Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

Área de Ingeniería Telemática

TRABAJO FIN DE GRADO

Desarrollo de un asistente de conducción basado en las emociones para mejorar la seguridad

(Development of an emotion-based driving assistant to improve safety)

D. Villanueva Fernández, Miguel

TUTOR: D. Corcoba Magaña, Víctor

FECHA: Julio de 2023

Contents

List of Figures

List of Tables

1. Introduction

Technology is advancing at a tremendous pace. It has revolutionize the way we live, work, and interact with the world around us. Rapid advancements across many different fields have driven innovation, transforming industries and shaping the present, and the future, of how society works. In the last decades, we have experienced an unprecedented wave of technological achievements. They range from Artificial Intelligence (AI) and machine learning to virtual reality, block-chain, and the Internet of Things (IoT). Their development pace has been staggering, as no other fields have experienced such exponential growth. These advancements have not only enhanced our capabilities but have also presented new opportunities and challenges.

Figure 1.1.- Examples of emerging technologies [1]

In particular, Artificial Intelligence is considered to be nowadays the spear head of the technological progress. It has enabled the creation of intelligent virtual assistants, personalized recommendation systems, natural language processing, computer vision, and autonomous vehicles, among other breakthroughs. These advancements have significantly changed our daily lives, providing us with smarter services and tools that feel more intuitive, responsive, and immersive. In addition, with machine learning, it is possible to analyze vast amounts of data, recognize patterns, and make intelligent decisions without any kind of human intervention.

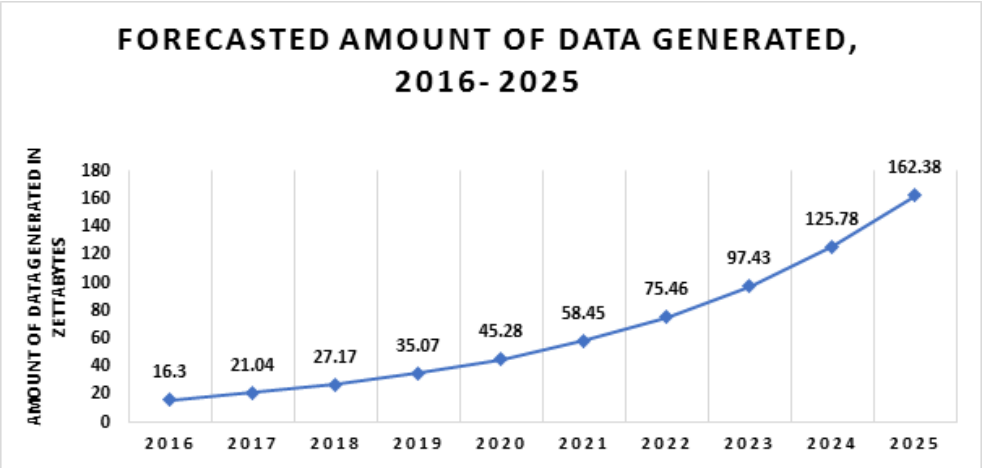


Figure 1.2.- Forecast of total data generation by AI in the world [2]

Behind these actions there are just predictive models, smart algorithms, and autonomous systems that continually evolve and optimize their performance.

But how could the human driver benefit from this advancements in AI? From the help of Advanced Driver Assistance Systems (ADAS). These systems can include a very wide range of features, such as adaptive cruise control, lane-keeping assist, automatic emergency braking, blind-spot monitoring, and parking assistance systems. They can benefit from AI to detect and interpret traffic conditions, road signs, pedestrians, and other vehicles, allowing the vehicle to proactively react to potential hazards or provide helpful guidance to the driver.

The integration of AI and ADAS has the important aim to improve overall road safety, reduce accidents, and complementing the driving experience by trying to minimize human errors. It enables vehicles to analyze and respond to complex situations faster than human reflexes alone, making driving safer and more efficient. Figure 1.3 shows the growing importance of driver monitoring and engagement systems in vehicle safety [3] at Euro NCAP for their recent safety campaigns.

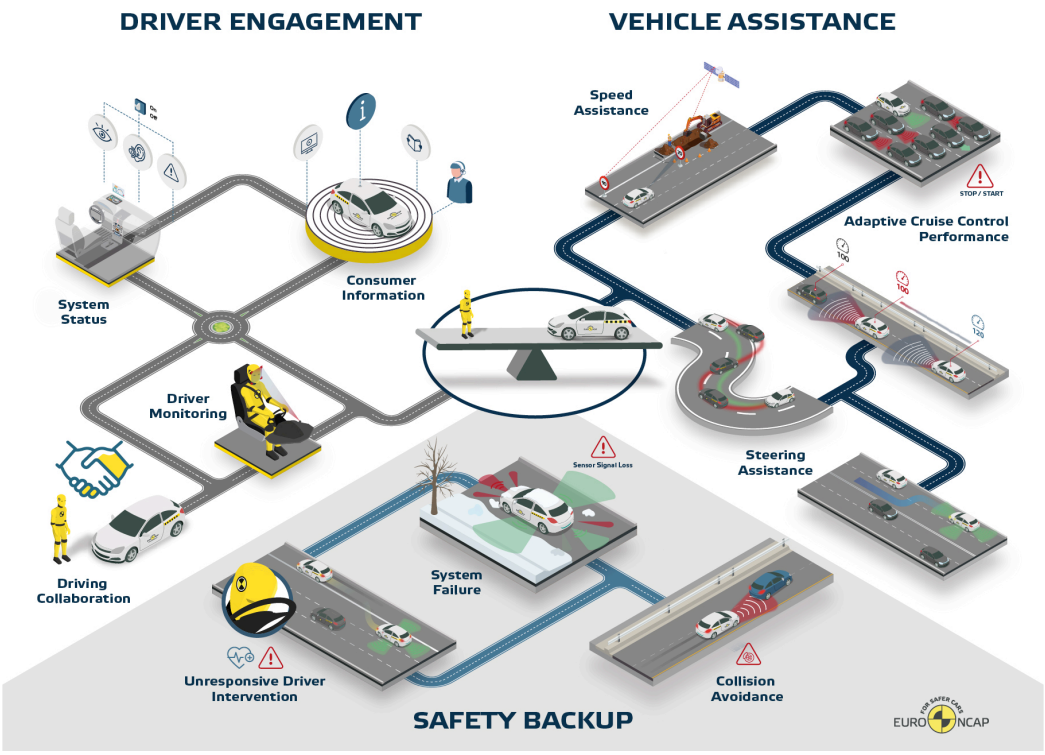


Figure 1.3.- Euro NCAP latest additions to safety assessments [3]

Euro NCAP stands for ”European New Car Assessment Programme”. It is an independent organization that evaluates the safety performance of new car models sold in Europe, although they have initiatives all around the globe. They provide customers with objective information about the safety features that can be found in vehicles currently in the market (with and without optional safety equipment), allowing them to make informed decisions when purchasing a car. By providing independent and standardized safety assessments, Euro NCAP has helped to raise awareness among consumers and contributed to the overall improvement of vehicle safety across Europe. It has become the most influential organization in the automotive industry, driving manufacturers to develop safer vehicles and saving lives on the road. So, it can be concluded that, if ADAS systems are on the spotlight for Euro NCAP, they are very important for road safety.

1.1.- Motivations

Safety, specially on the road, has always been a very serious matter.Traffic accidents have been one of the principal causes of not only deaths, but disabilities and serious injuries. It is estimated that around 1.3 million people die each year as a result of road crashes, and between 20 and 50 million people suffer from non-lethal injuries [4]. Even more worrying is that road traffic accidents are the leading cause of death for children and young adults aged 5-29 years. And although more than 90% of the fatalities around the world happen to be in low to middle income countries, it is still a big problem where we live. Every year, nearly twenty thousand people get killed in the EU. This is specially dramatic between young adults, as ”statistics show that young people are more likely to be involved in a fatal road collision” [5].

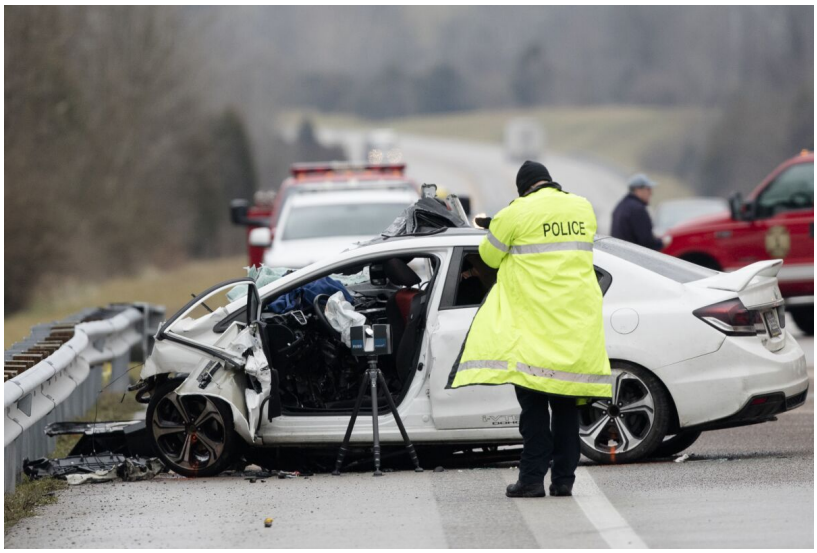


Figure 1.4.- Traffic accidents still take many lives [6]

There are many risk factors that can influence these events, but according to the World Health Organization [4], the main can be categorized as: speeding, driving under the influence of alcohol and other psychoactive substances, lack of use of essential safety equipment, distracted driving, unsafe road infrastructure and vehicles, inadequate post-crash care, and poor traffic laws and their enforcement. Although some of these factors are external, many of them are responsibility of the driver, and can be prevented.More strict law and regulations and road safety education have helped on reducing the mortality numbers, but it has not been enough on its own.

Not only they are the most vulnerable, but studies have also shown that the youngest population (between 10 and 29 years old) is related with risky and aggressive driving behaviour [7]. This driving style is exhibited as speeding, tailgating, abrupt lane changes, or disobeying traffic rules, and it is connected with a higher chance of an accident. The consequences are that it reduces reaction time, awareness and car control, and it can even provoke other drivers. This fact could be linked to many developmental reasons [8], but emotion and self-regulation is key to decrease the associated risk at the wheel [9], as it help reduce this threat.

Figure 1.5.- Aggressive driving manifestations [10]

All these facts and studies led to a great desire of reducing the accidents and mortality on the road, setting the objectives that can be seen in section 1.2.

1.2.- Objectives

The main and goal with this project is to develop an ADAS system that could help to prevent accidents on the road. There are many different ADAS systems present in cars, such as Anti-lock Brake System (ABS) or Adaptive Cruise Control (ACC), and others have been developed

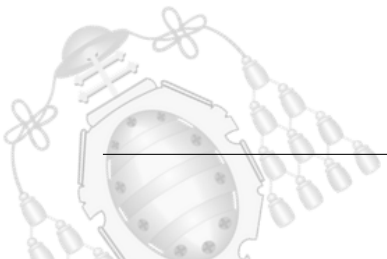
but have not been widely implemented yet, such as drowsiness detection. But in contrast, there has been little to no development of ADAS that consider the effect of emotions while driving, something that has been agreed that it is a very important factor while driving.

In general terms, the aim with the global vision is to create a system that can predict or calculate the likelihood of dangerous situations for every car trip by evaluating the driving patterns and the emotional state of the driver. Data is gathered from a lot of different sensors in the car to enhance the accuracy of the system, and an already developed system will be in charge of assessing the person's emotion. Once there is some certainty about the aggressiveness of the driving style, the system should counteract it by employing some kind of relaxing method to calm down the driver. These methods should try to avoid distracting him or worsening the situation by possibly disturbing him further.

Some specific objectives can be defined:

- Learn about new technologies for ADAS development.
- Learn about driving simulators for development purposes.
- Learn about distributed systems and software integration.
- Learn about 5G and its benefits over traditional wireless networks on vehicle communication.
- Learn about Python and AI techniques for software development.
- Create an ADAS that takes into account the driver emotion.
- Integrate all sensors and actuators into a distributed system.
- Develop a testbed for ADAS testing in a simulator.
- Test the whole system over a 5G network.

Consequently of the carried out work, it could serve as a test bed for the experimentation and further experimentation of new technologies that could benefit from the development of new ADAS, or the evaluation of people's emotions into other projects.



2. State of the art

First of all, a deep dive into the state of ADAS systems and emotional states while driving is needed.

The publicly available study by MIT in 2020 is a really good starting point, as it offers an overview of the research efforts and advancements in the field of driver emotion recognition for developing intelligent vehicles [11]. The primary objective of the study is to explore how emotions can be detected and analyzed to enhance the performance and safety of intelligent vehicles, so new developers and researchers can have a starting point. This is because there are many aspects and challenges associated with detecting and recognizing emotions accurately, that could become challenging at first.

The survey covers different types of sensors used for emotion detection, such as facial expression analysis, physiological sensors (e.g., heart rate monitors, electroencephalography), and voice analysis. It digs into the advantages, limitations, and challenges associated with each sensor type. Furthermore, the authors highlight the significance of data collection, as it is key for developing robust emotion recognition systems. They also discuss the potential applications of driver emotion recognition in intelligent vehicles, as they could be in form of an ADAS, personalized user experiences, detection of dangerous driving behaviours, or even for future self-driving car features.

This and many other studies have linked emotions with driving behaviour in a lot of different ways. Some examples are included in the following studies by researchers from many different universities and experts: [8], where they link self-regulation and risky driving behaviour; [13], where they investigated the potential contribution of sensation seeking, impulsiveness, and boredom proneness to driving anger in the prediction of aggressive and risky driving; [14], where they investigate the personality traits behind the risky and aggressive driving of young adults; [15], where they evaluated the effect of the emotional state on simulated driving behavior; [16], where they researched the impact of emotion, life stress and mental health issues on

Figure 2.1.- Many experimentation is done in simulators [12]

driving performance and safety; and [17], where they explored the relationships among emotional intelligence, emotional regulation, driving anger and related behaviour.

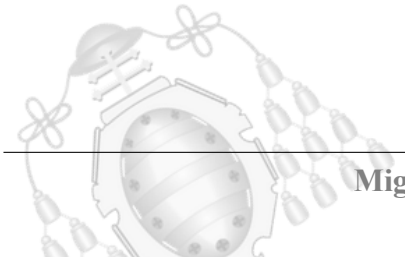
But despite the efforts, very few developed ADAS systems have included driver emotion recognition as a variable to monitor and act based upon it. Some developers have already developed systems that detect emotions in driving environments, such as [18], where they develop an AI that identifies driver emotions, [19], where they develop an AI to detect driver stress, and [20], where they develop a classification system that identifies driver emotions and behaviour for driving applications. But still, in none of them they used those emotions to prevent accidents by influencing the driver in some way.

One of the closest examples to it is the approach seen in this study [21], in which they proved that both driving performance and emotion parameters are helpful while evaluating the driver's state. They also tried to use acoustic stimuli in order to counteract bad driving performance, but did not get very relevant results.

In another study [12], the researchers used emotion detection to change ambient lights in the car as a way of feedback. Not only they could classify possibly risky emotions, such as anger or

arousal, but they could also help the driver react to them by providing a way of letting him know in a non-disturbing way. Thanks to it, lane keeping abilities were demonstrated to be improved during testing. But there are other additional countermeasures that could be implemented.

Something to be said is that simulators have been widely used to test driving patterns and ADAS, as seen in these studies [22] [23] [21]. They are not only much safer and economical than real driving testing, but they are still a very relevant and accurate representation of how drivers would act in a real world scenario.



3. Methodology

With the previous review in mind, the task of determining what the system has to offer becomes simpler. Every software project can be divided into three main parts: Data Input or Acquisition, Data Processing and Data Output. A fourth one can be considered in case the system has some kind of feedback.

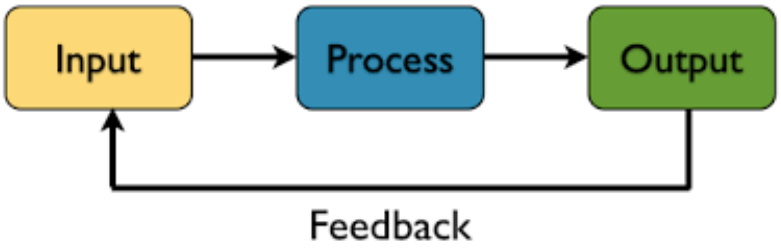


Figure 3.1.- Parts in a software system [24]

To start, the system will need data inputs, so that it has something to actually process. This way, the user or the environment can interact with the it. These inputs are usually in the form of sensor or network data. In this specific case, a considerable amount of input data is needed, as this information has to be enough in order to detect dangerous situations. In addition, this part may require the use of networks and communication protocols if the data does not come from the same computer executing the code. All these problems are described in the Data acquisition section 3.1.

Once all the required data is received, the system has to process it. This is not an easy task in this project, as it has to judge the contribution of each input, and take a decision on whether there could be a dangerous situation in real time. For this reason, a computer is needed. Further stress on this part is made in the Data processing section 3.2.

Finally, the system will need to act accordingly in order to try to avoid those dangerous situations. This part is addressed in the last section (3.3).

3.1.- Data acquisition

There are infinite ways to acquire data, but only limited resources to process it. This fact is specially important in cars, which are really limited on space, weight, budget and power consumption. This is the reason why there needs to be a limit on the amount of data that the system is going to acquire and process. Therefore, careful selection of these inputs is essential. But before deciding on which ones to take, a general understanding on what the system is expected to detect is needed.

