Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

Microscopic modelling and simulation of autonomous vehicle platooning on highways

Treball realitzat per:
Laura Oriol Herrera

Dirigit per:
Francesc Soriguera Martí

Màster en:
Enginyeria de Camins, Canals i Ports

Barcelona, 25 de Gener de 2019

Departament d'Enginyeria Civil i Ambiental

TREBALL FINAL DE MÀSTER

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Professor Francesc Soriguera, my research supervisor, for guiding me during the whole process and for bringing in continuous contributions to this project. My grateful thanks are also extended to Marcel Sala and Enrique Jiménez for the valuable and constructive suggestions during the development of this research work.

I would also like to thank Aimsun, the company I work for, for providing the tools and the flexibility to progress in parallel to the ongoing tasks at work. Specially to my work colleague Pablo Benavides, who has introduced me in the field of computer programming and taught me how to use this powerful knowledge.

Finally, I wish to thank my parents, friends and close relatives for their support and encouragement throughout my study.

ABSTRACT

Road space availability in cities can be rather limited and, in the future, it is expected to become even scarcer. Road infrastructures congestion is one of the major issues affecting cities worldwide. Thus, engineers and urban planners are challenged in studying the use of new Intelligent Transport Systems that could provide a solution to current difficulties. ITS solutions take advantage of the projected new technologies to find optimal solution for transportation.

This project aims to analyze the potential of car platooning to improve the overall road system performance. The concept of platooning refers to the linking of vehicles through wireless technology in groups of cars that travel very close one after another in highways. The leading vehicle controls the speed and direction, whereas the following vehicles respond to the leader's movement. Grouping vehicles into platoons might increase road capacity by reducing headways, safety by removing human errors and cost savings by reducing fuel consumption and emissions.

For platoons to be viable from the outset, the impact on the supporting infrastructure has to be minimal: platoons and autonomous vehicles need to coexist with human driven vehicles in a mixed freeway system.

To perform the platooning analysis from a traffic engineering perspective a platooning algorithm has been developed, programmed, implemented and simulated on a three-lane ring road of 1.5km length using Aimsun Next, a traffic simulation software. Results for different autonomous vehicles penetration rates in the market have been obtained and analyzed.

CONTENT

LIST OF FIGURES

LIST OF TABLES

RELEVANT ABBREVIATIONS

API	Application Programming Interface
AV	Autonomous Vehicle
CV	Connected Vehicle
CAV	Connected and Autonomous Vehicle
V2I	Vehicle-to-Infrastructure communication
V2V	Vehicle-to-Vehicle communication
V2X	Vehicle-to-Anything communication

1. Introduction

The future moves fast towards new technologies and the automotive sector is a great example of it. The rapid development of autonomous vehicles is a topic of deep concern for independent automobile companies, but also for governmental organizations and private individuals. At present, the latest commercialized car models include some on-board features that provide automated assistance to the driver, but full automation goes far beyond these tools. A full self-driving vehicle has to be able to drive under all conditions from its origin to its destination without any driver input or even without passengers.

Even if the technology is finally ready there are many legal and ethical implications that need to be carefully considered before a vehicle can drive without any human input. These questions will not be solved overnight, but at some point, it is expected that the technological developments coupled with several successful field tests will encourage governmental organizations to establish a legislation and increasingly authorize the presence of greater levels of automation in our roads.

It is clear that engineering and society are still some years away until reaching this point, but in the meanwhile intelligent traffic management strategies can be developed to guarantee success upon their introduction. Platooning is one of the set of technologies that self-driving cars might employ on highways to increase road efficiency. The fundamental idea of this mechanism is to benefit from wireless technology and other vehicle on-board sensors to remove human errors and enable cars to drive tight one after another at high speeds, into a trainlike group of cars. It is expected that the organization of vehicles into platoons, will increase the capacity of the existent roads and potentially improve safety.

Most of the research done in the filed considers dedicated infrastructure for platoons. Lanes might be restricted to autonomous vehicles when the penetration rate of autonomous vehicles is high, however for platoons to be viable from the outset, the impact on the supporting infrastructure has to be minimal. Platoons need to be able to operate on unmodified public motorways and interact with other road users. This research will study the potential and the associated challenges of grouping vehicle into platoons when sharing the road with non-autonomous vehicles.

2. State of the Art

There are two main areas of development in the domain of the automobile industry. Connected Vehicles (*CV*) and Autonomous Vehicles (*AV*).

Connected Vehicles are vehicles that can communicate with each other (*Vehicle-to-Vehicle, V2V*), with roadside infrastructure, such as traffic control signals or similar stationary structures (*Vehicle-to-Infrastructure, V2I*) or with other devices, such as mobile phones carried by road users (*Vehicle-to-Anything, V2X*).

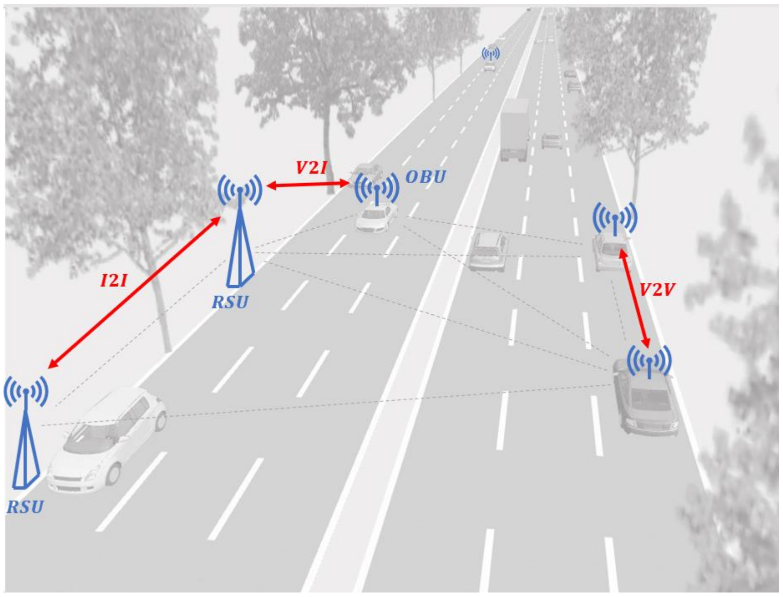


Figure 1. Connected Vehicles. Own Source.

The fundamental principle of connected vehicles is that they work as a mesh network, where every node (car, smart traffic signal, roadside device, etc.) can send, receive or retransmits signals. A pair of connected vehicles can therefore share information with each other about their speed, acceleration, location, direction and other similar messages that can help them to control their movements coordinately.

Autonomous Vehicles (AV) are vehicles that combine a variety of onboard sensors, cameras, GPS and other telecommunications that perceive and process information in real-time and analyze the data using artificial intelligence and other complex computer algorithms in order to allow the vehicle to navigate in a safe and appropriate way without direct driver input. Functions such as steering, acceleration/deceleration or the monitoring of environment are carried out by the vehicle in a fully autonomous vehicle.

The Society of Automotive Engineers, SAE, and the National Highway Safety Administration, NHTSA, provides a nomenclature with detailed definitions for six levels of driving automation, ranging from manually driven vehicles with no automation (*level 0*) to full driving automation (*level 5*). [7]

Level of Automation	**Terminology**	**Definition**
Level 0	No Automation	The driver is full in control of the vehicle at all times managing the operational and tactical aspects of driving tasks
Level 1	Driver Assistance	Some automated assistance through control of vehicle speed and lane position. At this level the driver is required to take over the vehicle at any instance, thereby requiring hands on the steering wheel at all times and feet near the brake and acceleration pedals.
Level 2	Partial Automation	The execution of steering and acceleration/deceleration are carried out by driver assistance systems. All other operational and tactical aspects of driving are carried out by the driver.
Level 3	Conditional Automation	The execution of steering, acceleration/deceleration and the monitoring of environment are carried out by the vehicles. When the vehicle is in control, the driver is not required to monitor at all times but is required to take control when required (upon request to intervene)
Level 4	High Automation	Driving Systems contribute to full self-driving under certain conditions. Under such conditions, e.g. urban ride sharing, the vehicle drives without human intervention. The role of the driver is only to provide the destination
Level 5	Full Automation	Full self-driving capabilities under all conditions. The vehicle can drive without any human input and without driver/passengers.

Table 1. Definition of different levels of automation. Source: SAE

The key difference between Connected and Autonomous Vehicles lies in the fact that a CV can hardly become fully autonomous as it depends on information received from the infrastructure and other surrounding vehicles through wireless network technology. It can provide some help to the driver but cannot drive the car by itself. On the other hand, AVs are independent of wireless communications and use on-board sensor-based technologies that would allow to control the vehicle without driver input. However, CAVs, Autonomous and Connected Vehicles, a combination of both sensor-based and connected technologies is compatible and could enhance performance by complementing the limitations of each other.

Platoons might be one of the beneficiaries of such combination. The vehicles in the platoon will use sensors and will be linked to each other through wireless V2V communication. In the field of platooning, most of the research has been done with trucks as they comprise a big interest for the goods industry. As in truck platooning, the formation of car platoons can happen in real time or be scheduled in advance. Variables like departure time, arrival time and route are mostly known in advance for trucks, therefore, a centralized platooning service provider could manage and plan optimal truck platooning solutions [3]. In the domain of private vehicles those variables are rather uncertain. Consequently, cars will most likely form platoons dynamically on the road without any prior planning.

Vehicular Ad-hoc Network, VANETs, is the spontaneous creation of a wireless network for V2V data exchange. The Institute of Electrical and Electronics Engineers, IEEE, specifies the technology suitable for vehicular communication networks. The defined IEEE 802.11p/1609.x families, are a suite of communication protocols to address vehicular communication networks. In terms of the present study there are two essential variables regarding the information transmission between two vehicles: communication range and latency. Communication range can be defined as the maximum distance between transmitter and receiver for normal operation and latency is defined as the time since the transmitter sends the information until it is successfully delivered to the receiver. In [13], real-world tests were reported in an uncontrolled approach to evaluate the IEEE 802.11p and LTE-based V2V communication in terms of end-to-end latency. The minimum latency observed is 10ms and the maximum communication range is 500m. In this project we will consider a latency of 100ms, 0.1s and the maximum communication range will be neglected. Anyway, it will be checked that the 500m are not exceeded.

The information sharing process between vehicles is a key point in the platooning system. There are different topologies that can be implemented to address the information sharing problem in a platoon. [9] In Figure 2, the most common topologies are presented.

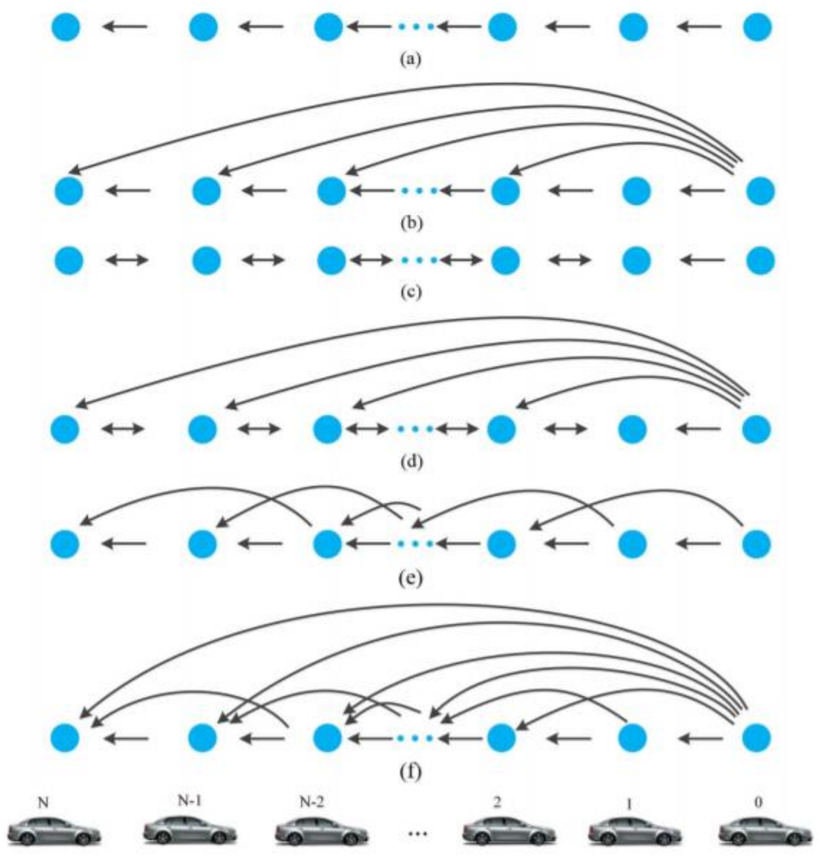


Figure 2. Possible communication topologies. Source: [10]

It is assumed that every vehicle in the platoon has its own controller. In order to make control decisions the controller needs information of all the platoon vehicles, or at least from the preceding ones, therefore topology (d) in Figure 2 is used, where every vehicle broadcasts its information to all other vehicles. Because of the communication latency, the received information will be 0.1s outdated. This delay and the related consequences will be addressed in the following sections.

3. Platooning Technology

3.1. Concept

The fundamental principle of platooning is to benefit from the available technology to enable vehicles to travel close one after another without compromising security. A platoon is composed by N + 1 vehicles, the platoon leader and the N following vehicles. There are therefore, two big roles in a platoon: platoon leader and follower.

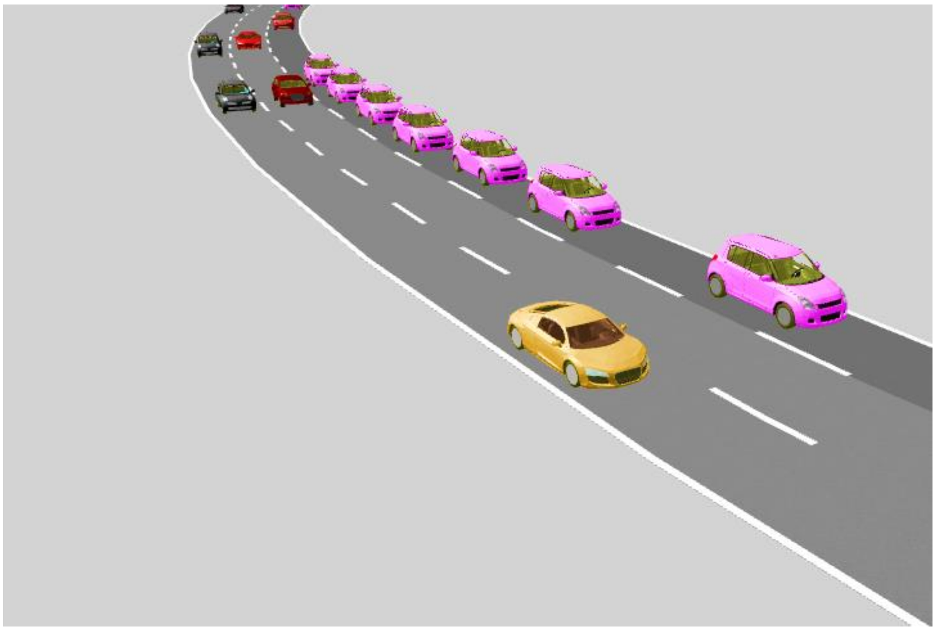


Figure 3. Platoon Illustration. Own Source.

As a platoon leader, the vehicle will travel like the rest of autonomous vehicle, using its sensors and any other available information obtained through wireless technology. A vehicle will take the leadership role if the preceding vehicle is a conventional car or if the preceding vehicle is not in its close proximity. Even if the communication range can reach up to greater distances, two vehicles will only join into a platoon if their distance is less than 25m. This is to ensure communication efficiency and to avoid long lasting accelerations. The platoon leader will adjust its speed to the traffic conditions and travel at a maximum speed equal to the speed limit. Once other vehicles start to join the platoon the leader will send the required information about its speed, acceleration or destination to its followers, which will follow according to a set of predefined rules.

As a follower, the vehicle goes through three stages in the platooning process, first of all, the platoon formation, where the vehicle starts to approach its preceding vehicle to join the platoon. Once the objective gap is reached the vehicles will travel as a unit in the platoon and finally the platoon dissolution, when the vehicle leaves the platoon.

3.1.1. Platoon Formation

During the formation process the vehicle will need to accelerate to approach the preceding vehicle. In order to do so the vehicle will be allowed to exceed by a 10% the speed limit. This is 110km/h. The formation of the platoon must be smooth to ensure the passenger's safety and comfort. Two vehicles driving one after another will be paired if they are inside the communication range and both are equipped with the necessary platooning technology. Cooperative Adaptive Cruise Control (CACC) uses radar-based systems to measures the distance to the predecessor and also exchanges information with it by wireless communication. This enables a vehicle to follow its predecessor at a closer distance under tighter control [11].

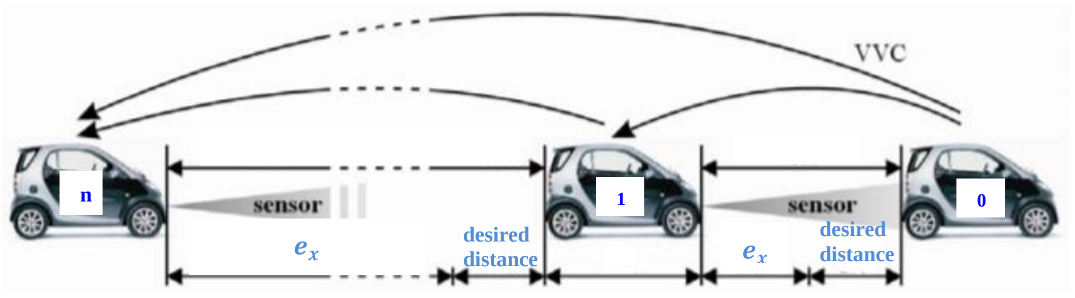


Figure 4. Schematic representation of a vehicle string with V2V communication and sensors. Adapted from [15].

The follower will approach the leader using the following CACC formula:

$$a_{follower} = k_1 \cdot a_{leader} + k_2 \cdot e_v + k_3 \cdot e_x \qquad with\ k1,\ k2,\ k3 > 0$$

$a_{follower}$	is the acceleration that the follower should apply in m/s^2
a_{leader}	is the acceleration of the leader in m/s^2
e_v	is the velocity error, equal to the relative speed between leader and follower in m/s
e_x	is the gap error, the difference between the desired gap and the current one in m
$k1,\ k2,\ k3$	are controller feedback loop gains for acceleration, velocity and gap error.

The values for those coefficients have been chosen to be:

$$\begin{cases} k_1 = 1.0 \\ k_2 = 0.3 \\ k_3 = 0.1 \end{cases}$$

The variable k_1 has been selected to take a value of 1 in order to apply the same acceleration as the leader, the follower will accelerate or decelerate in the same amount as the platoon leader at the time when the information was sent. The values for k_2 and k_3 represent the speed and space differences between leader and follower. The system slows down when it approaches a leader with a lower speed and accelerates if the space in between them is greater than desired. Selecting different values for k_2 and k_3, 0.3 and 0.1 have shown to be a good relation in order to approach the leader in a smooth way and with minimal speed difference. To enhance driving comfort and avoid abrupt movements the maximum accelerations during the process are constrained to -3m/s^2 and 1.5m/s^2. This process can be understood as a very efficient driver that reacts almost imminently to any acceleration of the leader and approaches it in a very smooth way by adjusting the speed to match the desired gap and the speed of the leader.

3.1.2. Platooning

When the objective gap has been reached vehicles will travel as a unit at the same speed. The speed of each joined vehicle is subject to minor adjustments to adapt the desired distance to the local leader when there is a change in speed.

