POLITECNICO DI TORINO

Master's degree in Mechatronic Engineering

Dynamic e-Traffic Light management using ITS systems

Supervisors:

Prof. Claudio Ettore Casetti

Eng. Mauro Scassa

Eng. Marco Rapelli

Candidate:

Marcello Sgarbi

Matriculation number: 236486

Acknowledgement

Before presenting the thesis project I would like to express my gratitude to the Prof. Claudio Casetti and Mauro Scassa to give me their trust for the development of this project.
A huge thanks is dedicated to my family that supported me during these years and that allowed me to do this master's degree and live this experience.
Last but not least, a huge thanks to all my friends and my girlfriend that helped me taking a break during the stressing periods and that gave me their suggestions and support.

Contents

List of Figures

List of Tables

List of Equations

Abstract

European Union's strategies to reduce traffic jams and reduce fatalities are constantly improved and they adopt all the newest technologies available to reach these targets. E-Mobility is having a key role trying to get this purpose, as pointed out in *Strategic Transport Research Innovation Agenda* (STRIA) [1].

In this context, the focus has been set on the congestion problem generated by traffic lights. This issue has been faced using vehicles driving into a Vehicular Ad Hoc Network (VANET), able to communicate using Vehicle-to-Vehicle (V2V) technology. SUMO simulator [2] has been used to develop the environment, while the simulation framework Veins (Vehicles in Network Simulation), based on OMNeT++ simulator [3], has been used to implement vehicles' networking.

To reduce traffic problem, a distributed heuristic algorithm has been designed. A bit-based representation has been used to mathematically describe the intersection and, using the data collected via V2V communication, search for the optimal solution. In addition, V2V allows vehicles to detect when they have the right to move into the intersection without producing hazardous situations.

To compare the result, the same intersection has been simulated using a traffic light whose phase's signal has been designed using Webster's method. The comparison shows that is worthy to use the developed algorithm to empty the intersection.

1. Introduction

According to worldwide statistics [4] and Italian ones [5], vehicle's market has been growing for few decades. The movement of a so high number of vehicles on the roads creates traffic jams and one of the main reasons for this phenomenon is that modifications to streets in urban areas are not always possible. Even if this problem seems to be getting worse, the automotive field improves year by year in performances, safety and on-board technology. Nowadays vehicles present high sophisticated electronic systems able to manage a big amount of data, but they also start being able to communicate with surrounding vehicles and infrastructure thanks to networking capability. Using this functionality, vehicles can now communicate with other vehicles or infrastructure in the VANET and, based on which is the receiver of the broadcast message from a vehicle, three main definitions have been created:

- Vehicle-to-Vehicle (V2V), when the communication is among vehicles;
- Vehicle-to-Infrastructure (V2I), when the communication happens among vehicles and surrounding infrastructure;
- Vehicle-to-Everything (V2X), referring to the communication among vehicles ad whatever able to interact with them.

V2X create new opportunities in developing strategies to orchestrate vehicles on the roads creating new possibility to reduce traffic jams and improve safety. Taking advantage from this, the presented project has been developed trying to reduce traffic jams at the intersection.

Before approaching the used technology an overview about safety and traffic has been presented allowing the reader to have a clear overview of the context in which this project has been developed.

1.1. Safety

The European Union pays great attention to the issues that concern road safety and it adopts as guideline a Sweden project, called Vision Zero [6], which aims to have zero fatalities on roads. The EU is constantly monitoring deaths on the road, but it is also classifying them by the type of vehicles involved. These statistics are collected mainly to have reliable data that can be used to develop safety strategies. In the last update of the document, it is possible to have an overview of the road deaths from 1991 to 2010 (*Figure 1*).

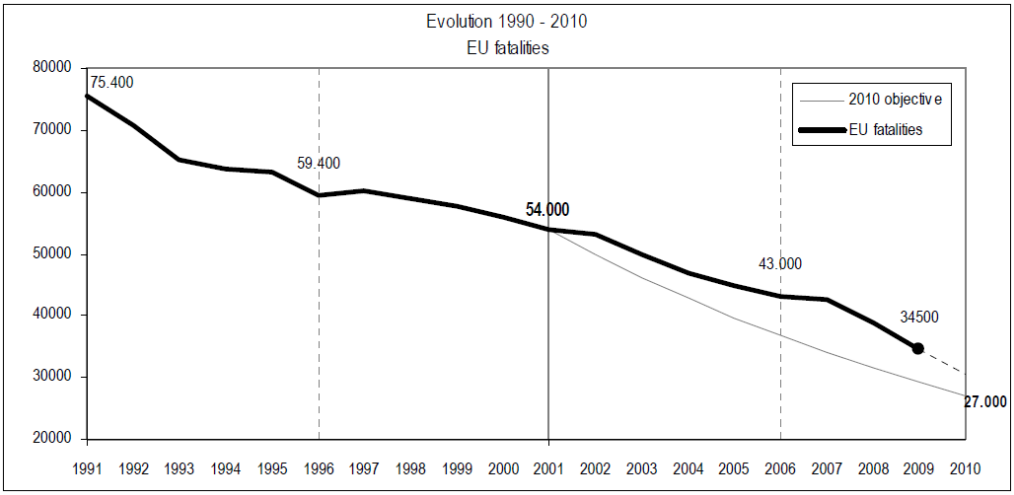


Figure 1 - European fatalities chart due to accidents

From the EU data it is also possible to see where these fatalities are decreasing in the Union and their frequency per million of inhabitants (*Figure 2* and *Figure 3)*.

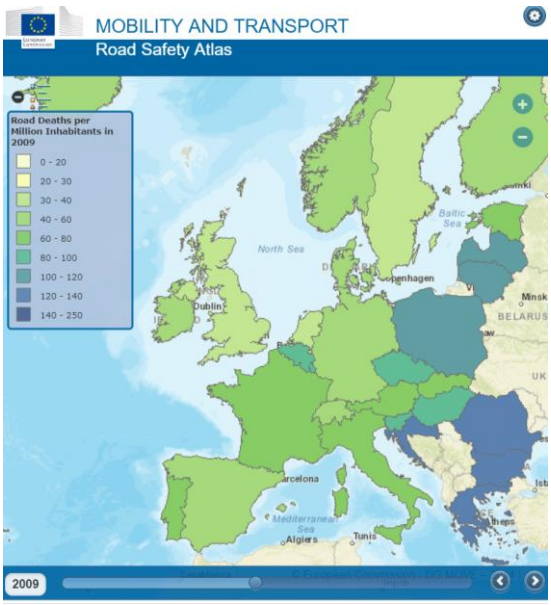


Figure 2 - Road death per million inhabitants 2009

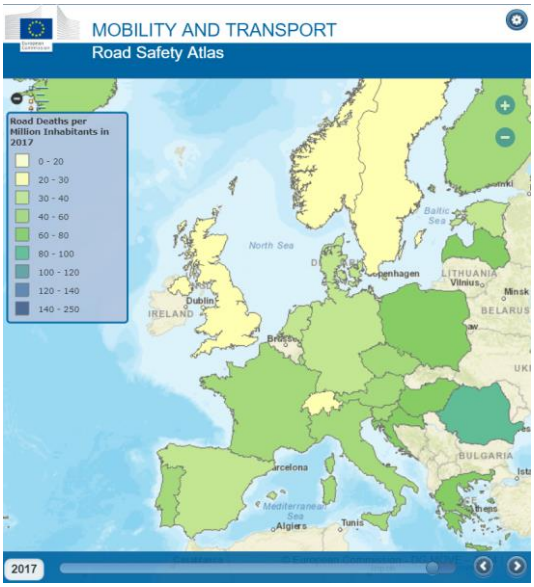


Figure 3 - Road death per million inhabitants 2017

The improvements are evident: fatalities decrease drastically after the adoption of Vision Zero and are not so far from the target.
Even if the results are great, according to the Vision Zero project, a roadmap [7] has been presented targeting to halve fatalities by 2030. In the guideline all the newest technologies are taken into consideration trying to reach the goal. It is interesting that smart mobility seems to have a key role for reduction of fatalities. It is also evident that safety is strictly related to traffic problems and how the type of mobility is intended.

1.2. Traffic

From national data [5] and international ones [4], it is possible to observe a continuous increasing trend of sold cars and it can be forecasted its growth for the current year. The bar chart below collecting worldwide data shows exactly this trend, providing a numerical reference.

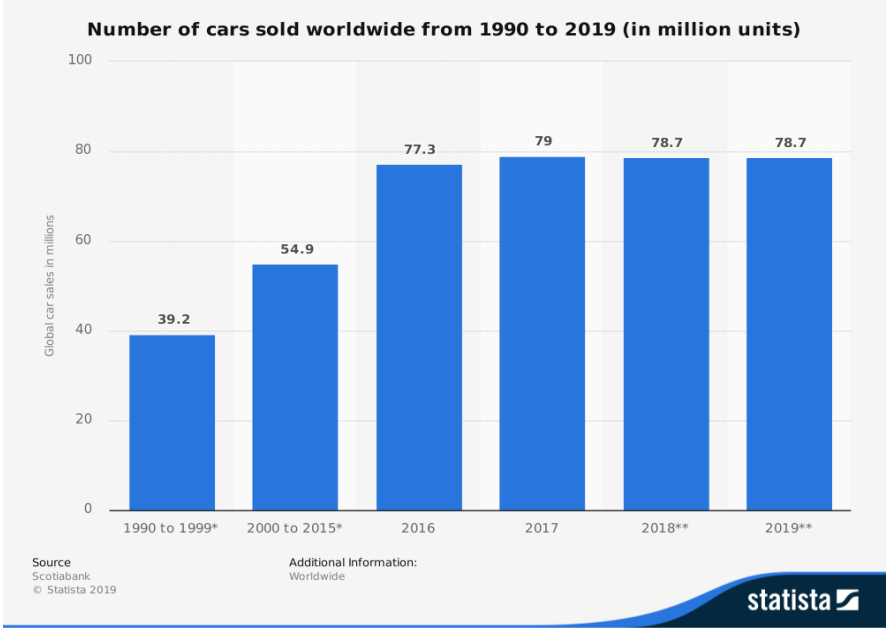


Figure 4 - Sold cars from 1990 to 2019 [1]

This growth causes the problem of the huge number of vehicles on the streets and the challenging task of managing them. The management difficulties increase in urban area, due to the limited possibility of resizing roads without involve the surrounding infrastructure.
The European Union is trying to face the problem investigating solutions which does not involve infrastructural modification. In fact, they are supporting the evolution of transportation from mobility as personal possibility to *Mobility as a Service*. It is possible to understand this intention in the STRIA (Strategic Transport Research Innovation Agenda) where, the will of massive use of smart mobility is target by 2030, even if some countries plan to reach this goal before. EU considers smart mobility as the starting point to avoid unnecessary vehicles movement in urban areas, integrating it with the public transport system to increase the urban mobility efficiency.

The introduction of V2X communication in new vehicles seems to be not just a technological improvement, but also a requirement to evolve mobility concept, increase safety and reduce traffic jams.

In this contest of smart mobility, this work has developed a project, focusing on a four-way intersection managed by traffic light. The intention is to overcome the traditional concept of traffic light using V2V communication systems. An algorithm for the orchestration of vehicles at the intersection has been designed avoiding external supervision and infrastructure changes.
A more detailed description is provided in Chapter 4.

2. Standards

V2X communication can be performed using many different technologies. During the years, mainly two of them have been identified as the most promising: the first belongs to the IEEE 802.11 standard deriving an ad hoc amendment 802.11p; the second belongs to cellular technology, before with 3GPP LTE standard and from this year with the 3GPP 5G one.
Since 5G protocols were not released when the thesis project started and considering that software frameworks implementing Wi-Fi protocols were already available and reliable, this last technology has been used to implement V2V capability among vehicles.

An overview of the 802.11p standard is presented in the next section.

2.1 IEEE 802.11p and 1609.x

Among the Dedicated Short-Range Communication (DSRC) one of the main is the so-called Wireless Access in Vehicular Environment (WAVE). It is composed by IEEE 802.11p, for the Physical and Media Access Control (MAC) layer of the ISO/OSI model, while IEEE 1609.x for the upper layers.
Next to IEEE standard, ETSI propose its own one, the so called ITS-G5. It is based on 802.11p Physical layer ad MAC layer, but different upper layers.
Moreover, SAE developed only the upper layers in its standard J2735.

	IEEE	ETSI	SAE
UPPER LAYERS	IEEE 1609.X-2011	ITS-G5	J2735
MAC LAYER	IEEE 802.11p	IEEE 802.11p	
PHYSICAL LAYER	IEEE 802.11p	IEEE 802.11p	

Table 1 - Standards' layers division

2.1.1 WAVE Physical Layer

WAVE adopt a physical layer similar to the 802.11a. Both use Orthogonal Frequency Division Multiplexing (OFDM) on frequencies from 5.850 GHz to 8.925 GHz, but the main difference is that 802.11p uses 10MHz-wide channels, instead of 20MHz of the 802.11a, in order to make the signal more robust against fading. Each sub-carrier can use different modulation like BPSK, QPSK, 16-QAM or 64-QAM allowing different data rate from 3 Mb/s (BPSK) to 27 Mb/s (64-QAM). Depending on the channel condition the Adaptive Modulation can choose the most suitable modulation scheme and it can also vary the coding rate, data rate and transmission power.

American and European policies about spectrum division are different and the number of used channels and their usage can vary. United States' policy divides the spectrum into 7 channels of 10 MHz width, instead European Union's standard uses 5 channels of 10 MHz width.

2.1.2 WAVE MAC layer

As for the Physical layer, also the MAC one is similar to the other 802.11 standards but presents some differences like that it is not necessary for stations and Access Point (AP) to join a Basic Service Set (BSS) going through scanning, authentication and association. This allows a direct communication between stations and AP but also among stations itself. To better understand the working principle of the WAVE MAC layer, the general 802.11 standards working principle is described.

In Wi-Fi based networks all nodes are half duplex so to access the channel an algorithm is needed. To enable communication between AP and stations time synchronization is required and it is maintained through Beacon frames. For 802.11 standard the algorithm is the Distributed Control Function (DCF).

The time is divided into *slots* whose time duration depends on the implementation of the physical layer. The frame transmission is regulated using Inter Frame Space (IFS) which define the time interval between two frame transmission. Four different IFS types exist: Short IFS (SIFS), Point coordination IFS (PIFS), Distributed IFS (DIFS) and Extended IFS (EIFS) and they are listed according to an increasing time duration.

SIFS separates transmissions belonging to the same dialogue and due to the fact that and they have the highest priority; PIFS are only used by the access point when it has to send a beacon frame; DIFS are used by stations to declare the channel as idle and the span for one SIFS plus two slots;

finally EIFS are used by a station when the physical layer notifies the MAC layer that a transmission has not been correctly received and its time duration depends on the implementation but must be longer than DIFS.
DCF manages the channel access so that if a frame must be transmitted and the channel has been idle for either a DIFS or an EIFS the transmission can begin. In fact, if the previous frame is received without errors, the medium is free for at least the DIFS, while if the previous transmission contained errors the medium must be free for an EIFS. Instead if a frame must be transmitted and the channel is busy, the station must wait for the channel to become idle. This waiting time is called *access deferral* in 802.11 standard and it works so that the station waits for the medium to be idle for a DIFS and execute an exponential *back-off* procedure.
The back-off procedure works selecting an integer number in the range [0, CW], where CW is the Contention Window spanning from a minimum and a maximum value, which will be the back-off counter. When the station remains idle for a DIFS, the station sets the beck-off counter and decrements it as long as the channel remains idle. If the channel becomes busy, the station freezes the back-off counter value. When the channel becomes idle again, the station will restart decreasing the value. When zero is reached, the station will be able to transmit.
The deferred access is used to desynchronize contending station, however frame collision cannot be avoided so, when it happens, the CW is increased and stations have to recompute a new back-off value. An important thing is that broadcast messages are never retransmitted in case of failure due to the absence of an acknowledgement frame.