Figure 3.2 shows the scheme of a typical data acquisition system (DAQ) of a racing car, in which many variables are measured. The aim of this project is very different from racing, but it is still useful to learn about it, as in both applications monitoring the driver is key. The transponder and a lot of general information about the car, such as engine and suspension state, is not very relevant, because it relates more to the maintenance of the car rather than to the driving attitude. Meanwhile, driver input in the form of wheel and pedal control and/or vehicle accelerations are directly related to how aggressive the driver is. A camera and some additional sensors could also be helpful at tracking the state of the person, as it is covered later.

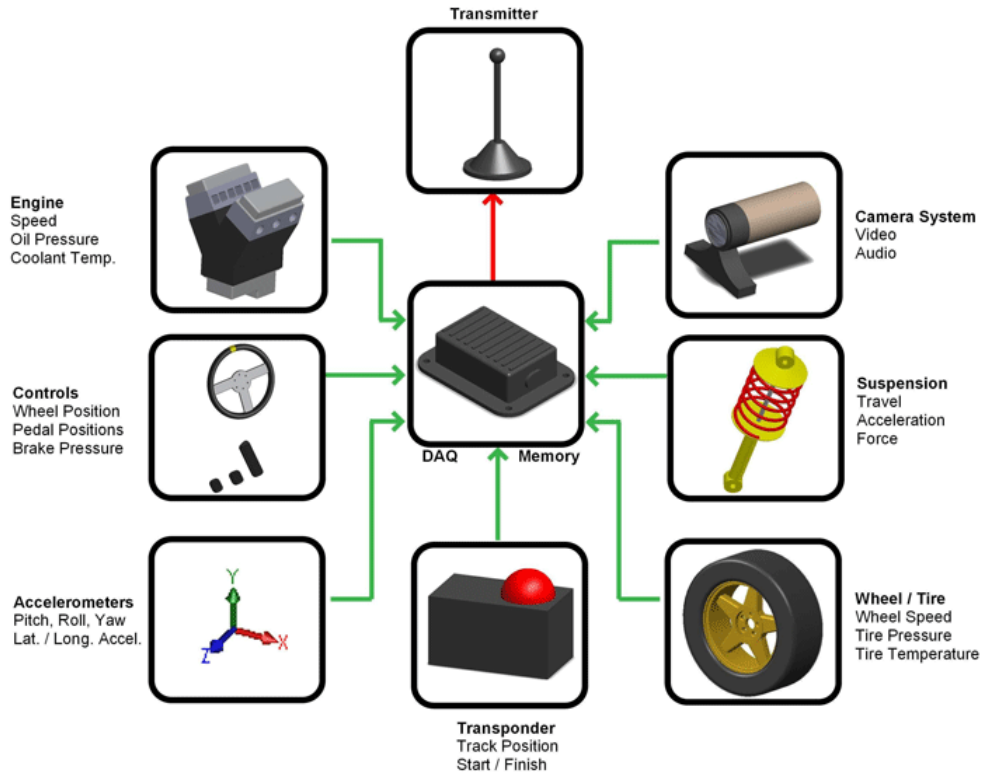


Figure 3.2.- Typical data acquisition system in a car[25]

3.1.1.- Aggressive driving and accident risk

Sensors are the system's input, but as advanced as they could be, they will not be able to directly give a measure of safety. But evaluating the driving patters through these sensors could yield to valuable information. Many studies have shown that aggressive driving is not only supposedly related to an increase in accident risk, but directly linked. Both an aggressive style of driving and impulsive personality traits are good predictors of traffic accidents [26], even if it is a professional driver or not. So we can determine that studying the aggressiveness of the person behind the wheel is a great way to evaluate accident risk.

3.1.2.- Driving behaviour and accelerations

Study [27] demonstrates that this aggressive driving behaviour can be measured as increased variability in speed and accelerations, and higher overall speed (usually over the speed limit).

This is an easy parameter to measure with an accelerometer sensor that can precisely measure the accelerations inside the car.

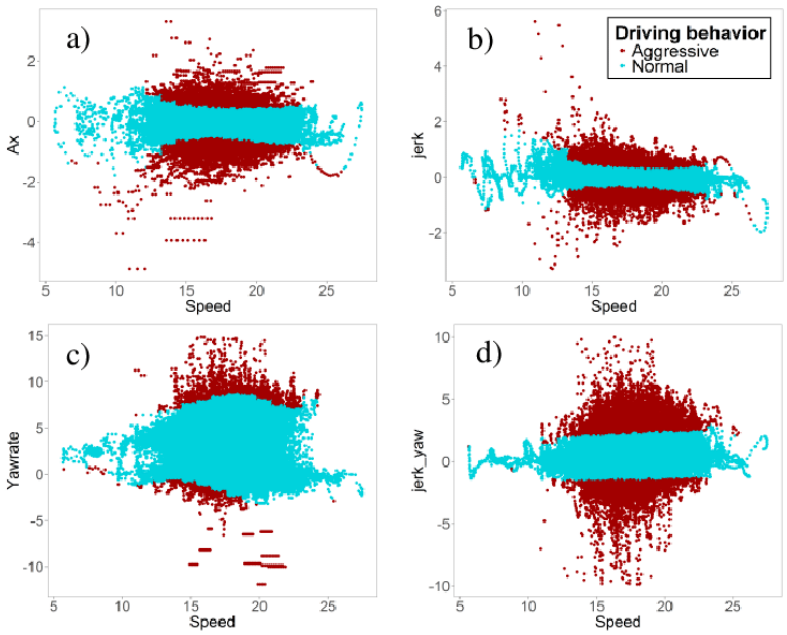


Figure 3.3.- Classification of aggressive and normal driving based on a) longitudinal acceleration, b) longitudinal jerk, c) yaw rate, and d) angular jerk [28]

3.1.3.- Emotions and driving behaviour

As explained before, emotions are linked to the driving behaviour. A driver is usually less focused on the activity at hand if he experiences strong emotions, specially anger. This type of conduits are related with distractions and road rage, which can lead into accidents. Having a way to predict the driver's emotion is a key factor in order to detect these dangerous situations.

That is the reason why Ignacio's software, which will be evaluated in the implementation section, can be an important piece of this system. The basic principle of this project is to predict the emotion of a person by interpreting their face gesture through a webcam. Such achievement was done by developing an artificial intelligence trained for this specific objective with an extensive database of face expressions. A few different algorithms were used in the process, and they were evaluated afterwards to check how precise they were. Near a 85% of accuracy was

achieved, a really good value considering the intrinsic ambiguity between face expressions and emotions.

3.1.4.- Heart rate data

As described in a study about emotions and cardiovascular variables [29], strong emotions such as anger can lead to a noticeable difference in this parameters. Probably, the best one to look at is heart-rate, as it is proven to rise as the observed person gets angrier. In addition, heart-rate is an easy variable to monitor, as there are many sensors that can accurately sense it without being intrusive or even uncomfortable. That is why the use of heart rate monitoring device, such as a cardiac band, could be helpful during the evaluation of possible risky situations while driving.

3.1.5.- Driving behaviour and noise

Higher sound levels than usual can be traced to a more aggressive style of driving [30]. This condition is due to cars emitting more noise when driven hard, as both engine and brakes cope with higher loads. A noise monitoring device could be helpful on evaluating this phenomenon. In addition, it is an easy to monitor variable, as these types of sensors are affordable and work well [31]. The only downside is that each car and road generates a different noise, so this parameter has to be evaluated dynamically and individually of each situation.

Figure 3.4.- Yelling angry driver[32]

3.1.6.- Data communication

Between the acquired data and the processing device there is a communication channel. If both are done under the same computer, it is just a matter of communicating processes or pipelines, but because the could be on different computers, a better solution is to use sockets and network protocols in order to communicate the sensors with the system. In that case, there are many possible networks that could be used, but as it could possibly be used in a car, it should be a wireless network with good coverage. Although 3G/4G mobile networks, or even LoRa could suit the project, the most beneficial network for this specific application is 5G.

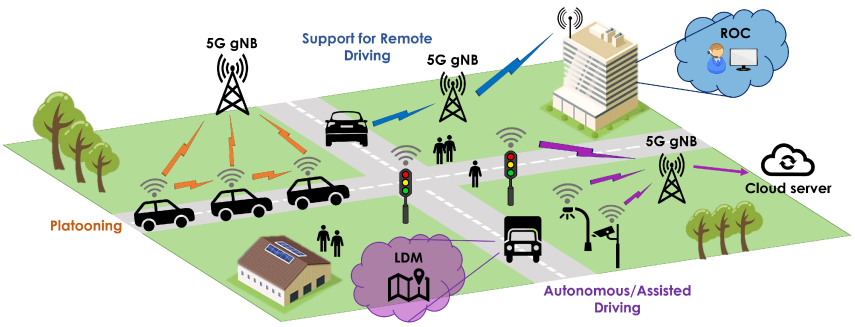


Figure 3.5.- New possible 5G use cases[33]

As discussed in many projects, 5G is becoming the new standard for cars, as it can enable a lot of new use cases with its low latency, high bandwidth and massive user count. This characteristics could enable new use cases, such as vehicle to vehicle reporting to avoid collisions, vehicle to infrastructure communication in order to instantly know about the traffic state, and even remote driving [34] [35] [36] [37].

3.2.- Data processing

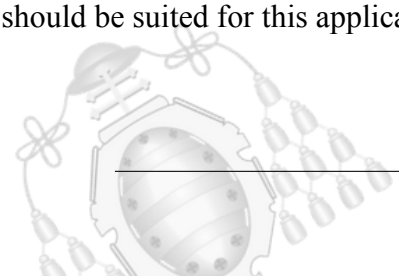
Data acquisition systems just gather data, but driver assistants help drivers become safer and better through algorithms that process live data. All collected statistics are interpreted and analyzed to aid the decision-making of the driver and avoid possible collisions. This task is usually done by an artificial intelligence, either onboard the vehicle or remotely. It is preferable to do it locally, as less latency is involved. However, this condition could limit the processing

capability of the ADAS, as power consumption has to be as low as possible in vehicles, therefore making very limited the computational power.

There are several types of algorithms for data processing in AI.The simplest one is a rule based system. This type of systems use a Knowledge Base, in which strict rules are defined (usually ”if” statements). These conditions are constantly evaluated in the Interference Engine, by constantly checking the Working Memory for new data. Once a rule is triggered, the output is also stored in the Working Memory in order to feedback the system. Being the simplest, it is really low on resources and requires little knowledge and effort to implement. The trade back is that you have to have a really good understanding on what exactly is going to enter the system and what do you want out of it. Rules have to be strictly defined, and therefore the output is a strict output, in which one or many of the rules are triggered observing the inputs and previous outputs.

On the other end of the spectrum, there are advanced artificial intelligence systems with neural networks that aid them at tasks like object and human recognition and image analysis. In addition, they can implement automated learning to help them get better at their task, and combine it with speech recognition, just like Siri or Google assistants can do. But these methods are implemented on big servers that are accessed through online services. As permanent and stable Internet connection in a car is still a very difficult task (until 5G becomes readily available), it is better to go for a less resource hog algorithm that can be implemented inside the car.

Somewhere in the middle lies a fuzzy logic algorithm. This is a branch of artificial intelligence that can process uncertain and inaccurate information. Although it is based in rules for decision-making, these rules are usually defined in vague terms to allow flexibility and adaptability. These characteristics allow to easily adapt and calibrate the system to work in different situations. Besides, the system is better suited at managing uncertainty, as it uses continuous values instead of binary values when representing variables. But in contrast to neural networks, fuzzy logic systems offer a more intuitive and understandable approach to the resulting knowledge, thanks to its simple way of defining rules and inputs. These are the reasons why this processing algorithm should be suited for this application, and has been for many projects [38] [39] [40].



3.3.- Data output

Once the system is able to gather and process enough data to determine a dangerous driving situation, the system has to have an outcome. The ideal situation would be to act and be able to calm down the driver, thereby reducing the driving aggressiveness and helping lower the risk of an accident. This is not a simple task, and there are many different ways to try to calm down someone, but as it will be seen, not all are suitable for a driving situation, as they could distract or interfere the driver.

3.3.1.- Visual indicator

As previously discussed in 2, some kind of visual indication can be helpful on warning the driver about its dangerous driving [12]. In that experiment, they filled the cabin with RGB LED lights that changed colour depending on the perceived emotion, but as long as the driver has a clear visual sign, it doesn't need to be so huge (although it can contribute to making him more aware).

Figure 3.6.- Cabin lights from the experiment[12]

3.3.2.- Music

On the one hand, music listening is proven to be a soothing and relaxing experience [41] [42]. It can calm the mind, reduce stress, and promote a sense of tranquility. Some examples are classical music, due to its calming instrumentation and steady rhythm, ambient music used for meditation and yoga, and even electronic music, due to its repetitive patterns. In addition, if the driver is enjoying the music, he is much more likely to exhibit a good mood and make the driving task less tedious.

Figure 3.7.- Yoga classes use music to help people relax[43]

On the other hand, music is known to be a potential distraction [30]. This is both in the case the driver wants to change song or adjust a parameter, or if the music is suddenly triggered without any human intervention, it can cause an accident by loosing attention from the road. Furthermore, high volume can impair the ability to notice important event while driving, like the siren of an emergency vehicle, car sounds (like the engine or wind sound, which can result in speeding), or information coming from the road and cars nearby.Finally, personal preferences and content choices can disagree with what the driver wants, which could lead to the opposite effect of creating tension.

With the acquired information, it is recommended that the driver listens to music of his liking that doesn't provoke strong emotional responses. As long as it doesn't distract him from the task of driving by using voice controls and keeping a moderate volume, it should be fine. But unless the ADAS system has information about all these details of the person's music taste, it is better to leave this option to the driver.

3.3.3.- Verbal de-escalation

For more than a decade, verbal de-escalation is the first and most common technique for dealing with agitated patients in hospitals [44]. Not only it is much less intrusive than routine restraints or involuntary medication, but if undertaken with genuine commitment, successful outcomes are very likely. These methods have been studied for nearly fifty years [45], and although they rely on the premise of a doctor speaking to a patient, many concepts have been adapted to work in

situations in which someone needs to calm down an agitated person. For example, de-escalation techniques have been really useful for bus drivers to deal with obnoxious and even dangerous passengers.

Figure 3.8.- Doctors normally use verbal de-escalation when dealing with agitated patients[46]

These techniques are very well known and effective. In consequence, many systems have implemented them. As an example, a few simulators have been developed so that health professionals can train their de-escalation skills without having to test on real patients [47]. One even makes us of the latest technologies, such as Virtual Reality, in order to help improve the reality of the simulation [48].

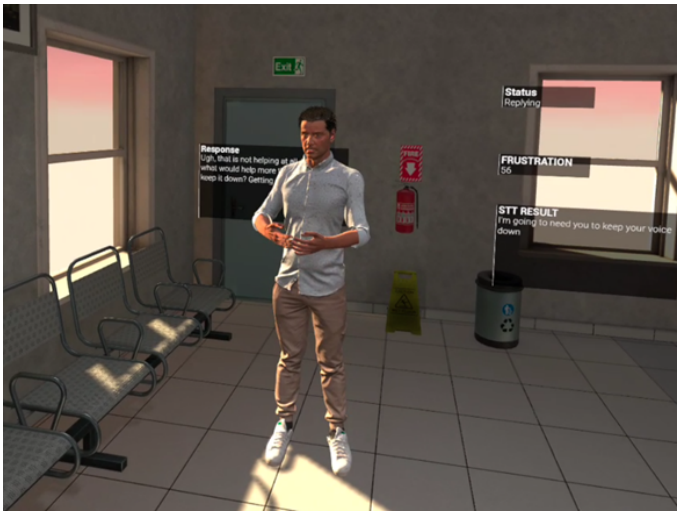


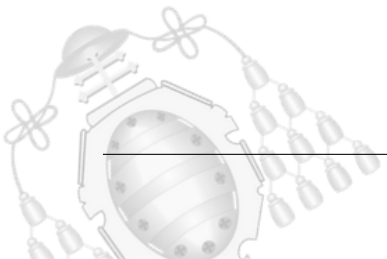
Figure 3.9.- VR patient de-escalation simulator[48]

This use case is a bit different, as it will be a virtual assistant the one trying to de-escalate the agitated driver. Although there are a few techniques that can not be implemented due to not being a real person, many others some others should still be helpful. The most relevant gathered from the assessed studies are:

- Be careful with the voice tone. Try to be as calm, empathetic and soothing as possible to help create a feeling of reassurance and calmness for the listener.
- Be clear and concise. Use simple and short sentences that make your message easy to understand. Avoid using complex, detailed or technical language that could confuse the person.
- Use positive reinforcement. Try to end messages on a positive note, and to express gratitude for the driver's cooperation and patience.
- Inform the user. Provide relevant information about the issue, helpful details to understand it and instructions to assist the listener.
- Acknowledge emotions. Try to recognize and validate the emotions the driver might be experiencing by using empathetic messages.
- Be repetitive with important messages. In order to avoid misunderstandings, repeat critical details. It also helps to get the message across.
- Offer solutions or alternatives. Help the driver to solve the issue by providing useful tips and options.

3.3.4.- Accident awareness

It may seem obvious, but conscious drivers that have been warned about the risk of driving take less risks while driving [49]. This study supports the rationality of using traffic accident materials to conduct safety education for drivers, and also discusses the significance of traffic-related negative emotions to social security.



4. Requirements Engineering

First of all, in every software project, the developer must determine what the user or the client needs. This is a vital step to ensure that the programmer can confidently create the system, knowing all the desired features are implemented. All stakeholders could have a different idea about which should they be, but it is essential to gather all of them in a clear and direct manner. This is the only way that, when the development begins, all parts of the system could be clearly defined. It is not an easy task, and usually involves many methods of elicitation, like interviews, questionnaires, use cases and even role playing. It also may require many iterations, as desired functionalities or specific priorities can change along the development process.