To calculate the speed of each joined vehicle the following relation will be applied:

For the first following vehicle (i=1) after the platoon leader:

$$v_{joined,1}(t+\Delta t) = v_{platoon\ leader}(t) + \frac{(D_{1\_PL}(t) - DD_1(t))}{3.6} * 0.1$$

$$\begin{cases} v_{platoon\ leader}(t) = speed\ of\ the\ platoon\ leader\ at\ time\ t \\ v_{joined,1}(t+\Delta t) = speed\ of\ the\ first\ platoon\ member\ to\ be\ applied\ at\ time\ t+\Delta t \\ D_{1\_PL}(t) = distance\ to\ the\ platoon\ leader\ at\ time\ t \\ DD_1(t) = desired\ distance\ to\ the\ platoon\ leader\ at\ time\ t \end{cases}$$

For the following vehicle that occupies the ith position (i >2):

$$v_{joined,i}\,(t+\Delta t) = v_{joined,i-1}(t+\Delta t) + \frac{(D_{i\_(i-1)} - DD_i)}{3.6} * 0.1$$

$$\begin{cases} v_{joined,i}\,(t+\Delta t) = speed\ of\ vehicle\ i\ to\ be\ applied\ at\ time\ t+\Delta t \\ v_{joined,i-1}(t+\Delta t) = speed\ of\ vehicle\ (i-1)\ to\ be\ applied\ at\ time\ t+\Delta t \\ D_{i\_(i-1)} = distance\ to\ vehicle\ (i-1)\ at\ time\ t \\ DD_i(t) = desired\ distance\ to\ vehicle\ (i-1)\ at\ time\ t \end{cases}$$

The first platoon vehicle after the leader will target a greater desired distance than the following vehicles, $D_{i\_(i-1)} < D_{1\_PL}$. Those distances will be defined in section 3.2. This is to consider the communication latency, the 0.1s of delay since the information has been sent and received. This latency will only exist between the platoon leader and the first platoon follower. The remaining followers can be considered as a unit, as if the first follower would be a very long vehicle.

3.1.3. Platoon Split

At some point the vehicle will need to change its course and leave the platoon. Vehicles will be aware of their need to leave the platoon 1000m before the off-ramp. To avoid having all vehicles doing the lane-changing at the same position, a distance variability of 20% has been set. Therefore, vehicles will know they need to leave the platoon in a range of 800m–1200m before the exit (a uniform random distribution determines the value). As of that moment, the vehicle will start searching for a gap in the adjacent lanes. As long as the gap isn't enough to ensure a safe lane-changing the vehicle will continue in the platoon. If the vehicle hasn't been able to change lanes 400m before the exit, it will start to become more aggressive. The vehicle will slow down and wait for cooperation in the adjacent lane in order to accomplish the lane-changing. The heading part of the platoon will continue as intended while the following part will brake accordingly, if this process takes too long it might split the platoon into two new platoons. Generally, the anticipation will be enough and the lance-changing will be done without affecting the platoon. Once the vehicle has departed from the lane, the formation process will be applied again between the leader and the follower of the missing vehicle to close the existing gap.

3.2. Security Gap

In general, the way in which a vehicle reacts to changes in acceleration or deceleration of the surrounding vehicles, depends on the driver's own characteristics and perceptions. When the preceding vehicle decelerates, the driver might also rely on the tail brake lights of the preceding vehicle. In an emergency, a driver must bring the vehicle to a stop in the shortest distance possible. This distance will be determined by two factors, the braking distance and the reaction time.

The *braking distance,* $x(t_{stop})$, refers to the distance the vehicle will travel once the driver has hit the brakes until it comes to a complete stop. The main factors that affect the braking distance are the speed at which the vehicle is travelling, v_0, and the deceleration capacity of the vehicle, a. The movement equations of the vehicle can be easily defined using Newton's second law of motion:

$$\begin{cases} x\ (t) = x_0 + v_0 \cdot t + \dfrac{1}{2} \cdot a \cdot t^2 \\ v\ (t) = v_0 + a \cdot t \end{cases}$$

$$\Lsh \text{ if } v(t) = 0 \ \rightarrow \ t_{stop} = -\frac{v_0}{a}$$

For a certain initial speed, v_0, and a uniform deceleration, a ($a<0$), the vehicle needs a time t_{stop} to stop the vehicle. And consequently, the total braking distance is:

$$x(t_{stop}) = x_0 - \frac{1}{2} \cdot \frac{{v_0}^2}{a}$$

Therefore, if both vehicles start reducing their speed at the same time the gap variation, Δ_{gap}, due between the rear bumper of the leader and the front bumper of the follower after the complete braking ($a<0$) can be determined as:

$$\Delta gap \quad = \overbrace{\left[x_{0_{Leader}} - \frac{1}{2} \cdot \frac{{v_{0_{Leader}}}^2}{a_{Leader}} - length_{Leader}\right]}^{\substack{\text{End position of the leader} \\ \text{(rear bumper)}}} - \overbrace{\left[x_{0_{Follower}} - \frac{1}{2} \cdot \frac{{v_{0_{Follower}}}^2}{a_{Follower}}\right]}^{\substack{\text{End position of the follower} \\ \text{(front bumper)}}} - \overbrace{\left[(x_{0_{Leade}} - length_{Leader}) - x_{0_{Follower}}\right]}^{\text{Initial gap}}$$

$$\Delta gap_{Braking} = \frac{1}{2} \cdot \left[\frac{{v_{0_{Follower}}}^2}{a_{Follower}} - \frac{{v_{0_{Leader}}}^2}{a_{Leader}} \right]$$

Reaction time, RT, is the time interval between the moment when the event is observed and when the driver gets to apply the brakes. Reaction times vary from person to person and can be affected by many external factors like tiredness or distractions. Ranging from 0.7s to 3s.

If the follower is driving at a constant speed gap the reduction due to this reaction time is basically the distance the vehicle travels before putting on the brakes

$$\Delta \text{gap}_{\text{Reaction Time}} = -\, \text{v}_{0_{\text{Follower}}} \cdot \text{RT}$$

The addition of these two values gives the total distance reduced ($\Delta_{gap}<0$) or increased ($\Delta_{gap}>0$) between a leader and its follower during the whole braking process.

$$\boxed{\Delta gap_{Total} = \frac{1}{2} \cdot \left[\frac{{v_{0}}_{Follower}^{\;2}}{a_{Follower}} - \frac{{v_{0}}_{Leader}^{\;2}}{a_{Leader}} \right] - {v_{0}}_{Follower} \cdot RT}$$

Under the same speed and deceleration conditions of a leader and its follower, during an emergency brake the loss of gap depends only on the linear term in the previous equation. See in the following figure how the gap reduction varies among different speeds and reaction times.

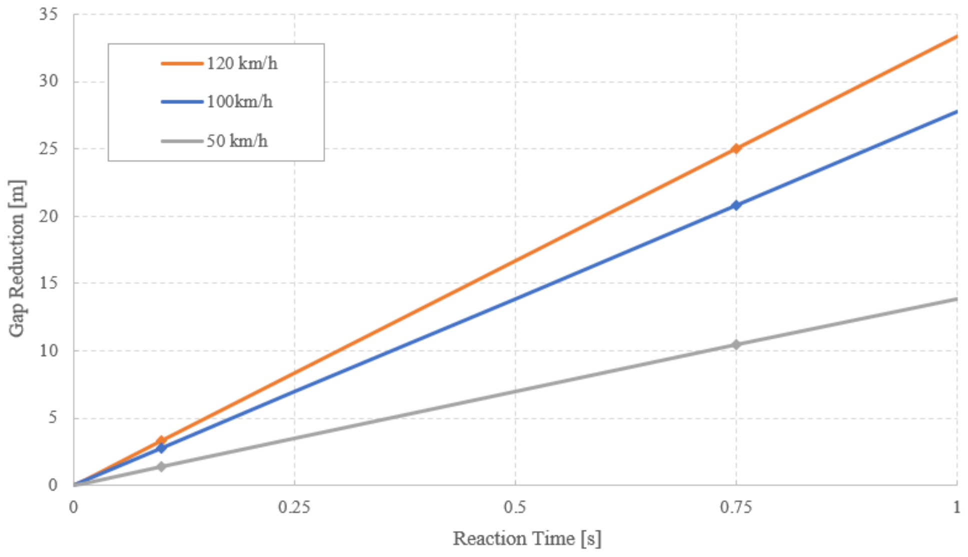


Figure 5. Gap reduction for different reaction times and speeds. Own Source.

The greater the reaction time of the follower is and the greater the speed difference is, the higher probability of collision. Therefore, any driving assistance that decreases the amount of time taken to react, reduces the risk of collision significantly. The communication latency for V2V communication has defined to be 0.1s which makes a big difference in terms of gap reduction.

The platoon followers will travel together as a unit at the same speed. They move coordinately as one. The desired distance, DD, between the vehicles will be limited to a headway of 0.1s and to a minimum spacing of 0.5m. The desired distance between the followers will therefore vary according to the speed, v.

$$DD = \max(0.5\,, \frac{v\,[\frac{km}{h}]}{3.6} \cdot 0.1 \cdot \gamma_{Latency})$$

γ_{Latency} is a safety factor that will be taken as 1.1 to cover any perturbation in the communication. Note that for speeds under 16,36 km/h (speed at which the second terms equals to 0.5m) the desired distance will be controlled by the spacing limitation of 0.5m and for greater speeds the headway will control the spacing.

To prevent chain collisions in the platoon, where an initial collision triggers a series of collisions involving the following vehicles, an additional security distance will be set between the platoon leader and the first following vehicle. The safety factor γ_{Latency} intends to cover any perturbation in the communication and the safety factor γ_{Braking} aims to consider other factors that can increase the braking distance. i.e. bad condition of the car's brakes, a poorly maintained road surface, poor weather conditions, etc.

$$security\ distance = -\Delta gap_{Total} = -\frac{1}{2} \cdot \left[\frac{{v_0}_{Follower}{}^2}{a_{Follower}} - \frac{{v_0}_{Leader}{}^2}{a_{Leader}}\right] \cdot \gamma_{Braking} + {v_0}_{Follower} \cdot RT \cdot \gamma_{Latency}$$

$$\begin{cases} \mathrm{v_0}_{\mathrm{Leader}}\,,\ \mathrm{v_0}_{\mathrm{Follower}}\ \text{in [m/s]} \\ \gamma_{Braking} = 1.1 \\ \gamma_{Latency} = 1.1 \end{cases}$$

In general, the speed of the leader and the follower will be very similar, because the formation process with the CACC approaches the leader with minimal differences in speed. In the worst-case scenario they would travel at 100km/h. Considering a latency of 0.1s and a maximum deceleration of -6m/s^2 the security distance is:

$$Security\ distance = \frac{1}{2}\cdot\left[\frac{[100m/s]^2}{-6m/s^2} - \cancelto{\sim 0}{\frac{[\sim 100m/s]^2}{-6m/s^2}}\right]\cdot\ 1.1\ +\ \frac{100}{3.6}m/s\cdot 0.1s\ \cdot\ 1.1 \cong\ \boldsymbol{3m}$$

In the following screenshot it can be observed how the first platoon follower keeps a greater distance to the leader than the rest of the platoon members, which travel at tighter distance

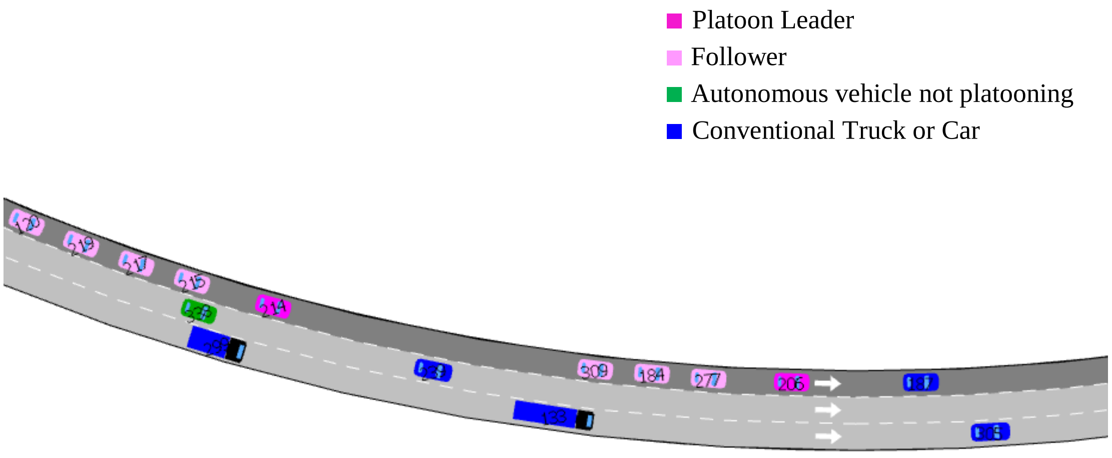


Figure 6. Screenshot of platoons during Aimsun simulation. Own Source.

4. Simulation Environment Definition

Intelligent transport systems (ITS) technologies require a first planning stage, where it is important to ensure that the studied new technology is feasible before any further development or field test is undertaken. The large investments usually required by transportation projects must be justified in a solid way. Therefore, the assessment of a technology like platooning requires from simulation tools that allow to test and visualize the expected impacts of complex models in a simple way. Simulation has been proven to be a powerful tool to analyze and draw reliable conclusions concerning the employment of new traffic management strategies or technologies that would require the analysis and treatment of sophisticated numerical models. Thanks to simulation, details can be appreciated that might be lost in analytical research.

Aimsun Next is a traffic modelling software with an integrated simulator, which allows to simulate at a microscopic, mesoscopic or macroscopic level of detail from a single intersection to an entire region. This study focuses on the movement of each individual vehicle, but also on the interaction with other vehicles, requiring a high level of detail. Therefore, a microscopic approach will be the most suitable analysis method. A basic traffic microsimulation model in Aimsun Next requires three main components, before the network can be considered ready to simulate:

I. The Road Network Model
II. Traffic Demand
III. Calibration of Aimsun Next parameters

In this project an additional component will be needed to allow the autonomous vehicles being simulated according to the platooning methodology. It is an externally programmed algorithm that will be read during the simulation and will update some of the vehicles according to it.

IV. Aimsun Next microscopic API

Those four steps are explained in detail in the following sections.

4.1. The Road Network Model

The first step towards building a traffic microsimulation model is to define the geometrical layout of the road network in Aimsun Next. The shape of the road sections and turnings can be modelled in a straightforward way by means of the software's user interface. To perform the platooning analysis a three-lane highway ring road of approximately 1.5km length will be used. The model includes an on-ramp and an off-ramp lane to allow vehicles merge and exit from the main highway traffic. The geometrical representation of the model can be seen in Figure 7.

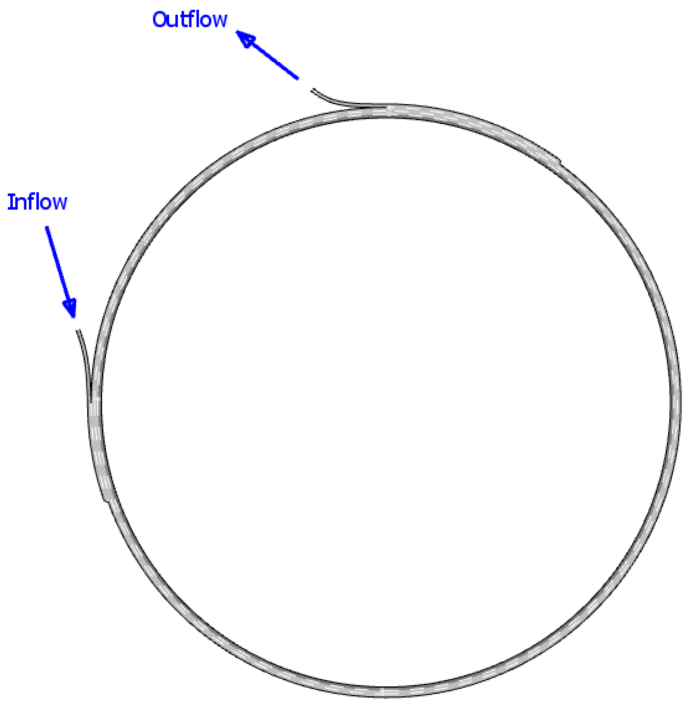


Figure 7. Network Layout. Own Source.

One of the great advantages of the proposed geometry is the possibility to assess the evolution of traffic for long periods of time, until reaching stationarity, in a finite length of the infrastructure. With a single on- and off-ramp it is possible to simulate the whole range of densities as well as any average trip distance. Another big advantage of this straightforward approach/model is that it enables to speed up the simulation. Large models usually need high times of simulation.

Many towns, cities or even countries are encircled by a ring road-shaped infrastructure. This kind of highway configurations aim to reduce traffic volumes in the streets of the urban center by appealing drivers to use the outskirt parts of the network. This kind of solution removes traffic from the center and offers an alternate route around the city for drivers traveling from one side to the other without being forced to access the congested and slow city roads.

Barcelona is a great example. See Figure 8. Its ring road consists of two parts, the so-called Ronda de Dalt or B-20 motorway (the upper ring road), near the mountain area, and the Ronda Litoral or B-10 motorway (the seaside ring road), near the coastal zone. Exits on the rondes are numbered from 1 to 30 and the ring road length sums up to a total of 46km, 26km and 20km respectively. The average distance between two consecutive exists is therefore 1.53km. (46km/30exits).

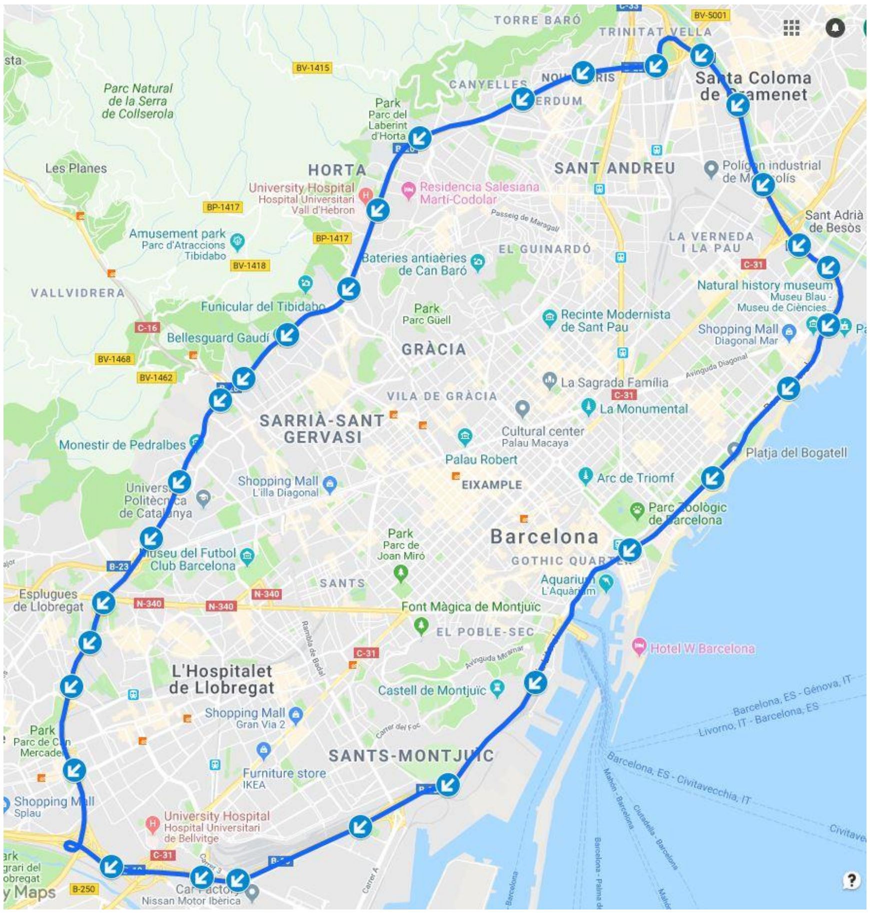


Figure 8. Barcelona’s ring-road and its exits. Source: Edited from Google Maps.

Note that this average distance between two consecutive exits in Barcelona's ring road matches up with the length of the network that will be used for simulation, 1.5km, meaning that a vehicle that completes a loop in the model could represent a vehicle that has traveled one exit in the highway.

