maximum distance at which the driver of a vehicle can see and identify prominent objects around the vehicle [28]. There are some parts in a vehicle that influence visibility:

- The windshield
- The dashboard
- The pillars

For example, blind spots can occur when the A-pillar blocks a driver’s view of the road. A blind spot in a vehicle is an area around the vehicle that cannot be directly observed by the

driver, under existing circumstances [27]. As far as pillars are concerned, they are defined as the vertical supports of a car's window area [29].

There are plenty of evidences that show that A-pillars do impede the driver's forward field of view [27][30].In particular, contributors in [30] state that A-pillar obscuration is a greater problem in newer, as opposed to older, cars and they find out that the obscuration areas which are most affected by the A-pillars are:

- Near-side: from 55° to 65° (i.e. a range of 10°) to the left of the driver's line of sight.
- Off-side: from 22° to 40° (i.e. a range of 18°) to the right of the driver's line of sight.

2.3 Pedestrian modeling in SUMO

Pedestrians are persons that walk. As in real-life, in SUMO pedestrians have special lanes and areas which differ from each other according to its permissions.

As far as pedestrian behavior is concerned, there are many rules that should be taken into account when analyzing a simulation in SUMO. When walking along an edge, pedestrians use sidewalks where available. With respect to zebra crossings, there are two possible cases. If the network contains walkingareas, pedestrians may only cross a street whenever there is a pedestrian crossing. However if the network does not include walkingareas, pedestrians will teleport between any two edges which allow pedestrians at an intersection.
Nevertheless, the behavior that pedestrians have when crossing a road does not only depend on the type of roads and of the models, but also on the cars (except in non-Interaction models).

SUMO tries to mimic the real world by trying to avoid collisions between cars and pedestrians. In the simulation, pedestrians will only use a crossing if the whole length of the crossing is free of vehicles for the whole time needed to cross. Like in real-life, vehicles do not drive towards an occupied pedestrian crossing. If a vehicle occupies the whole width of the lane and gets too close to a pedestrian, the pedestrian may briefly move to the side of the lane in order to let the vehicle pass.

Finally, the last version of SUMO provides the possibility of generating a pedestrian demand both in an explicit and in a random way:

- Explicit way: manually defining the pedestrian movement in a .xml file.
- Random way: using the tool randomTrips.py with the option --pedestrian which supports generating random pedestrian demand.

3 Background

The aim of this section is to delineate where the personal contribution begins, since this bachelor thesis is a part of a larger project. To be precise, a scenario of a simulator (CoAutoSim3D) was chosen for the purpose of enhancing its capabilities. Thus, the characteristics of the chosen scenario will be briefly explained in this section.

3.1 3DCoAutoSim

The simulator has an architecture which is independent from the operating system and always up-to-date to the latest version of Unity-3D. As it is presented in [2] in detail, when running 3DCoAutoSim the user can choose among 3 environments:

1) FHTW environment: a 3D reconstructed map of an area of Vienna, near the “Fachhochschule Technikum Wien”, with high quality visualization to emulate a realistic environment.
2) Race City (RC) environment: provided as a testing environment by the Realistic Car Controlled asset in Unity.
3) SUMO-environment: a dynamic and configurable environment by the user.

After analyzing the three environments, the SUMO-environment was considered interesting to build the bases of this project, since it allows to take advantage of the benefits provided by a microscopic 2D traffic-modeling tool (SUMO) and a powerful game-engine (Unity 3D). In fact, due to the flexibility provided by this option, the user can perform a 3D realistic-simulation of any area of interest. Furthermore, the 3D reconstructed map of 1) was used in order to maintain the high quality visualization provided by the FHTW environment. Apart from these environments 3DCoAutoSim has the capabilities explained in [2].

Figure 3: 3DCoAutoSim main menu graphical interface

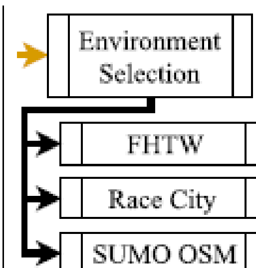

Figure 4: available environments in 3DCoAutoSim

3.2 SUMO-environment

When the user choses this scenario, the simulator takes a sumo configuration file (.cfg or .sumocfg) and performs the TCP connection to SUMO. SUMO acts as a server and Unity as a client. The configuration file contains all the necessary information to conduct the simulation in SUMO and therefore by loading it in Unity 3D, the simulation can be performed in a 3D scenario. The procedure is graphically shown in figure 5.

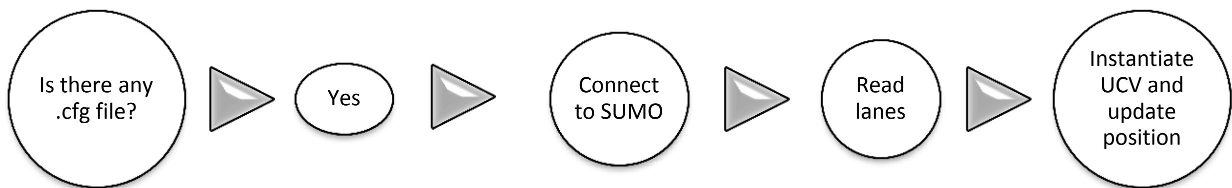

Figure 5: schematic representation of the first steps performed by CoAutoSim3D

After establishing the connection, all the roads contained in the .cfg file (which has a link to the .net file created from OSM data) are read in Unity. Thus, a parent Game-Object is created for the roads which contains many children Game-Objects obtained by successively instantiating a prefab which represents a single road-section. Finally, the simulation starts. For each game loop a new simulation step is performed in the micro-simulator through the API, asking for the position of the System-Controlled Vehicles (SCV) and communicating the current position of the User-Controlled Vehicle (UCV).

Figure 6: SUMO environment of CoAutoSim3D when a .cfg file is loaded containing basic environment to be further developed.

3.3 Data-Logging and device selection

3DCoAutoSim uses a simple architecture which comprises only two classes, `DataLogger` and `DeviceDataLogger` with the purpose of data logging.

- `DataLogger` is the one responsible to provide of different loggers for different devices.
- `DeviceDataLogger` is the specific logger for a device.

Authors in [2] Implement a configurable logger and abstract it from devices implementer so in the future, when a person implements a loggable device, if they follow the guide of implementing driver, the logger gets it automatically. SQL logger configurable to log in different database managers (such as Postgre or MySQL). They implement an output manager to record data during the experiment. The output manager is implemented to find the selected outputs by the user, start reading them from the environment and record them according to the saving options. Currently the saving options are limited to CSV files, however, My-SQL database and ROS messages are work in progress to be included as well. The logging frequency is 50 Hz (0.02 s).

Each enabled device has a corresponding `DeviceDataLoggerobject` associated. It's method `Log` is called with the device data and the implementing classes are required to handle this data properly. To be precise, the data is logged in a .CSV file. Thus, if this system of data-logging is to be maintained, in order to have an output saved, **a device is needed**, which will be linked to a Game-Object and will acquire data. The devices available are a GPS, a lidar and a camera and they are further explained in [2]. The contributors of the work used the devices for measuring parameters of the user-controlled car. In fact, they have a function *EnableSelectedDevices(GameObject player)* which takes the Game-Object of the user-controlled car as an argument and makes active the Game-Objects that represent the devices that the user has selected.

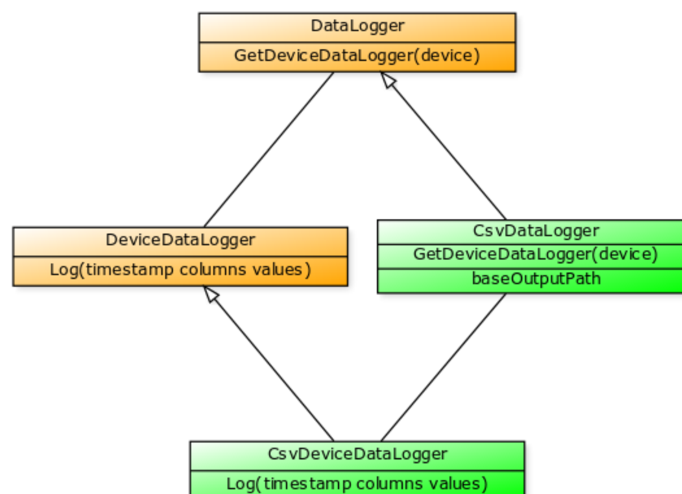

Figure 7: Logging framework design[2]

4 Methodology

This chapter explains the methodology for implementing a 3D simulation with microscopic modelling of vehicles and pedestrians, which also includes a pedestrian-detection system.

4.1 The simulation in SUMO: 2D scenario

The approach taken for explaining the generation of the 2D scenario can be divided in four subsections.

- Generation of a realistic network from OSM data
- Edition of the network with *netedit* and *netconvert*
- Generation of the vehicle routes and the pedestrian demand
- Executing a simulation with pedestrians in SUMO-GUI

Firstly, in order to have a realistic scenario a network had to be imported from OSM-Open Street Map-. Secondly, the network had to be edited with *netedit* since the conversion process included some imperfections that made the simulation differ from reality. *Netedit* is a GUI-Application for editing traffic networks. It can be used to create networks from scratch and to modify all aspects of existing networks. *Netconvert* is a network generator that provides SUMO-format networks. After obtaining the network file (.net file) and generating both the vehicle routes and the pedestrian demand, the simulation could be executed in SUMO-GUI, which is basically the same application as SUMO, just extended by a graphical user interface.

4.1.1 Generation of a realistic network from OSM data

For achieving this purpose *osmWebWizard.py* was used, which is a script provided in the tools folder of SUMO and it allows to search for an area of a city in a map. For this thesis, an area near the “Fachhochschule Technikum Wien” was selected. After selecting the area of interest, the script provided the following files:

- A .net.xml file that could be edited with *netedit* and contains all the information about the network: roads, connections between roads and intersections.
- Some .rou.xml files that described both the vehicle routes and the pedestrian demand generated for that particular network.
- Other files with additional information: a .poly file and a .view file that could be used to have a more detailed view.
- A .cfg file which is a link to the .rou and .net files and allows to visualize the simulation.

However, since the network was going to be edited, only the .net.xml file was needed. The .rou.xml files could have not been taken, because they had to be consistent with the .net.xml file obtained. The edition process, as it will be explained in the following section, consisted of removing some roads. Therefore, the routes that were described in the .rou.xml file were not

valid anymore and another .rou.xml files had to be generated according to the new .net.xml file, obtained after editing the network.

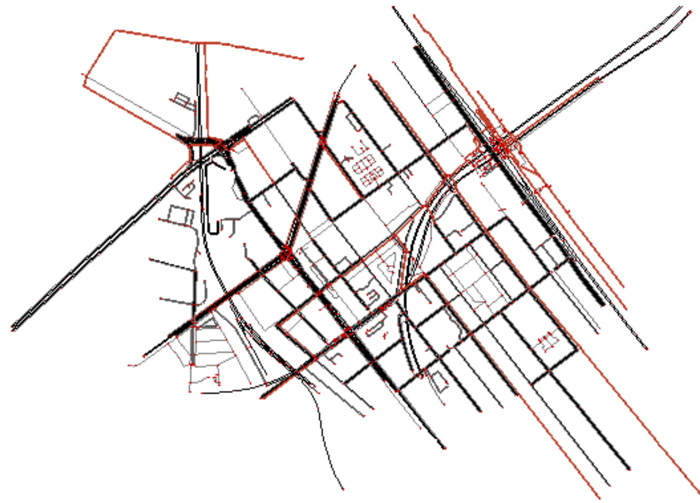

Figure 8: .net file obtained from *osmWebWizard* before editing

4.1.2 Edition of the network with *netedit* and *netconvert*

Due to the lack of information in the OSM data, the .net.xml file obtained contained some errors that had to be corrected. Some of those errors were improved manually using *netedit* and some others using the *netconvert* tool of SUMO, together with certain parameters. The errors were:

- Unsynchronized TLS programs that resulted in traffic jams and inconsistencies.
- Invalid connections that made the cars move in an unrealistic manner. In SUMO, vehicles are allowed to make U movements, while in reality it is not possible.
- Invalid pedestrian topologies.

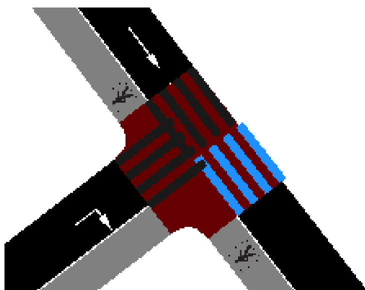

Figure 9: example of a pedestrian topology in *netedit* before edition

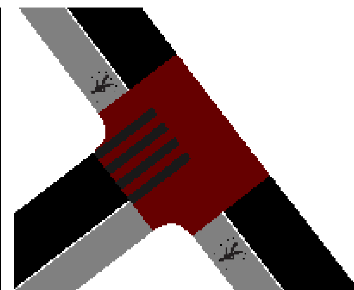

Figure 10: example of a pedestrian topology in *netedit* after edition

Figure 11: example of lane connections in *netedit* before edition

Figure 12: example of lane connections in *netedit* after edition

In addition, some lanes had to be removed so that Unity will not have to perform unnecessary computations. For example, the railways corresponding to the *S-Bahn* were deleted, since the current simulator does not include trains. Furthermore, some car lanes that were outside of the desired environment were deleted. The obtained network can be seen in figure 13.

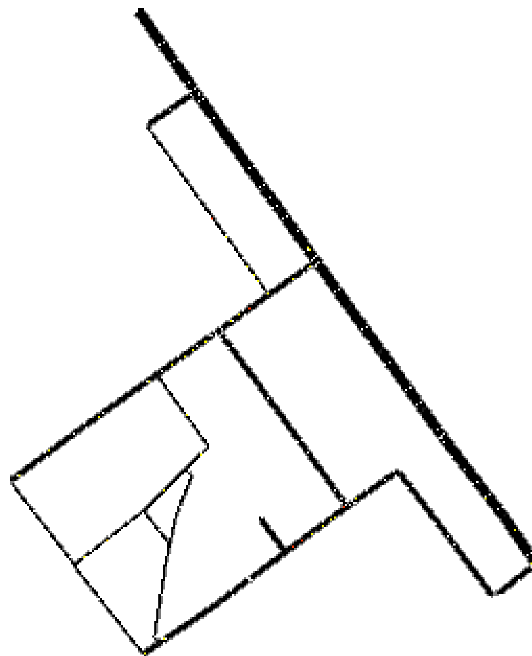

Figure 13: network obtained after the edition process

After obtaining a cleaner version of the network, it was further edited using the *netconvert* tool with the following attributes:

- *Junctions.join*, that joins junctions that are close to each other and it is recommended when importing OSM data as it helps to avoid sharp turns found at the start of some edges.
- *tls.join* for trying to cluster tls-controlled nodes.

Apart from the aforementioned attributes, some other attributes were used with different purposes. Since pedestrians are to be included in the simulation, sidewalks (lanes that allowed pedestrians) were added. Although some sidewalks were already generated, due to the current pedestrian models used by SUMO it was necessary to ensure that at least a sidewalk was present in each edge. Therefore, *netedit*'s possibilities to add sidewalks were explored.