Once these features are clear, the developer should create a document to describe the requirements needed to implement the whole project, and their respective priority or importance. This process of discovery, analysis and documentation of the desired services and requirements is called Requirements Engineering, and it is a vital step in order to develop a software application. If these steps are not followed, it is very likely to fail at delivering what the client wants, specially as a project gets bigger.

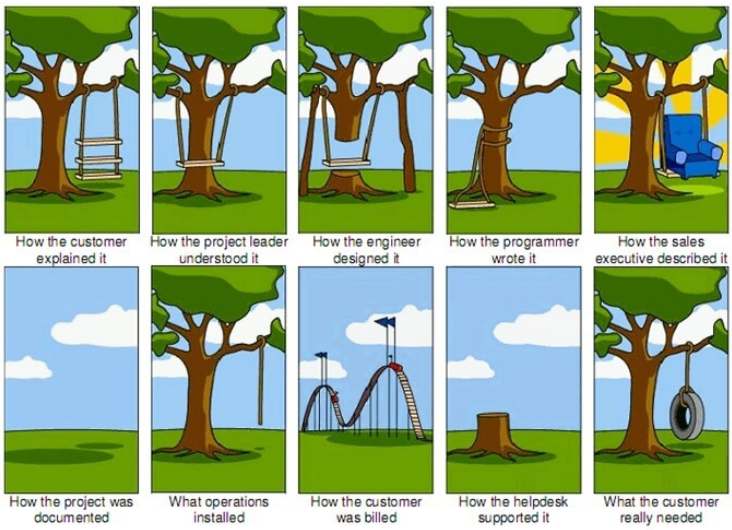


Figure 4.1.- Common mistakes while evaluating system requirements [48]

The result of this requirement gathering is usually a document with use cases, functional requirements and non functional requirements. Use cases describe situations in which a user interacts with the system. They are part of the initial requirements analysis. Meanwhile, functional and non functional requirements are written in the requirement specification part. Functional requirements define what exactly the application should do and how it should react to user interaction in each possible situation. Non functional requirements, on the other hand, describe system restrictions. These restrictions can be based on time, laws, size, ease of use, reliability, security, portability, power consumption, and many others depending on the application.

4.1.- Use Cases

This analysis led to defining some use cases for the system, which can be linked to later defined Functional Requirements.

1. Retrieve the driver's emotion
2. Retrieve the driver heart rate
3. Retrieve acceleration values
4. Launch the driving simulator server
5. Change simulation map
6. Add traffic to the simulation
7. Launch the driving simulator client
8. Change simulation vehicle
9. Change simulation driver POV (Point Of View)
10. Evaluate the accident risk involved
11. De-escalation audio message display
12. Visual danger pilot

Here they are illustrated in an interaction UML diagram.

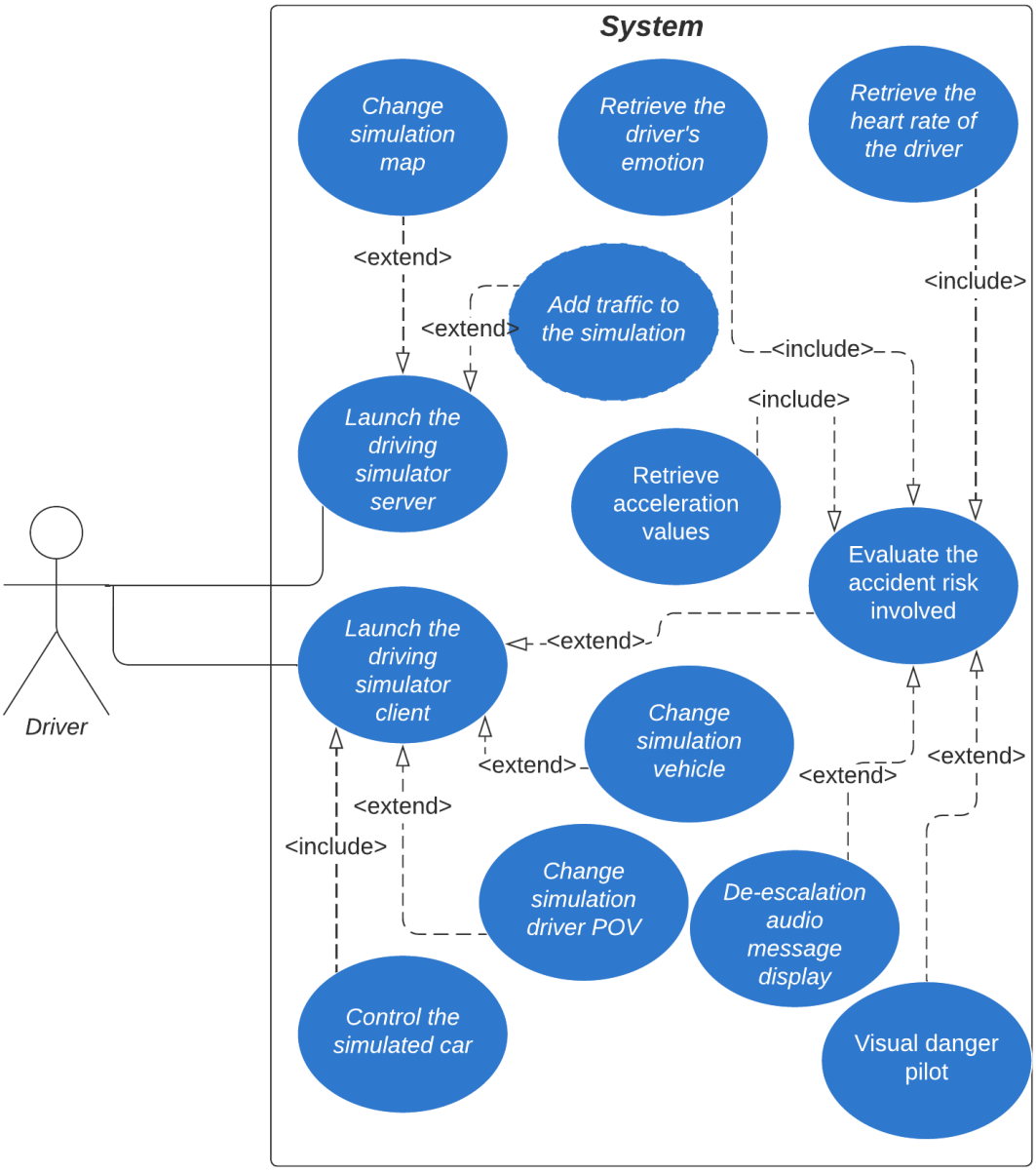
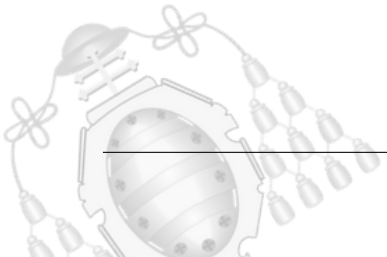


Figure 4.2.- Use cases diagram [24]

Next, all use cases are extended in detail one by one with a table.

ID	UC01
Title	Retrieve the driver's emotion
Goal	To detect if the driver is angry
Preconditions	The camera setup is running and connected to the message broker, and the driver is in the camera field of view
Postconditions	Messages with the actual detected emotion are sent
Actors	User, camera
Step description	1. The camera application launches 2. It detects the driver's face and its emotion 3. It sends the estimated emotion to the message broker
Exceptions	1. The program coudln't launch • No broker connection, check broker IP and internet connection and restart the program (step 1) • The camera is not connected, connect the camera and restart the program (step 1) 2. The driver is not in the camera field of view • The program won't send any emotion, focus the camera (step 2)

Table 4.1.- UC01: Retrieve the driver's emotion from the camera



ID	UC02
Title	Retrieve the heart rate of the driver
Goal	To detect a high heart rate level
Preconditions	The cardiac band setup is working, strapped to the driver and connected to the message broker
Postconditions	Messages with the heart rate are sent
Actors	User, cardiac band
Step description	1. The heart rate application connects to the heart rate band 2. It detects the driver's pulse 3. It sends the estimated heart rate to the message broker
Exceptions	1. Not detecting the heart rate band: driver is too far, restart the program (step 1) 3. No message sent to the broker: couldn't connect to the broker, restart the program (step 1)

Table 4.2.- UC02: Retrieve the driver's heart rate

ID	UC03
Title	Retrieve acceleration values
Goal	To detect sudden accelerations
Preconditions	The acceleration sensor setup is working (either the simulator or the real setup) and connected to the message broker
Postconditions	Messages with the acceleration values are sent
Actors	User, accelerometer
Step description	1. The accelerometer detects the car's acceleration values 2. It sends the estimated acceleration values to the message broker
Exceptions	2. No message sent to the broker: couldn't connect to the broker, restart the program (step 1)

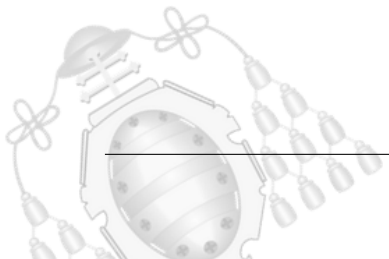
Table 4.3.- UC03: Retrieve the acceleration values

ID	UC04
Title	Launch the driving simulation server
Goal	To create a realistic computer driving simulation environment for system testing
Preconditions	The computer is configured for the simulation environment server (CARLA is installed)
Postconditions	The simulation server is running and a window with the map view appears
Actors	User, simulator
Step description	1. The simulation environment server launches 2. The server starts with a generic map 3. The server starts without traffic
Exceptions	-

Table 4.4.- UC04: Launch the driving simulation server

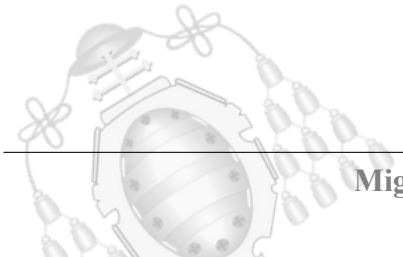
ID	UC05
Title	Change simulation map
Goal	To change to a different driving environment
Preconditions	The simulation server is running
Postconditions	The simulated map is changed
Actors	User, simulator
Step description	1. The user executes a file that connects to the simulation server 2. A different map appears on the server
Exceptions	1. No connection to the server can be established: check IP server address and restart app (step 1)

Table 4.5.- UC05: Change simulation map



ID	UC06
Title	Add traffic to the simulation
Goal	To change the cars driving autonomously inside the simulation
Preconditions	The simulation server is running
Postconditions	The traffic is changed
Actors	User, simulator
Step description	1. The user executes a file that connects to the simulation server 2. A different amount of traffic appears on the server
Exceptions	1. No connection to the server can be established: check IP server address and restart app (step 1)

Table 4.6.- UC06: Change simulation traffic



ID	UC07
Title	Launch the driving simulator client
Goal	To drive inside the simulation with steering wheel and pedals
Preconditions	The steering wheel setup and simulation server working, and having installed all dependencies
Postconditions	The simulator client appears on a new window
Actors	User, simulator, steering wheel
Step description	1. The simulation environment client is launched 2. A new window is created with the car view and the HUD 3. The simulation sends the estimated acceleration values to the message broker
Exceptions	1. Can't establish a connection to the simulation server: check IP server address and restart app (step 1) 2. No message sent to the broker: check IP broker address and restart app (step 1) 3. Not having the steering wheel connected: connect the steering wheel and restart app (step 1)

Table 4.7.- UC07: Launch the driving simulator client

ID	UC08
Title	Change simulation vehicle
Goal	To change to a different simulated vehicle
Preconditions	The steering wheel setup and simulator running
Postconditions	The simulated car is changed
Actors	User, simulator
Step description	1. The user presses a key 2. A new car is spawned
Exceptions	-

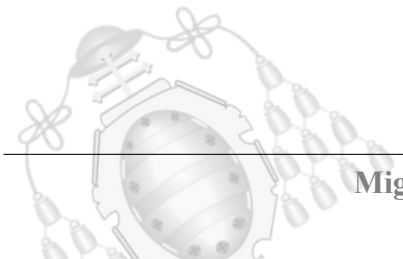
Table 4.8.- UC08: Change simulation vehicle

ID	UC09
Title	Change simulation driver POV
Goal	To change to a different camera Point Of View inside the simulation
Preconditions	The steering wheel setup and simulator running
Postconditions	The simulated driver POV is changed
Actors	User, simulator
Step description	1. The user presses a key 2. A new camera view is seen on screen
Exceptions	-

Table 4.9.- UC09: Change simulation driver POV

ID	UC10
Title	Evaluate the accident risk involved
Goal	To get a percentage value of the risk danger
Preconditions	Having received all data
Postconditions	A percentage value of a dangerous situation likelihood
Actors	ADAS
Step description	1. All sensor values are received 2. These values are checked with the defined rules 3. The evaluation gives out a risk percentage
Exceptions	1. The program can't connect to the MQTT broker: check IP server address and restart the app (step 1)

Table 4.10.- UC10: Evaluate the accident risk involved



ID	UC11
Title	De-escalation audio message display
Goal	To calm down the driver and reduce the risk percentage
Preconditions	Having a risk percentage value above a threshold
Postconditions	An audio message is played
Actors	User, audio system
Step description	1. The threshold condition is activated 2. Play a de-escalating message 3. Wait for a time interval 3.1. If the percentage is the same or higher, go back to point 2 3.2. If the percentage has lowered, continue to point 4 4. Deactivate threshold condition
Exceptions	1. Not being able to connect to the MQTT broker: check IP server address and restart the app (step 1) 2. Not being able to listen to the audio message when the condition is triggered: check audio connection (step 2)

Table 4.11.- UC11: De-escalation message display

ID	UC12
Title	Visual danger pilot
Goal	To inform the driver of a dangerous situation
Preconditions	Having processed all data and detected aggressive driving, and the simulation client running
Postconditions	A pilot light is triggered in the simulator HUD
Actors	User, simulator
Step description	1. The threshold condition is activated 2. A message is sent to the simulator 3. The received message triggers an indicator 4. The pilot will stay on till a deactivated condition message is received
Exceptions	-

Table 4.12.- UC12: Visual danger pilot

4.2.- Requirements Specification: Functional Requirements

After making a preliminary study and analysis of how the desired system should work, a list of Functional Requirements and Non Functional Requirements can be described.

ID	Priority	Description
FR01	Medium	The system should collect the driver's heart rate values
FR02	High	The system should assess the driver's emotion
FR03	High	The system should estimate the acceleration values of the car
FR04	High	The system should offer a car driving simulation environment
FR04	Low	The user should be able to change the simulation map
FR04	Low	The user should be able to change the simulation traffic
FR04	Low	The user should be able to change the controlled car
FR04	Low	The user should be able to change the camera point of view
FR05	High	The system should calculate the probability of a dangerous driving situation considering all collected data
FR06	High	The system should verbally warn the user when a dangerous driving situation is likely
FR07	Low	The system should optically warn the user when a dangerous driving situation is likely
FR08	Medium	The system should try to verbally de-escalate an agitated or angry driver

Table 4.13.- Functional Requirements

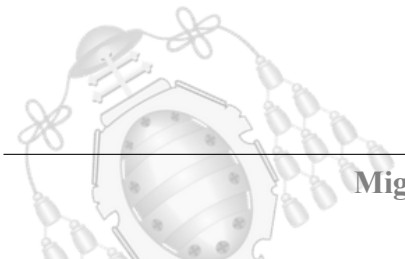
4.3.- Requirements Specification: Non Functional Requirements

ID	Priority	Description
NFR01	High	The system should be responsive enough. It can take some time to evaluate the driving patterns, but not too much so that the alerts are still relevant. The maximum allowed time for this task should be 60 seconds.
NFR02	Medium	The system should use Python as the programming language
NFR03	Low	The system should run on a low power Single Board Computer such as the Raspberry Pi 4
NFR04	High	The system should be suited and compatible to evaluate both simulated and real driving
NFR05	Medium	The audio messages from the system must be in English
NFR06	High	The system must be secure, so that both confidentiality, integrity and availability are maintained

Table 4.14.- Non Functional Requirements

4.4.- Development Steps Specification

Once all Functional Requirements and Non Functional Requirements have been described, the Software Developer can determine a series of steps to precisely implement them in code. These items are gathered from the step sequence described in each use case.



4.4.1.- Simulation

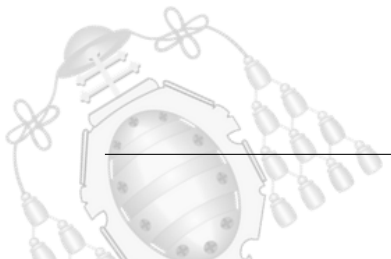
ID	Priority	Description
S-01	High	To manually test the system without a real car, a simulation must be available
S-02	High	The simulation must ensure a realistic car driving environment with steering wheel and pedal integration
S-03	Low	The simulation should have a way of adding traffic
S-04	Medium	The simulation should have a way of changing the simulation map
S-05	High	The simulation should support steering wheel and pedals input
S-06	Low	The simulation should offer a way of changing the vehicle
S-07	Low	The simulation should offer a way of changing the camera point of view

Table 4.15.- S: Simulation development steps

4.4.2.- Acceleration measurements

ID	Priority	Description
A-01	High	The system should get acceleration values from the real car or the simulation
A-02	High	The system should send the acceleration values to the message broker
A-03	High	The system should take into account the acceleration values in the FL algorithm

Table 4.16.- A: Acceleration development steps



4.4.3.- Emotion detection

ID	Priority	Description
E-01	High	The system should detect the driver's emotion
E-02	High	The system should send the driver's emotion to the message broker
E-03	Medium	The system should keep a timeline of the driver's emotion
E-04	High	The system should take the driver's emotion into account in the FL algorithm.