The width of the lanes has little or no effect on the model outputs, but for consistency with European roads, where the width of the lanes varies by country between 2.5 to 3.25m, a width of 3m has been set to all the existing lanes in the model.

The speed limit will be 100km/h.

The following table contains detailed information of length and radius for each of the lanes in the model. Also, a description is included in Table 2. This description will be used from now on to refer to any of the three lanes in the highway.

Lane	**Description**	**Radius [m]**	**Length [km]**
Lane 1	Shoulder Lane	256	1,608
Lane 2	Middle Lane	253	1,590
Lane 3	Platooning Lane	250	1,571
Lane entry	On-ramp Lane	259	0,1
Lane exit	Off-ramp Lane	259	0,1

Table 2. Lanes description.

In general, heavy vehicles are not supposed to use the left lane. Some countries have even laws prohibiting trucks from the left and its use is only allowed in certain circumstances. All the highway lanes will be shared between autonomous and non-autonomous vehicles, except for trucks, to which the use of the platooning lane will be restricted. Trucks will therefore use the middle and the shoulder lanes.

4.2. Traffic Demand

The Traffic Demand sets the number of vehicles that will arrive to the network for a certain period of time and how they will travel to their destination. Aimsun Next offers two principle means of describing a Traffic Demand, either by using *Origin-Destination Matrices (OD-Matrices),* where basically the amount of vehicles between origins and destinations are specified and then the simulator controls the path or by using *Traffic States,* where at each entrance point to the network the input flows are specified and also the turn percentages from every section to all the sections accessible from it need to be defined.

For the study we are interested in, the vehicle should stay for some loops in the model and randomly leave the highway after a certain travelled distance. An OD-Matrix would force the vehicle to exit immediately after reaching its exit, whereas a traffic state would allow the vehicle to stay in the highway with some probability. Defining the traffic demand using a traffic state means specifying basically two variables in our model. First, the input flow, which is the number of vehicles that will arrive per hour to the on-ramp lane. Second, the turn percentages of vehicles that will exit the highway and of vehicles that will stay in the highway (α and 100 – α in %). See Figure 9.

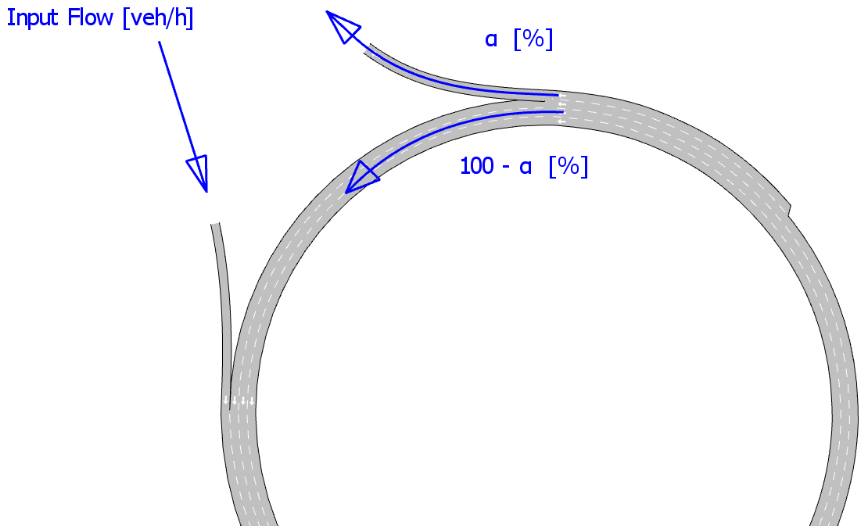


Figure 9. Input flow and turning percentages. Own source.

The input flow is a variable that will be changed among scenarios and will generate different demand situations. The higher the input flows are set, the higher the circulating flows will be in the highway roads.

The value of α represents the split rate at the off-ramp zone, at a microscopic level of detail, where single vehicles are considered, this can be seen as the likelihood that the vehicle stays or leaves the highway in the next exit. On a more general level, the value is also a measure of the expected average trip length of all the vehicles in the network. Once the average trip length is known, the split rate can be derived with the following relation:

$$\alpha = \frac{Length\ of\ the\ ring}{Average\ trip\ length}$$

Note that if for example α = 100% the average trip length will be 1.5km, equal to the ring length, as all the vehicles will leave the loop the first time they pass by the exit. For consistency with the criteria used to define the length of the loop, an average trip length for Barcelona's ring road will be considered to determine the split ratio. The average trip length should take a smaller value than the half of the length of the complete ring road, this is 23km. Based on the Workday Mobility Survey (EMEF) of 2017 [8] carried out every year on a sample of residents of the metropolitan area of Barcelona, 15km is a reasonable average trip length in an urban environment like Barcelona´s ring road. Using the previous relation, the split ratio would take a value of:

$$\boldsymbol{\alpha} = \frac{Length\ of\ the\ ring}{Average\ trip\ length} = \frac{1.597\ km}{15\ km} \cong \mathbf{10}\%$$

This split rate will be fix for all the scenarios, a percentage of 10% must be small enough to allow vehicles exit the highway without blocking the exit area. The off-ramp should not work as an active bottleneck. Therefore, the off-ramp side lanes will be long enough to ensure that the leaving traffic does not affect the traffic on the main stream.

Another important factor to consider is the time interval between two consecutive vehicles (the headway) at the entry points of the network. Whereas the Traffic Demand sets the average number of vehicles which will arrive in a time period, the arrivals algorithm defines how those arrivals are distributed in time. The headway model can be exponential, uniform, normal, constant, ASAP (as soon as possible) or externally defined with a custom distribution.

The default arrival model in Aimsun Next is the exponential distribution. In transportation it is one of the most common distributions used to model arrivals. Time intervals between two consecutive vehicle arrivals (headway) at input sections are sampled from an exponential distribution (Cowan 1975).

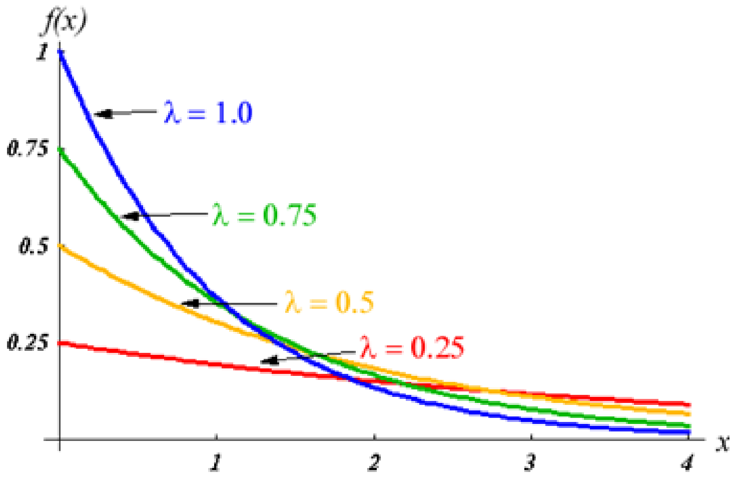


Figure 10. Exponential distribution (Cowan 1975). Source: Aimsun Next Manual.

This distribution assumes that each vehicle arrives at a random time without any dependence on the time the previous vehicle arrived. The mean input flow in vehicles/second can be estimated from the input flow in the Traffic Demand. The algorithm for calculating the time headway, t, in the exponential distribution is the following:

$$t = -\frac{1}{\lambda} \cdot \ln u$$

$$\begin{cases} u = random(0,1) \\ \lambda = mean\ input\ flow\ in\ veh/s \end{cases}$$

4.3. Aimsun Next microscopic simulator

In a microscopic simulation the whole duration of the simulation is split into small time intervals called simulation steps (Δt). At each simulation step the speed, acceleration and position of all the vehicles in the network are calculated and updated. The minimum time step available in Aimsun Next is 0.1s. This value matches with the communication latency of V2V communication, therefore a simulation step of 0.1s will be used for this project, meaning that the simulator will calculate the new state of all the vehicles in the network every 0.1s.

Aimsun Next allows to define different reaction times by vehicle type or even different reaction times for a specific vehicle type according to a discrete probability function. Where the sum of all the probabilities adds up to 1. For human-driven trucks and cars a reaction time of 0.8s is set. For autonomous vehicles that use wireless communication a reaction time of 0.1s will be set.

The logic of the simulation process in Aimsun Next is governed by different models. In our case mainly three different models will be applied. The car following, the lane changing and the on-ramp models [1]. They are explained hereunder:

- *Car-following model*

This model will be applied if the vehicle does not need to change lanes. The microscopic car following model implemented in Aimsun Next is based on the Gipps model (Gipps 1981 and 1986b) [9]. In a simplified form it basically consists on the evaluation of two components, V_a and V_d, the maximum speed in acceleration or deceleration evaluation. V (n, t+T), the speed of vehicle n, in the next simulation step, t+T, will be the minimum between those two components.

$$V\ (n, t+T) = \min(\ V_a\ , V_d)$$

$$\begin{cases} V_a\ =\ speed\ of\ vehicle\ n\ for\ its\ next\ simulation\ step\ when\ no\ preceding\ vehicle\ limits\ the\ speed \\ V_d\ =\ speed\ of\ vehicle\ n\ for\ its\ next\ simulation\ step\ when\ the\ preceding\ vehicle\ limits\ the\ speed \end{cases}$$

The first component, **V_a**, represents the intention of the vehicle to achieve its desired speed. In uncongested traffic conditions the vehicle will accelerate to reach this speed. The maximum speed to which the vehicle can accelerate during a time period (t, t+T) is given by the following formula[1]:

$$V_a(n,t+T) = V(n,t) + 2.5 \cdot a(n) \cdot T \cdot \left(1 - \frac{V(n,t)}{V^*(n)}\right) \cdot \sqrt{0.025 + \frac{V(n,t)}{V^*(n)}}$$

$$\left\{\begin{array}{ll} V_a(n,t+T) & maximum\ speed\ to\ which\ vehicle\ n\ can\ accelerate\ during\ time\ period\ (t,t+T) \\ V(n,t) & speed\ of\ vehicle\ n\ at\ time\ t \\ a(n) & maximum\ acceleration\ of\ vehicle\ n \\ V^*(n) & desired\ speed\ of\ vehicle\ n \\ T & time\ step \end{array}\right.$$

The second component, **V_d**, reproduces the limitations imposed by the preceding vehicle when trying to achieve the desired speed. The maximum speed a vehicle can reach during a time interval according to its own characteristics and the restrictions imposed by the presence of the lead vehicle is limited by:

$$V_d(n,t+T) = d(n) \cdot T + \sqrt{d(n)^2 \cdot T^2 - d(n)\left[2 \cdot \underbrace{\{x(n-1,t) - s(n-1) - x(n,t)\}}_{\text{gap}} - V(n,t) \cdot T - \frac{V(n-1,t)^2}{d'(n-1)}\right]}$$

$$\left\{\begin{array}{ll} d(n) & normal\ deceleration\ for\ vehicle\ n\ "d(n) < 0" \\ d'(n,t) & estimated\ normal\ deceleration\ for\ vehicle\ n\ "d(n) < 0" \\ x(n-1,t) & position\ of\ vehicle\ n-1\ at\ time\ t \\ s(n-1) & effective\ length\ of\ vehicle\ n-1 \\ x(n,t) & position\ of\ vehicle\ n\ at\ time\ t \end{array}\right.$$

The estimation of the leader's deceleration, d'(n-1), is a function α, the sensitivity factor:

$$d'(n-1) = d(n-1) \cdot \alpha$$

If α < 1, the vehicle underestimates the deceleration of the leader and consequently the vehicle becomes more aggressive, while if α > 1 the vehicle overestimates the deceleration of the leader and therefore becomes more prudent by increasing the gap. In this work α will be set to 1.

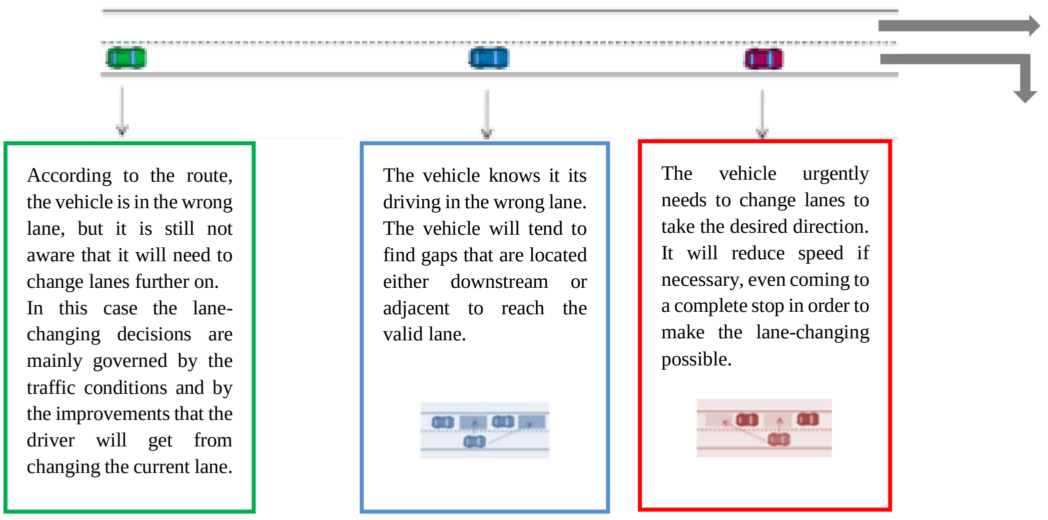
[1] *Note that if the vehicle is driving at its desired speed the acceleration term is 0, because* $V(n,t) = V^*(n,t)$

- *Lane-changing model*

This model will be applied if the vehicle desires to change lanes. The lane-changing model implemented in Aimsun is also a development of the Gipps lane-changing model (Gipps 1986a and 1986b). The vehicle takes the decision to change lanes according to three factors:

- The need of the lane change. This is mainly determined by the route, i.e., the accessibility options from the current lane to follow the desired path.
- The desirability of the lane change. This depends on whether there will be any improvement in the traffic conditions for the driver as a result of the change. I.e. when leader vehicle is slower than the desired speed.
- The feasibility of the lane. This requires that there is an adequate gap to make the lane change. It calculates both the braking imposed by the future downstream vehicle to the lane-changing vehicle and the braking imposed by the lane-changing vehicle to the future upstream vehicle. If both braking levels are acceptable, then lane changing is possible.

To understand the driver's behavior in the lane-changing decision process, see in Figure 11 a schematic representation of the three possible lane-changing situations of a vehicle that wants to go straight:



According to the route, the vehicle is in the wrong lane, but it is still not aware that it will need to change lanes further on. In this case the lane-changing decisions are mainly governed by the traffic conditions and by the improvements that the driver will get from changing the current lane.

The vehicle knows it its driving in the wrong lane. The vehicle will tend to find gaps that are located either downstream or adjacent to reach the valid lane.

The vehicle urgently needs to change lanes to take the desired direction. It will reduce speed if necessary, even coming to a complete stop in order to make the lane-changing possible.

Figure 11. Lane-changing example. Own Source. Adapted from Aimsun Next Manual.

- *On-Ramp model*

This is the model that will be applied when the vehicle is merging into traffic from an on-ramp lane.

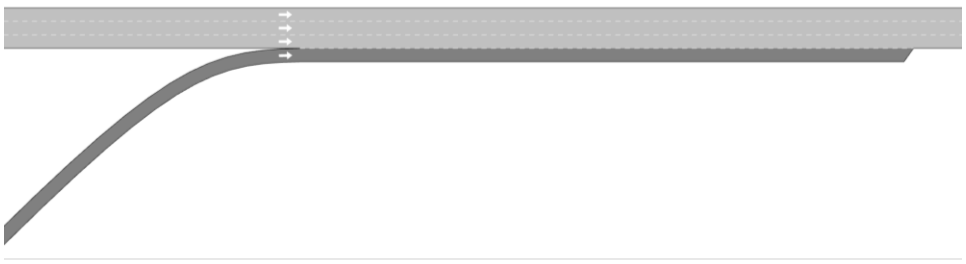


Figure 12. On-ramp lane. Own Source.

The lane changing model applied at on-ramps is the same cooperative model as for normal lane changes with three additional controls:

1.) First vehicle on is first vehicle off (FIFO): In our model this option will be toggled on, meaning that only the first vehicle on the ramp can change lane to move off it, if toggled off, all vehicles on the ramp may try to merge.

2.) Merging Distance:
The merging distance controls where the vehicle can start to merge onto the main carriageway. By default, it is the maximum value between five times the length of the vehicle, L, or five times the distance travelled in one reaction time, RT at the current speed of the vehicle, V.

$$Merging_{\ Distance} = \max\ [\,5 \cdot L\,, 5 \cdot V \cdot RT]$$

3.) Cooperation Distance:
The distance from the end of the ramp where a vehicle may start to receive co-operation from vehicles on the main carriageway to make its lane change. This distance is set to be at the start of the section.

4.4. Aimsun Next microscopic API

The Aimsun Next microscopic Application Programming Interface (API) is a set of functions, in Python or C++, that allow programmers to read and modify information concerning vehicles during an Aimsun Next microscopic simulation. Therefore, it can be used to model some connected and autonomous vehicle applications like platooning.

When the API is active in the simulation, Aimsun Next forces an update of the vehicles using the programmed algorithm at each simulation step. The present algorithm will be programmed in order to modify only vehicles that are subject to use the platooning technology. Therefore, autonomous vehicles will be updated according to the platooning API and a regular cars and trucks will be updated according to Aimsun Next simulator.

As mentioned before, the platooning API describes the sequence of specified actions that will be followed at each simulation step. Therefore, every 0.1s the algorithm is run from top to bottom. The executed actions aim to control the autonomous vehicles inside the network and make them behave according to the (in Section 3) defined platooning methodology. Note that the remaining vehicles, conventional trucks and cars, which do not support platooning technologies, will be simultaneously controlled by Aimsun's microscopic simulator and behave according to its predefined car following and lane changing models seen in the previous section.

4.4.1. Platooning API

The logic of the platooning API algorithm that is followed during each simulation step will be explained in a simplified way in this section. The complete code can be found in the Appendix.

The first step of the structure is to read the information of every single vehicle inside the network. The collected information includes the vehicle identifier, its type (VA, not VA or truck), its position in the network, the lane and the zone. The zone is an indicator of weather the vehicles desires to change lane or not. Three different zones are possible according to the status of the vehicle, these are explained further below.

Knowing the position allows to establish an order among the vehicles. The idea is to order them by position (see Figure 13) and iterate from the first to the last regardless from the lane.[2] The first thing that is checked is whether the vehicle is in the platooning lane or not.

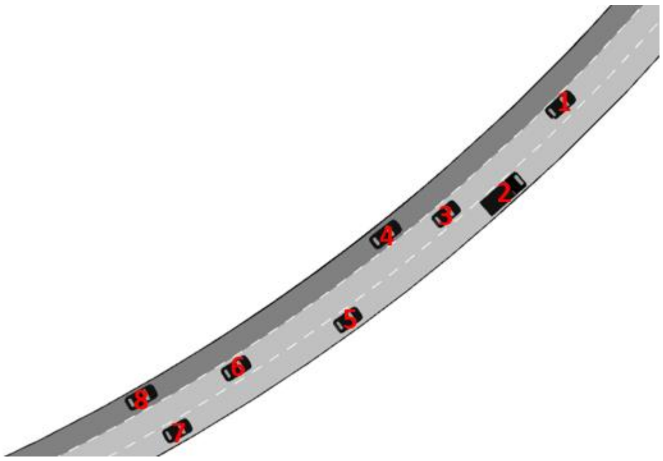


Figure 13. Order among vehicles for iteration. Own Source.

If, on the one hand, the vehicle is not in the platooning lane, and it is a conventional car or truck, the API will not update the vehicle and Aimsun Next simulator will move the vehicle according its default behavioral models. However, if the vehicle is of type self-driving and it is not leaving imminently, zone 1, then the algorithm will force the vehicle to apply a lane-changing to the left and try to reach the platooning lane. The lane-changing will only happen once there is a safe gap in the target lane. On the contrary, if the space is not big enough to enable a safe lane-changing without disturbing traffic, the vehicle will stay in the current lane.