In WAVE a similar MAC layer has been used. The channel access is still contention-based and the algorithm to access is called Enhanced Distributed Channel Access (EDCA) protocol. It is derived from DCF and includes some feature of the 802.11e standard. ECDA establishes that queues which are divided into four Access Categories (AC) have to be assigned to MAC, each one with a different priority. They are reported below with their parameters.

AC	CWmin	CWmax	AIFSN	TXOP limit
Background (AC_BK)	aCWmin	aCWmax	9	0
Best Effort (AC_BE)	aCWmin	aCWmax	6	0
Video (AC_VI)	(aCWmin+1)/2-1	aCWmin	3	0
Voice (AC_VO)	(aCWmin+1)/4-1	(aCWmin+1)/2-1	2	0

Table 2 - parameters of the Access Categories

These four queues work based on the Arbitration Inter Frame Space (AIFS) so that each queue must contend for accessing the queue contention on the channel. Contention among queues depends on the AIFS duration and the higher is the priority the shorter is the AIFS. In case of collision among queues data, a back-off procedure start so that the higher AC transmits, the lower is the AC back-off. After the queue contention it is won, data of the winning queue can contend for channel access as in DCF. The ECDA implements in this way a priority in data to be transmitted.
ECDA is part of the 802.11 standard and, on top of this, IEEE 1609.4 regulates multi-channel wireless radio operation. 1609.4 protocol defines two types of channel, the Control Channel (CCH) and the Service Channel (SCH), time synchronization and coordination between CCH and SCH, management services to allow channel switching and primitives designed for multi-channel operations.
The frequency band allocated for ITS span from the nominal carrier frequency (f_c) of 5860 MHz to 5920 MHz with a maximum channel bandwidth of 10 MHz. American and European channel allocation are different and in particular:

- E.U. uses only five channels, nominal carrier frequency from 5860 MHz to 5890 MHz are assigned to be SCHs, the f_c corresponding to 5900 MHz is assigned to be CCH, while f_c equal to 5910 MHz and 5920 MHz are for future development (ITS-G5D).
  E.U. provided an additional classification among channels, the first two channels (called ITS-G5B) are for non-safety application, while the remaining three channels are thought to be used for safety applications (called ITS-G5A)
- U.S. uses all seven channels and assign the CCH to the middle one, while the other six are used for SCH.

This description is summarized in the picture below:

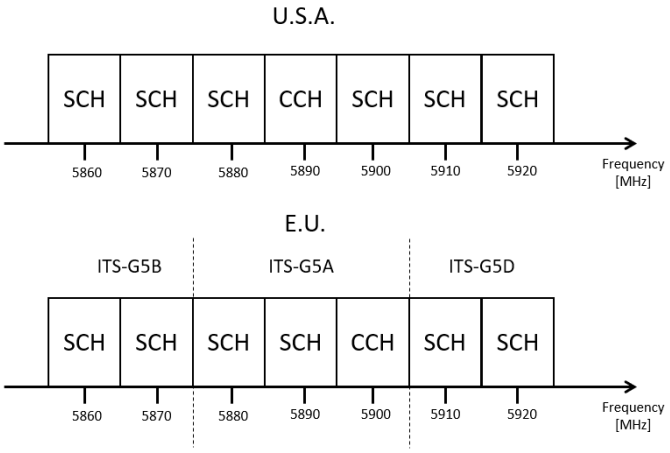


Figure 5 - American and European channel allocation

IEEE 1609.4 provides different switching modes for passing from SCH to CCH and vice versa. They are:

- Continuous channel access: when the channel is selected it does not change
- Alternating access between two channels: each time a predefined time interval expires, the channel switches. The switching can happen between a SCH and the CCH or between two SCHs
- Immediate channel access: the channel can be changed whenever it is required.

This channel distinction has been done in order to have a channel, the CCH, which transmits only non-IP messages, instead on SCH every type of service or message can be transmitted. Based on this difference, 802.11 provide two MAC entities, in this way messages priority can be handled in a better way. A representation of this mechanism is shown below.

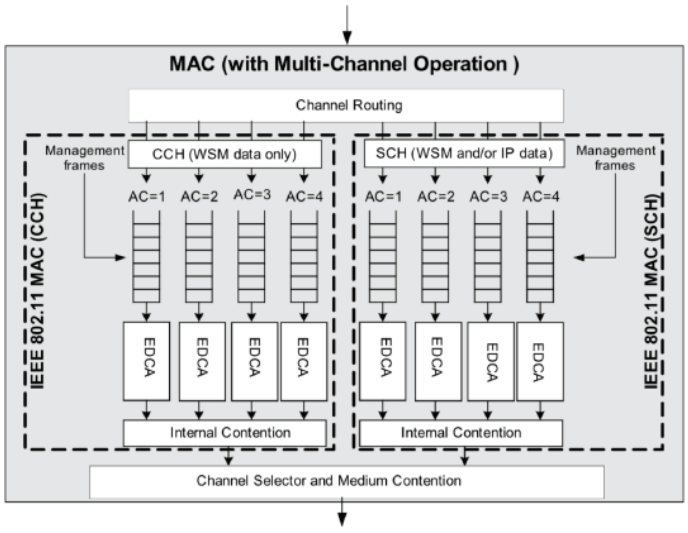


Figure 6 - IEEE 802.11 MAC layer

For what concerns time synchronization, usually 802.11 standards obtain it thanks to the beaconing with the AP. Due to the fact that in WAVE there is no need to join the BSS before start transit, the synchronization is obtained thanks to the timing signal coming from the GPS signal. In this way all stations are synchronized and, using a guard interval of 2 ms, differences in UTC estimation are considered. A simplified example of this is shown below.

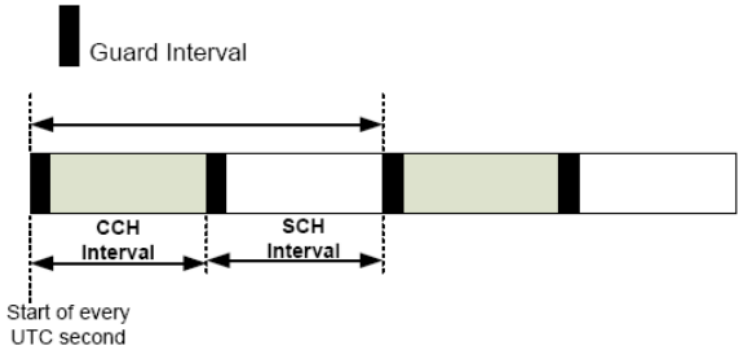


Figure 7 - Guard interval in message transmission

2.1.3 WAVE Network and Transport layer

The WAVE Network layer and Transport layers are defined in the IEEE 1609.3 and they are thought to allow a direct safety communication without a big overload. The constraint of short messages is to allow an easier communication among vehicles in movement. For this reason, the WAVE Short Message Protocol (WSMP) has been designed.

It is possible to see from the schematics below how WSMP is implemented. Logic Link Control (LLC) is responsible to tag messages sent and received so that it is possible to recognize if a message must be handled by the WSMP or instead it is an IP message. This division allows to manage in a safer way messages in a highly dynamic environment.

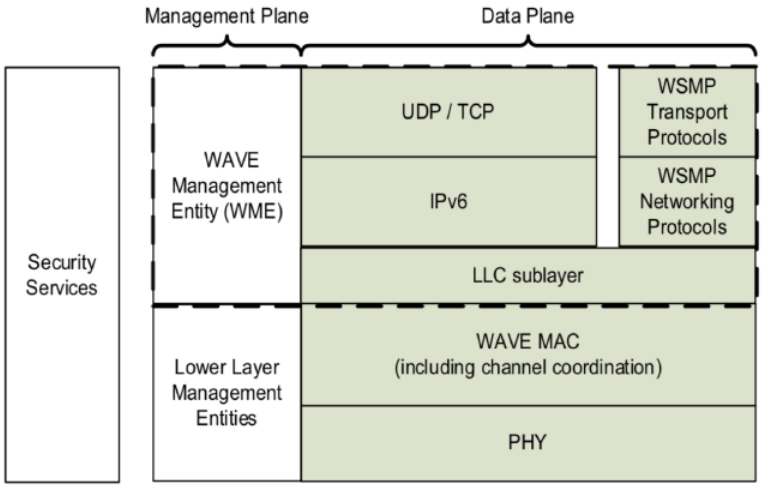


Figure 8 - WAVE stack

Another feature implemented by IEEE 1609.3 standard is the WAVE Management Entity (WME) that stores every service request and if needed selects the new service in the provided channel.

2.1.4 WAVE Security Services

Transmitted data require the implementation of safety features in order to guarantee privacy. To face these requirement IEEE in the 1609.2 standard, provides services to address this request. In particular five security services are described:

- Authenticity: Ensure that the sender is who it claims to be
- Authorization: Ensure that the sender is entitled to the privileges it requires
- Integrity: Ensure that all changes to the Secured Protocol Data Unit (SPDU) after it is signed can be detected
- Non-repudiation: it is the ability to demonstrate authenticity, integrity and authorization to a third part
- Confidentiality: Ensures that only the intended recipients can read the contents of the SPDU

In order to have all these security services certifications are needed and provided by Certification Authority (CA). Furthermore, certificates have an expired time, after which a new one must be released by the CA in order to create an obstacle for vehicle's tracking.

2.2 ETSI ITS-G5

As previously described ETSI designed its own standard. It is based on the IEEE physical and MAC layers and provides an alternative for upper layers implementation. It is possible to have an overview of layers implementation in the image below.

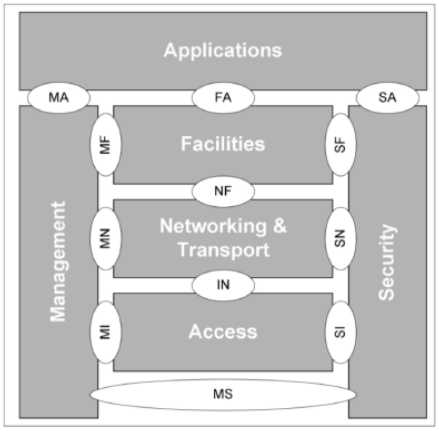


Figure 9 - ETSI ITS-G5 stack

It is possible to see that between the Network and Transport layer and the application one, an additional layer has been added and it will be presented in Section 2.2.4 .

2.2.1 ETSI Physical and MAC layer

As previously said, ETSI uses the same Physical and MAC layers of the IEEE 802.11p standard. Must be said that ETSI introduces a congestion-based selection of EDCA parameters and defines in a more detailed way the purpose of service channels. In particular channels are divided in the following way:

- ITS-G5A: it is set for ITS safety road traffic applications;
- ITS-G5B: it is set for ITS non-safety road traffic applications;
- ITS-G5C: it is possible to use this channel only in the context of a basic service set;
- ITS-G5D: it is reserved for future development.

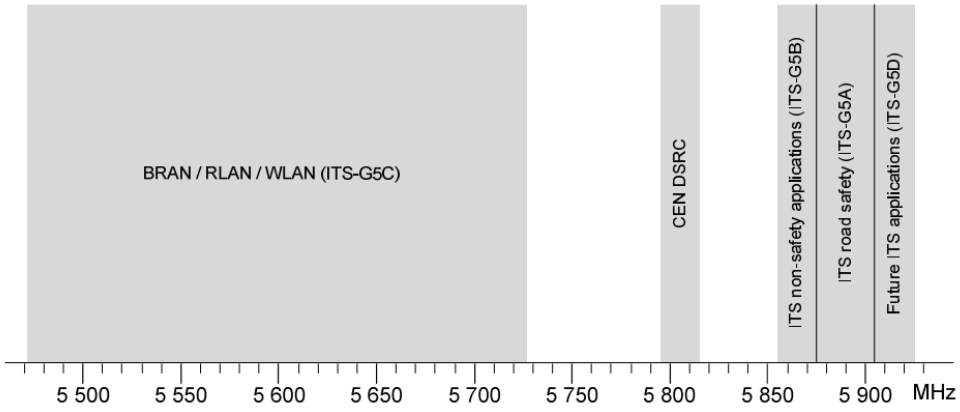


Figure 10 - ETSI channels for ITS systems

Another difference with respect to IEEE protocol is the channel management. Even if ETSI implements the same 1609.4 layer, it defines a multi radio multi antenna system so that channel switching is no more needed. This avoids problems regarding synchronizations and allows to have always one radio tuned on CCH and the additional radio tuned on SCH.
On CCH three messages are defined by the standard:

- Service Announcement Message (SAM): it is used to advertise aspecific services;
- Cooperative Awareness Messages (CAM): it is generated by the CAM Manager at mostly 1 every second or based on hazardous driving conditions;
- Decentralized Environment Notification Message (DENM): it is triggered by event like for example bad road conditions.

2.2.2 Decentralized Congestion Control

One of the problems in the 802.11 standard is that if there are many vehicles in the same area, trying to communicate among each other, may happen that vehicles have communication problems due to the so high number of exchanged messages which might cause collision. In crowded situation it is not possible to avoid collision, but the Decentralized Congestion Control (DCC) introduced by ETSI tries to reduce this phenomenon.

DCC is one of the core features of ETSI protocol that implements adaptive transmission parameters to avoid overload and managing:

- Transmission Power Control (TPC);
- Minimum inter-packet time (Transmission Rate Control TRC);

- Transmit Data rate Control (TDC);
- Sensitivity of Clear Channel Assessment (CCA – DSC – DCC Sensitive Control).

These transmission parameters are modified based on channel loads and they change based on three states:

- Relaxed;
- Active;
- Restricted.

DCC works as a state machine switching among these states based on thresholds. The result of this behaviour is that if a car drives in an area with an high vehicle density, it can switch from Relaxed state to Active state to limit the communication range, if the channel load is still over a maximum threshold it can switch to the Restricted state reducing as much as possible the communication distance. A schematic behaviour is shown below.

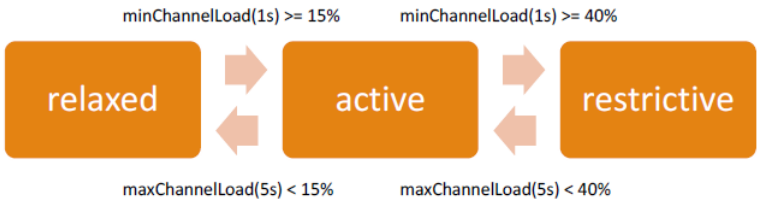


Figure 11 - DCC states

2.2.3 ITS-G5 Networking

The networking layer has been designed in order to support different applications, including safety. Furthermore, it provides three types of message routing:

- GeoUnicast in which the message is routed based on GPS position of a particular vehicle
- GeoBroadcast in which the message is sent to multiple nodes in a geographical area and they are responsible for rebroadcasting it
- Topologically-scoped broadcast in which the message is sent to multiple nodes in a geographical area, but the rebroadcasting is limited to a certain number of neighbourhoods

2.2.4 Facility layer

The additional layer introduced by ETSI allows to support the application layer by providing three types of application support messages, that can be used for different types of applications.
The Facility layer is subdivided into three sublayers as shown below.

