POLITECNICO DI TORINO

Master's Degree Course in Mechatronical Engineering

Master's Degree Thesis

Simulation of a virtual Traffic Light System using IEEE 802.11p

Supervisor:
prof. Claudio Casseti

Candidate:
Gabriel Martnez
Student Id: 238269

Academic Year 2018-2019

Abstract

The automotive industry is one of the most important in the world, moving annually billions of US dollars, and it is one that it is being reinvented year by year, together with the advancement in technology, and the development of new strategies for the different necessities that arises, specially in security.

Of special interest nowadays are the autonomous cars, cars that have for objective easing the labour of from the driver of the car, and this is done by either adding new functionalities to the car or in most extreme case, taken away the control of different labours inside the car. This as an end makes the the action of driving more automatic and in the ideal case, more secure for the drivers, passengers and actors on the road.

The different functionalities that offer an autonomous car are of a modular nature, this means that they can be discussed tested and analysed separately, instead of being a big ensemble, nonetheless like a final test they should always being tested in conditions where all the intended functionalities are working in parallel. This thesis is about the intersection control of junction in urban roads.

One of the new tools that will be a disposition of the autonomous car in the near future will be the 5G network, in which the cars will be using the protocol knows as IEEE 802.11p for network communication for the automotive industry. The analysis of a junction intersection in an urban road is done using the framework environment of Veins the open source vehicular network simulation framework, which use the simulator of urban mobility SUMO for the simulation of comportment of a car and the environment in which they are simulated, and OMNet++ for the simulation of network communication within the cars and of the control itself.

The focus of study is a control strategy for decreasing the total travel time of every car in the simulation when the cars are under a congestion situation of the junction under study. It is created a framework environment that contain a control algorithm that works with a decentralized scheme, that is, every car has the option to decide how it will operate, instead of using an external source or infrastructure. Using the aforementioned 5G network, the cars communicate between each other so they can arbitrate the travel priority on the intersection.

The control strategy is tested under four different situations, where the performance of it in every case is successful, enabling the system under study under all the different scenarios to reduce the total travel time of the cars. Even more, the control assure that the total travel doesn't exceed a certain max travel time, so it could be used for predicting certain situation in worst case scenarios.

The proposed framework of control present an innovative approach to the the problem of crossing intersection in urban environment, one that can be easily implemented and it nos process consuming, at the same time that effectively enough that can be considered for real life scenarios. Recommendations of new additions to the framework are mentioned whenever it is deemed loadable.

To my family, friends, and the ones that were left in the way, both in Italy & Chile

Acknowledgements

Si algunas personas me preguntan cuál fue uno de mis *turning point* en mi vida, seguramente tendría que remontarme a uno de esos días del año 2014 cuando sentado junto a Cesar O. y Daniel A., este último nos empez a convencer de ir a hacer un programa llamado doble título, a un país llamado Italia (esto a pesar de que al final este último no se unió al programa).

Y si bien al principio me mostré incrédulo, después de un poco de convencimiento y la opini´n de mi padre, empecé a considerarlo una verdadera opción.

De ahí comenzó el camino para poder quedar aceptado, un camino que en particular fue difícil. Me hubiera gustado que hubiera sido mejor, pero no lo fue, y solo gracias al apoyo de mis padres y mis amigos, los cuales estuvieron ahí en mis más bajos momentos es que logre ser aceptado.

Llegue a Italia, a una tierra desconocida, siendo la primera vez que tomaba un avión, la primera vez que salía del país de origen. Y muchas de las expectativas que tenía cuando comencé este viaje se cumplieron, y muchas otras que no imagine tener también se realizaron.

El doble título, que se está culminando a medida que escribo las últimas palabras de esta tesis, marca un ciclo en el que me hice adulto, crecí como persona y aprendí del mundo.

...

De los grupos de personas de los que tengo que hablar son dos, aquellos que conocía antes de este gran viaje, y aquellos que conocí en la tierra lejana conocida como Italia.

En mi tiempo en Italia conocí a un grupo de personas que realmente me hicieron crecer y madurar: al grupo de chilenos que me acogió y me prepararon para mis primeros días en un país extranjero, Víctor que hoy en día es uno de mis conocidos que más aprecio y Nicolás que sus conversaciones siempre me sirvieron para reflexionar sobre la vida.

Al grupo de extranjeros que conocí acá, al Argentino, que me permitió ver la vida desde el punto de vista de un artista frustrado, al Boliviano, al que seguramente le debo la mayor parte de mi crecimiento como persona en estos últimos meses fuera del ámbito académico.

A mi compañero de casa, Gunav, que me ha ayudado a conocer un gran grupo de gente, y que me permite mantenerme en contacto con la parte joven de mí y que por mucho tiempo hice caso omiso.

Al grupo de chilenos con el que llegue, en especial a Cesar O., que a lo largo de los años lo he llegado a considerar un verdadero amigo.

A mis amigos que se quedaron en Chile, los que me escucharon en mis momentos más complicados en mi estancia en Italia, y a los cuales hasta el día de hoy converso activamente; Rolf, quizás la única persona con la que siento que puedo hablar de cualquier cosa; José M., una de las personas con las que realmente siento que aprendo algo nuevo cada vez que hablo con él, y cuyas anécdotas de historia hicieron que se volviera a prender ese gusto por la historia, el cual había perdido; y Miguel D., quizás a mi amigo de más tiempo, el cual siempre me escucho cuando lo único necesitaba era que alguien me escuchara, y que siempre estuvo ahí cuando lo necesite.

Y finalmente, a mi madre y mi Padre, que sacrificaron mucho para poder tenerme donde estoy ahora. Sin ellos este pequeño sueño de ir a estudiar a otro país y terminar el doble título no sería posible. Sin olvidar a mi hermano, que si bien no hemos hablado, sé que se enorgullece de tenerme como hermano, y eso me llena de energía.

A todas estas personas que mencione, y a muchas otras que no puedo mencionar para no hacer esto más extenso, Muchas Gracias.

Contents

1 Introduction **12**
1.1 State of art . . . 12
1.2 Autonomous Driving Overview . . . 13
1.3 IEEE 802.11p Overview . . . 14

2 Software to use **16**
2.1 SUMO . . . 16
2.2 OMNeT++ . . . 16
2.3 Veins . . . 17
2.3.1 How it works . . . 17
2.3.2 Initialization . . . 18

3 Modeling the Problem **19**
3.1 Characterization of the intersection . . . 20
3.2 Possible Improvements or changes to the characterization . . . 20

4 Description of the control **21**
4.1 Assumptions . . . 21
4.2 Logic behind the controller . . . 21
4.2.1 Centralized vs Decentralized Control . . . 21
4.2.2 Adaptive Controller . . . 22
4.2.3 Traffic Congestion . . . 22
4.3 Parameters of the Controller . . . 23
4.4 Structure of the controller . . . 25
4.4.1 State of the semaphore . . . 25
4.4.2 Internal Message . . . 25
4.4.3 External Message . . . 27
4.4.4 Functional Functions . . . 29
4.5 Workflow . . . 30
4.5.1 Initialization of the Control Algorithm . . . 30
4.5.2 State of the Car . . . 31
4.5.3 Control of States . . . 32
4.5.4 Synchronization . . . 33

5 Tests **35**
5.1 Test 0 - Basic Base Case . . . 36
5.1.1 Uncontrolled Simulation . . . 36
5.1.2 Controlled Simulation . . . 38
5.2 Test 1 - Extended Base Case with heavy and intermittent flow . . . 40
5.2.1 Uncontrolled Simulation . . . 41
5.2.2 Controlled Simulation . . . 44
5.3 Test 2 - Heavy flow random vehicle . . . 46
5.3.1 Uncontrolled Simulation . . . 46
5.3.2 Controlled Simulation . . . 49
5.4 Test 3 - Early finish in a heavy congested lane in only one direction . . . 52
5.4.1 Uncontrolled Simulation . . . 52
5.4.2 Controlled Simulation with Early Finish On/Off . . . 54

6 Analysis **57**
6.1 Test 0 57
6.2 Test 1 58
6.3 Test 2 59
6.4 Test 3 60

7 Conclusions **62**

References **63**

List of Figures

1 Levels of autonomous Driving - Automated Vehicles for Safety, accessed on 2019 [3] 13
2 Veins - How does Veins Work?, accessed on 2019 [14] 18
3 Base Case: four-way intersection 19
4 Evaluation enter to activate control algorithm 31
5 Empiric Rules for activating Control 31
6 stateControl() Function 32
7 checkState Call 32
8 changeState() Function 33
9 Finish Early Routine 33
10 Synchronization Message Dynamic 34
11 Histogram Test 0 37
12 Histogram Horizontal Travel 37
13 Histogram Vertical Travel 38
14 Histogram Test 0 39
15 Histogram Horizontal Travel 39
16 Histogram Vertical Travel 40
17 Histogram Test 1 with Control Off 42
18 Histogram Starting from East 42
19 Histogram Starting from North 43
20 Histogram Starting from East 43
21 Histogram Starting from South 43
22 Histogram Test 1 with Control On 44
23 Histogram Starting from East 45
24 Histogram Starting from North 45
25 Histogram Starting from East 45
26 Histogram Starting from South 46
27 Histogram Test 2 with Control Off 47
28 Histogram Starting from East 48
29 Histogram Starting from North 48
30 Histogram Starting from East 48
31 Histogram Starting from South 49
32 Histogram Test 2 with Control On 50
33 Histogram Starting from East 50
34 Histogram Starting from North 51
35 Histogram Starting from East 51

36 Histogram Starting from South 51
37 Histogram Test 3 With controll Off 53
38 Histogram Horizontal Travel 53
39 Histogram Vertical Travel 54
40 Histogram of Test 3 With Control On 55
41 Histogram of Direction E-W 55
42 Histogram of Direction N-S 55
43 Histogram of Direction W-S 56
44 Histogram of Direction S-N 56
45 Histogram of No Control and Control Situation with Green 10s / Red 20s . 66
46 Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s 66
47 Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s 67
48 Histogram of No Control and Control Situation with Green 10s / Red 20s . 68
49 Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s 68
50 Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s 69
51 Histogram of No Control and Control Situation with Green 10s / Red 20s . 70
52 Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s 70
53 Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s 71
54 Histogram of No Control and Control Situation with Green 10s / Red 20s . 72
55 Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s 72
56 Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s 73
57 Histogram of No Control and Control Situation with Green 10s / Red 20s . 74
58 Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s 74
59 Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s 75
60 Histogram of No Control and Control Situation with Green 10s / Red 20s . 76
61 Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s 76
62 Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s 77

List of Tables

1 Floats 36
2 Data Test 0 with Control Off 36
3 Data Test 0 with Control On 38
4 Floats 41
5 Data Test 1 41
6 Data Test 1 44

7 Floats 46
8 Data Test 2 47
9 Data Test 2 49
10 Floats 52
11 Data Test 3 with Early Finish 52
12 Data Test 3 with Early Finish 54
13 Data Test 3 without Early Finish 54
14 Data Test 0 57
15 Results Test 0 58
16 Data Test 1 58
17 Results Test 1 59
18 Data Test 2 59
19 Results Test 2 60
20 Data Test 3 60
21 Results Test 3 61
22 Global result of test with car generation of 30 [s], for different state transition time 65
23 Global result of test with car generation of 16 [s], for different state transition time 67
24 Global result of test with car generation of 14 [s], for different state transition time 69
25 Global result of test with car generation of 14 [s], for different state transition time 71
26 Global result of test with car generation of 14 [s], for different state transition time 73
27 Global result of test with car generation of 14 [s], for different state transition time 75

1 Introduction

The automotive industry is one of the most relevant in our life, used for transportation of persons or goods, for moving from the house to the job or simply in order to go buy groceries, the impacts that this particular industry has in our life can't be objected, and the different advancement that are being produced and developed for this industry are of special interest for a great number of peoples.

Of special interest are two different technologies: Autonomous Driving, the capacity of the car to be self-driven, without or partial necessity of a driver; And the technology known by V2X - Vehicle to everything - the technology of the car to send and receive information from other cars or infrastructures (of interest in this infrastructures is the R.S.U - Road Side Unit).

Because of this two technologies, there is a necessity of development in two different areas, namely hardware and software, the need of specialized embed system for processing the most quickly possible the data that it is being capture by the different sensors of the car, together with better and reliable sensors, and specialized algorithms that has to be both quick and efficient, in the way that they don't have to suffer from delays when they are processing the data coming from the hardware.

This thesis present a framework environment that includes an algorithm thought for cars with autonomous driving, a technology that it is in development right now by different automakers in the world, and that has the possibility of using V2X communication - Technology thought for 5G networks. This algorithm is thought for intersection of urban areas, and search to reduce congestion time of cars waiting in line in this roads. The necessity of an algorithm that it is efficient (that doesn't require so many computations), and to be flexible enough that can be adapted to multiple situations are the goals of this. The approach is novel enough, because meanwhile there are other algorithms thought for similar situations, they require a R.S.U. for every intersection, adding in a cost of effectively producing this solution (installation and maintenance of them for every intersection under study), and the proposed solution use a decentralised scheme, where it is only required that the cars have a good enough channel of communication.

1.1 State of art

Autonomous driving is the next big stop for car makers, many of them are making sure to invest in this technology, or in one way or another to joint venture with smaller companies that work in this area.

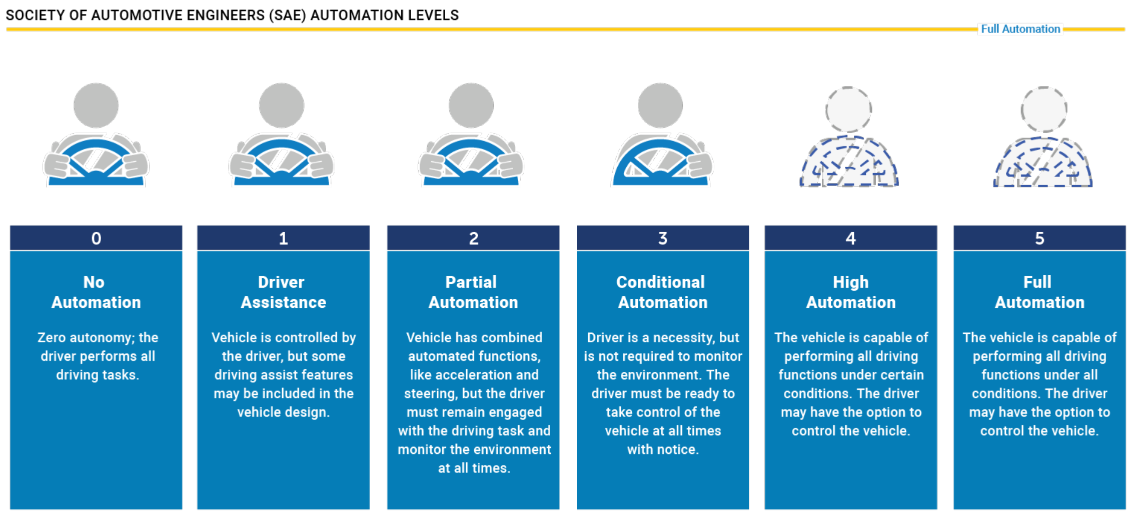
Meanwhile there isn't a deadline of when this technology is going to be implemented, given the difficulties of what this entails and the different definitions of what it is autonomous driving (there exist five levels, with level 5 being fully autonomous, and right now we are in level 2, as defined by 1.2), the different car makers are given different possible dates [1] of when this technology will be available for the public, with most of them coinciding in the next decade.

For cars with this technologies, it is not only necessary that other cars have to support this technology, but it is also important to create roads and infrastructure that support this technology. Meanwhile there are different car makers that are putting different cars on the road in order to test it, Volvo is testing this technology on a closed environment in a road in Gothenburg, Sweden, in order to develop not only autonomous driving, but also infrastructure to support this technology[2] .

In what respect to the control algorithm, the focus is not on how the car moves, so it will be assumed that the car has an autonomous level of at least 3. Calling to the combination of roads and cars a "system", the different algorithms require an external agent that control the cars, from outside the system, like in [4] where the computations of controls are done in a network that control all the cars of the systems, and it is updating constantly the position of the cars on the road, or like in the Autonomous Intersection Management (AIM) framework [5], where it is possible to reach a continuous flow of cars on an intersection, without ever stopping, but with the requirements that the computations has to be done on an R.S.U.

1.2 Autonomous Driving Overview

The US organization of National Highway Traffic Safety Administration (NHTSA) released on 2016 a document that intended to put on a framework a policy over the safe deployment of automated vehicles. This document was reviewed on 2017 and later on 2018 and in them, the NHTSA define the six different levels of automation on a vehicle as shown on Figure 1.


SOCIETY OF AUTOMOTIVE ENGINEERS (SAE) AUTOMATION LEVELS

Full Automation

0	1	2	3	4	5
No Automation	**Driver Assistance**	**Partial Automation**	**Conditional Automation**	**High Automation**	**Full Automation**
Zero autonomy; the driver performs all driving tasks.	Vehicle is controlled by the driver, but some driving assist features may be included in the vehicle design.	Vehicle has combined automated functions, like acceleration and steering, but the driver must remain engaged with the driving task and monitor the environment at all times.	Driver is a necessity, but is not required to monitor the environment. The driver must be ready to take control of the vehicle at all times with notice.	The vehicle is capable of performing all driving functions under certain conditions. The driver may have the option to control the vehicle.	The vehicle is capable of performing all driving functions under all conditions. The driver may have the option to control the vehicle.

Figure 1: Levels of autonomous Driving - Automated Vehicles for Safety, accessed on 2019 [3]

Nowadays, car makers says that we are in transition toward level 3 of autonomous driving, or level 2.5, with mixed functionalities of level 3 and 2. It is expected that for the year 2035, it will be available the technology for autonomous driving level 5 [6], but this doesn't mean that the technology will be comerciable right away.

1.3 IEEE 802.11p Overview

The organization know as IEEE (Institute of Electrical and Electronics Engineers) funded in 1884, it is the world's largest technical professional society, in which they research and promote different advancement of electrical and electronic engineering, telecommunications, computer engineering and allied disciplines[7].

In the area of telecommunications, the standard IEEE 802.11 is a set of protocols, for implementing wireless local area network (WLAN). This standard specifies:

1. Physical layer (PHY): The electronic circuit transmission technologies of a network.
2. Media Access control (MAC): Hardware responsible for the interaction with the wired, optical or wireless transmission medium.
3. Interconnection between devices: The procedures in which the different devices are connected.
4. Security: The standard of security that has to be reproduced that has to be present.

Following this, WiFi is a certification of interoperability and standard compliance released by the WiFi Alliance for devices based on 802.11 standards.

The first standard was created in 1999, with the creation of the IEEE 802.11. Since then, the IEEE has produced several amends to the original standard in order to update the technologies, or for incorporating new services.

IEEE 802.11p is an amend produced of 2010 to add Wireless Access in Vehicular Environment (WAVE), the so called vehicular communication system. As described by [8] and [9] it sets a series of requirements to be followed in order to support Intelligent Transportation Systems. This are:

1. A method to create a high velocity connection within the cars, without the necessity of establishing a basic service set (BSS), instead they used a wildcard basic service set identifier (BSSSID) to stablish a quick connection. Thus without the need to wait on the association and authentication procedures to complete prior to exchanging data. Because such stations are neither associated nor authenticated, the authentication and data confidentiality mechanisms provided by the IEEE 802.11 standard (and its amendments) cannot be used. These kinds of functionality must then be provided by higher network layers.
2. This amendment adds a new management frame for timing advertisement, which allows IEEE 802.11p enabled stations to synchronize themselves with a common time reference. The only time reference defined in the IEEE 802.11p amendment is UTC.
3. Some optional enhanced channel rejection requirements (for both adjacent and nonadjacent channels) are specified in this amendment in order to improve the immunity of the communication system to out-of-channel interference. They only apply to OFDM transmissions in the 5 GHz band used by the IEEE 802.11a physical layer.

4. Use of the frequency bands licensed for ITS applications. IEEE 802.11p standard typically uses channels of 10 MHz bandwidth in the 5.9 GHz band (5.850-5.925 GHz). This allows the receiver to better cope with the characteristics of the radio channel in vehicular communications environments, e.g. the signal echoes reflected from other cars or houses.

In the scope of this thesis, all the communications between the cars are effectuated by the simulation software that will be following this standard of communication, making it a valid ITS application.

2 Software to use

This thesis make use of two different programs and one computational framework for running the different simulations and for the programming of the control algorithm. The specifications used for each of this, and how they operate will be explained at continuation.

2.1 SUMO

As described by [10] and [11], "Simulation of Urban MObility" (Eclipse SUMO) is an open source, highly portable, microscopic and continuous road traffic simulation package designed to handle large road networks. SUMO is licensed under the Eclipse Public License V2.

SUMO is a traffic simulation package, that can simulate networks of any sizes, given that the computer power is large enough. SUMO is mainly a microscopic, space-continuous road traffic simulation. What it means to be a "microscopic" traffic simulator is that each vehicle and its dynamics are modeled individually. It supports multi-modal and inter-modal ground based traffic. SUMO models individual vehicles and their interactions using models for car-following, lane-changing and intersection behavior. It also uses pedestrian models to simulate the movement of persons and their interactions with vehicles.

For the purpose of this Thesis, the model of simulation used is the default model used by SUMO: It is a modification to the microscopic model defined by Stefan Krauβ in [12]. The model follows the same idea as that of Krauβ, namely: Let vehicles drive as fast as possibly while maintaining perfect safety (always being able to avoid a collision if the leader starts braking within leader and follower maximum acceleration bounds) with the following differences:

- Different deceleration capabilities among the vehicles are handled without violating safety (the original model allowed for collisions in this case).
- The formula for safe velocity was adapted to maintain safety when using the simulator, that is, changing the continuous time model of Krauβ to a discrete one, thus avoiding collisions.

In order for SUMO to operate with other software, it makes use of TraCI command, this is the short term for "Traffic Control Interface". Giving access to a running road traffic simulation, it allows to retrieve values of simulated objects and to manipulate their behaviour "on-line".

Given the limitation of the version used in Veins [2.3], the version of SUMO used is SUMO 0.32.0.

2.2 OMNeT++

As described by [13], OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queuing networks, and so on. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects. It has extensions for real-time simulation, network emulation, database integration, SystemC integration, and several other functions.

Although OMNeT++ is not a network simulator itself, it has gained widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community.

OMNeT++ provides a component architecture for models. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED).

It will be used with the simulation framework (model) of vehicular networks Veins [2.3]. Given the limitation of the version used is Veins [2.3], the version of OMNeT++ used is OMNeT++ 5.4.1.

2.3 Veins

As described by [14], Veins is an Open Source vehicular network simulation framework, ships as a suite of simulation models for vehicular networking. These models are executed by an event-based network simulator (OMNeT++ [2.2]) while interacting with a road traffic simulator (SUMO [2.1]). Other components of Veins take care of setting up, running, and monitoring the simulation.

This constitutes a simulation framework. What this means is that Veins is meant to serve as the basis for writing application-specific simulation code. While it can be used unmodified, with only a few parameters tweaked for a specific use case, it is designed to serve as an execution environment for user written code. Typically, this user written code will be an application that is to be evaluated by means of a simulation. The framework takes care of the rest: modeling lower protocol layers and node mobility, taking care of setting up the simulation, ensuring its proper execution, and collecting results during and after the simulation.

Veins contains a large number of simulation models that are applicable to vehicular network simulation in general. Not all of them are needed for every simulation – and, in fact, for some of them it only makes sense to instantiate at most one in any given simulation. The simulation models of Veins serve as a toolbox: much of what is needed to build a comprehensive, highly detailed simulation of a vehicular network is already there.

2.3.1 How it works

As discussed before, with Veins each simulation is performed by executing two simulators in parallel: OMNeT++ (for network simulation) and SUMO (for road traffic simulation). Both simulators are connected via a TCP socket. The protocol for this communication has been standardized as the Traffic Control Interface (TraCI). This allows bidirectionally-coupled simulation of road traffic and network traffic. Movement of vehicles in the road traffic simulator SUMO is reflected as movement of nodes in an OMNeT++ simulation. Nodes can then interact with the running road traffic simulation, e.g., to simulate the influence of IVC on road traffic.

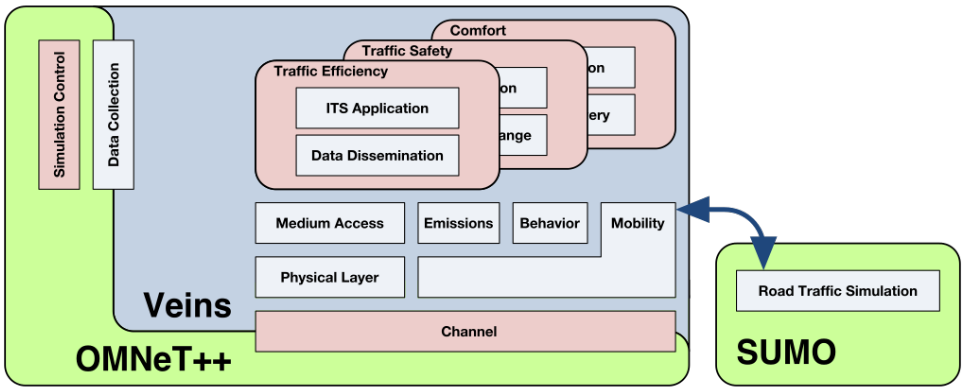


Figure 2: Veins - How does Veins Work?, accessed on 2019 [14]

2.3.2 Initialization

As per this Thesis, the last stable version of Veins is Veins 4.7.1, for which it is going to be used. From this version, it is going to be used the fully-detailed model of IEEE 802.11p.

As per documentation of Veins, and for this thesis in particular, Veins is going to be installed, in "..\src\veins-4.7.1" and from here on, this will be the path referred for any file used or modified in this thesis.

For simplification, we are going to use the example that bring Veins by default, and from there on modify files according to necessity.

3 Modeling the Problem

This section will refer to the description of the problem, and the characterization of the automotive intersection. The parameters introduced in the characterization of the intersection will be later be used on the control algorithm.

For the porpoise of this thesis, we will be using a four-way automotive intersection of 90 degree between each of the lanes (see Figure 3). The cars can come from one of the four possibly entrance, namely from L1 to L4, and can exit from one of the four possible exits, namely from E1 to E4.

Every road is a two car lane, forming what it is known as a four lane highway. The intersection will be an urban intersection, thus limiting the maximum velocity achievable to the cars, that in this case will be $14m/s$ ($50.4km/hr$).

Because the cars are simulated by SUMO and they have the possibilities of being controlled in an automatic and external way, it will be assumed that the cars are autonomous, and that the simulation program is simulating like this. This means that the cars travel in an efficient way without feedback from the conductors (this can be done because of the model used by the simulator, where it is impossible for accidents to happen unless they are from external cause).

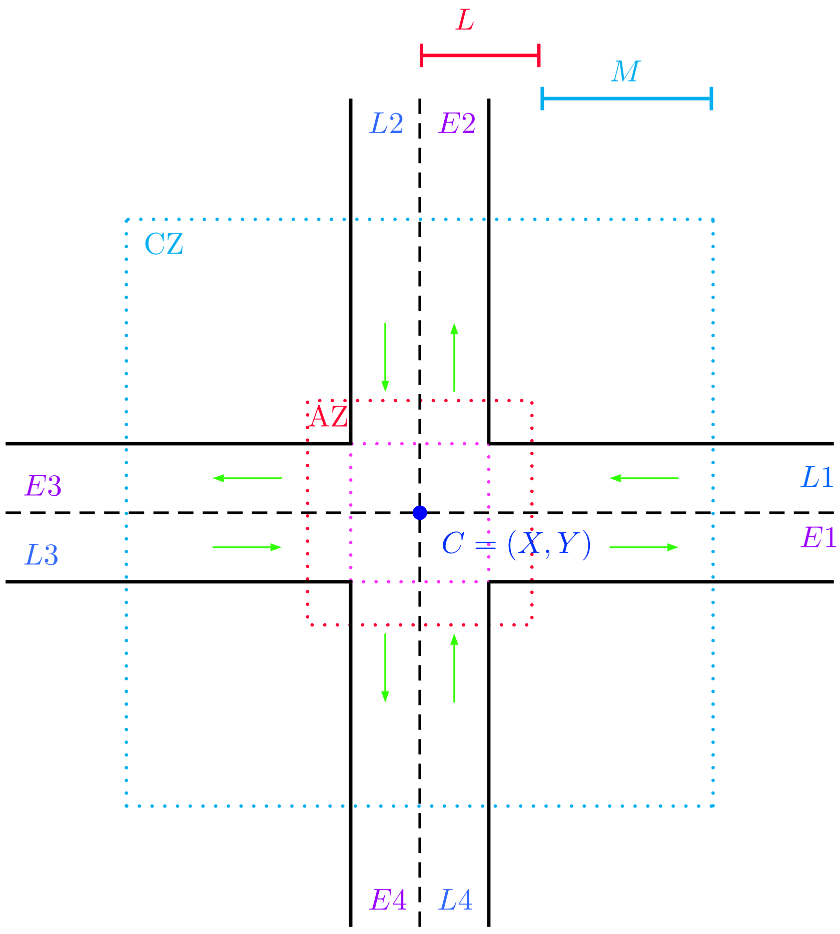


Figure 3: Base Case: four-way intersection

3.1 Characterization of the intersection

In order for the control algorithm to work, the cars has to known certain parameters that characterize the intersection. This parameters are:

- The center point of the intersection, "**C**" on Figure 3. This parameter count with two values, the X position on a cartesian map, and the Y position on a cartesian map, which can be thought as position on a GPS readily available inside the cars. It is the blue dot on Figure 3.
- The distance from the Center Point [**C**] until the start of the intersection, considering an error margin. This parameter is "**L**". In practical use, it is the point where cars stop when they are under the control algorithm. The margin error helps in two ways, the first one is the area that it is normally reserved for pedestrian to cross over. The second form of help it is because help to the simulation, and solve issues of incoming cars. Is is the red line of dots in Figure 3.

  Incidentally, the **L** parameter without this error margin it's called **trueL**, and it is the point where car start stopping in congestion. It is the pink line of dots in Figure 3.
- The distance from where the cars stop when they are under the control algorithm [**L**], until the area where the start to be controlled it is "**M**". It is the light-blue line of dots in Figure 3.

This four parameters have the possibility of characterize any 4-way intersection and T-type intersection.

These four parameter define three different area on the scheme of the control:

- The Control Zone (CZ on Figure 3), this zone goes from when a car enter throught the light-blue line until the red line.
- The Active Zone (AZ on Figure 3) compress the Control Zone plus the area between the red line until the pink line.
- Everything else is an idle zone, and that it is also true for other Active Zone or Control Zone once the car has passed the pink line.

3.2 Possible Improvements or changes to the characterization

The definitions of the different areas of control are done in a square manner because it helps to define boundaries in a more descriptive way, but this could also be done in a radial way. The square has the possibility to define different **L**'s or **M**'s when the intersection aren't symmetrical and it is necessary to have better simulations.

In the characterization, it is considered that the junction is a four way intersection with an incident angle of 90°, but one can characterize any kind of incident roads. The means to do this are is using the same approach presented here, where it would be necessary to have a parameter to describe the incoming incident road and their angles with respect to **C**. The calculus of this is done via trigonometric, and for cutting computation time on the processor of the car, one could use look up tables.

4 Description of the control

In this section we will talk about the different decisions that were made for the creation of the control algorithm. There is also a detailed description of the final control algorithm, finalizing with a state diagram of the different part of the algorithm.

4.1 Assumptions

Because this control work under the assumptions that the cars under simulation are of autonomous nature, it will not intervene with the internal control structure of what it is expected of the internal works of the autonomous driving algorithm, excepting for two situations:

- When the car enter the Active Zone, the rule of Brake hard of the simulator to avoid passing a red light (Speed Mode bit4 of SUMO see [16]) it is deactivated because the junction doesn't have a semaphore element on the road and because otherwise the simulation present problem with the cars that stop because of other cars.
- When the car is under the control algorithm and it is in Green State or Yellow State, the rule of Regard right of way at intersections in the simulator (Speed Mode bit3 of SUMO see [16]) it is deactivate because is it not necessary to check cars coming from the side - they are stopped and currently in Red State. This can cause problem for the algorithm if it is not deactivate, with car not responding to the velocity control.

In every other instance, the car respect the safe velocity speed to follow (speed of the road and speed following the car), and the car react to the car next in line following a scheme that it is similar to the Adaptive Cruise Control in Urban Areas that can be found in some cars being produced today. This last take priority over the instruction of the control.

4.2 Logic behind the controller

4.2.1 Centralized vs Decentralized Control

In general terms, when a control is being developed, outside of the algorithm being created the developer has to select the type of the scheme of the control, this is, design a centralized control algorithm or a decentralized control algorithm.

A centralized control algorithm is one in which all the decision are done by a central controller. The controller has to receive the information from different agents of the control developed, and then based on the information received by them, give orders to the actuators in the control. This is the approach generally done for what this thesis is trying to do (see State of Art 1.1), where the controller is the R.S.U. of the intersection.

There is a fundamental problem with this approach in the current situation of the thesis [3], is that to be valid it should be necessary to have one R.S.U. for every intersection. This is impracticable in practice because of the cost that imply an R.S.U in itself, plus the maintenance and installation of the structure. This make a costly solution, at least until better and more cheaper form of transmission exists.

A decentralized control algorithm is one where every agent of the control scheme has the possibility to control itself. To the problem of this thesis in particular, this means that every car has the possibility of select its own state, or arbitrate this state, without the necessity of having an exterior agent controlling them (the R.S.U.).