It has been seen that zone 1 refers to vehicles with no intention to leave the highway at the moment. Zone 2 and 3 concerns vehicles leaving at the following exit. The zone will be increased from 2 to 3 as the vehicles encloses to the exit. In zone 2 there is no rush, and the lane-changing will be smooth. Once the vehicle is close to the exit and has not been able to change lanes without disturbing traffic, it will start to slow down and wait to receive cooperation from other vehicles to access the shoulder

[2] *Note that vehicle 5 in Figure 13 is not the follower of vehicle 4, the follower is vehicle 8. Leader and follower are always in the same lane. It only denotes the way in which vehicles are updated in the API.*

lane. If the vehicle is already travelling in the shoulder lane it will keep the lane until the off-ramp lane is reached and then exit the network. The car following will be as it is for the regular vehicles and trucks. This is summarized as a flowchart in Figure 14.

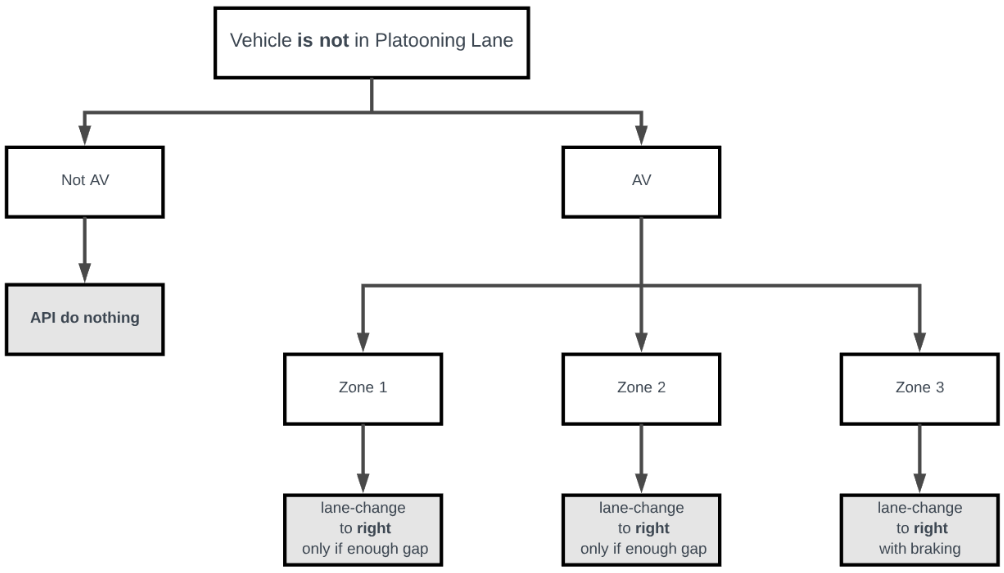

Vehicle **is not** in Platooning Lane
Not AV
AV
API do nothing
Zone 1
Zone 2
Zone 3
lane-change to **right** only if enough gap
lane-change to **right** only if enough gap
lane-change to **right** with braking

Figure 14. API decision tree for vehicles driving in the middle or shoulder lanes. Own Source.

If, on the other hand, the vehicle is driving on the platooning lane it will check the available information of the preceding vehicle. An autonomous vehicle can adopt three different statues depending on the distance and the role of the existing leader.

Plaooon Leader status**:** If the preceding is not autonomous then the vehicle automatically switches its status to platoon leader. Also, regardless of whether the preceding vehicle is autonomous or not, if it is at a greater distance than 25m, it is considered to be outside the communication range and the vehicle will adopt the leader role.

Follower status: If the preceding vehicle is within the communication range with the preceding vehicle it will approach it according to the CACC formula seen before. The vehicle will apply this algorithm until it reaches the desired distance to its local leader, DD, with a certain deviation acceptance. It has been seen, that the desired distance varies with speed, the higher the travel speed the greater the desired distance will be. Also, the first follower targets a greater gap than the rest of the platoon members.

Joined status: Once the vehicle has reached the desired distance to its leader, it will switch to the joined status and calculate the speed according to it. The algorithm applies the same speed to all joined vehicles. The speed of each joined vehicle is subject to minor adjustments to adapt the desired distance to its local leader when there is a change in speed. If a vehicle was already joined in the previous simulation step, it will keep the type as long as the distance is within a defined tolerance and the preceding vehicle remains unchanged

The following flowchart aims to summarize the three different statuses an autonomous vehicle can adopt when driving in the platooning lane according to its position and the position of its leader in the network.

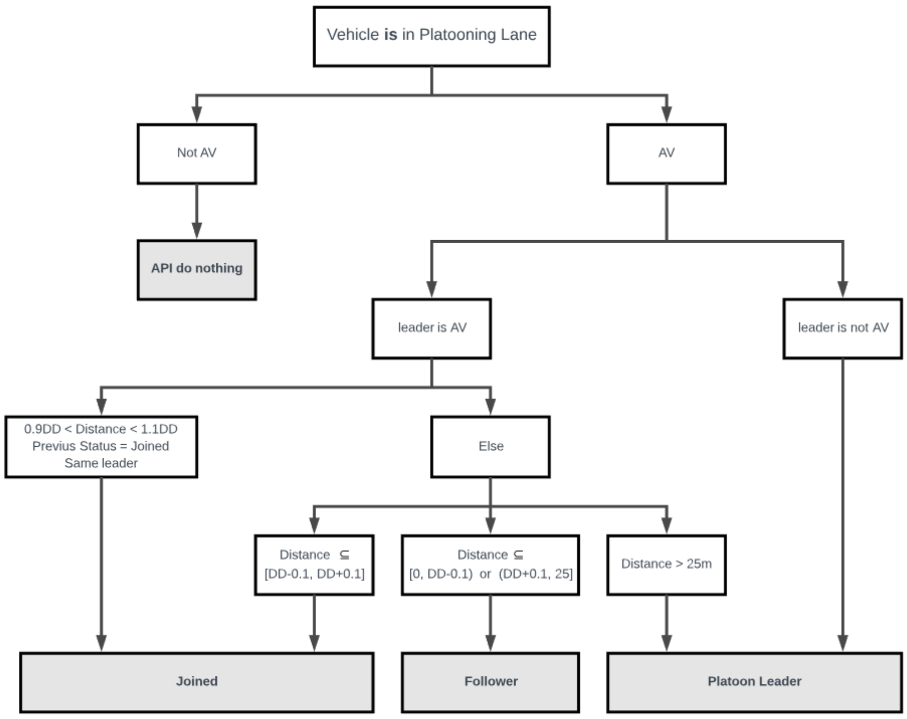


Figure 15. API decision tree for vehicles driving in the platooning lane. Own Source.

After the associated speed and acceleration are calculated according to the status it is checked if the vehicle needs to leave the highway. If the vehicle is in zone 1, meaning there is no information saying that the vehicle needs to leave, the previous calculated speed will be applied. Instead, if the vehicle wants to leave, it will be depending on the urgency. If the vehicle is in Zone 2, but if the vehicle is in zone 3, it will immediately try to reach the valid lane, looking for gaps upstream and reducing speed if necessary, even coming to a complete stop in order to make the lane change possible.

5. Simulation Scenarios and Results

5.1. Summary – Assumptions and previous considerations

To perform the study of platooning technology the following assumptions have been made regarding some of the previously mentioned aspects.

- Infrastructure
  - The road infrastructure is a highway of three lanes equipped with all necessary technologies
  - The geometrical layout is a ring road of 1.5km length
  - The road speed limit is 100km/h
  - Vehicles approach the highway from a slip road on the left
  - Vehicles leave the highway by means of an off-ramp lane on the left
  - Trucks are not allowed to use the platooning lane
  - Vehicles merge into the main road applying a first in first off criteria

- Vehicles
  - For simplicity all autonomous vehicles have the same properties during the platooning process. Length, weight, etc. The length of all the autonomous will be 4m. In reality cars will be from different companies with different parameters, allowing them to brake better/faster or not.
  - Connected and fully automated vehicles have a reaction time of 0.1s
  - Regular cars and trucks have a reaction time of 0.8s
  - Regular cars and trucks have a reaction time of 0.8s

- Platoons of two or more vehicles
  - Platoons free flow speed is lower or equal to the road speed limit
  - Platoons use the leftmost lane which will be referred to as the platooning lane
  - Platoons do not change lanes
  - Platoons do not have a maximum length
  - The V2V communication has a latency of 0.1s

- Traffic Demand
  - The infrastructure is shared by three vehicle types (AV, non-AV and trucks)
  - 5% of the total demand are Trucks
  - The remaining 95% of the vehicles will be split between AV and non-AV
  - Different input flows will be considered to cover congested and uncongested conditions → **Inflow = q_{in}(t) [veh/h]**
  - The percentage of vehicles that will exit the network, α will be a 10% of the circulating flow, q. → **Outflow = q_{out}(t) = α · q [veh/h]**

- Expected results
  - In uncongested conditions, after a warm-up period, the circulating flows will reach an equilibrium. This will be when the output flow equals the input flow. After this moment, t_s, a stationary situation will be reached, where:
    - $\boldsymbol{q_{out}(t_s) = q_{in}(t_s) = constant}$
    - $\boldsymbol{q(t_s) = \frac{q_{out}(t_s)}{\alpha} = \frac{q_{in}(t_s)}{\alpha} = constant}$
  - In congested conditions, $q_{in}(t)$ not be constant as the increasing interaction at the entry will limit the vehicles. The network will until reaching gridlock.
  - The maximum flow that can be achieved for the conventional lanes usually takes values around 2500veh/h.
  - The maximum flow that can be sustained by a platooning lane can be determined knowing the minimum headway of two consecutive vehicles. This is the time interval since the preceding vehicle passes on a point until the following crosses the same point. The minimum time gap between platoons has been defined to be 0.1s and the maximum speed is 100km/h . The maximum flow is therefore:

$$q_{max-tlane} = \frac{1}{h_{min,p}} = \frac{1\ veh}{\left(\frac{minimum\ time\ gap}{3600}\right)h + \frac{0.004\ km}{100\ km/h}} = \frac{1\ veh}{\left(\frac{0.1}{3600}\right)h + \frac{0.004\ km}{100\ km/h}} = 14750 veh/h$$

This value is very optimistic, as it assumes that all the vehicles in the platooning lane are autonomous vehicles and that there is an unique platoon.

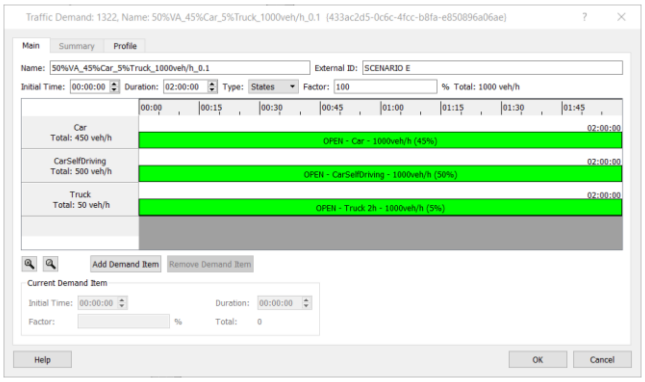
5.2. Scenarios

The primary objective of this study is to use simulation to analyze the performance of the road network when the platooning technology is activated. The analysis framework will be composed of 5 scenarios. In each scenario the penetration rate of autonomous vehicles will vary. Penetration rate is the percentage of vehicles of a type that are driving in the whole system.

- **Scenario A**. 0% Penetration Rate of AV – This is the base Scenario. It is the status quo, where all the vehicles are conventional, it is the existing state of affairs and the starting point for the subsequent comparation exercise.
- **Scenario B.** 5% Penetration Rate of AV
- **Scenario C.** 10% Penetration Rate of AV
- **Scenario D.** 25% Penetration Rate of AV
- **Scenario E.** 50% Penetration Rate of AV

Higher penetration ratios would need the development of a different strategy, e.g. enabling platoons to be formed in the middle lane as well.

The percentage of non-autonomous vehicles is further decomposed in 5% trucks and the remaining percentage of cars, for example, the traffic demand for Scenario E is formed of 50% AV, 45% non-AV and 5% trucks. For each scenario different demand situations will be analyzed. Figure 16, corresponds to Scenario E with an input flow of 1000veh/h.


Traffic Demand: 1322, Name: 50%VA_45%Car_5%Truck_1000veh/h_0.1 {433ac2d5-0c6c-4fcc-b8fa-e850896a06ae}

Name: 50%VA_45%Car_5%Truck_1000veh/h_0.1 External ID: SCENARIO E
Initial Time: 00:00:00 Duration: 02:00:00 Type: States Factor: 100 % Total: 1000 veh/h

Car Total: 450 veh/h — OPEN - Car - 1000veh/h (45%) — 02:00:00
CarSelfDriving Total: 500 veh/h — OPEN - CarSelfDriving - 1000veh/h (50%) — 02:00:00
Truck Total: 50 veh/h — OPEN - Truck 2h - 1000veh/h (5%) — 02:00:00

Current Demand Item
Initial Time: 00:00:00 Duration: 00:00:00
Factor: % Total: 0

Figure 16. Example of a 2h traffic demand definition in Aimsun. Own Source.

5.3. Data Collection

The easiest and most common way of measuring traffic flow variables is by means of locally fixed detectors. This kind of detection can provide information about local speed, flow or density. A total of 16 detector locations are sited along the road separated with an approximate spacing of 100m. Each of the detection locations, shown in Figure 17, consists of one detector covering the whole section and three single lane-detectors. See detail for detector 3. The on-ramp and off-ramp lanes are also equipped with detection.

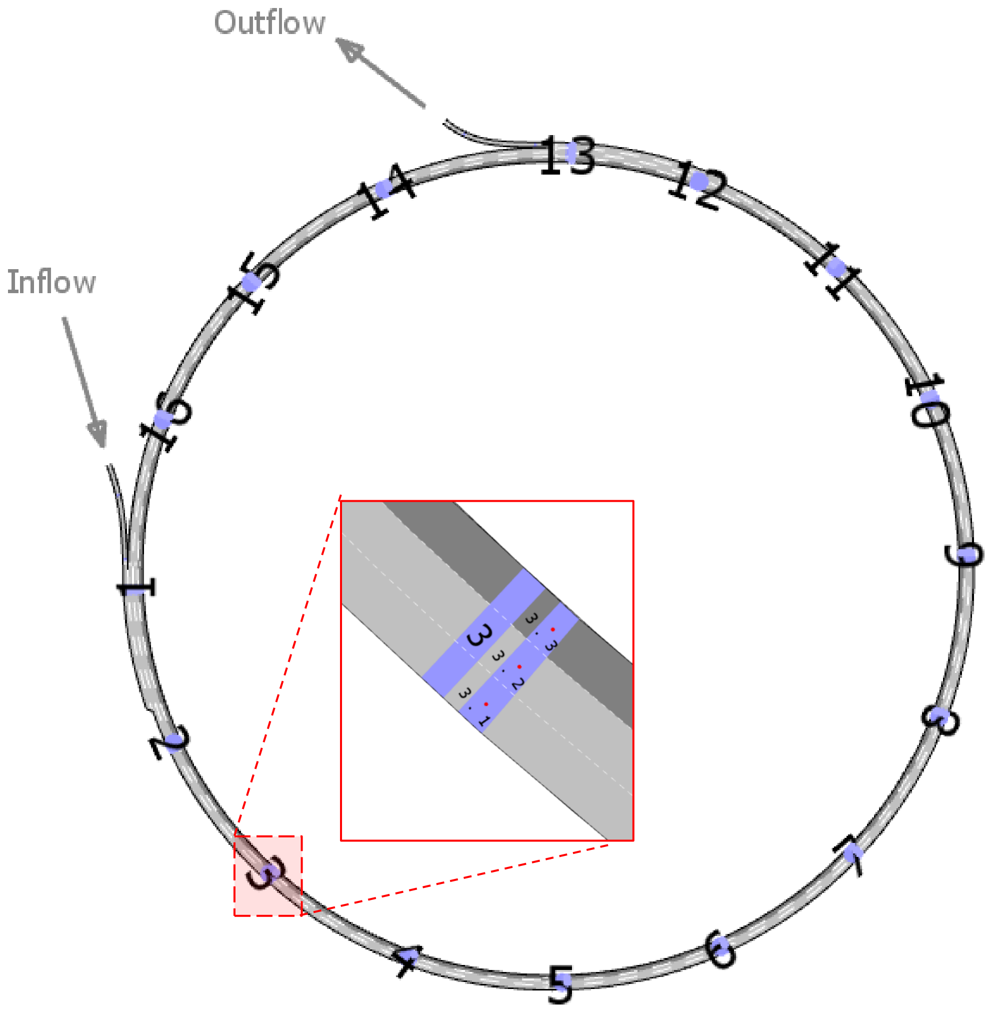


Figure 17. Configuration of detectors in the network to gather simulation data. Own Source.

As already mentioned, many detectors are needed because a single detector does not provide information of the global performance of the network, a vehicle's trajectory can vary greatly among time and space. The work of Edie (1965) introduces generalized definitions of traffic flow, density and speed along a highway that can be extended to the network level. Considering a region (A) in time and space with dimensions T and L, traffic flow variables can easily be deduced if the total travel time, TTT, and the total travelled distance, TTD, are known. As the interest relies on studying the traffic variables in the different lanes and for each type of vehicles, it would be needed to obtain those values by lane and vehicle type.

Aimsun Next provides information about TTT and TTD for the different vehicle types, but it does not provide the information by lane at a network level. There is information of such variables by lane at the section level, but it only considers vehicles that have crossed the whole section during the data collection interval. Vehicles inside the road segment during this interval are therefore not considered. Edie´s definitions of traffic flow variables are the optimal method to obtain traffic variables, however gathering them with Aimsun Next would add too much complication to a problem that can be addressed in a simple way using detectors.

Aimsun Next provides statistics at detector level and traffic variables can be easily collected by lane and vehicle type. It is also possible to create groups of detectors and get aggregated data for them, for example, a group including all the detectors placed over the platooning lane. These measurements will be calculated based on vehicles that have crossed the detectors defined before and shown in Figure 17 during a defined time interval. For congested conditions, data will be gathered every minute and for uncongested conditions every 5 minutes.

For consistency, only detectors 2 to 12 will be used to analyze the traffic flow variables. This is, because detectors located in between of the off-ramp and the on-ramp are expected to have different flows than the rest of the network due to the exiting flows of vehicles.

5.4. Results

This section presents the results obtained after carrying out the simulations with Aimsun Next for the five scenarios. To cover congested and uncongested conditions, the scenarios have been run with different demand levels. Using the results obtained with Aimsun, it is possible to assess the performance of the analysed scenarios. Therefore, it is interesting to see how the network performs in each scenario. For this purpose, the following performance indicators are used:

- Fundamental Diagram (FD)

The fundamental diagram of traffic flow is a diagram that relates flow, density and speed, the three main variables in traffic flow theory. On the x-axis the density, k, in vehicles/km and on the y-axis the flow, q, in vehicles/hour is plotted. One state of the network is defined by a pair of (q, k) and the speed can be deduced from the slope, which is the same as the following relation:

$$v = \frac{q}{k}$$

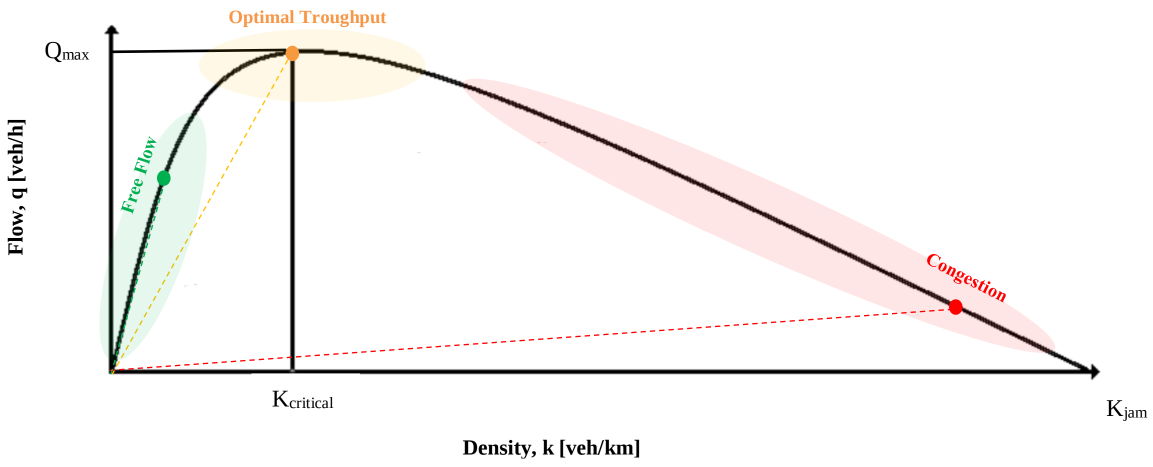


Figure 18. Fundamental Diagram. Own Source.