Table 4.17.- FRE: Emotion development steps

4.4.4.- Heart rate

ID	Priority	Description
H-01	High	The system should get heart rate values from a cardiac band
H-02	High	The system should send the heart rate values to the message broker
H-03	High	The system should take the heart rate into account in the FL algorithm

Table 4.18.- H: Heart rate development steps

4.4.5.- Message broker

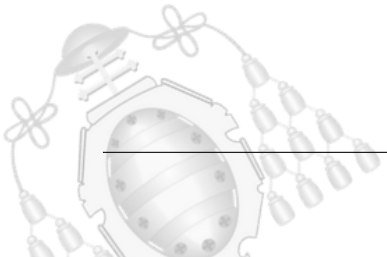
ID	Priority	Description
M-01	High	The system should receive all the messages from the sensors
M-02	High	The system should send the received values to the risk evaluation system
M-03	Low	The system should use 5G to route data

Table 4.19.- M: Message broker development steps

4.4.6.- Risk evaluation

ID	Priority	Description
R-01	High	The system should calculate a dangerous driving likelihood value with the given data using a FL algorithm
R-02	High	This dangerous likelihood has a numeric value between 0% and 100% (a percentage), with 0% being the lowest likelihood and 100% being the highest
R-03	High	The system should determine dangerous driving situations. A likelihood value over a certain limit should trigger this condition (f.ex. 50%)
R-04	High	If the driver's emotion is mostly "Angry", the likelihood value should increase
R-05	High	If the driver's emotion is mostly not "Angry", the likelihood value should decrease
R-06	High	If the driver's heart rate increases, the likelihood value should increase
R-07	High	If the driver's heart rate decreases, the likelihood value should decrease
R-08	High	If the driver's acceleration values increase, the likelihood value should increase
R-09	High	If the driver's acceleration values decrease, the likelihood value should decrease

Table 4.20.- R: Risk evaluation development steps



4.4.7.- System output

ID	Priority	Description
O-01	Medium	The system should warn the driver of a dangerous driving situation via relaxing audio messages
O-02	Medium	The system should continue warning the driver if the likelihood keeps rising or does not decrease over a period of time
O-03	Low	The system should reward the driver if the dangerous driving likelihood decrements below the maximum level achieved
O-04	Low	The system should warn the driver of a dangerous driving situation with a LED light or indicator in the HUD simulation when a threshold is reached
O-05	Low	The system should stop warning the driver with a LED light or indicator in the HUD simulation when it is below a threshold

Table 4.21.- O: Output development steps

5. Planning

5.1.- General Planning

With all the previous considerations in mind, a general planning of the project can be better done.

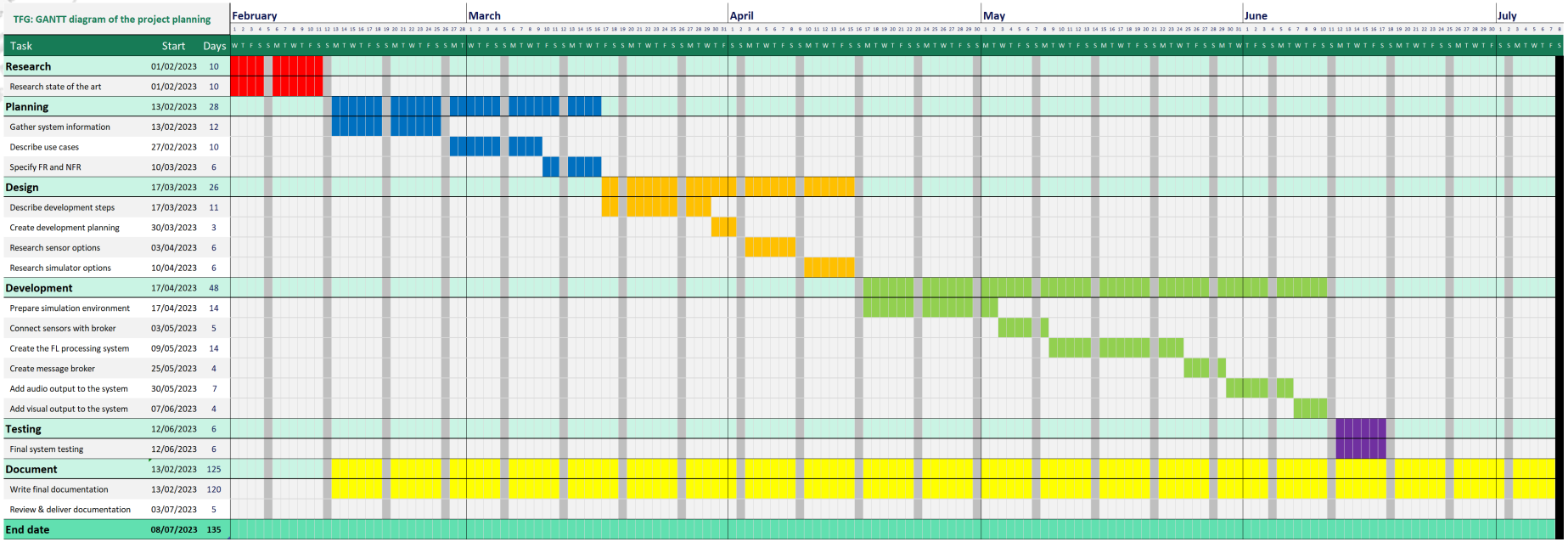
Table 5.1 shows the resulting planning of the project.

Nº	Stage	Task
1	Research	Gather information about the state of the art
2	Planning	Gather information about the desired system
		Describe use cases of the system
3	Design	Describe requisites of the system
		Research sensor and simulator options
		Create a development planning
4	Development	Elaborate simulation environment
		Connect the sensors to a message broker
		Create the Fuzzy Logic processing system
		Create the message broker
		Add audio output to the system
5	Testing	Final testing
6	Document	Write the final year project documentation

Table 5.1.- Project planning

5.2.- GANTT chart

The proposed GANTT chart appears below 5.1, with a total of 135 days. Work days go from Monday to Saturday every week, and it is expected to be two hours a day, making it a total of 270 hours for the whole project.



Task	Start	Days
Research	01/02/2023	10
Research state of the art	01/02/2023	10
Planning	13/02/2023	28
Gather system information	13/02/2023	12
Describe use cases	27/02/2023	10
Specify FR and NFR	10/03/2023	6
Design	17/03/2023	26
Describe development steps	17/03/2023	11
Create development planning	30/03/2023	3
Research sensor options	03/04/2023	6
Research simulator options	10/04/2023	6
Development	17/04/2023	48
Prepare simulation environment	17/04/2023	14
Connect sensors with broker	03/05/2023	5
Create the FL processing system	09/05/2023	14
Create message broker	25/05/2023	4
Add audio output to the system	30/05/2023	7
Add visual output to the system	07/06/2023	4
Testing	12/06/2023	6
Final system testing	12/06/2023	6
Document	13/02/2023	125
Write final documentation	13/02/2023	120
Review & deliver documentation	03/07/2023	5
End date	**08/07/2023**	**135**

Figure 5.1.- GANTT chart of the project [24]

6. Project development

6.1.- Sensors selection

Now that the general aim of the project and the desired variables to monitor are determined, the sensors can be selected.

6.1.1.- Heart rate sensor

In order to measure the heart rate of the subject, there are many different methods, but the best involve manual measurement or an electrocardiogram, which are not very comfortable and could even be intrusive while driving. Still, nowadays smartwatches and cardiac bands have become really good at measuring the heart rate while being very comfortable to wear, a key factor for general adoption between users. The SMIOT research team already had a cardiac band they could lend for the development of this project. The provided model was a *Polar H10*, which has Bluetooth and ANT+ connectivity, it is very accurate (one of the best at its class, used in professional sport training), and it is very comfortable to wear. In addition, there was already a developed piece of code in Python that connects the the band to the system via Bluetooth and sends the heart rate through MQTT messages.

Figure 6.1.- Polar H10 cardiac band [24]

6.1.2.- Camera

When choosing a camera for a computer, there are a few parameters to consider. First of all it is image quality, as it is what the user will see. Not only the video capture resolution is important, which nowadays FullHD is considered the basic standard, but also how good it processes the images, as two images with the same resolution can be perceived to be of different quality. Another hey factor is its connectivity and compatibility, as it may not be possible to use it for an specific system. Some additional features can also be considered, such as having a built-in microphone, the Field Of View (FOV), or software enhancements.

The face recognition system can utilize any webcam that can transmit video to a computer. Some options arose, like the Logitech StreamCam or the Insta360 Link or the Logitech Brio 500, but the Auket PC-LM1E was finally chosen. It has FullHD resolution with good picture quality for the price, USB A connection, compatibility with any modern Operating System (Windows, Linux and iOS), and even a built in microphone.

Figure 6.2.- Aukey PC-LM1E webcam [24]

6.1.3.- Steering wheel

There are three general types of controls that can be used inside the simulation to drive the car. These are keyboard keys, a controller with joysticks (like the ones from the PlayStation 4 or the Xbox One), but to ensure a more realistic behaviour, the best one is a steering wheel with pedals. Many models exist, ranging in price and features, but for many people, the *Logitech G29* combo offers a great balance, being the most common steering wheel. Not only that, but as it is so popular, it has a lot support behind from developers, communities and platforms. As an example, it is the only steering wheel for which MATLAB provides native support, and it is the default option in the sample codes of the CARLA simulator as well. It really makes for a perfect candidate to use in this project.

Figure 6.3.- Logitech G29 Steering wheel [24]

6.1.4.- Raspberry Pi and hat

The SMIOT department also provided a Raspberry Pi 4 with a very complete hat, which is an add-on board to increase the functionality of the Pi.

The Raspberry Pi is a Single Board Computer (SBC). These types of computers are known to be very small and low powered PCs that integrate all essential components to work, and even some additional features, like General Purpose Input/Output or GPIO. In consequence, they are highly versatile and very popular for many different types of projects. The Raspberry Pi 4 is probably the best SBC, as it is the latest version of the most popular option among all use cases.

It has a four core ARM Cortex-A72 Central Processing Unit (CPU) which can handle 4K at 60 fps, up to 8 GB of Random Access Memory (RAM), and both Bluetooth and WiFi wireless connectivity. Even with the price inflation, it is still has a reasonable price, and having such a big community behind makes it the perfect choice for this project. As described before, both the cardiac band and the add-on board are already integrated.

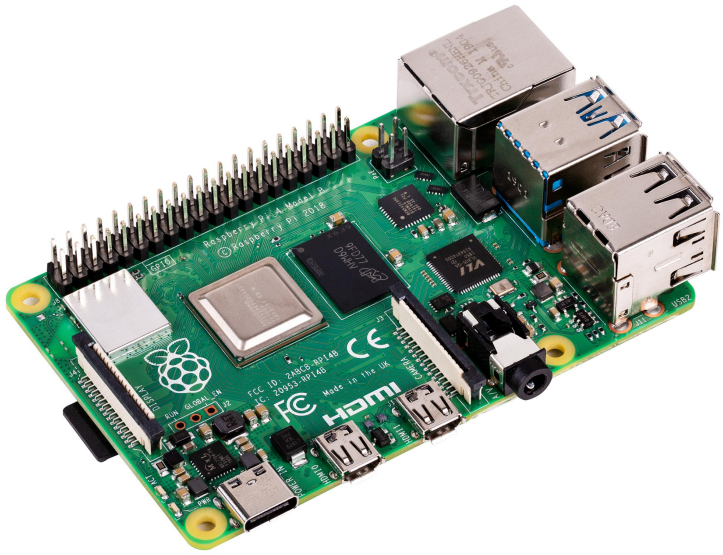


Figure 6.4.- Raspberry Pi 4 SBC [24]

The given Pi hat is a custom PCB that connects to the GPIO pins on the board and gets power from a USB A connector. It is designed to integrate many differnet types of sensors in order to monitor relevant parameters inside a car. It incorporates a GPS position module (GY-GPS6MV2), an IMU module to measure accelerations and magnetic fields (LSM9DS1), a sound detector (LMV324), an air quality sensor (CCCS811), a light detector (BH1750), and a temperature, pressure and humidity sensor (BME/BMP280).

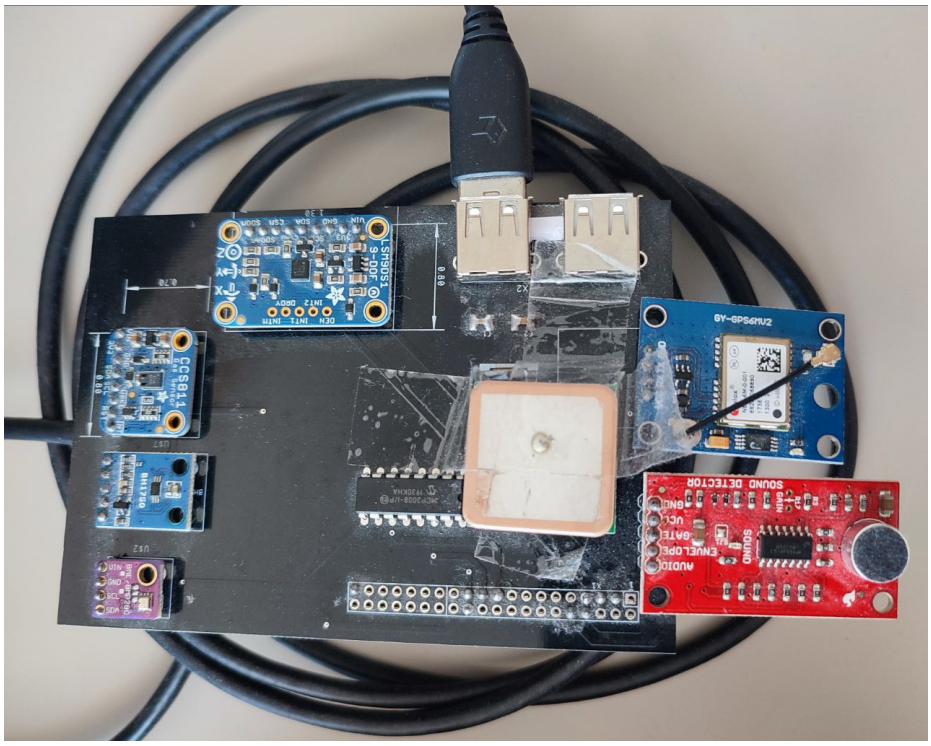


Figure 6.5.- Raspberry Pi hat provided [24]

6.2.- Simulator

Selecting a good simulator was one of the most important choices of the project. It will serve as a real test base for the developed system, without the danger and cost involved on using a real vehicle. Although on a car you have to choose and use real sensors, in a simulator you have to deal with what the development environment offers you. As there is a requirement of extracting data from the simulation, a good API is key to achieve this goal. From all the possible car driving simulator choices, two main ones appear for this purpose, which are *CARLA* and *BeamNG.tech*. Both offer a rich Python API, and are widely used in ADAS and autonomous driving development, but they present some differences that make them suited for different scenarios.

On the one hand, *BeamNG.tech* is a really realistic and detailed simulator that offers a wide range of maps and vehicles [50]. Its vehicle physics for dynamics and crashes far exceed the ones offered by any other simulator. All car components are rigorously simulated, from tire and suspension behaviour to even engine models. That is the reason why it is often used for studying

car performance, crash testing and general driving. In contrast, it is not open source and the API is somewhat complex to use, as it doesn't have the community and support that CARLA enjoys.

On the other hand, *CARLA* has less detailed physics and not as many customization variety, but it focuses more on research and development of autonomous driving algorithms and testing advanced driving systems, as it is primarily designed for that. As a result, it has more and better sensor solutions. It is also open source and it implements an API that is more easy to use, so anyone can run the simulator and therefore there is a lot of community behind that can help with any problem. There is a little drawback in that it is a more demanding simulator to run than BeamNG, but any modern PC with a reasonable amount of graphics computing power should be able to run both of them.

With these considerations in mind, CARLA was the winner choice for this project.

Figure 6.6.- CARLA simulator screenshot [24]

6.3.- Data communication

In order to send the received data from the sensors to the processing system, a communication protocol is needed.

6.3.1.- Application protocol

As the aim is to connect only a few sensors, one of the best choices is to use ZeroMQ. As described in their webpage, ”it is a high-performance asynchronous messaging library, aimed at use in distributed or concurrent applications” [51]. It provides many different functionality, like message queues, to enable communication in many types of formats. However, unlike message-oriented middleware, a ZeroMQ system can run without a dedicated message broker. This is a double sided sword, as point to point communication is made easy and direct, but it is still a more complex communication platform than those that use a central broker.

Another great choice is to use MQTT. It is designed to be extremely lightweight, in order to enable communication using few processing and bandwidth resources [52]. In this project, the messages are going to be for local communication between different programs inside the system, but the usage of this protocol presents the advantage of enabling easy remote processing of the data in case it could be required. MQTT also implements queues, disconnecting the consumer from the producer of the information, as it follows a centralized communication scheme.