For adding sidewalks with *netconvert*, there are two attributes:

- *sidewalk.guess*, used for guessing pedestrian sidewalks based on edge speed.
- *sidewalks.guess.from-permissions*, that adds sidewalks for edges that allow pedestrians on any of their lanes regardless of speed.

For the purpose of this thesis the second outcome was chosen, since edge permissions were available in the .net.xml file and the first outcome was considered too subjective.

4.1.3 Generation of vehicle routes and pedestrian demand

To determine a simulation, it is not enough with the information of the roads (.net.xml file). At least, cars and their routes are needed (rou.xml file) and in this thesis pedestrians and their routes will also be included.

In this subsection the generation of a vehicle-route file will be explained (carRoutes.rou.xml), which describes in .xml the routes that randomly created vehicles will have in the realistic scenario near the *Fachhochschule Technikum Wien*. Similarly, randomly generated pedestrian demand will be obtained and described in the file ped.Routes.xml.

Among the possible ways of generating pedestrian demand (explicit and random), the random approach was chosen, since it is considered more realistic than the explicit way and therefore it suits the purposes of this thesis. The way to obtain the aforementioned demand is to call the script *randomTrips.py* provided in the tools folder of SUMO. It takes a .net.xml file as an input and calls DUALROUTER, which is in charge of computing the routes for a specific network. The script can be called with some additional arguments that help to customize the obtained routes.

In order to obtain the vehicle demand the script was called in the command line of SUMO:

```
[Path where the Python executable is located] [Path where randomTrips.py is located]
–n [name of the input .net file] –r [name that the output .rou file will have] –b 0 –e [length
of the interval in seconds]
```

The attributes that were used in the first trial were:

- -b and –e that were used for specifying an interval of 1000 seconds.

As a result of this command, the file carRoutes.rou.xml was generated in the current directory. For generating the pedestrian demand, the same script was called, but this time including an additional attribute(--pedestrians)

```
[Path where the Python executable is located] [Path where randomTrips.py is located]
–n [name of the input .net file] –r [name that the output .rou file will have] –b 0 –e [length
of the interval in seconds] --pedestrians
```

The usual path for *randomTrips.py* is “C:\Program Files (x86)\DLR\Sumo\tools\randomTrips.py”.

Thus, the file pedRoutes.rou.xml was generated in the current directory, containing the desired random routes for the simulated pedestrians. However, the simulation suggested that some additional attributes should be included, since a traffic jam was obtained.

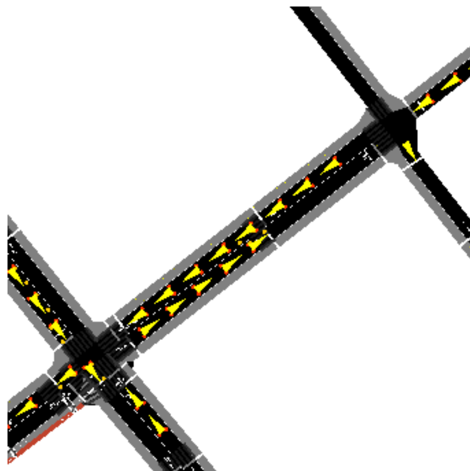

Figure 14: traffic jam obtained after running the first .cfg file in SUMO-GUI

Once again, the script was called both for generating vehicle and pedestrian demand, but this time the following additional attributes were included:

- Period: it is used for generating a fixed number of vehicles according to the formula:

Period =(end - begin) / (number of vehicles desired)

- Intermediate: it was set to a large value to ensure that vehicles remain in the simulation long enough.

Thus, the carRoutes.rou.xml and pedRoutes.rou.xml files were obtained again by means of the following commands:

```
[Path where the Python executable is located] [Path where randomTrips.py is located]
–n [name of the input .net file] –r [name that the output .rou file will have] –b 0 –e [length of the interval in seconds] –period [value of the period] –intermediate[value of the intermediate]
```

The selected values, which provided a good performance where period = 15 and intermediate = 500 both for creating the car routes and the pedestrian routes.

4.1.4 Executing a simulation with pedestrians in SUMO-GUI

At this point, a configuration file must be created. It gathers the information of the roads, the vehicles, the pedestrians and the routes by linking the following files:

- Network file: TechnikumPedestriansEditedC.net.xml
- Vehicle routes: carRoutes.rou.xml
- Pedestrian routes: pedRoutes.rou.xml

Therefore, it contains all the necessary information to perform the simulation.

After analyzing the simulation in SUMO, the pedestrian with id 0 had a route which was considered that could lead to an accident. Therefore, the route of five pedestrians were removed, and the same route that pedestrian 0 had was applied to those 5 pedestrians in order to increase the probability of having an accident. This fact will help to test the P2V system proposed, which is explained in section 4.3.6.

4.2 Connecting SUMO with Unity 3D: TraCI Protocol and TraaS library

3DCoAutoSim connects both a microscopic traffic simulator, SUMO, and a powerful 3D Graphic Engine, Unity 3D, by setting a TCP connection between them, which is explained in detail by the authors in [1].

The API used for that purpose is called TraCI and it is based on a client/server architecture. It allows to retrieve values of simulated objects from the server (SUMO) to the client (Unity 3D) and to manipulate their behavior on-line, as well as to handle the whole simulation from the client.

For this purpose, authors in [1] stated the need of a library which could be imported in Unity 3D and contained the necessary TraCI methods for dealing with the simulation duties. They chose TraaS (TraCI as a Service) since it allows to perform a remote control of SUMO as well as an immediate and easy interaction with the traffic simulation. In order to use the library with that purpose, contributors in [1] stated the need of a translating tool that would translate the whole library, since the programming language used by Unity 3D is C# while TraaS is written in Java. They found IKVM.net and used it to make the translating process. IKVM.net is an implementation of Java for the Microsoft .NET Framework.

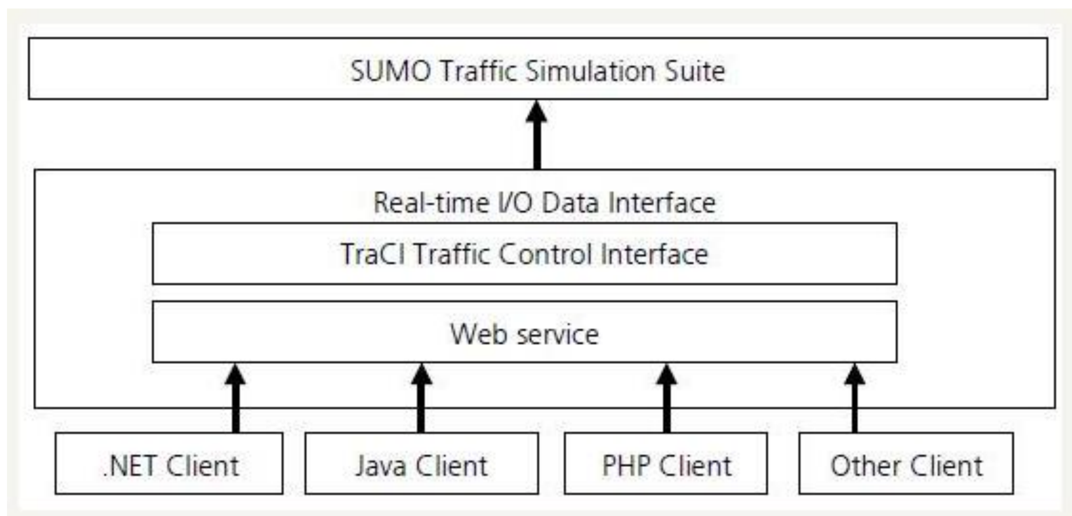

Figure 15: TraaS web service implementation

Therefore, if the aforementioned TraaS library included a class for pedestrians, it could be used for both retrieving and changing information about pedestrian objects between SUMO and Unity 3D without changing the internal logics and structure of the already implemented and proved simulator (3DCoAutoSim). In particular, if TraaS contained the class “Person” the available commands of “Person value retrieval” from TraCI could be used. An example of these commands *is Person.getIDList* that would send to Unity 3D a list of ids of all persons currently running within the SUMO scenario.

In addition, if the already translated library [1] had contained the necessary methods for pedestrians, the translating procedure would not have to be repeated. However, since the person class was not even included when the translating procedure was made, the library must be translated again. The *ikvmc* tool used in [1] is considered appropriate for this purpose, but the approach taken in this thesis for translating the library differs in some steps from the one taken in [1] since contributors removed plenty of classes needed for this project (WebService, Traci…).

4.2.1 Translating approach

The translating approach followed these steps:

1-. Download the last version of TraaS.jar library from its webpage and download of IKVM from its webpage.

2-. Extraction of the files and analysis of the classes contained in the file: some of them for implementing the TCP protocol (*Command.class*, *RequestMessage.class,…*), for handling the connection (*SummoTraciConnection.class*) and for obtaining information about SUMO variables (the desired *Person.class*).

3-. Due to the necessity of all the classes contained in TraaS, there is no need for a rebuilding step. A deleting procedure was considered, since the translating procedure makes the library very heavy.

4-. Setting up of the environment variables: the following environment variables must be added in order the translating procedure to work:

- The directory containing the IKVM executables.
- The directory containing the C# compiler (Windows: csc / Mono: mcs). On Windows, this is typically C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322.
- The directory containing a Java compiler (*javac* or *jikes*)

5-. Locate the TraaS.jar file in the same folder as the executable *ikvmc*

6-. Open the command line, go to the aforementioned directory and write the following command, which would produce TraaSconverted.dll as an output:

```
Ikvmc –out: TraaSconverted.dll TraaS.jar –target:library
```

7-. Copy the obtained TraaSconverted.dll together with the other .dll produced in the directory and locate them in the Unity 3D folder used by the simulator.
In order the translating procedure to work attention should be paid to the version of Java and IKVM used. For this thesis Java 8 and IKVM 8.1.5717.0 were used, since previous versions of IKVM were not able to translate some of the classes present in TraaS.jar and some errors appeared.

4.3 The simulation in Unity: 3D scenario

The 3D simulation contains:

1. Buildings created by CityEngine with data from the city of Vienna

2. Car lanes retrieved from SUMO
3. Cars retrieved from SUMO
4. Sidewalks retrieved from SUMO
5. Pedestrian crossings retrieved from SUMO
6. Pedestrians retrieved from SUMO.
7. Manually placed traffic lights
8. Road markings placed manually
9. A user-controlled car with capabilities explained in [2]
10. A bidirectional communication system between cars and pedestrians using P2V and V2P.
11. As many GPS-s as the number of pedestrians in the simulation.

However, in this bachelor thesis points 1,4,5.6,7 and 8 will be explained, since the functions used for points 2 and 3 were explained in detail by authors in [10].

Figure 16: Driver-Centric perspective of the 3D environment

As it is explained in [10], data can be categorized in two groups: static and dynamic. In this thesis different approaches have been taken for instantiating the Game-Objects classified in each of the groups.

For the static data, such as, buildings, trees, road-markings, sidewalks and traffic lights the approach taken consists on instantiating a single Game-Object in a certain position of the environment. Since their position is not going to change during the simulation, they must be instantiated just once.

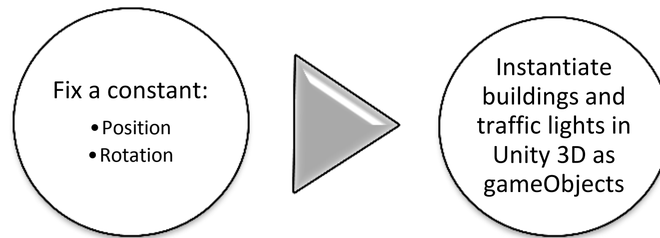

Figure 17: schematic representation of the approach used for instantiating static data

Firstly, a fixed position is defined for each static Game-Object within the environment. Then, the Game-Objects are instantiated in Unity 3D (in the mentioned positions). Every Game-object is in fact a Unity-prefab which contains more hierarchically organized Game-Objects, that is, the parent game-object named *trafficlights* contains all the traffic lights of the environment as child-objects and it will be instantiated in the position *(x,y,z) = (1493.762f,2.568404f,1620.374f)* with a rotation of 0 degrees towards the y axis of Unity 3D. The position where the traffic lights will be placed has been decided by analyzing the SUMO traffic lights and the 3D environment, that is, both Unity 3D and SUMO traffic lights have been placed in the same relative position of the roads.

The same procedure has been applied to instantiate all the static elements.

section of code for instantiating static data	SIMULATIONmanager.cs

```
else if (PlayerPrefs.HasKey("sumoConfigFile"))
        {
…
Vector3 trafficLightsPosition = new Vector3(1493.762f,2.568404f,1620.374f);
GameObject trafficlights = Instantiate(trafficLights, trafficLightsPosition
, Quaternion.identity) as GameObject;
      …
```

However, as far as the dynamic data is concerned, it may change from a SUMO time-step to another one. For example, pedestrians may change their position, since they are walking. Therefore, this kind of data will be instantiated once like in the previous case, but some of their parameters must be updated each SUMO time-step. The approach taken is carefully explained in section 4.3.3.

4.3.1 *Environment* Game-Object created by CityEngine with data from the city of Vienna

As far as the buildings are concerned, information about buildings was taken from the webpage of the City Government of Vienna in order to have a realistic scenario. Then, that information was used for creating a 3D scenario in CityEngine, which was combined with an .osm file in order to provide the scenario with the shape of the roads. Thus, a .fbx file was obtained, which was further imported in Unity 3D. This process was not made in this bachelor thesis, but its explanation is necessary to understand the faced problem. That is, the environment Game-Object was already available in CoAutoSim3D when this thesis was started.

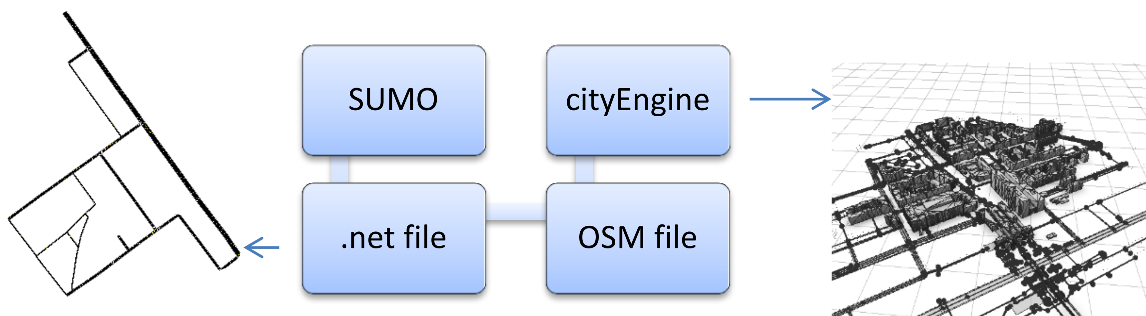

Figure 18: schematic representation of the inputs required and the outputs generated by each program

At first the taken buildings did not fit into the SUMO roads, because they were based on different .xml files. That is, the OSM area taken from making the cityEngine model of the buildings was not the same as the one taken for creating the .net SUMO file. Therefore, another SUMO .net file had to be created, which matched the one that the buildings were based on, as shown in the figure 19.