In Figure 18 three different states can be identified in the fundamental diagram: uncongested or free flow conditions, optimal throughput conditions or congestion. In terms of speed, it is observed that the slope achieves its maximum in free flow conditions and tends to zero for high densities. The maximum density that can be achieved is the jam density, $K_{jam,}$ at this density speed is zero. Vehicles are stopped one after another. Assuming a vehicle occupies a gap of 5-5.5m it would take a value of 180-200veh/km.

The fundamental diagram can therefore be used to evaluate the effects of platooning on the roadway system for all the possible range of densities. For the five scenarios the fundamental diagram for the platooning lane and the fundamental diagram for the two traditional lanes (middle and shoulder lane) is plotted. The comparison between these two diagrams will allow to compare independently the performance of each type of lane. An aggregated diagram for both lanes is also included in each plot.

➢ **Scenario A** - 0% Penetration Rate of AV

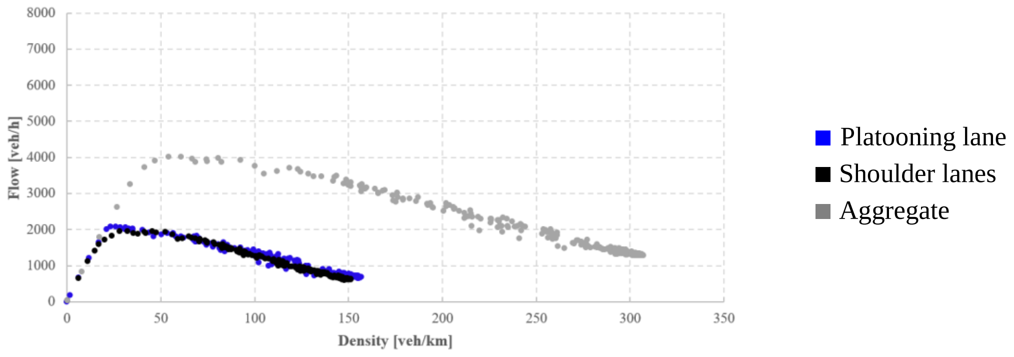


Figure 19. Scenario A – FD for each lane type

➢ **Scenario B -** 5% Penetration Rate of AV

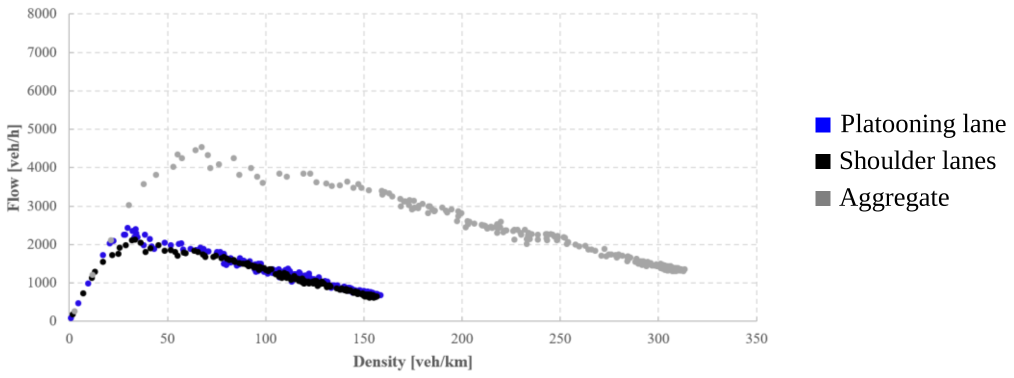


Figure 20. Scenario B – FD for each lane type

➢ **Scenario C -** 10% Penetration Rate of AV

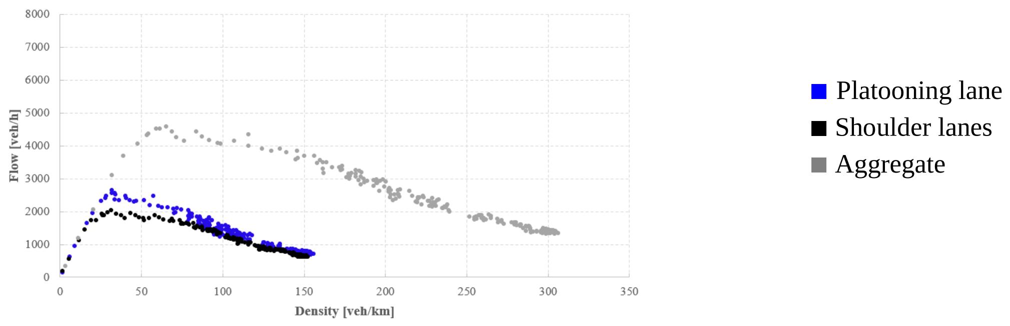


Figure 21. Scenario C – FD for each lane type

➢ **Scenario D -** 25% Penetration Rate of AV

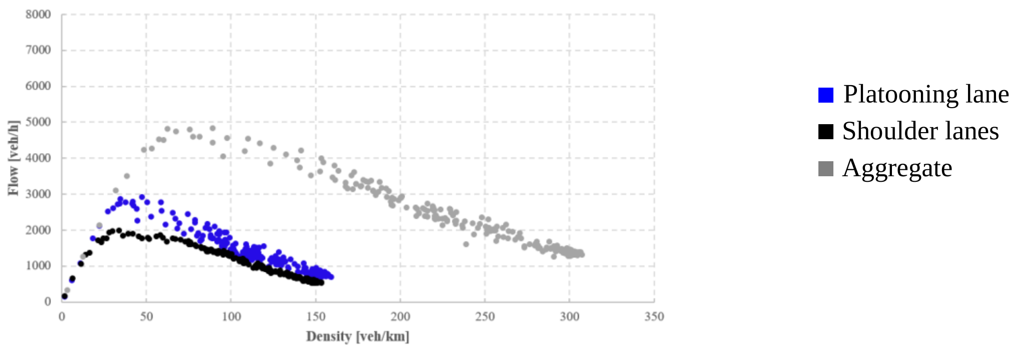


Figure 22. Scenario D - FD for each lane type

➢ **Scenario E -** 50% Penetration Rate of AV

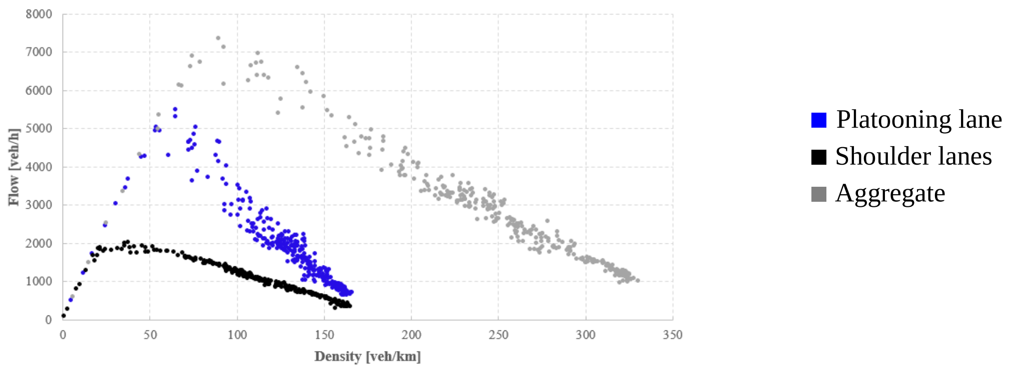


Figure 23. Scenario E - FD for each lane type

In the five scenarios it is shown that the free flow speed, defined by the slope, that can be observed on the uncongested branch of the fundamental diagrams tends to 100km/h for both the platooning and traditional lanes. Traditional lanes have a slightly lower free flow speed than the platooning lane, which can be explained by the presence of slower trucks. However, the value is very close to the speed limit of the network in both cases.

In all the scenarios it can also be appreciated that the maximum density tends to a value between 180-200veh/km. (The double for the aggregated fundamental diagram). Which has been seen that corresponds to realistic jam density values.

For better comparison purposes the results of the different scenarios have been merged into one single plot. In Figure 24 the fundamental diagrams of the conventional lanes can be compared. In general, no obvious differences can be observed among scenarios. Meaning that the platooning technology has little or no effect on the shoulder lanes. It does not enhance or reduce the performance in the traditional lanes.

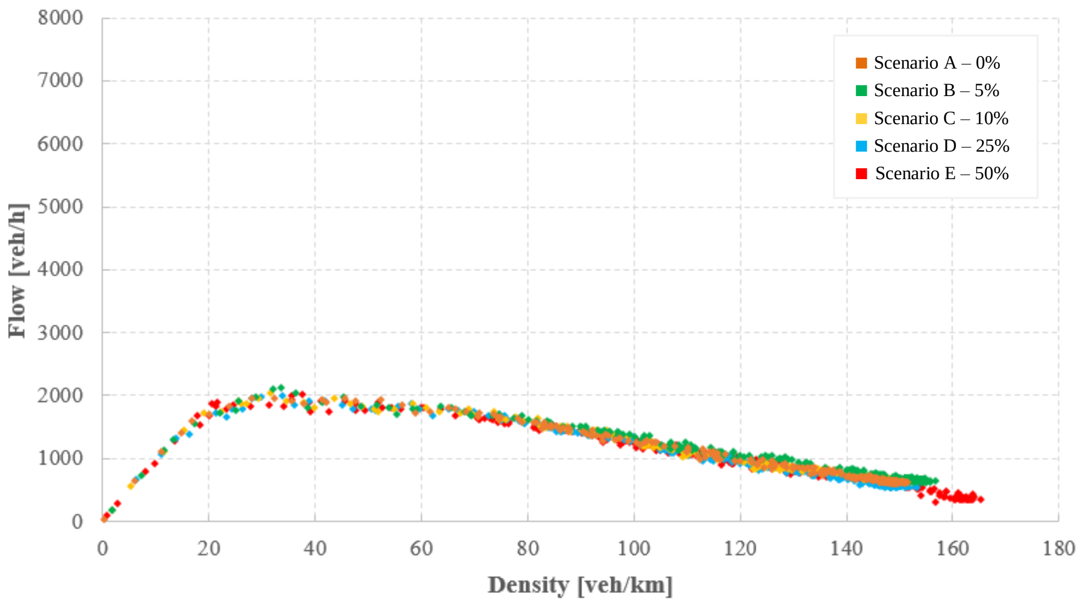


Figure 24. FD for traditional lanes - All scenarios

However, if we plot the fundamental diagrams corresponding to the platooning lane of all the scenarios in a single figure, the same cannot be said. In Figure 25 it can be observed how for the same density, the flows in the platooning lane are higher by increasing the penetration rate of autonomous vehicles. The greatest difference is seen for scenario E, where the maximum observed flow is approximately two or even three times higher than the maximum flow of the base scenario, scenario A.

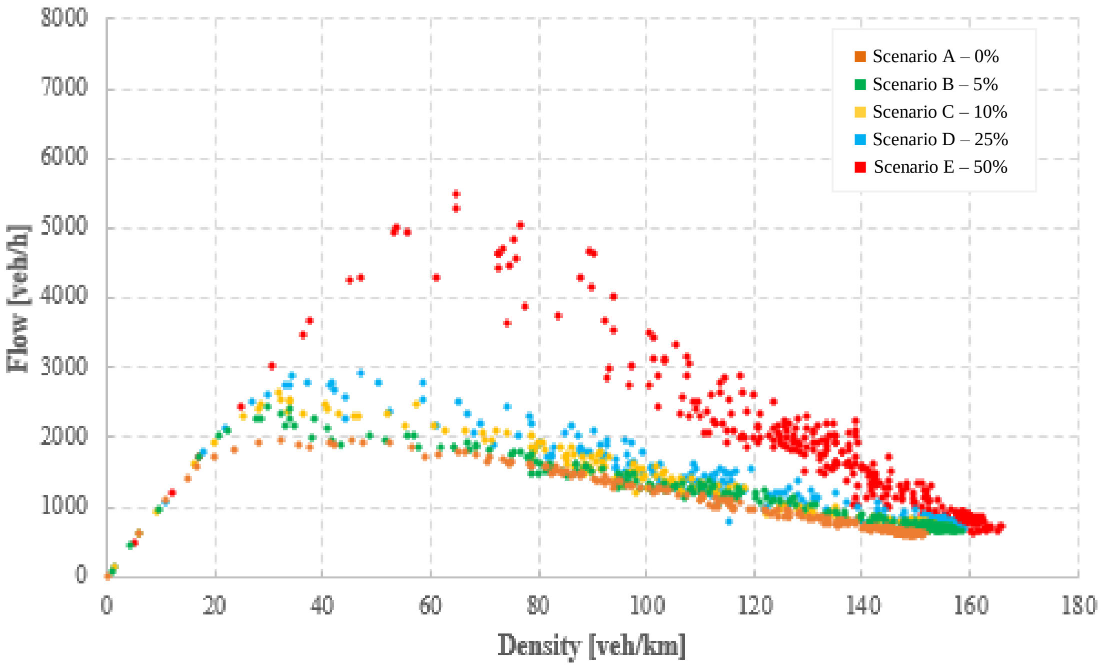


Figure 25. FD for platooning lanes - All scenarios

The higher scatter observed in the congested branch of the fundamental diagram for scenarios with higher penetration rates can be attributed to the different available spatial and temporal distributions of congestion. Meaning that for a same density different platoon configuration can exist providing different flows and the same the other way around. Whereas scenarios with lower proportion of autonomous vehicles in the network will difficulty form many different platoon configurations, therefore more stable shape define the possible states in the lane.

Another important subject to check, is whether the platooning lane and the shoulder lanes are experiencing big differences in speed in congested conditions during the same time period. If there is no big variance along both lane types, then the complete flow operates at one state, and this flow is called a one pipe regime. Otherwise, the lanes would operate independently of each other, defined as two pipes regime. From Figure 26 it can be deduced that the lanes operate at a one state, as in congested conditions the speed differences range between 0-5km/h.

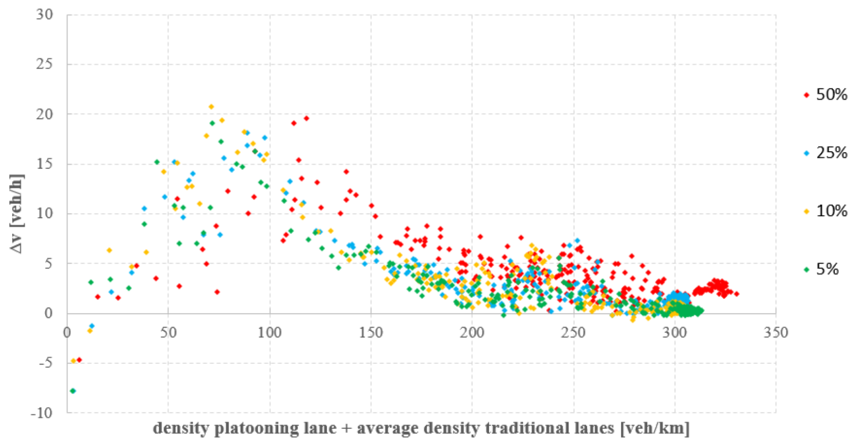


Figure 26. Speed difference between platooning and traditional lanes for a same time period

The average flow and density per lane can be therefore computed, regardless of the lane type. The following figure shows the average fundamental diagram per lane:

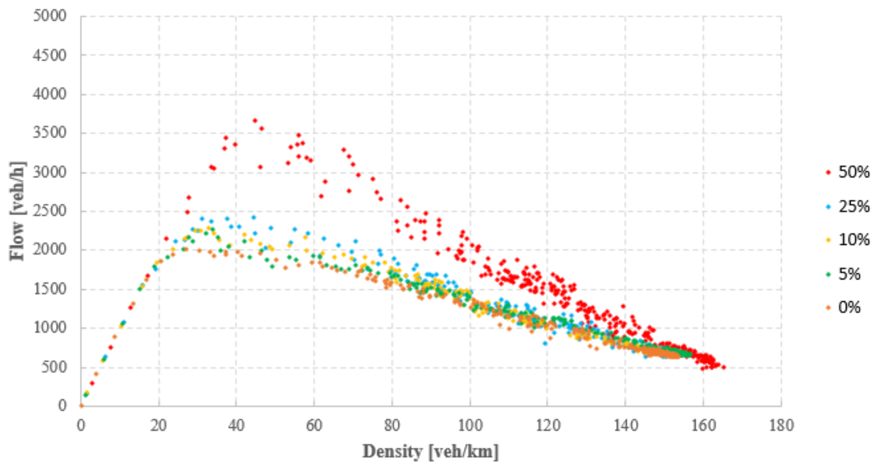


Figure 27. Average fundamental diagram per lane

It has been seen that for the scenario with a 50% of penetration rate of autonomous vehicles, there is a huge increase in traffic flow in the platooning lane, whereas for the rest of the scenarios this increase is not that important. This can only be attributed to the amount autonomous vehicles that are driving in a platoon, or, more specifically, to the number of platoon units with a significant length. In other words, if a total of 20 vehicles are driving in the platooning lane, it makes a big difference whether they are distributed in 2 platoons of 10 vehicles or in 10 platoons of 2 vehicles.

- Number of Platoons and Average Length

A key factor for the platooning concept is therefore the number of platoons that are present in the model and their average length. Any vehicle travelling in the platooning lane will be considered as a platooning unit. Irrespective of whether it is traveling alone or with followers. Scenario E will not be considered as there are no autonomous vehicles.

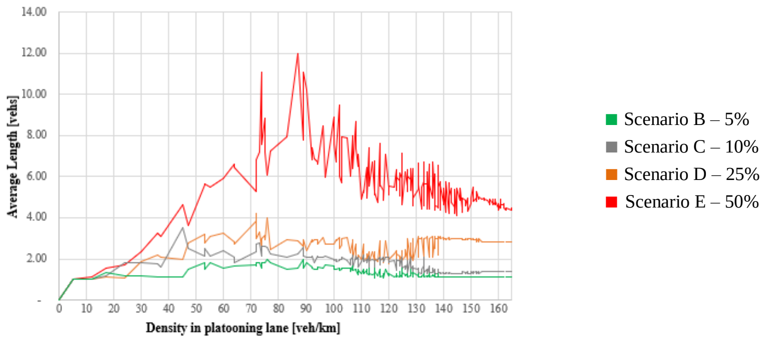


Figure 28. Average length of the platoons

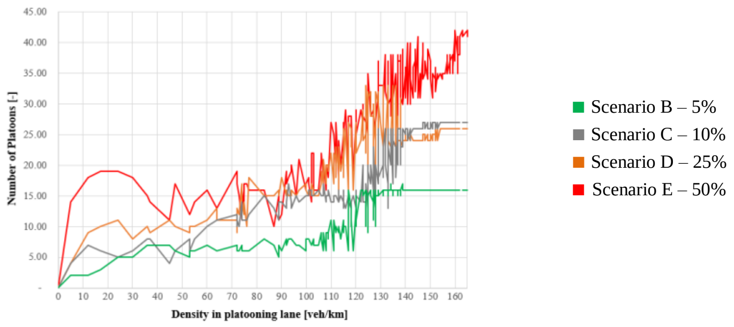


Figure 29. Average number of platoons

From the previous figures it can be observed that for penetration rates under 25% the platoon units hardly achieve platoon lengths of 2 vehicles. For a penetration rate of 25% the average length is of around 3 vehicles and for a penetration rates of 50%, average lengths range between a minimum of 4 vehicles and a maximum of 12 vehicles. Moreover, not only the length is greater but also the number of platoon units, explaining the high obtained throughputs for scenario E.