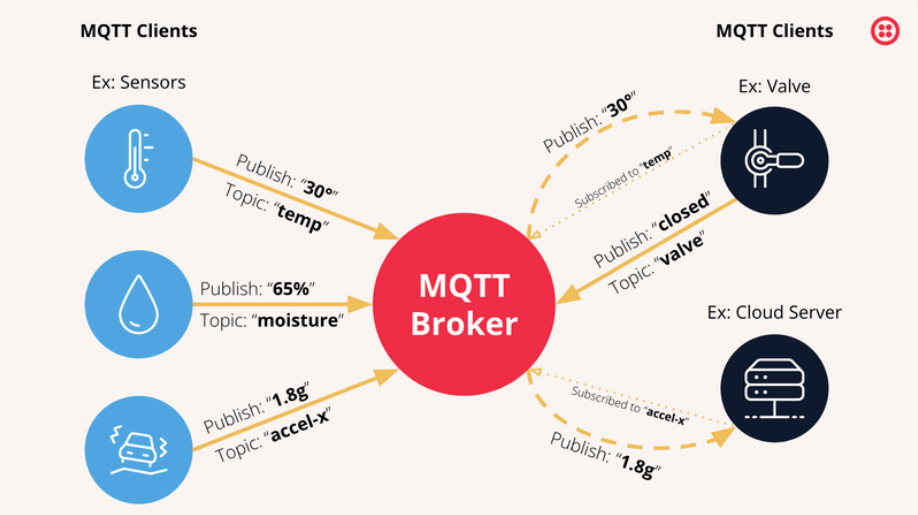


Figure 6.7.- MQTT generic architecture diagram [53]

The general architecture of MQTT is composed by three different nodes: the publishers, the message broker and the subscribers. Publishers are the nodes in charge of sending the messages. They categorize their information using topics, and they send it to the broker. Meanwhile, the broker is the intermediary between publishers and subscribers. It forwards data based on their

topic subscription. Lastly, the subscribers are the final nodes that receive the messages, so they have to notify the the broker which messages they want to receive using a specific topic (or a wildcard to subscribe to multiple).

As a few parts of the code used already rely on MQTT as a communication protocol for transmitting sensor data, there was no reason to change it. It is really easy to use, there is a lot of support for it, and it fits perfectly for the requirements of this project.

6.3.2.- MQTT client implementation

In Python, the most famous library is the Paho MQTT library [54]. Apart from the basic functions to connect to the broker, subscribe to topics and send messages, it defines some other functions that you can overload with the desired code. These special functions are called whenever an event happens, such as a successful connection to the broker server, or a received message.

There are three main options to define in MQTT messages: Quality of Service (QoS), message retention and topics. Quality of Service stands for how many times will the message be sent, and there are three levels. For QoS 0, the message will only be sent one time, without reception guarantee. With QoS 1, the message will be received at least one, because the receiver will acknowledge receiving it, but can result in received duplicates. Finally, QoS 2 is defined as exactly one delivery, so that no duplicates exist. In this application, QoS 0 is probably the best option, as one lost message is not as important because there is a constant rate of them (one every second), and using as little bandwidth as possible is better.

Next, message retention can be specified. This features enables a new subscriber of a topic to receive the last sent message with it. It is not important in this context, as once the system is working there is supposed to be a constant flux of messages.

Lastly, the topics have to be defined. The parent topic for all sensor data is ”sensors”, and for all output data is ”output”. All messages will use one of them, and each different piece of information from each device will have an specific topic derived from it. The used ones can be consulted here:

- ”sensors/h10/rate”: Heart rate from the cardiac band.
- ”sensors/cam/emotion”: Emotion detected by the camera.
- ”sensors/sim/totalacc”: Total acceleration value from the simulation.
- ”output/risk/percentage”: Percentage value of the risk evaluation in the FL system.
- ”output/risk/threshold”: Boolean value from evaluating the percentage value with a predefined threshold.

6.3.3.- MQTT broker implementation

As previously discussed, in order to take advantage of 5G, it was needed to setup a MQTT broker server in the 5G cloud of the Thin5G lab in the university. Many possible choices appear, but the chosen one was the Mosquitto MQTT broker, as it is open-source and has a really big community, it is lightweight and very efficient, can be run in any operating system, supports QoS, and has many other additional features [55]. In addition, it can be used without even installing it, by running the published docker image [56].

As explained in their own webpage, docker containers are very secure by default [57]. This is because it allows for process-level isolation and resource limitation, so that any running application can interfere with any other processes in the machine. Additionally, it provides network and disk segmentation to segregate even more the containers, and it includes a scanning and vulnerability assessment service. Lastly, images are created to be immutable and very secure from the start, so that they can’t be compromised.

A container can be launched with the previous image and a specific broker configuration by enabling an external volume, so that the ”mosquitto.conf” configuration file can be read from inside. Inside this file some parameters are specified, such as the port and interface that the server has to listen to (all interfaces in port 1883), authentication parameters and others.

6.3.4.- 5G network

In order to test the system with a 5G network, first it is needed an available network. The university provided access to the Thin5G laboratory, which has a deployed 5G NR private

network, with SIM cards and 5G compatible modems at our disposal. The base station was implemented using a Firecell server, which incorporates all the required modules to manage the 5G network in standalone mode, and an USRP device to act as the wireless access point of the network [58]. This base station is connected to a Local Area Network in the laboratory, so a private cloud for the connected 5G devices can be established. The MQTT broker was implemented in this LAN.

Figure 6.8.- Thin5G laboratory [24]

On the other side, the client needs to have a way of connecting to the 5G network. This is possible using a registered SIM card, and a 5G wireless router. The used model for the experimentation was the Teltonika RUTX50 [59]. It is an industrial-grade router capable of connecting with two different SIM cards for automatic failover. It can connect to the Raspberry Pi through dual-band WiFi or through one of the four Gigabit Ethernet ports. It has support for many different protocols, such as MQTT or SNMP, and it even has built-in Input/Output ports to connect external devices and sensors, making it suitable for IoT.

Figure 6.9.- 5G NR compatible router [59]

6.4.- Data processing

As evaluated before, between all the options, a Fuzzy Logic algorithm was chosen. But there are many ways and languages to implement a given algorithm.

6.4.1.- Programming platform

Obviously, there are lots of possibilities to implement the system, but the general characteristics of the system have to be considered. Programming languages can vary from very low level, like C or C++; to high level, like Python or MATLAB. Somewhere in the middle lie R and Java.

For example, a great development platform for fuzzy logic prototyping is *Matlab* [60]. It offers a really complete set of tools to develop complex FL systems with the usage of Simulink blocks. But it did not seem to be the best choice, as it does not offer good implementability because it lacks being able to compile a program and execute it freely. It is geared more towards testing and research, rather than development of real systems.

There are also platforms geared towards fuzzy logic implementation in really low powered devices, like micro controllers. C and C++ seem the best choices for this use case, or even *fuzzyTECH*, as being so low level they permit a lot of power optimization and efficiency. Still,

programming with these languages is very challenging, because it is really easy to make mistakes. For this specific use case in which the goal was to implement a real system, but not a mass produced system, a mid balance was desired.

Evaluating every language, a clear choice appeared. Python is a really accessible programming language, with an enormous quantity of libraries. It offers a good middle balance, as it has a simple syntax, support for many different tasks, and still a good possibility of implementation in a real system. Lots of libraries are available to simplify the development process in many different contexts, one of its main advantages. In addition, it is not slow at all, having its core functionality programmed in C, and is compatible with nearly any platform, as all mainstream operating system can run it [61]. Besides, it is also the language used in the other parts of the project, such as the API of CARLA, Ignacio's system, or the SMIOT provided code with the sensors. So, choosing it makes the hole development much more consistent and uniform.

In summary, Python seems like the best programming language for the whole project, as it combines ease of use and capability, and adds on to the consistency of the project.

6.4.2.- Fuzzy Logic Python library

Still, even if the programming language is already decided, more is still needed to develop the project. Libraries, specially in Python, have many benefits. For example, they bring functionality to the code without having to program all the details of an algorithm, making the development process much faster, as you just need to use the functions and classes given. Furthermore, they allow you to reuse highly stable and supported code that comes from many professional developers (in most cases).

But having so many libraries, it is hard to choose one. *snykAdvisor* is a page that recommends python libraries, helping with decision-making [62]. Many different libraries can be found, like *simful* [63], *JuzzyPy* [64], *scikit-fuzzy* [65], or *fuzzylogic* [66]. However, checking the previous webpage seems that the "healthiest" library in base of active user base and recent development is the last one, called *fuzzylogic*.

6.4.3.- Fuzzy Logic algorithm

When evaluating data with fuzzy logic, two main models appear: Mamdani and Sugeno. Some other approaches combine neural networks into their algorithms, but as these systems can introduce a lot of uncertainty on the resulting value, they are not very desirable for this application. Therefore, only those two are going to be considered for this application.

The Mamdani algorithm uses fuzzy sets, which represent truth values of linguistic variables, and can be associated to one or many inputs and/or outputs. In other words, fuzzy sets are the variables in which the defined rules depend on, and they represent the ”truthiness” of the value. These rules are also linguistically defined by ”If-Then” statements, and they represent how the system should behave in an intuitive way. With all that data, the system can determine an appropriate output by inference for each, calculating the degree of each membership for each rule, and then aggregating them to get a final output. Some different methods exist to get this result, such as weighted average, maximum membership, or centroid calculation. All these characteristics result in a very interpretable system, as all linguistic rules and membership functions can be easily understood. In addition, it is highly tunable, as the weight of each membership function on the total output can be modified.

In turn, the Sugeno model (also known as Takagi-Sugeno-Kang or TSK) makes use of linear or constant output functions. The output variables are represented by mathematical equations rather than linguistic terms, and they are then combined into a final output by a weighted average aggregation. As a result, the system is very precisely defined by mathematical modelling, which can vary a lot in complexity depending on the requirements, but is usually related to being less easy to understand. Meanwhile, the advantage of being so mathematically reliant is that it is more computationally efficient, and it can better approximate nonlinear systems by selecting appropriate functions and coefficients.

When comparing both of them, four main differences appear: their computational efficiency, their internal modelling, their interpolation, and their handling of uncertainty. Although the low power requirements of the system makes the Sugeno model a compelling option, the selected one

was Mamdani. In the end, it makes for a more interpretable option with its rule based approach, and it is more effective at dealing with uncertain of ambiguous situations, y common in ADAS.

6.4.4.- Fuzzy Logic implementation

All Fuzzy Logic code is implemented in a separate file. It contains all needed variables, and all MQTT related code to handle connections and messages. For example, the ”mqtt_connect()” function is in charge of connecting to the broker, starting a loop to process messages separately from the main thread, and subscribe to all necessary topics.

```
def mqtt_connect(ip):
    mqtt_client = mqtt.Client()
    mqtt_client.connect(ip, 1883, 60)
    mqtt_client.loop_start()
    mqtt_client.subscribe("sensors/sim/#")
    mqtt_client.subscribe("sensors/cam/emotion")
    mqtt_client.subscribe("sensors/h10/#")
    mqtt_client.on_message=on_message
    print("Connection to broker successful")
    return mqtt_client
```

Meanwhile, the ”on_message()” function is overloaded in order to be able to handle all incoming data from the sensors.

```
def on_message(client, userdata, message):
    global hr
    global accel_stdev
    global accel_list
    global emotion_list
    try:
        print("received message ", message.payload.decode("utf-8"), " with
   topic ", message.topic)
        if (message.topic == "sensors/sim/totalacc"):
```

```
            update_list(accel_list, float(message.payload.decode("utf-8")),
    30)
            accel_stdev = st.stdev(accel_list)

        if (message.topic == "sensors/cam/emotion"):
            update_list(emotion_list, message.payload.decode("utf-8"), 30)

        if (message.topic == "sensors/h10/pulse"):
            hr = int(message.payload.decode("utf-8"))
    except:
        print("Could not process incoming message")
```

A few additional functions were defined to help process data, being ”update_list()” in order to update lists in a Last In First Out (LIFO) order, and ”count_mean_list()” to count the percentage in a list of how full it is of a given element.

```
def update_list(list, element, target_length):
    global accel_list
    global emotion_list
    list.insert(0, element)
    if (len(list)>target_length):
        list.pop()

def count_mean_list(list, element):
    return (list.count(element)/len(list))*100
```

Before exploring the last function, the ”main()” has to be explained. In it, once it has been tested that the connection with the broker is established, the Fuzzy Logic sets are defined. Three sets are used as input, which they are for the actual heart rate value, accelerations standard deviation and the mean received emotion over the evaluation period. The standard deviation is used for accelerations, as it manages to represent in a single number the variation in accelerations over a length of time, and not just in a unique value. Once the percentage risk value is calculated, it is compared against two threshold values to activate or deactivate a condition with hysteresis.

```
def main():
    global hr
    global accel_stdev

    argparser = argparse.ArgumentParser(
        description='Emotion based ADAS FL processing system')
    argparser.add_argument(
        '--mqttip',
        metavar='M',
        default='127.0.0.1',
        help='IP of the MQTT server (default: 127.0.0.1)')
    args = argparser.parse_args()

    try:
        mqtt_client = mqtt_connect(args.mqttip)
    except:
        print("Could not connect to the broker")
        exit()


    accel = Domain("Accel standard deviation", 0, 20)
    heart = Domain("Heart Rate", 0, 300)
    emotion = Domain("Panic emotion", 0, 100)

    vals = {accel: accel_stdev, heart: hr, emotion: 0}
    fl_rules = rules(vals, accel, heart, emotion)

    while(True):
        vals = {accel: accel_stdev, heart: hr, emotion: round(
    count_mean_list(emotion_list, "Enfadado"))}
        risk = fl_rules(vals)
        print("=>", risk)
        mqtt_client.publish("output/risk/percentage", payload=risk, qos=0,
    retain=False)
        if (risk > 55):
            mqtt_client.publish("output/risk/threshold", payload="True", qos
    =0, retain=False)
```

```
        if (risk < 45):
            mqtt_client.publish("output/risk/threshold", payload="False",
    qos=0, retain=False)
        time.sleep(1)
```

Meanwhile, just one set is employed for the output, which is the ”risk” percentage, and it will be the output of combining the rules. This set has to be defined inside the ”rules()” function, because the way the library is designed requires it. In addition, inside this function As for the rules, only eight rules are needed in this scenario, because all different possibilities need to be covered, and just low and high values are defined for the three input sets. Rules are important, because they assign a weight to each input set on the final output, and bad definition could lead to strange behaviour on evaluating the end result. Nevertheless, they are not as important as defining good thresholds.

```
def rules(values, accel, heart, emotion):
    risk = Domain("Risk", 0, 100)

    accel.low = S(1, 2)
    accel.high = R(1, 2)
    heart.low = S(60, 100)
    heart.high = R(60, 100)
    emotion.low = S(5, 50)
    emotion.high = R(5, 50)

    risk.low = S(0, 35)
    risk.high = R(65, 100)
    risk.medium =  risk.low &  risk.high

    R1 = Rule({(accel.low, heart.low, emotion.low): risk.low})
    R2 = Rule({(accel.high, heart.low, emotion.low): risk.medium})
    R3 = Rule({(accel.low, heart.high, emotion.low): risk.medium})
    R4 = Rule({(accel.high, heart.high, emotion.low): risk.high})
    R5 = Rule({(accel.low, heart.low, emotion.high): risk.low})
    R6 = Rule({(accel.high, heart.low, emotion.high): risk.high})
    R7 = Rule({(accel.low, heart.high, emotion.high): risk.medium})
```

```
    R8 = Rule({(accel.high, heart.high, emotion.high): risk.high})

    rules = sum([R1, R2, R3, R4, R5, R6, R6, R7, R8])
    return rules
```

Thresholds for each variable ”low” and ”high” values have to be precisely defined. All functions make use of trapezoidal functions, as they are the easiest to implement and debug, and work the same way as the other types.

In the case of acceleration, ”low” is completely defined till 1, and high starts increasing from that value till 2. From that on, all acceleration standard deviation values are considered to be ”high”. X axis is acceleration in m/s^2, and Y axis is the ”membershipness” of a given value over 1. In blue ”low” and in red ”high” definitions.

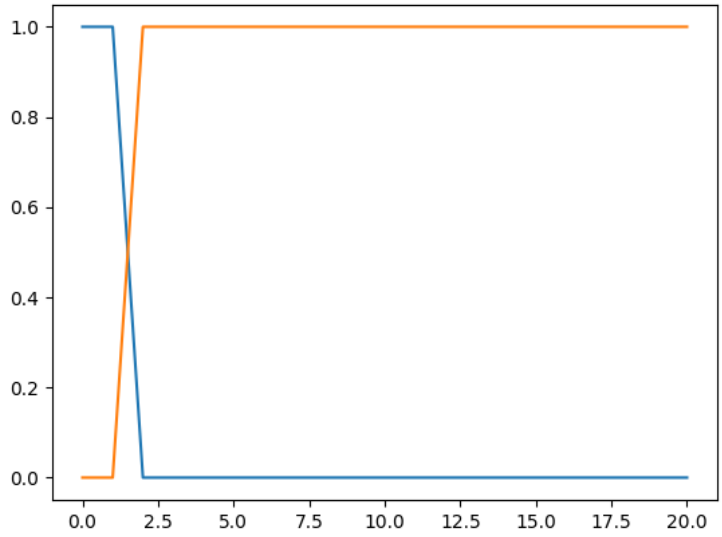


Figure 6.10.- Acceleration threshold membership function representation [24]

As for the case of the heart rate, ”low” is completely defined till 60 beats per second, and high starts increasing from that value till 100. From that on, all heart rate values are considered to be ”high”. X axis is heart rate in beats per second, and Y axis is the ”membershipness” of a given value over 1. In blue ”low” and in red ”high” definitions.