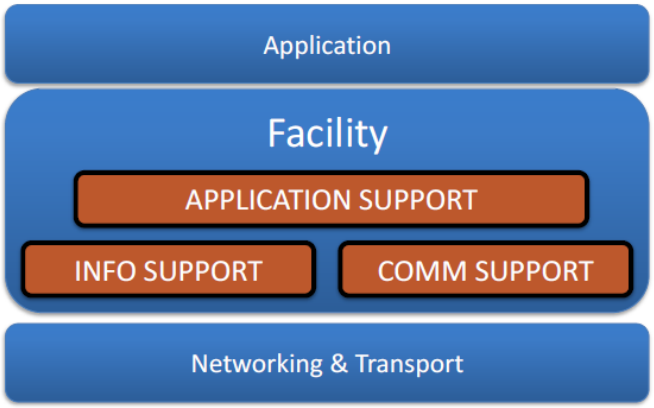


Figure 12 - ETSI Facility layer

The Application Support provides common services and/or functionalities for the Basic Set of Applications (BSA). The provided facilities are for example SAM processing and transmission, support the protocol processing of DEN messages, support the protocol processing of CAM, provide service assess to billing and payment service provider.
The Information support facility instead provides common data and database management functionalities for BSA execution, for example the Local Dynamic Map and map database.
Finally, the Communication support facility provides services for communications and session management, for example it supports geocasting and session support.

2.3 SAE J2735

Another important standard is the one defined by the Society of Automation Engineers (SAE). The J2735 is intended to standardize DSRC Message Set Dictionary, data frames and data elements that can be used by applications in a DSRC system.
The standard provides interoperability among DSRC applications and gives details on how implement messages from an application developer point of view.
The 16 messages described in the standard are now summarized below.

Basic Safety Message (BSM)
It is the basic message for safety applications, and it is intended for exchanging data among vehicles related to vehicle's state. The BSM is broadcast periodically with a maximum frequency of 10 messages every second.

Due to its purpose the message has a compulsory part, containing information about message ID, timing and position, but also and additional part which expands the message with data about speed, heading, steering angle and acceleration.

Common Safety Request (CSR)

It is a unicast message to ask for additional information for safety application during an exchange of BSM.

Emergency Vehicle Alert (EVA)

It is a message related to a certain area and advertises for incident or other types of vehicles emergency.

Intersection Collision Avoidance (ICA)

It is used in V2X communication to provide information for a collision avoidance system. It transmits data about the intersection and the recent path of the vehicle.

Map Data (MAP)

It is used to transmit geographic road information, mainly for multiple intersection lane geometry. It is typically transmitted between RSU (Road Side Unit) and vehicles, but it can also be used in V2V communication.

NMEA Corrections (NMEA)

It is used for differential correction for GPS radio navigation signal and it allows to increase the absolute and relative position accuracy provided by the GPS.

Probe Data Management (PDM)

It is used for exchanging management data between RSU and On-Board Unit (OBU) such as RSU coverage.

Prove Vehicle Data (PVD)

It is used between RSU and OBUs to exchange data about vehicles behaviour along a segment of road.

Road-Side Alert (RSA)

It is used to send alert from RSU and OBUs int the proximity. The aim is to advertise for hazardous situations in a certain area. Inside the RSA message a priority level is present to determine the type of message to present to the driver and minimize the risk of distraction.

RTCM Correction (RTCM)

It is used for differential correction for GPS and other radio navigation systems. As the NMEA, it is intended to increment the relative and absolute accuracy, but RTCM does not concern only GPS but also other navigation systems.

Signal Phase And Timing (SPAT)

It is sent by RSUs, positioned in proximity of an intersection, to communicate data to the OBUs about traffic light's phases and their duration.

Signal Request Message (SRM)

It is sent from an emergency vehicle to an RSU of a signalized intersection to request a pre-emption on the next signal phase. The vehicle identifies itself through the body of the BSM and sends the request to the RSU along with a duration time of the service. The recipient RSU examines the interested zones of the intersection (sent in the MAP message) and decides whether change the signal phase to allow the service of the emergency vehicle. The outcome of all the pending signal requests can be found in the SSM and may be reflected in the SPAT message if successful.

Signal Status Message (SSM)

It is a message broadcast by an RSU if it receives a SRM, and it is intended to list the status of all pre-emption or priority events acknowledge by the RSU. This message allows the surrounding vehicles to know the ranking of the request that they made.

Traveller Information Message (TIM)

It is used by RSUs to advertise and to send road sign messages to the nearby vehicles. The affected area is expressed using a short-defined region system.

Verbose Basic Safety Message (VBSM)

It is an extended version of the BSM containing also the Bit Error Rate (BER), but it is only used for system testing and development.

UPER Frame USA (UPER)

It is used to encapsulate the (USA Packet Encoding Rules) UPER encoded message in a simple DER (Distinguished Encoding Rules) framework.

3. Simulation software

Developing an application for the automotive field usually requires that this follows the so-called V-model, a schematic representation is shown in Figure 13. It starts with the requirement analysis and finishes with the release of the software for the production. Usually during the process developers execute many tests in order to satisfy the requirement and ensure its correct behaviour. These tests called Software in the loop (SIL) and Hardware in the loop (HIL), depending on how the software is tested, guarantee a correct behaviour of the software before its implementation on a real vehicle.

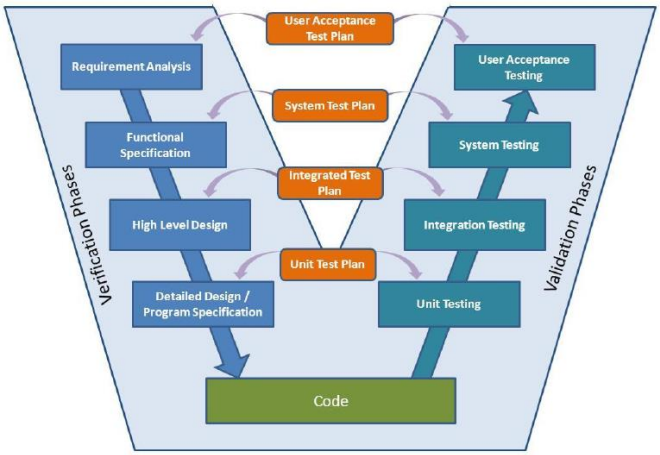


Figure 13 - V-Model representation

Developing applications which involve vehicles in a DSRC network require the use of had hoc simulators to test the behaviour of the software in a simulated environment. This is needed because it is not possible to just test the software execution, but it has to be exposed to situations which present multifactorial variable like signal attenuation, different vehicle's dynamics and all the variations that a simulated environment can introduce.
Depending on the applications requirement different software are available. In this thesis project a co-simulation has been used so that the scenario and the traffic flow have been simulated using SUMO, while the connectivity has been implemented using Veins, an OMNeT++ framework.
In the figure below is possible to have a look on the schematic working principle of the co-simulation.

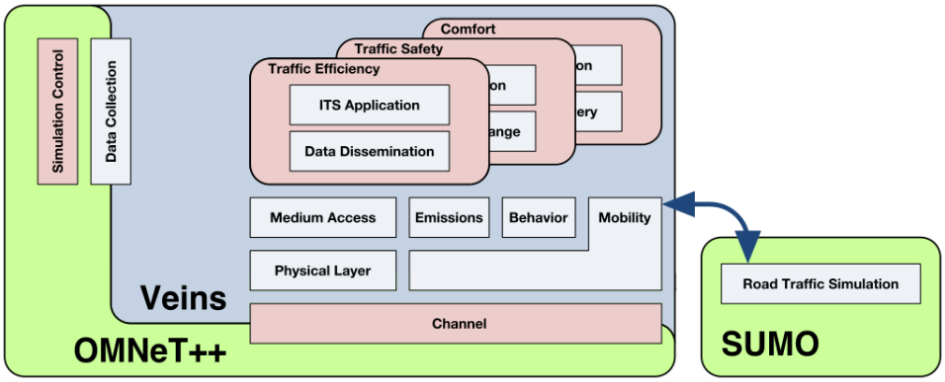


Figure 14 - Communication between OMNeT++ and SUMO

3.1 Simulation of Urban Mobility (SUMO)

SUMO has been developed by the DLR, the German Institute of Transport Systems, to evaluate how traffic behaves to infrastructure changes before implementing them in the real world.
In general, it is possible to differentiate among two main traffic simulators group:

- Microscopic traffic simulator: focussed on the mobility data of each simulated vehicle as speed, acceleration, pollution and other vehicles parameters
- Macroscopic traffic simulator: focused on road flow, traffic density, vehicle distribution and other analysis which require an overall view of the traffic

SUMO is a microscopic traffic simulator able to simulate complex scenario and it is also intended to simulate a huge number of vehicles. Due to these characteristics, its reliability and the fact that it is released as an open source software, it has been chosen for this project.

3.1.1 Network design

SUMO works mainly with three xml files, one which describes the scenario (called network), one which describes how the traffic flows over the network and the last which describes additional features as buildings.
SUMO defines roads as a two or more *nodes* connected by *edges*. Both nodes and edges are identified by a unique ID and their position is described based on a reference system centred in the

middle. After this main definition many other parameters can be set as the number of lanes, speed limits, etc.

The network designed for this thesis is a 4 ways intersection, each road has one lane and it is 1 km long. For what concerns the intersection definition, SUMO provided many different alternatives. The intersection is defined as a node in the *nod.xml* file and its behaviour depends on *type* definition. Firsts simulations have been done using the "right_before_left" type. Testing the vehicles' behaviour with this configuration turns out that SUMO always tries to give priority to the traffic coming from the right. Since the project needs an unregulated behaviour of vehicles to be managed by the developed algorithm, the intersection type has been modified into "traffic_light" and its lights have been set to be always green. Making this change ensures to have a traffic flow unregulated by SUMO.

It is possible to see the designed intersection in the following pictures.

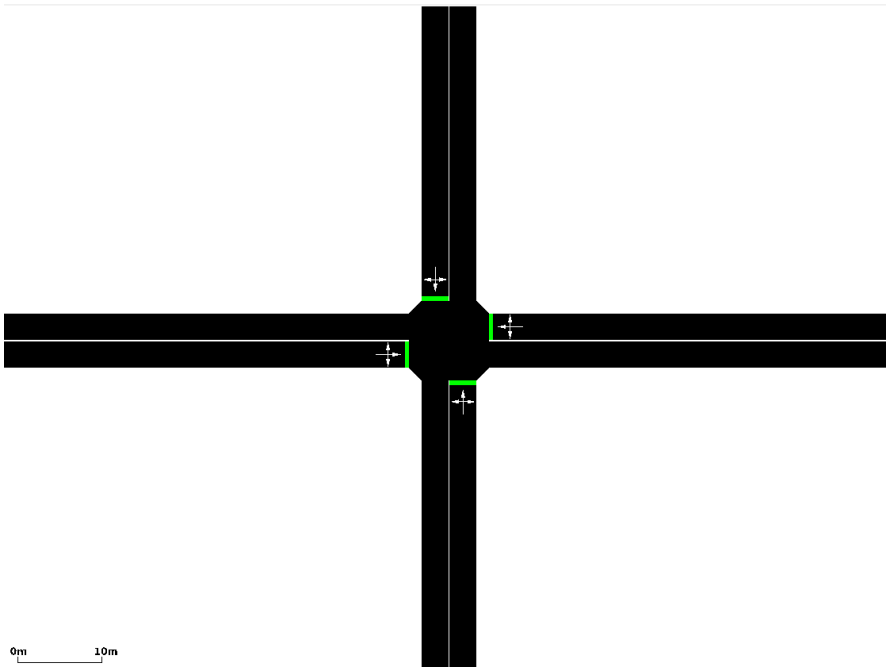


Figure 15 - Intersection with always green traffic lights

3.1.2 Traffic flow

The traffic flow is defined in a different file with respect to the network. The file describes vehicles characteristics such as maximum speed, maximum acceleration and deceleration, but also the path followed by each vehicle during the simulation. This path is described in the *child* "route" defined for each possible path; they are characterized by a unique attribute "id" and an attribute "edges" containing all edges involved in a path. Each vehicle generated has an attribute "route", which must match the child route id, that defines the path imposed for that vehicle.

Due to the need of test different combination of vehicles flow, this file is generated by a python script in order to easily generate a big number just providing the number of vehicles for each road and the direction of each vehicle is standardized based on numbers, in detail:

- 2 = turn right;
- 3 = turn left;
- 4 = go straight on.

An example for vehicles approaching the intersection from East is shown below:

```
Evehicles = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
Esignals =  [4, 2, 4, 3, 4, 2, 2, 3, 4, 3, 4, 4, 2, 2, 2, 2]
```

3.2 Veins and OMNeT++

OMNeT++ is an object-oriented modular discrete event network simulation framework. It does not have a specific architecture so that it can be used in different fields such as:

- Modelling of wired and wireless communication networks;
- Protocol modelling;
- Modelling of queueing networks;
- Modelling of multiprocessor and other distributed hardware system.

In general, it is intended for modelling and simulating any system where the discrete event approach is suitable.
The simulation structure of OMNeT++ is based on hierarchical nested modules that communicate by passing messages. The model's module interaction is described using NED language, while inside each module the algorithm is implemented in C++. The main model parameters are implemented in the *.ini* file and users can run simulation using among three possible options: command line (Cmdenv), Tkenv or Qtenv. These last two options are graphical interfaces. For the developed thesis only Qtenv and Cmdenv have been used.

Veins is based on OMNeT++ and is an open source framework for running vehicular network simulator. Typically, it is used to validate applications by means of a simulation, since the model takes care of setting up the simulation and ensure its proper execution. Veins connects OMNeT++ and SUMO via TCP socket and the protocol for the communication between the two has been standardized as the Traffic Control Interface (TraCI). This implementation allows bidirectionally-coupled simulation of network traffic, the vehicles movement in SUMO is reflected in OMNeT++ simulation.
Via TraCI many commands are already implemented and ready to be used. In this project just few commands have been added, following the documentation on SUMO's website.

Last but not least, the development environment provided by OMNeT++ and Veins is done so that the designer has an "in vehicle" point of view and he can already develop the application as if it was implemented on an ECU, then OMNeT++ takes care of "distribute" the application to all the simulated vehicles.

4. Workflow of the algorithm functions

The thesis project starts by the last available release of Veins (version 4.7.1) and with the SUMO version 0.30.0 since the newer version (1.0.1) was still unstable and under development.
The first task faced was to develop the so-called *Car Internal Storage,* a data structure able to collect all data sent and received by each vehicle. The selected structure is a class called *VehicleData* and its data has been set as private, to avoid data overriding. In order to organize the data considering which vehicle generate them, a *map*, called *storage*, has been created having as *key* the vehicle ID and as *value* the *VehicleData* class. This allows to access easily and in an efficient way to vehicle's data.
The implementation is shown below:

```
std::map<int, VehicleData> m_storage;
```

Vehicles communicate exchanging BSM, its data has been expanded with respect to the original version and contains the following data:

- ID (nodeIDBSM);
- Speed (senderSpeedBSM);
- Heading (headingBSM);
- Acceleration (accelerationBSM);
- Intersection position (intersectionXBSM and intersectionYBSM);
- Yaw angle (yawBSM);
- Sending time (sendtimeBSM);
- Car length (carLengthBSM);
- Car width (carWidthBSM);
- Position (lanePositionBSM);
- Road ID (roadID);
- Intersection flag, to communicate that the intersection has been crossed (intersectionFlagBSM);
- Vehicle's signaling light (signalBSM);
- Leader's string (leaderStringBSM);
- The optimization dataset (stringFlagBSM);
- A flag to advertise that the vehicles has been optimized (optimizationFlagBSM);
- An optimization's counter (optimizationCounterBSM).