The cons of this approach are that because the cars don't have a central decision center that know the state of all the agents in the system in a any given time, the cars needs to stop under certain situation during the travel because of the change of state (at least in this scheme), given a greater travel time that the one that can be obtained with [5], where the cars never stop, but it is a control much more approachable in the shorter term.

4.2.2 Adaptive Controller

When making test to see the average time of cars with a normal semaphore scheme (continuous semaphore controller), in every test under uncongested conditions, cars without control where better that cars with controller (See annex [7]). Instead, the controller was successful in regulate the flow of cars in congested situation, and maintain a continuous rate of cars. From this experiment, and because the cars travel in an efficient way, this because of the assumption that they are autonomous, we extract the following conclusions:

- In uncongested conditions of travel, it is better that the cars decide by themselves how to travel. This sometimes enable the cars to never stop meanwhile they are travelling.
- In congested situation, the algorithm of a virtual traffic light system is an acceptable solution for solving congestion and for minimizing time of travel.

4.2.3 Traffic Congestion

Automotive traffic congestion refer to the situation when in roads, the velocity of the flow of cars decrease, increasing the total travel time of the different vehicles currently travelling by the road. When the congestion occur, it is not necessary that cars stop altogether, this last situation is referred by travel jam. When cars stop, or regularly stop, the congestion also bring an increase in the queue time of the cars.

The goal for this thesis is to reduce total travel time of the cars when confronted with an urban intersection, so for this they have to detect the situations when cars are under congestion. Because one of the goal of the control algorithm it is to be the lightest possible so it doesn't affect the processor of the cars (thinking of it for a real possible implementation over different brand of cars), it has been selected empiric rules that indicate when a car enter a congestion, they are:

- A car has been stopped for more than **timeStopCar**
- A number of car equal to **adaptiveStarCar** have stopped, and it hasn't elapse **timeStopCar**.

In this manner, is not necessary that every car has to know the state of system in any given time, but when the congestion happens, they will known that they are under congestion without problem when any of the two situations above occurs. After this, the first car that detect the congestion will announce to the other cars in the system that there is a congestion.

Of relevance are two parameters defined on [15]:

- Duration of the congestion - It is the maximum amount of time that the congestion can last at any time in the system. Because normally cities has this duration characterized, it is equal to $2 \times$ **longCyclesNumber**, this give the possibility of an early exit to the adaptive control, if the intersection has a constant flow but that don't require to be controlled.
- Extent of the congestion - It is the maximum geographic extension of congestion on the transportation system at any given time. the Parameters of **L** and **M** are directly correlated to this. In the junctions when the extent of congestion is greater than $\mathbf{L} + \mathbf{M}$, **M** should increase in value. Ideally, this should be less than $\mathbf{L} + \mathbf{M}$

4.3 Parameters of the Controller

The controller use different parameters in the algorithm. This parameters can be adjusted before running the simulation. There is the possibility to implements a R.S.U. that sends the parameters via message, but this is not implemented in this thesis:

- Parameters related to characterization of an intersection, as seen on Figure 3.
  - **C_X** This parameter correspond to the coordinates X of the center of the intersection.
  - **C_Y** This parameter correspond to the coordinates Y of the center of the intersection.
  - **L** This parameter is the distance from the center of the intersection until the start of the intersection, plus a certain space for security reason.
  - **trueL** This parameter represent the true distance from the center of the intersection until the start of the intersection
  - **M** This parameter is the distance from **L** until the start of the Control zone
- Parameters related to scheduling time
  - **tControl** It is the self scheduling time that the car use to check its own state meanwhile it is in the Active Zone.
  - **tIdle** It is the self scheduling time that the car use to check its own state meanwhile it is in the Idle Zone.
  - **tGreen** It is the duration of the State 0 or Green State of the virtual semaphore.
  - **tRed** It is the duration of the State 2 or Red State of the virtual semaphore. **tYellow**, the parameter of the duration of the State 1 or Yellow State of the virtual semaphore is defined like $\mathbf{tRed} - \mathbf{tGreen}$, so **tRed** needs to be always greater or equal to **tGreen**.

  - **timeStopCar** It is the time for triggering **checkCongestion** (see [4.4.2]) when the car stop for the first time in the Active zone.

- Parameters related to thresholds or counting
  - **longCyclesNumber** It is the number of half cycles that are done in the control cycle before exiting.
  - **adaptiveStarCar** It is the number of cars that have to be surpassed by **counterStopCar** in order to activate the control algorithm.
  - **thresholdWaiting** It is the threshold to surpass in every iteration of the control for cars remaining in different roads, so the control can continue.

Apart from this, the controller has other internal parameters that can only be adjusted before running the simulation:

- **securityFactorDetention** Number used for calculating a safe distance for the detention of the car when it does a transition to State 2 or Red State, or in case that calculates that have to stop when it does a transition to State 1 or Yellow State.

Other parameters are internally to the controller, like boolean variables, or the internal state of the virtual semaphore, they normally interact with external message or are used for sending information to others cars.

- **timeAdaptive** It is the time stamp of the control equal to the moment when it is triggered the state 0 or 2 of the virtual semaphore meanwhile the control algorithm is activate. When it isn't being used for controlling, it is equal to SIMTIME_MAX.
- **semaphoreState** It is the actual state of the virtual semaphore, once the control is triggered and meanwhile it is activated.
- **DirectionReceivedWSM[2]** This parameter store the direction of travel of the received message, whenever it is relevant. This parameter relates to direction[2] [4.4.3].
- **countChanges** This is the parameter that store how many changes of half cycles has ocurred in the control algorithm. When the car trigger the control, this variable start with a value of 0, in other case, it adopt the value of the received car. This parameter relates with countMessage[4.4.3].
- **adaptiveControl** This is a boolean variable that indicates if the control is ON if its value is true, OFF otherwise.
- **counterStopCar** Variable for storing the number of cars waiting on an intersection. If exceeds **adaptiveStarCar**, the car activate the control.
- **counterWaitingCar** This is the variable that in every iteration of the cycle count how many car are waiting on a different road, in order to exit the control in an early manner if deemed loadable.
- **checkEnd** Internal wait time for triggering **checkWaiting**

4.4 Structure of the controller

The controller use internal an external messages for scheduling and for communication with other cars, respectively, together with specialized functions that call them or send them (the other functions are for code optimization or for functionality that aren't part of the control). Next it will be explained all of the critical parts of the controller.

4.4.1 State of the semaphore

Because we are simulating a virtual traffic light system, there is the necessity to simulate internally this system. It is created three different state, than in normal conditions operate in a semi periodic way. It is called "normal conditions" when the car is already synchronized to the controller established in case that it is running, and the change of state is triggered by the internal message **checkState** [**4.4.2**]. The state are:

- **Green State**: also called state 0 of the semaphore, correspond to the green light on a normal semaphore. It overrun any restriction that the car could have imposed over their velocity, restarting services.
  It comes after a **Red State** or state 2, and has a duration of **tGreen**. Meanwhile in this state, the function **stateControl()** run **doGreen()** every **tControl**.
- **Yellow State**: also called state 1 of the semaphore, correspond to the yellow light on a normal semaphore. When a car enter this state, depending on the amount of time remaining of this state, and their relative position to the end of their Control Zone, the car can continue like it would if it was in Green State, can decrease their velocity given that it would not be able to cross the intersection, or comport like if it was in red state, stopping before crossing the junction. It comes after a **Green State** or state 0, and has a duration of **tYellow**. Meanwhile in this state, the function **stateControl()** run **doYellow()** every **tControl**.
- **Red State**: also called state 2 of the semaphore, correspond to the red light on a normal semaphore. In this state, the car can't cross the junction, having only the possibility of advancing with a reduce on velocity or remain stopped. It comes after a **Yellow State** or state 1 and it last **tRed**. Meanwhile in this state, the function **stateControl()** run **doRed()** every **tControl**.

When the car are synchronizing with each other, normally on the first cycle of the control, or when the car enter to a control zone and the controller is activated, the car enter immediately to the actual state of the system, and it last the time given by the function that triggered this synchronization, that it should be the remaining time of the actual state.

4.4.2 Internal Message

The internal message are the schedule procedure of a normal program. They are called by other functions with a certain delay and normally activate other functions when they are called.

- **event** is the schedule procedure that check the position of the car on the map respect to the characterized intersection. With this we can know if the car is in the control zone or outside of it.

It is called by the function “**stateControl()**” every “**tControl**” if the car is under the control algorithm or “**tControl**” otherwise.

- **checkCongestion** when it is called means that “**timeStopCar**” has elapsed without the car starting or triggering the control by **counterStopCar** surpassing **adaptiveStarCar**. This trigger the control algorithm to take place using ”**normalCycleTraffic()**”. It is called by the function ”**stateControl()**” the first time that the car stop after entering a new Active Zone.
- **checkWaiting** When the car do a transition from state 2 to 0 or from state 1 to 2 - meaning that this message is called by the function **changeState()**, the cars check the state of the roads apart from the one in which they are, in order to determine if they can exit the control algorithm in a preemptive way if the amount of cars is inferior to **thresholdWaiting**. This is done via a message that the cars send to other cars in range. This procedure is called after it has elapsed **checkEnd** - accounting for the delay in transmission of message of the cars - and do the above mentioned.
- **checkState** has the function of making the virtual traffic light system work as intended; this means to trigger the change of semaphore in a timely manner, and it is also the responsible for ending the different cycles of control.
  It is called by different functions and at different times depending of the situation:
  - By ”**changeState()**” in a periodic manner if the semaphore has already entered this function before and it is running in a normal way. This happens always after the first synchronization of the virtual semaphore, or after the first time the car trigger the control cycle, all of this cases described by the following items. It is called in **tGreen** if the last state was Red, it is called in ”**tYellow**” if the last state was Green, and it is called in ”**tRed**” if the last state was Yellow.
  - By ”**changeStateTime(simtime_t timeToNewState)**” This functions synchronize a car that it is entering to a Control Zone with a control cycle already running. This functions it is used when the response signal of synchronization come from the same road, independent of the direction of the car. Call **checkState** in timeToNewState, that is, the remaining time of the current state synchronized with every other car already in control, in order to change the state at the same time.
  - By ”**scheduleStateTime(simtime_t timeToNewState)**” This functions synchronize a car that it is entering to a Control Zone with a control cycle already running. This functions it is used when the response signal of synchronization come from a different road, independent of the direction of the car. Call **checkState** in timeToNewState, that is, the remaining time of the current state synchronized with every other car already in control, in order to change the state at the same time.
  - By ”**changeAdaptiveStateTime(simtime_t timeToNewState)**” This functions synchronize a car that is in the control zone when a cycle is triggered, and the car isn’t the one that it is triggering the cycle. This functions is used when the initial signal of synchronization come from the same road, independent of the direction of the car. Call **checkState** in timeToNewState, that is, the

time of the remaining current state synchronized with the car that triggered the cycle, in order to change the state at the same time.

  - By "**scheduleAdaptiveStateTime(simtime_t timeToNewState)**" This functions synchronize a car that is in the control zone when a cycle is triggered, and the car isn't the one that it is triggering the cycle. This functions is used when the initial signal of synchronization come from a different road, independent of the direction of the car. Call **checkState** in timeToNewState, that is, the time of the remaining current state synchronized with the car that triggered the cycle, in order to change the state at the same time.
  - By "**normalCycleTraffic()** This functions trigger the synchronization signal with the rest of the car in the control Zone. By default, call **checkState** in "**tYellow**", in order to let any car already crossing the intersection, to finish it and prepare cars in the same road - that should be in queue because of the congestion, to cross over.

4.4.3 External Message

The external message are the medium of communication and synchronization of the cars between each other.

The message sent that comply with IEEE 802.11p in format of WSM (Wave Short Message) has the following parameters, whether all of this information is present or not depend of the type of message. This made it able to distinguish between obligatory parameters or situational:

- Obligatory Parameters
  - ***simtime_t* timestamp** Used in every instance of synchronization between cars or RSU with cars, or for verification of the message. It is the time at which the control is activated or every time that the car change from Red to Green State, or from Yellow to Red State. It is the value that send the parameter of timeAdaptive [4.3]. Can have any value from 0 until SIMTIME_MAX, that it is the maximum value that can have a simulation.
  - ***int* ccmType** Parameter for indicating the type of Message. The value can be from 0 until 4, but there is the possibility of adding another type of external message.
- Situational Parameters
  - ***int* semaphoreState** This parameter indicated the actual state of the virtual semaphore, sending semaphoreState [4.3]. It is used in message Type 1 and 4. The values can only be 0, 1 or 2.
  - ***int* countMessage** This parameter indicated how many half a cycles has occurred in the control algorithm. This is for synchronization porpoise, and also for knowing when to finish the algorithm. It is used in message Type 1 and 4. The values that can adopt goes from 0 until **longCyclesNumber** $\times 2$
  - ***int* direction[2]** This parameter has the coordinates in relation with axis X and Y, indicating direction of traveling: 1 if the car is going in a positive direction, -1 if it is going in a negative direction, and 0 if it doesn't apply. It

is for knowing the direction and lane of travel of the other cars. It is used in message Type 1, 3 and 4.

The different messages that can be sent by the cars can be understood by the type of message, under which circumstance it is send, under which circumstance it is accepted (otherwise it is ignored), the objective of the message and what it does once it is accepted.

- Type 0 Message
  - **When it is send:** Car enter control zone.
  - **When it is accepted:** Car is in active zone; Car is being controlled by the algorithm; Car has done a cycle of **event** in the control zone.
  - **Reason of the Message:** Message is asking if the control algorithm is working.
  - **What it does to the car accepted:** Send back a message Type 1 with the synchronization data.
- Type 1 Message
  - **When it is send:** Car received a Type 0 Message; Car is in active zone; Car is being controlled by the algorithm; Car has done a cycle of **event** in the control zone.
  - **When it is accepted:** timeStamp of the message less than timeAdaptive; adaptiveControl is false; Car is in Active Zone.
  - **Reason of the Message:** Send synchronization data request to cars in control.
  - **What it does to the car accepted:** Synchronized the car to the other cars already in the control cycle.
- Type 2 Message
  - **When it is send:** Car is stopped; **adaptiveControl** is false; Car hasn't sent this message before in the same Control Zone without exiting it before.
  - **When it is accepted:** Car is virtually stopped (velocity is so small that can be considered to be stopped); adaptiveControl is false; Car is in Active Zone.
  - **Reason of the Message:** Car sends a message to add a count to counterStopCar, in order of possible triggering a normal cycle of control algorithm.
  - **What it does to the car accepted:** Add a count in counterStopCar, if this value exceed adaptiveStarCar, car trigger a normal cycle of the control algorithm, sending a message Type 4.
- Type 3 Message
  - **When it is send:** Car changed semaphore state from 1 to 2 or from 2 to 0.
  - **When it is accepted:** Car in Active Zone; **adaptiveControl** is true.
  - **Reason of the Message:** Check if it is viable to exit the control algorithm in a preempive manner.

  - **What it does to the car accepted:** Check if the message received come from the other road than from the car that received the message. If it is from a different road, **counterWaitingCar** increase in one.

- Type 4 Message

  - **When it is send:** meanwhile the car is stopped, elapsed **timeStopCar** or **counterStopCar** surpass **adaptiveStarCar** before elapsing **timeStopCar**.
  - **When it is accepted:** Car is in the Active Zone; timeStamp of the message less than **timeAdaptive**.
  - **Reason of the Message:** Car start the control Algorithm, sending a message so the other cars in Active Zone, or cars that enters after the trigger, are synchronized with it.
  - **What it does to the car accepted:** Synchronize the car with the current state of the control algorithm imposed by the triggering car.

4.4.4 Functional Functions

The control algorithm has different functions, ones for optimizing the code, other for verifying certain states of the cars - for example if it is inside a certain zone - complementary functions like the ones of synchronization, and function that do the functional work of the car. The cars that will be listed next are the most important for the work of the code.

- **doGreen()** This function is done in order to apply to the car a comportment befitting the Green State, this means that delete any restriction over the velocity of the car, minus the safe speed.

  This function works every **tControl** meanwhile the car is under the control algorithm and the semaphore state of the car is Green.

- **doYellow()** This function is done in order to apply to the car a comportment befitting the Yellow State. This function sees the remaining time until the Yellow State change to Red, and depending of this, adjust the velocity accordingly in order that the cars with enough time and velocity can cross the junctions, otherwise the cars stop in the Control Zone.

  This function works every **tControl** meanwhile the car is under the control algorithm and the semaphore state of the car is Yellow.

- **doRed()** This function is done in order to apply to the car a comportment befitting the Red State. This function sees the remaining time until the Red State change to Green, and depending of this, adjust the velocity accordingly in order that the cars remain in Control Zone without crossing to the exclusive part of the Active Zone.

  This function works every **tControl** meanwhile the car is under the control algorithm and the semaphore state of the car is Red.

- **stateControl()** Every time that it is invoked, check the position of the car in the map with respect to the junction and act accordingly. In this way, send the initial message asking if the system is under control, check the state of the car the first time that stop in Active Zone, when it is under control modify the velocity of the according to the state, and once it exits the active zone, reset variables of control and procure to exit the control algorithm.

  This function works every **tControl** meanwhile the car is in Active Zone and **tIdle** otherwise.

- **enteringToControl()** When the car enter the Active Zone, and it is recognized like so by stateControl(), the car send a Message Type 1 asking if there is a running control algorithm.
- **exitingControl()** When the car exit the Active Zone, and it is recognized like so by stateControl(), the car reset all the variables that were involved or could have been involved in case the control algorithm wasn't running. It also reinstate the velocity control of the velocity back to the Autonomous Driving Car.
- **updateParametersOnWSMAdaptive()** This is a function of synchronization to the ongoing control algorithm if it hasn't elapsed a half cycle of control. It procures that all the variables, including the semaphore states, be in synchronization with the ongoing control algorithm, independent of the direction of the incoming synchronization message.

  This function is triggered with every Message Type 4 and some Message Type 1.
- **updateParametersOnWSM()** This is a function of synchronization to the ongoing control algorithm if it has elapsed more than half a cycle. It procures that all the variables, including the semaphore states, be in synchronization with the ongoing control algorithm, independent of the direction of the incoming synchronization message.

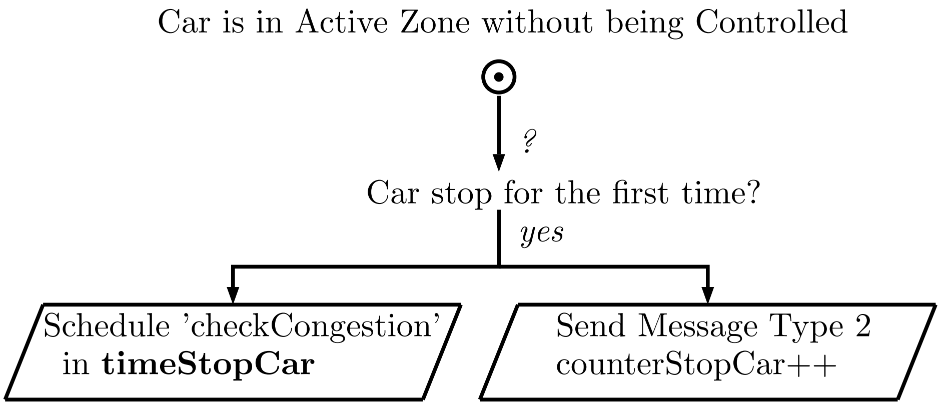
  This function is triggered with most of the Message Type 1.

4.5 Workflow

Here will be explained with states diagram the different parts of the code already discussed.

4.5.1 Initialization of the Control Algorithm

How it was described in 4.2.3, for initializing the control algorithm, first the control enter in an evaluation phase like it is show in Figure 4 where the car start the two evaluation rules.

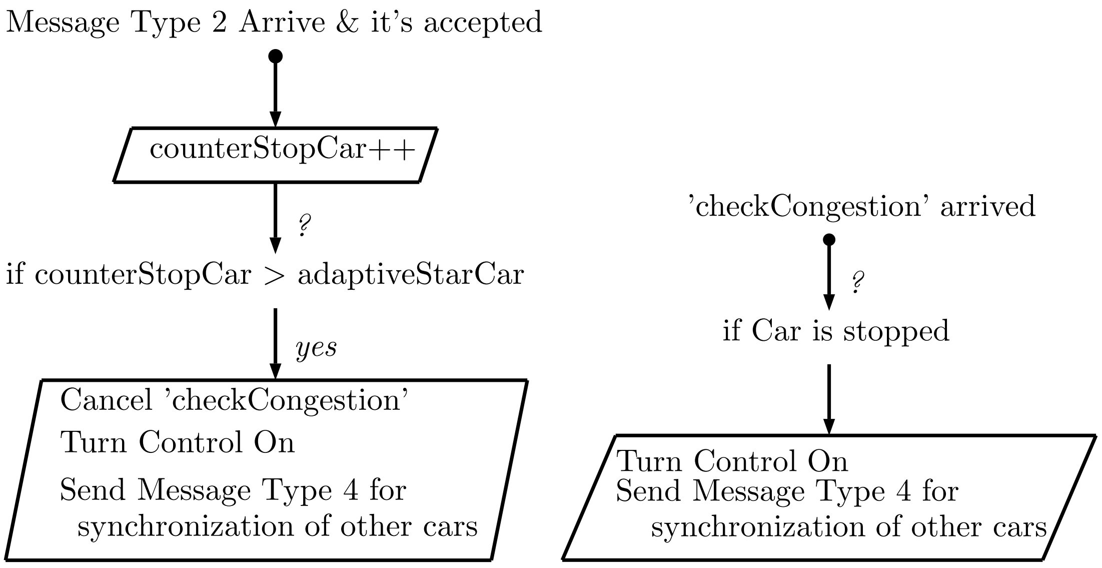


Car is in Active Zone without being Controlled
?
Car stop for the first time?
yes
Schedule 'checkCongestion' in timeStopCar
Send Message Type 2 counterStopCar++

Figure 4: Evaluation enter to activate control algorithm

From here, for activating the control, there are two possible ways that this can be done, like it is shown in Figure 5, where the empiric rule N°1 has a greater priority than the rule N°2, only by the account that when the first rule apply, cancel the second one.

This also imply that the maximum time that the car will be stopped during the control algorithm, before it resume going again will be **tYellow** + **timeStopCar**



Message Type 2 Arrive & it's accepted
counterStopCar++
?
if counterStopCar > adaptiveStarCar
yes
Cancel 'checkCongestion'
Turn Control On
Send Message Type 4 for synchronization of other cars
'checkCongestion' arrived
?
if Car is stopped
Turn Control On
Send Message Type 4 for synchronization of other cars

(a) Empiric Rule N°1 (b) Empiric Rule N°2

Figure 5: Empiric Rules for activating Control

4.5.2 State of the Car

Every a certain amount of time, the car check its position on the map, and depending of this and other boolean parameters that modify their value outside of the iteration of **stateControl()**, it can send a signal for synchronising to the control algorithm if it is running, activate the evaluation phase described before, or simple schedule the next check of position, like it is shown on Figure 6.

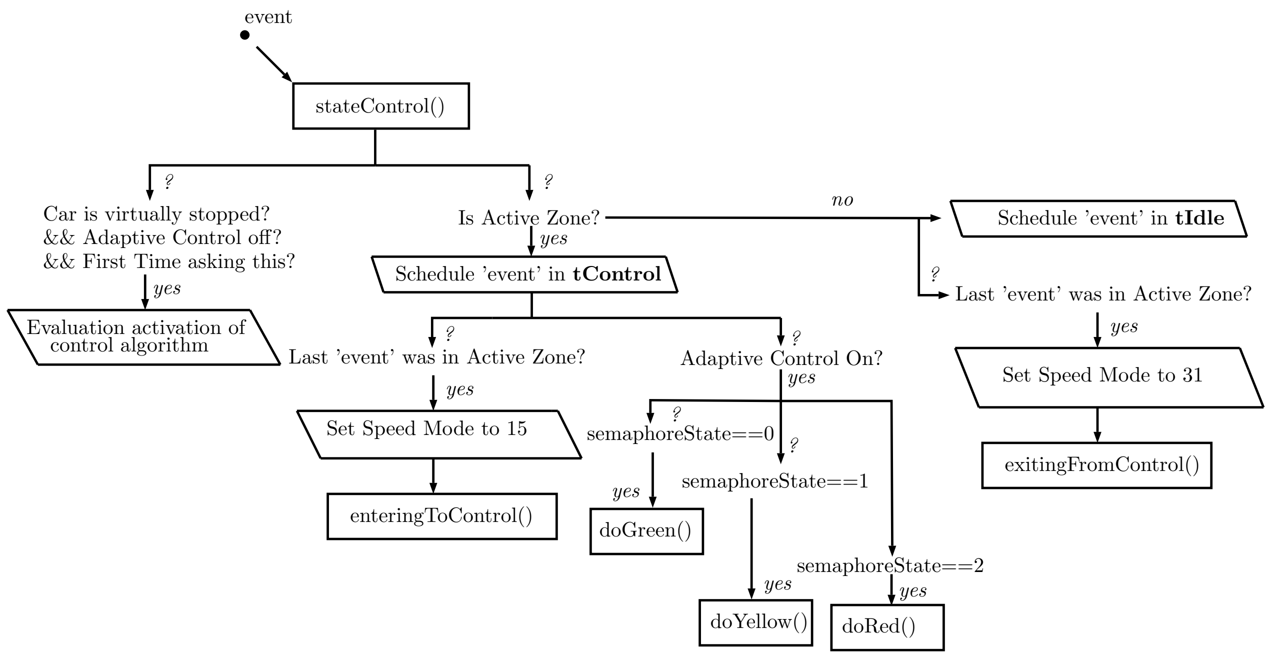


Figure 6: stateControl() Function

How it can be appreciated, this function call itself every **tControl** or **tIdle** depending of their position on the map. The only exception is when the car is initialized. When this happen, the car call the message 'event' at the same time. This could be interpreted like that the car schedule this message when the car start.

4.5.3 Control of States

The control algorithm is highly dependent of three things: Activation of the control, synchronization of the control, and maintaining the control until the end of itself or exiting the Active Zone.
Meanwhile the synchronization is a transitory state, the maintaining could be considered the normal state. While the control is active and in normal operation, the message checkState is schedule, and this call change State. This dynamic can be appreciated in Figure 7 and 8.

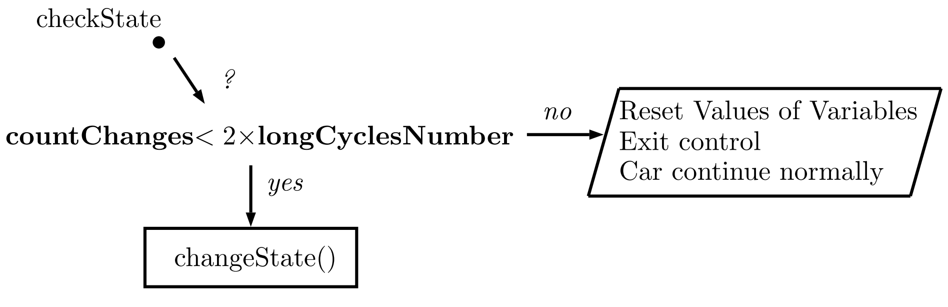
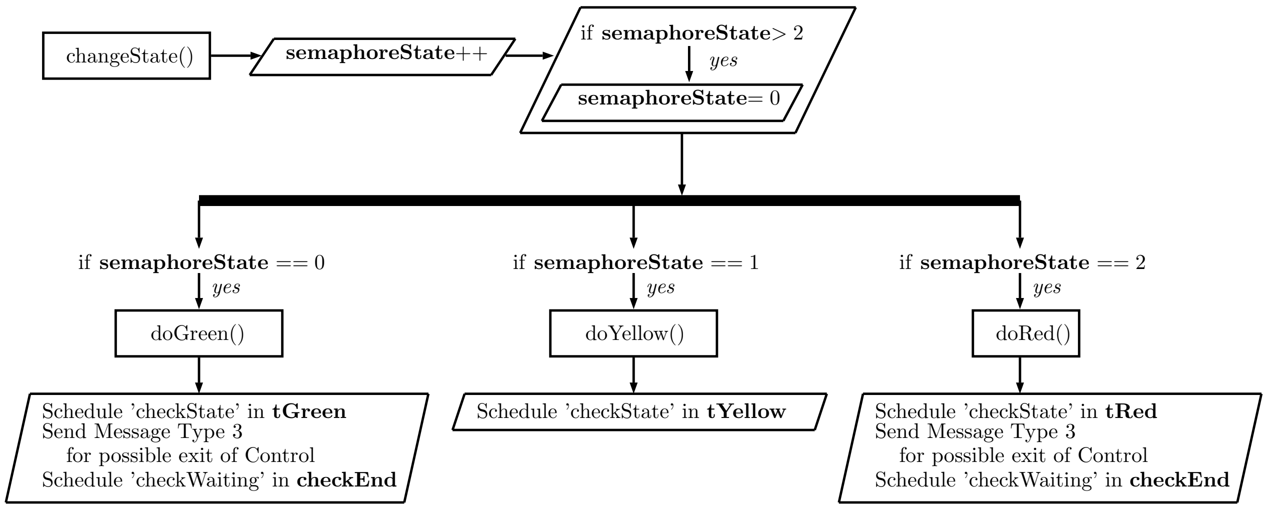


Figure 7: checkState Call



changeState()
semaphoreState++
if semaphoreState> 2
yes
semaphoreState= 0
if semaphoreState == 0
yes
doGreen()
Schedule 'checkState' in tGreen
Send Message Type 3
for possible exit of Control
Schedule 'checkWaiting' in checkEnd
if semaphoreState == 1
yes
doYellow()
Schedule 'checkState' in tYellow
if semaphoreState == 2
yes
doRed()
Schedule 'checkState' in tRed
Send Message Type 3
for possible exit of Control
Schedule 'checkWaiting' in checkEnd

Figure 8: changeState() Function

After it has elapsed 'checkWaiting', it is activated the routine for seeing if the control exit in an early manner, described in Figure 9.

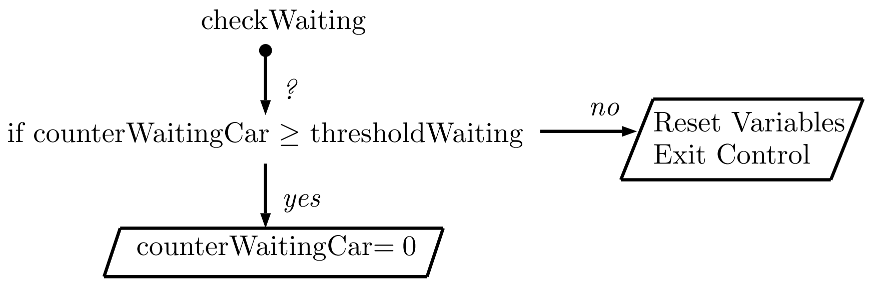
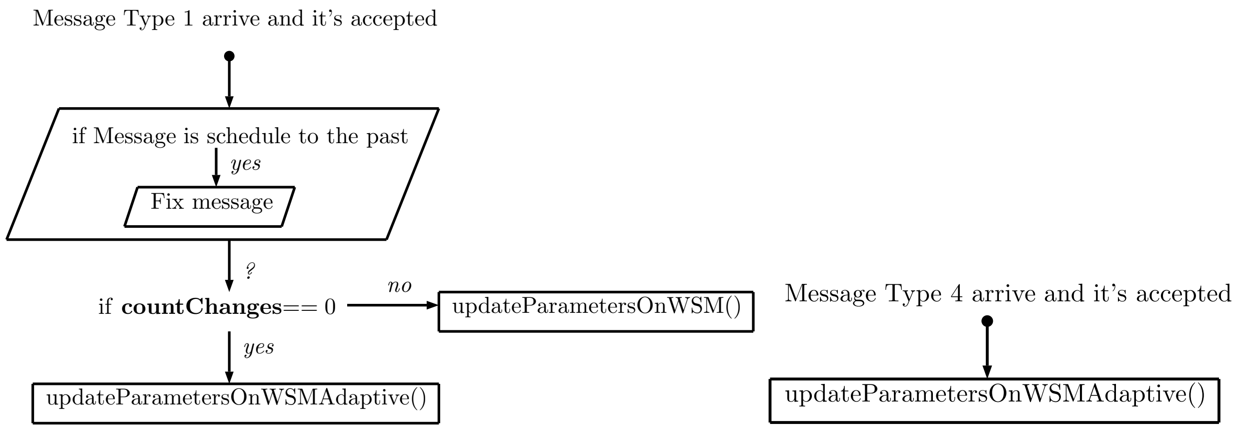


Figure 9: Finish Early Routine

The variable **counterWaitingCar** increase in one every time that the car receive a message Type 3 from a car that it is from a different road. This value is reset to 0 once the car exit the Active Zone or end the Control Algorithm.

4.5.4 Synchronization

For synchronization porpoise, the control algorithm use message Type 1 and Message Type 4. The difference between this two is that the latter is used when the car is already in the Active Zone in the moment that the control algorithm is triggered, meanwhile the former is a generic form of synchronization. The dynamic of the two message is showed on Figure 10.


Message Type 1 arrive and it's accepted
if Message is schedule to the past
yes
Fix message
?
if countChanges== 0
no
updateParametersOnWSM()
yes
updateParametersOnWSMAdaptive()
Message Type 4 arrive and it's accepted
updateParametersOnWSMAdaptive()

(a) Message Type 1

(b) Message Type 4

Figure 10: Synchronization Message Dynamic

The parameter of **countChanges** is parameter that increase in one every time that the state of the semaphore changes from Red to Green or from Yellow to Red. This serves two porpoise, it helps to know when to exit when a certain number of cycles have elapsed and that every car is in the same cycle.