Figure 19: buildings imported in Unity 3D and instantitated within the environment

4.3.2 Classication of roads retrieved from SUMO: instantiating sidewalks and pedestrian crossings

The method consists on iterating the lanes retrieved from SUMO and analyzing the permissions on each lane. Depending on the aforementioned permissions, a first classification is made, which separates the lanes in two groups: walking lanes and car lanes. Nevertheless, in order to know if the current lane is a sidewalk or a pedestrian crossing it is not enough to analyze the permissions of the lanes, since both allow pedestrians.
A closer analysis of the .net file retrieved from SUMO suggests two ways for classifying sidewalks and pedestrian crossings:

- Take all the edges in the simulation one by one and see if their function is “crossing”.
- Take the id of each lane and see if it contains “_c”.

The first way has some drawbacks. Firstly, it has not been found any method in TraCI that retrieves the function of the edges. The second drawback is related to the current way that the simulator uses to read the data retrieved from SUMO. Unity does not read all the edges from SUMO, but it does read all the lanes. Hence, if the first method had to be implemented, all the edges retrieved from SUMO would have to be read apart from the lanes. This could increase the computational complexity considerably (by a factor of 2 approximately) and could cause an additional delay that can be avoided since the information of pedestrian crossings is implicitly present in the lanes (second way).

Thus, the second approach can be used for solving those inconveniences, which consists of analyzing the id of every lane retrieved by SUMO. The id of the edges in SUMO has a character referring to their function. If the function of a specific edge is crossing, then that edge will have a _c in its id. As far as the lane id is concerned, it is the same as the edge id followed by some additional characters that depend on the index of that lane. Therefore, the information of the function of and edge is implicitly given in all the lane-ids contained on that edge.

Example of a crossing edge

```
<edge id=":1682055180_c0" function="crossing" crossingEdges="155850180 -155850180">
        <lane id=":1682055180_c0_0" index="0" allow="pedestrian" speed="1.00" length="6.50" width="4.00" shape="1566.40,1666.93 1571.56,1670.88"/>
 </edge>
```

At this point in which sidewalks and pedestrian crossings are classified, a different prefab can be instantiated for each the three cases as shown in figure 20 and represented in the code.

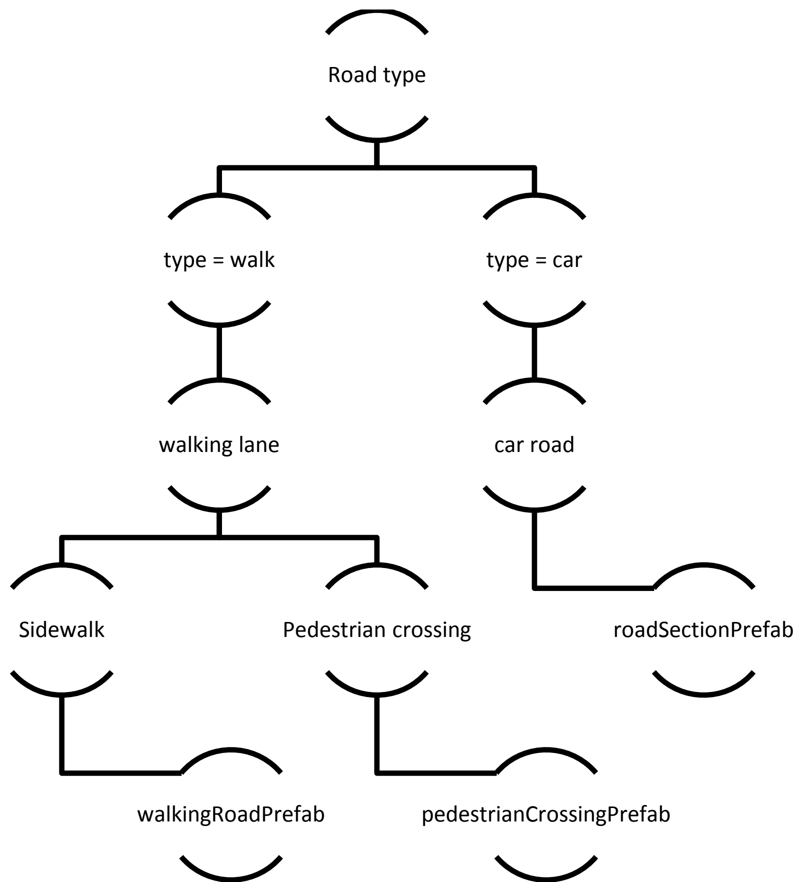

Figure 20: schematic representation of the approach used to classify roads

The way of instantiating consecutive walking lanes is the same as the way in which authors in [10] instantiated car roads and will not be explained in this thesis.
Nevertheless, when instantiating pedestrian crossings the following points have been taken into account:

- The pedestrian crossing should be as long as it is the car lane containing that pedestrian crossing.
- The orientation in which the pedestrian crossing must be instantiated is the same as the angle that the car lane containing that pedestrian crossing has respect to the coordinate system.

Classifying roads SUMOmanager.cs

```
if (linkedLane.getType().Equals("walk")){

   Vector3   roadSectionCenterPosition   =   new   Vector3((currentVertexX   +
previousVertexX) / 2, 0.05f,  (currentVertexZ + previousVertexZ) / 2);
// instantiate new road section as game object, with  walking road section
prefab, location, and orientation

GameObject      walkRoadSection      =      Instantiate(walkingRoadPrefab,
roadSectionCenterPosition, Quaternion.identity) as GameObject;

walkRoadSection.transform.rotation                                        =
Quaternion.AngleAxis((float)roadSectionOreintationAngle, Vector3.up);
// scale the new road section
walkRoadSection.transform.localScale = new Vector3((float)(0.33 * laneWidth /
3.2), 1, (float)(0.105 * roadSectionLength));
// set new road section name to lane ID
walkRoadSection.name = laneID;
// set the parent of the new road section to lanes game object
walkRoadSection.transform.SetParent(lanesParent.transform, false);
//See if the current lane is a pedestrian crossing

if(linkedLane.getID().Contains("_c")){
      //setting the position where the pedestrian crossing will be instantiated
      Vector3 roadzebraPosition = new Vector3((currentVertexX + previousVertexX)
/ 2, 0.065f, (currentVertexZ + previousVertexZ) / 2);
      //instantiating the pedestrian crossing as a game object in Unity
      GameObject      zebra      =      Instantiate(pedestrianCrossingPrefab,
roadzebraPosition, Quaternion.identity) as GameObject;
      //Adding a rotation which is equal to the angle of that road
      zebra.transform.rotation                                            =
Quaternion.AngleAxis(roadSectionOreintationAngle, Vector3.up);
      //Setting the width of the pedestrian crossing
      zebra.transform.localScale  =  new  Vector3((float)(0.33  *  (laneWidth)  /
3.2), 1, (float)(0.105 * roadSectionLength));
            }
              }
```

4.3.3Instantiating and moving pedestrians retrieved from SUMO

The sumo-retrieved-objects are instantiated in Unity 3D as Game-Objects.

Firstly, in order to have each SUMO pedestrian well classified in Unity, a Class Pedestrian (see “Appendix C”) is created, which has the following attributes: position, id, speed and angle. The Class Pedestrian created also contains some getters that allow to get both X and Z coordinates of the position of each pedestrian (*getX()* and *getZ()*), the angle (*getAngle()*), the speed (*getSpeed()*) and the ID (*getId()*).

When the connection between both programs has been established and all the lanes retrieved from SUMO have been read, the function *ReadPedestrians()* is called (every 5 frames in Unity in order to match SUMO timestep), which is in charge of creating a list of pedestrians that will be further printed and moved in the environment (in *PrintPedestrians()*). Thus, every time that there is a change in SUMO, the list is updated and filled with the pedestrians existing in that time-step. To be precise, the position, the ID, the speed and the angle for every pedestrian is retrieved from SUMO using TraCi commands contained in the so-called “person value retrieval”.

Then, a Pedestrian object is created in Unity 3D for each simulated pedestrian in SUMO and its attributes position, id, speed and angle are set to the SUMO 2D position, the SUMO id, the SUMO speed and the SUMO 2D angle (in degrees, with respect to the Y axes) of every simulated pedestrian respectively. The created pedestrian object is added to the list of pedestrians in the last position. It should be noted that until now, anything has been displayed in Unity 3D.

The information about the simulated pedestrians is in Unity, but it cannot be seen in the game view or in the scene view. Finally, all the pedestrians from the list are printed in Unity and moved in *PrintePedestrians()* as it will be explained in detail.

Function ReadPedestrians()	SUMOmanager.cs

```
//------------------------READING THE RETRIEVED PEDESTRIANS FROM SUMO-----------------
-------------------------
public void ReadPedestrians()
    {
      //Start with an empty list every SUMO time-step
      pedlist.Clear();
      //Read the ID-s of the pedestrian in the current time-step
```

```
      pedIDs = (SumoStringList)sumoTraciConnection.do_job_get(
Person.getIDList());

      //For every pedestrian in the list
          foreach (string perid in pedIDs)
        {
      //---------------------------Retrieving information from SUMO--------
---------

      //SUMO 2D position of this pedestrian
            positionperson                                              =
(SumoPosition2D)sumoTraciConnection.do_job_get(Person.getPosition(perid));
      //SUMO speed of this pedestrian
            speedperson=
((java.lang.Double)sumoTraciConnection.do_job_get(
Person.getSpeed(perid))).doubleValue();
      //SUMO angle of this pedestrian
            angleperson                                                 =
((java.lang.Double)sumoTraciConnection.do_job_get(
Person.getAngle(perid))).doubleValue();
      //Create the pedestrian object with the SUMO information of this
pedestrian
            pers   =  new  Pedestrian(positionperson,  perid,  speedperson,
angleperson);
         //-----------------------------------------------------------------
-------------------------------------

//add this pedestrian object to the list
            pedlist.Add(pers);
        }
//Create the pedestrian game-object in Unity 3D
// if it has not been previously created or update
// the game-object if it has been previously created
        PrintPedestrians();
//Perform another time-step in SUMO
        sumoTraciConnection.do_timestep();

      }
//-----------------------------------------------------------------------------------------------------------------
```

In order to display the pedestrians in Unity 3D and move them to the positions that they have in SUMO, 2 approaches have been considered:
The first approach consists in directly acting in the position of the Game-Object of each pedestrian every SUMO time-step as authors in [9] do for moving the cars in the 3D environment. That is, every time that a pedestrian with id=i changes its position in SUMO, the transform.position of the Game-Object Per_i representing that pedestrian in the 3D environment is set to the SUMO retrieved position in the current time-step.

As far as the approach 2 is concerned, a *ThirdPersonCharacter* of Unity Assets Store was analyzed, since it was considered to be appropriate for representing pedestrians in Unity. The character consists of a person that can be moved with the keyboard. Although in this thesis the keyboard is not used for the purpose of moving pedestrians, in the scripts written by the developers an interesting public function *Move()* was found in "ThirdPersonCharacter.cs". The code used for moving the character is written in the "ThirdPersonUserControl.cs" script by calling to the aforementioned Move function:

ThirdPersonUserControl.cs

```
…
// pass all parameters to the character control script
            m_Character.Move(m_Move, crouch, m_Jump);
…
```

Where:

- m_Move is a Vector 3 obtained as local coordinates
- crouch is a Boolean variable. When crouch = true the pedestrian crouches.
- m_Jump is a Boolean variable. When m_Jump = true the pedestrian jumps

The analysis reveals that if both crouch and m_Jump are set to false this function allows to move the character to a desired position which is given in m_Move. However, in order to know how m_Move is computed a further analysis of the code must be done.

ThirdPersonUserControl.cs

```
…
// calculate move direction to pass to character
            if (m_Cam != null)
            {
                // calculate camera relative direction to move:
                m_CamForward = Vector3.Scale(m_Cam.forward, new Vector3(1,
0, 1)).normalized;
                m_Move = v*m_CamForward + h*m_Cam.right;
            }
```

```
            else
            {
                // we use world-relative directions in the case of no main camera
                m_Move = v*Vector3.forward + h*Vector3.right;
            }
...
```

Thus, the developers calculate the Vector3 which will be given to the Move() function in 2 different cases:

- When the character has the main camera.
- When the character has not the main camera.

In the case of this bachelor thesis the main camera is in the user-controller car. Therefore we are interested in the case where the Vector3 is computed as:

```
m_Move = v*Vector3.forward + h*Vector3.right
```

That is, the m_Move vector3 is computed as the movement based on the pressed buttons (up, down, left, right). In fact if we try to expand the formula in more details:

- v: is the vertical Input.GetAxis (up or down arrows), which returns a value ranges in [-1,1]
- h: is the horizontal Input.GetAxis (left/right), which returns a value ranges in [-1,1]
- Vector3.forward: is the Shorthand for writing Vector3(0, 0, 1)
- Vector3.right: is the Shorthand for writing Vector3(1, 0, 0)

In conclusion, since the next position in which the character will be moved is given to the move function as **local coordinates**, the mentioned function together with the character can be used for representing a SUMO pedestrian in Unity 3D, provided that the retrieved positions from SUMO are first converted from global coordinates to local.

Therefore, the second approach have been, because the first approach does not handle the movement of pedestrians as real humans. It is only in charge of **teleporting** the character from a certain position to the next SUMO-retrieved position in the 3D environment. However, approach 2 **moves** the pedestrian in a very realistic manner, as well as its legs instead of only teleporting the Game-Object.

The function *PrintPedestrians()* which uses the *Move()* function is schematically explained in figure 21.