The average trip length of all the vehicles in the model tends to 15km, for lower penetration rates this length does not enable to maintain and provide enough autonomous vehicles flow in the platooning lane to augment the probability to form longer platoons. If we take a look in the proportion of flow in the platooning lane that corresponds to AVs, we observe the following:

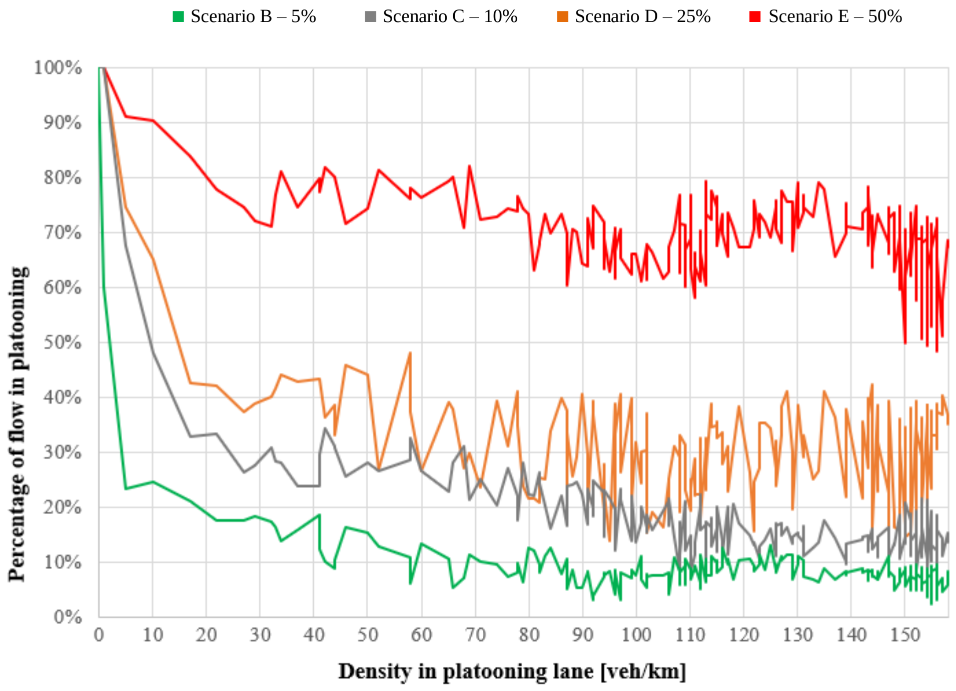
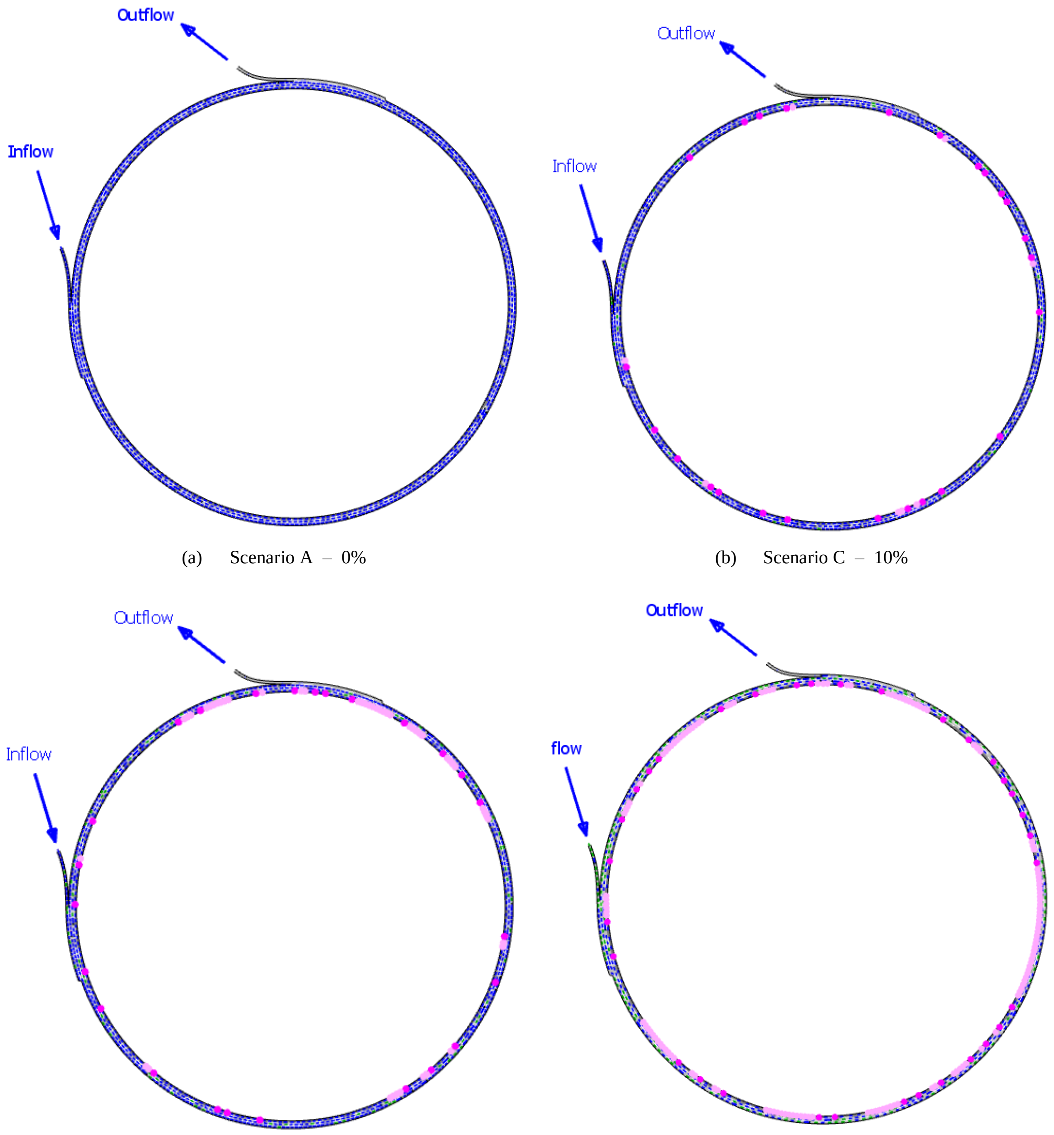


Figure 30. Percentage of flow in platooning lane of AVs

To ensure platooning efficiency, at least 50% of the flow in the platooning lane should be AVs. Note that a penetration of 50% AVs provides a flow of 70-80% AVs in the platooning lane and a penetration of 25% of AVs a flow of around 35%. Platooning technology has been shown to be very efficient, but only if vehicles have the option to platoon in longer units. For lower penetration rates either a higher trip length has to be set, or other strategies have to be used to control the proportion of AVs, for example, to enable AVs to access the platooning lane only if there is the option to platoon or if the expected trip distance is long, in order to ensure the vehicle stays for a longer period of time in the platooning lane.

In Figure X a screenshot at the end of the simulation is shown. This visually strengthens the previously drawn hypothesis.



(a) Scenario A – 0%

(b) Scenario C – 10%

(d) Scenario D – 25%

(c) Scenario E – 10%

Figure 31. Geometrical distribution of platoons

- Entry flow at the on-ramp

Traffic demand defines the average number of vehicles that aim to enter the network per hour. Vehicles join the main road using an on-ramp. As long as the input flow demand is lower than the capacity of the on-ramp this rate will be fulfilled.

When there is no traffic on the main stream, the maximum inflow can be reached. This is the capacity of the on-ramp. To find out this capacity by means of simulation a traffic demand that increases by 500veh/h every 2h is defined (see Figure 32), the point in which the on-ramp cannot handle the increasing input flows will be defined as the capacity.

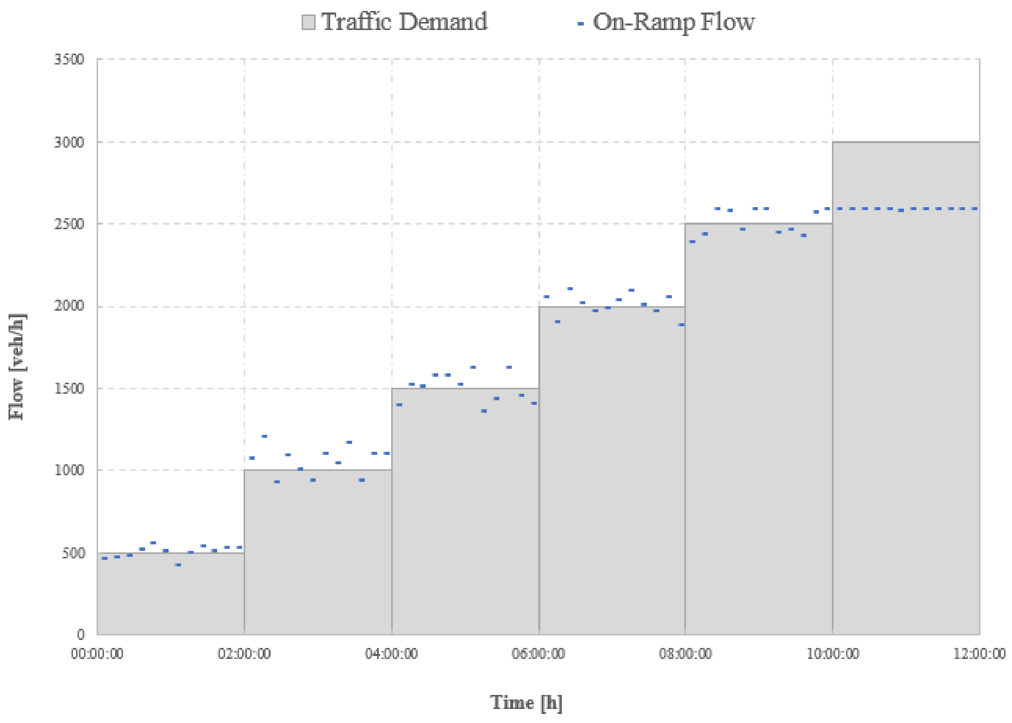


Figure 32. Traffic Demand at the on-ramp Vs On-Ramp Flow

From the figure above, it can be deduced that the maximum capacity of the on-ramp for a speed limit of 100km/h is around 2590veh/h. This is a very optimistic value as it assumes there are no vehicles in the highway. Most likely, vehicles will have to give way to traffic before merging and as soon as the main stream starts to be congested, the capacity of the on-ramp will drop significantly. If the demand is higher than this capacity, not all the vehicles will be able to access the roadway without delay and they will start to queue. It is important that the demand is served at the entry of a highway, because queues at such locations can cause spillbacks that spread very fast into the city roads. Therefore, it can be considered that congestion also starts when a proportion of the vehicles are not able to enter the highway. A performance indicator is therefore the amount of the demand that can be withstood at the entrance of the network after a certain period of time.

If we analyze how many vehicles are waiting to enter the model after 2h of simulation, we obtain the following chart for different constant demand patterns:

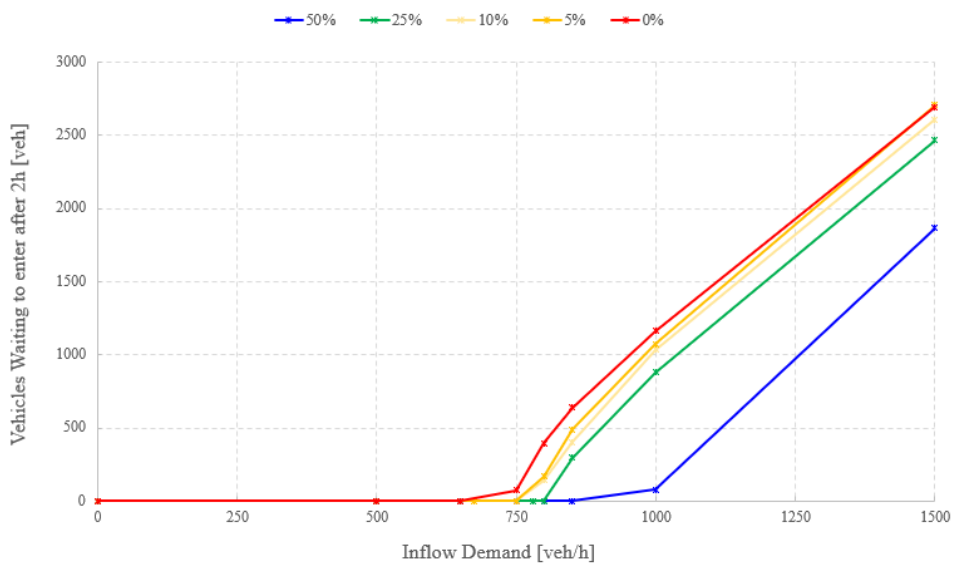


Figure 33. Number of vehicles that were not able to enter the network for different demands

Peak hour relates to the time of the day at which most of the trips take place. Demand is at its highest and congestion is likely to take place. The peak period often lasts more than one hour. For example, from 7-9h in the morning. In Figure 33 it is observed that for a constant inflow of 800 veh/h the only scenario that has served the demand during the two hours is scenario A, and it is even capable to bear 1000veh/h, whereas the rest of scenarios would have accumulated huge queues and delays at the entry of the network.

6. Conclusions and further research

The latest advances in telecommunications and engineering, are expected to make new instruments available for transportation. Grouping vehicles into platoons might be one of the technological benefits of autonomous and connected vehicles. Intelligent Transport Systems such as platooning aim to provide optimal solutions to road performance and safety. In this research, the platooning theory has been reviewed and a model has been defined to address the platooning concept.

To find an answer and asses the proposed strategy from the traffic engineering perspective, Aimsun Next, a microsimulation program has been used. Microsimulation allows to analyze complex mathematical models in a simple and graphic way. The simulator relies on models that reproduce the behavior of manually driven vehicles. To integrate the proposed methodology in the simulation, an algorithm has been programed that manipulates the autonomous vehicles during the simulation and updates them according to the defined platooning strategy.

Simulation has been applied to a three-lane ring road of approximately 1.5km length with a single on- and off-ramp. Five scenarios have been considered, in each scenario the penetration rate of autonomous vehicles varies, ranging from 0% to 50% of market penetration. Higher penetration ratios would probably need the development of a different strategy, e.g. enabling platoons to be formed in the middle lane as well.

For penetration rates over 25% the deployment of platooning technology has shown to increase traffic throughput significantly. Allowing two- or three-times higher flows in the platooning lane than if no platooning concept is applied. Platooning technology with a penetration rate of 50% has been capable to bear high traffic demands at the entry of the network, whereas the rest of scenarios with lower penetration rates have accumulated huge queues and delays at the on-ramp.

The general pattern shows that platooning concept with an autonomous vehicle market penetration of 30 to 50% increases the overall road significantly. However, lower penetration rates do not provide a sufficiently good improvement to consider the effort and economic cost of investing in developing platooning technologies. The research has shown that this is due to the inability to form long enough platooning units for low proportions of AVs. The length of the platoon has shown to be a key factor for the platooning concept.

Although results suggest that platooning could be a very good solution to coordinate autonomous vehicles when reasonable penetration rates are achieved, it is important to highlight that such models are very sensible to changes and may vary with a variety of other factors that have been assumed in this research.

The main achievement of this research is the definition and the development of an initial platooning system, its subsequent programming in Python language and the final implementation into Aimsun next. The developed tool can be subject to future modifications and will allow to analyze many new scenarios. Future research will be able to easily simulate and examine the platooning concept in Aimsun Next microsimulator modifying factors in the present algorithm like speed, average trip length, maximum platoon length, simulation parameters or even modifying the car-following and lane-changing behavior strategies. However, there might be other strategies or network layouts where the definition of the technology will need to be adapted, including greater penetration rates, the number of platooning lanes or even different geometry configurations.

From my point of view, a possible way to improve the current performance of the platooning concept could be to enable communication between vehicles driving in other lanes and coordinating those vehicles to cooperate with vehicles trying to leave the platoon. Modifying the infrastructure, could also enhance the performance, especially for high penetration rates, adding an additional on-ramp directly accessible from the platooning lane could be a very interesting strategy for high penetration rates.

This study has shown that the implementation of platooning technology is an optimal solution to address road performance with high penetration rates. In the first instance, when autonomous vehicles begin to coexist sharing the road with human-driven vehicles, platooning technology will difficultly help to improve overall traffic. However, technology is not static and new technologies and strategies are developed every day at fast speeds, that will offer solutions to address these first vehicles.

References

1. Aimsun 2017, Aimsun Next 8.3 User's Manual, Aimsun, Barcelona, Spain.

2. Bergenhem, Carl & Huang, Qihui & Benmimoun, Ahmed & Robinson, Tom. (2018). Challenges of Platooning on Public Motorways.

3. Edie, L.C., 1956. Discussion of traffic stream measurements and definitions. In: Almond, J. (Ed.), Proceedings of the Second International Symposium on the Theory of Traffic Flow. OECD, Paris, pp. 139 -154

4. Gipps, P.G., (1981), A behavioural car-following model for computer simulation, Transportation Research Part B: Methodological, 15, issue 2, p. 105-111, https://EconPapers.repec.org/RePEc:eee:transb:v:15:y:1981:i:2:p:105-111.

5. J. B. Kenney, "Dedicated Short-Range Communications (DSRC) Standards in the United States," in Proceedings of the IEEE, vol. 99, no. 7, pp. 1162-1182, July 2011.

6. Kishore Bhoopalam, Anirudh & Agatz, Niels & A. Zuidwijk, Rob. (2017). Planning of Truck Platoons: A Literature Review and Directions for Future Research. SSRN Electronic Journal. 10.2139/ssrn.2988195

7. The SAE On-Road Automated Vehicle Standards Committee. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles; SAE International: Warrendale, PA, USA, 2018.

8. Metropolitan Transport Authority of the Barcelona (ATM). Mobility survey 2018. EMEF-Enquesta de Mobilitat en dia Feiner (EMEF). Barcelona, Spain.

9. Zheng, Y.; Li, S.E.; Wang, J.; Cao, D.; Li, K. Stability and Scalability of Homogeneous Vehicular Platoon: Study on the Influence of Information Flow Topologies. IEEE Trans. Intell. Transp. Syst. 2016, 17, 14–26.

10. Zhou, Hao & Saigal, Romesh & Dion, Francois & Yang, Li. (2012). Vehicle Platoon Control in High-Latency Wireless Communications Environment: Model Predictive Control Method. Transportation Research Record: Journal of the Transportation Research Board. 2324. 81-90. 10.3141/2324-10.

11. B. van Arem, C. J. G. van Driel and R. Visser, "The Impact of Cooperative Adaptive Cruise Control on Traffic-Flow Characteristics," in IEEE Transactions on Intelligent Transportation Systems, vol. 7, no. 4, pp. 429-436, Dec. 2006.

12. G. Cecchini, A. Bazzi, B. M. Masini and A. Zanella, "Performance comparison between IEEE 802.11p and LTE-V2V in-coverage and out-of-coverage for cooperative awareness," 2017 IEEE Vehicular Networking Conference (VNC), Torino, 2017, pp. 109-114.

13. A. B. Böhm, et al., "Evaluating CALM M5-based vehicle-to-vehicle communication in various road settings through field trials," in 2010 IEEE 35th Conference on Local Computer Networks, 2010, pp. 613-620.