The functions updateParametersOnWSMAdaptive() and updateParametersOnWSM() make sure to update the internal variables of the car in order to be synchronized in state and iteration of the control algorithm, independent of the direction from which they received the synchronization signal.

5 Tests

For realizing the different test, we will consider the base case described on [3] like if they were actual urban intersection. This means that the limit of the velocity on every lane is $14 m/s$ or $50.4 km/hr$.

The assumptions of simulation will be:

- Cars have a minimum gap of 2.5 meters.
- Cars have a length of 2.5 meters.
- Cars have a deceleration of $4.6 m/s$.
- Cars have an acceleration of $2.6 m/s$.
- Cars follow the modified Krauβ model of simulation, because of this there can't be accidents.
- Cars have an Adaptive Cruise Control incorporated that take precedence over the safe velocity of the cars, taking in account the cars in front of them.
- There aren't buildings that create conflicts with the transmitted signals.
- There aren't pedestrians.
- The parameter tIdle, that the car use for scheduling every time it checks its own state outside the Control Zone is 1 second.
- The parameter tControl, that the car use for scheduling every time it checks its own state in the Control Zone and how update its own speed depending on the state of the virtual semaphore is 1 second.
- The message that the cars send to the others cars have a delay that it is modeled with a fixed delay of 0.3s plus a delay with normal distribution with mean 0.7 and standard deviation of 0.1 seconds.

The flow of cars will be indicated in the same way that in the SUMO file, i.e.: by flow, time in which the flow start, ordered in order of appearance in the flow, number of cars in the flow, period in seconds in which every car appear, and route of travel indicated by starting direction and exiting direction.

Because the simulation it is done in SUMO, we have to take in account that the program give priority to cars coming from the north and south.

Of importance to note is that every test activate some type of control algorithm, this means that every test is subjected to congestion. Otherwise, the control is not activated and the flow of cars is optimal.

The comparison of the different tests will be with respect to their uncontrolled simulation, Showing first the uncontrolled simulation and after that, the controlled simulation.

5.1 Test 0 - Basic Base Case

The test 0, basic base case present the four lanes starting almost at the same time, and with a heavy rate of cars, but with low number. This is the basic test showing the less features of the simulation.

The flow of cars for this simulation are:

Flow	**Starting [s]**	**Number of Cars**	**Period of Cars [s]**	**Route**
1	0	25	4	N-S
2	0.5	25	5	E-W
3	1	25	5	S-N
4	1.5	25	4	W-e

Table 1: Floats

5.1.1 Uncontrolled Simulation

The results of the full simulation and of every route of the simulation can be visualized in the table 2, with the parameter of study being the total travel time of the cars.

	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	132	172.280	257	44.536	100
E-W	160	207.280	257	29.573	29.573
N-S	132	132	132	0	25
W-E	183	217.840	256	22.330	25
S-N	132	132	132	0	25

Table 2: Data Test 0 with Control Off

With the data of the entire simulation and every lane, we can proceed to see the histogram of distribution of the simulation for each case:

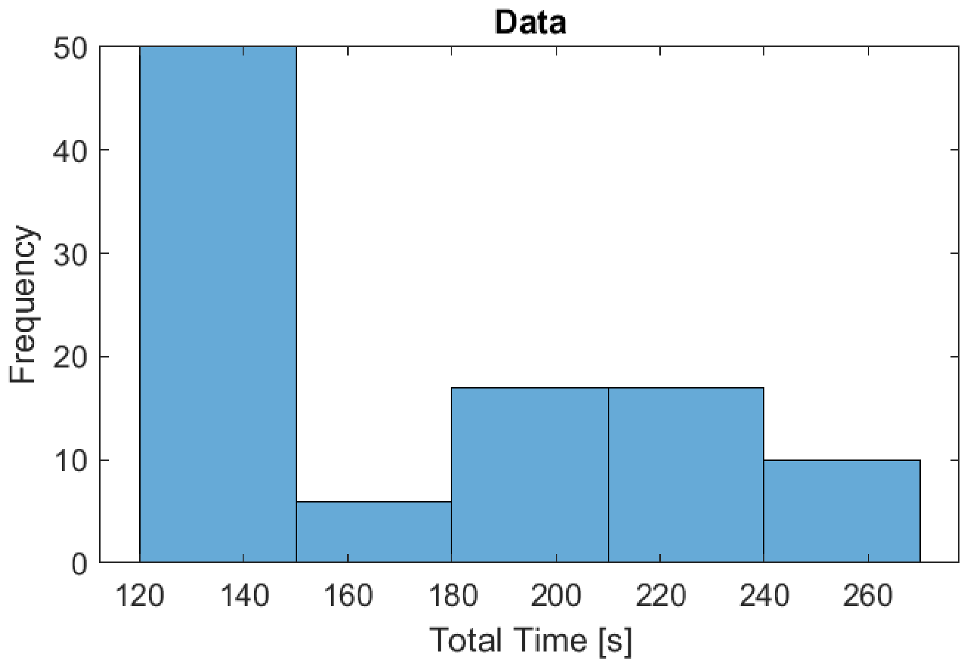
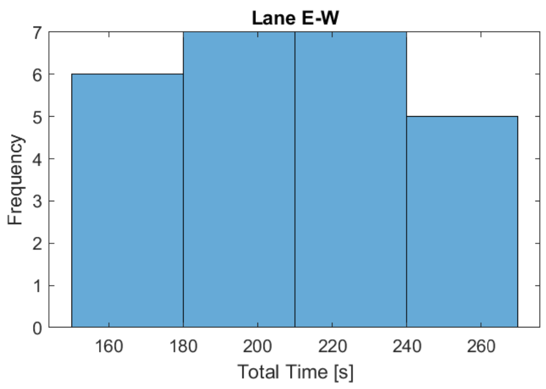
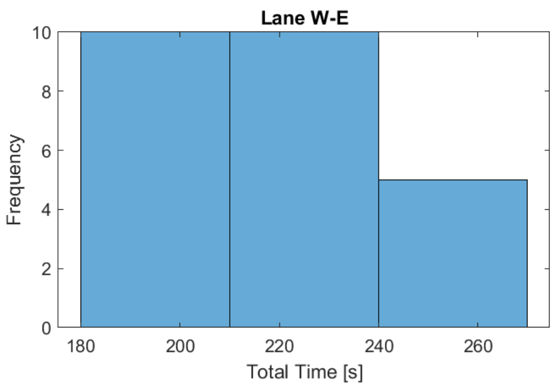


Figure 11: Histogram Test 0

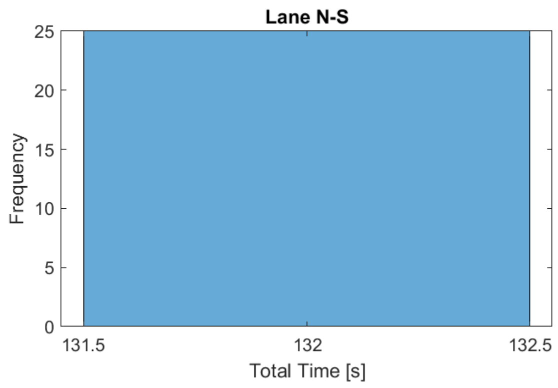


(a) Histogram E-W Direction

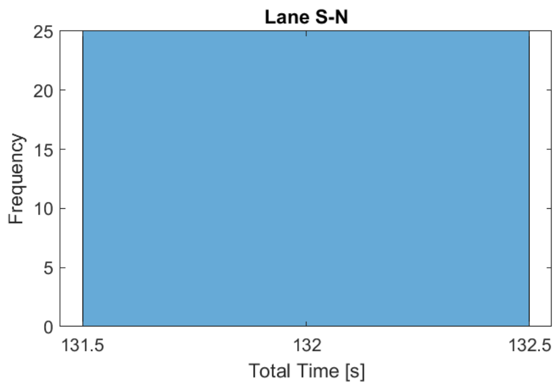


(b) Histogram W-E Direction

Figure 12: Histogram Horizontal Travel



(a) Histogram N-S Direction



(b) Histogram S-N Direction

Figure 13: Histogram Vertical Travel

How it can be appreciated in the histograms, the cars that are going from Nort to South and vice versa never stop, because internally they have better priority. The other 2 flows are stopped until the two previous flow pass the junction, and from then it depends since which time they enter the junction, and the queu of the line.

5.1.2 Controlled Simulation

The results of the full simulation and of every route of the simulation can be visualized in the table 3, with the parameter of study being the total travel time of the cars.

	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	132	142.730	159	9.046	100
E-W	133	145.880	158	9.248	25
N-S	132	142.240	159	9.462	25
W-E	133	144.840	157	8.494	25
S-N	132	144.960	157	9.208	25

Table 3: Data Test 0 with Control On

With the data of the entire simulation and every lane, we can proceed to see the histogram of distribution of the simulation for each case:

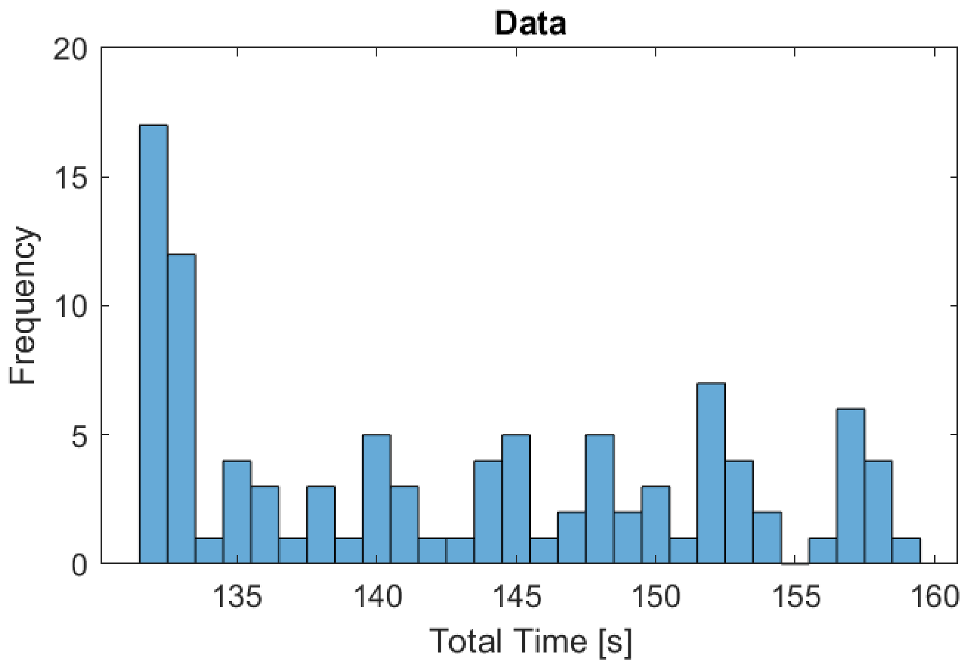
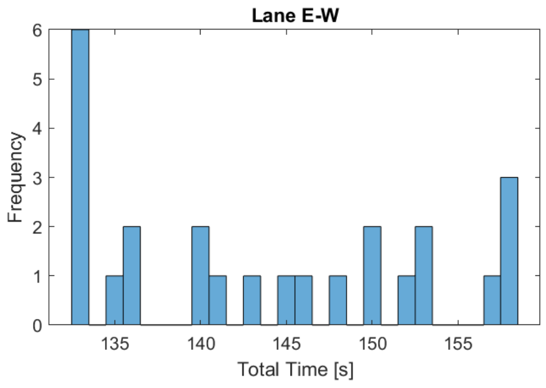
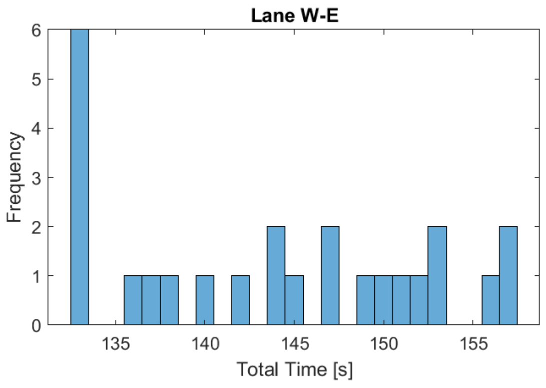


Figure 14: Histogram Test 0

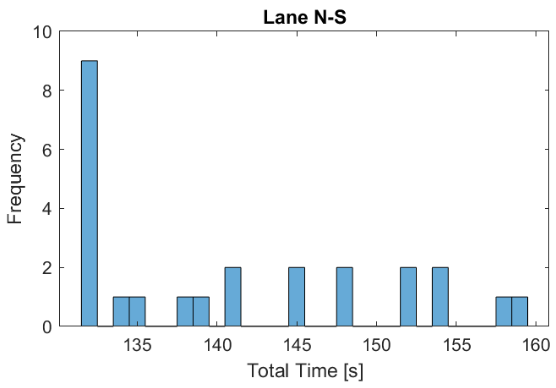


(a) Histogram E-W Direction

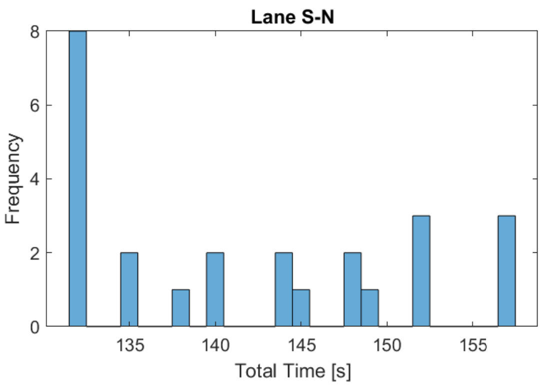


(b) Histogram W-E Direction

Figure 15: Histogram Horizontal Travel



(a) Histogram N-S Direction



(b) Histogram S-N Direction

Figure 16: Histogram Vertical Travel

The histograms show that the four flow show a similar comportment, where each of the flow has a similar mean, and similar standard deviation of the total travel time, from which it can be seen that the control help decrease the total travel time of Flow 2 and 4, and doesn't increase greatly the travel time of the other two flows.

Of interest is that the travel time for any situation never surpass 160 seconds, meaning that the maximum waiting time of any car with the parameter selected increase in at most 30 seconds aproximately.

5.2 Test 1 - Extended Base Case with heavy and intermittent flow

This test case it is an extension of Test 0, where now we have the same idea, incident flows that come at almost the same time and with similar rate generation of car. The different is that some flows can turn right, there are 16 flows separated in groups of four, making this a very heavy congested intersection.

The flow of cars for this simulation are:

Flow	**Starting [s]**	**Number of Cars**	**Period of Cars [s]**	**Route**
1	0	10	5	E-W
2	0.5	10	5	N-S
3	1	10	6	W-E
4	1.5	10	6	S-N
5	10	10	6	E-N
6	11	10	6	N-W
7	12	10	5	W-S
8	13	10	5	S-E
9	60	25	6	E-W
10	60.5	25	6	N-S
11	61	25	6	W-E
12	61.5	25	6	S-N
13	70	15	5	E-N
14	71	15	5	N-W
15	72	15	5	W-S
16	73	15	5	S-E

Table 4: Floats

5.2.1 Uncontrolled Simulation

The results of the full simulation and of every route of the simulation can be visualized in the table 5, with the parameter of study being the total travel time of the cars.

	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	131	204.95	342	77.251	240
E-W	196	270	342	42.988	35
E-N	253	289.760	331	24.054	25
N-S	132	132.886	133	0.323	35
N-W	131	131.760	132	0.436	25
W-E	195	268.886	340	41.781	35
W-S	250	287.840	330	25.519	25
S-N	132	132.857	133	0.355	35
S-E	131	131.680	132	0.476	25

Table 5: Data Test 1

With the data of the entire simulation and every lane, we can proceed to see the histogram of distribution of the simulation for each case:

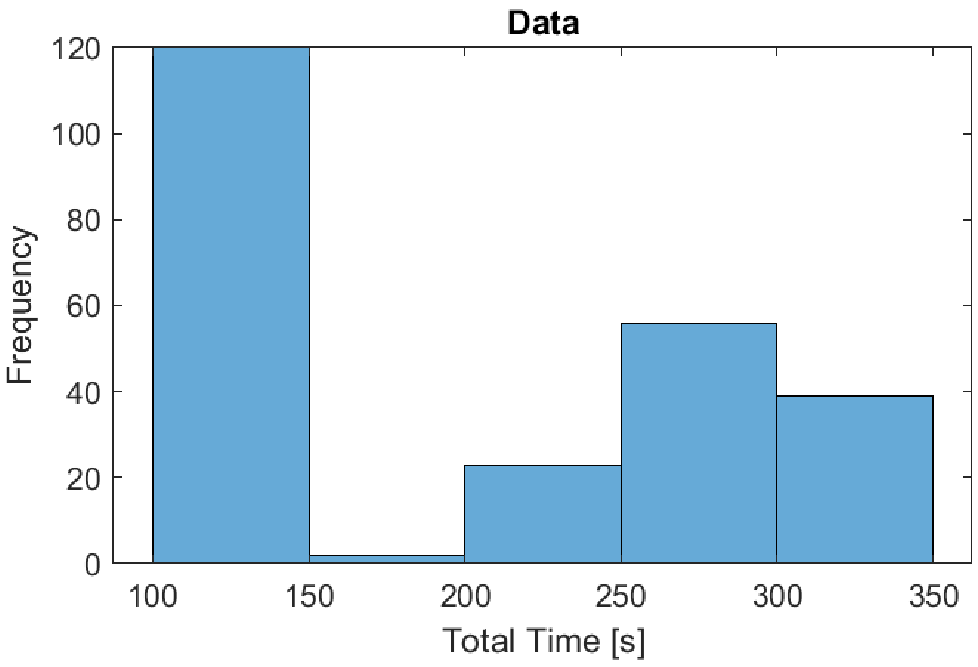
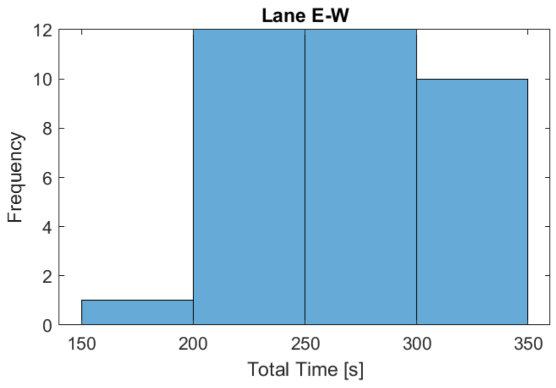
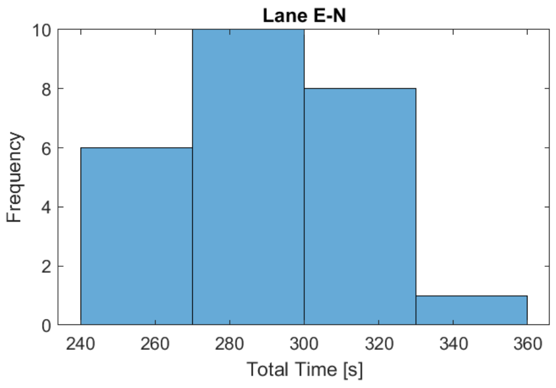


Figure 17: Histogram Test 1 with Control Off

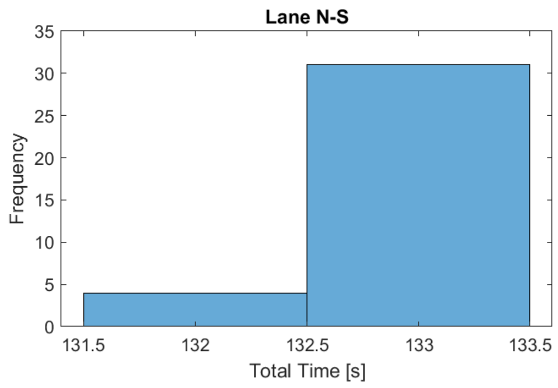


(a) Histogram N-S Direction

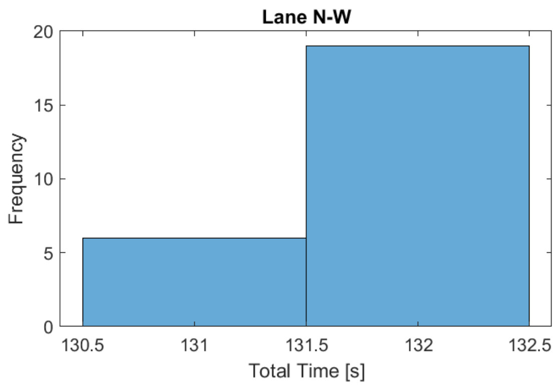


(b) Histogram N-W Direction

Figure 18: Histogram Starting from East

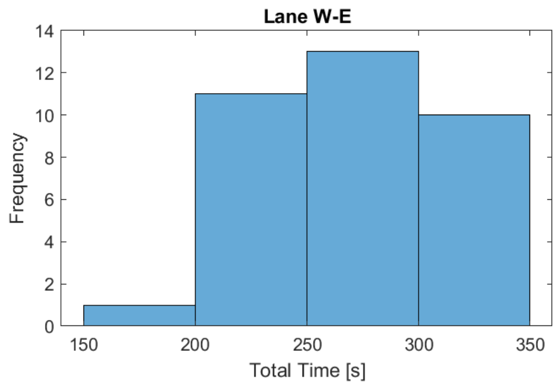


(a) Histogram N-S Direction

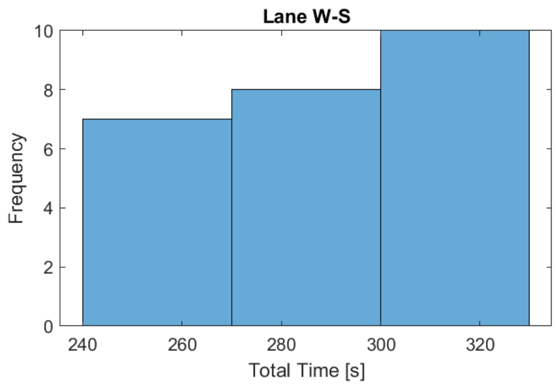


(b) Histogram N-W Direction

Figure 19: Histogram Starting from North

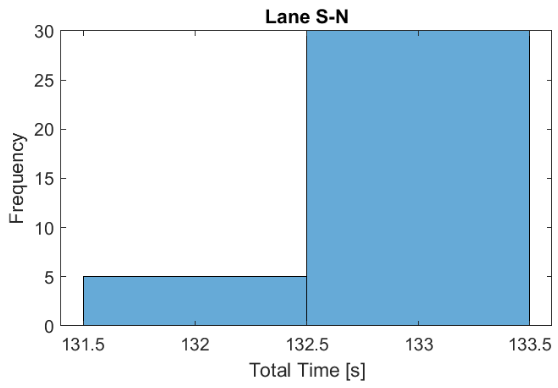


(a) Histogram W-E Direction

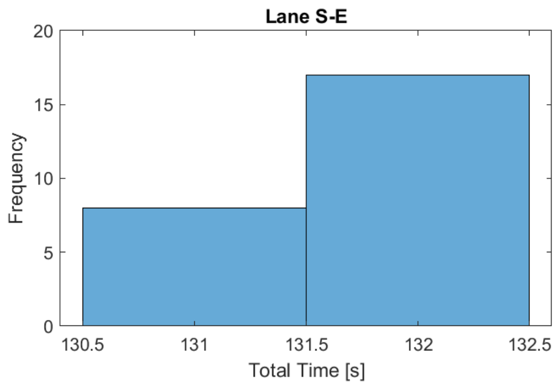


(b) Histogram W-S Direction

Figure 20: Histogram Starting from East



(a) Histogram S-N Direction



(b) Histogram S-E Direction

Figure 21: Histogram Starting from South

The histograms shows that the flows that are originated from Noth and South direction have a better comportment, with a standard deviation that can be omitted (inferior to 0.5). Instead, the other flows have a noticeable worst time compared to test 0. It is shown, that like expected, with an increase in cars and thus, in the level of congestion, the intersection perform worst in an uncontrolled situation.

5.2.2 Controlled Simulation

The results of the full simulation and of every route can be visualized in the table 6, with the parameter of study being the total travel time of the cars.

	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	131	144.021	161	9.041	240
E-W	132	144.229	160	9.343	35
E-N	131	145.600	159	8.529	25
N-S	133	143.029	160	8.652	35
N-W	132	143.6	159	9.367	25
W-E	132	142.8	160	9.132	35
W-S	131	145.000	159	9.170	25
S-N	131	142.914	159	8.621	35
S-E	131	146.240	161	10.097	25

Table 6: Data Test 1

With the data of the entire simulation and every lane, we can proceed to see the histogram of distribution of the simulation for each case:

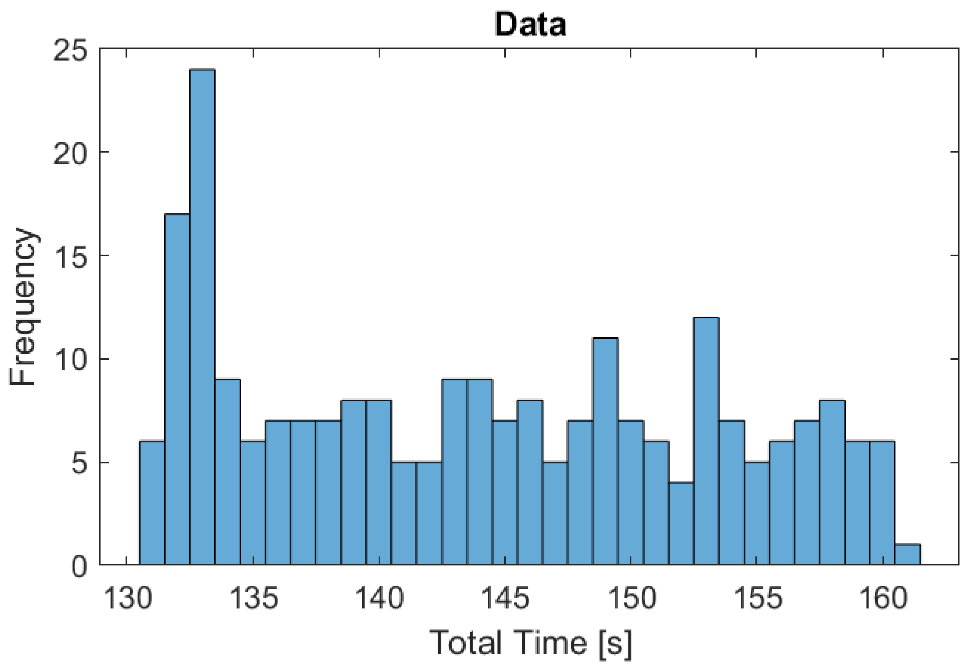
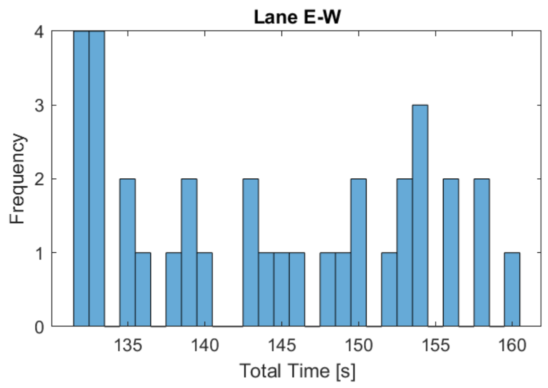
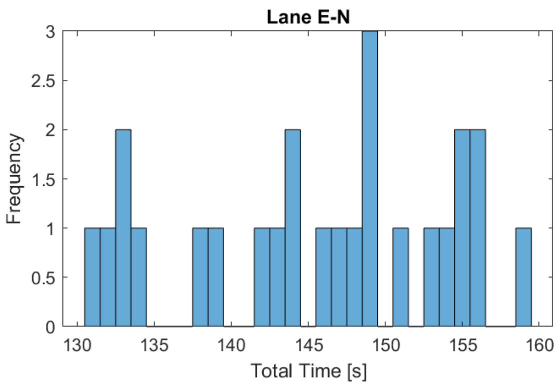


Figure 22: Histogram Test 1 with Control On

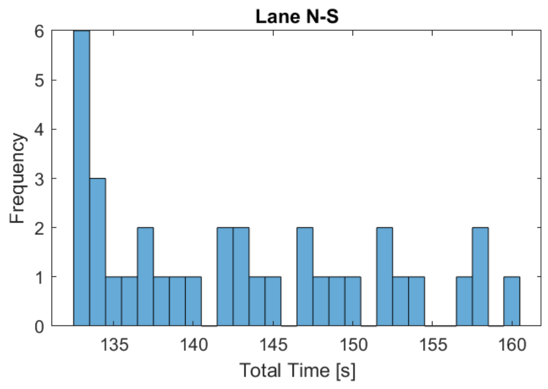


(a) Histogram E-W Direction

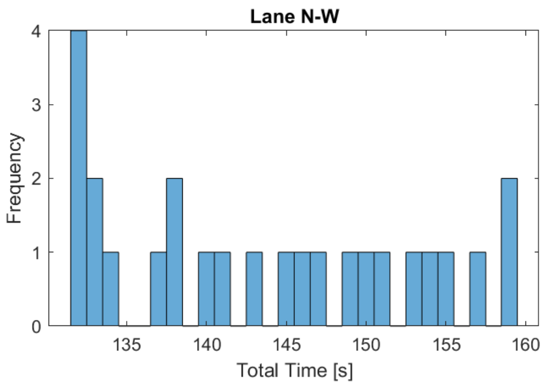


(b) Histogram E-N Direction

Figure 23: Histogram Starting from East

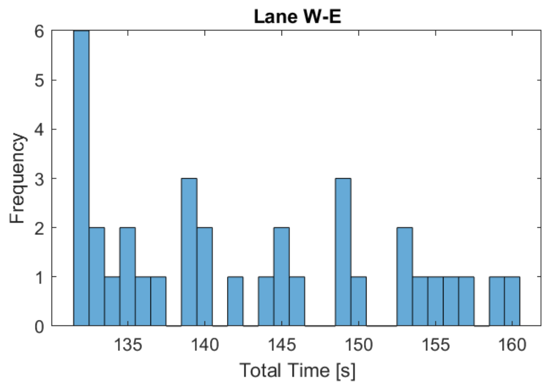


(a) Histogram N-S Direction

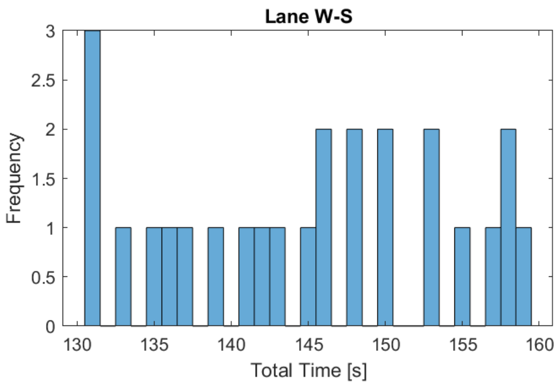


(b) Histogram N-W Direction

Figure 24: Histogram Starting from North

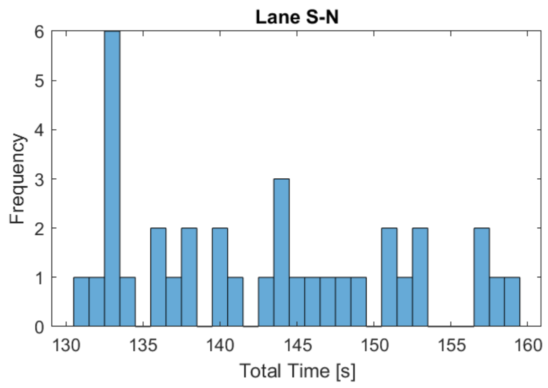


(a) Histogram W-E Direction

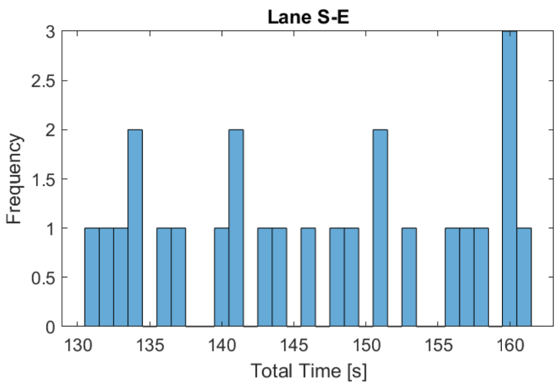


(b) Histogram W-S Direction

Figure 25: Histogram Starting from East



(a) Histogram S-N Direction



(b) Histogram S-E Direction

Figure 26: Histogram Starting from South

Because of the increase of car in the simulation, it can be seen with better detail that the control algorithm let every route to have a similar comportment in terms of mean total time and standard deviation. The histograms also show that the greater amount of car tends to be over the left - this means that most cars tends to have lower values of total time.

5.3 Test 2 - Heavy flow random vehicle

Test 2 is about a heavy congested intersection, but the flow of cars generated at random, with random rate generation of cars, number of cars and start. In this way, it can be seen how the control fare in situation that goes outside periodic situations.

The flow of cars for this simulation are:

Flow	Starting [s]	Number of Cars	Period of Cars [s]	Route
1	0	15	4	E-W
2	0	17	5	N-S
3	2	18	6	W-E
4	5	10	3	S-N
5	10	10	10	E-N
6	15	13	7	W-S
7	20	10	10	E-W
8	22	14	10	N-S
9	30	13	5	W-E
10	35	12	4	S-N
11	40	8	5	N-W
12	50	9	7	S-E

Table 7: Floats

5.3.1 Uncontrolled Simulation

The results of the full simulation and of every route can be visualized in the table 8, with the parameter of study being the total travel time of the cars.