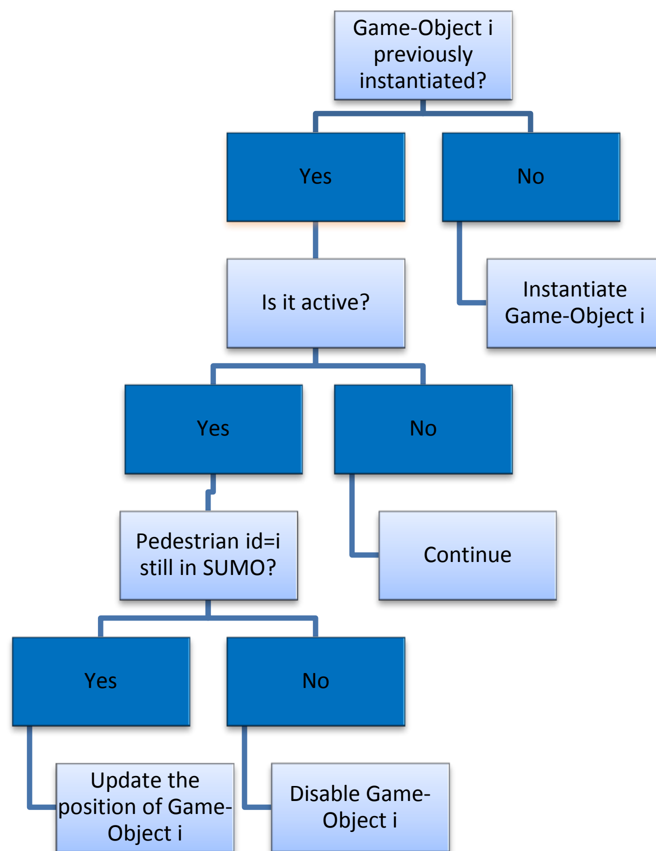

Figure 21: schematic representation of the algorithm for printing and moving pedestrians

Firstly, the updated list of pedestrians for this time-step is iterated. For each SUMO pedestrian with id=i a string is created: Per_i. Then, we check if any Game-Object with that name already exists in the Unity 3D scenario:

- If there is not any Game-Object with the name Per_i, it means that it is the first time-step where the pedestrian with id=i appears in SUMO simulation and therefore a new Game-Object representing that pedestrian must be instantiated in the 3D scenario. In order to store the position where the pedestrian will be instantiated, a Unity Vector3 is created. The X coordinate of the vector will have the X coordinate that the pedestrians has in SUMO and the Z coordinate of the vector3 will have the Y coordinate that the pedestrian has in SUMO. This is because of Unity coordinate system that can be seen in figure 22. Every time a new Game-Object is instantiated, the created object is added to a list containing the person objects in the position of their SUMO-id.

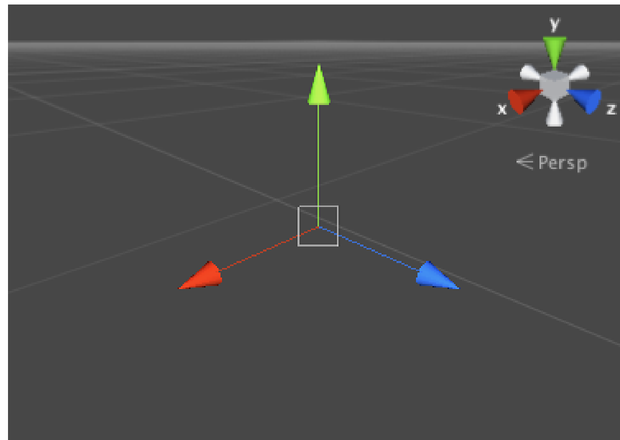

Figure 22: Coordinate system of Unity 3D

- If a Game-Object with the name Per_i already exists, it means that the pedestrian with id=i in SUMO already existed and its position may have been updated (unless the pedestrian is waiting in front of a pedestrian crossing). In order to move the Game-Object to the new SUMO position firstly the *ThirdPersonCharacter* component attached to that Game-Object must be found, since the *Move()* function is contained there. Then, the new local position that will be given to the function must be obtained, by subtracting the current position of the Game-Object to the global position retrieved from SUMO.

Until now, the pedestrians in Unity appear when pedestrians in SUMO appear and they move when pedestrians in SUMO move. However, when a pedestrian in SUMO leaves the simulation because its route has been finished, it does not leave the simulation in the Unity environment.

For that purpose, a list, *indexes*, was created. At the beginning of the time-step the list is empty. When all the pedestrians in SUMO are read, the positions of the list that are equal to the existing pedestrian ID-s are filled with a 1. That is, if in a time-step there are 2 pedestrians with id=1 and id=5, then the first position and the fifth position of the list will have a 1. The remaining positions of the list will remain empty. When all the list of pedestrians has been iterated, the list *indexes* is iterated and a 0 is placed in the empty positions. This way we can know which pedestrian indexes are in SUMO in the current time-step.
This information is useful because the pedestrians that have finished their routes in the current time-step are to be disabled. Thus, as the last instructions of the code, the list *indexes* is iterated. If a certain position, g, has the value 0 a string is created: Per_g. Now, there are two possibilities:

- If a Game-Object with that name is active, it means that although the pedestrian has already left SUMO simulation (because the value of the position corresponding to its

index is 0), the Game-Object has not been disabled yet. Then, the Game-Object must be disabled.

- If a Game-Object with that name does not exist, it means that the Game-Object has already been disabled in previous time-steps. Then, the Game-Object should not be disabled.

Function PrintPedestrians()	SUMOmanager.cs

```
//*--Printing the pedestrians kept in pedlist---------------------------
public void PrintPedestrians(){
   //start with an empty list every frame
   indexes.Clear();
   for (int i = 0; i < pedlist.Count; i++){

            Pedestrian per = pedlist[i];
            string checknamep = "Per_" + per.getId().ToString();
            string index = per.getId().ToString();
            int indexnum =int.Parse(index);
  //if that SUMO pedestrian ID exists in this timestep then add a 1 in that
position of the list
      indexes.Add(indexnum,1);
  //Is there any Unity Game-Object with that name?
            GameObject checkp = GameObject.Find(checknamep);
            if (checkp == null){ //No-----> Instantiate the Game-Object
                 //Take the 2D position from SUMO and build a 3D Vector
           Vector3 pedestrianPos = new Vector3((float)per.getX(), 0.2f,
(float)per.getZ());
                //Instantiate the Game-Object in the 3D environment
                 GameObject newPerson =
 Instantiate(personPrefab, pedestrianPos, Quaternion.identity) as
GameObject;
                //Add it to a list of Game-Objects in the position of its id
                PersonObjects.Add(newPerson);
               //Change the name of the Game-Object
                newPerson.name = "Per_" + per.getId();
              //Apply a rotation in the Y axis which is equal to the SUMO-
retrieved angle
                newPerson.transform.rotation =
Quaternion.AngleAxis((float)per.getAngle(), Vector3.up);
              //Enable the GPS if it is selected
         simulationmanager.EnableSelectedDevices(PersonObjects[i],false);
```

```
          }else{ //Yes-------> Update it

          //Angle that pedestrian i has in SUMO
      var angle =(float)per.getAngle();

      ThirdPersonCharacter newPerson =
PersonObjects[indexnum].GetComponentInChildren<ThirdPersonCharacter>();
          //Compute the local position from the global position of SUMO
      Vector3 localpos = per.getPos()-checkp.transform.position;
          //Move the pedestrian to the new position retrieved from SUMO
      newPerson.Move(localpos,false,false);
         //Make sure that the pedestrian Game-Object has the correct
rotation
      checkp.transform.rotation = Quaternion.AngleAxis((float)per.getAngle(),
Vector3.up);
          }
        }
     //Update the list of alive pedestrians-----------------------------------
---------------------------------------
     for(int g = 0; g < indexes.Count; g++){
      int result;
      if(indexes.TryGetValue(g,out result)){
          //value found
          if(result==1){
                continue;}
      }else{
           //value not found
          indexes.Add(g,0);}
        }
     //------------------------------------------------------------------------
------------------------------------------------
     //Disable the Game-Object i when the pedestrian i leaves SUMO
simulation-----------------------------
     for(int g = 0; g < indexes.Count; g++){
          if(indexes[g]==0){
          string namep = "Per_" +g;
          GameObject perobject = GameObject.Find(namep);
          if(perobject!=null){
```

```
                if(perobject.activeSelf){
                    perobject.SetActive(false);

                        }
                    }
                }
            }
        }
    //------------------------------------------------------------------------------------------------------------------

//------------------------------------------------------------------------------------------
```

Figure 23: Driver-Centric perspective of a pedestrian crossing a pedestrian crossing

Figure 24: front view of a pedestrian walking on a sidewalk

4.3.4 Manually placed and coordinated traffic lights

In the .net file some TLS-s were defined. The routes for the cars in the .rou file were created according to those TLS-s, since randomTrips.py took the .net file as an input in order to create the necessary routes for the simulated cars. Therefore, the cars retrieved from SUMO had a behavior which depended on those TLS that until now, were not displayed in Unity 3D. That is, in Unity 3D a certain vehicle stopped at an instant t = x [s] because in SUMO it had a traffic light with a red light in front of it. Therefore, traffic lights needed to be added in the Unity 3D scenario and coordinated according to the durations defined in the .net file, for the purpose of maintaining both programs synchronized.

In order to display traffic lights in Unity, traffic light game-objects had to be manually placed and coordinated for reasons explained by authors in [10], instead of instantiating them within a script. Among the packages available in Unity Assets Store, two packages were found. Both of them included some scripts to manipulate the behavior of the traffic light game-objects. The difference between both approaches lies in the parameters that they handle in the script.

- Approach 1: the traffic light modes were defined by giving to the script some green-light-time, red-light-time and yellow-light-time.
- Approach 2: a direction-time that could be externally modified was defined and when that time is over, the traffic light switches each mode.

Approach 2 was taken, since the TLS defined from SUMO do not always have the same green/yellow or red duration times and therefore this approach was considered to be more suitable for coordinating both traffic lights from SUMO and Unity.

4.3.4.1 TLS coordination between SUMO and Unity 3D

As it has been explained, the traffic lights had a script which allowed them to switch between the 3 modes: green, yellow and red. Although the script had a fixed duration that could externally be modified, a flexible duration was desired, that could be changed within a script according to the TLS phases defined in the .net file. The following steps were taken in order to coordinate the TLS system:

1. Analyze every traffic light in SUMO using both the .net and the .cfg files and obtain a green time, t_{green}, a red time, t_{red} and a yellow time, t_{yellow} for each of the traffic lights.
2. Correct some inconsistencies made by SUMO.
3. Associate a Unity game-object to every traffic light in SUMO.
4. Modify the behavior of each traffic light within a script

1-Obtaining mode times

In order to explain the way to compute the mode times, a very simple TLS will be explained as an example. The chosen TLS is defined in the .net file as TLS number 655061 and can be seen in the figure 25, taken from the .cfg file loaded in SUMO.

```xml
<tlLogic id="655061" type="static" programID="0" offset="0">
    <phase duration="86" state="GGGGgr"/>
    <phase duration="4" state="yyyyyr"/>
    <phase duration="5" state="rrrrrG"/>
    <phase duration="5" state="rrrrrr"/>
</tlLogic>
```

Figure 25: example of xml code of the TLS created for junction 655061 as section of the.net file linked in the configuration file loaded in Unity.

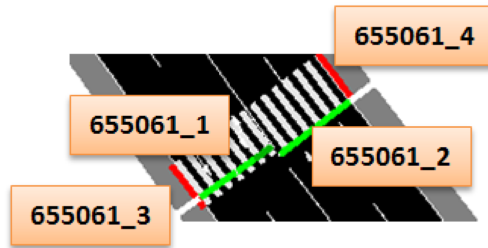

Figure 26: example of the TLS for junction 655061 displayed in SUMO-GUI

For example the left traffic light of the Traffic Light System 655061 has the following time durations:

TL 655061_1	Notation	Duration [s]
Green time	T_{green}	85
Yellow time	T_{yellow}	4
Red time	T_{red}	10
Period	T_{period}	100

Table 1: example of the mode-durations computed for the traffic light TLS 655061_1

For the purpose of computing the aforementioned times, the phase duration of the .net file was checked. The way in which the .net file defines the TLS-s is explained in [10] in detail. T_{green} = 85 s instead of 86 s because the simulation starts in t = 1s and it was observed in the .cfg file that in t = 86 s the traffic light was displaying a yellow light.

It should be noted that this TLS has a pediod of, t_{period} = 86 + 4 + 5 + 5 = 100s. That is in t=1s and in t = 101s the traffic lights of the TLS 655061 display the same lights.
2-Correcting inconsistencies made by SUMO.

In more complex junctions some inconsistencies were observed. For example the TLS controlling joined S10 had some traffic lights which had an unusual order of modes.

```
<tlLogic id="joinedS_10" type="static" programID="0" offset="0">
    <phase duration="28" state="rGGGGggrrrgGgGGrrrGrr"/>
    <phase duration="5" state="rGGGGggrrrgGgGGrrrrrr"/>
    <phase duration="3" state="ryyyyggrrryygyyrrrrrr"/>
    <phase duration="6" state="rGGrrGGrrrrrGrGrrrrrr"/>
    <phase duration="3" state="ryyrryyrrrrryryrrrrrr"/>
    <phase duration="27" state="GGGrrrrggGrrrrGrrGrGr"/>
    <phase duration="5" state="GGGrrrrggGrrrrGrrrrrr"/>
    <phase duration="3" state="yyyrrrryyyrrrryrrrrrr"/>
    <phase duration="5" state="rrrrrrrrrrrrrrrGGGGGG"/>
    <phase duration="5" state="rrrrrrrrrrrrrrrrrrrrr"/>
</tlLogic>
```

Figure 27: example of the xml code of the TLS created for junction joinedS_10 as section of the.net file linked in the configuration file loaded in Unity.

As it can be seen in the .net file for this TLS, the durations for the traffic light S10_4 can be seen in table 2.

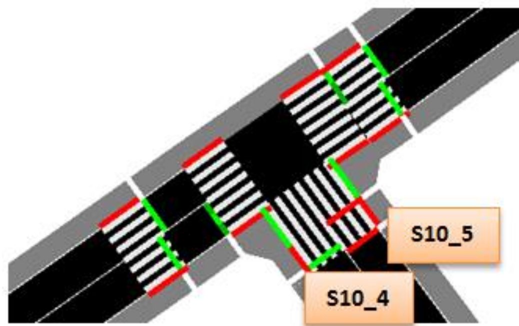

Figure 28: example of the TLS for junction Joined S10 displayed in SUMO-GUI

TL Joined S10_4	Notation	Duration [s]
First green time	T_{green1}	32
Yellow time	T_{yellow}	3
Second green time	T_{green2}	41
Second yellow time	$T_{yellow2}$	3
Red time	T_{red}	10

Table 2: example of the mode-durations computed for the traffic light Joined S10_4

Since it is not realistic for a traffic light to display a green light after having displayed a yellow one, a new duration times were computed. Now, $t_{green,new}$ = 32+3+41 = 76s.

TL Joined S10_4	Notation	Duration [s]
Green time	T_{green}	76
Yellow time	T_{yellow}	3
Red time	T_{red}	10
Period	T_{period}	90

Table 3: example of the corrected mode-durations computed for the traffic light Joined S10_4

Furthermore, it was observed that SUMO created unnecessary traffic lights. In Unity 3D only the traffic lights that were considered to be necessary for a good behavior of simulated cars were instantiated.

3-Associate a Unity game-object to every traffic light in SUMO.

A Unity game-object was created for every traffic light in SUMO. For example, for SUMO traffic lights 65506_1 and 65506_2 which correspond to the Traffic Light System 65506, the game-objects EU_TrafficLight_02(6) and EU_TrafficLight_02(15) were created in Unity 3D. All the traffic lights were put as child elements of a parent element called *trafficlights,* which was saved as a prefab for a further instantiation in a fixed position of the 3D environment.

Figure 29: example of the traffic lights and road-markings placed on an intersection in Unity 3D

4-Modify the behavior of each traffic light within a script

Firstly, the default script attached to each traffic light as a script component was disabled. Then, a new script was created based on the disabled default script. The variable duration_time of every mode (green, yellow and red) was modified according to the t_{green}, t_{yellow} and t_{red} computed in 1. This new script was attached to the traffic light game-object as a new script component. How the script looks like can be seen in [9].