14. Liu, Zishan & Liu, Zhenyu & Meng, Zhen & Yang, Xinyang & Pu, Lin & Zhang, Lin. (2016). Implementation and performance measurement of a V2X communication system for vehicle and pedestrian safety. International Journal of Distributed Sensor Networks. 12. 10.1177/1550147716671267.

15. TNO report, 2007-D-R0280/B, Design and evaluation of an Integrated Full-Range Speed Assistant, March 20th, 2007.

16. Yashiro, T & Kasagi, K & Ariyasu, Kyoko & Kishida, S & Matsushita, Y. (1996). Construction and performance evaluation of the platoon-formation algorithm considering the destination of each vehicle. 35 - 40. 10.1109/IVS.1996.566347.

17. Wen, S., Guo, G., Chen, B., & Gao, X. (2018). Event-triggered cooperative control of vehicle platoons in vehicular ad hoc networks. Inf. Sci., 459, 341-353.

Appendix

This section includes the Python code conforming the platooning API.

```
from AAPI import *
import sys
from PyANGBasic import *
from PyANGKernel import *


#Get model from system
model = GKSystem.getSystem().getActiveModel()
#Define section type
sectionType = model.getType('GKSection')

# ---------------------------------VARIABLES---------------------------------------------

# radius of the lanes in [m]
r0 = 259
r1 = 256
r2 = 253
r3 = 250
r = [r0, r1, r2, r3]

# used when desiredDistance was not variable with speed
desiredDistance = 3 # m

# the extra distance added to the first platoon follower due to security reasons
securityDistance = 3 # m

# weight in the CACC formula for acceleration, speed error and distance error
#acceleration
k1 = 1.0
#speed
k2 = 0.3
#distance
k3 = 0.1
```

```
# -------------open OUTPUT FILE for platoon number and platoon length ------------------

platoonsFilePath = 'C:/Users/laura.oriol/Desktop/Outputs TFM/PN.csv'

openFile = open( platoonsFilePath, 'w')

#parameter to write outputs every minute

timeFactor = 1.0



# ----------------------------------DEBUG------------------------------------------------

IS_DEBUG = False

DEBUG_AIMSUN_ID = 5



# ------------------------Non variable info of sections and nodes----------------------


# Specify SubpathID containing the main highway route (no entry and exit sections)

subpathId = 1035


# Get the maximum allowed speeds and the type of sections

subpath = model.getCatalog().find(subpathId)

ruta = subpath.getRoute()

ruta_ids = []

maxspeed_sections_dict = dict()

normal_sections_dict = dict()

for section in ruta:

    if section.getDestination() != None:

        ruta_ids.append(int(section.getId()))

        ruta_ids.append(int(section.getDestination().getId()))

    else:

        ruta_ids.append(int(section.getId()))

    if section.getType() == sectionType:

        maxSpeed = section.getSpeed()

        maxspeed_sections_dict[section.getId()] = maxSpeed

        numberLanesEnd = section.getNbFullLanes()                           # Number of lanes that are complete for the 
complete length of the section

        numberLanesStart = section.getNbLanesAtPos(0.0)                     # Number of lanes at the beginning of the 
section

        numberOfFullLanes = section.getNbLanesAtPos(section.length2D())     # Number of lanes at the end of the section

        if numberLanesEnd == numberLanesStart == numberOfFullLanes:

            normal_sections_dict[section.getId()] = True
        else:
            normal_sections_dict[section.getId()] = False
for section in model.getCatalog().getObjectsByType( sectionType).itervalues():
    if section.getId() not in normal_sections_dict.keys():
        normal_sections_dict[section.getId()] = False
print("rutaids: %s" %ruta_ids)


# Length of the subpath
def lengthOfTheCircuit(ruta_ids):
    totalLength = 0
    for sectionID in ruta_ids:
        section = model.getCatalog().find(sectionID)
        if section.getType() == sectionType:
            totalLength += section.getLaneLength2D(0) #0 the left-most lane position and N-1 the right-most lane according
to the section direction
    print (totalLength)
    return totalLength

totalCircuiteLength = lengthOfTheCircuit(ruta_ids)



# --------------------- Define Vehicle Attribute Columns --------------------------------
platoonStatusCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "Status" ))
platoonLeavingCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "Leaving" ))
platoonNumberCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "PlatoonNumber" ))
platoonPositionCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "PlatoonPosition" ))
platoonPrecedingVehicleCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "PrecedingVehiclePlatoon" ))
platoonFollowingVehicleCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "FollowingVehiclePlatoon" ))
platoonLeaderCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "PlatoonLeader" ))
DistanceToLeaderCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "DistanceToLeader" ))
zoneCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "GKSimVehicle::zoneAtt" ))
obstacleTypeCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "GKSimVehicle::obstacleType" ))
initialSpeedLimitTypeCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "InitialSpeedLimitAcceptance" ))
nextSpeedCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "NextSpeed" ))
desiredDistanceToLeaderCol = ANGConnGetAttribute( AKIConvertFromAsciiString( "DesiredDistanceToLeader" ))


# ---------------------VEHILCE TYPE POSITION---------------------------------------------
```

```
def getPositionForVehicleName(name):
    numberVehicleTypes = AKIVehGetNbVehTypes()
    anyNonAsciiChar = boolp()
    for position in range(numberVehicleTypes+1):
        vehName = str(AKIConvertToAsciiString(AKIVehGetVehTypeName(position), True, anyNonAsciiChar) )
        if vehName == name:
            # print "%s" %position
            return position
    return None

 # ------------------------------CACC----------------------------------------------------
def CACC(followerId, leaderId, DD, distanceleaderfollower, desiredSpeeds):

    leader = AKIVehGetInf (leaderId)
    follower = AKIVehGetInf (followerId)

    if ANGConnGetAttributeValueInt(platoonPositionCol, ANGConnVehGetGKSimVehicleId(followerId)) == 1:
        leaderCurrentSpeed = leader.CurrentSpeed
        leaderPreviousSpeed = leader.PreviousSpeed
    else:
        leaderIdVehANG = ANGConnVehGetGKSimVehicleId(leaderId)
        leaderCurrentSpeed = ANGConnGetAttributeValueDouble(nextSpeedCol, leaderIdVehANG)
        leaderPreviousSpeed = leader.CurrentSpeed

    # Apply the CACC formula to calculate the desired acceleration

    #k1,k2 and k3 will be applied respectively to the following terms
    leaderCurrentAcc = ((leaderCurrentSpeed - leaderPreviousSpeed)/3.6) / AKIGetSimulationStepTime() # [m/s2] (v_leader - 
v_leader_old)/timeStep
    errorSpd = (leader.CurrentSpeed - follower.CurrentSpeed)/3.6 # [m/s] - V_leader-v_follwer
    errorDst = distanceleaderfollower - DD

    #the acceleration that the follower should apply
    caccAcc = (k1 * leaderCurrentAcc) + (k2 * errorSpd) + (k3 * errorDst) # [m/s2]

    #Constrain the calculated acceleration to have smooth movements
    acceleration = max(-3.0, min(1.5, caccAcc)) # [m/s2]
    #print "leader acceleration=%s - speed difference=%s - errorDst=%s - caccAcc=%s - acceleration=%s" %(leaderCurrentAcc, 
errorSpd, errorDst, caccAcc, acceleration)
```

```
    # Calculate the speed and constrain it
    newSpeed = min(desiredSpeeds, acceleration * AKIGetSimulationStepTime() * 3.6 + follower.CurrentSpeed) # [km/h] donde
esta el 55 tendria que ir un desiredSpeeds!!!

    if IS_DEBUG:
        if followerId == DEBUG_AIMSUN_ID:
            print ("---CACC---")
            print ("leader: %s follower: %s leadercurrentacc: %s errorSpd: %s errorDst: %s") %(leaderId, followerId,
leaderCurrentAcc, errorSpd, errorDst)
            print (caccAcc)
            print (acceleration)
            print (newSpeed)
    return newSpeed



# ----------------------------------- Leader Speeed ------------------------------------

#not used
def platoonLeaderSpeed(platoonLeaderId):
    platoonLeaderInf = AKIVehGetInf (platoonLeaderId)
    if platoonLeaderInf.PreviousSpeed < maxSpeed:
        platoonLeaderSpeed = min( maxSpeed, platoonLeaderInf.PreviousSpeed + 1*3.6*AKIGetSimulationStepTime() )
    elif platoonLeaderInf.PreviousSpeed > maxSpeed:
        platoonLeaderSpeed = max( maxSpeed, platoonLeaderInf.PreviousSpeed - 1*3.6*AKIGetSimulationStepTime() )
    elif platoonLeaderInf.PreviousSpeed == maxSpeed:
        platoonLeaderSpeed = maxSpeed
    return platoonLeaderSpeed

# ---------------------------------- XY DISTANCE --------------------------------------

#not used
def xyDistance(leader, follower):
    distance = ((leader.xCurrentPosBack - follower.xCurrentPos)**2 + (leader.yCurrentPosBack -
follower.yCurrentPos)**2)**0.5 # [m]
    return distance



# -------------------------EXPLORE NETWORK FIND VEHICLES --------------------------------
```

```
def explore_network_find_vehicles():
    vehicles_prelist = []
    vehicles_preIDlist = []
    p = [0, 0, 0, 0]
    for idSectionOrNode in ruta_ids:
        sectionOrNode = model.getCatalog().find(idSectionOrNode)
        if sectionOrNode.getType() == sectionType:
            # print "it's a section %s" %idSectionOrNode
            nbVehs = AKIVehStateGetNbVehiclesSection(idSectionOrNode, True)
            for vehPos in range((nbVehs-1),-1,-1):
                car = AKIVehStateGetVehicleInfSection(idSectionOrNode, vehPos)
                #folID = AKIVehGetFollowerId(car.idVeh)
                #print "leader:%s follower:%s" %(car.idVeh, folID)
                idVehANG = ANGConnVehGetGKSimVehicleId(car.idVeh)
                zone = ANGConnGetAttributeValueInt(zoneCol, idVehANG)
                if car.report ==
0:
                    if normal_sections_dict[idSectionOrNode] is True:
                        carLane = car.numberLane                            # The lane number in the segment (from 1: the
rightmost lane, to N: the leftmost lane)
                    else:
                        carLane = car.numberLane -1
                carPosition = ( car.CurrentPos/ r[carLane]) * r[3] + p[3]
                #print "ID: %s CurrentPos: %s CarPos:%s Lane: %s" %(car.idVeh, car.CurrentPos, carPosition, carLane)
                vehicles_prelist.append([car.idVeh, car.type, carPosition, carLane, idSectionOrNode, zone, idVehANG])
                vehicles_preIDlist.append(car.idVeh)
            p0 = p[0] + sectionOrNode.getLaneLength2D(3)
            p1 = p[1] + sectionOrNode.getLaneLength2D(2)    # The lane number (from 0: the leftmost lane, to N-1: the
rightmost lane)
            p2 = p[2] + sectionOrNode.getLaneLength2D(1)
            p3 = p[3] + sectionOrNode.getLaneLength2D(0)
            p = [p0, p1, p2, p3]
        else:
            # print "it's a node %s" %idSectionOrNode
            nbVehs = AKIVehStateGetNbVehiclesJunction(idSectionOrNode)
            for vehPos in range((nbVehs-1),-1,-1):
                print ("cuidado que ha entrado en el nodo: %s") %idSectionOrNode
                car = AKIVehStateGetVehicleInfJunction(idSectionOrNode, vehPos)
                idVehANG = ANGConnVehGetGKSimVehicleId(car.idVeh)
                zone = ANGConnGetAttributeValueInt(zoneCol, idVehANG)
                if normal_sections_dict[car.idSectionTo] is True:
                    carLane = car.idLaneTo
                else:
                    carLane = car.idLaneTo - 1
                carPosition = car.CurrentPos
                vehicles_prelist.append([car.idVeh, car.type, carPosition, carLane, idSectionOrNode, zone, idVehANG])
                vehicles_preIDlist.append(car.idVeh)

    vehicles_list = sorted(vehicles_prelist, key=lambda Pos: Pos[2] , reverse=True)
    vehicles_IDlist = []

    m=0
    for item in vehicles_list:
        vehicles_IDlist.append(vehicles_list[m][0])
        m = m + 1

    #print vehicles_list
    return vehicles_list, vehicles_IDlist

# -------------------------SET ATTRIBUTE -------------------------------

def setAttributeCar(newAttribute, idVehANG, column):

    #Set a new attribute of type int, str or float to the car
    if type(newAttribute) == int:
        if ANGConnGetAttributeValueInt(column, idVehANG) != newAttribute:
            ANGConnSetAttributeValueInt(column, idVehANG, newAttribute)
    elif type(newAttribute) == str:
        if ANGConnGetAttributeValueStringA(column, idVehANG) != newAttribute:
            ANGConnSetAttributeValueStringA(column, idVehANG, newAttribute)
    elif type(newAttribute) == float:
        if ANGConnGetAttributeValueDouble(column, idVehANG) != newAttribute:
            ANGConnSetAttributeValueDouble(column, idVehANG, newAttribute)
    else:
        print ("The attribute does not belong to the types String, Integer or Float")
    return
```

```
# -------------------------DEF LEADER -------------------------------

def managePlatoonLeader (platoonleaderId, idVehANG, leaderId, distanceleaderfollower):
    AKIVehTrackedModifyLane(platoonleaderId, 0)
    parametersleader = AKIVehTrackedGetStaticInf(platoonleaderId)
    parametersleader.speedAcceptance = 1.0
    parametersleader.sensitivityFactor = 0
    parametersleader.minDistanceVeh = 0.5
    parametersleader.reactionTime = 0.1
    parametersleader.reactionTimeAtStop = 0.1
    parametersleader.reactionTimeAtTrafficLight = 0.1
    #parametersleader.type = selfDrivingCarPosL3
    AKIVehTrackedSetStaticInf(platoonleaderId, parametersleader)
    IDSection = AKIVehGetInf(platoonleaderId).idSection
    desiredSpeedPL = maxspeed_sections_dict[IDSection]
    #print "ID: %s maxSpeed: %s" %(platoonleaderId,desiredSpeedPL)
    platoonLeaderCurrentSpeed = AKIVehTrackedGetInf(platoonleaderId).CurrentSpeed
    dD = max( 0.5, (platoonLeaderCurrentSpeed/3.6*0.8) )
    setAttributeCar(dD, idVehANG, desiredDistanceToLeaderCol)
    newSpeed = CACC(platoonleaderId, leaderId, dD, distanceleaderfollower, desiredSpeedPL)
    setAttributeCar(newSpeed, idVehANG, nextSpeedCol)
    # Update the speed of the vehicle
    res = AKIVehTrackedModifySpeed(platoonleaderId, newSpeed)

    if IS_DEBUG:
        if platoonleaderId == DEBUG_AIMSUN_ID:
            print ("-- Platoon Leader --")
            print ("ID: %s res: %s ") %(platoonleaderId,res)




# -------------------------DEF FOLLOWER -------------------------------

def manageFollower (followerId, idVehANG, leaderId, dD, distanceleaderfollower):
    AKIVehTrackedModifyLane(followerId, 0)
    parametersfollower = AKIVehTrackedGetStaticInf(followerId)
    parametersfollower.speedAcceptance = 1.1
    parametersfollower.sensitivityFactor = 0
    parametersfollower.minDistanceVeh = 0.5
    parametersfollower.reactionTime = 0.1
    #parametersfollower.type = selfDrivingCarPosL3
    parametersfollower.reactionTimeAtStop = 0.1
    parametersfollower.reactionTimeAtTrafficLight = 0.1
    AKIVehTrackedSetStaticInf(followerId, parametersfollower)
    desiredSpeeds = min(parametersfollower.maxDesiredSpeed, parametersfollower.speedAcceptance * maxSpeed)
    newSpeed = CACC(followerId, leaderId, dD, distanceleaderfollower, desiredSpeeds)    # Calculate the speed by applying
the CACC formula
    setAttributeCar(newSpeed, idVehANG, nextSpeedCol)
    # Update the speed of the vehicle
    res = AKIVehTrackedModifySpeed(followerId, newSpeed)

    if IS_DEBUG:
        if followerId == DEBUG_AIMSUN_ID:
            print ("-- Follower --")
            print ("ID: %s res: %s ") %(followerId, res)

# -------------------------DEF JOINED ------------------------------

def manageJoined (joinedId, idVehANG, leaderId, platoonLeaderId, distanceleaderfollower, DD, cumulative):
    AKIVehTrackedModifyLane(joinedId, 0)
    parametersjoined = AKIVehTrackedGetStaticInf(joinedId)
    parametersjoined.speedAcceptance = 1.1
    parametersjoined.sensitivityFactor = 0
    parametersjoined.minDistanceVeh = 0.5 #testDist - 0.3
    parametersjoined.reactionTime = 0.1
    #parametersjoined.type = selfDrivingCarPosL3
    parametersjoined.reactionTimeAtStop = 0.1
    parametersjoined.reactionTimeAtTrafficLight = 0.1
    AKIVehTrackedSetStaticInf(joinedId, parametersjoined)
    platoonLeaderCurrentSpeed = AKIVehTrackedGetInf(platoonLeaderId).CurrentSpeed
    platoonLeaderPreviousSpeed = AKIVehTrackedGetInf(platoonLeaderId).PreviousSpeed
    platoonLeaderCurrentAcc = ((platoonLeaderCurrentSpeed - platoonLeaderPreviousSpeed)/3.6) / AKIGetSimulationStepTime()
    #adjustedPlatoonLeaderSpeed = platoonLeaderCurrentSpeed + platoonLeaderCurrentAcc*AKIGetSimulationStepTime()*3.6
    platoonLeaderIdVehANG = ANGConnVehGetGKSimVehicleId(platoonLeaderId)
    adjustedPlatoonLeaderSpeed = ANGConnGetAttributeValueDouble(nextSpeedCol, platoonLeaderIdVehANG)
    #print "%s speedleader %s" %(joinedId, adjustedPlatoonLeaderSpeed)
    joinedCurrentSpeed = AKIVehTrackedGetInf(joinedId).CurrentSpeed
    dd = max(0.5, (adjustedPlatoonLeaderSpeed/3.6*0.11))

    leaderIdVehANG = ANGConnVehGetGKSimVehicleId(leaderId)

    if ANGConnGetAttributeValueInt(platoonPositionCol, ANGConnVehGetGKSimVehicleId(joinedId)) == 2:

        dd = dd + securityDistance

        joinedSpeed = adjustedPlatoonLeaderSpeed + (distanceleaderfollower-dd)*3.6/AKIGetSimulationStepTime()*0.1

        cumulative = (distanceleaderfollower-dd)*3.6/AKIGetSimulationStepTime()

        #print "cumulative from first platoon %s" %cumulative

    else:

        leaderNewSpeed = ANGConnGetAttributeValueDouble(nextSpeedCol, leaderIdVehANG)

        distanceDifference = DD - distanceleaderfollower

        joinedSpeed = leaderNewSpeed + (distanceleaderfollower-dd)*3.6/AKIGetSimulationStepTime()*0.1


    if joinedSpeed >= joinedCurrentSpeed:

        nextSpeed = min (joinedSpeed, joinedCurrentSpeed + 2 * AKIGetSimulationStepTime() * 3.6)

        cumulative = nextSpeed - adjustedPlatoonLeaderSpeed


    elif joinedSpeed < joinedCurrentSpeed:

        nextSpeed = max (joinedSpeed, joinedCurrentSpeed - 1.5 * AKIGetSimulationStepTime() * 3.6)

        cumulative = nextSpeed - adjustedPlatoonLeaderSpeed


    setAttributeCar(nextSpeed, idVehANG, nextSpeedCol)

    # Update the speed of the vehicle

    res = AKIVehTrackedModifySpeed(joinedId, nextSpeed)


    if IS_DEBUG:

        if joinedId == DEBUG_AIMSUN_ID:

            print ("-- Joined --")

            print ("ID: %s res: %s ") %(joinedId, res)


    return cumulative



# -------------------------Lane Changint to Right -------------------------------


def manageLaneChangingRight (current, leaderChangingLane, vehicles_list, index):

    existingFollowerChangingLane = False

    for followerChangingLane in vehicles_list[index:]:

        if existingFollowerChangingLane is True:

            break
        if followerChangingLane[2] < current[2] and followerChangingLane[3] == 2 and not existingFollowerChangingLane:
            existingFollowerChangingLane = True
            #print "%s leader %s posicion:" %(leaderChangingLane[0], leaderChangingLane[2])
            #print "%s follower %s posicion:" %(followerChangingLane[0], followerChangingLane[2])
            if  leaderChangingLane != None:
                spaceCooperation  = leaderChangingLane[2] - followerChangingLane[2] - 4
                spaceBack = current[2] - followerChangingLane[2] - 4
                if spaceBack>11 and spaceCooperation-spaceBack-4>10:
                    AKIVehTrackedModifyLane(current[0], -1)
                    #print "%s   ya que tiene espacio suficiente zona trasera, hay coche delante y atras" %current[0]