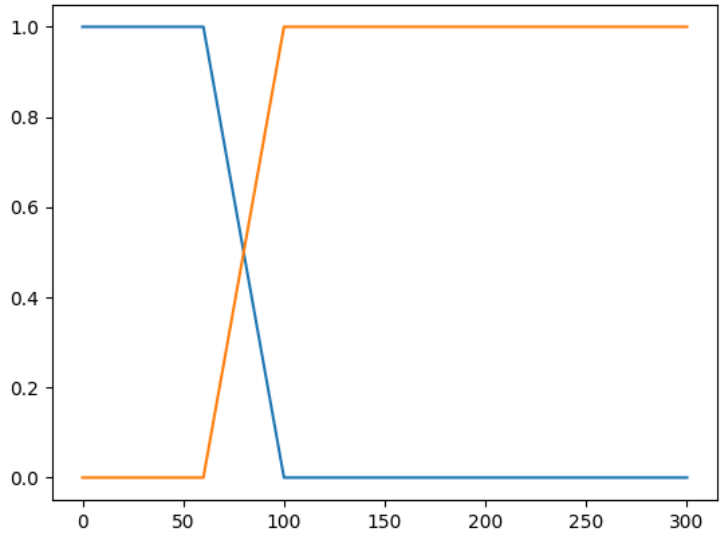


Figure 6.11.- Heart rate membership function representation [24]

Next is the case for emotions. ”Low” is completely defined till 5%, and high starts increasing from that value till 50%. From that on, all emotion values are considered to be ”high”. X axis is the percentage of the mean count of the angry emotion, and Y axis is the ”membershipness” of a given value over 1. In blue ”low” and in red ”high” definitions.

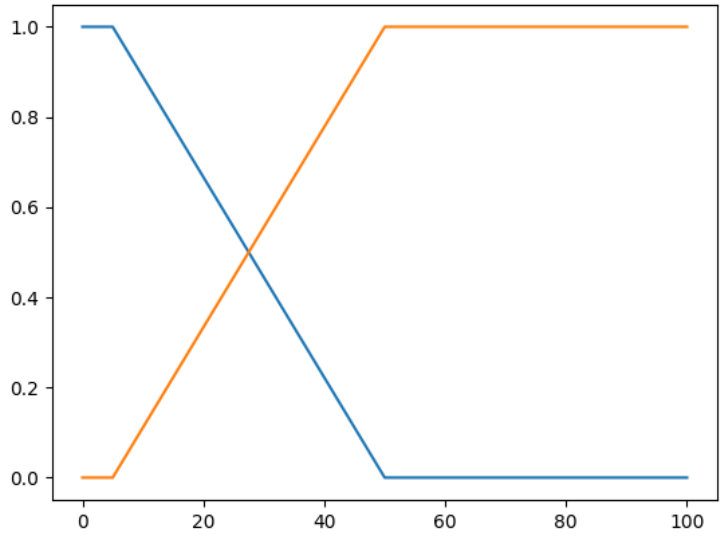


Figure 6.12.- Emotion mean count membership function representation [24]

Finally, the only output function, the risk membership function. ”Low” already starts to decline from the start, and it reaches 35% till it reaches 0. ”Medium” is a the maximum level from 35% till 65%, and ”high” starts increasing from that last value till the end. X axis is the percentage of the calculated driving risk associated to the situation, and Y axis is the ”membershipness” of a given value over 1. In blue ”low”, in red ”medium”, and in green ”high” definitions.

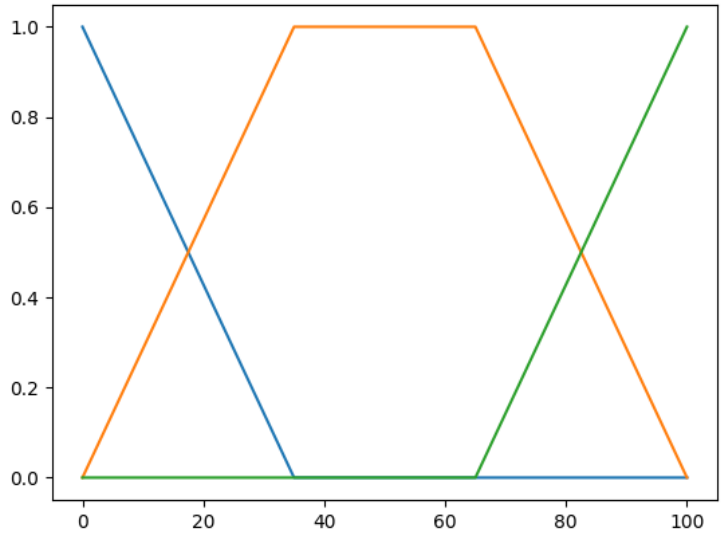


Figure 6.13.- Risk membership function representation [24]

6.4.5.- Emotion detection system

The code for the detection of emotions was developed by Ignacio Prieto Sánchez as his TFG. He created an Artificial Intelligence with a convolutional neural network that he trained with an image data set using machine learning techniques. The result was an AI that managed to detect a person’s emotion with up to a 85% of accuracy. Surprisingly, this task was achieved only by interpreting its face using a normal webcam, while running on a low power device like a Single Board Computer, such as the Raspberry Pi 4. One of the emotions that the system detects is ”Enfadado”, translated to angry, so it is a perfect fit for this project.

In order for the program to work with the rest of the system, some lines were added. In the initial configuration part of the program, some needed variables were defined, and the connection to the broker tries to be established.

```
# Some needed variables are defined
emotion_list = []
clock_seconds = 0
mqtt_broker_ip = "127.0.0.1"

# MQTT broker connection
try:
    client = mqtt.Client()
    client.connect(mqtt_broker_ip, 1883, 60)
    print("Broker connection successful")
except:
    print("Could not connect to the broker")
```

The other modified part is inside the program loop, and it just evaluates the most appeared emotion over a second and sends it to the broker using an MQTT message.

```
        emotion_list.append(emotion_number)
        if (round(time.time()) != clock_seconds):
            clock_seconds = round(time.time())
            max_emotion = max(set(emotion_list), key = emotion_list.count)
            emotion_list.clear()
            client.publish("sensors/cam/emotion", emotion_dict[max_emotion],
    qos=0, retain=False)
```

6.4.6.- Simulation configuration

First of all, the server needs to be properly configured. Once the CARLA simulator has been launched with the .exe executable in Windows, or the .sh script in Linux, a few further steps have to be done before the client connects. The environment has to be changed so that there is traffic

inside the simulation, and even a different map instead of the default one if possible. Both are achieved through the use of two included Python files in the server folder.

Inside the ”PythonAPI
util” folder lies ”config.py”. Launching this file from the command line with the argument ”–map=Town0$”, and replacing the $ for the number of the desired map would change it.

Meanwhile, the traffic generator file can be found at the ”PythonAPI
examples” folder with the name ”generate_traffic.py”. Launching this file from the command line would add around thirty cars that drive autonomously inside the simulation.

The last Python file related to the simulation is an example as well, but this one has been modified to meet some demands. The file is called ”manual_control_steeringwheel.py”, and it implements the client that connects to the server and creates a window to control a car inside the simulation. This car has to be controlled with a Logitech G29 steering wheel, otherwise the client won’t launch. A configuration file (”wheel_config.ini”) is included with the file to configure the controls for this device.

The client is mainly modified to include the HUD modifications (described in subsection 6.5.2), to include some functions to connect to the MQTT broker, and send and receive messages from the Fuzzy Logic processing system. The MQTT functions are very similar to the ones seen before (for example, in 6.4.4, just with different topics), so probably the most relevant part of the code are the added lines inside the game loop, which appear below. That code estimates every second the average acceleration over that second, and sends it in a MQTT message to the broker. Furthermore, it updates the HUD with the new received values from the FL system.

```
            # Gets the acceleration value for each fps
            a = world.player.get_acceleration()
            tot_accel = a.length()
            accel_list.append(tot_accel)

            # Send average acceleration value each second
            if (round(pygame.time.get_ticks() / 1000) != clock_seconds):#
Executes every second
```

```
                accel_avg = sum(accel_list)/len(accel_list)             # Calculate the average acceleration
                accel_list = []                                         # Empty the list
                # Send the acceleration value in a MQTT message
                mqtt_client.publish("sensors/sim/totalacc", payload=accel_avg, qos=0, retain=False)
                clock_seconds = round(pygame.time.get_ticks() / 1000)   # Waits for a new second

            # Update HUD with received values from MQTT
            hud.risk_percentage = risk_percentage
            hud.risk_threshold = risk_threshold
```

Another relevant modification is the inclusion of the ”–mqttip” argument in the argument parser, as it enables to change the IP of the MQTT broker to which it connects without modifying the code. This functionality was later added to the ”audio_output.py” and ”fuzzy_logic_fl.py” files, as they could integrate it and it makes changing the broker IP much easier.

```
    argparser.add_argument(
        '--mqttip',
        metavar='M',
        default='127.0.0.1',
        help='IP of the MQTT server (default: 127.0.0.1)')
```

6.5.- Data output

Finally, with all the processed data, and having a final result on the risk assessment, the system can output the designed responses.

6.5.1.- Audio messages

A few audio messages have been recorded following the de-escalation tips explained in subsection 3.3.3, and remembering the driver about the possibility and consequences of an accident. Once

the condition has been triggered, it is evaluated again after a period of time, so that the driver continues listening to the messages if he does not yield. This interval would have to be between fifteen second and five minutes, so that alerts are still relevant, but that the system can take some time to evaluate the patterns and not stress the driver with too many messages. Once the testing was done, it was concluded that between twenty and forty seconds was the best interval, choosing the smallest for the live demonstration and the biggest for a real-world scenario.

The used Python library is called ”playsound” [67]. It is very simple, as it just takes and audio file as an input (.mp3 or .wav), and reproduces it. On the one hand, it is a bit annoying, as it blocks the thread until the file has been completely played, so no other audio file can be played at the same time. But on the other hand, its simplicity makes it very suitable for projects like this, as no complicated dependencies are needed (such as ffmpeg with pydub), or doesn’t require operating system calls. Furthermore, as the audio files are not that long because messages need to be concise, it is perfectly suited. Just one line is needed to reproduce an audio:

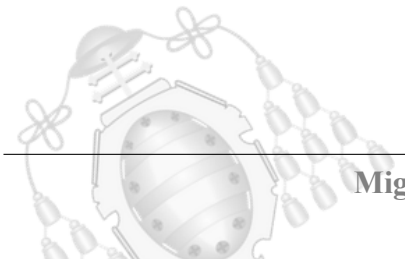
```
playsound('Absolute//Path//To//Audio.mp3')
```

The recorded phrases are listed here, and they play alternately:

- Be careful, only safety truly matters. Please, be careful.
- I feel you are a bit agitated. Please, calm down.
- Please, slow down. If you continue speeding, you may never arrive. Slow down.

Once the threshold condition has been deactivated, a congratulating audio reproduces: ”I feel you are a little bit more relaxed. Thank you.”

The audios were recorded with Audacity in four different .mp3 format files, and the program that reproduces them is called ”audio_output.py”. It is recommended that both the Python and the files are in the same folder. The folder path should be specified with the ”–path” argument in the argument parser when executing the program in a different system from the Raspberry Pi.



6.5.2.- Display alerts

The display alert is done through the Head Up Display (HUD) of the simulation. Once a threshold value of the risk is exceeded, a MQTT message triggers the condition to activate the risk alert on the display with the ”output/risk/threshold” topic and a Boolean value as payload. Another one sends the percentage value of the danger assessment so it can be displayed as well on the screen.

This result was achieved modifying the code of the client that connects to the simulation server (”manual_control_steeringwheel.py” modified example file). Inside it, there is a class called ”HUD” in charge of creating and managing the Heads Up Display of the driver. Adding a few more lines to that piece of code in the ”tick” function, it was possible to include some additional display parameters. This function refreshes the HUD every time the simulation recalculates every parameter, which happens at the refresh rate. The added values were the current acceleration values, the calculated risk percentage and a pilot that indicates if the threshold condition has been reached. Retrieved acceleration values are also sent in an MQTT message to the broker.

```
        self._info_text = [
            'Server:  % 16.0f FPS' % self.server_fps,
            'Client:  % 16.0f FPS' % clock.get_fps(),
            '',
            'Vehicle: % 20s' % get_actor_display_name(world.player, truncate
    =20),
            'Map:     % 20s' % world.world.get_map().name.split('/')[-1],
            'Simulation time: % 12s' % datetime.timedelta(seconds=int(self.
    simulation_time)),
            '',
            'Speed:   % 15.0f km/h' % (3.6 * math.sqrt(v.x**2 + v.y**2 + v.z
    **2)),
           u'Heading:% 16.0f\N{DEGREE SIGN} % 2s' % (t.rotation.yaw,
    heading),
            'Location:% 20s' % ('(% 5.1f, % 5.1f)' % (t.location.x, t.
    location.y)),
            'GNSS:% 24s' % ('(% 2.6f, % 3.6f)' % (world.gnss_sensor.lat,
    world.gnss_sensor.lon)),
```

```
            'Height:  % 18.0f m' % t.location.z,
            'Acceleration:  % 8.0f m/s^2' % (round(a.length)),      # Acceleration
            'Risk %%: % 19.0f %%' % self.risk_percentage,           # Risk %
            '']
        if isinstance(c, carla.VehicleControl):
            self._info_text += [
                ('Throttle:', c.throttle, 0.0, 1.0),
                ('Steer:', c.steer, -1.0, 1.0),
                ('Brake:', c.brake, 0.0, 1.0),
                ('Reverse:', c.reverse),
                ('Hand brake:', c.hand_brake),
                ('Manual:', c.manual_gear_shift),
                ('RISK:', self.risk_threshold), # Risk pilot, activated with threshold message
                'Gear:        %s' % {-1: 'R', 0: 'N'}.get(c.gear, c.gear)]
```

7. Testing

Testing is a vital part of the software development process, as it helps identify errors in order to fix them, it also helps ensure requirements and desired functionality are met, and it even helps improve user experience by assessing how the user would interact with the system. The following sections will test all code functionality, with valid and invalid test cases associated with the previously defined Use Cases.

7.1.- Use Case 01 Tests

7.1.1.- Valid test

ID	VT01
Test name	UC01:Retrieve the driver's emotion
Step description	1. The user launches the application 2. The application connects to the broker 3. The application connects to the camera 4. An emotion is estimated and sent to the broker
Outcome	A notification appears on the screen and a MQTT message is sent with the percieved emotion
Approval	Valid

Table 7.1.- Valid Test 01

```
pi@raspberrypi:~/TFG_Miguel $ python3 predicción_emociones_PC.py
Broker connetion successful
2023-07-15 18:21:00.166718: I tensorflow/compiler/mlir/mlir_graph_optimization_p
ass.cc:176] None of the MLIR Optimization Passes are enabled (registered 2)
2023-07-15 18:21:00.167892: W tensorflow/core/platform/profile_utils/cpu_utils.c
c:118] Failed to find bogomips or clock in /proc/cpuinfo; cannot determine CPU f
requency
```

Figure 7.1.- Demonstration of VT01

7.1.2.- Invalid tests

ID	IT01-a
Test name	UC01:No connection to the broker
Step description	1. The user launches the application 2. The application can't connect to the broker
Outcome	A notification appears on the screen and the application is closed
Approval	Invalid

Table 7.2.- Invalid Test 01-a

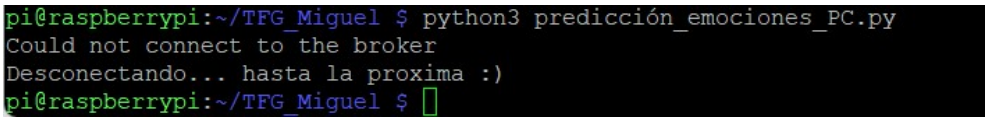


Figure 7.2.- Demonstration of IT01-a

ID	IT01-b
Test name	UC01:No connection to the camera
Step description	1. The user launches the application 2. The application connects to the broker 3. The application can't connect to the camera
Outcome	A notification appears on the screen and the application is closed
Approval	Invalid

Table 7.3.- Invalid Test 01-b

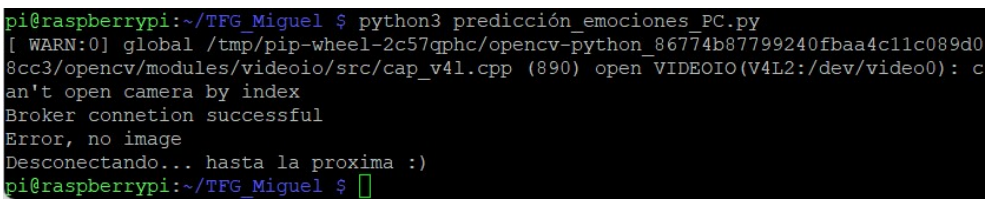


Figure 7.3.- Demonstration of IT01-b

7.2.- Use Case 02 Tests

7.2.1.- Valid test

ID	VT02
Test name	UC02:Retrieve the heart rate of the driver
Step description	1. The user launches the application 2. The application connects to the band 3. The application connects to the broker 4. The heart rate is estimated and sent to the broker
Outcome	A notification appears on the screen and a MQTT message is sent with the perceived heart rate
Approval	Valid

Table 7.4.- Valid Test 02

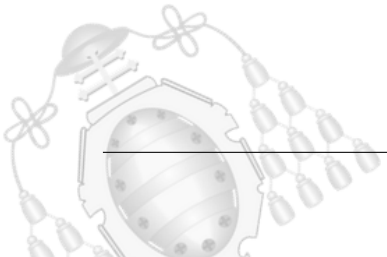
```
pi@raspberrypi:~/CARLA/tfg_carla_control_api $ python3 h10.py
D0:6B:CC:83:95:CB
Connected to MQTT broker
```

Figure 7.4.- Demonstration of VT02

7.2.2.- Invalid tests

ID	IT02-a
Test name	UC02:No Bluetooth connection to the band
Step description	1. The user launches the application 2. The application can't connect to the band
Outcome	A notification appears on the screen and the application has to be closed
Approval	Invalid