In some cases, two duration times were found for a certain mode. That is, some traffic lights had for example 2 duration_times for the red light, one for the first time that they were displaying a red light in the simulation, $t_{red,1}$, and the other one for the remaining times, $t_{red,usual}$. This can happen because of the way that SUMO uses to define the traffic lights. For example, the traffic light joinedS10_5 (see table 4) has the following durations that are shown in table 4.

TL Joined S10_5	Notation	Duration [s]
First red time	$T_{red,1}$	44
Yellow time	T_{yellow}	3
Green time	T_{green}	32
Second red time	$T_{red,usual}$	55

Table 4: example of the mode-durations computed for the traffic light Joined S10_5

Thus, in the script attached to the game-object of SUMO traffic light joinedS10_5 a counter was created. The counter decided whether it was the first time that the traffic light was

displaying a red light or not. If it was the first time, then direction_time would be set to 44. Furthermore, it would be set to 55 for the remaining cases in red mode.

Section of code belonging to the Update() function	JoinedS10_02_8.cs

```
                              ...
// Mode 0: Direction 1: Green
        if (traffic_lights_mode == 0)
        {
            direction_time = 32;
            traffic_lights_counter += Time.deltaTime * 2.0f;
            if (traffic_lights_counter >= 2.0f * direction_time)
            {
                //Debug.Log("Change to mode 1");
                traffic_lights_mode = 1;
                traffic_lights_counter = 0.0f;
            }
        }
        //-------------------------------------------------------------------
---------------------------------------------
        // Mode 1: Direction 1: Yellow
        if (traffic_lights_mode == 1)
        {
            direction_time = 3;
            traffic_lights_counter += Time.deltaTime * 2.0f;
            if (traffic_lights_counter >= 2.0f * direction_time)
            {
                //Debug.Log("Change to mode 2");
                traffic_lights_mode = 2;
                traffic_lights_counter = 0.0f;
            }
        }
        //-------------------------------------------------------------------
---------------------------------------------
        // Mode 2: Direction 1: Red
        if (traffic_lights_mode == 2)
        {
            if(count==0){//first time  that is red
                              direction_time = 44;
                  }else{//remaining cases
                              direction_time = 55;
```

```
                }
            traffic_lights_counter += Time.deltaTime * 2.0f;
            if (traffic_lights_counter >= 2.0f * direction_time)
            {
                count++;
                traffic_lights_mode = 0;
                traffic_lights_counter = 0.0f;
            }
        }
                                    ...
```

4.3.5 Road markings placed manually

3D CoAutoSim reads a sumo configuration file which contains the information of real-life roads (in a .net file) and instantiates them in Unity 3D as game-objects [10]. The instantiated game-object is a gray road section (a plane in Unity 3D).
However in this bachelor thesis, road markings need to be added on the roads so that the scenario looks realistic and the drivers can easily know which way do they have to follow and in which directions are allowed to drive. To be precise, arrows, lane-separating markings and markings for traffic lights are needed. In order to add these markings, three approaches have been considered:

- Approach 1: create a new prefab with markings on both sides. As the whole roads are created by instantiating this prefab in the center of each road vertice retrieved from SUMO, a change in the appearance of this prefab will change the appearance of the whole roads. This approach could be used to separate the two lanes of an edge in an easy manner.
- Approach 2: create roads in CityEngine (with the proper markings) using the same .osm file that has been linked in the .cfg file (loaded in Unity) as an input. These roads can be instantiated in the environment by means of a script when the simulation starts. The mentioned roads are not going to be created in this bachelor thesis.
- Approach 3: create a prefab for every type of marking that is to be instantiated and instantiate it through a script.

It should be noted that the roads cannot be edited piece by piece in the simulation environment just editing the scene in Unity since the roads are not available as a game-objects in Unity before the simulation has started and the lanes have been read from SUMO.

The three approaches have some problems and some advantages. The first approach is suitable for separating two lanes but it cannot be used for instantiating arrows or traffic light lines, since these markings must be instantiated only in some parts of the environment. The second approach allows to have a very realistic representation, since it allows to customize every part of the roads, but it has a serious drawback. The problem is that SUMO roads and the game-object which has been created using cityEngine do not exactly match each other although they are based on the same .osm file. Therefore, at some points Unity 3D is displaying pedestrian lanes in the positions that SUMO has car lanes. Finally, the third approach would also allow to customize every part of the road. However it has some disadvantages. The first ones is that the position where each of the marking-game-object must be instantiated is, at first, unknown. The second one is that once the position for each game-object is determined, all markings will be instantiated using a script regardless of the information of the configuration file introduced.

After analyzing the advantages and drawbacks of each approach, an intermediate solution could be taken in order to overcome the problems of some of the approaches and benefit from their advantages. The solution consists on using approach 1 to separate the different lanes of an edge and using approach 3 to instantiate the arrows and traffic-light lines through a script. This solution does not look as realistic as the one in approach 2, but it could be a trade-off between behavior and appearance.

4.3.6 Bidirectional communication system between cars and pedestrians using P2V and V2P.

P2V and V2P technologies are to be tested in the simulated scenario for the purpose of checking if the use of these technologies would enhance VRU safety in real life. Thus, both P2V and V2P systems have been implemented in the 3D simulation.

This subsection is in charge of explaining the approach taken to implement a V2P and P2V system with the following characteristics:

- A V2P system that warns every pedestrian when a vehicle is nearby (40m or less) relying on the work in [17] with the purpose of increasing the visual situational awareness of VRU regarding the nearby location of both autonomous (SUMO-simulated cars) and manually-controlled vehicles (user-controlled car) [15].

- A P2V system that warns the driver when a pedestrian is nearby. To be precise, the system is in charge of estimating a collision point [15] between the user-controlled car and a pedestrian that is nearby. The system displays a green rectangle and the

estimated time to reach the collision point in the HUD, if it is visible for the driver. If not, the system displays an arrow pointing to the position where the pedestrian is. The system is activated if ad only if the distance to the nearby pedestrian is less than 40m and the time predicted to reach the collision point is less than 5s. An acoustic alarm has also been included which sounds once when the mentioned condition is fulfilled.

3.3.6.1 Vehicle-to-Pedestrian (V2P) communication in the 3D environment

Firstly, a distance of 40m is defined as the threshold distance for detection. The list of all the simulated pedestrians is then iterated and an array of renderers is created for every person-object. The array stores all the renderers that each person-object has attached as a component in its children elements. A renderer is what makes an object appear on the screen. In this case, renderers will be used to access and modify the materials of the person-objects instead of for changing the visibility. Then, the difference between the position of each pedestrian and the position of the user-controlled car is obtained. After that, the list of all the simulated cars is iterated. For every pedestrian, say pedestrian *i*, a subtraction is calculated, between the position of that pedestrian and the position of every simulated car. At this point, if the distance of pedestrian *i* and any simulated car or the distance of the pedestrian *i* and the user-controlled car is less than 40m, the colour of the pedestrian object *i*, which represents the pedestrian with id=i in SUMO, is changed in the 3D environment from the initial colour (white) to red. If the distance is more than 40m the colour of the *ith* pedestrian object must be changed to white, just in case it had been previously changed to red. That would be the case of a car which is approaching a pedestrian (and therefore they distance is less than 40m), but after some time it is far enough to not fulfil the condition. Note that the break statement is necessary because a pedestrian *i* can satisfy both conditions with different cars. That is, a pedestrian *i* may be close to a car *j* but far from a car *j+1*. Without the break statement the colour of the pedestrian object *i* would be changed to read and then to white within the same frame. The effect would have been the same as if the colour had never been changed. Thus, the break statement implies that when a pedestrian *i* is close to a car *j*, the colour of the object representing that pedestrian will be changed to red regardless of the distance of that pedestrian and the other cars. The reason for changing the colour of the pedestrians is just a way of proving that the V2P communication has been achieved in the simulated 3D environment.

Note that when the term difference is used, it refers to a difference between 3D points. Thus, in order to compute the distance between both 3D points, the module of the vector obtained as a difference must be calculated. The property *sqrMagnitude* of the class *Vector3* does not obtain the module of the vector, but the square of the module:

$$(x,y,z).sqrMagnitude = x^2 + y^2 + z^2$$

Hence in the written function squared magnitudes are compared against the squares of distances since the comparison will give the same result and it is much faster as it allows to avoid the slow square root operator [30].

Function DetectCars() SUMOmanager.cs

```
public void DetectCars(){
            Player = GameObject.Find("Player");
            float detectiondistance = 40f;
            Color RedColor = Color.red;
            Color WhiteColor = Color.white;
            Color currentColor=Color.black;

            if(Player!=null){
                  Vector3 carpos = Player.transform.position;
                  for (int i = 0; i < pedlist.Count; i++){
                        Pedestrian p = pedlist[i];
                        Renderer[] renderers = 
PersonObjects[i].GetComponentsInChildren<Renderer>();
                        Vector3 diffuser2ped = p.getPos()-carpos;
                        float norm = diffuser2ped.sqrMagnitude;
                        written = false;
                        for(int j = 0; j < vehicleList.Count; j++){
                              Auto vehic = vehicleList[j];
                              Vector3 diff = vehic.getPos()-p.getPos();
                              float diffLen = diff.sqrMagnitude;

      if((diffLen<detectiondistance*detectiondistance)||(norm<detectiondist
ance*detectiondistance)){

                                          foreach (Renderer r in renderers)
{
                                                foreach (Material m in 
r.materials) {
                                                      if 
(m.HasProperty("_Color"))
                                                            m.color = 
```

```
RedColor;
                                                }
                                          }
                                          break;

                              }else{
                                    foreach (Renderer r in renderers) {
                                                foreach (Material m in
r.materials) {
                                                      if
(m.HasProperty("_Color"))
                                                            m.color =
WhiteColor;
                                                }
                                          }
                              }

                        }

                  }

            }
      }
```

3.3.6.2 Pedestrian-to-Vehicle (P2V) communication in the 3D environment

First and foremost, a detection distance of 40m is defined. Then, the pedestrian list is iterated. Since the function *OnGUI()* is called every frame, the variable *pedlist* contains all the pedestrians that exist currently in SUMO and therefore all the pedestrians that are instantiated as Game-Objects in Unity as explained in section 4.3.3. For every pedestrian the difference between its position and the position of the user-controlled car is obtained. This difference is obtained by subtracting both 3D positions and computing the norm of the three dimensional vector obtained as a result of the mentioned subtraction. If the distance is less than 40 m, two parameters are calculated:

- A prediction of a collision point is obtained for every pedestrian (who satisfy the distance condition) and the user-controlled car. The formula used for obtaining this estimation is explained in [15].

Function computeCollisionPoint(float, float, float, float ,float ,float)	SUMOmanager.cs

```
public Vector3 computeCollisionPoint(float x1, float x2, float z1, float
       z2,float t1,float t2){
               //Obtain the angles in radians
       double t1rad = t1*(System.Math.PI / 180);
       double t2rad = t2*(System.Math.PI / 180);
       //Numerator of the given expression for calculating the X coordinate
       float numXc = ((z2-z1)-(x2*Mathf.Tan((float)t2rad)-
       x1*Mathf.Tan((float)t1rad)));
       //Denominator of the given expression for calculating the X coordinate
       float denomXc = (Mathf.Tan((float)t1rad)-Mathf.Tan((float)t2rad));
       //Numerator of the given expression for calculating the Z coordinate
       float numZc = ((x2-x1)-(z2*(1/Mathf.Tan((float)t2rad))-
       z1*(1/Mathf.Tan((float)t1rad))));
       //Denominator of the given expression for calculating the Z coordinate
         float denomZc = ((1/Mathf.Tan((float)t1rad))-
(1/Mathf.Tan((float)t2rad)));
       //Obtaining the collision point as a Vector 3
       float Xc = numXc/denomXc;
       float Zc = numZc/denomZc;
       Vector3 result = new Vector3(Xc,0.5f,Zc);
       return result;
      }
```

- The remaining time until the user-controlled car will reach the collision point, obtained as t= s/v, where
  - s[m] is the difference between the position of the user-controlled car and the position of the pedestrian who has satisfied the condition (and therefore is near the user-controlled car).
  - v[m/s] is the module of the speed vector of the user-controlled car, which has x and z components.

  Thus, if the pedestrian with id=i who is near the user-controlled car has a position $P(p_1,p_2,p_3)$, the user-controlled car has a position $Q(q_1,q_2,q_3)$ and a speed vector **v** = (x_1,y_2,z_1)

$$s[m] = \sqrt{(p_1 - q_1)^2 + (p_3 - q_3)^2} \quad \textbf{(1)}$$

$$v[m/s] = \|v\| = \sqrt{{x_1}^2 + {z_1}^2} \quad \textbf{(2)}$$

Note that the ground of the environment is in the XZ plane and that the speed vector of the user-controlled car does not have Y component, since it is placed in a fixed Y position of the environment.

If the obtained time is less than 5 seconds three conditions are checked.

(a) If the pedestrian is in the field of view of the driver and in front of it a green square is displayed on the screen position of the pedestrian, so that the driver is aware that a collision may happen.
(b) If the pedestrian is on the left of the driver, an arrow pointing to left is displayed.
(c) If the pedestrian is on the right of the driver, an arrow pointing to the right is displayed.

Figure 30: driver-centric perspective when when condition (a) is satisfied and the P2V system is activated

Figure 31: driver-centric perspective when when condition (b) is satisfied and the P2V system is activated

Note also that when the user-controlled car is stopped, the remaining time until it reaches the collision point is very high (ideally infinite), since v[m/s] = $||v|| \approx 0$ m/s. Nevertheless, this case is considered to be dangerous for the pedestrian because it may be walking through a pedestrian-crossing, out of the field of view of the driver, but approaching the user-controlled car. Until this point, since the remaining time is not less than 5 seconds anything is displayed in the simulated head-up display. In order to fix that, the condition v[m/s]<*0.000005f* is checked. If it is true, then it means that the user-controlled car is stopped (or almost) and in that case the variable for storing the remaining time to reach the collision point is forced to be 3 seconds. This is just a way to make sure that the imposed condition for displaying the warnings in the head-up display will then be fulfilled. However in this case the remaining time will not be displayed in the screen and just a warning arrow pointing to the pedestrian will be displayed.