	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	131	174.765	238	41.603	149
E-W	171	215.760	235	13.405	25
E-N	166	195.000	218	16.653	10
N-S	132	132.258	133	0.445	31
N-W	131	131.750	132	0.463	8
W-E	198	217.968	238	9.548	31
W-S	197	207.769	214	4.850	13
S-N	132	132.000	132	0	22
S-E	131	131.111	132	0.333	9

Table 8: Data Test 2

With the data of the entire simulation and every lane, we can proceed to see the histogram of distribution of the simulation for each case:

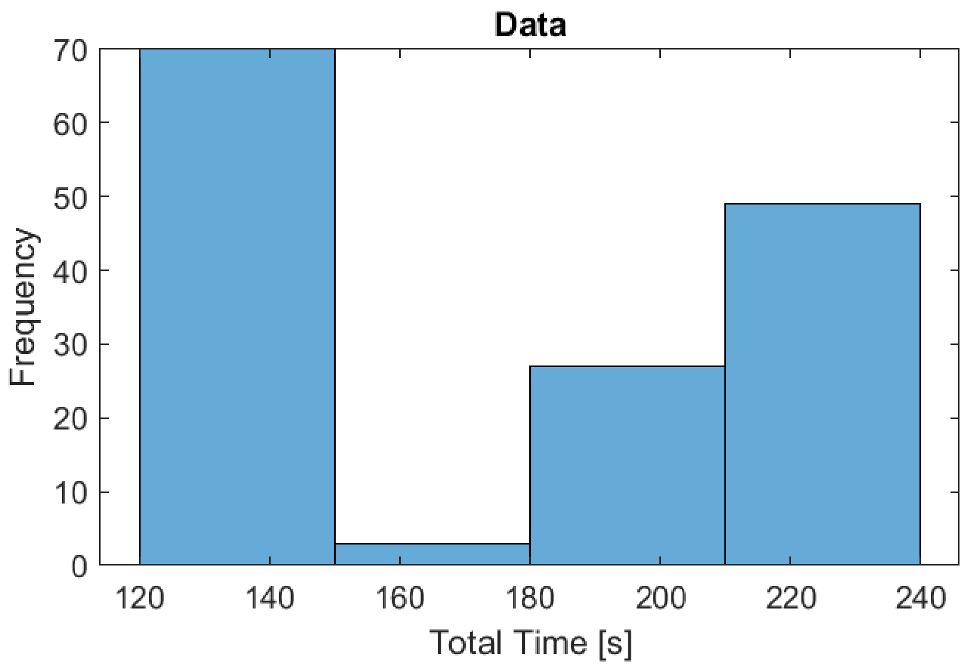
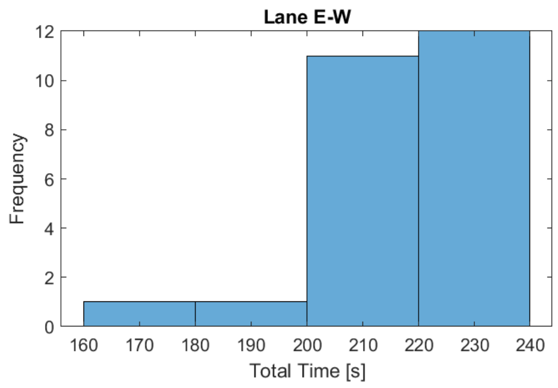
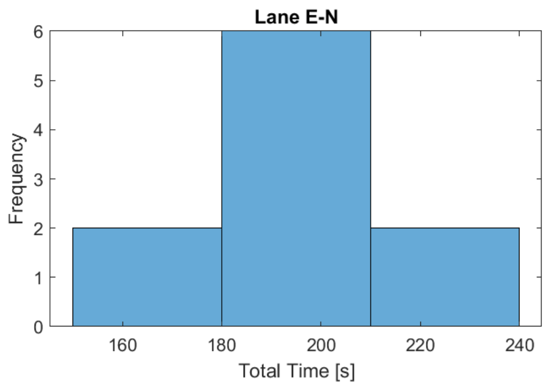


Figure 27: Histogram Test 2 with Control Off

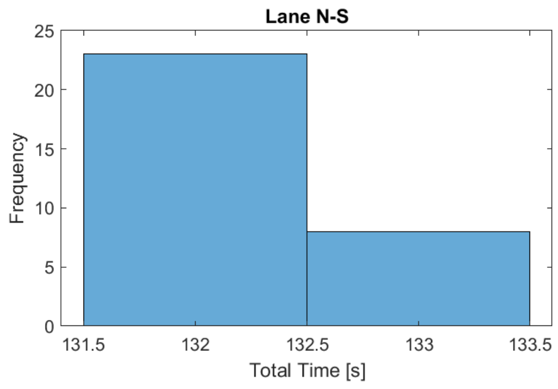


(a) Histogram E-W Direction

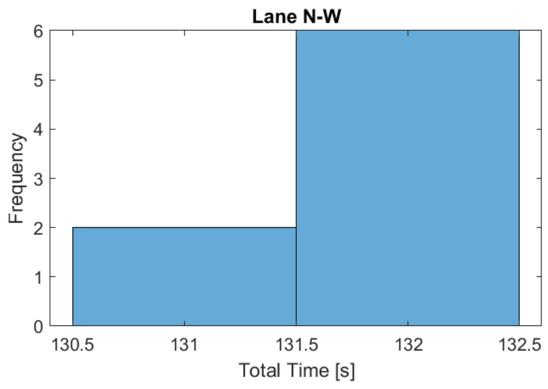


(b) Histogram E-N Direction

Figure 28: Histogram Starting from East

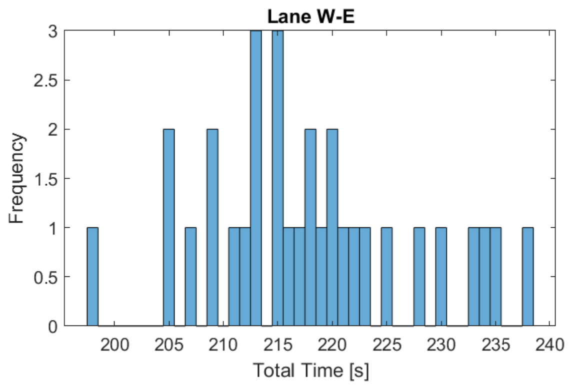


(a) Histogram N-S Direction

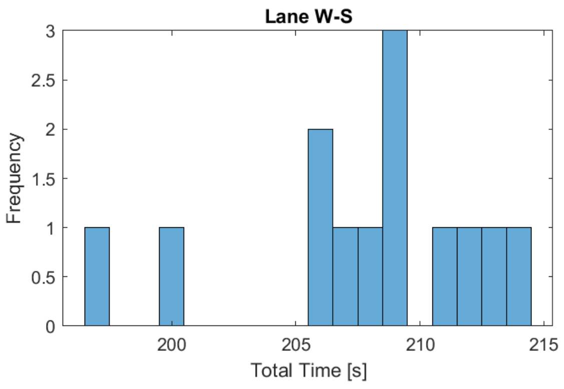


(b) Histogram N-W Direction

Figure 29: Histogram Starting from North

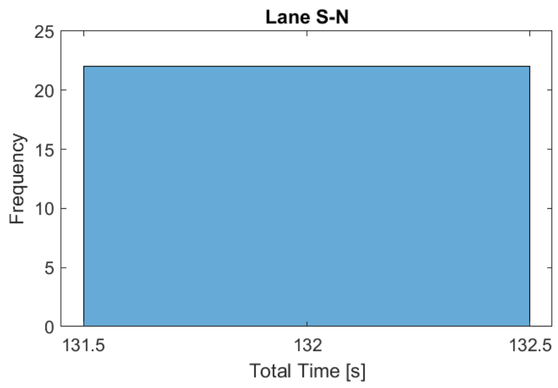


(a) Histogram W-E Direction

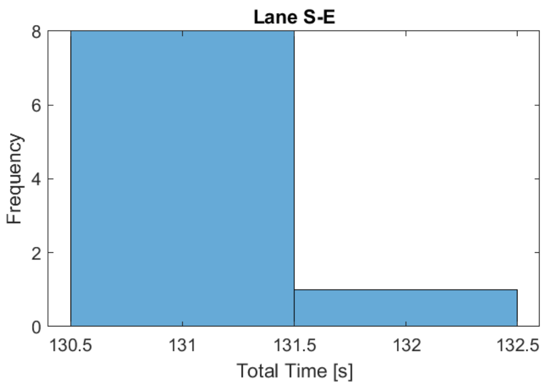


(b) Histogram W-S Direction

Figure 30: Histogram Starting from East



(a) Histogram S-N Direction



(b) Histogram S-E Direction

Figure 31: Histogram Starting from South

Like it would have been expected, routes originated from the north or the south have a total time very low, almost without variation. In a random type of simulation, other types of flow present a more distributed form, but still present a mean total time of travel in comparison with the flows originated from north or south.

5.3.2 Controlled Simulation

The results of the full simulation and of every route can be visualized in the table 9, with the parameter of study being the total travel time of the cars.

	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	131	144.114	160	8.914	149
E-W	133	145.960	160	8.914	25
E-N	132	143.300	157	8.460	10
N-S	132	142.290	159	9.060	31
N-W	132	144.375	157	9.395	8
W-E	132	144.742	160	8.489	31
W-S	131	143.462	158	9.024	13
S-N	132	144.046	159	9.609	22
S-E	131	144.889	158	10.191	9

Table 9: Data Test 2

With the data of the entire simulation and every lane, we can proceed to see the histogram of distribution of the simulation for each case:

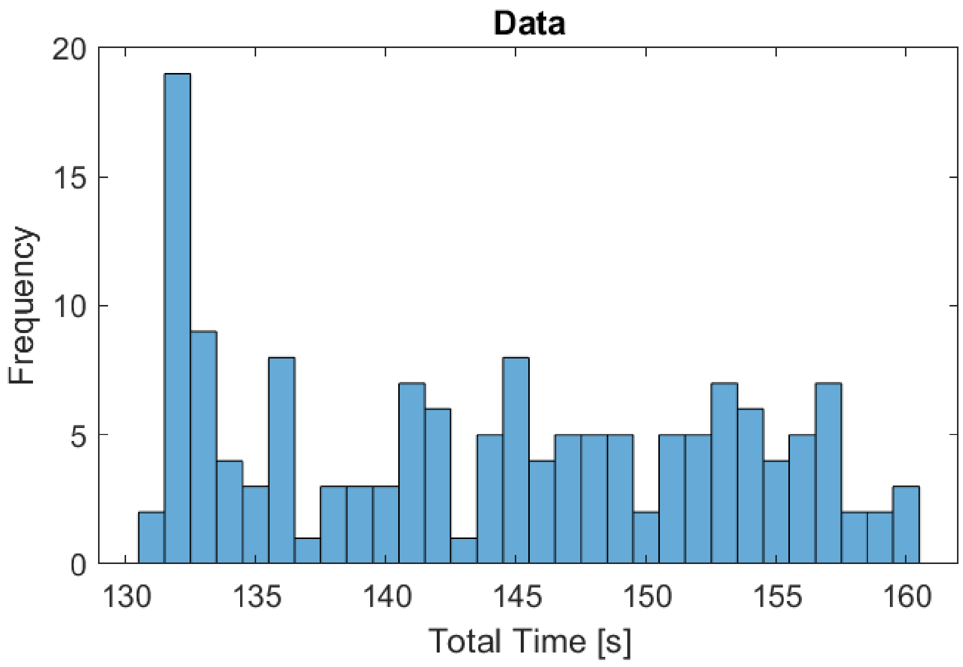
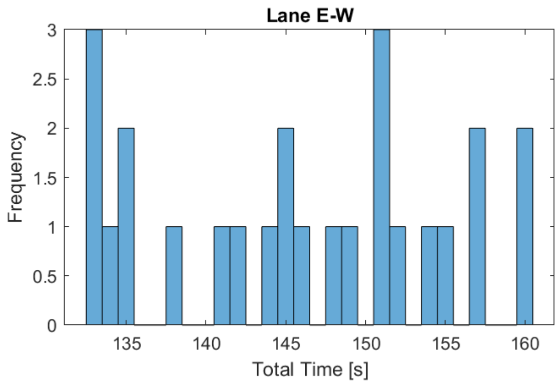
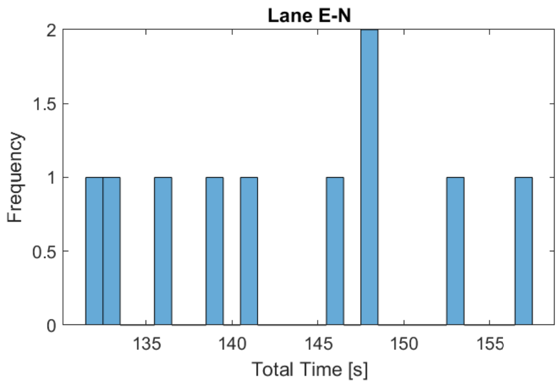


Figure 32: Histogram Test 2 with Control On

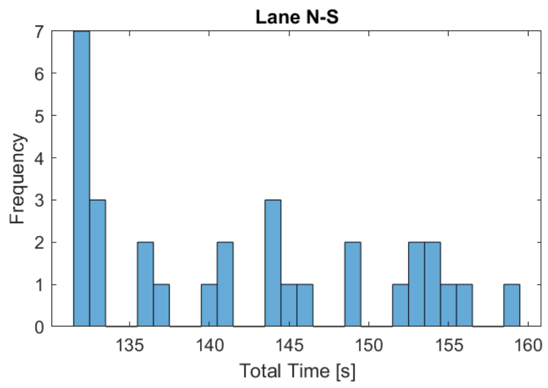


(a) Histogram E-W Direction

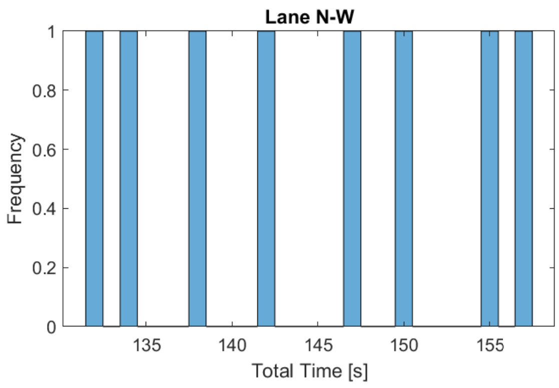


(b) Histogram E-N Direction

Figure 33: Histogram Starting from East

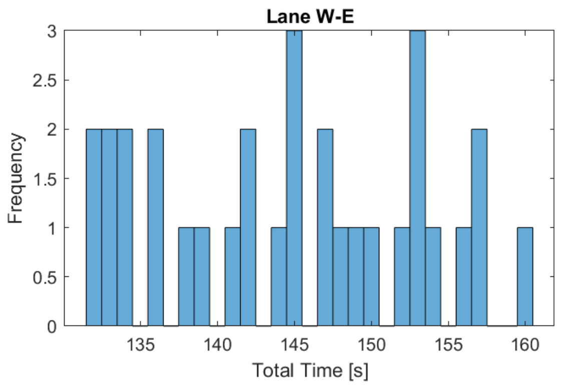


(a) Histogram N-S Direction

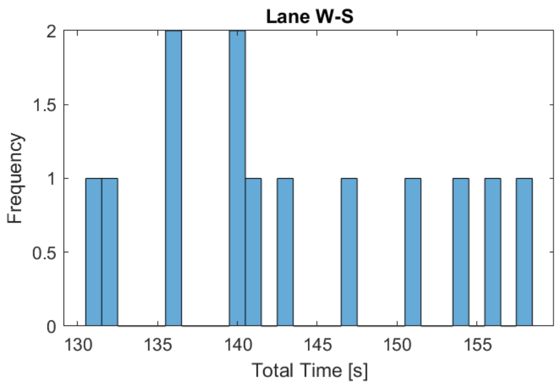


(b) Histogram N-W Direction

Figure 34: Histogram Starting from North

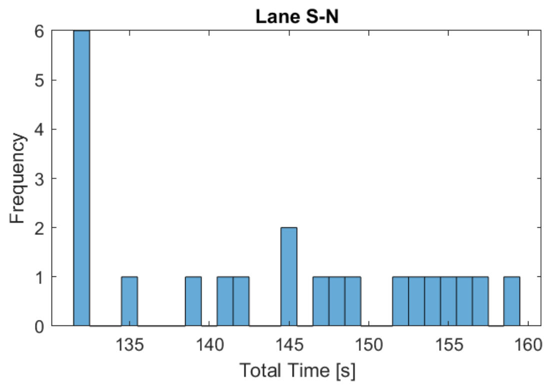


(a) Histogram W-E Direction

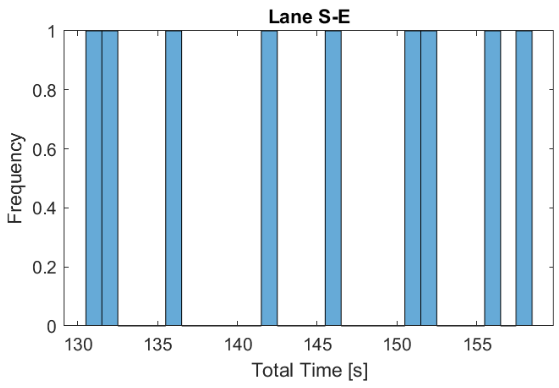


(b) Histogram W-S Direction

Figure 35: Histogram Starting from East



(a) Histogram S-N Direction



(b) Histogram S-E Direction

Figure 36: Histogram Starting from South

In the random case scenario, we have to filter the result data of the routes where there are a considerable amount of cars and the one where there aren't. In the flow with over 20 cars, we can see that the comportment it's similar to the test 0 and 1, with a similar form in the histogram. In the flow with lower amount the car, the histogram show a distributed form across the different total travel time, but even in this case, we can observe that the maximum travel time never exceed 160 seconds. What's more, seeing the results of the data, we can see that even with lower amount of car, the mean total travel time and the standard deviation show a similar comportment to the other case. Thus the control has a similar comportment indiferent of the density of the flow of cars.

5.4 Test 3 - Early finish in a heavy congested lane in only one direction

In test 3, there are 2 heavy flow in an uncongested situation, flow N-S and S-N. There is an incident in which arrive two flow with different arriving time, flow from E-W and flow W-S. we can see that because of the early activation the cars that are in the heavy flow can exit from the control once the incident car exit the active zone.

The flow of cars for this simulation are:

Flow	**Starting [s]**	**Number of Cars**	**Period of Cars [s]**	**Route**
1	0	25	5	N-S
2	0	25	5	S-N
3	20	25	10	E-W
4	30	25	8	W-S

Table 10: Floats

5.4.1 Uncontrolled Simulation

The results of the full simulation and of every route with early finish can be visualized in the table 11

	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	132	149.079	261	36.740	76
E-W	195	228.250	261	22.814	8
N-S	132	132	132	0	30
W-S	135	198	235	39.283	8
S-N	132	132	132	0	30

Table 11: Data Test 3 with Early Finish

With the data of the entire simulation and every lane, we can proceed to see the histogram of distribution of the simulation for each case:

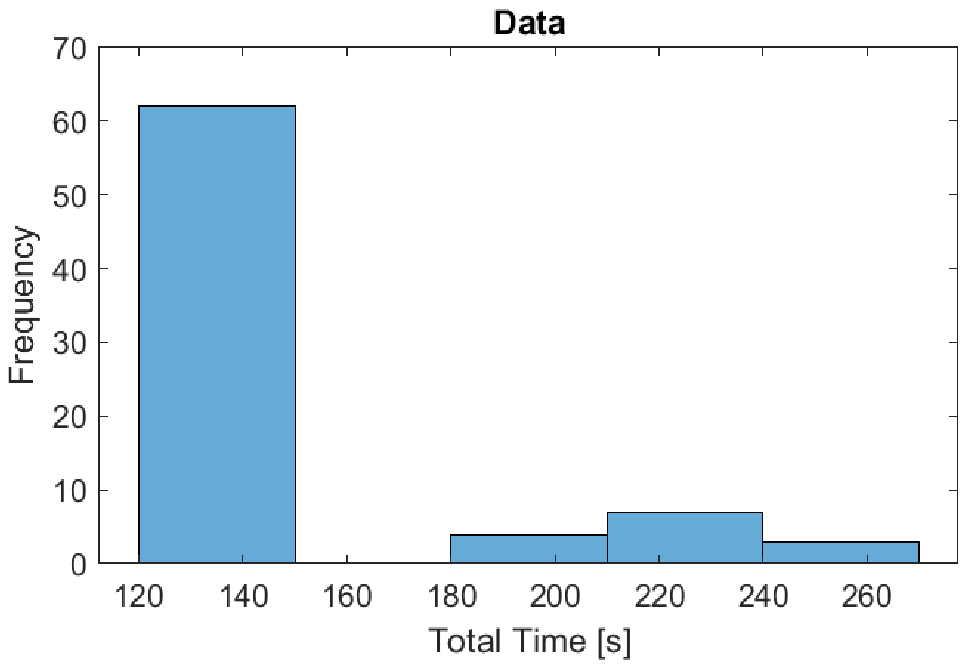
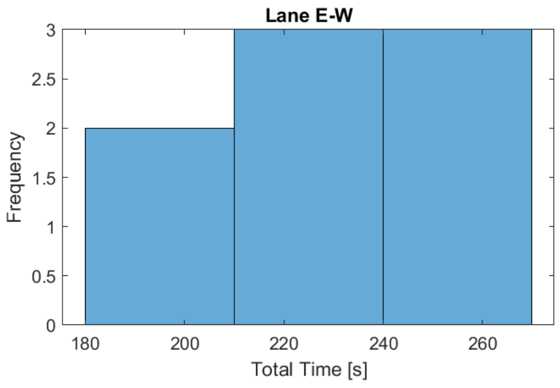
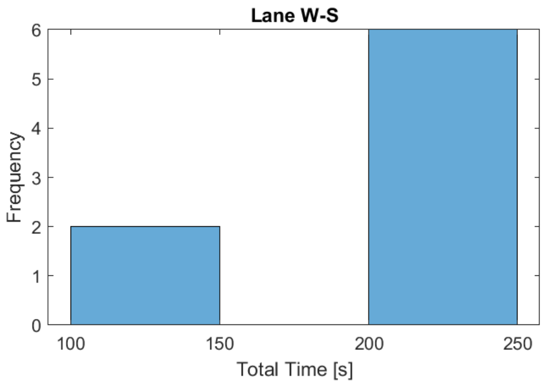


Figure 37: Histogram Test 3 With controll Off

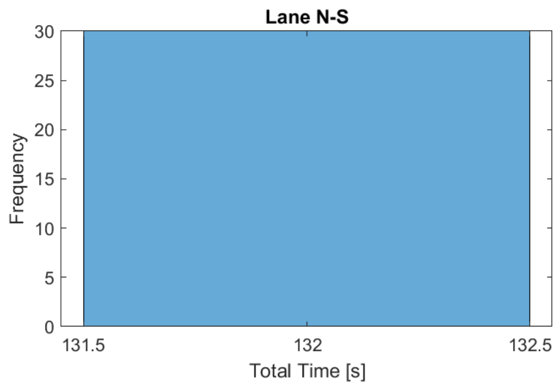


(a) Histogram E-W Direction

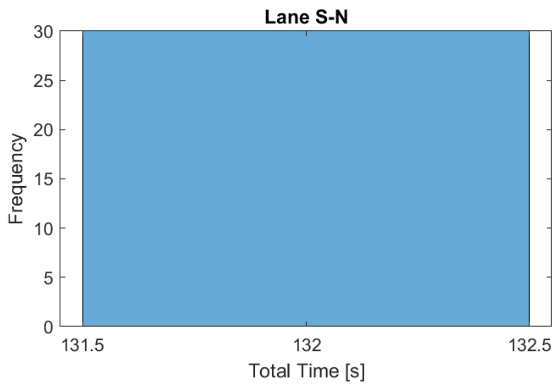


(b) Histogram W-S Direction

Figure 38: Histogram Horizontal Travel



(a) Histogram N-S Direction



(b) Histogram S-N Direction

Figure 39: Histogram Vertical Travel

Seeing Figure 37, it can be seen that there are two groups of cars, the big uninterrupted flows, and the incident flows. Nothing much to say, the situation is similar to the uncontrolled situation of the other tests.

5.4.2 Controlled Simulation with Early Finish On/Off

The results of the full simulation and of every route for the case with early finish can be visualized in the table 12, with the parameter of study being the total travel time of the cars.

	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	131	138.276	160	8.550	76
E-W	132	142.500	160	10.351	8
N-S	132	137.233	156	8.076	30
W-S	131	141.875	159	9.628	8
S-N	132	137.233	156	8.054	30

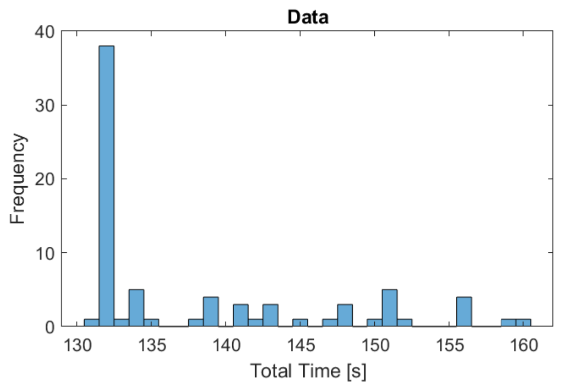
Table 12: Data Test 3 with Early Finish

The results of the full simulation and of every route for the case without early finish can be visualized in the table 13, with the parameter of study being the total travel time of the cars.

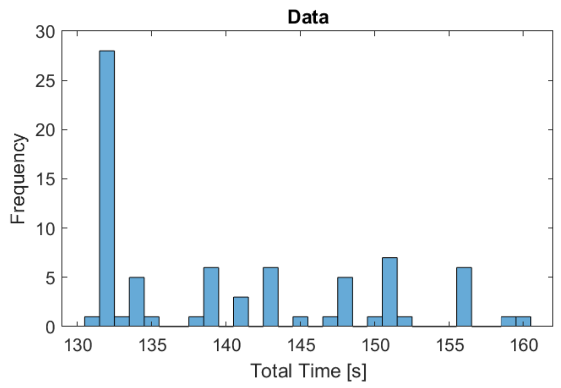
	Min Value	Mean	Max Value	Standard Deviation	# Cars
Data	131	140.316	160	8.921	76
E-W	132	142.500	160	10.351	8
N-S	132	139.833	156	8.820	30
W-S	131	141.875	159	9.628	8
S-N	132	139.800	156	8.790	30

Table 13: Data Test 3 without Early Finish

With the data of the entire simulation and every lane, we can proceed to see the histogram of distribution of the simulation for each case:

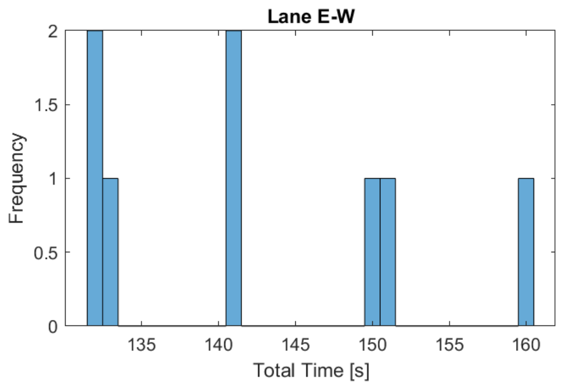


(a) Histogram Data with Early Finish

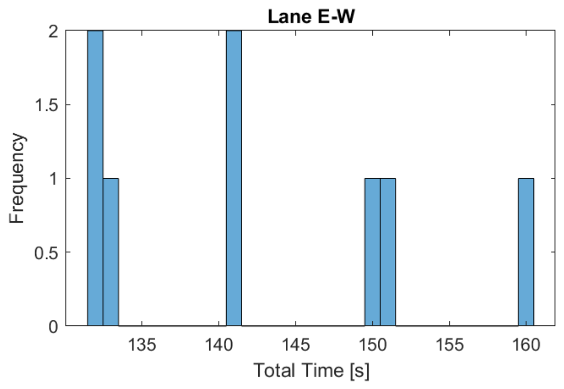


(b) Histogram Data without Early Finish

Figure 40: Histogram of Test 3 With Control On

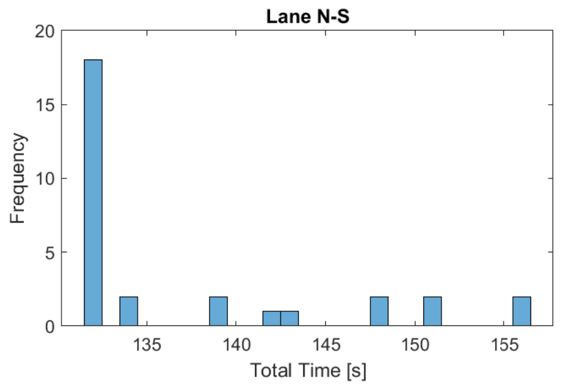


(a) Histogram with Early Finish

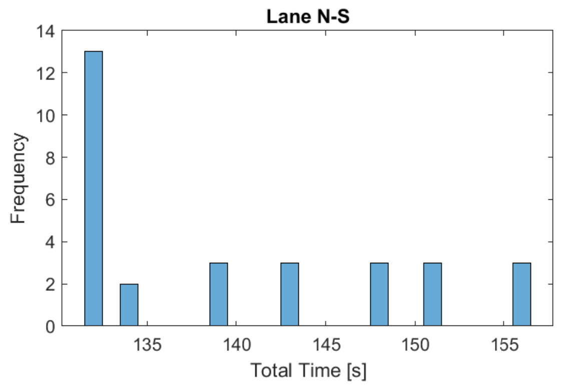


(b) Histogram without Early Finish

Figure 41: Histogram of Direction E-W

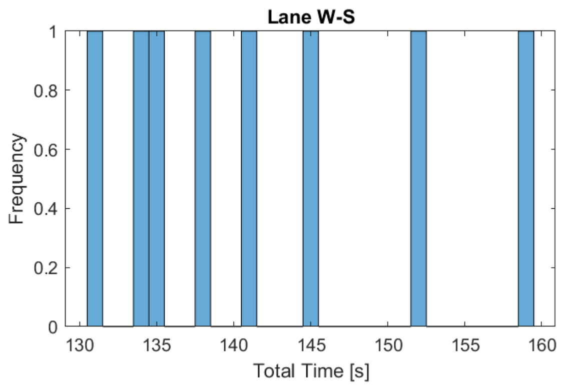


(a) Histogram with Early Finish

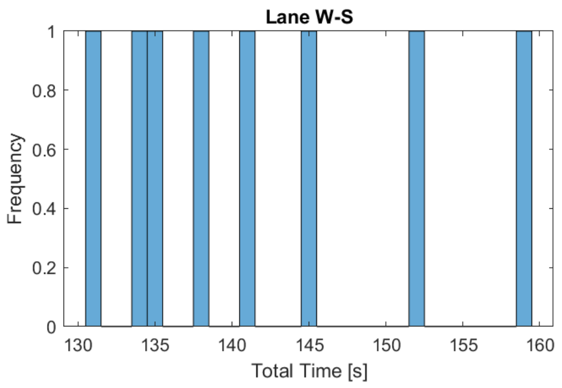


(b) Histogram without Early Finish

Figure 42: Histogram of Direction N-S

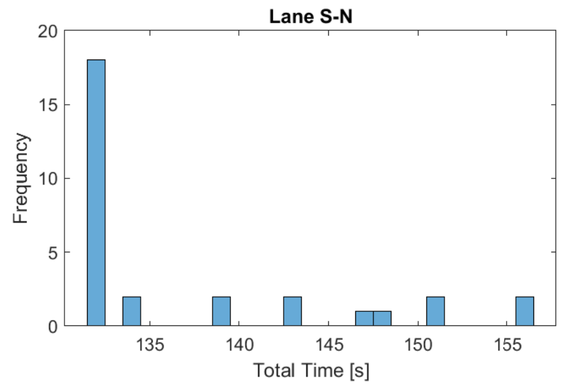


(a) Histogram with Early Finish

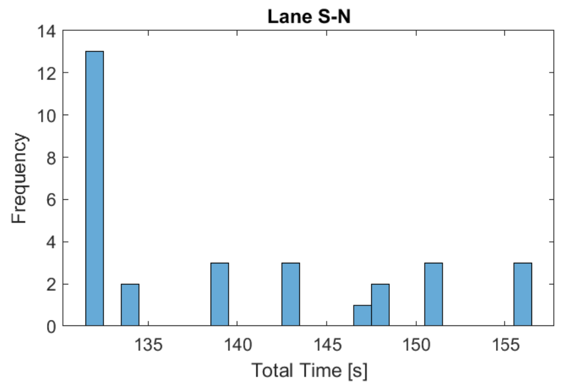


(b) Histogram without Early Finish

Figure 43: Histogram of Direction W-S



(a) Histogram with Early Finish



(b) Histogram without Early Finish

Figure 44: Histogram of Direction S-N

For this experiment in particular, there are two situations of control, the early finish on and off. The reason of why this happen is that when the incident flow of cars disappear, the car that are still in the Active Zone don't have to be in a control that it isn't doing its work.

The control even in situation of incident flows shows that achieve a distribution of the total travel time centered around a similar value for any route. Not only that, the normal distribution of the histogram of heavy flow are skewed towards the left, meaning that the greater amount of cars tend to have a reduced time travel. Even in the cars that have a greater travel time, the total travel time don't exceed 160 seconds, showing that the control, with this parameters, is very reliable in predicting a total travel time.