Workflow of the algorithm functions

When a vehicle receives a BSM, it has to execute seven functions and their sequence diagram representation is shown below.

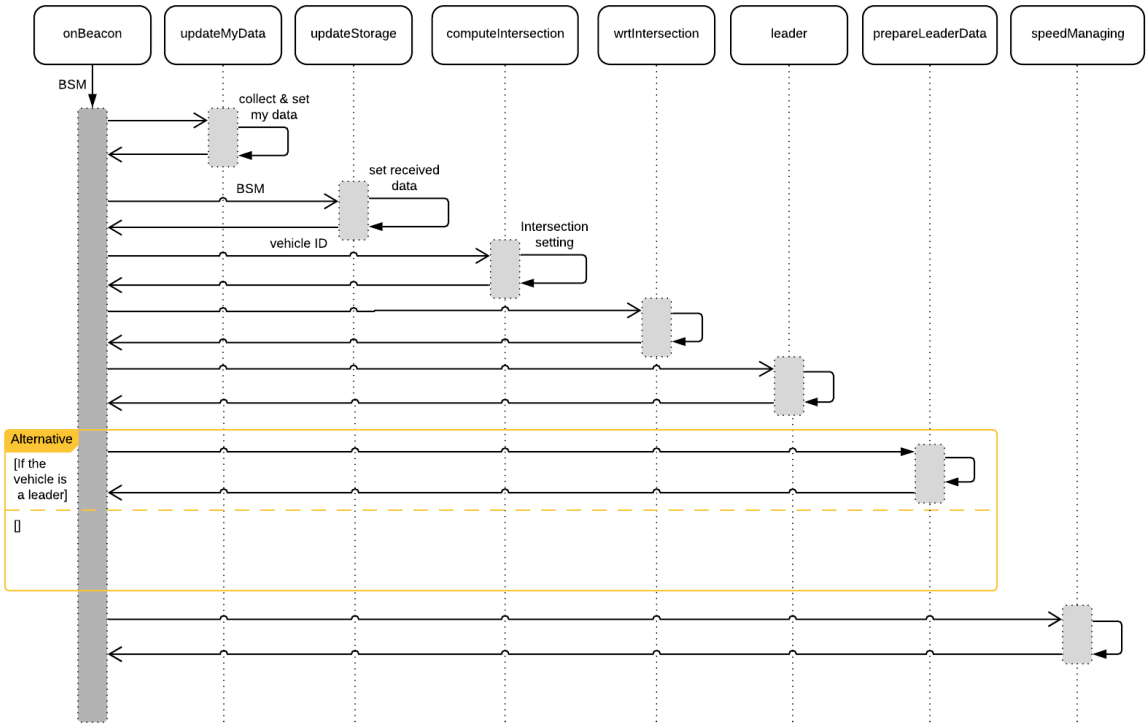


Figure 16 - Sequence diagram of the functions executed at BSM reception

These *methods* allow to store vehicles data and manage the vehicle's behavior, so a detailed explanation is provided in the following section.

4.1 updateMyData

The *updateMyData* function is responsible for saving the vehicles data in its storage. From SUMO data about id, position, speed, heading, position, acceleration, road ID and light signalling are extracted via TraCI and stored in the vehicles storage.

It is important to point out also some controls which are executed on some variables to set important flags. The first is about the road ID in which, if the ID change, it means that the vehicle is crossing the intersection. In addition, if the previously sored name belongs to the intersection's ones and the new vehicle position had a road ID, which does not belong to the intersection set, this means that the vehicle crossed the intersection, so a flag to detects this change is set

(*intersectionFlag)*. This is possible because SUMO assigns to intersection road an ID which starts with " : ". The pseudocode for these tests is shown below.

```
if(previouslySavedRoadID != newRoadID){
    storage[myID].setCrossingIntersection(1);
}

if(previouslySavedRoadID[0] == ':' && newRoadID != ':'){
    storage[myID].setIntersectionFlag(1);
}
```

Another function, that requires additional description, is the one which sets the vehicles signal light. SUMO set the car indicators when the vehicle distance from the intersection is lower than 100m. During the simulation, signals are valid only for the considered simulation step and they are encoded as integers so that:

- 1 means turn right
- 2 means turn left
- 8 means breaking light

It is possible that two signals come together as breaking light and turning one; in this case the provided value is the sum of them. The previously described values have been extracted from ad hoc simulations due to a mismatch between the data coming from the simulator and the online documentation. Even if there is this difference in the returned signals of Veins, they remain consistent among all the tested simulations.
Once this behavior became known, a function for setting the signal, which takes care of all these variations, has been developed and the pseudocode is reported below.

```
void setSignal(int signal){
    //right
    if(signal == 1 or signal == 9){ save_turn_right; }
    //left
    if(signal == 2 or signal = 10){ save_turn_right; }
    //straight
    if(no_one_of_the_previous_cases){ save_go_straight_on; }
}
```

4.2 updateStorage

The *updateStorage* function is used to extract data from the received BSM and save them in the storage. It tests if the vehicle ID is already present in the storage. If so, it updates the current data with the new ones, otherwise it inserts the new vehicle's data in the storage.
Inside it there are also two functions (*leaderCheckOptimization* and *AllVehicleCheckOptimization*) which will be explained in detail in Chapter 7.

4.3 computeIntersection

In the first phase of the thesis project the intersection was computed based on vehicles position and heading. In that phase only SUMO has been used, the developing and integration of the function were made in python as a function to be executed for each vehicle in the storage. This was done considering the need to translate the code in C++ when moving the development on Veins.
The computation required at least two vehicles approaching the intersection from different directions. In this way the crossroad was computed as the intersection of two lanes:

While using just SUMO no problems were detected, when it has been implemented in Veins some problems came out. The first is about the coordinate system (CS). As described in the Veins documentation three differences exists between the CS of SUMO and Veins:

- SUMO has its origin (0,0) in the bottom left corner, while Veins in the top left;
- SUMO maps are non-normalized coordinates, instead OMNeT++ canvas is;
- SUMO maps use negative coordinates, while OMNeT++ uses only positive ones.

For these reasons, simulators generate different output when asked to provide the coordinates and this affect the computation of the intersection result, increasing its uncertainty.
In addition, in SUMO this information was available and accurate just few second after vehicles generation. This is not possible in Veins since signal attenuation has been considered, consequently vehicles approaching the intersection from different direction can communicate among each other just few meters before the crossroad. Receiving this information so late, vehicles would postpone the computation of other data like the distance left before reaching the intersection, the leader election, which in turn cause a delay in the beginning of the optimization and management of the vehicle.

Due to the previously described problems, the assumption that vehicles know in advance the intersection position has been done. This is a rational assumption considering that it can be extracted from maps and then provided to the implemented algorithm. The function implementation has been left in the code for further developing, but for this project it just assigns the intersection position.

4.4 wrtIntersection

In the *wrtIntersection* function the distance from the vehicle position to the intersection and the time needed to drive this distance have been computed. The former has been called distance2intersection (D2I), the latter has been named time2intersection (T2I). Each vehicle performs these two computations for each car data saved in the storage, then the results are assigned to each of them.

The computation of the D2I is done based on the vehicle coordinate and on the intersection one. The computation of T2I hes been implemented dividing the D2I by the vehicle speed. T2I has been computed as istantaneous time instead of using the uniformly accelerated motion formula because SUMO does not provide reliable acceleration values. This has been understood because assigned a constant speed to a vehicle, its acceleration values alternate a positive and a negative value each simulation step. The acceleration values have been extracted and plotted to understand if filtering the values, the acceleration might be used reliably .
These extracted data rapresent the values of a vehicle accelerating from zero and then moving a constant speed and the speed profile is reported below. As it is possible to see below, even using filtered values it is not possible to use the acceleration, and neither the uniformly accelerated motion formula, to obtain a more accurate result for the computation of T2I.

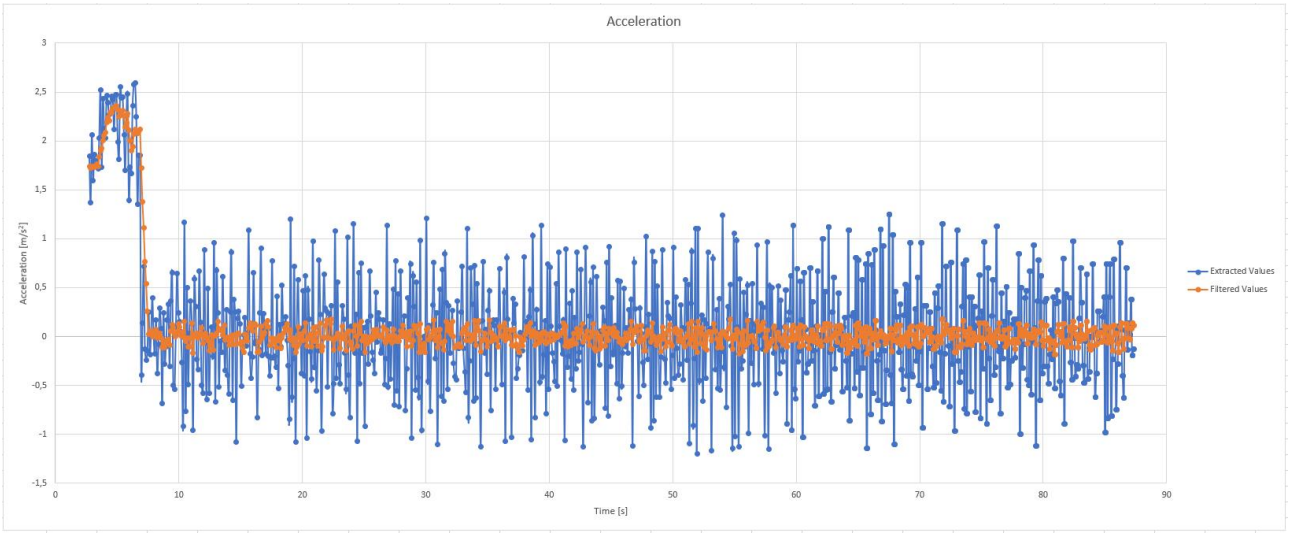


Figure 17 - Acceleration profile extracted from the simulation.

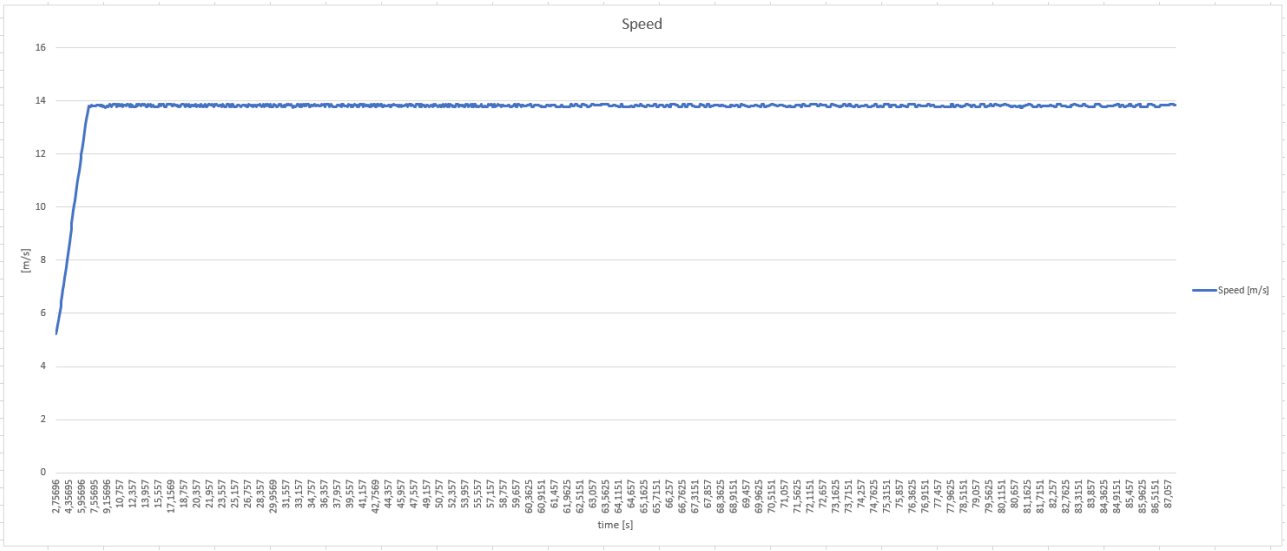


Figure 18 - Speed profile extracted from the simulation.

4.5 leader

The *leader* function is intended to detect which is the first vehicle, for each direction, that drives towards the intersection and that has not been optimized. This search is performed on storage's data and it is mainly based on the D2I.
The leader function calls *optimization_already_happened(storage)* which tests if a vehicle has been already optimized, if it is found the function return true. If this happen, the subset of vehicles that are not optimized is extracted from the storage, otherwise the whole storage is considered. Then the procedure is the same:

1) Vehicles are divided by their heading in a *map<double, vector<double>>* structure;
2) They are sorted based on their D2I;
3) The closest vehicle to the intersection for each direction is selected as leader;
4) The leader ID is assigned to all vehicles which has the same heading.

```
void leader(){
    if(optimization_already_happened(storage) == false) {
        order_vehicles_based_on_their_heading;
        for( i-th heading) {
            sort_vehicles_based_on_their_D2I;
        }
        collect_each_leader;
        assign_the_leader_to_each_vehicle_with_the_corresponding_heading;
    }
    if(optimization_already_happened(storage) == true) {
        not_optimized = extract_not_optimized_vehicles_from_the_storage;

        order_not_optimized_based_on_heading;
        for(i-th_heading_of_not_optimized) {
            sort_vehicles_based_on_their_D2I;
        }
        collect_each_leader;
        assign_the_leader_to_each_vehicle_with_the_corresponding_heading;
    }
}
```

4.6 prepareLeaderData

The *prepareLeaderData* function is intended to collect and share among the other leaders the dataset among which the considered leader would like to run the optimization. This function is executed only once and only if the vehicle:

- is the leader;
- has a D2I <= threshold.

The threshold has been selected so that leaders have enough meters to stop safely but considering also that they must be as close as possible to the intersection to maximize the number of seen vehicles with a D2I lower than 100m. This last requirement considers that SUMO activates vehicles signaling when they are at least 100m away from the intersection.

In *prepareLeaderData* each leader searches in the storage for cars with a D2I value lower than 100 and collect them in a vector called *under100*. Based on their di D2I, index of the position in the queue (*queue position)* has been assigned to them, so that starting from the leader they are numbered in an increasing order. To clarify the queue position, its graphical representation is shown below:

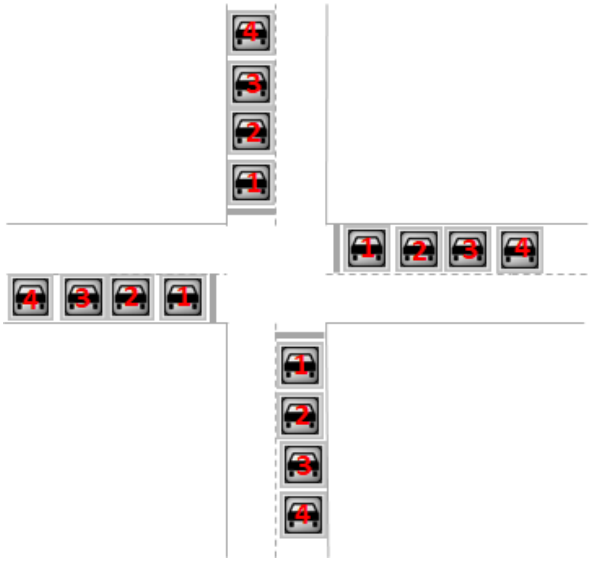


Figure 19 - Queue positions of the vehicles

Workflow of the algorithm functions

Once the under100 vector is filled, vehicle data need to be converted before been broadcast. In fact, not all data are transmitted, but only the ones needed to run the optimization plus the vehicle ID. They are:

- Vehicle ID;
- Queue position;
- Turn signal;
- Heading.