Figure 32: driver-centric perspective of a dangerous situation when the detection system is not active

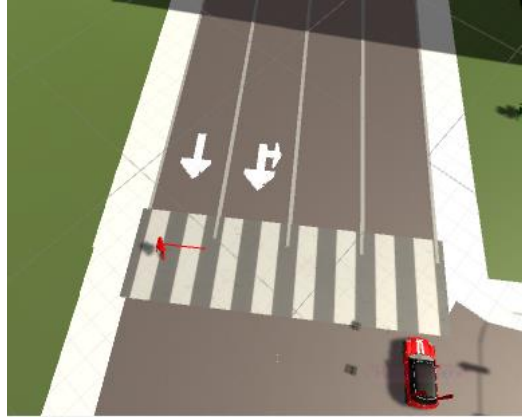

Figure 33: top-view of a dangerous situation

Figure 34: driver-centric perspective of a dangerous situation when the detection system is active

In order to decide whether the pedestrian is on the right or on the left of the driver, the screen position of the pedestrian is checked. If the screen position (measured in pixels) is less than half the width of the screen, the pedestrian is considered to be on the left of the driver. Note that the piece of code which is in charge of that is introduced on the *OnGUI()* function, which is in principle executed every frame. However, this instruction should be executed just once and when to execute it was crucial, since when a pedestrian is very far away from the driver and behind it, it can be on the left of the driver, but its screen position, measured in pixels, will be higher than half the width of the screen. Thus, whether the pedestrian was on the right or on the left was decided when the time remaining until the user-controlled car will reach

the collision point is less than 5s. Furthermore, a variable (*timesped*) was used in order to make sure that the instruction is only executed once for each pedestrian.

Function OnGUI()	SUMOmanager.cs

```
void OnGUI(){
 Player = GameObject.Find("Player");
 Camera cam =
GameObject.Find("RCCCamera").GetComponentInChildren<Camera>();
  Rigidbody rb = Player.GetComponent<Rigidbody>();
  float detectiondistance = 40f;
  if(Player!=null){
      Vector3 carpos = Player.transform.position;
      for (int i = 0; i < pedlist.Count; i++){
                 Pedestrian person = pedlist[i];
            double speedPed = person.getSpeed();
            Vector3 objpos = PersonObjects[i].transform.position;
            //Obtain the screen position of the pedestrian object
            Vector3 screenPos = cam.WorldToScreenPoint(objpos);
            //Obtain the screen position of the user-controlled car
            Vector3 usercarscreen =
cam.WorldToScreenPoint(Player.transform.position);
            float carangle = Player.transform.rotation.x;
            float perangle = (float)person.getAngle();

            Vector3 offsetd = carpos-objpos;
            float sqrLend = offsetd.sqrMagnitude;
            //Are car and the pedestrian close enough?
            if((sqrLend<detectiondistance*detectiondistance)){
                                  //Estimation of the collision point
                                   Vector3 col =
computeCollisionPoint(objpos.x, carpos.x, objpos.z,
carpos.z,perangle,carangle);
                                 Vector3 offset = carpos-col;
                                  //Distance to collision point (squared)
                                 float sqrLen = offset.sqrMagnitude;
                  //Obtain the speed of the user-controlled car as a vector
                  Vector3 speedVectorCar = rb.velocity;
```

```
                    //Obtain the module of the vector
                    float speedCar = Mathf.Sqrt(speedVectorCar.x *
speedVectorCar.x + speedVectorCar.z* speedVectorCar.z);
                    float time2ArrivePosition;
                    string stringToEdit;
                    //Case 1: The user-controlled car is stopped in front of
a traffic light
                    if(speedCar<0.000005f){
                          //set the time to arrive to the position of the
pedestrian to 3   seconds
                          time2ArrivePosition=3;
                          stringToEdit = "";
                    //Case 2: user-controlled car is not stopped
                    }else{
                          //Obtain the time to arrive to the position of the
pedestrian as t=s/v
                          time2ArrivePosition =
Mathf.Sqrt(offset.x*offset.x+offset.z*offset.z)/speedCar;
                          stringToEdit =
(Mathf.Round(time2ArrivePosition)).toString();
                                }


//If the time is less than 5seconds see if the pedestrian that is close to
the car is on the left or
//on the right of the car----------------------------------------------------
--------------------------
                                if(time2ArrivePosition<5){
                                      timesped[i]++;
                                      if(timesped[i]==1){
                                            if(screenPos.x<(Screen.width/2)){
                                                  relative2car[i]="left";


                                            }else{

      if(screenPos.x>(Screen.width/2)){

      relative2car[i]="right";
                                                  }
```

```
                                        }
                                  }
//---------------------------------------------------------------------------
----------------------------------------------


//---------------------DISPLAYING A RECTANGLE OR AN ARROW IN THE SCREEN
DEPENDING ON THE PEDESTRIAN POSITION--------------------------------------
--------------
//Case 1: pedestrian visible for the driver: display a green rectangle and
the time remaining to reach the position
                                                              if
((PersonObjects[i].GetComponentInChildren<Renderer>().isVisible)&&(PersonO
bjects[i].activeSelf)&&(screenPos.x>150)&&(screenPos.x<900)&&(screenPos.y>
30)&&(screenPos.y<432)&&(screenPos.z>usercarscreen.z)){
                                   DrawScreenRectBorder( new Rect(
screenPos.x-8, screenPos.y, Screen.width/13, Screen.height/7), 2,
Color.green );
                                   stringToEdit = GUI.TextField(new
Rect(screenPos.x-8, screenPos.y, Screen.width/13, Screen.height/7),
stringToEdit, 30);
                                   GUI.skin.textField.fontSize = 40;


//Case 2: pedestrian not visible
                                       }else{
      //Case 2.1 Pedestrian on the left: display an arrow pointing to the
left
                                   if(relative2car[i].Equals("left")){
                                            float aspectRatio = 99/95;
                                           GUI.DrawTexture(new Rect(300,
50, aspectRatio*(Screen.width/14), Screen.width/14), flechapeaton,
ScaleMode.StretchToFill, true, 10.0F);
                                      if(!stringToEdit.Equals("")){
                                         stringToEdit = GUI.TextField(new
Rect(300, 90, 40, 40), stringToEdit, 40);

GUI.skin.textField.fontSize = 30;


                                                    }


                                        }
      //Case 2.2: Pedestrian on the right: display an arrows pointing to
```

```
the right
                              }else{

if(relative2car[i].Equals("right")&&(PersonObjects[i].GetComponentInChildr
en<Renderer>().isVisible==false)){
                              float aspectRatio = 99/95;
                              GUI.DrawTexture(new Rect(980, 90,
aspectRatio*(Screen.width/14), Screen.width/14), flechaderecha,
ScaleMode.StretchToFill, true, 10.0F);
                              if(!stringToEdit.Equals("")){
                                    stringToEdit = GUI.TextField(new
Rect(980, 110, 40, 40), stringToEdit, 40);

GUI.skin.textField.fontSize = 30;
                                                      }
                                                }
                                          }
                                    }
                              }
                        }
                }
            }else{
            }
      }
```

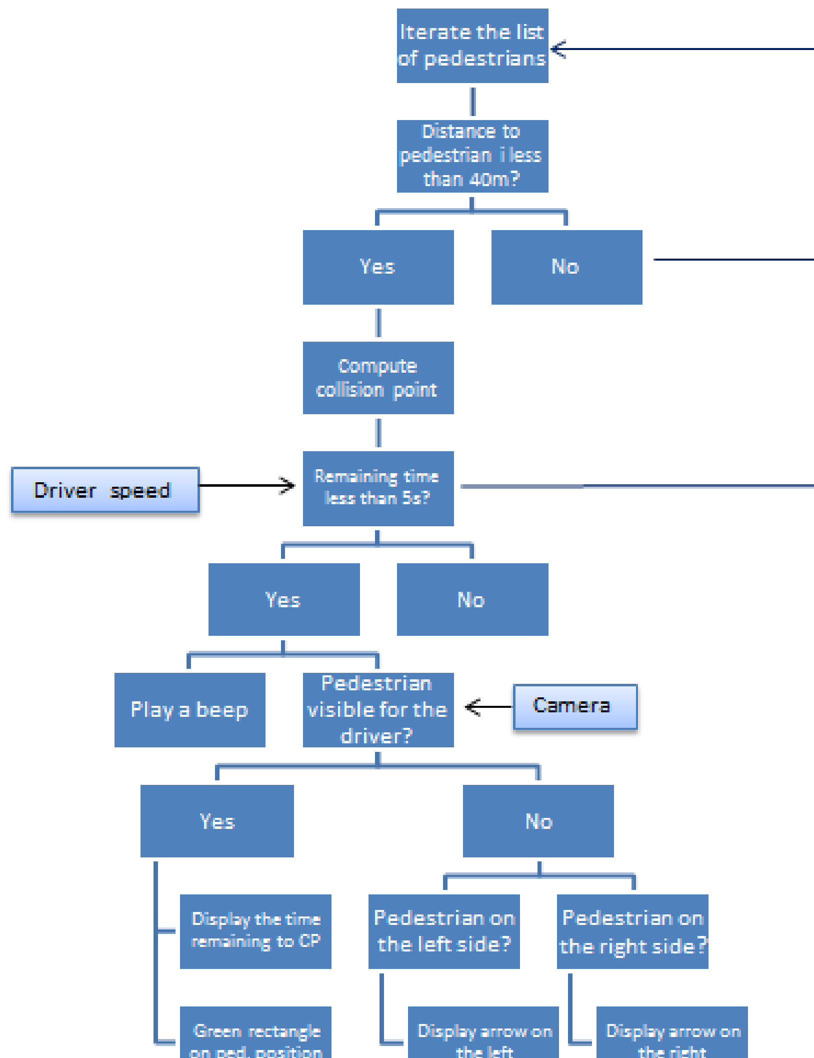

Figure 35: algorithm used for developing the P2V system

4.3.7 Logging the position of every pedestrian using GPS devices

As explained in section 3.3 if the system of data-logging of 3DCoAutoSim is to be maintained, in order to have an output saved, a device is needed, which will be linked to a Game-Object and will acquire data for it. Among the devices available, the GPS was considered to be the most useful, since it retrieves the altitude, latitude and longitude of the Game-Object it is attached to.

Since authors in [2] used the GPS for retrieving parameters regarding the user-controlled car, the logics of the function which is in charge of enabling the devices that the user has selected have been modified in this thesis. The modifications made consist of adding a new Boolean argument to the function so that when the function is executed it can be known from

the inside if the GPS is attached to a pedestrian or if it is the GPS of the user-controlled car. The function will be called after one of this two outcomes happen:

- The user-controlled car is instantiated. The function is called with the command *EnableSelectedDevices(player,true).* Thus, inside the function the boolean variable *implayer* will be true and the command will not activate the pedestrian GPS.

- The Game-Object for the pedestrian with id=i is instantiated inside *PrintPedestrians()*. The function is called with the command *simulationmanager.EnableSelectedDevices(PersonObjects[i],false).* Hence, inside the function the Boolean variable *implayer* will be false and the command will activate the GPS attached to the Game-Object with the name *Per_i* if the user has externally selected it. Note that in this case the instruction it is written in *SUMOmanager*, where the pedestrian is instantiated. Therefore, in order to implement the public method *EnableSelectedDevices* which is defined in *SimulationManager.cs*, a variable *simulationmanager* implementing the *class SimulationManager* has been created inside the script *SUMOmanager.cs.*

Function EnableSelectedDevices(GameObject player,bool implayer) SimulationManager.cs

```
// enable selected devices for the player game object
    public void EnableSelectedDevices(GameObject player,bool implayer)
    {
        // variable for the number of selected devices
        int devicesNumber = 0;
        // check if there is an devices list saved in player prefs
        if (PlayerPrefs.HasKey("DevicesList"))
        {
            // retrieve the devices list from player prefs
            string devices = PlayerPrefs.GetString("DevicesList");
            // split the devices list to an array based on the comma separator
            string[] devicesList = devices.Split(',');
            // get the total number of devices subtracting the last comma
            devicesNumber = devicesList.Length - 1;
            // loop over all selected devices list
            for (int i = 0; i < devicesNumber; i++)
            {
//-----------The function has been called with the user-controlled car Game-
Object---------------
```

```
            if(implayer==true){
                  if (devicesList[i].Equals("Lidar", StringComparison.Ordinal))
{
                        GameObject lidar =
player.transform.Find("Lidar").gameObject;
                        lidar.SetActive(true);
                        continue;
                  }

                       if (devicesList[i].Equals("Camera",
StringComparison.Ordinal)) {


                        GameObject camera = player.transform.Find("Photo
camera").gameObject;
                        camera.SetActive(true);
                        continue;
                  }

                  if (devicesList[i].Equals("GPS", StringComparison.Ordinal)) {

                        GameObject gps =
player.transform.Find("GPS").gameObject;
                        gps.SetActive(true);
                        continue;
                  }

                  if (devicesList[i].Equals("Compass",
StringComparison.Ordinal)) {
                        continue;
                  }
//--------------------------------------------------------------
/--------------------The function has been called with a pedestrian Game-Object-
------
            }else{
                   if (devicesList[i].Equals("GPSped",
StringComparison.Ordinal))                {
                  if(implayer==false){
                        GameObject gpsped =
player.transform.Find("GPSped").gameObject;
```

```
                    gpsped.SetActive(true);
                    continue;
               }
                              continue;
          }
     }

         }
      }

   }
```

4.4 Experimental setup

Data (see section 4.5.1) from N=20 test subjects was collected (80% males, 20%females, mean age = 26,4 , SD = 6,5406) to measure the driver's response to the implemented P2V system. A scenario has been developed (as explained in section.4.3) in order to conduct an experiment. The experiment involved two sets of driving tests. The order of the sets was alternated to avoid bias. The flow chart of the experimental procedure can be seen in figure 36.

4.4.1 Instructions

After being welcomed, instructions regarding the experimental procedure were given to each participant. They were asked to drive according the traffic rules and not to exceed speed limits. The exact instructions given to each participant can be seen in "Appendix A".

4.4.2 Familiarization phase and driving tests

First and foremost, each participant drove for a period of 3 minutes without any system activated, this phase being defined as the familiarization phase. The goal of this phase was that the participants get used to the steering wheel and the pedals. Next, the participants performed two sets of driving tests:

a) Test 1: the user-controlled car had the pedestrian detection system **activated** and the driver followed a specific route of the environment which was signalized by arrows.
b) Test 2: the user-controlled car had the pedestrian detection system **disabled**, but the driver followed the same route as in test 1.

The conditions that the drivers have faced in Test 1 and Test 2 are the same except from the fact that in Test 1 a pedestrian detection system was assisting the driver. The simulated pedestrians in both tests had the same routes at the same instants of the simulation.

In order to test whether the potential benefit of the system provided was affected by the performance of an additional task both sets will involve the execution of a secondary task relying on the paper [35]. The secondary task consisted on searching for a specific pen in a full bag and it took approximately 20 seconds.