In the case of the cars from the incident flow, they are evenly distributed, but more centered towards the left. This is because the reduced amount of cars doesn't let to see a significant results, but by the data amount, it can be seen that the control works like intended.

6 Analysis

In section 5 it was shown different tests with the control turned ON and OFF, and a little interpretation of what happen in every case. Now it will be shown the effects of the control over the different test, taking in account the average time of travel in the simulation for the different routes.

Because every lane has a different comportment depending on the flows of cars, the number of cars and their route of travel, the approach to take will be:

- First, for every flow in the simulation, it will be taken the variation of average total travel time of their respective route in the no control situation vs the control situation.
- Second, this variation that can be negative if the travel time decrease, or positive otherwise, will be multiplied by their weight in the simulation, defined like the number of cars in the flow divided by the total number of cars in the simulation.
- Third, the total sum of the previous valour with all the other flows in the simulation will result in a value that can be positive or negative. In this weighted sum if the value is positive, means that the control increase the total travel time, otherwise it decrease the total travel time, and there is a positive value in using it.

This approach give us an objectively way of comparing the different flows, where the flows with a greater number of car have a bigger weight in the final sum, and otherwise, incident cars have less weight.

6.1 Test 0

In test 0, there are four flows of cars, all of them with almost the same rate and start, so the test make for a good comparison for every lane with all conditions the same.

In the table 14 we can see the comparison of the mean travel time with the variation per route of travel:

	No Control	Control	Variation [%]	# Cars
Data	172.280	142.730	-20.7%	100
E-W	207.280	145.880	-42.09%	25
N-S	132	142.240	7.2%	25
W-E	217.84	144.840	-50.40%	25
S-N	132	144.960	8.94%	25

Table 14: Data Test 0

The overall performance, can be analysed in table 15:

Flow	**Route**	**Variation [%]**	**weight**	**Total**
1	N-S	7.2	1/4	1.8
2	E-W	-42.09	1/4	-10.523
3	S-N	8.94	1/4	2.235
4	W-E	-50.40	1/4	-12.6
Total				-19.088

Table 15: Results Test 0

How it can be seen in the table, the performance of the control has a positive impact in the overall simulation.

6.2 Test 1

Test 1 it is an extension over test 0, with more cars per lane, 16 different flow, with different activation time and rate time. In this case the number of cars per lane it is the same in all case, making it a heavy affluent intersection.

In the table 16 we can see the comparison of the mean travel time with the variation per route of travel:

	No Control	Control	Variation [%]	# Cars
Data	204.95	144.021	-42.31%	240
E-W	270	144.229	-87.2%	35
E-N	289.760	145.600	-99.01%	25
N-S	132.886	143.029	7.09%	35
N-W	131.760	143.600	8.25%	25
W-E	268.886	142.800	-88.30%	35
W-S	287.840	145.000	-98.51%	25
S-N	132.857	142.914	7.04%	35
S-E	131.680	146.240	-9.96%	25

Table 16: Data Test 1

The overall performance, can be analysed in table 17:

Flow	Route	Variation [%]	weight	Total
1	E-W	-87.2	10/240	-3.633
2	N-S	7.09	10/240	0.295
3	W-E	-88.3	10/240	-3.679
4	S-N	7.04	10/240	0.293
5	E-N	-99.01	10/240	-4.125
6	S-W	8.25	10/240	0.344
7	W-S	-98.51	10/240	-4.105
8	S-E	9.96	10/240	0.415
9	E-W	-87.2	25/240	-9.198
10	N-S	7.09	25/240	0.739
11	W-E	-88.3	25/240	-9.198
12	S-N	7.04	25/240	0.733
13	E-N	-99.01	15/240	-6.188
14	N-W	8.25	15/240	0.516
15	W-S	-98.51	15/240	-6.157
16	S-E	9.96	15/240	0.623
Total				-42.21

Table 17: Results Test 1

How it can be seen in the tablet he performance of the control has a positive impact in the overall simulation.

6.3 Test 2

Test 2 it is a random test with different flow, with different numbers of cars per flow, which starts at different times in the simulation.

In the table 18 we can see the comparison of the mean travel time with the variation per route of travel:

	No Control	Control	Variation [%]	# Cars
Data	174.765	144.114	-21.27%	149
E-W	215.760	145.960	-47.82%	25
E-N	195.000	143.300	-36.08%	10
N-S	132.258	142.290	7.05%	31
N-W	131.750	144.375	8.74%	8
W-E	217.968	144.742	-50.59%	31
W-S	207.769	143.462	-44.83%	13
S-N	132.000	144.046	8.36%	22
S-E	131.111	144.889	9.51%	9

Table 18: Data Test 2

The overall performance, can be analysed in table 19:

Flow	**Route**	**Variation [%]**	**weight**	**Total**
1	E-W	-47.82	15/149	-4.814
2	N-S	7.05	17/149	0.804
3	W-E	-50.59	18/149	-6.112
4	S-N	8.36	10/149	0.561
5	E-N	-36.08	10/149	-2.42
6	W-S	-44.83	13/149	-3.911
7	E-W	-47.82	10/149	-3.209
8	N-S	7.05	14/149	0.662
9	W-E	-50.59	13/149	-4.414
10	S-N	8.36	12/149	0.673
11	N-W	8.74	8/149	0.469
12	S-E	9.51	9/149	0.574
Total				-21.142

Table 19: Results Test 2

How it can be seen in the tablet he performance of the control has a positive impact in the overall simulation.

6.4 Test 3

Test 3 at different than the others test that were searching to see how the control fare in heavy congested conditions, search to see how incident cars fare in a heavy flow conditions, and how the control improves when there is the posibility of exiting early from it. Seeing the comparison of graph in 5.4.2, the Early Finish reduce the total time with respect to the case with Early Finish Off.

In any case, the analysis of the control will be done with respect to the control with Early Finish On, because it is the one with better results.

In the table 20 we can see the comparison of the mean travel time with the variation per route of travel:

	No Control	Control	Variation [%]	# Cars
Data	149.079	138.276	-7.81%	76
E-W	228.25	142.5	-60.18%	8
N-S	132	137.233	3.81%	30
W-S	198	141.875	-39.56%	8
S-N	132	137.233	3.81%	30

Table 20: Data Test 3

The overall performance, can be analysed in table 21:

Flow	**Route**	**Variation [%]**	**weight**	**Total**
1	N-S	3.81	30/76	1.504
2	S-N	3.81	30/76	1.504
3	E-W	-60.18	8/76	-6.335
4	W-S	-39.56	8/76	-4.164
Total				-7.491

Table 21: Results Test 3

How it can be seen in the tablet he performance of the control has a positive impact in the overall simulation, even if the incident flow was minimum.

7 Conclusions

In this work it is presented a framework for the correct traveling of autonomous cars meanwhile they travel in intersections, using the communication network 5G and the protocol for automobiles IEEE 802.11p, and it is further simulated and tested to see the comportment of it. The technologies of autonomous cars and the communication network are assumed to be fully implemented and this enable to center the development only on the framework.
The communication platform is given to us by another framework, Veins, which give the standards for the protocol to be used, and implement the different routines and commands for sending and receiving message between cars.

The simulation are done in conjunction with SUMO that enable the simulation of every car in a microscopic model, together with OMNET++ that enables network communication. The framework developed is developed in the latter, and eventually simulation of the cars plus the networks are also running using it.

Regarding the results, it is only presented and considered one iteration of parameter that meanwhile could be that they aren't the optimal, present good overall results. The principal characteristic of the control are that it is a decentralized scheme that doesn't use an external calculation for arbitrating the cars, with an adaptive start an exit, so it only works in critical situations of the cars, namely in congestion.

Every simulation analysed show a similar comportment, where the parameter of evaluation show that when the control is activated, there is an improvement over a situation of no arbitration. The control shows that when the number of cars per flow become higher (generally over the 20 cars), the comportment of the standard deviation and the mean of the total travel time become similar between each flow.

The weakness of the framework presented are the necessity of having certain parameters defined from before running the simulation, and that cars are stopped over a certain amount of time when the control is activated. The first is something that can be changed with an exterior message, coming from an R.S.U. This solutions contemplate that there is an R.S.U. per big zone, and every time the car enter a new zone, the points of junctions and the parameters of the particular junctions change with it, this solutions liberate of the necessity of having an R.S.U. per intersection. The second problem correspond with tuning of parameters and the necessity of quitting processing capability of the ECU of the cars.

The framework presented is a proposed solution over the problem of car travelling in junctions road, that help to mitigate the problem of congestion in urban intersection, and it is thought for the future of automobiles, particularly, for when all the cars have integrated the ability to send and received messages from other cars. More and more the different manufacturer of cars are taking hands on the new technologies, or the would be new technologies, and selecting the best one between all the different options is something that depends not only in the better specifications of the controls, but also in the availability of implementation given the resources or capabilities at hand.

References

[1] Emerj, "The Self-Driving Car Timeline Predictions from the Top 11 Global Automakers", [Online]. Avaible on https://emerj.com/ai-adoption-timelines/self-driving-car-timeline-themselves-top-11-automakers/. Accessed on October, 2019.

[2] Test Site Sweded. "DriveMe" [Online]. Avaible on https://www.testsitesweden.com/en/projects-1/driveme. Accessed on October, 2019.

[3] NHTSA, "Automated Vehicles for Safety", [Online]. Avaible on https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety. Accessed on October, 2019.

[4] S. Alireza Fayazi ; Ardalan Vahidi ; Andre Luckow. "Optimal scheduling of autonomous vehicle arrivals at intelligent intersections via MILP". IEEE, 2017.

[5] Peter Stone ; Guni Sharon ; Michael Albert ; Josiah Hanna, "Autonomous Intersection Management", [Online]. Avaible on http://www.cs.utexas.edu/~aim/. Accessed on October, 2019.

[6] Bimbraw, K. "Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology". Informatics in Control, Automation and Robotics (ICINCO), 2015 12th International Conference on (Vol. 1, pp. 191-198). IEEE.

[7] IEEE, "History of IEEE", [Online]. Available on https://www.ieee.org/about/ieee-history.html. Accessed on 2019.

[8] Daniel Jiang, Luca Delgrossi, "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments", Mercedes-Benz Research & Development North America, Inc., 2008.

[9] IEEE, "IEEE 802.11p-2010 - IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments", 2010.

[10] SUMO, "Simulation of Urban MObility - Wiki", [Online]. Available on https://sumo.dlr.de/wiki/Simulation_of_Urban_MObility_-_Wiki. Accessed on 2019.

[11] SUMO, Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Fltterd, Robert Hilbrich, Leonhard Lcken, Johannes Rummel, Peter Wagner, and Evamarie Wiener. "Microscopic Traffic Simulation using SUMO ". IEEE Intelligent Transportation Systems Conference (ITSC), 2018.

[12] Stefan Krauβ, " Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics", 1998.

[13] OMNeT++, "What is OMNeT++?", [Online]. Available on https://omnetpp.org/intro/. Accessed on 2019.

[14] Veins, "Documentation", [Online]. Available on `http://veins.car2x.org/documentation/`. Accessed on 2019.

[15] Transportation Research Board, "Highway Capacity Manual". HCM200, 2000.

[16] SUMO, "TraCI/Change Vehicle State". [Online]. Avaible on `https://sumo.dlr.de/docs/TraCI/Change_Vehicle_State.html`. Accessed on October, 2019.

Annex A - Initial Base Case Test in a continous control framework

During the creation of the framework environment, there was the proposition of using a control that was always active, and the cars would enter a control state when they entered the Active Zone. The control would give priority to the first car that enter the Active Zone and thereafter the cars would synchronize with respect to this.

The control wasn't optimized for congested situations, where the mean travel time was around the same for cases with the control on and off, but it improved the standard deviation of the cars, distributing the total travel time between the different flows. The problem was that in uncongested situations, the control turned off was always better that situations with the control turned on, so it was opted to change to an adaptive solution.

At continuation it will be shown the global results of the test performed for different rates of cars for four different flows in uncongested situation, the flows being: North to South; South to North; East to West; West to East. This four flows start at the same time and have similar conditions at the test performed in the adaptive scenario. This test also served to selected a good time transition for the states Green, Yellow and Red, so in the results there will be shown the global result of the simulation compared between the control turned off and controls turned on with different time transitions, listing only the Green state and the Red state, giving that the Yellow state is the different between the two of them.

Results

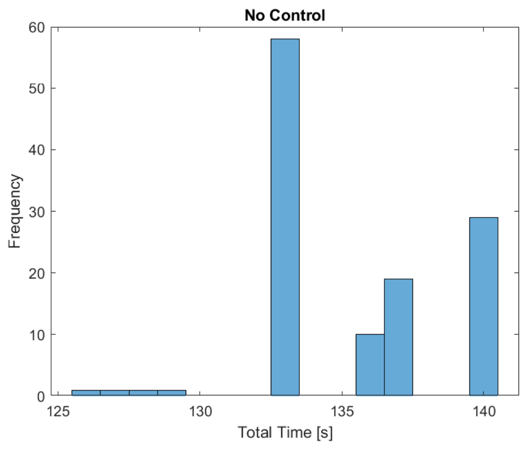
Test 30 cars per lane every 30 seconds

The test performed consisted on the four flows producing a car every 30 seconds, for a total of 120 cars in the simulation, the numeric results can be seen at continuation:

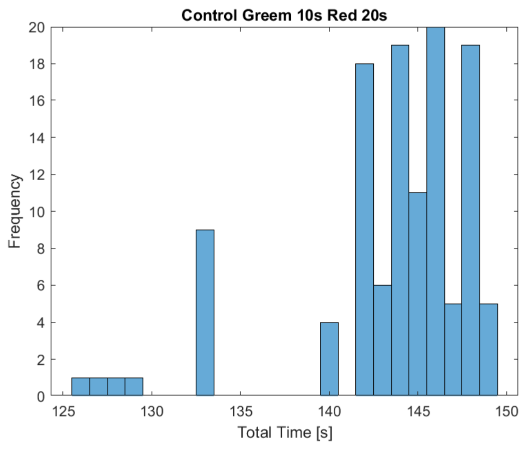
	Mean Time	Min Time	Max Time	St. Deviation
No Control	135.39	126	140	3.247
Green 10 / Red 20	143.53	126	149	4.91
Green 15 / Red 25	146.37	126	154	6.32
Green 20 / Red 30	147.63	126	161	10.07
Green 30 / Red 40	151.25	126	182	17.08
Green 40 / Red 50	151.56	126	180	14.56

Table 22: Global result of test with car generation of 30 [s], for different state transition time

We can also see the distribution of the car in the different situations of control:

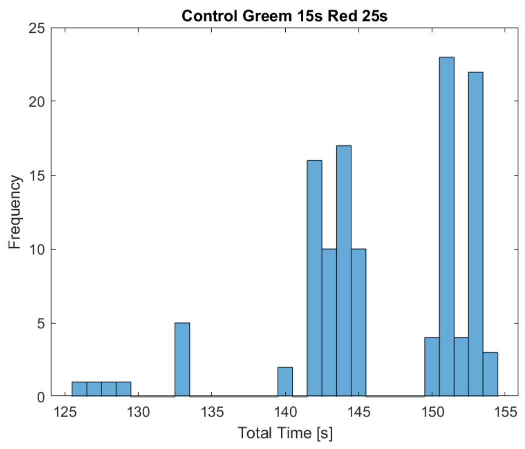


(a) No Control Situation

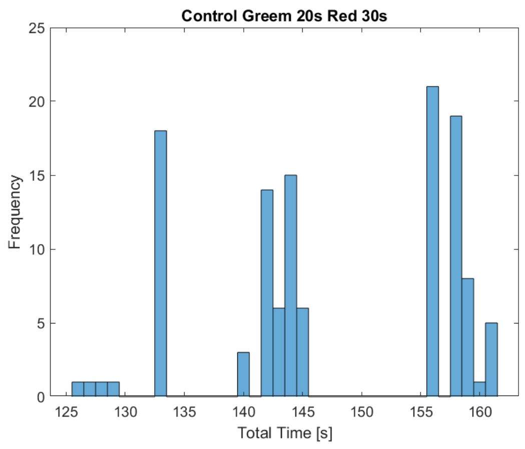


(b) Control with Green 10s / Red 20s

Figure 45: Histogram of No Control and Control Situation with Green 10s / Red 20s

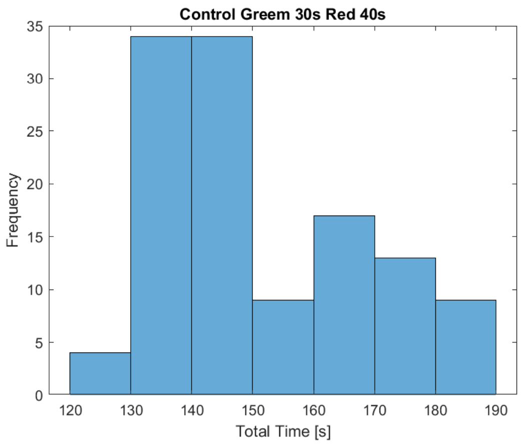


(a) Control with Green 15s / Red 25s

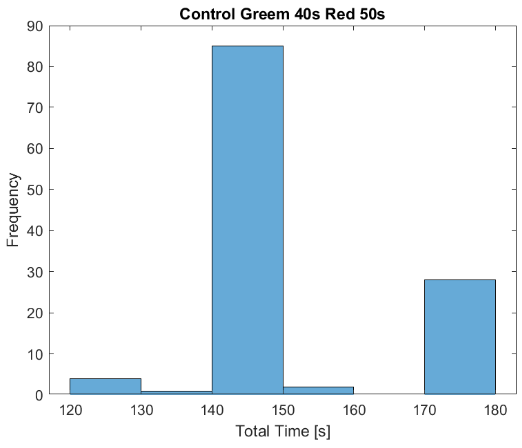


(b) Control with Green 20s / Red 30s

Figure 46: Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s



(a) Control with Green 30s / Red 40s



(b) Control with Green 40s / Red 50s

Figure 47: Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s

It's easy to see than in every iteration, the situation that has the best performance is the one with no control.

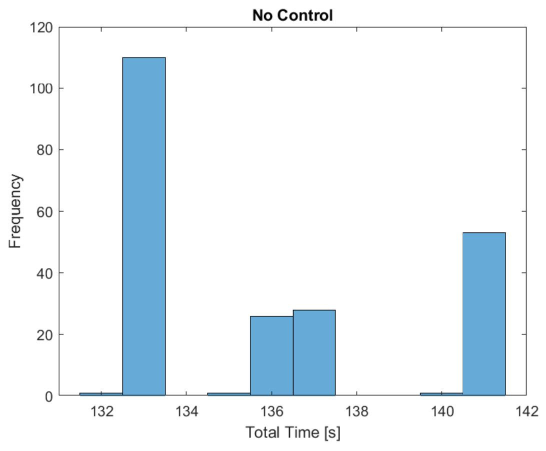
Test 55 cars per lane every 16 seconds

The test performed consisted on the four flows producing a car every 16 seconds, for a total of 880 cars in the simulation, the numeric results can be seen at continuation:

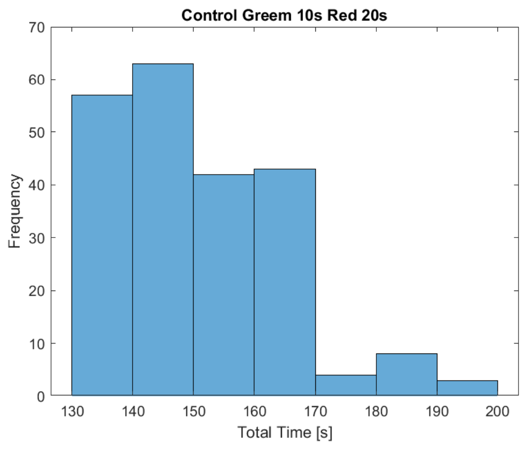
	Mean Time	Min Time	Max Time	St. Deviation
No Control	135.83	132	141	3.29
Green 10 / Red 20	150.37	132	194	13.67
Green 15 / Red 25	149.29	132	178	12.25
Green 20 / Red 30	154.66	132	222	18.89
Green 30 / Red 40	154.04	132	223	18.95
Green 40 / Red 50	156.37	132	243	23.15

Table 23: Global result of test with car generation of 16 [s], for different state transition time

We can also see the distribution of the car in the different situations of control:

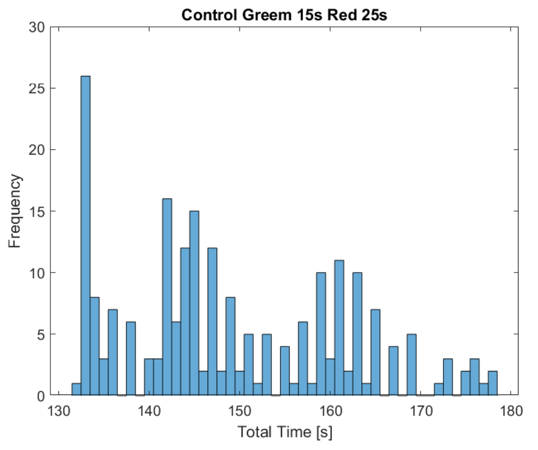


(a) No Control Situation

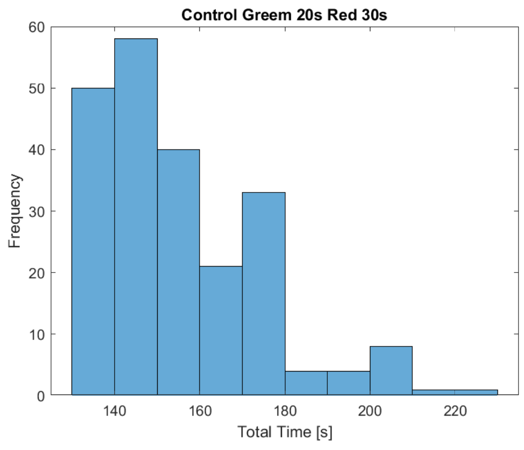


(b) Control with Green 10s / Red 20s

Figure 48: Histogram of No Control and Control Situation with Green 10s / Red 20s

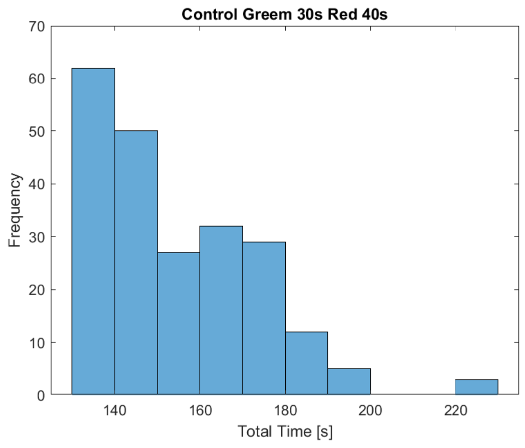


(a) Control with Green 15s / Red 25s

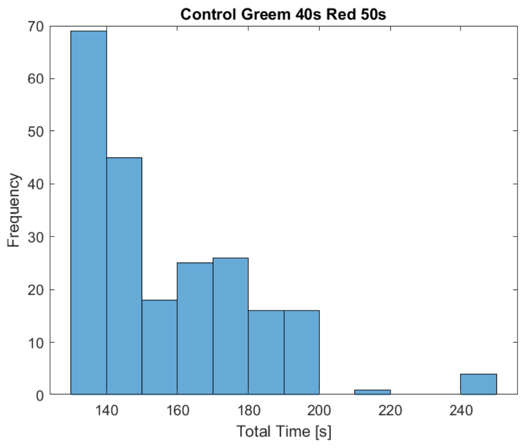


(b) Control with Green 20s / Red 30s

Figure 49: Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s



(a) Control with Green 30s / Red 40s



(b) Control with Green 40s / Red 50s

Figure 50: Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s

It's easy to see than in every iteration, the situation that has the best performance is the one with no control.

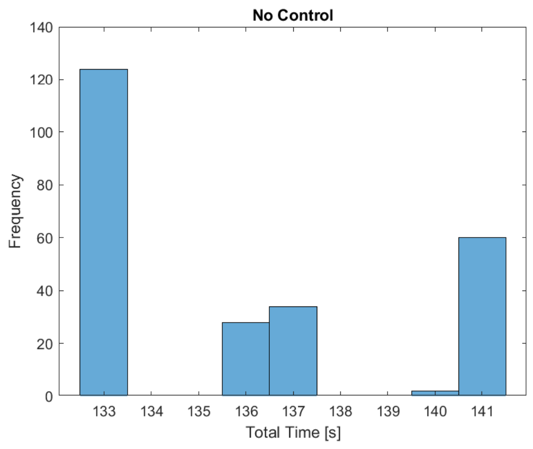
Test 62 cars per lane every 14 seconds

The test performed consisted on the four flows producing a car every 16 seconds, for a total of 868 cars in the simulation, the numeric results can be seen at continuation:

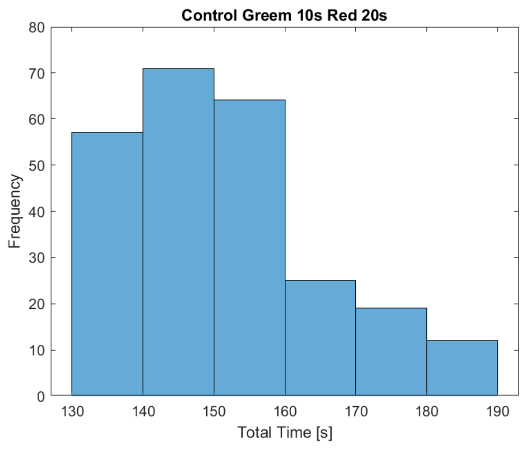
	Mean Time	Min Time	Max Time	St. Deviation
No Control	135.88	133	141	3.29
Green 10 / Red 20	150.98	133	187	13.86
Green 15 / Red 25	152.26	133	203	14.99
Green 20 / Red 30	152.88	133	213	17.33
Green 30 / Red 40	157.7	133	227	22.37
Green 40 / Red 50	159.63	133	256	26.44

Table 24: Global result of test with car generation of 14 [s], for different state transition time

We can also see the distribution of the car in the different situations of control:

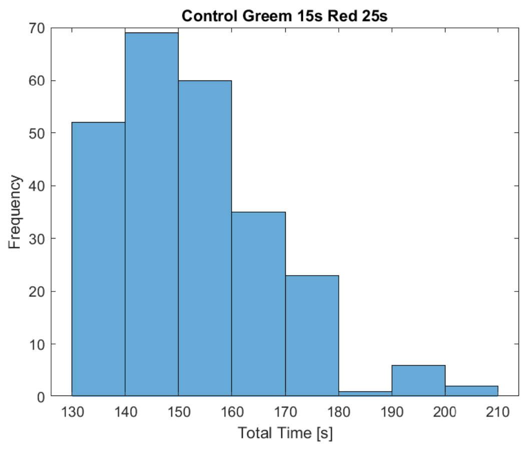


(a) No Control Situation

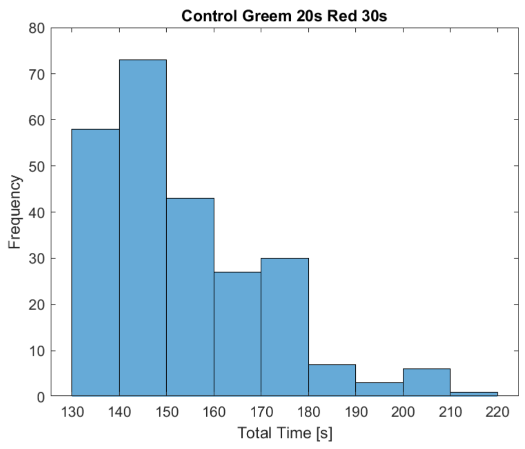


(b) Control with Green 10s / Red 20s

Figure 51: Histogram of No Control and Control Situation with Green 10s / Red 20s

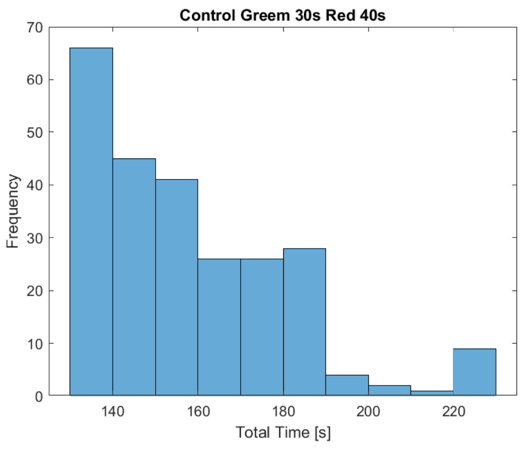


(a) Control with Green 15s / Red 25s

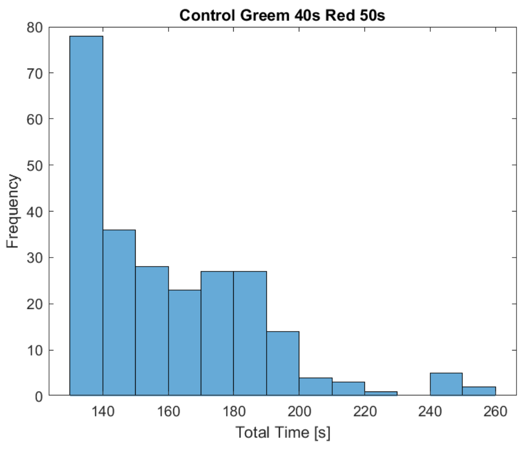


(b) Control with Green 20s / Red 30s

Figure 52: Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s



(a) Control with Green 30s / Red 40s



(b) Control with Green 40s / Red 50s

Figure 53: Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s

It's easy to see than in every iteration, the situation that has the best performance is the one with no control.

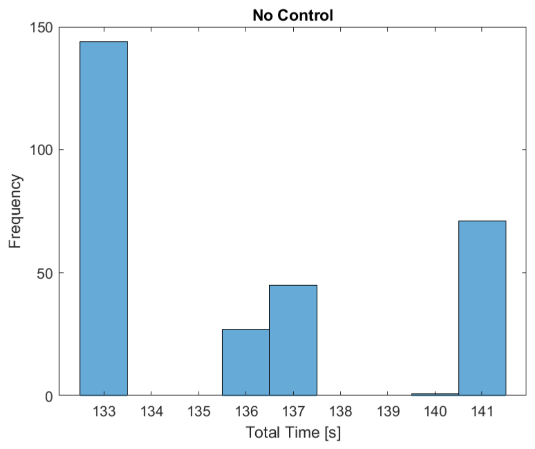
Test 72 cars per lane every 12 seconds

The test performed consisted on the four flows producing a car every 16 seconds, for a total of 864 cars in the simulation, the numeric results can be seen at continuation:

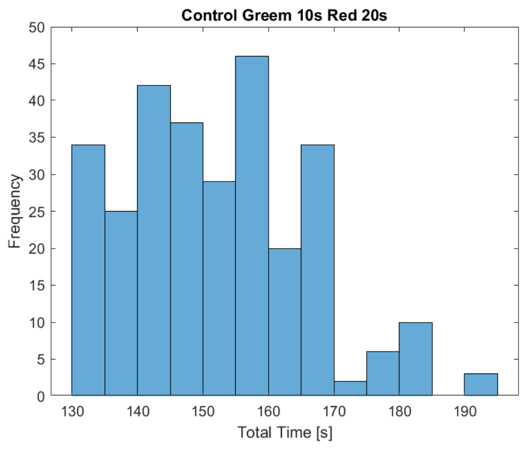
	Mean Time	Min Time	Max Time	St. Deviation
No Control	135.9	133	141	3.3
Green 10 / Red 20	151.62	133	191	13.37
Green 15 / Red 25	152.96	133	213	15.66
Green 20 / Red 30	154.14	133	214	18.12
Green 30 / Red 40	157.32	133	227	21.35
Green 40 / Red 50	164.54	133	265	30.06

Table 25: Global result of test with car generation of 14 [s], for different state transition time

We can also see the distribution of the car in the different situations of control:

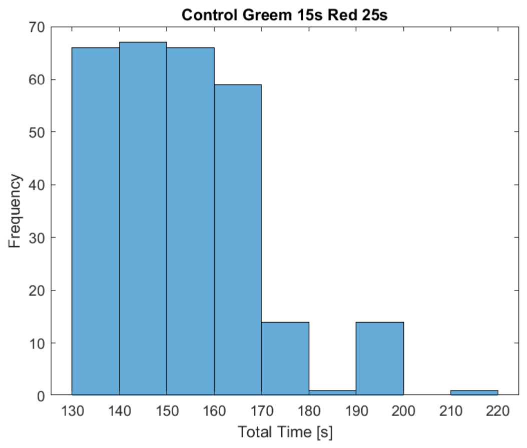


(a) No Control Situation

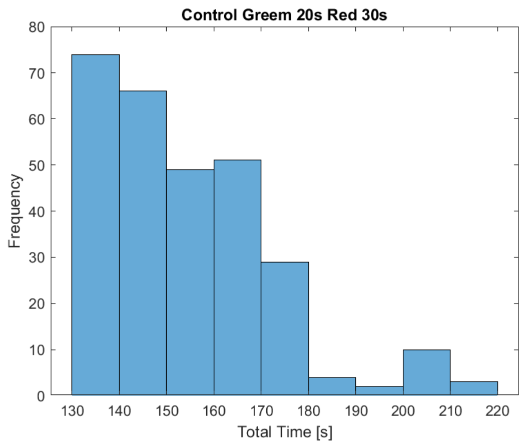


(b) Control with Green 10s / Red 20s

Figure 54: Histogram of No Control and Control Situation with Green 10s / Red 20s

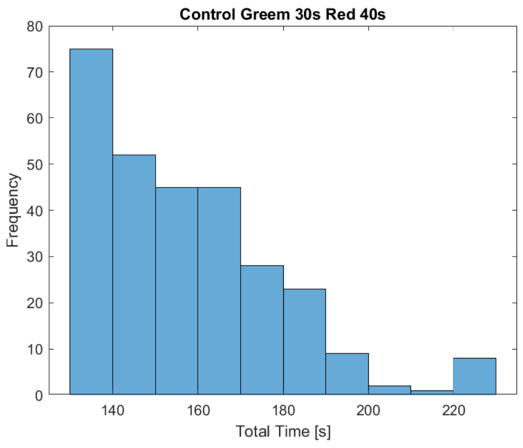


(a) Control with Green 15s / Red 25s

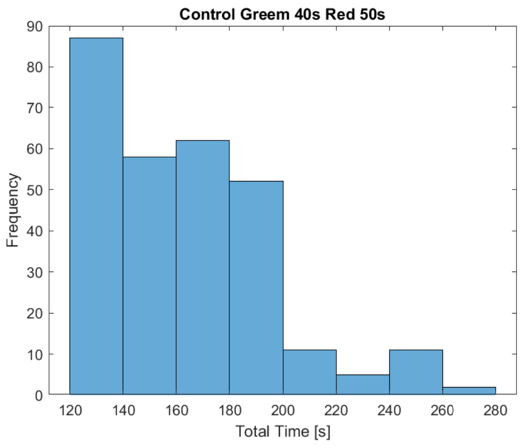


(b) Control with Green 20s / Red 30s

Figure 55: Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s



(a) Control with Green 30s / Red 40s



(b) Control with Green 40s / Red 50s

Figure 56: Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s

It's easy to see than in every iteration, the situation that has the best performance is the one with no control.