In order to transmit those data in a single field of the BSM, they have been coded into a string of bits. To do so, the under100 vector is passed to a function designed ad hoc and the output is saved into a map, ready to be broadcast.

```
void prepareLeaderData()
{
    collect_all_vehicles_not_optimized_with_D2I_lower_then_100;
    divide_vehicles_based_on_their_direction;
    order_vehicles_based_on_their_D2I;
    assign_to_each_of_them_their_queue_position;
    convert_main_data_in_bit();
    save_the_bitstring_to_be_broadcast;
}
```

4.7 speedManaging

The *speedManaging* function is intended to control the vehicle's behavior while it approaches the intersection. Its algorithm workflow can be seen below.

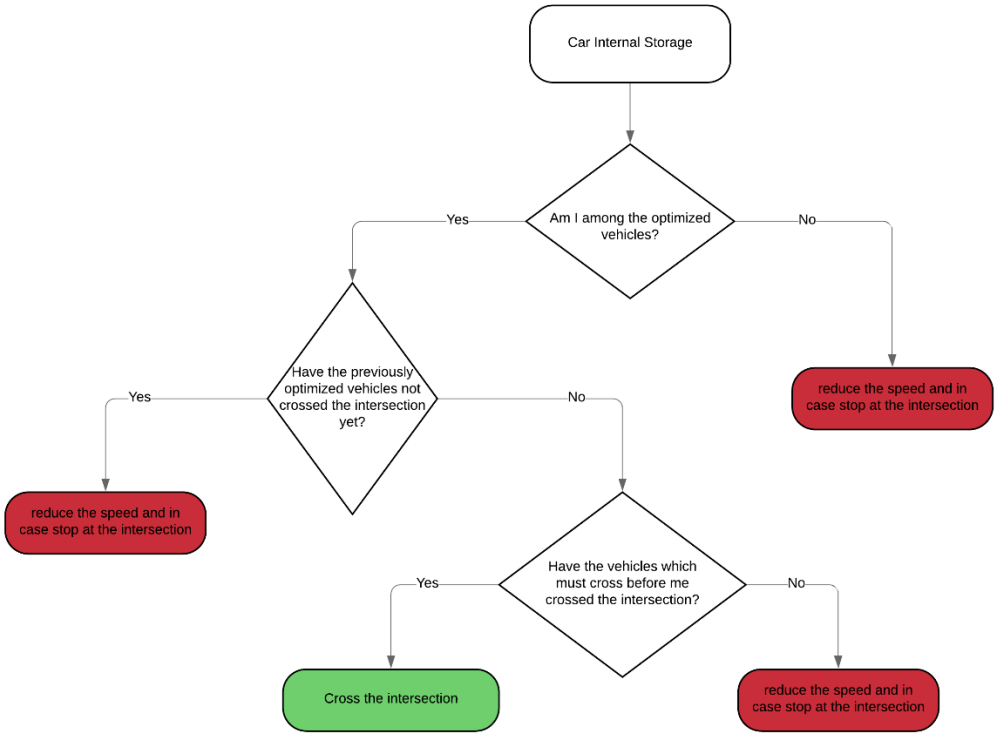


Figure 20 - Speed managing flow chart.

It mainly relies on four functions which implement this flowchart:

- *previousOptimizationNotCrossed(storage, myId)* is responsible to take care of the second test. Vehicles of a new optimization must wait for vehicles of the previous one to cross the intersection. Even if this behavior may introduce some delay, it guarantees to avoid hazardous situations.
- *canICorss(storage, optimizatonSolution, myId)* is responsible to implement the third test shown and it combines the solution of the optimization process and the data of the storage to evaluate if the considered vehicle has to cross the intersection or not.
- *speedAproachingIntersectionManagement(storage, myId)* is responsible to manage the deceleration and to stop the vehicle which approaches the intersection based on the vehicle D2I.
- The fourth function is the command to set the speed of the vehicle equal to 50 km/h. By doing so the vehicle accelerates up to the set speed.

5. Optimization development

The development of the optimization algorithm starts with the mathematical description of the intersection. Initially existing implementation of an intersection has been searched, but only description on how the intersection is divided has been found. Since the intersection implementation does not have to follow a predefined model, a “chess board” based representation has been chosen. The implementation is based on a mathematical description of the chess board presented by Peter W. Frey [8]. Frey describes a representation of the chess board implemented using bits (*bitboard* or *bitmap*) in order to speed up computational time.
Since the project needed a mathematical description of the intersection and a way to test legal movement of the cars, a derived model of the bitboard has been developed.

5.1 Mathematical model of the intersection

To represent the intersection as a bitmap, it is necessary to divide it in small cells and each of them is modeled as a bit. As it is possible to see from the picture below, in this way each bit can describe all intersection areas.

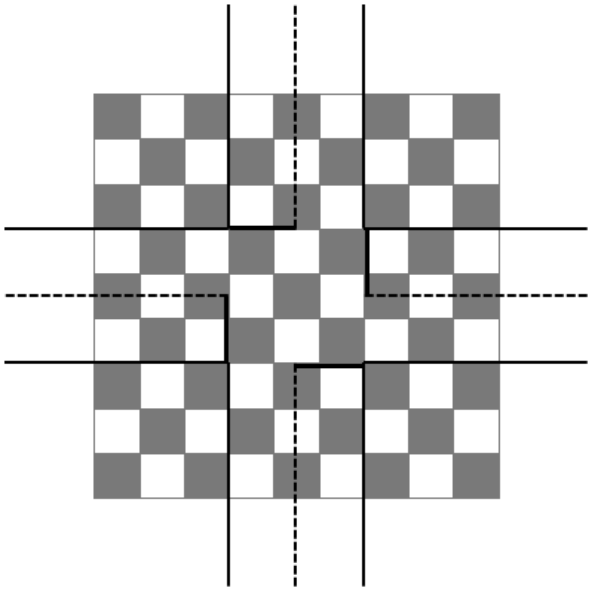


Figure 21 - Modeling the intersection as a chess board

The first step is to decide how many cells are needed to represent the area in which vehicles cross the intersection. Testing possible combinations to consider all crossing cases, the minimum number of cells obtained is a square 3x3. In fact, with a 2x2 intersection representation, if two vehicles approach the intersection from perpendicular directions and one of wants to turn right while the

other wants to turn left, it is not always possible to represent correctly this situation. This problem is solved with a 3x3 description of the intersection as shown below.

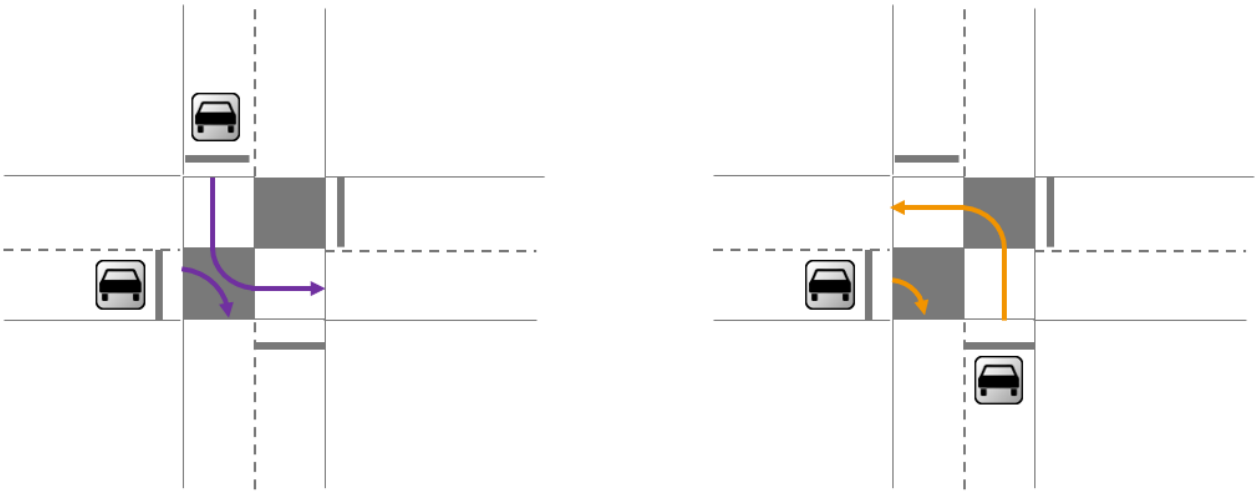


Figure 22 - 2x2 crossing area

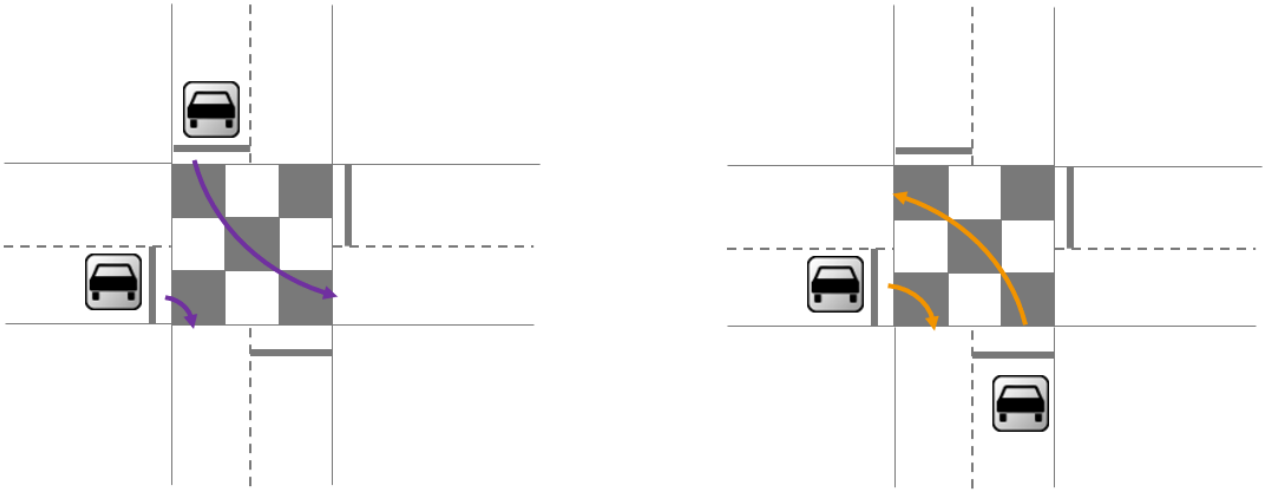


Figure 23 - 3x3 crossing area

After that the crossing area has been sized, it has been decided how many vehicles for each direction must be considered. In order to design an optimizer which does not assign a preference to a direction with respect to another one, it must be considered the same number of vehicles for each direction. Following the Frey's implementation of the chess board, a squared bitmap has been selected.

To understand how many bits are needed to implement all the intersection area based on the number of considered vehicles, the following formula has been derived:

$$Number\ of\ required\ bits\ = (2 * V + 3)^2$$

Equation 1 - Required bits to represent an intersection with V vehicles for each direction

Where V is the number of vehicles considered to be optimized for each direction. The following table shows the tested values:

	Vehicles per direction	Required bits	Standard bits
Number	1	25	64
Number	2	49	64
Number	3	81	128
Number	4	121	128
Number	5	169	256
Number	6	225	256
Number	7	289	512
Number	8	361	512
Number	9	441	512
Number	10	529	1024

Table 3 - Number of bits required for number of vehicles for each direction

Considering that a squared number must be selected among the standard ones, the admissible number of vehicles per direction can be 6 or 10. The representation of 6 vehicles has been chosen to better handle the bitmap creation during its implementation.

For simplicity the vehicle heading will be described using cardinal direction, so that a vehicle which is at East means that it is approaching the cross intersection from East towards West.

5.1.1 Bitmap implementation

The code implementation of the bitboard is done using the C++ *bitset* class of the standard library which emulates an array of bool elements but optimized for space allocation and in general occupies only one bit. The bitset representing the intersection it is defined as shown below.

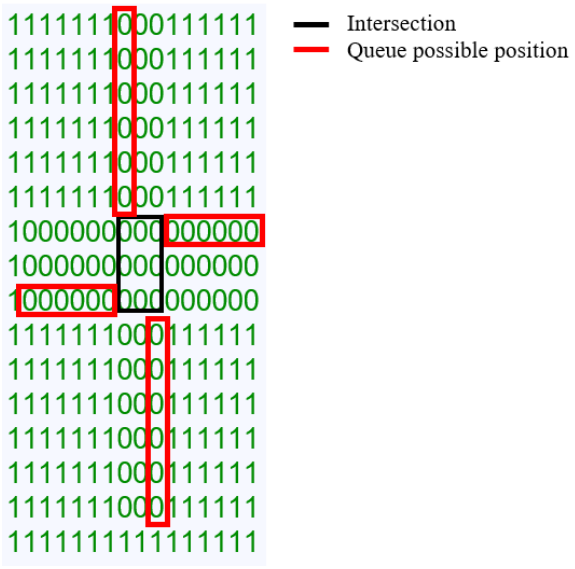


Figure 24 - Bitboard with intersection area and queue positions highlighted

As it is possible to see the intersection is represented using zeros, which defines allowed positions, and ones, which identify forbidden positions. The shown image is the bit series used to build the bitset string used in the code where the top left corner represents the Most Significant Bit (MSB) and the bottom right the Least Significant Bit (LSB). The knowledge of which is the LSB is important because the bitset template builds itself starting from a string which its first value is the LSB.

In the *Figure 24* has been highlighted the possible position which vehicles can occupy in the map. When a vehicle occupies a position in the bitmap, the bit is set to 1, so that for other vehicles is forbidden the movement in that position.

5.1.2 Allowed vehicle's movements

Once described the intersection and the position occupied by the vehicles, the possible vehicle's movements have been defined. Only four movements can be done by each vehicle and they have been associated with a number so that:

- 1 = Stop;
- 2 = Turn right;
- 3 = Turn left;
- 4 = Go straight on.

The only common signal representation is the Stop that is implemented as the crossing intersection are composed by all zeros. All other movements implementation depends on the direction of the vehicle, so they have been implemented separately.
As example implementation movement for East vehicles has been shown below.