4.4.3Post-task questionnaire

As the last phase of the experiment, each participant was required to complete a post-task questionnaire. The purpose of this phase was to perform a subjective evaluation of certain parameters. The participants were asked to answer the questions shown in "Appendix B" in order to know if they were satisfied with the P2V system. Summarizing, participants were asked to answer two types of tests:

1) 3 questions related to the general characteristics of the system to be evaluated with the 5-points Likert-type scale
2) 2 precise questions related to the user-satisfaction of the P2V system

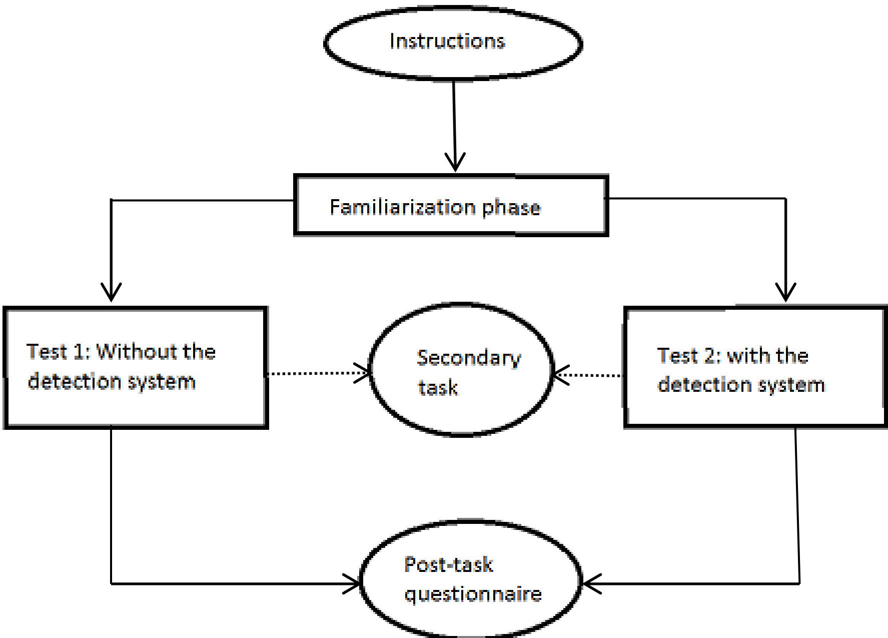

Figure 36: flow chart of the experimental procedure to measure driver response to the P2V system

4.5 Data collection and parameters of interest

4.5.1 Driver performance metrics

To study the effect of the system on driving performance, several metrics were measured. For each participant 5 files were logged, containing the information retrieved from 5 devices: the Bus-Reader, the GPS (for the car and the pedestrian), the lidar and the camera. Table 5 contains the content of each file.

File name	Measured metric	Car/Pedestrian
BusReader-0	Timestamp, Speed[m/s], Steering angle ([-1, 1]), Throttle pedal position ([0, 1]),Brake pedal position ([0, 1])	Used-controlled car (UCC)
GPS-0	Timestamp, Latitude, Longitude , Altitude	Used-controlled car (UCC)
GPS-i	Timestamp, Latitude, Longitude , Altitude	Pedestrian id=i
Lidar	timestamp,PCL	Used-controlled car (UCC)
Camera	Photos	Simulation

Table 5: content of each of the logged file

That is, for each driver, information related to the speed, position, braking and accelerating strength was measured every 0.1 s ($f_{logging}$ = 10Hz). Furthermore, a camera took photos of the simulation with the purpose of rebuilding the whole experiment and performing a further analysis of each frame. The metrics retrieved from the lidar were not used in this bachelor thesis, but were not removed, just in case they were needed in a later study.

4.5.2 Driver performance calculations and parameters

The information logged was used to compute and analyze some parameters of interest for each participant. These parameters will only be parameters of interest when the drivers enter the area of the environment which has pedestrians crossing and an accident may happen. That is, **the parameters will be analyzed when the driver enters the area of interest**. The chosen area is a straight road section. The time where the driver reaches the desired region is known, since it is the same instant of time when the driver was told to perform a secondary task.

1) Deceleration Change Rate: the deceleration change rate represents the ratio of decelerations within two successive frames[36]

$$DCR = \frac{a}{t_f} = \frac{V_{f-1} - V_{ftf}}{tf} \quad \textbf{(3)}$$

Where:
V_{f-1}:= Velocity of the vehicle during the previous frame while braking
V_f := Velocity of the vehicle during the current frame while braking
t_f :=Duration of the frame

That is, the deceleration will be calculated according to (3) when the driver starts braking in the area of interest. Since the area of interest is a straight toad section, the Deceleration Change Rate represents the amount of reduction on the speed of the driver due to the sudden appearance of a pedestrian.

Note that in this case, the duration of each frame is not available. Hence the acceleration will be computed as the difference in the speed of two timestamps of the simulation. In order to ensure the consistency of the results retrieved from (3), the acceleration will be divided by the difference of the same two timestamps used to compute the acceleration.

2) Maximum and minimum values retrieved from the steering wheel sensor. As the area of interest is a straight road section, the information conveyed by this parameter can be comparable to the information that the lateral deviation would convey due to the lack of curves in the desired region. Thus, in this case the values retrieved from the steering sensor can be used to know the distraction that the driver is suffering as authors in [37] do.

3) Number of collisions between each driver and dangerous pedestrians: this parameter is a Boolean number (1=collision) obtained after the frame-to-frame analysis of the 2N = 40 experiments.

4.5.3 Methodology for analyzing simulation data

The approach taken for analyzing the data is based on the “*five-step procedure*” explained in [34].

4.5.3.1Comparison between scenarios: steering wheel sensor and deceleration

For the comparison of the parameters of interest between scenarios X=”*Without the P2V system*” and Y=”*With the P2V system*” the **paired t-test for means** will be applied. Using α = 0.05 it is to be assessed whether the mean of the measurements steering wheel angle and deceleration are statistically different from each other. The paired t-test has been chosen because **the performance of the same driver in a certain environment is measured twice and the net change from the first condition to the next is to be computed**. Thus, first those two parameters are to be calculated for each participant, in order to get their mean values (see section 5).

Proceeding with the "*five-step procedure*" the mathematical procedure to test the hypothesis will be explained. The procedure explained calculates both the rejection region (I) and the φ-value (II). In fact , in order to know if the null hypothesis can be rejected or not it is enough just with one of the two approaches taken (I or II). However, both of them have been used for the purpose of validating the results.

Step 1: **Hypotheses:** the null and alternative hypotheses are established to test if the average Deceleration Change Rate provided that the P2V system is activated is greater than if the P2V system is disabled.

-H_0: $\mu_D = 0$
-H_1: $\mu_D > 0$

Where $\mu_D = \mu_x - \mu_y$ and $\mu_x = E[X]$, $\mu_y = E[Y]$

Step 2: **Test Statistic**: the test statistic chosen is $\bar{D}$ because $E[\bar{D}] = \mu_D$. The value of this test statistic is $\bar{d}$ = 0.1113 (see section 5). The standardized test statistic, under the assumption that H_0 is true and its distribution are $\frac{\bar{d}-\delta 0}{\frac{Sd}{\sqrt{nd}}} \sim t_{20-1}$

Step 3: **Rejection Region Calculations**:

I. Rejection region: Because the standardized test-statistic is distributed as t_{19} and H_1 is an upper one-sided hypothesis, the rejection region is $t_{obs} > t_{0.95;19}$ = 1.73 (calculated in R, see below) .The value of the standardized test-statistic is $\frac{\bar{d}-\delta 0}{\frac{Sd}{\sqrt{nd}}}$ $=1.02$ (computed in R, see below).

II. The φ-value is $P(t_{19} \geq 1{,}02)$ =0.1602.

Step 4: **Statistical Conclusion**:

I. From the rejection region, **accept H_0** because t_{obs} = 1,02 is not greater than 1.73.
II. From the φ-value, **accept H_0,** because φ-value = 0.1602 < 0.05 = α

Step 5: **English Conclusion**: see section 5

$t_{0.95;19}$
qt(.95,19) [1] 1.729133

T-test

```
>d<-c(-0.0115,0.6075,-0.6892,-0.0352,-0.8826,0.0054,
0.2310,0,0,0,0.8,0.0352,0,0, 0.0207 ,2.0534,0.0431,0.473,0,0)
>t.test(d,alternative="greater")
#Results-------------------------------------------------------------
-------------
        One Sample t-test

data:  d
t = 1.0205, df = 19, p-value = 0.1602
alternative hypothesis: true mean is greater than 0
95 percent confidence interval:
 -0.09204006         Inf
sample estimates:
mean of x
  0.13254
```

The same test was also applied to the values obtained from the steering wheel sensor. The results are shown in section 5 and the R commands are shown below. The procedure was exactly the same as the one used for the DCR and will not be repeated.

T-test

```
a<-c(0.1689,0.4063,-0.0551,0.3747,0.4905,0,-0.0100,-0.4766,-
0.309,0.9853,0.7357,0.4095,0.6593,0.4607,-0.8443,-0.6694,0.0350,-0.3999, -
0.0011, -0.0111)
t.test(a,alternative="greater")
#Results
       One Sample t-test

       data:  a
       t = 0.91003, df = 19, p-value = 0.1871
       alternative hypothesis: true mean is greater than 0
       95 percent confidence interval:
        -0.08773049         Inf
       sample estimates:
              mean of x
               0.09747
```

4.5.3.2 Comparison between scenarios: number of collisions

For the comparison of the number of collisions between scenarios **Fisher's Exact Test** was applied. Let's denote:

X = "*Number of collisions observed when the P2V system is activated*" and
Y=" *Number of collisions observed when the P2V system is disabled.*"

Since X~Bin(m,π_x) and Y~Bin(n,π_Y) this test is perfectly suitable to assess whether the number of collisions is significantly higher or not when the system is disabled.

	Success	Failure	Total
X Sample	x	$m-x$	m
Y Sample	y	$n-y$	n
	k	$N-k$	N

Table 6: general form of a 2x2 table to apply Fisher's Exact Test[34]

	Collision	**No collision**	Total
System activated	1	19	20
System disabled	5	15	20
	6	34	40

Table 7: 2x2 table obtained from section 5 to apply Fisher's Exact Test

Step 1: **Hypotheses:** the null and alternative hypotheses are established for the purpose of testing whether the proportion of collisions when the P2V system is activated is less than when it is disabled.

-H_0: $\pi_x = \pi_Y$
-H_1: $\pi_x < \pi_Y$
$\pi_x = E[X]$, $\pi_y = E[Y]$

Step 2: **Test Statistic**: the test statistic chosen is X, where X = "*Number of collisions observed when the P2V system is activated*". Provided H_0 is true, and conditioning on the fact that X+Y = k, (ie 1+5 = 6 see table 7) X~Hyper(m,n,k). The observed value of the test statistic is x_{obs}=1.

Step 3: **Rejection Region Calculations**: Rejection is based on the φ-value, so none are required.

Step 4: **Statistical Conclusion**: to calculate the φ-value, compute:

$$P(X \leq x_{obs} = 1) = \sum_{i=0}^{1} \frac{\binom{1+19}{i}\binom{5+15}{6-i}}{\binom{40}{6}}$$

This calculation has been done in 2 ways using R with the following S code:

φ-value	*Approach 1*
> pval<-phyper(1,1+19,5+15,1+5) > pval [1] 0.09088209	

Fisher exact test	*Approach 2*

```
> HA<-matrix(c(1,5,19,15),nrow=2,
+
dimnames=list(System=c("Activated","Disabled"),
+ Outcome=c("Collision","No collision")))
>fisher.test(HA,alternative="less")

#Results:
data:  HA
p-value = 0.09088
alternative hypothesis: true odds ratio is less
than 1
95 percent confidence interval:
 0.000000 1.300986
sample estimates:
odds ratio
 0.1646833
```

Because the φ-value is higher than 0.05, **accept H_0**

Step 5: **English Conclusion**: see section 5.

5 Results

In order to get the mean values to assess whether the P2V system improves significantly the driving performance, first the parameters of interest must be calculated from the logged data for every participant. This results are calculated at the instant of the simulation explained in 4.5.2 while the driver is performing a secondary task and enters the area of interest.

5.1 Objective results

	Collision	*No collision*	*Total*	**φ-value**	**α**
System activated	1	19	20	0.09088	0.05
System disabled	5	15	20		

Table 8: 2x2 table obtained to apply Fisher's Exact Test

Metric	*System activated*		*System disabled*	
	Mean	**SD**	**Mean**	**SD**
Deceleration Change Rate (DCR)	0.1991	0.4889	0.0879	0.2480
Steering wheel sensor	*0.2632*	*0.3068*	*0.3606*	*0.3397*

T-test (α=0.05,$t_{0.95;19}$ = 1,73)

Metric	*System activated vs System disabled*	
	φ-value	**t_{obs}**
Deceleration Change Rate (DCR)	0.1602	1.0205
Steering wheel sensor	*0.1871*	0.91003

Table 9: driving performance values depending on whether the P2V system is activated or not

As far as the number of accidents is concerned, table 8 shows that the number of accidents when the P2V system disabled is 5, while it is just 1 for the case of having the system activated. This gives a probability of 5% of having an accident with the proposed P2V system (P(accident | system activated)=0.05) *vs* P(accident | system disabled)=0.25. Furthermore, after analysing the speed of participant 14 (the one who had an accident while the system was activated), it was found out that he/she was not respecting speed limits and that could be one of the reasons for colliding. Although there is not enough evidence to suggest that the P2V system, can be used, in general, to reduce accidents at a level α = 0.05, the difference between the confidence level and the φ-value is not large (φ-value =0.09088). In fact, if a less restrictive confidence level of 0.9 had been chosen (α=0.1), which is typical in data analysis [34] it could have been showed that the system could be used to reduce the number of accidents still with a high level of reliability.

Nevertheless, the mean values of the Deceleration Change Rate of test1 and test2 are not significantly different. In fact, there is not enough evidence to suggest that the P2V system, can be used, in general, to increase the mean Deceleration Change Rate with 95% reliability, since φ-value =0.1602. The confidence level that would be required, 1-α = 0.8, which is not

used in data analysis, since it implies that there is a relatively high probability (20%) to reject the null hypothesis when it is true (type I error).

Finally the analysis of the values retrieved from the steering wheel sensor reveals that there is not statistical evidence to prove that the P2V system is distracting the driver while driving. In fact φ-value = *0.1871* which is significantly higher than $\alpha = 0.05$. Besides, the mean value obtained from the deviations in the case where the system is activated is even less than the one obtained when the system is disabled.

Looking at particular results for every participant, it is also remarkable that 32/40 dangerous pedestrians appeared in the N=40 trials, that is, the probability of having a potentially hazardous road situation involving pedestrians was 0.8 for each participant regardless of the system. Thus, the strategy used for acting on the routes of the pedestrians has been proved to be efficient.

It is important to note that in order to have a collision between a car and a pedestrian, a dangerous pedestrian must have appeared in the driver's way. Although the probability that a dangerous pedestrian appears has been considerably increased by acting on the pedestrian-routes file, there is still a 20% probability that the driver does not face a hazardous pedestrian. This happens because although the .rou file generated from SUMO (which is in charge of defining the routes for every pedestrian) is the same for every participant, participants might have different velocities, which leads to not facing the same pedestrians at the same instants.

5.2 Subjective results

The results of the post-task questionnaire are shown in tables 10 and 11.