Table 7.5.- Invalid Test 02-a



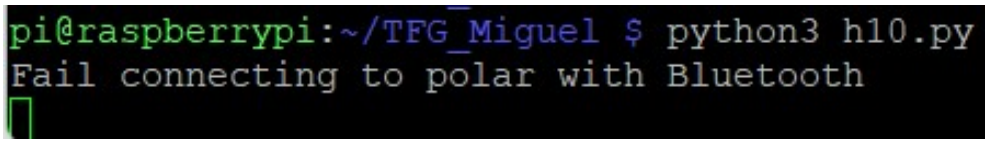


Figure 7.5.- Demonstration of IT02-a

ID	IT02-b
Test name	UC02:No connection to the broker
Step description	1. The user launches the application 2. The application connects to the band 3. The application can't connect to the broker
Outcome	A notification appears on the screen and the application has to be closed
Approval	Invalid

Table 7.6.- Invalid Test 02-b

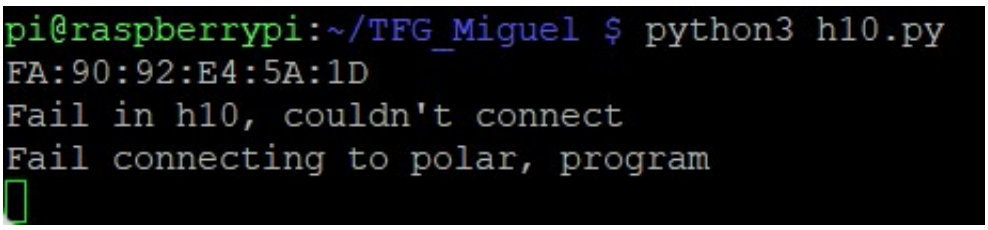


Figure 7.6.- Demonstration of IT02-b

7.3.- Use Case 03 Tests

7.3.1.- Valid test

ID	VT03
Test name	UC03:Retrieve the driver's acceleration values
Step description	1. The user has launched the application and it connects to the broker 2. The driver is controlling the vehicle 3. The acceleration values are sent to the broker
Outcome	The acceleration value appears on the screen HUD and it is sent in a MQTT message
Approval	Valid

Table 7.7.- Valid Test 03

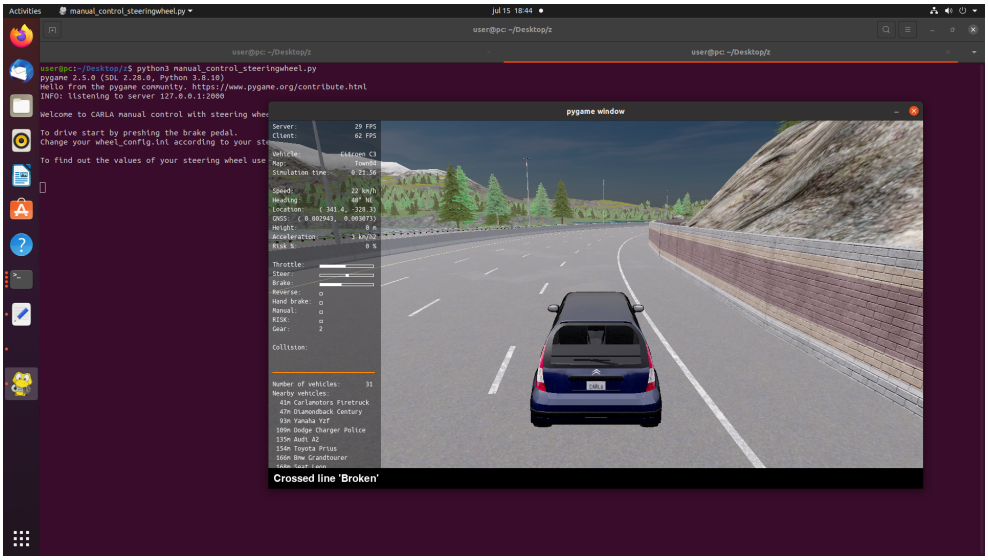


Figure 7.7.- Demonstration of VT03

7.3.2.- Invalid tests

ID	IT03-a
Test name	UC03:No connection to the broker
Step description	1. The user has launched the application but it doesn't connect to the broker
Outcome	A notification appears on the screen and the application has to be closed
Approval	Invalid

Table 7.8.- Invalid Test 03-a

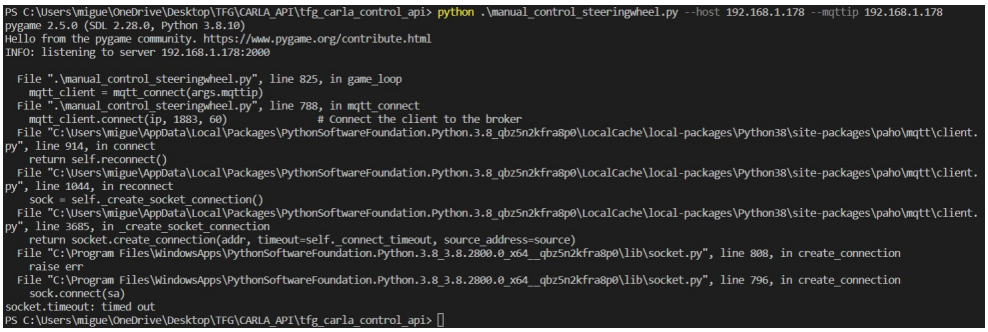


Figure 7.8.- Demonstration of IT03-a

7.4.- Use Case 04 Tests

7.4.1.- Valid test

ID	VT04
Test name	UC04:Launch the driving simulation server
Step description	1. The user launches the server with the ”CarlaUE4.sh” shell script 2. The Unreal Engine version shows on screen 3. A new window with the environment appears
Outcome	A new window with the map view
Approval	Valid

Table 7.9.- Valid Test 04

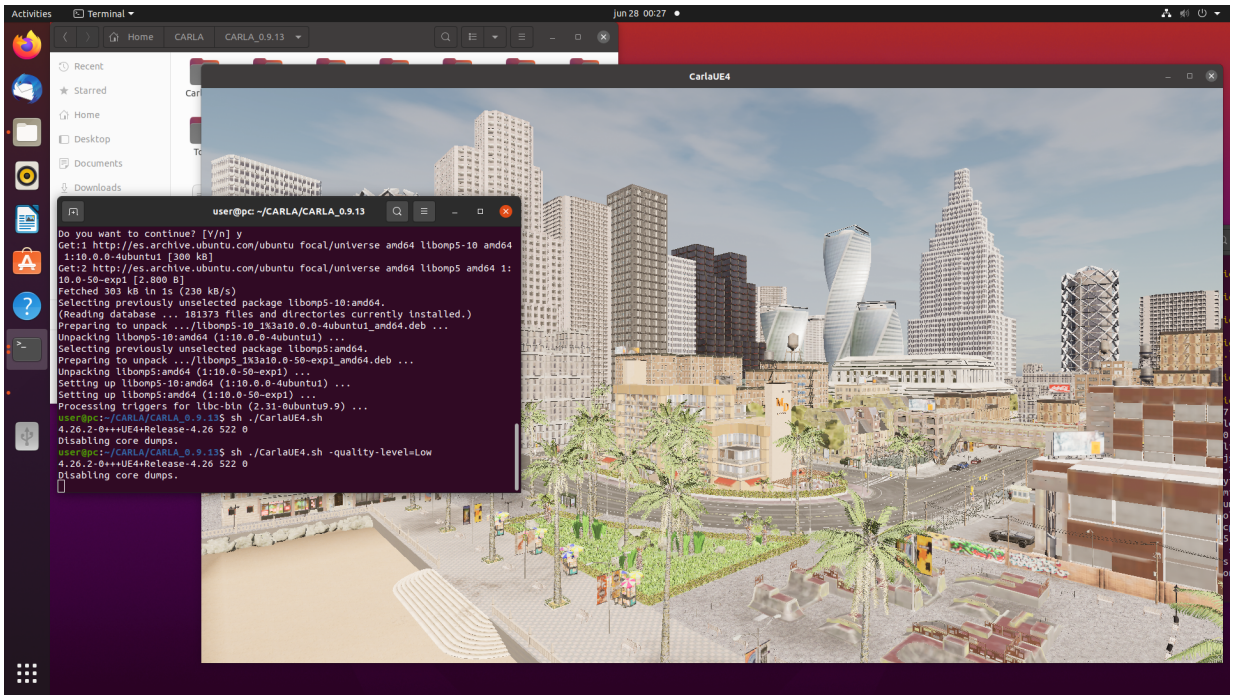


Figure 7.9.- Demonstration of VT04

7.5.- Use Case 05 Tests

7.5.1.- Valid test

ID	VT05
Test name	UC05:Change simulation map
Step description	1. The user has launched the simulation server successfully 2. Run the ”config.py” file with the desired map as an argument 3. The Python program connects to the simulation server
Outcome	The program finishes execution and the map inside the simulation server is changed
Approval	Valid

Table 7.10.- Valid Test 05

```
user@pc:~/CARLA/CARLA_0.9.13/PythonAPI/util$ python3 config.py --map=Town04
load map 'Town04'.
user@pc:~/CARLA/CARLA_0.9.13/PythonAPI/util$ python3 generate_traffic.py
```

Figure 7.10.- Demonstration of VT05

7.5.2.- Invalid tests

ID	IT05-a
Test name	UC05:No connection to the simulation
Step description	1. The user runs the ”config.py” file with the desired map as an argument 2. The Python program can’t connect to the simulation server
Outcome	A notification appears on the screen and the application is closed
Approval	Invalid

Table 7.11.- Invalid Test 05-a

```
user@pc:~/CARLA/CARLA_0.9.13/PythonAPI/util$ python3 config.py --map=Town04
load map 'Town04'.
time-out of 10000ms while waiting for the simulator, make sure the simulator is ready and connected to localhost:2000
```

Figure 7.11.- Demonstration of IT05-a

7.6.- Use Case 06 Tests

7.6.1.- Valid test

ID	VT06
Test name	UC06:Add traffic to the simulation
Step description	1. The user has launched the simulation server successfully 2. Run the ”generate_traffic.py” file with the number of cars as an argument 3. The Python program connects to the simulation server
Outcome	The program keeps executing and cars that drive autonomously appear in the simulation
Approval	Valid

Table 7.12.- Valid Test 06

```
user@pc:~/CARLA/CARLA_0.9.13/PythonAPI/examples$ python3 generate_traffic.py
ERROR: Spawn failed because of collision at spawn position
ERROR: Spawn failed because of collision at spawn position
ERROR: Spawn failed because of collision at spawn position
ERROR: Spawn failed because of collision at spawn position
ERROR: Spawn failed because of collision at spawn position
ERROR: Spawn failed because of collision at spawn position
spawned 30 vehicles and 4 walkers, press Ctrl+C to exit.
```

Figure 7.12.- Demonstration of VT06

7.6.2.- Invalid tests

ID	IT06-a
Test name	UC06:No connection to the simulation
Step description	1. The user runs the ”config.py” file with the desired map as an argument 2. The Python program can’t connect to the simulation server
Outcome	A notification appears on the screen and the application is closed
Approval	Invalid

Table 7.13.- Invalid Test 06-a

```
user@pc:~/CARLA/CARLA_0.9.13/PythonAPI/examples$ python3 generate_traffic.py

destroying 0 vehicles

done.
Traceback (most recent call last):
  File "generate_traffic.py", line 165, in main
    world = client.get_world()
RuntimeError: time-out of 10000ms while waiting for the simulator, make sure the simulator is ready and connected to 127.0.0.1:2000

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "generate_traffic.py", line 377, in <module>
    main()
  File "generate_traffic.py", line 363, in main
    client.apply_batch([carla.command.DestroyActor(x) for x in vehicles_list])
RuntimeError: rpc::timeout: Timeout of 10000ms while connecting to 127.0.0.1:2000
```

Figure 7.13.- Demonstration of IT06-a

7.7.- Use Case 07 Tests

7.7.1.- Valid test

ID	VT07
Test name	UC07:Launch the driving simulator client
Step description	1. The user launches the car driving client (”manual_control_steeringwheel.py”) 2. The app connects to the simulation server 3. The app connects to the MQTT broker
Outcome	A new window launches with the car and it can be controlled with the steering wheel
Approval	Valid

Table 7.14.- Valid Test 07

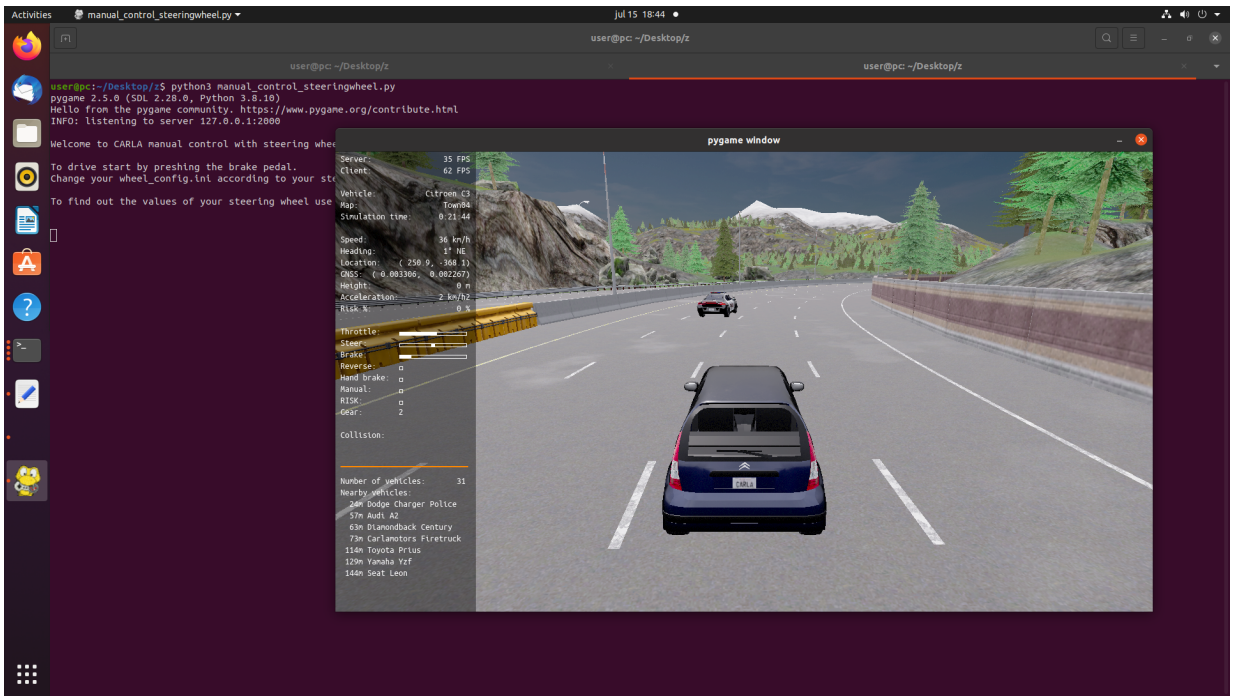


Figure 7.14.- Demonstration of VT07

7.7.2.- Invalid tests

ID	IT07-a
Test name	UC07:Can't establish a connection to the simulation server
Step description	1. The simulation environment client is launched 2. A new window is created
Outcome	A notification appears on the screen and the application is closed
Approval	Invalid

Table 7.15.- Invalid Test 07-a

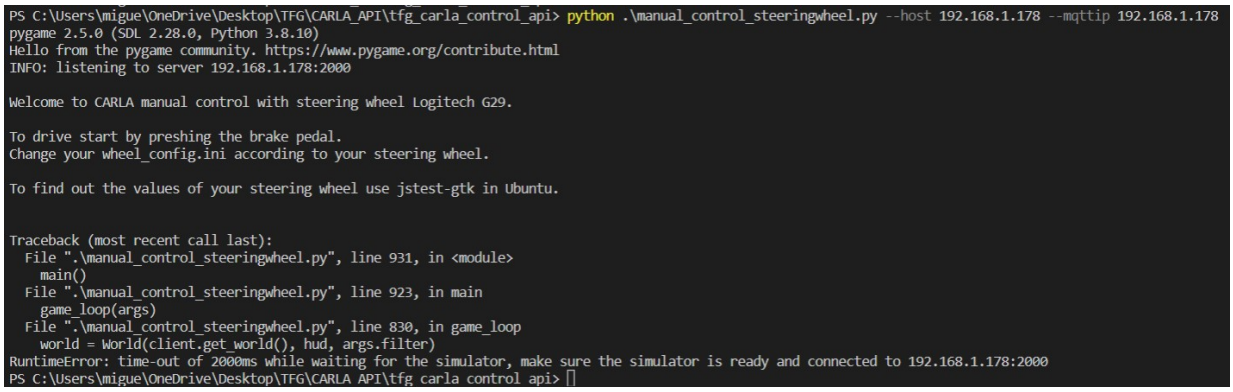


Figure 7.15.- Demonstration of IT07-a

ID	IT07-b
Test name	UC07:No message sent to the broker
Step description	1. The simulation environment client is launched 2. A new window is created
Outcome	A notification appears on the screen and the application is closed
Approval	Invalid