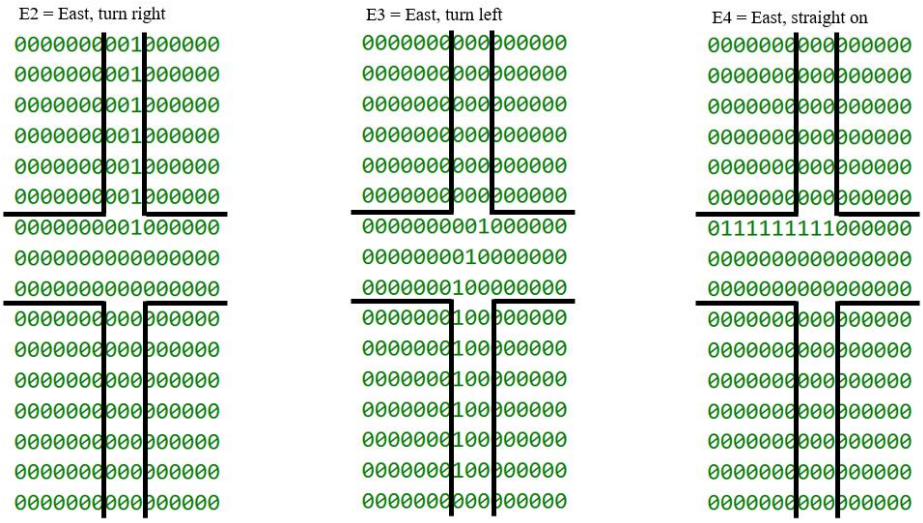


Figure 25 - Possible movements for East vehicles

The selected implementation method uses the shown standardized movements, without considering the vehicles position. The integration of vehicle's position and movement is described in the following section.

5.1.3 Testing vehicle's movements

Using bitset template allows to adopt all bitwise operations and test in an efficient way allowed movement. The masking method has been used heavily to extract information to the bitboard and test for the needed information.
During the implementation of the function which allows to test if a movement of a considered vehicle is legal, the vehicles heading used in the simulation has been considered. In particular Veins assigns the following heading, in radiant angles, to vehicles approaching the intersection from:

- East = -3.14;
- North = -1.57;
- West = 0.0;
- South = 1.57.

It is important to take care of the heading of each vehicle because the corresponding representation of movement changes.

The function *isPossibleMovement* takes care of testing if the desired movement can be performed or not. It checks the vehicle heading and if it is the first in its direction; on the contrary the function returns false. If the vehicle is the first, then the function masks the bitmap with the intended movement; if no bits set to 1 are present in the masked bitset, then the function returns true. The pseudocode for one direction is presented below.

```
bool isPossibleMovement(heading, signal, position)
{
    if (heading == East && isTheFirst(heading, position) == true)
    {
        if(signal == 2){
            maskedBitmap = Bitmap & signalE2;
            if(maskedBitmap.any() == false) {return true;}
            else {return false;}
        }
        if(signal == 3){
            maskedBitmap = Bitmap & signalE3;
            if(maskedBitmap.any() == false) {return true;}
            else {return false;}
        }
        if(signal == 4){
            maskedBitmap = Bitmap & signalE4;
            if(maskedBitmap.any() == false) {return true;}
            else {return false;}
        }
    }

    ...
}
```

Based on the presented function, the *checkAndSet* method has been designed. Vehicles movements are set in the bitboard just using an OR operator and reassigning the result to the bitboard itself.

Having described the most relevant methods, in the next section is possible to go through the working principle of the optimization.

5.2 Working principle of the optimizer

The aims of the optimization function are:

- Maximize the number of vehicles crossing the intersection for each iteration;
- Minimize the number of starts and stops.

During the implementation only data available in the vehicle storage can be used. Moreover, the output must be produced as fast as possible in order to avoid that vehicles have to stop to wait the result of the optimization process.

Starting from these requirements, a function which tests all possible combinations of *queue positions* and *vehicles movements* for each vehicle in each direction has been created. Saving only the legal solutions, it is possible to have a dataset among which a recursive search runs. The output contains many solutions and they are organized based on the number of iterations needed to empty the intersection. Among the solutions which require the minimum number of steps, meaning that at each iteration the maximum number of vehicles cross the intersection, the one which requires the minimum number of starts and stops is selected and provided as final output.

In the following sections a more detailed description has been provided.

5.2.1 Research of all possible combinations

During the design phase, there were two possible alternatives about how to implement the algorithm based on an input dataset of vehicles' data:

- Test at run time all legal movements of that dataset, order the solutions to maximize the number of vehicles crossing the intersection and select the solution with the minimum number of start and stop;
- Test a priori all legal movements for all the possible combinations, at runtime load as a variable all possible solutions, at runtime search among the solutions which are the one

fitting an input dataset ordering the solutions to maximize the number of vehicles crossing the intersection and select the solution which minimize the number of start and stop.

The second solution has been chosen to be implemented in this project. This because has been seen that having a big input dataset, the search algorithm to test legal solutions can last also minutes, depending on the used hardware.

To explain in a clear way the following steps, the "*circle of vehicles*" must be defined. This term will be used to indicate a group of vehicles which have the same queue position. As the example in the *Figure 19* there are 6 circles of vehicles, the one composed by all vehicles in first position, the one composed by all vehicles which are in queue position number 2, etcetera.

To build the set containing all the combinations of legal movement among which the recursive function can search for possible solutions, all possible combinations have been tested. The algorithm searches for a possible solution among the first circle of vehicles. For almost all solutions some vehicles cannot cross the intersection. In this case the function stores the tested solution and updates the bitmap taking care of the vehicles which crossed the intersection. The pseudocode describing how to test all possible movements for all possible directions for all possible vehicles position is shown below.

```
for(heading_configuration)
{
    for(queuePosition1) {
        for(queuePosition2) {
            for(queuePosition3) {
                for(queuePosition4) {

                    for(i-th_signalVector) {
                        for(j-th_signalVector) {
                            for(k-th_signalVector) {
                                for(z-th_signalVector) {

                                    if(isPossibleMovement) {
                                        myChess.SetSignal();
                                        SetSolution2Map(TestedSolution);
                                    }
                                }
                            }
                        }
                    }

                }
            }
        }
    }
}
```

Figure 26 - Pseudocode for testing all possible configuration of the vehicles in the intersection

The output is organized in circle of vehicles. Each vehicle in the circle is identified by two parameters:

- Queue position (1,2,3,4,5,6);
- Signal corresponding to the vehicle movement (1,2,3,4).

If the cars in the circle of vehicles produce a legal solution, then their signal is associated to them in the output, otherwise their signal is set to 1, meaning that they have to stop.

Some results are shown below in order to clarify the output organization.

```
(-3.14, -1.57, 0, 1.57)
    {{ {1,1},{1,1},{1,1},{1,1} }},
    {{ {1,1},{1,1},{1,1},{1,2} }},
    {{ {1,1},{1,1},{1,1},{1,3} }},
    ...
    {{ {2,2},{1,2},{3,2},{4,1} }},
    {{ {2,2},{1,2},{3,2},{4,2} }},
    ...
    {{ {2,6},{6,1},{6,1},{1,2} }},
    {{ {2,7},{6,1},{6,1},{1,1} }},
    ...
    {{ {3,4},{1,1},{6,2},{5,2} }},
    {{ {3,4},{1,1},{6,4},{5,1} }},
    {{ {3,5},{1,1},{6,2},{5,1} }},
    ...
    {{ {6,7},{6,1},{6,1},{6,1} }},
    {{ {6,7},{6,1},{6,1},{6,2} }}
```

It is formatted as:

(East, North, West, South)

{{ {(q_{h1},s_{h1}},{q_{h2},s_{h2}},{q_{h3},s_{h3}},{q_{h4},s_{h4}} }}

Where:

q = queue position

h = heading

s = signal

After having all possible vehicles configurations, they have been saved in XML format. In this way it is possible to load this dataset for each vehicle in the simulation when the vehicle is created. This map containing all solutions has been called *mapDataset* and an example of the XML implementation is shown below:

```
<mapkey>
    <key1>-3.14</key1>
    <key2>-1.57</key2>
    <key3>0</key3>
    <key4>1.57</key4>
    <element>
        <h1>1</h1>
        <s1>1</s1>
        <h2>1</h2>
        <s2>1</s2>
        <h3>1</h3>
        <s3>1</s3>
        <h4>1</h4>
        <s4>1</s4>
    </element>
    <element>
        <h1>1</h1>
        <s1>1</s1>
        <h2>1</h2>
        <s2>1</s2>
        <h3>1</h3>
        <s3>1</s3>
        <h4>1</h4>
        <s4>2</s4>
    </element>
```

Each *element* child represents a solution.

5.2.2 Optimizer implementation

The optimizer needs to know which are:

- the heading
- the position in the queue
- the desired direction

of each vehicle involved in the optimization. These data will have to be collected during the simulation to allow the algorithm execution but, in this development phase, it is supposed that they have been collected and ready to feed the base function of the optimizer. This function is called *possibleSolXCircle* and, fed with a circle of vehicles, it searches in the *mapDataset* for all sets of solution able to empty the intersection.
The output of the function is a map which its key value is the number of iterations required to empty the intersection, while its value is a vector containing possible solution sets. Often the algorithm finds different sets of solutions with the same iteration number. This happened because there exist situations in which choosing a different crossing order does not always produce a different number of steps to empty the intersection.

The development of *possibleSolXCircle* has been done starting from an input dataset which takes care of testing: various movements, empty directions and configurations which cannot be solved using just one solution.
The development dataset fed to the function is composed by three vehicles:

- a vehicle approaching the intersection from East that must turn left;
- a vehicle approaching the intersection from North that must turn left;
- a vehicle approaching the intersection from West that must turn right;

which describes the shown situation.

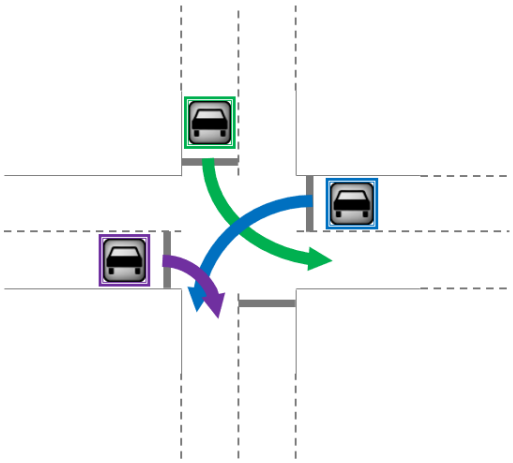


Figure 27 - Input data set representation for designing possibleSolXCircle function

In this and analogous situations, the algorithm inserts an *auxiliary vehicle* each time an empty direction is founded. The auxiliary vehicle's queue position value and its signal are always set to 1, representing for the algorithm a vehicle in first queue position whose intention is to stop at the intersection.

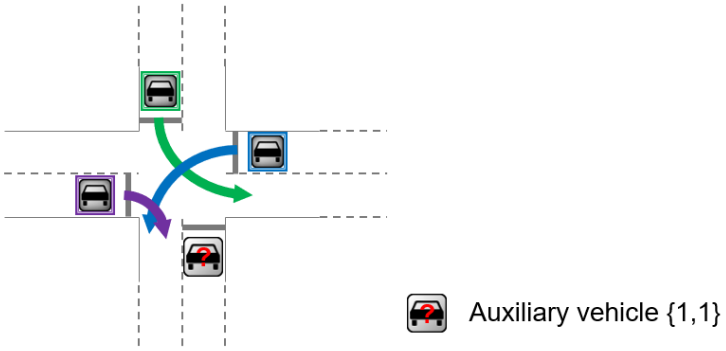


Figure 28 - Representation of auxiliary vehicle insertion

Without the auxiliary vehicle with these values (since no information in that direction has been provided) the *possibleSolXCircle* function selects as admissible solution all possible combinations in that direction, providing in this way a huge number of wrong solution sets. The auxiliary vehicle is in this way a stratagem to limit the search function in the *mapDataset* and be sure that undesired solutions are not provided.

Testing the development dataset, three solutions which requires three iteration to empty the crossroad have been found and these imply that just one vehicle at time can cross it. In the output there are also two sets of solution which requires only 2 steps. Both these sets let cross the intersection at the same time the vehicle from the North and form the West and the difference among the two solution is if the vehicle from the East crosses the intersection before or after the others.
An output example of this second case is shown below.

```
Number of steps: 2

  W    N    E    S
{1,1}{1,3}{1,2}{1,1}
{1,3}{1,1}{1,1}{1,1}

{1,3}{1,1}{1,1}{1,1}
{1,1}{1,3}{1,2}{1,1}
```

As it is possible to see the auxiliary vehicle values may correspond to a real vehicle in first queue position which has to stop in the considered iteration. This behavior has been taken into account in the implementation part of the algorithm.

5.2.3 Recursive implementation of the function

The *possibleSolXCircle* function has been used to test recursively the vehicles at the intersection. The *recursive* function takes as argument:

- The set of vehicles to be tested organized in circles;
- The i-th tested solution;
- The map containing all solutions set (*fullSolutions*);
- The *mapDataset;*
- The escape Boolean;
- The actual time converted in double.

Been a *pass by reference* function, the solutions are stored in the *fullSolution* variable which is a map whose key is the number of iterations needed to empty the intersection and the value contains all possible solution sets of that dimension.
Since testing all possible combination can take a huge amount of time, some assumptions have been done to limit the solutions exploration. In particular:

- Among the *possibleSolXCircle* output solutions, only the ones which require the minimum number of steps to solve the circle has been considered. The aim is to investigate only the solutions which maximize the number of vehicles which cross the intersection;
- Maximum two branches can be generated for each tested solution;
- If a new solution is not set in the *fullSolution* map for 2 seconds the function returns.

By using these assumptions, the optimization execution extremely speeds up and avoids that the testing tree grows without control.

Optimization development

```
void recursive(vehicleMap, testedSolutions, fullSolution, escape, time) {
    if(escape == true){ return;}
    if(isEmpty(vehicleMap) == true){

        insert_the_solution_in_fullSolution_map;
        if((actual_time - time) > 2){ return; }
    }
    else{
        solutionMap = possibleSolXCircle(vehicleMap.at(first_circle), mapDataset);
        reverse_iterator it = solutionMap.rbegin();
        auxiliary_solution_container = {};
        if(it->second.size() > 2){
            auxiliary_solution_container = {last, penultimate};
            for(i-th element in auxiliary_solution_container){
                //save configuration for the next iteration
                testedSolutionBefore = testedSolution;
                //store the actual solution for the current iteration
                testedSolution.push_back(i-th element);
                //saving the map configuration for the next iteration
                vehicleMapBefore = vehicleMap;
                //update the bitmap with the current solution
                updateMap(vehicleMap, i-th element);
                recursive(vehicleMap, testedSlution, fullSolution, mapDataset, escape, time);
                //Resetting configuration for the actual iteration
                vehicleMap = vehicleMapBefore;
                testedSolution = testedSolutionBefore;
            }
        }
        else{
            for(i-th element in solutionMap){
                //save configuration for the next iteration
                testedSolutionBefore = testedSolution;
                //store the actual solution for the current iteration
                testedSolution.push_back(i-th element);
                //saving the map configuration for the next iteration
                vehicleMapBefore = vehicleMap;
                //update the bitmap with the current solution
                updateMap(vehicleMap, i-th element);
                recursive(vehicleMap, testedSlution, fullSolution, mapDataset, escape, time);
                //Resetting configuration for the actual iteration
                vehicleMap = vehicleMapBefore;
                testedSolution = testedSolutionBefore;
            }
        }
    }
}
```

The *recursive* function collects all results in a map, based on the number of iteration needed to epty the crossroad. To maximize the number of vehicles crossing the intersection at the same time, the solutions set which has the lowest index has been chosen.

To identify which is the best set of solutions, start and stop signals have been set to the *fullSolution* in order to minimize them.

5.2.4 Setting start and stop and choose the solution

After that the *fullSolution* map has been filled with the recursive function, start and stops counters have been set to each solution using the function *setPresetSignal*.

Start and stop have been developed as two counters for each vehicle:

- The startCounter = 1, since vehicles are considered in motion approaching the intersection;
- The stopCounter = 0, since no stop happened.