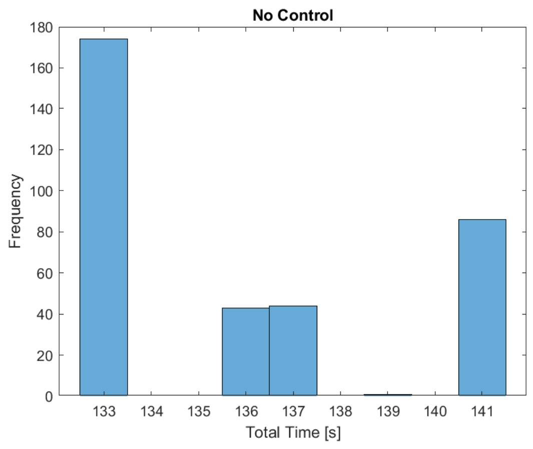
Test 87 cars per lane every 10 seconds

The test performed consisted on the four flows producing a car every 16 seconds, for a total of 870 cars in the simulation, the numeric results can be seen at continuation:

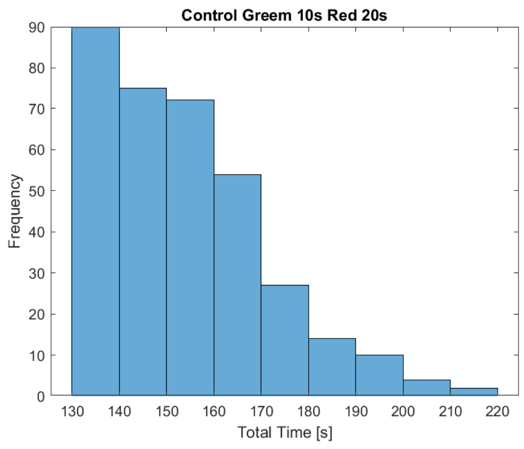
	Mean Time	Min Time	Max Time	St. Deviation
No Control	135.87	133	141	3.29
Green 10 / Red 20	153.78	133	214	16.82
Green 15 / Red 25	155.65	133	211	16.34
Green 20 / Red 30	164.69	133	238	24.23
Green 30 / Red 40	156.15	133	232	20.5
Green 40 / Red 50	190.21	133	348	50.01

Table 26: Global result of test with car generation of 14 [s], for different state transition time

We can also see the distribution of the car in the different situations of control:

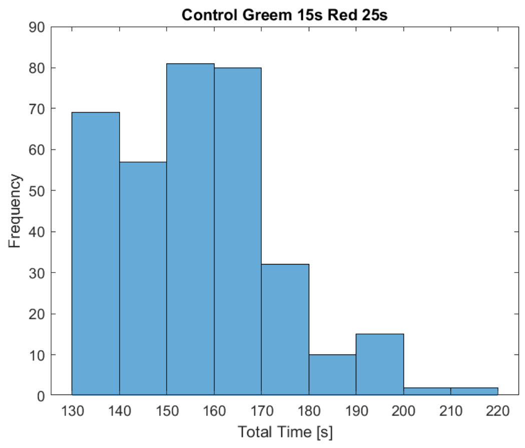


(a) No Control Situation

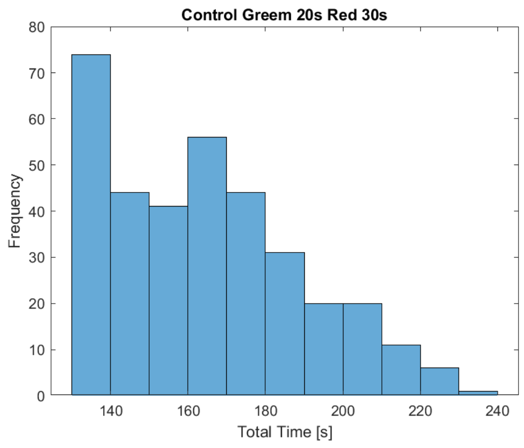


(b) Control with Green 10s / Red 20s

Figure 57: Histogram of No Control and Control Situation with Green 10s / Red 20s

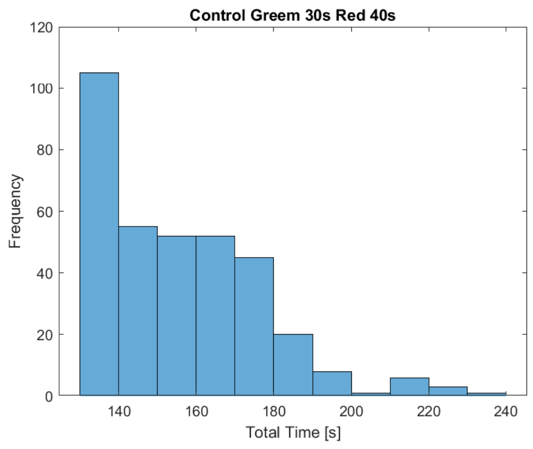


(a) Control with Green 15s / Red 25s

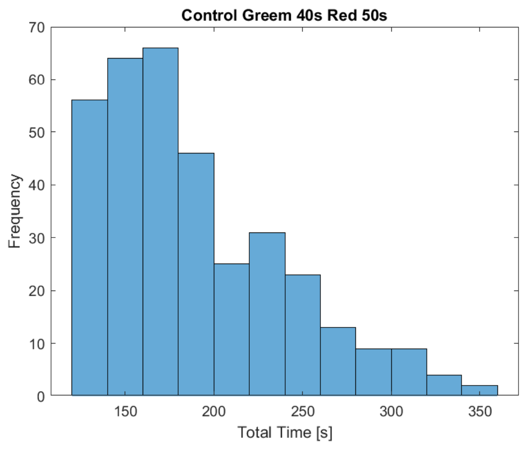


(b) Control with Green 20s / Red 30s

Figure 58: Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s



(a) Control with Green 30s / Red 40s



(b) Control with Green 40s / Red 50s

Figure 59: Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s

It's easy to see than in every iteration, the situation that has the best performance is the one with no control.

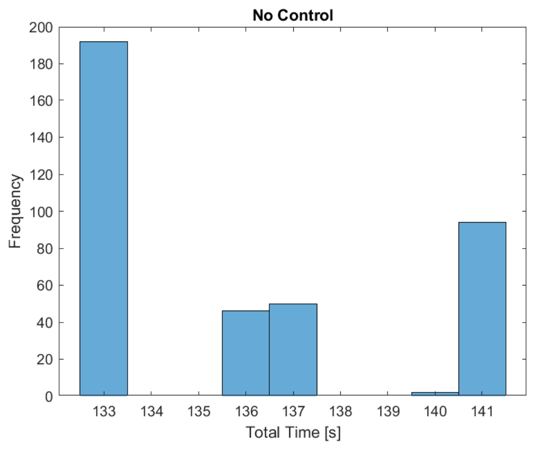
Test 96 cars per lane every 9 seconds

The test performed consisted on the four flows producing a car every 16 seconds, for a total of 864 cars in the simulation, the numeric results can be seen at continuation:

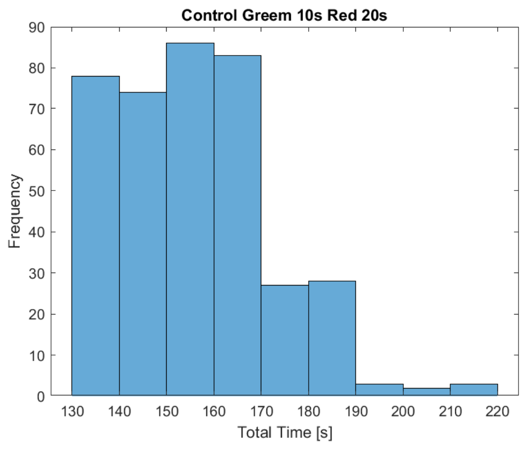
	Mean Time	Min Time	Max Time	St. Deviation
No Control	135.88	133	141	3.29
Green 10 / Red 20	155.45	133	214	16.06
Green 15 / Red 25	158.01	133	240	22.01
Green 20 / Red 30	156.77	133	223	19.50
Green 30 / Red 40	159.13	133	242	24.01
Green 40 / Red 50	164.38	133	260	30.96

Table 27: Global result of test with car generation of 14 [s], for different state transition time

We can also see the distribution of the car in the different situations of control:

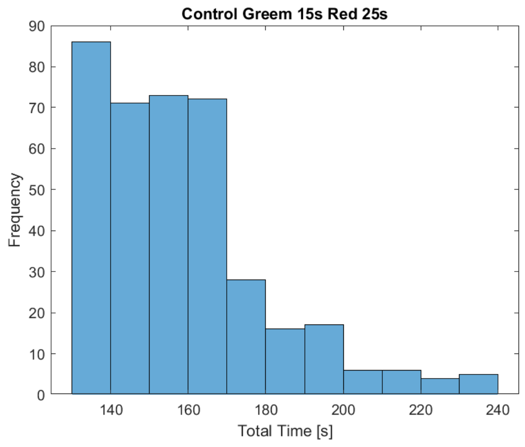


(a) No Control Situation

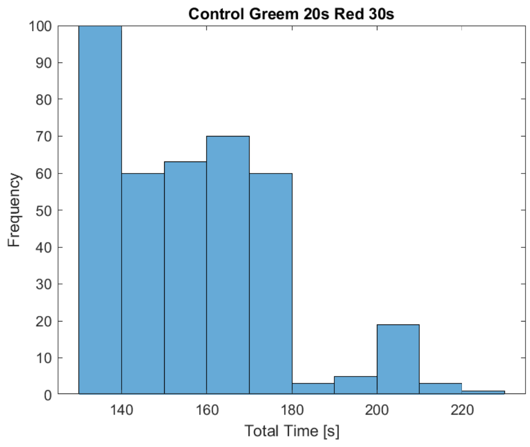


(b) Control with Green 10s / Red 20s

Figure 60: Histogram of No Control and Control Situation with Green 10s / Red 20s

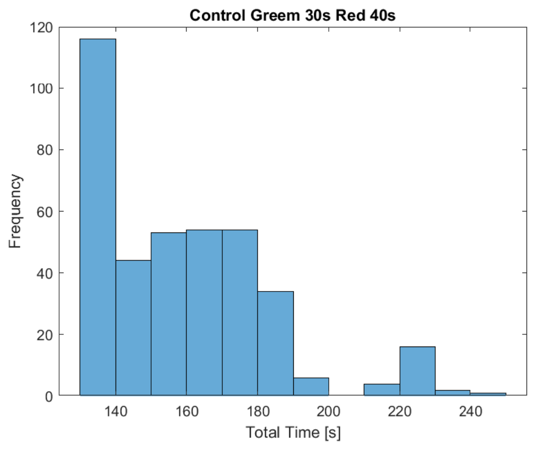


(a) Control with Green 15s / Red 25s

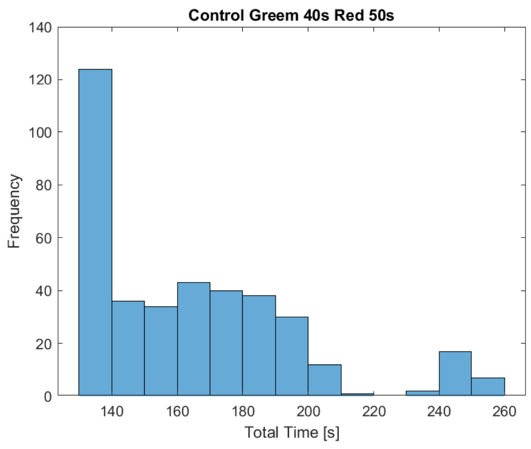


(b) Control with Green 20s / Red 30s

Figure 61: Histogram of Control Situation with Green 15s / Red 25s & Green 20s / Red 30s



(a) Control with Green 30s / Red 40s



(b) Control with Green 40s / Red 50s

Figure 62: Histogram of Control Situation with Green 30s / Red 40s & Green 40s / Red 50s

It's easy to see than in every iteration, the situation that has the best performance is the one with no control.

Short Analysis

In every test performed, the No Control case was better than all the other controls. This was to be expected, because there wasn't an origin to a possible congestion and thus, the travel time of the car shouldn't be increased in an considerably manner. This is contrasted with the control situations where they impose a restriction over the system, and a restricted system always has worse optimum compared to a less restricted system.

Of notice is that in general, smaller time transition has better performance than greater time transition. Evidently, this is not always a rule of thumbs, but in the different test performance, this was always true, and thus the time transition selected for the thesis was the control with smaller time transition of all the different time transition tested.

Annex B - Definition of functions and Vareables

This file Is the ".h" where it is defined the different functions and vareables of the developed framework. The locations is:

```
...\src\veins\modules\application\traci\TraCIDemo11p.h
```

The code is:

```
#ifndef TraCIDemo11p_H
#define TraCIDemo11p_H

#include "veins/modules/application/ieee80211p/BaseWaveApplLayer.h"

class TraCIDemo11p : public BaseWaveApplLayer {
    private:
        cOutVector      changeStateVector;
    public:
        virtual void initialize(int stage);
        virtual void finish();
    protected:
        simtime_t lastDroveAt;
        bool sentMessage;
        int currentSubscribedServiceId;

        //startControl: If the control algorithm start
            //at the initialization
        bool startControl = true;


        //Position of the road
        double const C_X =  927.6;
        double const C_Y =  927.6;

        //Lenght of the Road
        double const L = 30;
        //Area of Control
        double const M = 100;

        //Parameter or the real Road, for porpouse
            //of identifying car in same place
        double const trueL  =   5;

        //Extra parameter when calculating Time to Junction
        double const securityFactorDetention =   2;


        // decel of the car, this in later versions of veins
            //  will be obtained by a method but right now
            // we have to use the definition of the car.
        double const decelMax  =   4.5;

////////////stateControl() ALGORITHM//////////////

        //Schedule Message that invoque stateControl()
        cMessage *event              =  new cMessage("event");

        //Time stamp of when the stateControl() is invoked
        simtime_t   timeStateCtrl   =     0;

        //When you enter the Control Zone it becomes false.
            //Synchronization porpouse
        bool Enter                 = true;

        //Time to wait in idle state of stateControl() in order
            //to check Position and work in Idle Zone
        simtime_t tIdle=1;

        //Time to wait in control state of stateControl()
            //in order to check Position and work in Active Zone
        simtime_t tControl=1;

/////////////END stateControl() ALGORITHM//////////


/////////////////ADAPTIVE CONTROL ALGORITHM///////////

        //Bool vareable that become true if the control it is activated.
        bool    adaptiveControl = false;

        //Constructor of Message to Send to others cars
        WaveShortMessage* wsm;

        //Variables of direction to be filled with
            //the direction of travel of the car
        int DirectionReceivedWSM[2];

        //Message for activating the control when there
            //aren't adaptiveStarCar waiting in lane
        cMessage *checkCongestion   =  new cMessage("checkCongestion");

        //Time to surpass in order to trigger checkCongestion
        double      timeStopCar      =   10;

        //Variable for counting the number of stopped car on a lane
        int      counterStopCar;
        //Trigger to hit of waiting car in a road in
            //order to activate the control
        int      adaptiveStarCar =   2;

        //Time stamp of change of state in order to check
            //priority of starting car
        simtime_t    timeAdaptive     =      SIMTIME_MAX ;

        //Number of cycles before exiting adaptive control.
        int      longCyclesNumber         =    50;


        //Variable that it is counting the changes of states.
            //+1 when it is 0 or 2.
        int      countChanges     =    0;

        //When car stop for the first time in the active zone,
            //this vareable becomes true and send message type 2,
            //making it possible to only send this type of message
            //only one time
        bool     stopCar          =    false;


////////////////END ADAPTIVE CONTROL ALGORITHM//////////////////////

///////////////////SEMAPHORE CONTROL ALGORITHM//////////////////

        //Message for making the change of the state of the semaphore
        cMessage *checkState          =   new cMessage("checkState");

        //Time to wait between states of the semaphore
        simtime_t tGreen   = 10;
        simtime_t tRed     = 20;
        simtime_t tYellow          = tRed-tGreen;

        //Definition of the different states of the virtual semaphore
        //0 = Green  ;    1 = Yellow   ;    2 = Red
        double      semaphoreState;

/////////////////END SEMAPHORE CONTROL ALGORITHM//////////////

///////////////////FINISH EARLY CTRL/////////////////////

        //Message to see if there are cars waiting in any lane.
        cMessage *checkWaiting        =    new cMessage("checkWaiting");

        //Counter for cars waiting in line during adaptive control.
        int counterWaitingCar    =    0;
        //Threshold that to surpass in order to exit Control
        int thresholdWaiting    =   1;

        //Window of time for checking if there are cars waiting
            //in other lane
        simtime_t checkEnd  =   2;

////////////////////END FINISH EARLY CTRL//////////////////

/////////////////////////DEBUG//////////////////////////

        // Used for storing the route number defined in the
            //model of the simulation
        //  10  ->  E - W   ;   12  ->  E - N
        //  20  ->  N - S   ;   22  ->  N - W
        //  30  ->  W - E   ;   32  ->  W - S
        //  40  ->  S - N   ;   42  ->  S - E
        cMessage *route             =  new cMessage("route");

        //Variable where it is going to be stored the route
        int routeId     =   0;

        //Number of time the control algorithm is activated
        int numActivation   =   0;

        //Type of activation, if any
        int typeActivation  =   0;

        //Time of control trigger, if any
        simtime_t   timeActivation  =   0;

        //Counter finish normal cycle, if any
        int counterFinishNormal  =   0;

        //Counter finish short cycle, if any
        int counterFinishShort  =   0;

        //Debug Variable for counting if the early finish activate
            //and how much
        int counterFinishEarly  =   0;

        //If Message Type 1 Trigger bad time acepting.
        //0 = No    ; 1 = Yes
        int activationBadTime   =   0;

        //Type of update function used for synchronizing
        //  0   ->  Nothing
        //  1   ->  updateParametersOnWSMAdaptive
        //  2   ->  updateParametersOnWSM
        int functionActivation  =   0;

        //Type of update synchronize function used in synchronizing
        //  0   ->  Nothing
        //  1   ->  changeState
        //  2   ->  scheduleState
        int syncActivation  =   0;

        //Type of Message Used for synchronization of Control Algorithm
        //  0   ->  Nothing
        //  1   ->  Received Message Type 1
        //  2   ->  Received Message Type 4
        //  3   ->  Send Message Type   4
        int messageActivation  =   0;

        //  Id of Car that Trigger the control Algorithm
        int carIdTrigger;

        //  Id of the Car
        int carIdOwn;
////////////////////////END DEBUG//////////////////


    protected:
        virtual void onWSM(WaveShortMessage* wsm);
        virtual void onWSA(WaveServiceAdvertisment* wsa);

        virtual void updateParametersOnWSM();
        virtual void updateParametersOnWSMAdaptive();

        bool         checkTimeUpdate(simtime_t In);

        virtual void changeState();
        virtual void changeStateTime(simtime_t timeToNewState);
        virtual void scheduleStateTime(simtime_t timeToNewState);

        virtual void changeAdaptiveStateTime(simtime_t timeToNewState);
        virtual void scheduleAdaptiveStateTime(simtime_t timeToNewState);

        virtual void doGreen();
        virtual void doRed();
        virtual void doYellow();

        simtime_t timeToCurrentState();
        simtime_t timeToOtherState();
        virtual void handleSelfMsg(cMessage* msg);

        virtual void stateControl();
        virtual void enteringToControl();
        virtual void exitingFromControl();
        virtual void resetVariables();

        virtual void cancelSMessage(cMessage* msg);

        bool goingRigth();
        bool controlGRightZone();
        bool activeGRightZone();

        bool goingLeft();
        bool controlGLeftZone();
        bool activeGLeftZone();

        bool goingUpwards();
        bool controlGUpZone();
        bool activeGUpZone();

        bool goingDownwards();
        bool controlGDownZone();
        bool activeGDownZone();

        bool isControlZone();
        bool isActiveZone();

        bool isDiffDirection();


        virtual void populateInitialCCM();
        virtual void responseCCM();

        double timeToJunction();

        virtual void startCounter();
        virtual void normalCycleTraffic();

        virtual void verifyEndControl();

};

#endif
```

Annex C - Main Framework Structure

This file Is the ".cc" where it is developed the code and the brain of the framework.