Question	Average	Variance
Q1	3.8	0.8944
Q2	2.95	0.7592
Q5	3.85	0.9333

Table 10: average and variance of the results obtained from the post-task questionnaire(questions 1,2 and 5)

Question	Annoying/Yes	Useful/No	Cannot answer
Q3	10%	80%	10%
Q4	5%	95%	0%

Table 11: results obtained from the short-questions of the post-task questionnaire

Results regarding the qualitative evaluation showed that almost all participants found the pedestrian detection system very useful. Only 15% of the participants did not notice the alarm included in the pedestrian detection system. 95% of the participants claimed that the green square displayed in the head-up-display was not reducing their visibility while driving. Question 2 obtained the less punctuation with a mean of approximately 3, that is the participants do not agree nor disagree with the level of reality of the simulation. Nevertheless, some participants affirmed that the problem was not that the environment was not realistic, but the hardware used to develop the experiment.

6 Conclusions

The behaviour of the driver has been shown to be positively affected by the P2V system proposed in this bachelor thesis, since the proportion of accidents that have happened with the system activated is significantly less than when the system is not activated and there is statistical evidence with high confidence (90%) to show that the P2V system can be used, in general, to reduce the number of accidents.

Furthermore, both subjective and objective results show that the P2V system proposed is not distracting the driver while driving and that it is not reducing its visibility.

In conclusion, there is enough evidence to state that the P2V system with the features explained in this bachelor thesis could assist the driver to drive in a better manner, improving the safety of VRU by reducing accidents.

Bibliography

[1] C. Biurrun, L. Serrano, C. Olaverri (2017) "Microscopic Driver-Centric Simulator: Linking Unity3D and SUMO.", In: Rocha Á., Correia A., Adeli H., Reis L., Costanzo S. (eds) Recent Advances in Information Systems and Technologies. AISC, volume 569, pp. 851-860, Springer, Cham

[2] A. Hussein, A. Diaz and C. Olaverri, “3DCoAutoSim : Simulator for Cooperative ADAS and Automated Vehicles,” no. Iv, 2018, unpublished.

[3] D. H. Biedermann, P. M. Kielar, Q. Aumann, C. M. Osorio, and C. T. W. Lai, “CarPed -A Hybrid and Macroscopic Traffic and Pedestrian Simulator,” vol. 1985, p. 8, 2015.

[4] Z. Kokkinogenis, L. Sanchez Passos, R. Rossetti, and J. Gabriel, “Towards the next-generation traffic simulation tools: a first evaluation,” 6th Iber. Conf. Inf. Syst. Technol., pp. 15–18, 2011.

[5] Koh S.Y Doina, Chin H.C, “Traffic Simulation Modelling: VISSIM”

[6] D. Salgado, D. Jolovic, P. T. Martin, and R. M. Aldrete, “Traffic Microsimulation Models Assessment - A Case Study of International Land Port of Entry,” Procedia Comput. Sci., vol. 83, no. Ant, pp. 441–448, 2016.

[7] C. Siddiqui, M. Abdel-aty, and K. Choi, “Macroscopic spatial analysis of pedestrian and bicycle crashes,” vol. 45, p. 2012, 2012.

[8] A. Kormanová, “A Review on Macroscopic Pedestrian Flow Modelling,” Acta Inform. Pragensia, vol. 2, no. 2, pp. 39–50, 2013.

[9] J. Erdmann and D. Krajzewicz, “Modelling Pedestrian Dynamics in SUMO,” no. May, 2015.

[10] C. Olaverri, J. Errea, A. Díaz (2018) "Implementation and Evaluation of a Traffic Light Assistance System in a Simulation Framework based on V2I Communication;, Journal of Advanced Transportation, Special Issue “Cooperative Systems for Autonomous Vehicles” (CSAV), Hindawi Publishing Corporation.

[11] F. Johansson, “Microscopic Modeling and Simulation of Pedestrian Traffic,” p. 119, 2013.

[12] L. Guo, P. S. Ge, M. H. Zhang, L. H. Li, and Y. B. Zhao, “Pedestrian detection for intelligent transportation systems combining AdaBoost algorithm and support vector machine,” Expert Syst. Appl., vol. 39, no. 4, pp. 4274–4286, 2012.

[13] R. Wehbe and E. Otayek, “Using Intelligent Transportation Systems to Enhance Pedestrian Safety at Beirut Signalized Intersection,” 2017 5th IEEE International Conference pp. 574–579, 2017.

[14] L. T. Hagen, “Selecting the most effective ITS application for pedestrian safety in Florida,”.

[15] A. Hussein, F. García, J. M. Armingol, and C. Olaverri-Monreal, “P2V and V2P communication for pedestrian warning on the basis of autonomous vehicles,” IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC, pp. 2034–2039, 2016.

[16] M.P.Nkosi, G.P. Hancke, R.M.A. dos Santos: “Autonomous pedestrian detection”.

[17] A. Allamehzadeh and C. Olaverri-Monreal, “Automatic and manual driving paradigms: Cost-efficient mobile application for the assessment of driver inattentiveness and detection of road conditions,” IEEE Intell. Veh. Symp. Proc., vol. 2016–August, no. Iv, pp. 26–31, 2016.

[18] “Annual global road crash statistics,” http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics .

[19] P. Merdrignac, O. Shagdar, and F. Nashashibi, “Fusion of Perception and V2P Communication Systems for the Safety of Vulnerable Road Users,” IEEE Trans. Intell. Transp. Syst., vol. 18, no. 7, pp. 1740–1751, 2017.

[20] A. Rostami, B. Cheng, H. Lu, M. Gruteser, and J. B. Kenney, “Reducing Unnecessary Pedestrian-to-Vehicle Transmissions Using a Contextual Policy,” Proc. 2nd ACM Int. Work. Smart, Auton. Connect. Veh. Syst. Serv. - CarSys ’17, pp. 3–10, 2017.

[21]https://ec.europa.eu/transport/themes/its/road/action_plan/its_and_vulnerable_road_users_en

[22] http://www.bmva.org/apps:pedestrian

[23] P. Viola and M. J. Jones. Robust real-time face detection. IJCV, 57(2):137–154, 2004.

[24] J. Gall, A. Yao, N. Razavi, L. V. Gool, and V. Lempitsky. Hough forests for object detection, tracking, and action recognition. IEEE Trans. PAMI, 33(11):2188–2202, 2011. 2

[25] S. Zhang, C. Bauckhage, and A. B. Cremers, “Informed Haar-like Features Improve Pedestrian Detection.”

[26] D. Geróonimo, A. M. López, A. D. Sappa, and T. Graf, “Survey of pedestrian detection for advanced driver assistance systems,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 7, pp. 1239–1258,Jul. 2010.

[27] "Blind spot definition". *Thefreedictionary.com*. Retrieved 2011-11-13.

[28] https://en.wikipedia.org/wiki/Vehicle_blind_spot#cite_note-1

[29] https://en.wikipedia.org/wiki/Pillar_(car)

[30] C. Quigley, “Field of vision (A-pillar geometry) - a review of the needs of drivers : nal report . Field of vision,” 2001.

[31] https://docs.unity3d.com/ScriptReference/Vector3-sqrMagnitude.html

[32] http://www.sumo.dlr.de/index.html

[33] https://unity3d.com/es

[34] M.D Ugarte, A F. Militino, Alan T. Arnholt, “Probability and Statistics with R”,2008.

[35] C. Olaverri, M. Gvozdic, and B. Muthurajan, “Effect on Driving Performance of Two Visualization Paradigms for Rear-End Collision Avoidance,” pp. 37–42, 2017.

[36] Olaverri-Monreal, C., Gomes, P., Krüger Silvéria M., Ferreira, M. (2012) “In-Vehicle Virtual Traffic Lights: a Graphical User Interface", *Proceedings 7th Iberian Conference on Information Systems and Technologies, CISTI'2012,* Madrid, Spain, pp. 1 – 6

[37] G. K. Kountouriotis, P. Spyridakos, O. M. J. Carsten, and N. Merat, “Identifying cognitive distraction using steering wheel reversal rates,” *Accid. Anal. Prev.*, vol. 96, pp. 39–45, 2016.

List of Figures

Figure 1: Road accident fatalities by category of vehicle in European Union in 2013……….………..…7

Figure 2: example of an appearance and motion-based pedestrian detection system……….........11

Figure 3: 3DCoAutoSim main menu graphical interface……………………….………..……..……..14

Figure 4: available environments in 3DCoAutoSim…………………………………………..……14

Figure 5: schematic representation of the first steps performed by CoAutoSim3D…………………………………………………………………………….……….....15

Figure 6: SUMO environment of CoAutoSim3D when a .cfg file is loaded containing basic environment to be further developed………………………………………………………………………...…………..…15

Figure 7: Logging framework design[2]…………………………………………………………...…16

Figure 8: .net file obtained from *osmWebWizard* before editing……………………………..............18

Figure 9: example of a pedestrian topology in *netedit* before edition……………………………….……..18

Figure 10: example of a pedestrian topology in *netedit* after edition…………………………………….….18

Figure 11: example of lane connections in *netedit* before edition………………………………….…......19

Figure 12: example of lane connections in *netedit* after edition……………………………………………..19

Figure 13: network obtained after the edition process………………………………………………………...20

Figure 14: traffic jam obtained after running the first .cfg file in SUMO-GUI…………………………….21

Figure 15: TraaS web service implementation………………………………………………………………...23

Figure 16: Driver-Centric perspective of the 3D environment…………………………………………..25

Figure 17: schematic representation of the approach used for instantiating static data…………..…26

Figure 18: schematic representation of the inputs required and the outputs generated by each program…………………………………………………………………………………………………..…27

Figure 19: buildings imported in Unity 3D and instantitated within the environment…………….…..27

Figure 20: schematic representation of the approach used to classify roads……………………….…29

Figure 21: schematic representation of the algorithm for printing and moving pedestrians……..….35

Figure 22: Coordinate system of Unity 3D……………………………………………………………………………….......36

Figure 23: Driver-Centric perspective of a pedestrian crossing a pedestrian crossing…………………..….39
Figure 24: front view of a pedestrian walking on a sidewalk…………………………………………...…………….39

Figure 25: example of xml code of the TLS created for junction 655061 as section of the.net file linked in the configuration file loaded in Unity……………………………………………………..………………….....40

Figure 26: example of the TLS for junction 655061 displayed in SUMO-GUI…………………….………….....41

Figure 27: example of the xml code of the TLS created for junction joinedS_10 as section of the.net file linked in the configuration file loaded in Unity…………………………………………………..…………….41

Figure 28: example of the TLS for junction Joined S10 displayed in SUMO-GUI…….……………………......42

Figure 29: example of the traffic lights and road-markings placed on an intersection in Unity 3D……...…….43

Figure 30: driver-centric perspective when when condition (a) is satisfied and the P2V system is activated…………………………………………………………………………………………………..……….…...51

Figure 31: driver-centric perspective when condition (b) is satisfied and the P2V system is activated………...…….51

Figure 32: driver-centric perspective of a dangerous situation when the detection system is not active………...…..….52

Figure 33: top-view of a dangerous situation……………………………………………………..…….....……..52

Figure 34: driver-centric perspective of a dangerous situation when the detection system is active……..52

Figure 35: algorithm used for developing the P2V system…………………………………………………….......57

Figure 36: flow chart of the experimental procedure to measure driver response to the P2V system………....61

List of Tables

Table 1: example of the mode-durations computed for the traffic light TLS 655061_1……………..……41

Table 2: example of the mode-durations computed for the traffic light Joined S10_4……………..…..42

Table 3: example of the corrected mode-durations computed for the traffic light Joined S10_4……..…42

Table 4: example of the mode-durations computed for the traffic light Joined S10_5…………………..43

Table 5: content of each of the logged file……………………………………………………………………………..62

Table 6: general form of a 2x2 table to apply Fisher's Exact Test[34]……………………………………………..66

Table 7: 2x2 table obtained from section 5 to apply Fisher's Exact Test………………………………………66

Table 8: 2x2 table obtained to apply Fisher's Exact Test ……………………………………………………….…67

Table 9: driving performance values depending on whether the P2V system is activated or not………..68

Table 10: average and variance of the results obtained from the post-task questionnaire(questions 1,2 and 5)…………………………………………………………………………...68

Table 11: results obtained from the short-questions of the post-task questionnaire……………………..68

List of Abbreviations

ITS	Intelligent Transportation System
V2P	Vehicle-to-Pedestrian
P2V	Pedestrian-to-Vehicle
P2I	Pedestrian-to-infrastructure
APD	Automated Pedestrian Detection
CPS	Countdown Pedestrian Signal
HOG	Histogram of Oriented Gradients
TOR	Take Over Request
SUMO	Simulation of Urban Mobility
TraaS	TraCi as a Service
TraCI	Traffic Control Interface
UCV	User-Controlled Vehicle
SCV	System-Controlled Vehicle
VRU	Vulnerable Road Users
TCP	Transport Control Protocol
OSM	Open Street Map
TLS	Traffic Light System
HUD	Heads-Up Display
FOV	Field Of View
PSM	Personal Safety Messages
GPRS	General Packet Radio Services
GPS	Global Positioning System
DCR	Deceleration Change Rate

A: Instructions and pre-task questionnaire

You're going to take part in a driving simulator experiment. We appreciate your availability and help with this study.
Before starting you are requested to please answer these short questions:

Name	
Age	
Gender	
Experience with driving simulators (Y/N)	
Do you have any visual impairment? If yes, describe	

Experiment: you will drive three times. You will start driving for 3 minutes to get familiarized with the scenario. No data will be logged in this case. Then, after restarting the platform you will be driving for approximately 5 minutes until you reach three stop signals in a given path that will be signalized by arrows and data will be logged. You will be searching for a specific pen in a full bag for 20 seconds. Then you will drive again in the same scenario.

This data will not be given to third parties and will only be used for this research.

You must respect speed limits and it is recommended to drive approximately to 40 km/h in straight sections. Since the scenario may contain pronounced curves, it is recommended to drive below 20 km/h when reaching a curve.

B: Post-task questionnaire

Q1: Please rate your simulator experience (circle one number)
1 2 3 4 5

Q2: How realistic did you find the visual display?
1 2 3 4 5

Q3: Was the alarm included in the detection system annoying or useful?
Annoying / Useful

Q4: Did the green square of the head-up display reduce driver's visibility?
Yes / No

Q5: The pedestrian detection system was useful
1 2 3 4 5

Punctuation	Meaning
1	Strongly disagree
2	Disagree
3	Neither agree nor disagree
4	Agree
5	Strongly agree

C: Class pedestrian

Class Pedestrian	Pedestrian.cs

```
public class Pedestrian
{
//attributes
    private double x, z, speed, angle;
    private string id;

//Constructor
    public  Pedestrian(SumoPosition2D  position,  string  id,  double  speed,
double angle)
    {
        this.x = position.x;
        this.z = position.y;
        this.id = id;
        this.speed = speed;
        this.angle = angle;
    }
//Getters
    public double getX()
    {
        return this.x;
    }
    public double getZ()
    {
        return this.z;
    }
    public Vector3 getPos()
    {
            return new Vector3((float)x, 0.5f, (float)z);
    }
    public double getAngle()
    {
        return angle;
    }
    public double getSpeed()
    {
        return this.speed;
```

```
    }
    public string getId()
    {
        return this.id;
    }
}
```