They rely on two flags *m_start* and *m_stop* which manage the counters incrementation avoiding its growing when not needed. In fact, using these flags the startCounter incrementation is avoided if the vehicle is already in motion, the stopCounter incrementation is avoided if the vehicle has already stop. The workflow is shown below.

Figure 29 - Workflow of the start and stop counter

Starting from the *fullSolution* output, the *setPresetSingal* starts from the vehicle in first queue position and applies its signal to the vehicle itself but also to the other in the queue after it. This process is performed for all vehicles in all queue position.
In addition the algorithm can distinguish among real vehicles and auxiliary vehicles. If auxiliary vehicles are present, their counters will be equal to the counters of the last real vehicle.
Having data organized in this way, by computing the sum of the start and stop counters of the last row of the solution and dividing them by 2, it is possible to obtain the number of times that a vehicle stops, since all vehicles has the start counter which begins from 1.
The selected solution is the one which has the lowest start and stop value.

Before explaining how the optimization process has been implemented, an overview on how the simulation environment has been modified to take care of signal attenuation effects has been presented.

6. Introduce Fading and Shadowing effects in the simulation

To implement attenuations effects in the simulation, buildings surrounding the streets have been introduced. They have been designed as continuous buildings that run along the streets. They have been designed using a CAD software to extract the polygons' vertexes coordinates needed to describe their shape in OMNeT++.

6.1 Design obstacles

The quoted designed geometry at the intersection is shown below.

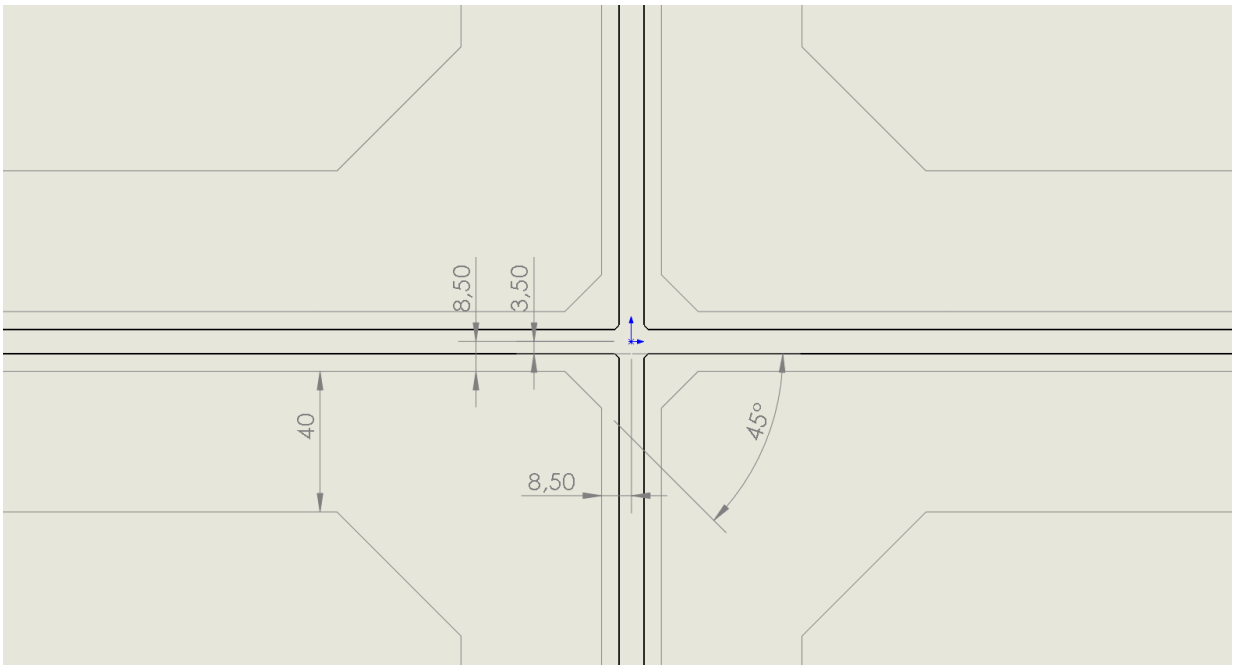
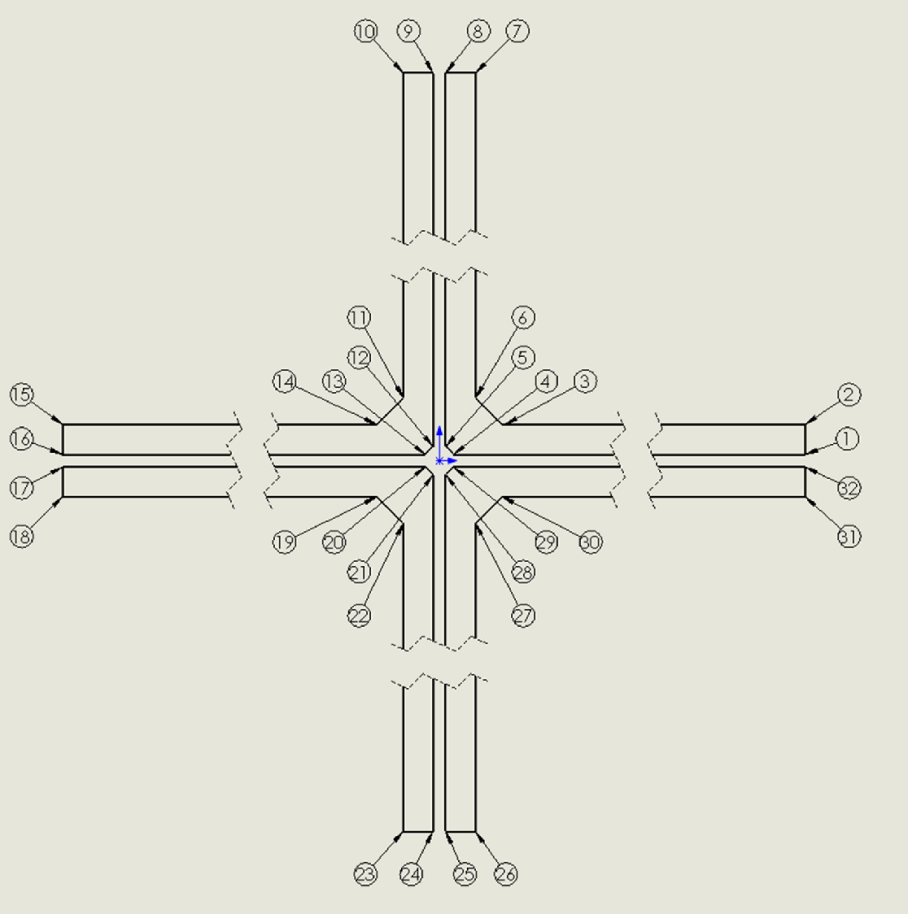


Figure 30 - Buildings design with respect to the road

The vertexes coordinates have been reported below.



Reference	X [m]	Y [m]
1	1000	8,5
2	1000	48,5
3	83,8	48,5
4	18,9	8,5
5	8,5	18,9
6	48,5	83,8
7	48,5	1000
8	8,5	1000
9	-8,5	1000
10	-48,5	1000
11	-48,5	83,8
12	-8,5	18,9
13	-18,9	8,5
14	-83,8	48,5
15	-1000	48,5
16	-1000	8,5
17	-1000	-8,5
18	-1000	-48,5
19	-83,8	-48,5
20	-18,9	-8,5
21	-8,5	-18,9
22	-48,5	-83,9
23	-48,5	-1000
24	-8,5	-1000
25	8,5	-1000
26	48,5	-1000
27	48,5	-83,8
28	8,5	-8,5
29	18,9	-8,5
30	83,8	-48,5
31	1000	-48,5
32	1000	-8,5

Figure 31 - Vertexes coordinates

Once that the buildings coordinates have been implemented in the XML file, the attenuations of the signal have been introduced, considering the designed obstacles.

6.2 Implementing signal attenuations

The Veins project already implements different *models* for signal attenuation. During the simulations three of them have been used. They are the following:

- Simple Path Loss Model which computes the attenuation of the signal in spaces without obstacles. It works taking into account the distance between the sender and the receiver and scales the transmission power into a logarithmic way of a certain amount of dB per meters.
- Simple Obstacle Shadowing Model which takes into account obstacles between the sender and the receiver. The transmission power is reduced by some dB per meters discerning

between reduction caused by walls and reduction caused by transmission through the building.

- Nakagami Fading Model which is used to model attenuation due to fading interference. The fading is a variation of the signal attenuation caused by fast variation of the signal produced by multipath propagation. Nakagami model represents these variations of the signal with a random attenuation created according to the Gamma distribution.

Thanks to these implemented models, the simulations represent more in detail an urban usage of the developed algorithm.

In the next section it is possible to see how the algorithm has been implemented.

7. Implementation of the optimization process

Once that the optimizer has been developed, the simulation environment has been enhanced and the signal attenuations models have been implemented, the algorithm which allows to use the optimizer has been developed into the Veins project.
As explained in Chapter 4.6, each leader searches in its storage data which vehicles have a distance to the intersection lower than 100m and encode them into a bit-format string, finally the leader set the string in the *leaderStringBSM* field of the BSM. By doing so the procedure to run the optimization starts. In fact, when a leader receives a message containing the *leaderStringBSM* field which is not empty, a procedure starts in order to define a unique data set to be used by the optimizer.

7.1 Leaders' dataset fusion

To allow vehicles to read the *leaderStringBSM* field of the BSM, an additional function has been introduced in the *updateStorage* method. This function has been called *leaderCheckOptimization* and it is accessible only by leaders. The function firstly checks if the received message contains a not empty *leaderStringBSM* field. If it is so, it is tested if the vehicle ID is coded into the bit-format string. Moreover, if the test returns a positive answer and the message has not been previously received, it is saved.
When a leader collects all bit-format messages from all leaders, it extracts from all messages the coded vehicles data contained into them. Then data are reorganized into a unique data set, removing duplications and verifying queue positions, and the information sent by the actual leader. Finally, the obtained vehicles set is coded into a bit-format string and assigned to the *stringFlagBSM* field of the BSM, ready to be sent to all vehicles.

The flowchart below outlines these steps.

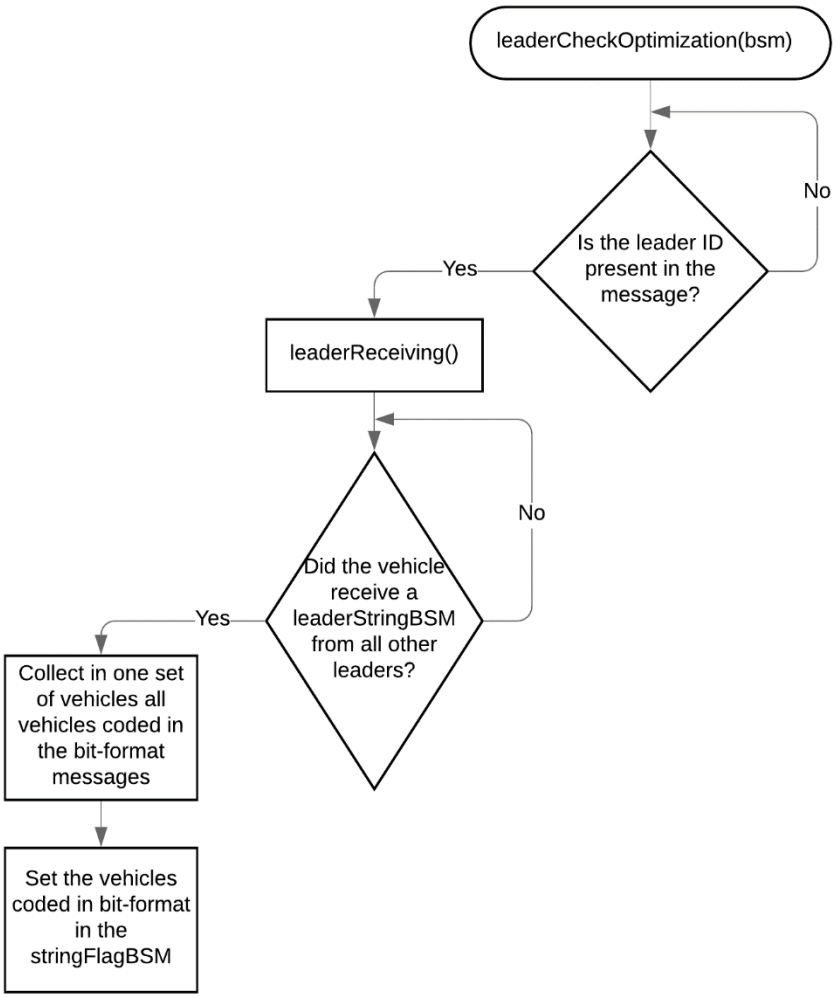


Figure 32 - Flowchart of leader's dataset fusion

This agreement procedure ensures that the set of vehicles among which the optimization has to be performed is unique. This is important because obstacles limit the communications among vehicles which approach the crossroad from different directions. In fact, it is possible to understand from the simulated data that vehicles communicate easily with the others in the same direction, whereas communication among vehicles from different directions is only possible in proximity of the intersection.

Having an agreement procedure allows to overcome the communication restrictions due to surrounding buildings and perform the optimization among a common data set of vehicles.

7.2 Execution of the optimization process

Once that the *stringFlagBSM* is set and the message is broadcast, the main function which manage the optimizer has been developed. It has been called *allVehicleCheckOptimization* and it is mainly responsible for:

- The optimization counter;
- Run the optimization.

A counter which takes care of enumerating how many times the optimizer is executed has been implemented to allow the algorithm to manage multiple optimizations. In fact, implementing just a flag to identify if a vehicle has been optimized is not enough to discern if it has been optimized in the current optimization or in another one. Having a counter allows to let the *speedManaging* function to understand if previously optimized vehicles cross the intersection or not.
The implementation of the counter relies on the received *stringFlagBSM* message. Each vehicles data in the storage has a container aimed to collect all messages received. Each new message is added to the containers. If some vehicles are not present in the storage, they are added to the storage and their containers are filled with the previously received messages of the other vehicles contained in the storage. If a vehicle is among the ones contained in the set to be optimized, and it has not been optimized yet, its optimization counter is set equal to the size of the container.

The *allVehicleCheckOptimization* function algorithm test if the received *stringFlagBSM* is new or has been already received. If it is new the *intersectionFlag* of the vehicles of the previous optimization is test, if it is not set for all vehicles than it is set to true. This is done because it can happen that not all vehicles get directly in contact with others. When a message contains a new *stringFlagBSM* guarantees that new leaders have been elected, in fact this field of the message can be updated only after a new agreement procedure among leaders, meaning that the previous ones have crossed the intersection. Moreover, due to communication restriction caused by surrounding buildings, some vehicles may not receive messages from others which crossed the intersection. This situation implies that some vehicles do not set the *intersectionFlag* of vehicles which already crossed the intersection. To face the problem when a new *stringFlagBSM* is received, the intersection flag of vehicles of the previous optimization is set.