                elif 15>spaceBack>12 and spaceCooperation>26:
                    AKIVehTrackedModifyLane(current[0], -1)
                    #print "%s   ya que tiene espacio suficiente entre vehiculos, hay coche delante y atras"
%current[0]
                else:
                    AKIVehTrackedModifyLane(current[0], 0)
                    #print "%s   no cambiamos" %current[0]
            else:
                spaceBack = current[2] - followerChangingLane[2]
                if spaceBack > 10:
                    AKIVehTrackedModifyLane(current[0], -1)
                    #print "%s   cambio porque hay mas de 15m detras y no hay vehiculo delante" %current[0]
                else:
                    AKIVehTrackedModifyLane(current[0], 0)
                    #print "%s   no cambio porque no hay mas de 10m detras(no hay veh delante)" %current[0]
    if not existingFollowerChangingLane and leaderChangingLane is None:
        AKIVehTrackedModifyLane(current[0], -1)
    if not existingFollowerChangingLane and leaderChangingLane is not None:
        AKIVehTrackedModifyLane(current[0], -1)



# -------------------------Lane Changint to Left ------------------------------
def manageLaneChangingLeft (current, leaderChangingLane, vehicles_list, index):
    existingFollowerChangingLane = False
    for followerChangingLane in vehicles_list[index:]:
        if existingFollowerChangingLane is True:
            break
        if followerChangingLane[2] < current[2] and followerChangingLane[3] == (current[3]+1) and not
existingFollowerChangingLane:
```

```
            existingFollowerChangingLane = True
            if  leaderChangingLane != None:
                setAttributeCar(leaderChangingLane[0], current[6], platoonPrecedingVehicleCol)
                setAttributeCar(followerChangingLane[0], current[6], platoonFollowingVehicleCol)
                spaceCooperation  = leaderChangingLane[2] - followerChangingLane[2] - 4
                spaceBack = current[2] - followerChangingLane[2] - 4
                if spaceBack>10 and spaceCooperation-spaceBack-4>10:
                    AKIVehTrackedModifyLane(current[0], 1)
                    #print "%s   ya que tiene espacio suficiente zona trasera, hay coche delante y atras" %current[0]
                elif 10>spaceBack>8 and spaceCooperation>25:
                    AKIVehTrackedModifyLane(current[0], 1)
                    #print "%s   ya que tiene espacio suficiente entre vehiculos, hay coche delante y atras"
%current[0]
                else:
                    AKIVehTrackedModifyLane(current[0], 0)
                    #print "%s   no cambiamos" %current[0]
            else:
                spaceBack = current[2] - followerChangingLane[2]
                if spaceBack > 8:
                    AKIVehTrackedModifyLane(current[0], 1)
                    #print "%s   cambio porque hay mas de 15m detras y no hay vehiculo delante" %current[0]
                else:
                    AKIVehTrackedModifyLane(current[0], 0)
                    #print "%s   no cambio porque no hay mas de 10m detras(no hay veh delante)" %current[0]

    if not existingFollowerChangingLane and leaderChangingLane is None:
        AKIVehTrackedModifyLane(current[0], 1)
        #print "%s   cambiamos porque leader es none y follower tambien" %current[0]
    if not existingFollowerChangingLane and leaderChangingLane is not None:
        AKIVehTrackedModifyLane(current[0], 1)
        #print "%s   cambiamos porque follower es none" %current[0]




def AAPILoad():
    AKIPrintString( "AAPILoad" )
    return 0


def AAPIInit():
    # AKIPrintString( "AAPIInit" )
    global selfDrivingCarPos
    global carPos
    global truckPos
    #global carPosL3
    #global selfDrivingCarPosL3
    ANGConnEnableVehiclesInBatch(True)
    selfDrivingCarPos = getPositionForVehicleName("CarSelfDriving")
    carPos = getPositionForVehicleName("Car")
    truckPos = getPositionForVehicleName("Truck")
    #carPosL3 = getPositionForVehicleName("CarL3")
    #selfDrivingCarPosL3 = getPositionForVehicleName("CarSelfDrivingL3")
    #print selfDrivingCarPosL3

    return 0

def AAPIManage(time, timeSta, timeTrans, acycle):
    # AKIPrintString( "AAPIManage" )
    return 0

#________________________________________________________________________________________
def AAPIPostManage(time, timeSta, timeTrans, acycle):
    global selfDrivingCarPos
    global carPos
    global truckPos
    #global carPosL3
    #global selfDrivingCarPosL3
    global timeFactor
    global desiredDistance
    global cumulative

    cumulative = 0

    #print timeFactor
    #print time/60.0

    isWritingTime = False
    if time/60 == timeFactor:
        print time/60.0
        isWritingTime = True

        timeFactor += 1.0


    #exploramos la red para generar una serie de listas de coches

    vehicles_list, vehicles_IDlist = explore_network_find_vehicles()

    #print vehicles_IDlist



    # vehicle = [car.idVeh, car.type, carPositionX, carLane, idSectionOrNode, zone, idVehANG]


    platoonNum = 0

    platoonPos = 0

    resto = 0


    leader1VA = False

    leader2VA = False

    leader3VA = False


    leader1 = None

    leader2 = None

    leader3 = None


    for veh in vehicles_list:

        if veh[3] == 1:

            newPos = totalCircuiteLength + veh[2]

            leader1 = [veh[0], veh[1], newPos, veh[3], veh[4], veh[5], veh[6]]

            if leader1[1] == selfDrivingCarPos:

            #if ((leader1[1] == selfDrivingCarPos) or (leader1[1] == selfDrivingCarPosL3)):

                leader1VA = True

            else:

                leader1VA = False

            continue

        elif veh[3] == 2:

            newPos = totalCircuiteLength + veh[2]

            leader2 = [veh[0], veh[1], newPos, veh[3], veh[4], veh[5], veh[6]]

            if leader2[1] == selfDrivingCarPos:

            #if ((leader2[1] == selfDrivingCarPos) or (leader2[1] == selfDrivingCarPosL3)):

                leader2VA = True

            else:
                leader2VA = False
            continue
        elif veh[3] == 3:
            newPos = totalCircuiteLength + veh[2]
            leader3 = [veh[0], veh[1], newPos, veh[3], veh[4], veh[5], veh[6]]
            if leader3[1] == selfDrivingCarPos:
            #if ((leader3[1] == selfDrivingCarPos) or (leader3[1] == selfDrivingCarPosL3)):
                leader3VA = True
            else:
                leader3VA = False
            continue
    if leader3 != None and leader3VA == True:
        platoonPos = ANGConnGetAttributeValueInt(platoonPositionCol, leader3[6])
        platoonLeaderId = ANGConnGetAttributeValueInt(platoonLeaderCol, leader3[6])
        PP = platoonPos

    #print "leader1: %s leader2: %s leader3: %s" %(leader1,leader2,leader3)

    for index, current in enumerate(vehicles_list):
        idVehANG = current[6]

        #carril 3
        if current[3] == 3:
            if current[1] != selfDrivingCarPos:
            #if ((current[1] != selfDrivingCarPos) and (current[1] != selfDrivingCarPosL3)):    # -----------------------
-----------------------no es VA
                leader3VA = False
                leader3 = current
            elif current[1] == selfDrivingCarPos and not leader3VA:
            #elif ((current[1] == selfDrivingCarPos or current[1] == selfDrivingCarPosL3)  and not leader3VA): #----------
------------------VA y no hay leader
                distanceToLeader = leader3[2] - current[2] - 4
                setAttributeCar(distanceToLeader, idVehANG, DistanceToLeaderCol)
                leader3VA = True
                if isWritingTime and platoonPos > 0:
                    if platoonNum == 0:
                        resto = platoonPos - PP
                        #print "%s el resto es %s" %(time/60, resto)
                    else:
                        openFile.write( '%s,%s,%s \n' %(time/60, platoonNum, platoonPos))
                        #print " %s: %s %s" %(time/60, platoonNum, platoonPos)
                platoonNum += 1
                platoonPos = 1
                setAttributeCar("PlatoonLeader", idVehANG, platoonStatusCol)
                setAttributeCar(leader3[0], idVehANG, platoonPrecedingVehicleCol)
                setAttributeCar(platoonNum, idVehANG, platoonNumberCol)
                setAttributeCar(platoonPos, idVehANG, platoonPositionCol)
                platoonLeaderId = current[0]
                setAttributeCar(platoonLeaderId, idVehANG, platoonLeaderCol)
                managePlatoonLeader (platoonLeaderId, idVehANG, leader3[0], distanceToLeader)
                leader3 = current
                if IS_DEBUG:
                    if current[0] == DEBUG_AIMSUN_ID:
                        print "%s is platoonleader "
            elif current[1] == selfDrivingCarPos and leader3VA: #--------------------------------VA y si hay leader
            #elif ((current[1] == selfDrivingCarPos or current[1] == selfDrivingCarPosL3) and leader3VA):
                distanceToLeader = leader3[2] - current[2] - 4
                setAttributeCar(distanceToLeader, idVehANG, DistanceToLeaderCol)
                if ANGConnGetAttributeValueStringA(platoonStatusCol, leader3[6]) == "PlatoonLeader": #el leadera actual es
platoonleader
                    currentSpeed = AKIVehGetInf(current[0]).CurrentSpeed
                    desiredDistance = max(0.5, (currentSpeed/3.6*0.11))
                    #vL=AKIVehGetInf(leader3[0]).CurrentSpeed
                    #securityDistance = max( 0.5, 1/12/3.6/3.6*1.1*((currentSpeed**2)-(vL**2))+1.1*0.1*currentSpeed/3.6 )
                    #if IS_DEBUG:
                        #print "aqui"
                        #print desiredDistance
                        #print securityDistance
                    DD = desiredDistance + securityDistance
                else:
                    DD = desiredDistance
                setAttributeCar(DD, idVehANG, desiredDistanceToLeaderCol)
                if DD*0.8<distanceToLeader<DD*1.2 and ANGConnGetAttributeValueStringA(platoonStatusCol, idVehANG) ==
"Joined" and ANGConnGetAttributeValueInt(platoonPrecedingVehicleCol, idVehANG) == leader3[0]:
                    platoonPos += 1
                    setAttributeCar(leader3[0], idVehANG, platoonPrecedingVehicleCol)
                    setAttributeCar(platoonNum, idVehANG, platoonNumberCol)
                    setAttributeCar("Joined", idVehANG, platoonStatusCol)
                    setAttributeCar(platoonPos, idVehANG, platoonPositionCol)
                    if IS_DEBUG:
                        if current[0] == DEBUG_AIMSUN_ID:
                            print "%s is joined "
                    if ANGConnGetAttributeValueStringA(platoonStatusCol, leader3[6]) == "Follower":
                        platoonLeaderId = leader3[0]
                        setAttributeCar(platoonLeaderId, idVehANG, platoonLeaderCol)
                        cumulative = manageJoined (current[0], current[6], leader3[0], platoonLeaderId, distanceToLeader,
DD, cumulative)
                        #print cumulative
                    else:
                        setAttributeCar(platoonLeaderId, idVehANG, platoonLeaderCol)
                        cumulative = manageJoined (current[0], current[6], leader3[0], platoonLeaderId, distanceToLeader,
DD, cumulative)
                        #print cumulative
                    leader3VA = True
                    leader3 = current
                else:
                    if (DD-0.1 <= distanceToLeader <= DD+0.1):
                        platoonPos += 1
                        setAttributeCar("Joined", idVehANG, platoonStatusCol)
                        setAttributeCar(leader3[0], idVehANG, platoonPrecedingVehicleCol)
                        setAttributeCar(platoonNum, idVehANG, platoonNumberCol)
                        setAttributeCar(platoonPos, idVehANG, platoonPositionCol)
                        setAttributeCar(platoonLeaderId, idVehANG, platoonLeaderCol)
                        if IS_DEBUG:
                            if current[0] == DEBUG_AIMSUN_ID:
                                print "%s is joined aqui "
                        cumulative = manageJoined (current[0], current[6], leader3[0], platoonLeaderId, distanceToLeader,
DD, cumulative)
                        #print cumulative
                        leader3VA = True
                        leader3 = current
                    elif (0 <= distanceToLeader < DD-0.1 or DD+0.1 < distanceToLeader < 25):
                        platoonPos += 1
                        setAttributeCar("Follower", idVehANG, platoonStatusCol)
                        setAttributeCar(leader3[0], idVehANG, platoonPrecedingVehicleCol)
                        setAttributeCar(platoonNum, idVehANG, platoonNumberCol)
                        setAttributeCar(platoonPos, idVehANG, platoonPositionCol)
                        setAttributeCar(platoonLeaderId, idVehANG, platoonLeaderCol)
                        manageFollower (current[0], idVehANG, leader3[0], DD, distanceToLeader)
                        if IS_DEBUG:
                            if current[0] == DEBUG_AIMSUN_ID:
                                print "%s is follower "
                        leader3VA = True
                        leader3 = current
                    else:
                        leader3VA = True
                        if isWritingTime: #if platoonNum>0 and isWritingTime:
                            if platoonNum == 0:
                                resto = platoonPos - PP
                                #print "%s el resto es %s" %(time/60, resto)
                            else:
                                openFile.write( '%s,%s,%s \n' %(time/60, platoonNum, platoonPos))
                                #print "%s: %s %s" %(time/60, platoonNum, platoonPos)
                        platoonNum += 1
                        platoonPos = 1
                        setAttributeCar("PlatoonLeader", idVehANG, platoonStatusCol)
                        setAttributeCar(leader3[0], idVehANG, platoonPrecedingVehicleCol)
                        setAttributeCar(platoonNum, idVehANG, platoonNumberCol)
                        setAttributeCar(platoonPos, idVehANG, platoonPositionCol)
                        if IS_DEBUG:
                            if current[0] == DEBUG_AIMSUN_ID:
                                print "%s is platoonleader aqui abajo"
                        platoonLeaderId = current[0]
                        setAttributeCar(platoonLeaderId, idVehANG, platoonLeaderCol)
                        managePlatoonLeader (platoonLeaderId, idVehANG, leader3[0], distanceToLeader)
                        leader3 = current
            if current[5] == 2 and current[1] == selfDrivingCarPos:
            #if current[5] == 2 and (current[1] == selfDrivingCarPos or current[1] == selfDrivingCarPosL3): #and
ANGConnGetAttributeValueStringA(obstacleTypeCol, idVehANG) == "Turn":
                setAttributeCar("Yes", idVehANG, platoonLeavingCol)
                manageLaneChangingRight (current, leader2, vehicles_list, index)
            if current[5] == 3 and current[1] == selfDrivingCarPos:
            #if  current[5] == 3 and (current[1] == selfDrivingCarPos or current[1] == selfDrivingCarPosL3): #and
ANGConnGetAttributeValueStringA(obstacleTypeCol, idVehANG) == "Turn":
                setAttributeCar("Urgent", idVehANG, platoonLeavingCol)
                AKIVehTrackedModifyLane(current[0], -1)
```

```
        #carril 2
        if current[3] == 2:
            if current [1] == 3 and current[5] == 1 and ANGConnGetAttributeValueStringA(platoonLeavingCol, idVehANG) !=
"Yes" and ANGConnGetAttributeValueStringA(platoonLeavingCol, idVehANG) != "Urgent":
                manageLaneChangingLeft (current, leader3, vehicles_list, index)
                leader2 = current
            elif current [1] == 3 and current[5] == 2 and ANGConnGetAttributeValueStringA(obstacleTypeCol, idVehANG) ==
"Turn":
                setAttributeCar("Yes", idVehANG, platoonLeavingCol)
                manageLaneChangingRight (current, leader2, vehicles_list, index)
                #AKIVehSetAsNoTracked (current [0])
                leader2 = current
            elif current [1] == 3 and current[5] == 3 and ANGConnGetAttributeValueStringA(obstacleTypeCol, idVehANG) ==
"Turn":
                setAttributeCar("Urgent", idVehANG, platoonLeavingCol)
                AKIVehTrackedModifyLane(current[0], -1)
                #AKIVehSetAsNoTracked (current [0])
                leader2 = current
            else:
                leader2 = current

        #carril 1
        if current[3] == 1:
            if current[1] == selfDrivingCarPos and AKIVehTrackedGetStaticInf(current[0]).sensitivityFactor == 0:
                parameterscurrent = AKIVehTrackedGetStaticInf(current[0])
                parameterscurrent.sensitivityFactor = 1
                parameterscurrent.minDistanceVeh = 1
                AKIVehTrackedSetStaticInf(current[0], parameterscurrent)
            if current [1] == selfDrivingCarPos and current[5] == 1 and ANGConnGetAttributeValueStringA(platoonLeavingCol,
idVehANG) != "Yes" and ANGConnGetAttributeValueStringA(platoonLeavingCol, idVehANG) != "Urgent":
                manageLaneChangingLeft (current, leader2, vehicles_list, index)
                leader1 = current
            elif current [1] == selfDrivingCarPos and current[5] == 2 and ANGConnGetAttributeValueStringA(obstacleTypeCol,
idVehANG) == "Turn":
                setAttributeCar("Yes", idVehANG, platoonLeavingCol)
                AKIVehTrackedModifyLane(current[0], 0)
                leader1 = current
            elif current [1] == selfDrivingCarPos and current[5] == 3 and ANGConnGetAttributeValueStringA(obstacleTypeCol,
idVehANG) == "Turn":
                setAttributeCar("Urgent", idVehANG, platoonLeavingCol)
                AKIVehTrackedModifyLane(current[0], -1)
                leader1 = current
            else:
                leader1 = current

    if isWritingTime: #if platoonNum>0 and isWritingTime:
        openFile.write( '%s,%s,%s \n' %(time/60, platoonNum, platoonPos+resto))
        #print "3. %s: %s %s" %(time/60, platoonNum, platoonPos+resto)

    return timeFactor

#________________________________________________________________________________________

def AAPIFinish():
    openFile.close()
    # AKIPrintString( "AAPIFinish" )
    return 0

def AAPIUnLoad():
    # AKIPrintString( "AAPIUnLoad" )
    return 0

def AAPIPreRouteChoiceCalculation(time, timeSta):
    # AKIPrintString( "AAPIPreRouteChoiceCalculation" )
    return 0

def AAPIEnterVehicle(idveh, idsection):
    # Automatically set the vehicle as Tracked when it enters the model if it of type autonomous
    car = AKIVehGetInf(idveh)
    if car.type == selfDrivingCarPos:
        idVehANG = ANGConnVehGetGKSimVehicleId(car.idVeh)
        AKIVehSetAsTracked(car.idVeh)
        parameters = AKIVehTrackedGetStaticInf(idveh)
        initalSpeedAcc = parameters.speedAcceptance
        ANGConnSetAttributeValueDouble(initialSpeedLimitTypeCol, idVehANG, initalSpeedAcc)
    return 0
```

```python
def AAPIExitVehicle(idveh, idsection):
    return 0

def AAPIEnterPedestrian(idPedestrian, originCentroid):
    return 0

def AAPIExitPedestrian(idPedestrian, destinationCentroid):
    return 0

def AAPIEnterVehicleSection(idveh, idsection, atime):
    return 0

def AAPIExitVehicleSection(idveh, idsection, atime):
    return 0
```