`...\src\veins\modules\application\traci\TraCIDemo11p.cc`

The code is:

```
#include "veins/modules/application/traci/TraCIDemo11p.h"

Define_Module(TraCIDemo11p);

//Initializing function of every car Module
void TraCIDemo11p::initialize(int stage) {
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {
        sentMessage = false;
        lastDroveAt = simTime();
        currentSubscribedServiceId = -1;

        scheduleAt(simTime()+1, route);

        if(startControl){
            stateControl();
            carIdOwn      =      cSimpleModule::getId();
            changeStateVector.setName("stateChange");
            semaphoreState   =    2;
            counterStopCar   =    0;
            timeAdaptive     =    SIMTIME_MAX;
        }
    }
}

//Finishing function of every car Module
void TraCIDemo11p::finish(){
    recordScalar("routeId", routeId);
    recordScalar("numberActivation",numActivation);
    recordScalar("typeC", typeActivation);
    recordScalar("timeActivation",timeActivation);

    recordScalar("functionActivation",functionActivation);
    recordScalar("messageActivation",messageActivation);
    recordScalar("syncActivation",syncActivation);

    recordScalar("ActivationBadTime",activationBadTime);

    recordScalar("triggerCarId",carIdOwn);
    recordScalar("triggerCarReceive",carIdTrigger);
    recordScalar("counterFinishNormal",counterFinishNormal);
    recordScalar("counterFinishShort",counterFinishShort);
    recordScalar("counterFinishEarly",counterFinishEarly);
}

void TraCIDemo11p::onWSA(WaveServiceAdvertisment* wsa) {
    if (currentSubscribedServiceId == -1) {
        mac->changeServiceChannel(wsa->getTargetChannel());
        currentSubscribedServiceId = wsa->getPsid();
        if  (currentOfferedServiceId != wsa->getPsid()) {
            stopService();
            startService((Channels::ChannelNumber)
                wsa->getTargetChannel(), wsa->getPsid(),
                "Mirrored Traffic Service");
        }
    }
}

//Activate every time the car module receive a message type wsm
void TraCIDemo11p::onWSM(WaveShortMessage* wsm) {

    EV << "onWSM()\n";

    //The car is already on control. Send a message to the asking car.
    if( (wsm->getCcmType()==0) && isActiveZone() && !Enter
        && adaptiveControl){
           EV << "0\n";

           responseCCM();
       }

    //Response to message type 0. The car enter to control
    else if((wsm->getCcmType()==1) && (wsm->getTimestamp() < timeAdaptive)
        && !adaptiveControl && isActiveZone() ){
        EV << "1\n";
        EV << "Received synchronization Signal (second type)\n";

        messageActivation        =    1;
        timeActivation           =    simTime();

        numActivation++;
        adaptiveControl          =    true;
        stopCar                  =    true;

        carIdTrigger             =    wsm->getCarId();

        timeAdaptive             =    wsm->getTimestamp();
        semaphoreState           =    wsm->getSemaphoreState();
        countChanges                =    wsm->getCountMessage();
        DirectionReceivedWSM[0]     =    wsm->getDirection(0);
        DirectionReceivedWSM[1]     =    wsm->getDirection(1);

        //Prevent Schedule to the past
        if( !checkTimeUpdate(wsm->getTimestamp() ) ){
            activationBadTime    =    1;
            countChanges++;

            if(semaphoreState ==1){
                timeAdaptive     +=   tRed;
            }
            else if (semaphoreState ==2){
                timeAdaptive     +=   tGreen;
            }
            semaphoreState++;

            if(semaphoreState>2){
                semaphoreState=0;
            }
        }

        if(countChanges == 0){
            updateParametersOnWSMAdaptive();
        }
        else{
            updateParametersOnWSM();
        }

    }

    //Car send a message indicating states
    else if( (wsm->getCcmType()==2) && (mobility->getSpeed()<1)
        && !adaptiveControl && isActiveZone() ){
        EV << "2\n";

        counterStopCar++;
        EV <<"Counter Stop Car= " << counterStopCar << "\n";
        if(counterStopCar > adaptiveStarCar){
            adaptiveControl =    true;
            stopCar         =    true;

            cancelSMessage(checkCongestion);

            typeActivation   =    2;
            normalCycleTraffic();
        }
    }
```

```
        //Early Finish check
        else if( (wsm->getCcmType()==3) && isActiveZone()
            && adaptiveControl ){
            EV << "3\n";

            DirectionReceivedWSM[0] =   wsm->getDirection(0);
            DirectionReceivedWSM[1] =   wsm->getDirection(1);

            if(isDiffDirection()){
                counterWaitingCar++;
            }
        }

        //Sunchronization when car is already on the active zone
        else if( (wsm->getCcmType()==4) && isActiveZone()
            && (wsm->getTimestamp() < timeAdaptive)    ){
            EV << "4\n";

            messageActivation           =   2;
            timeActivation              =   simTime();

            numActivation++;
            adaptiveControl             =   true;
            stopCar                     =   true;

            carIdTrigger                =   wsm->getCarId();

            timeAdaptive                =   wsm->getTimestamp();
            semaphoreState              =   wsm->getSemaphoreState();
            countChanges                =   wsm->getCountMessage();

            DirectionReceivedWSM[0] =   wsm->getDirection(0);
            DirectionReceivedWSM[1] =   wsm->getDirection(1);

            cancelSMessage(checkCongestion);
            updateParametersOnWSMAdaptive();
        }

        else{
            EV << "Received signal has been ignored\n";
        }

        wsm = nullptr;
}

//When the car judge that the incoming message is good,
//synchronize the parameter of the car with of the
```

```
//ongoing control algorithm
void TraCIDemo11p::updateParametersOnWSM() {
    EV <<"updateParametersOnWSM()\n";

    cancelSMessage(checkCongestion);

    EV << "New Time Control is: " << timeAdaptive << "\n";

    if( isDiffDirection() ){
        EV << "Signal is of different orientation\n";
        changeStateTime(timeToOtherState()-simTime()+timeAdaptive);
        syncActivation   =   1;
    }
    else{
        EV << "Signal is of the same orientation\n";
        scheduleStateTime(timeToCurrentState()-simTime()+timeAdaptive);
        syncActivation   =   2;
    }

    functionActivation   =   2;
}

//When the car judge that the incoming message is good,
//synchronize the parameter of the car with of the ongoing
//control algorithm
void TraCIDemo11p::updateParametersOnWSMAdaptive() {
    EV <<"updateParametersOnWSMAdaptive()\n";

    cancelSMessage(checkCongestion);

    EV << "New Time Control is: " << timeAdaptive << "\n";

    if( isDiffDirection() ){
        EV << "Signal is of different orientation\n";
        changeAdaptiveStateTime(tYellow-simTime()+timeAdaptive);
        syncActivation   =   1;
    }
    else{
        EV << "Signal is of the same orientation\n";
        scheduleAdaptiveStateTime(tYellow-simTime()+timeAdaptive);
        syncActivation   =   2;
    }

    functionActivation   =   1;
}

// Return true if the incoming time can be accepted, otherwise false
bool TraCIDemo11p::checkTimeUpdate(simtime_t In) {
    bool var    =   false;

    if( (isDiffDirection() ) && (timeToOtherState()-simTime()+In > 0) ){
        var =   true;
    }
    else if( (!isDiffDirection() ) &&
        (timeToCurrentState()-simTime()+In > 0)  ){
        var =   true;
    }

    return var;
}

//Function for self scheduling message
void TraCIDemo11p::handleSelfMsg(cMessage* msg) {

    EV << "handleSelfMsg()\n";
    if (msg == event){
        stateControl();
    }

    else if (msg==checkState){
        EV  << "countChanges before update = " << countChanges << "\n";

        if(countChanges  <  2*longCyclesNumber){
            changeState();
            EV << "It has been updated the change of semaphore\n";
        }
        else{
            EV << "The car Exit from the Control\n";
            resetVariables();
            traciVehicle->setSpeed(-1);
            counterFinishNormal++;
        }
    }
    else if (msg==checkCongestion){
        //Start control for one cycle
        if(mobility->getSpeed() < 1 ){
            adaptiveControl =   true;
            typeActivation  =   1;
            normalCycleTraffic();
        }
    }

    else if(msg==checkWaiting){
        if(counterWaitingCar >= thresholdWaiting){
            counterWaitingCar   =   0;
        }
        else{
            resetVariables();
            traciVehicle->setSpeed(-1);
            cancelSMessage(checkState);
            counterFinishEarly++;
        }
    }
    else if (msg==route){
        routeId      =    std::stoi(traciVehicle ->getRouteId());
    }
}

//Function for populating the message type 1, and sending it
void TraCIDemo11p::populateInitialCCM() {        // Not used
    EV<<"populateInitialCCM()\n";

    wsm = new WaveShortMessage();
    populateWSM(wsm);

    wsm->setCcmType(0);
    wsm->setTimestamp(timeAdaptive);
    //wsm->setSemaphoreState(semaphoreState);
    //wsm->setCountMessage(countChanges);

    //Going to rigth
    if( goingRigth() ){
        wsm->setDirection(0,1);
        wsm->setDirection(1,0);
    }
    //Going Left
    else if( goingLeft() ){
        wsm->setDirection(0,-1);
        wsm->setDirection(1,0);
    }
    //Going Down
    else if( goingDownwards() ){
        wsm->setDirection(0,0);
        wsm->setDirection(1,1);
    }
    //Going Up
    else if( goingUpwards() ){
        wsm->setDirection(0,0);
        wsm->setDirection(1,-1);
    }

    EV <<"The Control Variable Time is: "<< timeAdaptive << "\n";
    sendDelayedNormalD(wsm);
    wsm = nullptr;
}


//Give a WSM message response with the time of control,
//the current state of the semaphore, and the type of WSM Message
void TraCIDemo11p::responseCCM() {
    wsm = new WaveShortMessage();
    populateWSM(wsm);

    wsm->setCarId(carIdOwn);

    wsm->setCcmType(1);
    wsm->setTimestamp(timeAdaptive);
    wsm->setSemaphoreState(semaphoreState);
    wsm->setCountMessage(countChanges);

    //Going to rigth
    if( goingRigth() ){
        wsm->setDirection(0,1);
        wsm->setDirection(1,0);
    }
    //Going Left
    else if( goingLeft() ){
        wsm->setDirection(0,-1);
        wsm->setDirection(1,0);
    }
    //Going Down
    else if( goingDownwards() ){
        wsm->setDirection(0,0);
        wsm->setDirection(1,1);
    }
    //Going Up
    else if( goingUpwards() ){
        wsm->setDirection(0,0);
        wsm->setDirection(1,-1);
    }

    sendDelayedNormalD(wsm);

    wsm = nullptr;
}

/////////////////////////////////////////////////////////////
//Functions for the correct work of the virtual semaphore,
//and their synchronization in the ongoing control algorithm
void TraCIDemo11p::changeState() {
```

```
EV << "changeState()\n";
//Update the current state
semaphoreState++;
if(semaphoreState > 2)
    semaphoreState=0;

if(semaphoreState==0){

    cancelSMessage(checkState);

    EV << "It has been updated to Green Semaphore\n";

    timeAdaptive=simTime();
    countChanges++;

    doGreen();

    EV << "Time to new State is: " << tGreen <<" With State: "
        << semaphoreState<<"\n";
    EV << "Control Time: " << timeAdaptive <<"\n";

    verifyEndControl();
    scheduleAt(simTime()+tGreen, checkState);
}
else if(semaphoreState==1){

    EV << "It has been updated to Yellow Semaphore\n";

    doYellow();

    EV << "changeState: Time to new State is: " << tYellow
        <<" With State: "<< semaphoreState<<"\n";

    cancelSMessage(checkState);
    scheduleAt(simTime()+tYellow, checkState);
}
else if(semaphoreState==2){
    EV << "It has been updated to Red Semaphore\n";

    timeAdaptive=simTime();
    countChanges++;

    doRed();

    EV << "changeState: Time to new State is: " << tRed
        <<" With State: "<< semaphoreState<<"\n";
    EV << "Control Time: " << timeAdaptive <<"\n";
```

```
        cancelSMessage(checkState);
        verifyEndControl();
        scheduleAt(simTime()+tRed, checkState);
    }
    else
        EV <<"Semaphore State is Wrong with number semaphoreState="
            <<semaphoreState<<"\n";



    changeStateVector.recordWithTimestamp(simTime(),semaphoreState);
}

void TraCIDemo11p::changeStateTime(simtime_t timeToNewState) {

    EV << "changeStateTime()\n";

    if(semaphoreState==0 || semaphoreState ==1){
        semaphoreState=2;
    }
    else{
        if(simTime()-timeAdaptive < tGreen ){
            semaphoreState=0;
        }
        else
            semaphoreState=1;
    }


    if(semaphoreState==0){
        EV << "It has been updated to Green Semaphore\n";
        doGreen();
    }
    else if(semaphoreState==1){
        EV << "It has been updated to Yellow Semaphore\n";
        // Tiempo al nuevo estado tiene que ser lo sufientemente
            //alto como para efectuar cambios a la velocidad
        doYellow();
    }
    else if(semaphoreState==2){
        EV << "It has been updated to Red Semaphore\n";
        doRed();
    }
    else
        EV <<"Semaphore State is Wrong with number semaphoreState="
            <<semaphoreState<<"\n";

    EV << "changeStateTime: Time to new State is: " << timeToNewState
```

```
            <<" With State: "<< semaphoreState<<"\n";

        changeStateVector.recordWithTimestamp(simTime(),semaphoreState);

        cancelSMessage(checkState);
        scheduleAt(simTime()+timeToNewState, checkState);
}

void TraCIDemo11p::scheduleStateTime(simtime_t timeToNewState) {

        EV << "scheduleStateTime()\n";
        if(semaphoreState==0){
            EV << "It has been updated to Green Semaphore\n";
            doGreen();
        }
        else if(semaphoreState==1){
            EV << "It has been updated to Yellow Semaphore\n";
            // Tiempo al nuevo estado tiene que ser lo sufientemente
                //alto como para efectuar cambios a la velocidad
            doYellow();
        }
        else if(semaphoreState==2){
            EV << "It has been updated to Red Semaphore\n";
            doRed();
        }
        else
            EV <<"Semaphore State is Wrong with number semaphoreState="
                <<semaphoreState<<"\n";

        EV << "Time to new State is: " << timeToNewState <<" With State: "
            << semaphoreState<<"\n";

        changeStateVector.recordWithTimestamp(simTime(),semaphoreState);

        cancelSMessage(checkState);
        scheduleAt(simTime()+timeToNewState, checkState);
}

//Different Direction
void TraCIDemo11p::changeAdaptiveStateTime(simtime_t timeToNewState) {
        if(semaphoreState==1){
            semaphoreState=2;
            doRed();
        }
        else{
            semaphoreState=1;
            doYellow();
        }
```

```
    changeStateVector.recordWithTimestamp(simTime(),semaphoreState);

    cancelSMessage(checkState);
    scheduleAt(simTime()+timeToNewState, checkState);
}

//Same Direction
void TraCIDemo11p::scheduleAdaptiveStateTime(simtime_t timeToNewState)
    if(semaphoreState==2){
        semaphoreState=2;
        doRed();
    }
    else{
        semaphoreState=1;
        doYellow();
    }

    changeStateVector.recordWithTimestamp(simTime(),semaphoreState);

    cancelSMessage(checkState);
    scheduleAt(simTime()+timeToNewState, checkState);
}

//////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////
//Functions for Afecting Velocity when they pass to certain States

void TraCIDemo11p::doGreen() {
    EV <<"Car is in Green";
    traciVehicle->setSpeedMode(7);
    traciVehicle->setSpeed(-1);
}

void TraCIDemo11p::doRed() {
    EV <<"Car is in Red";
    traciVehicle->setSpeedMode(15);

    simtime_t   remainingTime;
    simtime_t   scheduleTime;

    if(countChanges==0){
        remainingTime   =   tYellow + timeAdaptive - simTime();
    }
    else{
        remainingTime   =   tRed + timeAdaptive - simTime();
    }
    if(timeStateCtrl    !=   simTime()){
        scheduleTime    =    timeStateCtrl+tControl-simTime();
    }
    else{
        scheduleTime    =    tControl;
    }

    double  remainingTimeDouble =   SIMTIME_DBL(remainingTime);
    double  scheduleTimeDouble  =   SIMTIME_DBL(scheduleTime);

    // onlyin control zone
    if(isControlZone()){
        double timeJunction =   timeToJunction();
        double decelActual  =   mobility->getSpeed()/timeJunction;

        if( (timeJunction > remainingTimeDouble)
            && (mobility->getSpeed()>0.5)  ){
            traciVehicle->setSpeed(-1);
        }
        else if( (timeJunction <= remainingTimeDouble)
            && (mobility->getSpeed()>0.5)  ){
            if(decelActual > decelMax){
                traciVehicle->slowDown(abs(mobility->getSpeed() -
                    decelMax*scheduleTimeDouble),
                        scheduleTimeDouble*1000);
            }
            else{
                traciVehicle->slowDown(abs(mobility->getSpeed() -
                    decelActual*scheduleTimeDouble),
                        scheduleTimeDouble*1000);
            }
        }
        else{
            traciVehicle->setSpeed(0);
        }
    }
    else{
        EV <<"Car Remain Stopped";
        traciVehicle->setSpeed(0);
    }
}

void TraCIDemo11p::doYellow() {
    EV <<"Car is in Yellow";
    traciVehicle->setSpeedMode(7);
    simtime_t    remainingTime;
    simtime_t    scheduleTime;
```

```
if(countChanges==0){
    remainingTime    =    tYellow + timeAdaptive - simTime();
}
else{
    remainingTime    =    tRed + timeAdaptive - simTime();
}

if(timeStateCtrl    !=   simTime()){
    scheduleTime    =    timeStateCtrl+tControl-simTime();
}
else{
    scheduleTime    =    tControl;
}

double  remainingTimeDouble =    SIMTIME_DBL(remainingTime);
double  scheduleTimeDouble  =    SIMTIME_DBL(scheduleTime);


if(remainingTimeDouble >  SIMTIME_DBL(tYellow)*0.7){
    EV <<"Car will not stop";
    traciVehicle->setSpeed(-1);
}
// only in control zone
else if(isControlZone() ){
    double timeJunction =    timeToJunction();
    double decelActual  =    mobility->getSpeed()/timeJunction;

    if( remainingTimeDouble >=  SIMTIME_DBL(tYellow)*0.4){
        if((timeJunction > remainingTimeDouble)  ||
            (timeJunction*2 < remainingTimeDouble)  ){
            traciVehicle->setSpeed(-1);
        }
        else{
            if(decelActual > decelMax){
                traciVehicle->slowDown(abs(mobility->getSpeed() -
                    decelMax*scheduleTimeDouble),
                        scheduleTimeDouble*1000);
            }
            else{
                traciVehicle->slowDown(abs(mobility->getSpeed() -
                    decelActual*scheduleTimeDouble),
                        scheduleTimeDouble*1000);
            }
        }
    }
    else if( remainingTimeDouble <  SIMTIME_DBL(tYellow)*0.4  ){
        if( (timeJunction > remainingTimeDouble) &&
            (mobility->getSpeed()>0.5)  ){
                    if(decelActual > decelMax){
                        traciVehicle->slowDown(abs(mobility->getSpeed() -
                            decelMax*scheduleTimeDouble),
                                scheduleTimeDouble*1000);
                    }
                    else{
                        traciVehicle->slowDown(abs(mobility->getSpeed() -
                            decelActual*scheduleTimeDouble),
                                scheduleTimeDouble*1000);
                    }
                }
                else{
                    traciVehicle->setSpeed(0);
                }
            }
        }
        else{
            if( (remainingTime < 5) && mobility->getSpeed() < 6 ){
                traciVehicle->setSpeed(0);
            }
            else{
                traciVehicle->setSpeed(-1);
            }
        }
    }

    ////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////
    //Functions for calculating remaining time in different states.

    simtime_t TraCIDemo11p::timeToCurrentState(){
        simtime_t varTime;

        if(countChanges==0){
            varTime =   tYellow;
        }
        else{
            if (semaphoreState==0){
                varTime = tGreen;
            }
            else if (semaphoreState==1){
                varTime = tGreen+tYellow;
            }
            else if (semaphoreState==2){
                varTime = tRed;
            }
        }
        return varTime;
```

```
}

simtime_t TraCIDemo11p::timeToOtherState(){
    simtime_t varTime;

    if(countChanges==0){
        varTime =   tYellow;
    }
    else{
        if (semaphoreState==0 || semaphoreState==1){
            varTime = tRed;
        }
        else{
            if(simTime()-timeAdaptive < tGreen){
                varTime=tGreen;
            }
            else
                varTime=tGreen+tYellow;
        }
    }

    return varTime;
}

//Function to calculate the time given the actual speed
    //at the junction more nearby
double TraCIDemo11p::timeToJunction() {
    double distance =    0;
    double time_double = 0;

    if( goingRigth() ){ //Going to rigth
        distance = C_X-L-securityFactorDetention -
            mobility->getCurrentPosition().x;
        if((mobility->getCurrentSpeed().x)>0.5){
            time_double=2*distance/abs(mobility->getCurrentSpeed().x);
        }
        else{
            time_double=4*distance;
        }
    }
    else if( goingLeft() ){//Going to Left
        distance = mobility->getCurrentPosition().x -
            C_X-L-securityFactorDetention;
        if(abs(mobility->getCurrentSpeed().x)>0.5){
            time_double=2*distance/abs(mobility->getCurrentSpeed().x);
        }
        else{
            time_double=4*distance;
        }
    }
    else if( goingDownwards() ){//Going downwards
        distance = C_Y-L-securityFactorDetention -
            mobility->getCurrentPosition().y;
        if(abs(mobility->getCurrentSpeed().y)>0.5){
            time_double=2*distance/abs(mobility->getCurrentSpeed().y);
        }
        else{
            time_double=4*distance;
        }
    }
    else if( goingUpwards() ){//Going upstairs
        distance = mobility->getCurrentPosition().y -
            C_Y-L-securityFactorDetention;
        if(abs(mobility->getCurrentSpeed().y)>0.5){
            time_double=2*distance/abs(mobility->getCurrentSpeed().y);
        }
        else{
            time_double=4*distance;
        }
    }
    return time_double;
}

//////////////////////////////////////////////////////////////////
// Function for controlling the control state transition of the car
void TraCIDemo11p::stateControl() {
    EV<<"stateControl()\n";
    EV<<"Position: [" << mobility->getCurrentPosition().x << "," <<
        mobility->getCurrentPosition().y << "]\n";
    timeStateCtrl   =   simTime();

    if(isActiveZone()){
        EV <<"Car is in Active Zone\n";
        scheduleAt(simTime()+tControl, event);

        if(Enter){
            // Put this when control activate
            traciVehicle->setSpeedMode(15);
            enteringToControl();
        }

        if(adaptiveControl){
            EV <<"Car is being controlled\n";
            EV  << "countChanges = " << countChanges << "\n";
            if(countChanges==0){
                EV <<"Control Time is: " << timeAdaptive
```

```
                  << ". Time to State Change is: "
                      << 10-simTime()+timeAdaptive
                          <<" With State: " << semaphoreState<<"\n";
          }
          else{
              EV <<"Control Time is: " << timeAdaptive <<
                  ". Time to State Change is: "<<
                      timeToCurrentState()-simTime()+timeAdaptive
                          <<" With State: "<< semaphoreState<<"\n";
          }

          /////////Responsive Velocity change with redundancy
          if(semaphoreState==0){
              doGreen();
          }
          else if(semaphoreState==1){
              doYellow();
          }
          else if(semaphoreState==2){
              doRed();
          }

          ////////////////////////////////////////////////////////

      }
      else{
          EV << "Car is not being controlled\n";
      }
      //Next line is a try before discovering speed mode
      //if(semaphoreState==0){
      //    traciVehicle->setSpeed(-1);
      //}
      EV <<"Moving to: ["<< mobility->getCurrentDirection().x
          << ","<<mobility->getCurrentDirection().y <<"]\n";
      EV <<"Velocity of the car is: " << mobility->getSpeed()
          << "\n";

  }
  else{
      if(!Enter){
          exitingFromControl();
      }


      EV <<"Car is not in Active Zone\n";
      EV <<"Control Time is: " << timeAdaptive << "\n";
      EV <<"Velocity of the car is: " << mobility->getSpeed()
          << "\n";
```

```
        scheduleAt(simTime()+tIdle, event);
    }

    if( (mobility->getSpeed()<0.5) && !adaptiveControl && !stopCar){

        stopCar =   true;
        scheduleAt(simTime()+timeStopCar, checkCongestion);

        startCounter();
    }
}

//////////////////////////////////////////////////////////////////
//Complementary Functions of ControlState()

void TraCIDemo11p::enteringToControl() {
    EV<<"enteringToControl()\n";
    recordScalar("enterControlZone",simTime());

    Enter           =   false;

    if(!adaptiveControl){
        semaphoreState  =       2;
        populateInitialCCM();
    }
}

void TraCIDemo11p::exitingFromControl() {
    EV<<"exitingFromControl()\n";

    Enter   =   true;
    resetVariables();
    traciVehicle->setSpeedMode(31);

    timeAdaptive        =   SIMTIME_MAX;

    cancelSMessage(checkCongestion);
    cancelSMessage(checkState);

    traciVehicle->setSpeed(-1);
}

void TraCIDemo11p::resetVariables() {
    EV<<"resetVariables()\n";

    traciVehicle->setSpeedMode(15);
    stopCar             =   false;
    adaptiveControl     =   false;

    countChanges     =   0;
    counterStopCar   =   0;
    counterWaitingCar    =   0;
}


///////////////////////////////////////////////////////////////////
//Boolean functions for direction of traveling of the car,
    //and position on the map
bool TraCIDemo11p::goingRigth() {
    bool var    = false;
    if( mobility->getCurrentDirection().x > 0.87){
        var=true;
    }

    return var;
}

bool TraCIDemo11p::controlGRightZone() {
    bool var    = false;

    if( (mobility->getCurrentPosition().x  >= C_X-L-M ) &&
        (mobility->getCurrentPosition().x < C_X-L)){
        var=true;
    }

    return var;
}

bool TraCIDemo11p::activeGRightZone() {
    bool var    = false;

    if( (mobility->getCurrentPosition().x  >= C_X-L-M ) &&
        (mobility->getCurrentPosition().x < C_X-trueL)){
        var=true;
    }

    return var;
}


bool TraCIDemo11p::goingLeft() {
    bool var    = false;
    if( mobility->getCurrentDirection().x < -0.87){
        var=true;
    }

    return var;
}

bool TraCIDemo11p::controlGLeftZone() {
    bool var    = false;

    if( (mobility->getCurrentPosition().x  <= C_X+L+M) &&
        (mobility->getCurrentPosition().x  > C_X+L)){
        var=true;
    }

    return var;
}

bool TraCIDemo11p::activeGLeftZone() {
    bool var    = false;

    if( (mobility->getCurrentPosition().x  <= C_X+L+M) &&
        (mobility->getCurrentPosition().x  > C_X+trueL)){
        var=true;
    }

    return var;
}


bool TraCIDemo11p::goingUpwards() {
    bool var    = false;
    if( mobility->getCurrentDirection().y < -0.87){
        var=true;
    }

    return var;
}

bool TraCIDemo11p::controlGUpZone() {
    bool var    = false;

    if( (mobility->getCurrentPosition().y  <= C_Y+L+M) &&
        (mobility->getCurrentPosition().y  > C_Y+L)){
        var=true;
    }

    return var;
}
```

```
bool TraCIDemo11p::activeGUpZone() {
    bool var    = false;

    if( (mobility->getCurrentPosition().y  <= C_Y+L+M) &&
        (mobility->getCurrentPosition().y  > C_Y+trueL)){
        var=true;
    }

    return var;
}


bool TraCIDemo11p::goingDownwards() {
    bool var    = false;
    if( mobility->getCurrentDirection().y > 0.87){
        var=true;
    }

    return var;
}

bool TraCIDemo11p::controlGDownZone() {
    bool var    = false;

    if( (mobility->getCurrentPosition().y  >= C_Y-L-M ) &&
        (mobility->getCurrentPosition().y  < C_Y-L  )){
        var=true;
    }

    return var;
}

bool TraCIDemo11p::activeGDownZone() {
    bool var    = false;

    if( (mobility->getCurrentPosition().y  >= C_Y-L-M ) &&
        (mobility->getCurrentPosition().y  < C_Y-trueL  )){
        var=true;
    }

    return var;
}


//It is true if the car is in control zone with the direction of
    //control
bool TraCIDemo11p::isControlZone() {
    bool var    = false;
    if( ( goingRigth() ) && ( controlGRightZone() ) )
        var =  true;
    else if ( ( goingLeft() ) && ( controlGLeftZone() ) )
        var =  true;
    else if ( ( goingUpwards() ) && ( controlGUpZone() ) )
        var =  true;
    else if ( ( goingDownwards() ) && ( controlGDownZone() ) )
        var =  true;

    if(var){
        EV << "Car is in Control Zone\n";
    }
    else
        EV <<"Car is not in Control Zone\n";

    return var;
}

bool TraCIDemo11p::isActiveZone() {
    bool var    = false;

    if( ( goingRigth() ) && ( activeGRightZone() ) )
        var =  true;
    else if ( ( goingLeft() ) && ( activeGLeftZone() ) )
        var =  true;
    else if ( ( goingUpwards() ) && ( activeGUpZone() ) )
        var =  true;
    else if ( ( goingDownwards() ) && ( activeGDownZone() ) )
        var =  true;

    if(var){
        EV << "Function: Car is in Active Zone\n";
    }
    else
        EV <<"Function: Car is not in Active Zone\n";

    return var;
}

////////////////////////////////////////////////////////////////
//Function for seeing if the receiving message is from a car
    //from different road

bool TraCIDemo11p::isDiffDirection(){
    bool var    = false;

    if(( goingRigth() )||( goingLeft() )){
        if(DirectionReceivedWSM[0]==0){
            EV << "Message Received is of Direction Y: "
                << DirectionReceivedWSM[1] << "\n";
            var = true;
        }
    }

    else if(( goingUpwards() )||( goingDownwards() )){
        if(DirectionReceivedWSM[1]==0){
            EV << "Message Received is of Direction X: "
                << DirectionReceivedWSM[0] << "\n";
            var = true;
        }
    }

    return var;
}

//If a message is scheduled, cancel it.
void TraCIDemo11p::cancelSMessage(cMessage* msg){
    if (msg->isScheduled()){
        cancelEvent(msg);
    }
}

//Function for initializing and sending message type 2
void TraCIDemo11p::startCounter() {
    EV <<"Sending Message for Counting Stop Car\n";
    counterStopCar++;

    wsm = new WaveShortMessage();
    populateWSM(wsm);

    wsm->setCcmType(2);
    wsm->setTimestamp(timeAdaptive);

    if( goingRigth() ){                          //Going to rigth
        wsm->setDirection(0,1);
        wsm->setDirection(1,0);
    }
    else if( goingLeft() ){
        wsm->setDirection(0,-1);                 //Going Left
        wsm->setDirection(1,0);
    }
    else if( goingDownwards() ){                 //Going Down
        wsm->setDirection(0,0);
        wsm->setDirection(1,1);
    }
```

```
    else if( goingUpwards() ){                         //Going Up
        wsm->setDirection(0,0);
        wsm->setDirection(1,-1);
    }

    sendDelayedNormalD(wsm);
    wsm = nullptr;

}

//Function for initializing and sending message type 4
void TraCIDemo11p::normalCycleTraffic() {
    EV <<"Sending Message for full Cycle Control\n";

    messageActivation   =   3;

    numActivation++;
    timeAdaptive=simTime();
    semaphoreState  =   2;
    counterStopCar=0;

    wsm = new WaveShortMessage();
    populateWSM(wsm);

    wsm->setCarId(carIdOwn);

    wsm->setCcmType(4);
    wsm->setTimestamp(timeAdaptive);

    wsm->setSemaphoreState(semaphoreState);
    wsm->setCountMessage(countChanges);

    if( goingRigth() ){                                //Going to rigth
        wsm->setDirection(0,1);
        wsm->setDirection(1,0);
    }
    else if( goingLeft() ){
        wsm->setDirection(0,-1);                       //Going Left
        wsm->setDirection(1,0);
    }
    else if( goingDownwards() ){                       //Going Down
        wsm->setDirection(0,0);
        wsm->setDirection(1,1);
    }
    else if( goingUpwards() ){                         //Going Up
        wsm->setDirection(0,0);
        wsm->setDirection(1,-1);
    }
    sendDelayedNormalD(wsm);
    wsm = nullptr;

    timeActivation  =   simTime();

    changeStateVector.recordWithTimestamp(simTime(),semaphoreState);

    scheduleAt(simTime()+tYellow, checkState);
}

//Function for initializing and sending message type 3
void TraCIDemo11p::verifyEndControl() {
    EV <<"Sending Message for Veryfing ctrl\n";

    wsm = new WaveShortMessage();
    populateWSM(wsm);

    wsm->setCcmType(3);

    if( goingRigth() ){                          //Going to rigth
        wsm->setDirection(0,1);
        wsm->setDirection(1,0);
    }
    else if( goingLeft() ){
        wsm->setDirection(0,-1);                 //Going Left
        wsm->setDirection(1,0);
    }
    else if( goingDownwards() ){                 //Going Down
        wsm->setDirection(0,0);
        wsm->setDirection(1,1);
    }
    else if( goingUpwards() ){                   //Going Up
        wsm->setDirection(0,0);
        wsm->setDirection(1,-1);
    }

    sendDelayedNormalD(wsm);
    wsm = nullptr;

    scheduleAt(simTime()+checkEnd, checkWaiting);
}
```

Annex D - Modified Comunication Network

In order to run the simulation it is necessary to add a random delay in the way that the car or R.S.U. send messages. This is done by modifying the files BaseWaveApplLayer.h and BaseWaveApplLayer.cc. In the thesis, it was added the function

```
virtual void sendDelayedNormalD(cMessage* msg);
```

That send a message of the type WSM/BSM/WSA with a normal delay. This delay is modified directly in inside the function in BaseWaveApplLayer.cc.

BaseWaveApplLayer.h

Location of the files is:

```
...\src\veins\modules\application\ieee80211p\BaseWaveApplLayer.h
```

The code is:

```
// Based on the original code by David Eckhoff
// Copyright (C) 2016 David Eckhoff <eckhoff@cs.fau.de>
//
// Documentation for these modules is at http://veins.car2x.org/
//
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation; either version 2 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
USA
//

#ifndef BASEWAVEAPPLLAYER_H_
#define BASEWAVEAPPLLAYER_H_

#include <random>

#include <map>
#include "veins/base/modules/BaseApplLayer.h"
#include "veins/modules/utility/Consts80211p.h"
#include "veins/modules/messages/WaveShortMessage_m.h"
```

```
#include "veins/modules/messages/WaveServiceAdvertisement_m.h"
#include "veins/modules/messages/BasicSafetyMessage_m.h"
#include "veins/base/connectionManager/ChannelAccess.h"
#include "veins/modules/mac/ieee80211p/WaveAppToMac1609_4Interface.h"
#include "veins/modules/mobility/traci/TraCIMobility.h"
#include "veins/modules/mobility/traci/TraCICommandInterface.h"

using Veins::TraCIMobility;
using Veins::TraCICommandInterface;
using Veins::AnnotationManager;
using Veins::TraCIMobilityAccess;
using Veins::AnnotationManagerAccess;

//#define DBG_APP std::cerr << "[" << simTime().raw() << "] "
//    << getParentModule()->getFullPath() << " "

#ifndef DBG_APP
#define DBG_APP EV
#endif

/**
 * @brief
 * WAVE application layer base class.
 *
 * @author David Eckhoff
 *
 * @ingroup applLayer
 *
 * @see BaseWaveApplLayer
 * @see Mac1609_4
 * @see PhyLayer80211p
 * @see Decider80211p
 */
class BaseWaveApplLayer : public BaseApplLayer {

    public:
        ~BaseWaveApplLayer();
        virtual void initialize(int stage);
        virtual void finish();

        virtual void receiveSignal(cComponent* source,
            simsignal_t signalID,
                cObject* obj, cObject* details);

        enum WaveApplMessageKinds {
            SEND_BEACON_EVT,
            SEND_WSA_EVT
        };
```

```
protected:

    static const simsignalwrap_t mobilityStateChangedSignal;
    static const simsignalwrap_t parkingStateChangedSignal;

    /** @brief handle messages from below and calls the onWSM,
        onBSM, and onWSA functions accordingly */
    virtual void handleLowerMsg(cMessage* msg);

    /** @brief handle self messages */
    virtual void handleSelfMsg(cMessage* msg);

    /** @brief sets all the necessary fields in the WSM,
        BSM, or WSA. */
    virtual void populateWSM(WaveShortMessage*  wsm,
        int rcvId=-1, int serial=0);

    /** @brief this function is called upon receiving a
        WaveShortMessage */
    virtual void onWSM(WaveShortMessage* wsm) { };

    /** @brief this function is called upon receiving
        a BasicSafetyMessage, also referred to as a beacon  */
    virtual void onBSM(BasicSafetyMessage* bsm) { };

    /** @brief this function is called upon receiving a
        WaveServiceAdvertisement */
    virtual void onWSA(WaveServiceAdvertisment* wsa) { };

    /** @brief this function is called every time the
        vehicle receives a position update signal */
    virtual void handlePositionUpdate(cObject* obj);

    /** @brief this function is called every time the vehicle
        parks or starts moving again */
    virtual void handleParkingUpdate(cObject* obj);

    /** @brief This will start the periodic advertising of
        the new service on the CCH
     *
     *  @param channel the channel on which the service is provided
     *  @param serviceId a service ID to be used with the service
     *  @param serviceDescription a literal description of the
        service
     */
    virtual void startService(Channels::ChannelNumber channel,
        int serviceId, std::string serviceDescription);
```

```
/** @brief stopping the service and advertising for it */
virtual void stopService();

/** @brief compute a point in time that is guaranteed to
    be in the correct channel interval plus a random offset
 *
 * @param interval the interval length of the periodic message
 * @param chantype the type of channel, either type_CCH or
    type_SCH
 */
virtual simtime_t computeAsynchronousSendingTime(
    simtime_t interval, t_channel chantype);

/**
 * @brief overloaded for error handling and stats recording
    purposes
 *
 * @param msg the message to be sent. Must be a WSM/BSM/WSA
 */
virtual void sendDown(cMessage* msg);

/**
 * @brief overloaded for error handling and stats recording
    purposes
 *
 * @param msg the message to be sent. Must be a WSM/BSM/WSA
 * @param delay the delay for the message
 */
virtual void sendDelayedDown(cMessage* msg, simtime_t delay);


/**
 * @brief overloaded for error handling and stats recording
    purposes
 *
 * @param msg the message to be sent. Must be a WSM/BSM/WSA
 */
virtual void sendDelayedNormalD(cMessage* msg);


/**
 * @brief helper function for error handling and stats
    recording purposes
 *
 * @param msg the message to be checked and tracked
 */
virtual void checkAndTrackPacket(cMessage* msg);
```

```
protected:

    /* pointers ill be set when used with TraCIMobility */
    TraCIMobility* mobility;
    TraCICommandInterface* traci;
    TraCICommandInterface::Vehicle* traciVehicle;

    AnnotationManager* annotations;
    WaveAppToMac1609_4Interface* mac;

    /* support for parking currently only works with TraCI */
    bool isParked;
    bool communicateWhileParked;

    /* BSM (beacon) settings */
    uint32_t beaconLengthBits;
    uint32_t  beaconUserPriority;
    simtime_t beaconInterval;
    bool sendBeacons;

    /* WSM (data) settings */
    uint32_t  dataLengthBits;
    uint32_t  dataUserPriority;
    bool dataOnSch;

    /* WSA settings */
    int currentOfferedServiceId;
    std::string currentServiceDescription;
    Channels::ChannelNumber currentServiceChannel;
    simtime_t wsaInterval;

    /* state of the vehicle */
    Coord curPosition;
    Coord curSpeed;
    int myId;
    int mySCH;

    /* stats */
    uint32_t generatedWSMs;
    uint32_t generatedWSAs;
    uint32_t generatedBSMs;
    uint32_t receivedWSMs;
    uint32_t receivedWSAs;
    uint32_t receivedBSMs;

    /* messages for periodic events such as beacon and WSA
        transmissions */
        cMessage* sendBeaconEvt;
        cMessage* sendWSAEvt;
};

#endif /* BASEWAVEAPPLLAYER_H_ */
```

BaseWaveApplLayer.cc

Location of the files is:

`...\src\veins\modules\application\ieee80211p\BaseWaveApplLayer.cc`

The code is:

```
// Based on the original code by David Eckhoff
// Copyright (C) 2011 David Eckhoff <eckhoff@cs.fau.de>
//
// Documentation for these modules is at http://veins.car2x.org/
//
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation; either version 2 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
USA
//

#include "veins/modules/application/ieee80211p/BaseWaveApplLayer.h"

const simsignalwrap_t BaseWaveApplLayer::mobilityStateChangedSignal =
    simsignalwrap_t(MIXIM_SIGNAL_MOBILITY_CHANGE_NAME);
const simsignalwrap_t BaseWaveApplLayer::parkingStateChangedSignal =
    simsignalwrap_t(TRACI_SIGNAL_PARKING_CHANGE_NAME);

void BaseWaveApplLayer::initialize(int stage) {
    BaseApplLayer::initialize(stage);

    if (stage==0) {

        //initialize pointers to other modules
```

```
if (FindModule<TraCIMobility*>::findSubModule(
    getParentModule()) ) {
    mobility = TraCIMobilityAccess().get(getParentModule());
    traci = mobility->getCommandInterface();
    traciVehicle = mobility->getVehicleCommandInterface();
}
else {
    traci = NULL;
    mobility = NULL;
    traciVehicle = NULL;
}

annotations = AnnotationManagerAccess().getIfExists();
ASSERT(annotations);

mac = FindModule<WaveAppToMac1609_4Interface*>::findSubModule(
        getParentModule() );
assert(mac);

myId = getParentModule()->getId();

//read parameters
headerLength = par("headerLength").longValue();
sendBeacons = par("sendBeacons").boolValue();
beaconLengthBits = par("beaconLengthBits").longValue();
beaconUserPriority = par("beaconUserPriority").longValue();
beaconInterval =  par("beaconInterval");

dataLengthBits = par("dataLengthBits").longValue();
dataOnSch = par("dataOnSch").boolValue();
dataUserPriority = par("dataUserPriority").longValue();

wsaInterval = par("wsaInterval").doubleValue();
communicateWhileParked = par("
    communicateWhileParked").boolValue();
currentOfferedServiceId = -1;

isParked = false;


findHost()->subscribe(mobilityStateChangedSignal, this);
findHost()->subscribe(parkingStateChangedSignal, this);

sendBeaconEvt = new cMessage("beacon evt", SEND_BEACON_EVT);
sendWSAEvt = new cMessage("wsa evt", SEND_WSA_EVT);

generatedBSMs = 0;
generatedWSAs = 0;
```

```
        generatedWSMs = 0;
        receivedBSMs = 0;
        receivedWSAs = 0;
        receivedWSMs = 0;
    }
    else if (stage == 1) {
        //simulate asynchronous channel access

        if (dataOnSch == true && !mac->isChannelSwitchingActive()) {
            dataOnSch = false;
            std::cerr << "App wants to send data on SCH but MAC
                doesn't use any SCH. Sending all data on CCH"
                    << std::endl;
        }
        simtime_t firstBeacon = simTime();

        if (par("avoidBeaconSynchronization").boolValue() == true) {

            simtime_t randomOffset = dblrand() * beaconInterval;
            firstBeacon = simTime() + randomOffset;

            if (mac->isChannelSwitchingActive() == true) {
                if ( beaconInterval.raw() %
                    (mac->getSwitchingInterval().raw()*2)) {
                    std::cerr << "The beacon interval ("
                        << beaconInterval << ") is smaller than or
                            not a multiple of  one synchronization
                                interval (" << 2*mac ->
                                    getSwitchingInterval() << "). "
                            << "This means that beacons are generated
                                during SCH intervals" << std::endl;
                }
                firstBeacon = computeAsynchronousSendingTime(
                    beaconInterval, type_CCH);
            }

            if (sendBeacons) {
                scheduleAt(firstBeacon, sendBeaconEvt);
            }
        }
    }
}

simtime_t BaseWaveApplLayer::computeAsynchronousSendingTime(
    simtime_t interval, t_channel chan) {

    /*
     * avoid that periodic messages for one channel type are
```

```
        scheduled in the other channel interval
     * when alternate access is enabled in the MAC
     */

    simtime_t randomOffset = dblrand() * beaconInterval;
    simtime_t firstEvent;
    simtime_t switchingInterval = mac->getSwitchingInterval();
        //usually 0.050s
    simtime_t nextCCH;

    /*
     * start event earliest in next CCH (or SCH) interval.
        For alignment, first find the next CCH interval
     * To find out next CCH, go back to start of current interval
        and add two or one intervals
     * depending on type of current interval
     */

    if (mac->isCurrentChannelCCH()) {
        nextCCH = simTime() - SimTime().setRaw(simTime().raw() %
            switchingInterval.raw()) + switchingInterval*2;
    }
    else {
        nextCCH = simTime() - SimTime().setRaw(simTime().raw() %
            switchingInterval.raw()) + switchingInterval;
    }

    firstEvent = nextCCH + randomOffset;

    //check if firstEvent lies within the correct interval and,
        if not, move to previous interval

    if (firstEvent.raw() % (2*switchingInterval.raw()) >
        switchingInterval.raw()) {
        //firstEvent is within a sch interval
        if (chan == type_CCH) firstEvent -= switchingInterval;
    }
    else {
        //firstEvent is within a cch interval, so adjust for SCH
            //messages
        if (chan == type_SCH) firstEvent += switchingInterval;
    }

    return firstEvent;
}

void BaseWaveApplLayer::populateWSM(WaveShortMessage* wsm,
    int rcvId, int serial) {
```

```
    wsm->setWsmVersion(1);
    wsm->setTimestamp(simTime());
    wsm->setSenderAddress(myId);
    wsm->setRecipientAddress(rcvId);
    wsm->setSerial(serial);
    wsm->setBitLength(headerLength);


    if (BasicSafetyMessage* bsm = dynamic_cast<BasicSafetyMessage*>(
        wsm) ) {
        bsm->setSenderPos(curPosition);
        bsm->setSenderPos(curPosition);
        bsm->setSenderSpeed(curSpeed);
        bsm->setPsid(-1);
        bsm->setChannelNumber(Channels::CCH);
        bsm->addBitLength(beaconLengthBits);
        wsm->setUserPriority(beaconUserPriority);
    }
    else if (WaveServiceAdvertisment* wsa =
        dynamic_cast<WaveServiceAdvertisment*>(wsm)) {
        wsa->setChannelNumber(Channels::CCH);
        wsa->setTargetChannel(currentServiceChannel);
        wsa->setPsid(currentOfferedServiceId);
        wsa->setServiceDescription(currentServiceDescription.c_str());
    }
    else {
        //will be rewritten at Mac1609_4 to actual Service Channel.
            //This is just so no controlInfo is needed
        if (dataOnSch) wsm->setChannelNumber(Channels::SCH1);
        else wsm->setChannelNumber(Channels::CCH);
        wsm->addBitLength(dataLengthBits);
        wsm->setUserPriority(dataUserPriority);
    }
}

void BaseWaveApplLayer::receiveSignal(cComponent* source,
    simsignal_t signalID, cObject* obj, cObject* details) {
    Enter_Method_Silent();
    if (signalID == mobilityStateChangedSignal) {
        handlePositionUpdate(obj);
    }
    else if (signalID == parkingStateChangedSignal) {
        handleParkingUpdate(obj);
    }
}

void BaseWaveApplLayer::handlePositionUpdate(cObject* obj) {
    ChannelMobilityPtrType const mobility =
        check_and_cast<ChannelMobilityPtrType>(obj);
    curPosition = mobility->getCurrentPosition();
    curSpeed = mobility->getCurrentSpeed();
}

void BaseWaveApplLayer::handleParkingUpdate(cObject* obj) {
    //this code should only run when used with TraCI
    isParked = mobility->getParkingState();
    if (communicateWhileParked == false) {
        if (isParked == true) {
            (FindModule<BaseConnectionManager*>::findGlobalModule())
                ->unregisterNic(this->getParentModule()->
                    getSubmodule("nic"));
        }
        else {
            Coord pos = mobility->getCurrentPosition();
            (FindModule<BaseConnectionManager*>::findGlobalModule())
                ->registerNic(this->getParentModule()->getSubmodule("
                    nic"),(ChannelAccess*) this->getParentModule()->
                        getSubmodule("nic")->getSubmodule("phy80211p"),
                            &pos);
        }
    }
}

void BaseWaveApplLayer::handleLowerMsg(cMessage* msg) {

    WaveShortMessage* wsm = dynamic_cast<WaveShortMessage*>(
        msg);
    ASSERT(wsm);

    if (BasicSafetyMessage* bsm = dynamic_cast<BasicSafetyMessage*>(
        wsm)) {
        receivedBSMs++;
        onBSM(bsm);
    }
    else if (WaveServiceAdvertisment* wsa =
        dynamic_cast<WaveServiceAdvertisment*>(wsm)) {
        receivedWSAs++;
        onWSA(wsa);
    }
    else {
        receivedWSMs++;
        onWSM(wsm);
    }

    delete(msg);
```

```
}

void BaseWaveApplLayer::handleSelfMsg(cMessage* msg) {
    switch (msg->getKind()) {
    case SEND_BEACON_EVT: {
        BasicSafetyMessage* bsm = new BasicSafetyMessage();
        populateWSM(bsm);
        sendDown(bsm);
        scheduleAt(simTime() + beaconInterval, sendBeaconEvt);
        break;
    }
    case SEND_WSA_EVT:   {
        WaveServiceAdvertisment* wsa = new WaveServiceAdvertisment();
        populateWSM(wsa);
        sendDown(wsa);
        scheduleAt(simTime() + wsaInterval, sendWSAEvt);
        break;
    }
    default: {
        if (msg)
            DBG_APP << "APP: Error: Got Self Message of unknown
                kind! Name: " << msg->getName() << endl;
        break;
    }
    }
}

void BaseWaveApplLayer::finish() {
    recordScalar("generatedWSMs",generatedWSMs);
    recordScalar("receivedWSMs",receivedWSMs);

    recordScalar("generatedBSMs",generatedBSMs);
    recordScalar("receivedBSMs",receivedBSMs);

    recordScalar("generatedWSAs",generatedWSAs);
    recordScalar("receivedWSAs",receivedWSAs);
}

BaseWaveApplLayer::~BaseWaveApplLayer() {
    cancelAndDelete(sendBeaconEvt);
    cancelAndDelete(sendWSAEvt);
    findHost()->unsubscribe(mobilityStateChangedSignal, this);
}

void BaseWaveApplLayer::startService(Channels::ChannelNumber channel,
    int serviceId, std::string serviceDescription) {
    if (sendWSAEvt->isScheduled()) {
        error("Starting service although another service
            was already started");
    }

    mac->changeServiceChannel(channel);
    currentOfferedServiceId = serviceId;
    currentServiceChannel = channel;
    currentServiceDescription = serviceDescription;

    simtime_t wsaTime = computeAsynchronousSendingTime(
        wsaInterval, type_CCH);
    scheduleAt(wsaTime, sendWSAEvt);

}

void BaseWaveApplLayer::stopService() {
    cancelEvent(sendWSAEvt);
    currentOfferedServiceId = -1;
}

void BaseWaveApplLayer::sendDown(cMessage* msg) {
    checkAndTrackPacket(msg);
    BaseApplLayer::sendDown(msg);
}

void BaseWaveApplLayer::sendDelayedDown(cMessage* msg,
    simtime_t delay) {
    checkAndTrackPacket(msg);
    BaseApplLayer::sendDelayedDown(msg, delay);
}

void BaseWaveApplLayer::sendDelayedNormalD(cMessage* msg) {


    checkAndTrackPacket(msg);
    double  distribution =  normal(0.7,0.1);
    if (distribution < -0.2){
        distribution    =   0;
    }
    BaseApplLayer::sendDelayedDown(msg, 0.3+distribution );
}

void BaseWaveApplLayer::checkAndTrackPacket(cMessage* msg) {
    if (isParked && !communicateWhileParked) error("
        Attempted to transmit a message while parked, but
            this is forbidden by current configuration");

    if (dynamic_cast<BasicSafetyMessage*>(msg)) {
        DBG_APP << "sending down a BSM" << std::endl;
        generatedBSMs++;
    }
    else if (dynamic_cast<WaveServiceAdvertisment*>(msg)) {
        DBG_APP << "sending down a WSA" << std::endl;
        generatedWSAs++;
    }
    else if (dynamic_cast<WaveShortMessage*>(msg)) {
        DBG_APP << "sending down a wsm" << std::endl;
        generatedWSMs++;
    }
}
```

Annex E - Message Constructor

The content of the different messages that the cars and the R.S.U. can send and receive in OMNETT++ are defined on a constructor file, that also create the methods to assign and read this content. For this reason, only the constructor code will be given now. The location of the file is:

`...\src\veins\modules\messages\WaveShortMessage.msg`

The code is:

```
// Based on the code of David Eckhoff
// Copyright (C) 2011 David Eckhoff <eckhoff@cs.fau.de>
//
// Documentation for these modules is at http://veins.car2x.org/
//
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation; either version 2 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
USA
//

packet WaveShortMessage {
        //Version of the Wave Short Message
        int wsmVersion = 0;
        //Determine which security mechanism was used
        int securityType = 0;
        //Channel Number on which this packet was sent
        int channelNumber;
        //Data rate with which this packet was sent
        int dataRate = 1;
        //User priority with which this packet was sent
            //(note the AC mapping rules in Mac1609_4::mapUserPriority)
        int userPriority = 7;
        //Unique number to identify the service
        int psid = 0;
        //Provider Service Context
        string psc = "Service with some Data";
        //Length of Wave Short Message
        int wsmLength;
        //Data of Wave Short Message
        string wsmData="Some Data";

        int       semaphoreState;
        int       direction[2];
        int       ccmType;
        int       countMessage;
        int       carId;

        int senderAddress = 0;
        int recipientAddress = -1;
        int serial = 0;
        simtime_t timestamp = 0;
}
```

Annex F - Test Cases

All the different test are build in the same manner, with a document that describe the type of cars, the possible routed that the cars can follow and the flow to generate. At continuation there are presented the different test used for the thesis, where the location of the file is:

```
...\examples\veins\erlangen.rou.xml
```

Test 0

```
<routes>
<vType color="1,1,0" maxSpeed="14" minGap="2.5" length="2.5" sigma="0.5"
    decel="4.5" accel="2.6" impatience="off" id="vtype0"/>

<route id="10" edges="L1 E3"/>
<route id="12" edges="L1 E2"/>

<route id="20" edges="L2 E4"/>
<route id="22" edges="L2 E3"/>

<route id="30" edges="L3 E1"/>
<route id="32" edges="L3 E4"/>

<route id="40" edges="L4 E2"/>
<route id="42" edges="L4 E1"/>


<flow id="flow201" number="25" period="4" begin="0" route="20" type="vtype0"/>
<flow id="flow101" number="25" period="5" begin="0.5" route="10" type="vtype0"/>
<flow id="flow401" number="25" period="5" begin="1" route="40" type="vtype0"/>
<flow id="flow301" number="25" period="4" begin="1.5" route="30" type="vtype0"/>
</routes>
```

Test 1

```
<routes>
<vType color="1,1,0" maxSpeed="14" minGap="2.5" length="2.5" sigma="0.5"
    decel="4.5" accel="2.6" impatience="off" id="vtype0"/>

<route id="10" edges="L1 E3"/>
<route id="12" edges="L1 E2"/>

<route id="20" edges="L2 E4"/>
<route id="22" edges="L2 E3"/>

<route id="30" edges="L3 E1"/>
<route id="32" edges="L3 E4"/>
```

```
<route id="40" edges="L4 E2"/>
<route id="42" edges="L4 E1"/>


<flow id="flow101" number="10" period="5" begin="0" route="10" type="vtype0"/>
<flow id="flow201" number="10" period="5" begin="0.5" route="20" type="vtype0"/>
<flow id="flow301" number="10" period="6" begin="1" route="30" type="vtype0"/>
<flow id="flow401" number="10" period="6" begin="1.5" route="40" type="vtype0"/>

<flow id="flow121" number="10" period="6" begin="10" route="12" type="vtype0"/>
<flow id="flow221" number="10" period="6" begin="11" route="22" type="vtype0"/>
<flow id="flow321" number="10" period="5" begin="12" route="32" type="vtype0"/>
<flow id="flow421" number="10" period="5" begin="13" route="42" type="vtype0"/>

<flow id="flow102" number="25" period="6" begin="60" route="10" type="vtype0"/>
<flow id="flow202" number="25" period="6" begin="60.5" route="20" type="vtype0"/>
<flow id="flow302" number="25" period="6" begin="61" route="30" type="vtype0"/>
<flow id="flow402" number="25" period="6" begin="61.5" route="40" type="vtype0"/>

<flow id="flow122" number="15" period="5" begin="70" route="12" type="vtype0"/>
<flow id="flow222" number="15" period="5" begin="70.5" route="22" type="vtype0"/>
<flow id="flow322" number="15" period="5" begin="71" route="32" type="vtype0"/>
<flow id="flow422" number="15" period="5" begin="71.5" route="42" type="vtype0"/>
</routes>
```

Test 2

```
<routes>
<vType color="1,1,0" maxSpeed="14" minGap="2.5" length="2.5" sigma="0.5"
    decel="4.5" accel="2.6" impatience="off" id="vtype0"/>

<route id="10" edges="L1 E3"/>
<route id="12" edges="L1 E2"/>

<route id="20" edges="L2 E4"/>
<route id="22" edges="L2 E3"/>

<route id="30" edges="L3 E1"/>
<route id="32" edges="L3 E4"/>

<route id="40" edges="L4 E2"/>
<route id="42" edges="L4 E1"/>


<flow id="flow101" number="15" period="4" begin="0" route="10" type="vtype0"/>
<flow id="flow201" number="17" period="5" begin="0" route="20" type="vtype0"/>
```

```
<flow id="flow301" number="18" period="6" begin="2" route="30" type="vtype0"/>
<flow id="flow401" number="10" period="3" begin="5" route="40" type="vtype0"/>

<flow id="flow121" number="10" period="10" begin="10" route="12" type="vtype0"/>
<flow id="flow321" number="13" period="7" begin="15" route="32" type="vtype0"/>

<flow id="flow102" number="10" period="8" begin="20" route="10" type="vtype0"/>
<flow id="flow202" number="14" period="10" begin="22" route="20" type="vtype0"/>
<flow id="flow302" number="13" period="5" begin="30" route="30" type="vtype0"/>
<flow id="flow402" number="12" period="4" begin="35" route="40" type="vtype0"/>

<flow id="flow221" number="8" period="5" begin="40" route="22" type="vtype0"/>
<flow id="flow421" number="9" period="7" begin="50" route="42" type="vtype0"/>
</routes>
```

Test 3

```
<routes>
<vType color="1,1,0" maxSpeed="14" minGap="2.5" length="2.5" sigma="0.5"
    decel="4.5" accel="2.6" impatience="off" id="vtype0"/>

<route id="10" edges="L1 E3"/>
<route id="12" edges="L1 E2"/>

<route id="20" edges="L2 E4"/>
<route id="22" edges="L2 E3"/>

<route id="30" edges="L3 E1"/>
<route id="32" edges="L3 E4"/>

<route id="40" edges="L4 E2"/>
<route id="42" edges="L4 E1"/>


<flow id="flow201" number="30" period="5" begin="0" route="20" type="vtype0"/>
<flow id="flow401" number="30" period="5" begin="0" route="40" type="vtype0"/>

<flow id="flow102" number="8" period="10" begin="20" route="10" type="vtype0"/>
<flow id="flow321" number="8" period="8" begin="30" route="32" type="vtype0"/>



</routes>
```

Annex G - Map for base case

The map used for the simulation is created in NETEDIT to be used in SUMO. The coordinates used in NETEDIT for the creation of the map doesn't correspond to the one used in OMNET++, this are done indepently. The location of the file is:

```
...\examples\veins\erlangen.net.xml
```

Even if the file is done externally, it can be created in text format, in which in this case is presented. The code is:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on 5/13/2019 11:02:13 PM by Netedit Version 0.32.0
<?xml version="1.0" encoding="UTF-8"?>

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=
        "http://sumo.dlr.de/xsd/netconvertConfiguration.xsd">

    <input>
        <sumo-net-file value=
            "C:\Users\gi_ma\src\veins-4.7.1\examples\veins\erlangen.net.xml"/>
    </input>

    <output>
        <output-file value=
            "C:\Users\gi_ma\src\veins-4.7.1\examples\veins\erlangen.net.xml"/>
    </output>

    <processing>
        <no-turnarounds value="true"/>
        <offset.disable-normalization value="true"/>
        <lefthand value="false"/>
        <junctions.corner-detail value="0"/>
        <rectangular-lane-cut value="false"/>
        <walkingareas value="false"/>
    </processing>

</configuration>
-->

<net version="0.27" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/net_file.xsd">

    <location netOffset="0.00,0.00" convBoundary="0.00,-1805.20,1805.20,0.00"
        origBoundary=
        "-10000000000.00,-10000000000.00,10000000000.00,10000000000.00"
            projParameter="!"/>
```

```
<edge id=":o_0" function="internal">
    <lane id=":o_0_0" index="0" speed="15.00" length="5.24" width="3.50"
        shape="897.20,-893.95 897.00,-895.37 896.39,
            -896.39 895.37,-897.00 893.95,-897.20"/>
</edge>
<edge id=":o_1" function="internal">
    <lane id=":o_1_0" index="0" speed="15.00" length="8.35" width="3.50"
        shape="897.20,-893.95 897.00,-896.95 896.39,
            -899.09 895.37,-900.37 893.95,-900.80"/>
</edge>
<edge id=":o_2" function="internal">
    <lane id=":o_2_0" index="0" speed="15.00" length="17.30" width="3.50"
        shape="897.20,-893.95 897.20,-911.25"/>
    <lane id=":o_2_1" index="1" speed="15.00" length="17.30" width="3.50"
        shape="900.80,-893.95 900.80,-911.25"/>
</edge>
<edge id=":o_4" function="internal">
    <lane id=":o_4_0" index="0" speed="15.00" length="5.24" width="3.50"
        shape="911.25,-897.20 909.83,-897.00 908.81,
            -896.39 908.20,-895.37 908.00,-893.95"/>
</edge>
<edge id=":o_5" function="internal">
    <lane id=":o_5_0" index="0" speed="15.00" length="8.35" width="3.50"
        shape="911.25,-897.20 908.25,-897.00 906.11,
            -896.39 904.83,-895.37 904.40,-893.95"/>
</edge>
<edge id=":o_6" function="internal">
    <lane id=":o_6_0" index="0" speed="15.00" length="17.30" width="3.50"
        shape="911.25,-897.20 893.95,-897.20"/>
    <lane id=":o_6_1" index="1" speed="15.00" length="17.30" width="3.50"
        shape="911.25,-900.80 893.95,-900.80"/>
</edge>
<edge id=":o_8" function="internal">
    <lane id=":o_8_0" index="0" speed="15.00" length="5.24" width="3.50"
        shape="908.00,-911.25 908.20,-909.83 908.81,
            -908.81 909.83,-908.20 911.25,-908.00"/>
</edge>
<edge id=":o_9" function="internal">
    <lane id=":o_9_0" index="0" speed="15.00" length="8.35" width="3.50"
        shape="908.00,-911.25 908.20,-908.25 908.81,
            -906.11 909.83,-904.83 911.25,-904.40"/>
</edge>
<edge id=":o_10" function="internal">
    <lane id=":o_10_0" index="0" speed="15.00" length="17.30" width="3.50"
        shape="908.00,-911.25 908.00,-893.95"/>
    <lane id=":o_10_1" index="1" speed="15.00" length="17.30" width="3.50"
        shape="904.40,-911.25 904.40,-893.95"/>
```

```
</edge>
<edge id=":o_12" function="internal">
    <lane id=":o_12_0" index="0" speed="15.00" length="5.24" width="3.50"
        shape="893.95,-908.00 895.37,-908.20 896.39,
            -908.81 897.00,-909.83 897.20,-911.25"/>
</edge>
<edge id=":o_13" function="internal">
    <lane id=":o_13_0" index="0" speed="15.00" length="8.35" width="3.50"
        shape="893.95,-908.00 896.95,-908.20 899.09,
            -908.81 900.37,-909.83 900.80,-911.25"/>
</edge>
<edge id=":o_14" function="internal">
    <lane id=":o_14_0" index="0" speed="15.00" length="17.30" width="3.50"
        shape="893.95,-908.00 911.25,-908.00"/>
    <lane id=":o_14_1" index="1" speed="15.00" length="17.30" width="3.50"
        shape="893.95,-904.40 911.25,-904.40"/>
</edge>

<edge id="E1" from="o" to="h1" priority="1" length="900.00">
    <lane id="E1_0" index="0" speed="15.00" length="900.00" width="3.50"
        shape="911.25,-908.00 1805.20,-908.00"/>
    <lane id="E1_1" index="1" speed="15.00" length="900.00" width="3.50"
        shape="911.25,-904.40 1805.20,-904.40"/>
</edge>
<edge id="E2" from="o" to="v1" priority="1" length="900.00">
    <lane id="E2_0" index="0" speed="15.00" length="900.00" width="3.50"
        shape="908.00,-893.95 908.00,0.00"/>
    <lane id="E2_1" index="1" speed="15.00" length="900.00" width="3.50"
        shape="904.40,-893.95 904.40,0.00"/>
</edge>
<edge id="E3" from="o" to="h2" priority="1" length="900.00">
    <lane id="E3_0" index="0" speed="15.00" length="900.00" width="3.50"
        shape="893.95,-897.20 0.00,-897.20"/>
    <lane id="E3_1" index="1" speed="15.00" length="900.00" width="3.50"
        shape="893.95,-900.80 0.00,-900.80"/>
</edge>
<edge id="E4" from="o" to="v2" priority="1" length="900.00">
    <lane id="E4_0" index="0" speed="15.00" length="900.00" width="3.50"
        shape="897.20,-911.25 897.20,-1805.20"/>
    <lane id="E4_1" index="1" speed="15.00" length="900.00" width="3.50"
        shape="900.80,-911.25 900.80,-1805.20"/>
</edge>
<edge id="L1" from="h1" to="o" priority="1" length="900.00">
    <lane id="L1_0" index="0" speed="15.00" length="900.00" width="3.50"
        shape="1805.20,-897.20 911.25,-897.20"/>
    <lane id="L1_1" index="1" speed="15.00" length="900.00" width="3.50"
        shape="1805.20,-900.80 911.25,-900.80"/>
</edge>
```

```
<edge id="L2" from="v1" to="o" priority="1" length="900.00">
    <lane id="L2_0" index="0" speed="15.00" length="900.00" width="3.50"
        shape="897.20,0.00 897.20,-893.95"/>
    <lane id="L2_1" index="1" speed="15.00" length="900.00" width="3.50"
        shape="900.80,0.00 900.80,-893.95"/>
</edge>
<edge id="L3" from="h2" to="o" priority="1" length="900.00">
    <lane id="L3_0" index="0" speed="15.00" length="900.00" width="3.50"
        shape="0.00,-908.00 893.95,-908.00"/>
    <lane id="L3_1" index="1" speed="15.00" length="900.00" width="3.50"
        shape="0.00,-904.40 893.95,-904.40"/>
</edge>
<edge id="L4" from="v2" to="o" priority="1" length="900.00">
    <lane id="L4_0" index="0" speed="15.00" length="900.00" width="3.50"
        shape="908.00,-1805.20 908.00,-911.25"/>
    <lane id="L4_1" index="1" speed="15.00" length="900.00" width="3.50"
        shape="904.40,-1805.20 904.40,-911.25"/>
</edge>

<junction id="h1" type="dead_end" x="1805.20" y="-902.60"
    incLanes="E1_0 E1_1" intLanes=""
        shape="1805.20,-902.55 1805.20,-909.75 1805.20,-902.65"/>
<junction id="h2" type="dead_end" x="0.00" y="-902.60"
    incLanes="E3_0 E3_1" intLanes=""
        shape="0.00,-902.65 0.00,-895.45 0.00,-902.55"/>
<junction id="o" type="priority" x="902.60" y="-902.60"
    incLanes="L2_0 L2_1 L1_0 L1_1 L4_0 L4_1 L3_0 L3_1"
        intLanes=":o_0_0 :o_1_0 :o_2_0 :o_2_1 :o_4_0 :o_5_0 :o_6_0 :o_6_1
            :o_8_0 :o_9_0 :o_10_0 :o_10_1 :o_12_0 :o_13_0 :o_14_0 :o_14_1"
                shape="895.45,-893.95 909.75,-893.95 911.25,
                    -895.45 911.25,-909.75 909.75,-911.25 895.45,
                        -911.25 893.95,-909.75 893.95,-895.45">
    <request index="0" response="0000000000000000" foes="0000000011000000"
        cont="0"/>
    <request index="1" response="0000000000000000" foes="0000000011000000"
        cont="0"/>
    <request index="2" response="0000000000000000" foes="1111000011000000"
        cont="0"/>
    <request index="3" response="0000000000000000" foes="1111000011000000"
        cont="0"/>
    <request index="4" response="0000110000000000" foes="0000110000000000"
        cont="0"/>
    <request index="5" response="0000110000000000" foes="0000110000000000"
        cont="0"/>
    <request index="6" response="0000110000001111" foes="0000110000001111"
        cont="0"/>
    <request index="7" response="0000110000001111" foes="0000110000001111"
        cont="0"/>
    <request index="8" response="0000000000000000" foes="1100000000000000"
        cont="0"/>
    <request index="9" response="0000000000000000" foes="1100000000000000"
        cont="0"/>
    <request index="10" response="0000000000000000" foes="1100000011110000"
        cont="0"/>
    <request index="11" response="0000000000000000" foes="1100000011110000"
        cont="0"/>
    <request index="12" response="0000000000001100" foes="0000000000001100"
        cont="0"/>
    <request index="13" response="0000000000001100" foes="0000000000001100"
        cont="0"/>
    <request index="14" response="0000111100001100" foes="0000111100001100"
        cont="0"/>
    <request index="15" response="0000111100001100" foes="0000111100001100"
        cont="0"/>
</junction>
<junction id="v1" type="dead_end" x="902.60" y="0.00" incLanes="E2_0 E2_1"
    intLanes="" shape="902.55,0.00 909.75,0.00 902.65,0.00"/>
<junction id="v2" type="dead_end" x="902.60" y="-1805.20"
    incLanes="E4_0 E4_1" intLanes="" shape="902.65,-1805.20 895.45,
        -1805.20 902.55,-1805.20"/>

<connection from="L1" to="E2" fromLane="0" toLane="0" via=":o_4_0" dir="r"
    state="m"/>
<connection from="L1" to="E2" fromLane="0" toLane="1" via=":o_5_0" dir="r"
    state="m"/>
<connection from="L1" to="E3" fromLane="0" toLane="0" via=":o_6_0" dir="s"
    state="m"/>
<connection from="L1" to="E3" fromLane="1" toLane="1" via=":o_6_1" dir="s"
    state="m"/>
<connection from="L2" to="E3" fromLane="0" toLane="0" via=":o_0_0" dir="r"
    state="M"/>
<connection from="L2" to="E3" fromLane="0" toLane="1" via=":o_1_0" dir="r"
    state="M"/>
<connection from="L2" to="E4" fromLane="0" toLane="0" via=":o_2_0" dir="s"
    state="M"/>
<connection from="L2" to="E4" fromLane="1" toLane="1" via=":o_2_1" dir="s"
    state="M"/>
<connection from="L3" to="E4" fromLane="0" toLane="0" via=":o_12_0" dir="r"
    state="m"/>
<connection from="L3" to="E4" fromLane="0" toLane="1" via=":o_13_0" dir="r"
    state="m"/>
<connection from="L3" to="E1" fromLane="0" toLane="0" via=":o_14_0" dir="s"
    state="m"/>
<connection from="L3" to="E1" fromLane="1" toLane="1" via=":o_14_1" dir="s"
    state="m"/>
<connection from="L4" to="E1" fromLane="0" toLane="0" via=":o_8_0" dir="r"
```

```
        state="M"/>
    <connection from="L4" to="E1" fromLane="0" toLane="1" via=":o_9_0" dir="r"
        state="M"/>
    <connection from="L4" to="E2" fromLane="0" toLane="0" via=":o_10_0" dir="s"
        state="M"/>
    <connection from="L4" to="E2" fromLane="1" toLane="1" via=":o_10_1" dir="s"
        state="M"/>

    <connection from=":o_0" to="E3" fromLane="0" toLane="0" dir="r" state="M"/>
    <connection from=":o_1" to="E3" fromLane="0" toLane="1" dir="r" state="M"/>
    <connection from=":o_2" to="E4" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":o_2" to="E4" fromLane="1" toLane="1" dir="s" state="M"/>
    <connection from=":o_4" to="E2" fromLane="0" toLane="0" dir="r" state="M"/>
    <connection from=":o_5" to="E2" fromLane="0" toLane="1" dir="r" state="M"/>
    <connection from=":o_6" to="E3" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":o_6" to="E3" fromLane="1" toLane="1" dir="s" state="M"/>
    <connection from=":o_8" to="E1" fromLane="0" toLane="0" dir="r" state="M"/>
    <connection from=":o_9" to="E1" fromLane="0" toLane="1" dir="r" state="M"/>
    <connection from=":o_10" to="E2" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":o_10" to="E2" fromLane="1" toLane="1" dir="s" state="M"/>
    <connection from=":o_12" to="E4" fromLane="0" toLane="0" dir="r" state="M"/>
    <connection from=":o_13" to="E4" fromLane="0" toLane="1" dir="r" state="M"/>
    <connection from=":o_14" to="E1" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":o_14" to="E1" fromLane="1" toLane="1" dir="s" state="M"/>

</net>
```

Annex H - Building Parameters

The simulation of OMNeT++ and SUMO requires to input buildings in order to test the reflection of the message being transmitted over the air. Nonetheless, this only difficult the simulation being that it is not the end of this thesis, so it is replaced by a dot in the map.

The locations of the file that control this is:

```
...\examples\veins\erlangen.poly.xml
```

and the code of this file is:

```
<shapes>
  <poly id="poly0" type="building" color="1,0,0" fill="true"
    layer="-1" shape="195,45 195,45 195,45 195,45"/>
</shapes>
```

Annex I - OMNeT++ parameters

In order to run the simulation, it is necessary that the stage to be defined, and the model of the IEEE 802.11p to be described. For this it is necessary to modify two files. of Notice is that the stage is the same that the original one, but the R.S.U. are not initialized.

Simulation Parameters

The locations of the file that control this is:

`...\examples\veins\omnetpp.ini`

and the code of this file is:

```
[General]
cmdenv-express-mode = true
cmdenv-autoflush = true
cmdenv-status-frequency = 1s
**.cmdenv-log-level = info

ned-path = .
image-path = ../../images

network = RSUExampleScenario
**.*.vector-recording = true


##########################################################
#              Simulation parameters                     #
##########################################################
debug-on-errors = true
print-undisposed = true

sim-time-limit = 1800s

**.scalar-recording = true
**.vector-recording = true

**.debug = false
**.coreDebug = false

*.playgroundSizeX = 2000m
*.playgroundSizeY = 2000m
*.playgroundSizeZ = 50m


##########################################################
# Annotation parameters                                  #
##########################################################
```

```
*.annotations.draw = true

##########################################################
# Obstacle parameters                                     #
##########################################################
*.obstacles.debug = false
*.obstacles.obstacles = xmldoc("config.xml",
    "//AnalogueModel[@type='SimpleObstacleShadowing']/obstacles")

##########################################################
#            TraCIScenarioManager parameters              #
##########################################################
*.manager.updateInterval = 1s
*.manager.host = "localhost"
*.manager.port = 9999
*.manager.autoShutdown = true
*.manager.launchConfig = xmldoc("erlangen.launchd.xml")

##########################################################
#                       RSU SETTINGS                      #
#                                                         #
#                                                         #
##########################################################
*.rsu[0].mobility.x = 950
*.rsu[0].mobility.y = 950
*.rsu[0].mobility.z = 3

*.rsu[*].applType = "TraCIDemoRSU11p"
*.rsu[*].appl.headerLength = 80 bit
*.rsu[*].appl.sendBeacons = false
*.rsu[*].appl.dataOnSch = false
*.rsu[*].appl.beaconInterval = 1s
*.rsu[*].appl.beaconUserPriority = 7
*.rsu[*].appl.dataUserPriority = 5

##########################################################
#            11p specific parameters                      #
#                                                         #
#                    NIC-Settings                         #
##########################################################
*.connectionManager.sendDirect = true
*.connectionManager.maxInterfDist = 500m
*.connectionManager.drawMaxIntfDist = false

*.**.nic.mac1609_4.useServiceChannel = false

*.**.nic.mac1609_4.txPower = 20mW
*.**.nic.mac1609_4.bitrate = 6Mbps
```

```
*.**.nic.phy80211p.sensitivity = -89dBm

*.**.nic.phy80211p.useThermalNoise = true
*.**.nic.phy80211p.thermalNoise = -110dBm

*.**.nic.phy80211p.decider = xmldoc("config.xml")
*.**.nic.phy80211p.analogueModels = xmldoc("config.xml")
*.**.nic.phy80211p.usePropagationDelay = true

*.**.nic.phy80211p.antenna = xmldoc("antenna.xml",
    "/root/Antenna[@id='monopole']")

##########################################################
#                     WaveAppLayer                       #
##########################################################
*.node[*].applType = "TraCIDemo11p"
*.node[*].appl.headerLength = 80 bit
*.node[*].appl.sendBeacons = false
*.node[*].appl.dataOnSch = false
*.node[*].appl.beaconInterval = 20s

##########################################################
#                       Mobility                         #
##########################################################
*.node[*].veinsmobilityType.debug = true
*.node[*].veinsmobility.x = 0
*.node[*].veinsmobility.y = 0
*.node[*].veinsmobility.z = 1.895
*.node[*0].veinsmobility.accidentCount = 0
*.node[*0].veinsmobility.accidentStart = 15s
*.node[*0].veinsmobility.accidentDuration = 30s

[Config Default]
```

Stage Parameters

The locations of the file that control this is:

`...\examples\veins\RSUExampleScenario.ned`

and the code of this file is:

```
import org.car2x.veins.nodes.RSU;
import org.car2x.veins.nodes.Scenario;

network RSUExampleScenario extends Scenario
{
    submodules:
}
```