Then the algorithm continues extracting from the bit-format string the set of vehicles for the optimization. Moreover, if the considered vehicle has not been already optimized, the previously described procedure to set the optimization counter is performed. Finally, if the vehicle is among the ones to be optimized, the optimizer is executed.
The variable which stores the set of vehicles used in the optimization has been defined in the *.h* file, allowing to use its data in the *speedManaging* function, in this way it is possible to control the crossing turn of vehicles.
The flowchart representing the *allVehicleCheckOptimization algorithm is shown below.*

Implementation of the optimization process

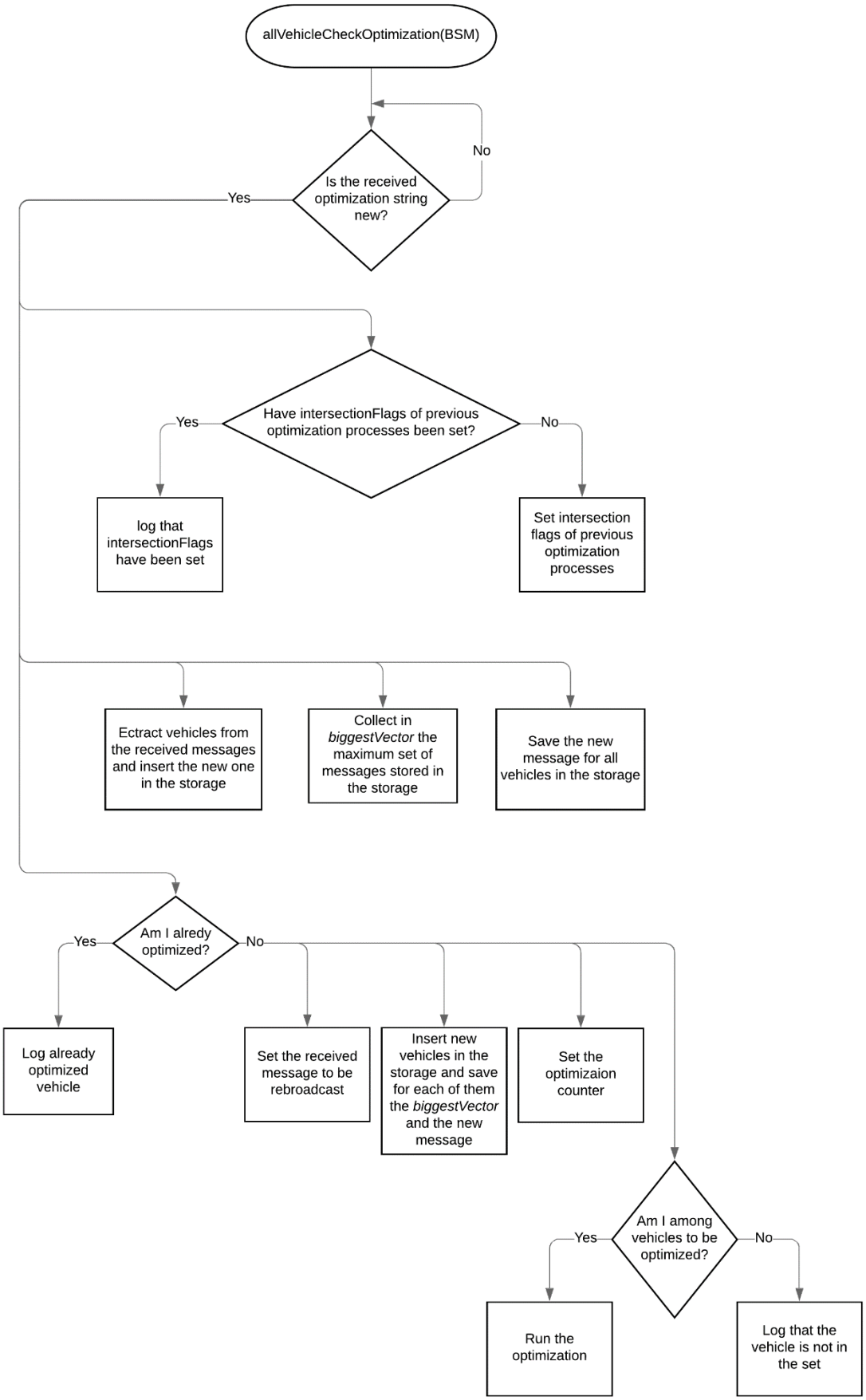


Figure 33 - Flowchart of the developed optimizer

8. Simulation results

The implemented management model has been compared with the same intersection managed using a traffic light which phases has been designed using the F. V. Webster method [9] and with an intersection without regulation.

8.1 Webster's method

F. V. Webster developed a methodology for estimating the optimum setting for fixed-time traffic light signals. It is a method currently used by civil engineers when they have to design the signals phases of this type of traffic light.

Before explaining how the method has been used, some terms must be defined:

- *Cycle length* is the time taken by a signal to complete one full cycle. It is composed by:
  - *Green interval* (the time in which the signal is green);
  - *Red interval* (the time in which the signal is red);
  - *Change interval* (the time in which the signal is yellow);
  - *Clearance interval* (the time interval in which all traffic lights have a red signal to enable vehicles to clear the intersection).
- *Saturation Flow (S)* describes the maximum number of vehicles able to cross the intersection in one hour, considering a homogeneous flow of vehicles without stop. It is measured in vehicles per hour.
- *Lost time (L)* describes all time losses which vehicles in queue have to deal with. It is measure in second.
- *Critical flow rate (y_i)* describes the ratio between the estimate *volume of vehicles* (v_i) for one direction and the saturation flow.

- *Optimum cycle length (C_0)* describes which is the duration of the cycle time which cause the least delay to all the traffic. It is computed using the formula below and it is measured in second.

$$C_0 = \frac{1.5 * L + 5}{1 - \sum_i y_i}$$

Equation 2 - Optimum cycle length

- *Green time (G_i)* describes which is the optimum green interval for each traffic light. It is measured in second and the formula is provided below.

$$G_i = \frac{(y_i)}{\sum_i y_i} * (C_0 - L)$$

Equation 3 - Green time formula

Having defined the definitions used in the Webster's method, it is possible to describe how parameters have been obtained, computed and implemented to allow the comparison of the results.

8.2 Computation of parameters

The main data needed to apply this method are the saturation flow and the volumes of vehicles. The saturation flow has been obtained feeding the simulation environment with a continuous flow of vehicles which cross the intersection going straight on. Fifty-two vehicles have been fed in the simulation and they take 159.5 seconds to empty the intersection, then this result has been scaled to obtain the flow in one hour, obtaining that:

$$S = \frac{52 * 3600}{159.5} = 1173.6 \rightarrow 1174 \left[\frac{vehicles}{hour}\right]$$

Equation 4 - Saturation flow of the simulation

The estimated volumes of vehicles have been obtained by referring to the flow rate used in the J. He paper [10]. The choice to use those data depends on the fact that for this project no experimental data has been collected from a real intersection. Considering that the intersection considered by J. He is comparable to the presented project, those flow rate has been used to scale the estimated volumes. The obtained values and the computations used have been reported below.

Simulation results

$$v_{N\,\&\,S} = 360\ [vehicles] \qquad v_W = 360\ [vehicles] \qquad v_E = 480\ [vehicles]$$

$$y_{N\,\&\,S\,\&\,W} = \frac{360}{1174} = 0.31 \qquad y_E = \frac{480}{1174} = 0.4$$

Equation 5 - Critical flow rate for North, South and West vehicles

Equation 6 - Critical flow rate for East vehicles

The loss time L has been chosen to last for 2 seconds for two phases, so:

$$L = 2 * 2 = 4\ [s]$$

Equation 7 - Lost time equation

Once these values have been obtained, the optimum cycle length has been computed.

$$C_0 = \frac{1.5 * L + 5}{1 - (y_1 + y_2)} = \frac{1.5 * 4 + 5}{1 - 0.71} = 37.9 \rightarrow 38\ [s]$$

Equation 8 - Computing the Optimum cycle length

The green times for the two phases are:

$$G_{N\,\&\,S} = \frac{0.31}{0.31 + 0.4} * (38 - 4) = 14.84 \rightarrow 15\ [s]$$

$$G_{E\,\&\,W} = \frac{0.4}{0.31 + 0.4} * (38 - 4) = 19.15 \rightarrow 20\ [s]$$

Equation 9 - Computation of the green times for the phases

To avoid that SUMO alerts for emergency brake of vehicles during the simulation, a yellow phase has been added of 1s during the clearance interval. The final timing representation is shown below.

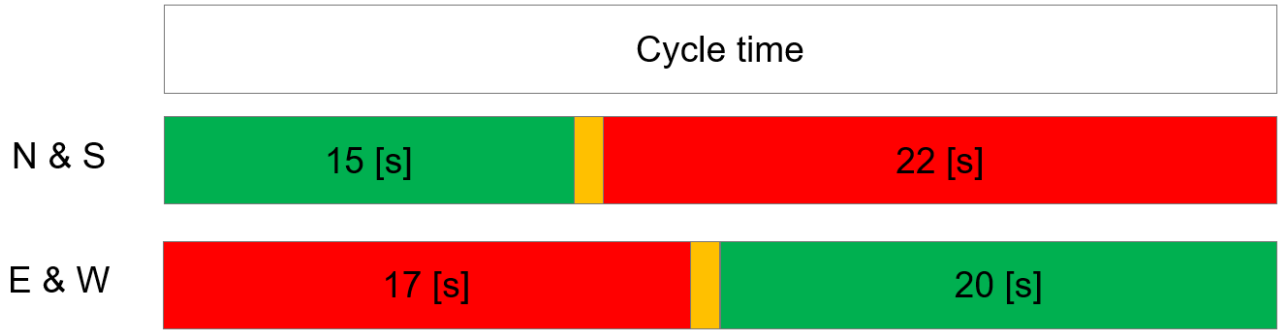


Figure 34 - Optimum cycle representation

The implementation in SUMO has been done modifying the traffic lights' phases with the computed one and the implementation is shown below.

```
<phase duration="14" state="GGGrrrGGGrrr"/>
<phase duration="1" state="yyyrrryyyrrr"/>
<phase duration="3" state="rrrrrrrrrrrr"/>
<phase duration="19" state="rrrGGGrrrGGG"/>
<phase duration="1" state="rrryyyrrryyy"/>
```

8.3 Results comparison

The comparison among the simulation is based on the time spent by the vehicles to empty the intersection. Ten simulations have been used to obtain average data. For each simulation the directions of the 66 simulated vehicles have been changed. The vehicles directions have been computed by means of a Python script which randomly selects values from an admissible range. Below is shown the table containing the results.

Set number	Right before Left [s]	Webster [s]	Optimization [s]
1	197.2	167.0	165.2
2	173.6	177.6	167.6
3	187.2	179.0	174.6
4	187.7	193.0	177.1

5	192.2	163.6	171.7
6	183.9	174.7	179.1
7	200.3	174.0	168.2
8	178.6	195.7	180.5
9	179.4	194.3	179.2
10	183.0	193.1	178.0
Average ->	186.3	181.2	174.1

Table 4 - Final results table

From the obtained results it is possible to deduce that both the traffic light and the developed method for intersection managing provide better results that an unregulated intersection. Instead between the traffic light and the presented method can be pointed out an average improvement. In fact, in just two cases the traffic light performs better than the thesis method and this can depend on the chosen vehicles directions, but it is also due to hazardous situations. Indeed, must be pointed out that simulations performed using unmanaged intersection (right before left priority) and using the traffic light with phases designed with the Webster method, allows situations that the developed algorithm does not. In particular, vehicles can stop in the middle of the intersection and authorize other vehicles to cross it as shown below.

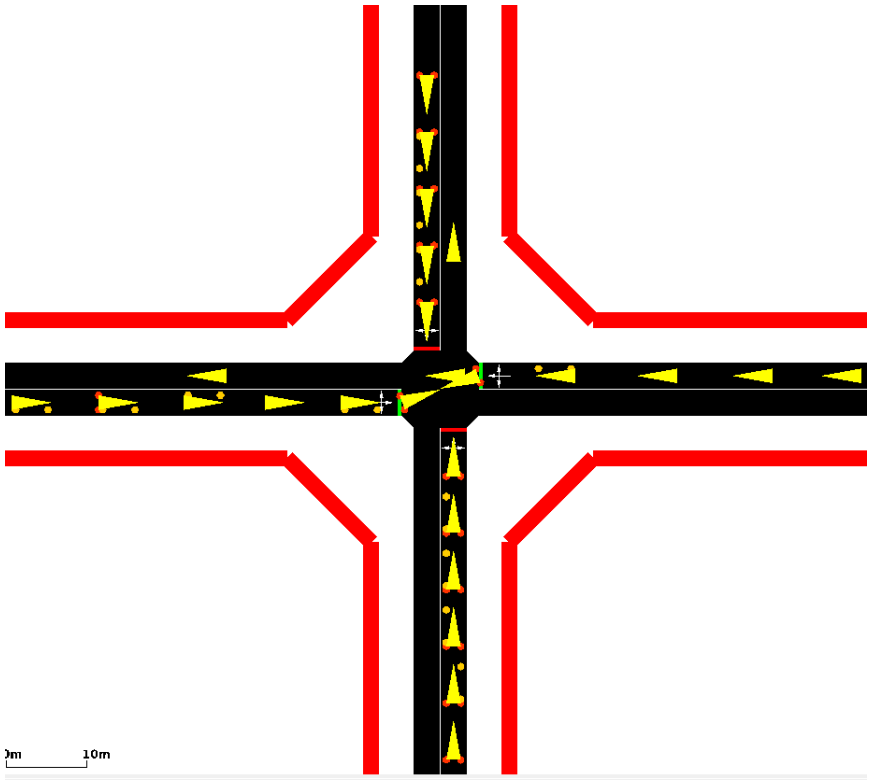


Figure 35 - Not allowed configurations in the optimized process

These types of vehicles configuration allow to speed up the empty process of the intersection. However, it has been chosen to avoid them due to the hazardous situation which can be generated by using them. Finally, these highly dynamic situations require additional information coming from different types of on-board sensors to be safely managed, while in this final project only basic information has been used.

The obtained results can be the thrust for further development and possible branches have been presented in the next chapter.

9. Future development

The developed algorithm can be used as starting point for other projects. Some possible implementations can be:

- Introducing the capability of managing multiple intersections;

9. Future development

- Introducing the mathematical description of the roundabout and its management;
- Introducing managing methods for multiple lanes;
- Introducing a management method for vehicles without communication capability.

Exploring all these possibilities it is possible to create an algorithm able to manage multiple situations. Once the algorithm has some of these extensions implemented, it is possible to test it using a map of a real urban area.

References

[1] F. Lennert *et al.*, “Strategic Transport Research Innovation Agenda (STRIA)- Smart Mobility and Services Roadmap,” p. 15, 2016.

[2] T. January, F. E. V Italia, and F. E. V Europe, “Simulation of Urban Mobility,” vol. 8594, 2014.

[3] A. Varga, “The OMNeT++ Discrete Event Simulation System.” Proc. European Simulation Multiconf. (ESM ’01), 2001.

[4] Scotiabank, “Number of cars sold worldwide from 1990 to 2019,” *Statista*. [Online]. Available: https://www.statista.com/statistics/200002/international-car-sales-since-1990/.

[5] “ISTAT vehicle statistics.” [Online]. Available: https://www.istat.it/it/archivio/225505.

[6] N. (Sweden), “Vision Zero Initiative.” [Online]. Available: https://trimis.ec.europa.eu/?q=project/vision-zero-initiative#tab-outline.

[7] “Towards a European road safety area: policy orientations on road safety 2011-2020,” 2010.

[8] P. W. F. Benjamin Mittman, *Chess Skill in Man and Machine*. Springer Science & Business Media, 2012.

[9] F. V. Webster, “Traffic signal settings,” *Gt. Britain Road Res. Lab.*, vol. Road Resea, 1958.

[10] J. He and Z. Hou, “Ant colony algorithm for traffic signal timing optimization,” *Adv. Eng. Softw.*, vol. 43, no. 1, pp. 14–18, 2012.