Table 7.16.- Invalid Test 07-b

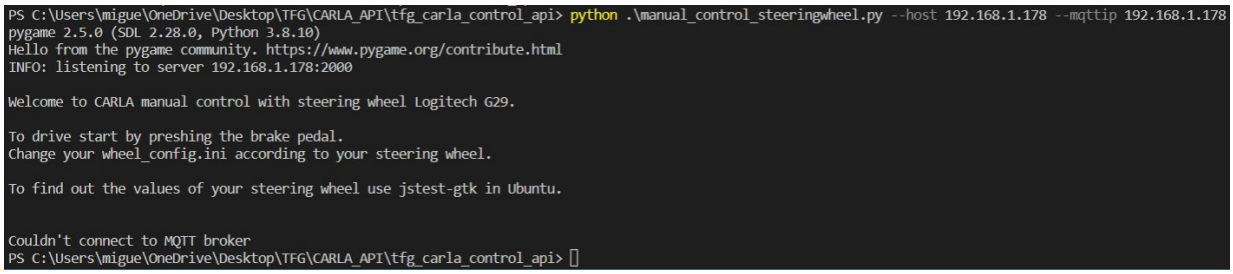


Figure 7.16.- Demonstration of IT07-b

ID	IT07-c
Test name	UC07:No steering wheel is connected
Step description	1. The simulation environment client is launched
Outcome	A notification appears on the screen and the application is closed as the Python program can't start (missing input device)
Approval	Invalid

Table 7.17.- Invalid Test 07-c

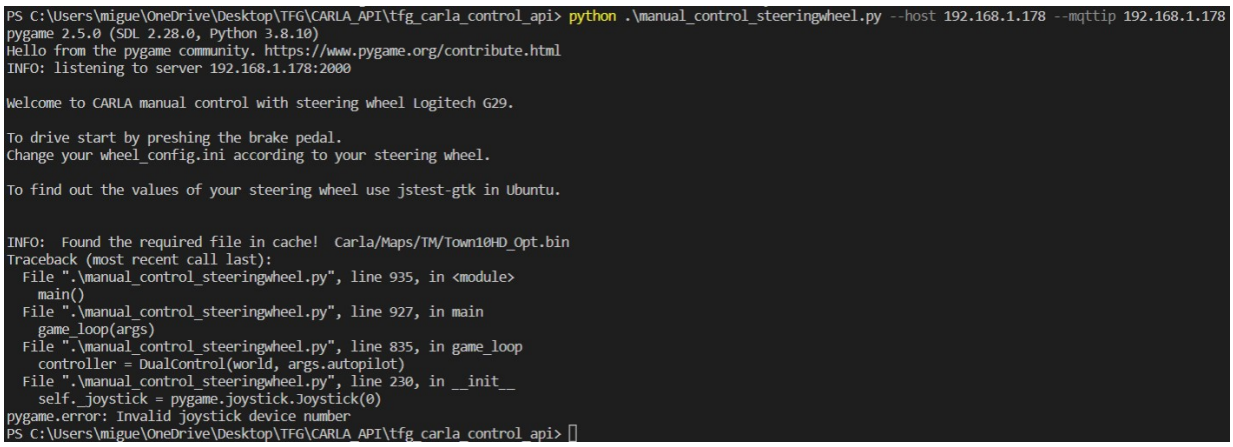


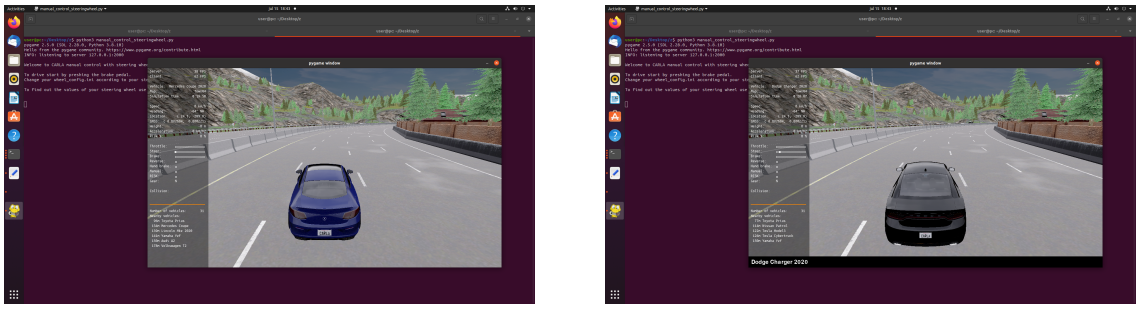
Figure 7.17.- Demonstration of IT07-c

7.8.- Use Case 08 Tests

7.8.1.- Valid test

ID	VT08
Test name	UC08:Change simulation vehicle
Step description	1. The user has launched the car driving application successfully 2. The user presses the key
Outcome	The car has changed and a notification appears below the screen
Approval	Valid

Table 7.18.- Valid Test 08



(a) Before (b) After

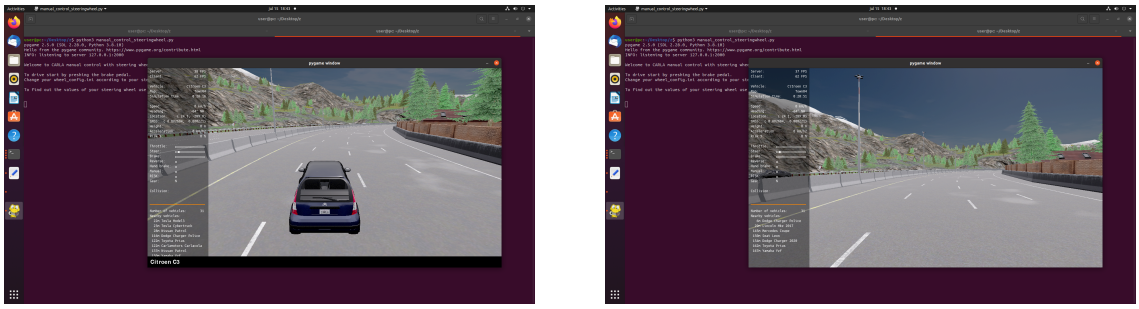
Figure 7.18.- Demonstration of VT08

7.9.- Use Case 09 Tests

7.9.1.- Valid test

ID	VT09
Test name	UC09:Change simulation driver POV
Step description	1. The user presses a key 2. The driver Point Of View changes
Outcome	The client camera POV has moved of position
Approval	Valid

Table 7.19.- Valid Test 09



(a) Before (b) After

Figure 7.19.- Demonstration of VT09

7.10.- Use Case 10 Tests

7.10.1.- Valid test

ID	VT10
Test name	UC10:Evaluate the accident risk involved
Step description	1. The user launches the ”fuzzy_logic_fl.py” program 2. The application connects to the MQTT broker 3. It starts calculating the risk percentage with the received information
Outcome	A percentage result appears on the screen
Approval	Valid

Table 7.20.- Valid Test 10

```
/Desktop/TFG/CARLA_API/tfg-driver-assistant/fuzzy_logic_fl.pyrol_api>
Connection to broker successful
=> 16.006600660066006
=> 16.006600660066006
```

Figure 7.20.- Demonstration of VT10

7.10.2.- Invalid tests

ID	IT010-a
Test name	UC10:No connection to the broker
Step description	1. The user launches the ”fuzzy_logic_fl.py” program 2. The Python program can’t connect to the MQTT broker
Outcome	A notification appears on the screen and the application is closed
Approval	Invalid

Table 7.21.- Invalid Test 10-a

```
PS C:\Users\migue\OneDrive\Desktop\TFG\CARLA_API\tfg_carla_control_api> & C:/Users/migue/AppData/Local/Microsoft/WindowsApps/python3.8.exe c:/Users/migue/OneDrive
/Desktop/TFG/CARLA_API/tfg-driver-assistant/fuzzy_logic_fl.py
Could not connect to the broker
PS C:\Users\migue\OneDrive\Desktop\TFG\CARLA_API\tfg_carla_control_api>
```

Figure 7.21.- Demonstration of IT10-a

7.11.- Use Case 11 Tests

7.11.1.- Valid test

ID	VT11
Test name	UC11:De-escalation audio message display
Step description	1. The user launches the ”audio_output.py” application 2. The service connects to the broker server 3. If the threshold condition activates, a de-escalation audio will be played. If the threshold condition deactivates, a congratulating audio will be played
Outcome	Play one of the recorded audios
Approval	Valid

Table 7.22.- Valid Test 11

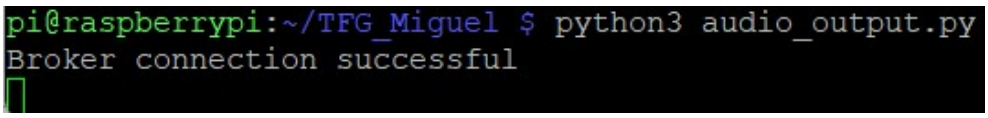
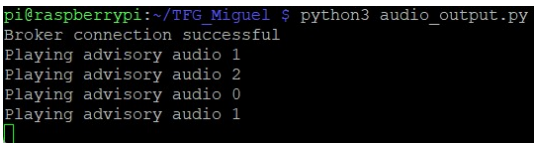
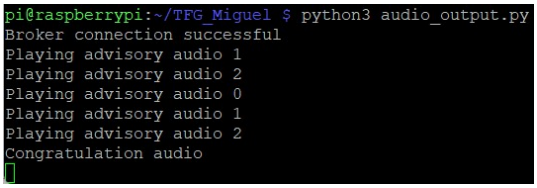


Figure 7.22.- Demonstration of VT11 - Successful broker connection



(a) De-escalating audios



(b) Congratulating audio

Figure 7.23.- Demonstration of VT11 - Audio messages playback

7.11.2.- Invalid tests

ID	IT011-a
Test name	UC11:No connection to the broker
Step description	1. The user launches the ”audio_output.py” application 2. The Python program can’t connect to the MQTT broker
Outcome	A notification appears on the screen and the application is closed
Approval	Invalid

Table 7.23.- Invalid Test 11-a

```
pi@raspberrypi:~/TFG_Miguel $ python3 audio_output.py
Could not connect to the broker
pi@raspberrypi:~/TFG_Miguel $
```

Figure 7.24.- Demonstration of IT11-a

7.12.- Use Case 12 Tests

7.12.1.- Valid test

ID	VT12
Test name	UC12:Visual danger pilot
Step description	1. The user had launched the simulator client successfully, and it had connected to the broker server 2. If the threshold condition activates, a visual danger pilot will lit up
Outcome	An indicator pilot inside the HUD is lit up
Approval	Valid

Table 7.24.- Valid Test 12

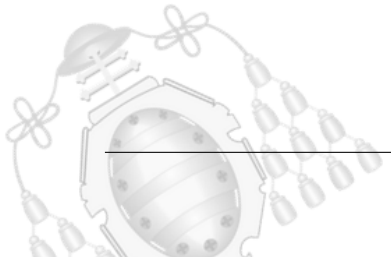


Figure 7.25.- Demonstration of VT12

8. Budget

This chapter summarizes all related costs of the whole project, divided intro three different sections: hardware, software and human resources. Section 8.1 includes the costs of buying new every physical element needed for the development of the project, which includes computers, sensors and additional equipment needed. An additional table (8.2) is shown describing all the different components used in the custom Raspberry Pi Hat made in the SMIOT department. Section 8.2 contains all software used for the development of the project. Section 8.3 summarizes the depreciation and amortization of the hardware and software acquired components, as described in the Spanish Tax Agency (Agencia Tributaria Española). Section 8.4 describes the associated costs of hiring all the people involved in the project. Section 8.5 outlines all the resulting costs associated with the project.

8.1.- Hardware

Equipment	Concept	Units	Cost(€)
Raspberry Pi 4	Single Board Computer	1	59.90
Custom Raspberry Pi HAT	Sensorized board	1	138.94
Samsung EVO Plus 64 GB	Micro SD card	1	13.75
Aukey PC-LM1E	Webcam	1	22.48
Polar H10	Heart Rate sensor	1	75.98
Logitech G29	Steering wheel	1	273.90
Asus VG24VQE	PC monitor	1	179.00
AOC 24B1H	PC monitor	2	100.00
Computer Workstation	Graphics oriented PC	1	969.00
Total Hardware Budget:			1,832.95

Table 8.1.- Hardware Budget

Sensor	Part	Cost(€)
Sound Detector	LMV324	12.45
GPS Module	GY-GPS6MV2	8.53
Accelerometer + Gyro + Magnetometer Module	LSM9DS1	23.39
Air Quality Sensor	CCS811	35.43
Light Sensor	BH1750	2.00
Temperature, Pressure and Humidity Sensor	BME/BMP280	1.51
ADC 10 bit to SPI converter	MCP3008-I/P	2.95
2 USB type A plug	87583-2010RLF	1.97
Custom 2-layer PCB	120x80mm PCB	50.71
Total Custom Hat Budget:		138.94

Table 8.2.- Custom Raspberry Pi Hat Budget

8.2.- Software

Software	Concept	Units	Cost(€)
Microsoft Windows 11 Pro	Operating System	1	195.00
Microsoft 365 Suite (University)	Office Automation Tools	1	0.00
Raspberry Pi OS	Operating System	1	0.00
CARLA	Car Driving Simulator	1	0.00
PyCharm Community Edition	Code Editor	1	0.00
Visual Studio Code	Code Editor	1	0.00
Lucidchart	Figure Designer	1	0.00
PuTTY	SSH Client	1	0.00
WinSCP	SFTP Client	1	0.00
Overleaf	LaTeX Online Editor	1	0.00
Bitbucket	Git Repository Service	1	0.00
Total Software Budget:			195.00

Table 8.3.- Software Budget

8.3.- Amortization

All acquired items are not specific for this project, but are shared along with other projects, so they are susceptible of amortization. As described in the Tax Agency tables, the general amortization of hardware and software equipment over a year is up to 26%, and can be applied for over a maximum of 10 years [68]. In this case, as the project just lasts six months, the amortization over a year is calculated and then the cost is divided by twelve months and multiplied by the desired months.

Part	Cost(€)
Hardware	1,832.95
Software	195.00
Subtotal	2,027.95
Depreciation over 1 year	527.27
Depreciation over 1 month	43.94
Depreciation over 6 months	263.63
Total Hardware and Software cost:	263.63

Table 8.4.- Total Hardware and Software cost of the project (with amortization)

8.4.- Human Resources

Concept	Cost(€/h)	Quantity (hours)	Total Cost(€)
Requirements Analyst	22.00	76	1,672.00
Software Architect	25.00	52	1,300.00
Junior Programmer	15.00	96	1,440.00
Software Tester	16.00	46	736.00
Total Software Budget:			5,148.00

Table 8.5.- Human Resources Budget

8.5.- Total Cost

Part	Cost(€)
Hardware and Software	263.63
Human Resources	5,148.00
Indirect costs (water and light)	350.00
Subtotal	5,761.63
Industrial benefit (6%)	345.70
Subtotal	6,107.33
IVA (21%)	1,282.54
Total Project Budget:	**7,389.87**

Table 8.6.- Total budget of the project

The total cost of the system, as shown in 8.6, is SEVEN THOUSAND THREE HUNDRED EIGHTY NINE EUROS AND EIGHTY SEVEN CENTS (7,389.87€).

9. Conclusions and future work

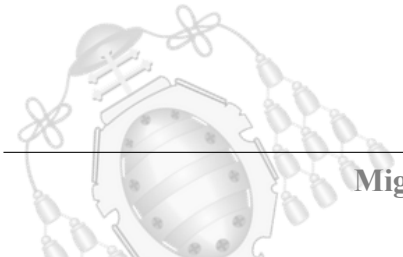
It can be concluded that the system works as designed. All use cases and requirements (either functional or non functional) have been met. In addition, the system has been tested over the 5G network and the performance was good. Additional experiments over the 5G network with more devices are still left, but that is out of the scope of this work, as it would evaluate the network performance, and not the system itself. The overall efficacy of the system could not be verified, because not enough testing could be done across many different drivers to be sure it is relevant and useful for everybody. However, having developed such a unique ADAS that evaluates the driver's emotions is an achievement in itself, as barely no other ADAS used this data to asses the driver and act upon it. Furthermore, all found criteria to design such an special system was followed, so it is likely that the system should be able to perform well under real-world circumstances, maybe with just a little bit more of fine tuning in the thresholds.

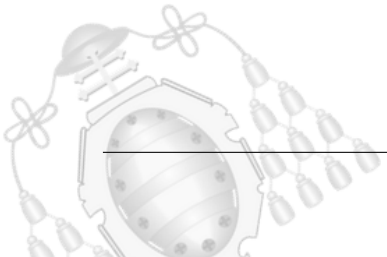
Still, some further work could be useful to improve the developed system:

- Further improved testing: not only test the system with more people, but using additional driving environments such as different cars or maps could help fine tune the thresholds of the system.
- Test other FL algorithms: to implement the Sugeno algorithm or a hybrid approach with neural networks and compare results is a great way to ensure the chosen algorithm was the best choice, because the algorithm was chosen just on basic facts and not system performance.
- Add additional inputs: more variables to monitor can help the system make better choices when evaluating the driver, as there are many other influencing factors that can be measured, such as heart rate variability or noise.
- Add additional outputs: although the main outputs are already programmed, some additional ones could be tested. For example, more different phrases could be recorded. In addition, as previously discussed, music could be another ingredient that could help the driver to relax. But this has to be done through extensive testing, as it may depend on the

driver or even on the specific situation. Moreover, some other senses could be evaluated, such as smell or even touch, as they have not been regarded in seen studies.

- General launcher: create a launcher that is able to start all required programs with a more user friendly graphic user interface.





Bibliography

[1] Mindmajix Technologies. *Top 12 New Technology Trends in 2023 | Latest Tech Trends*. en. Section: Looker. Apr. 2021. URL: https://mindmajix.com/technology-trends (visited on 07/15/2023).

[2] Sidhartha Roy. *Why do we need Privacy-Preserving Machine Learning?* en. June 2020. URL: https://towardsdatascience.com/why-do-we-need-privacy-preserving-machine-learning-7480ddf9f114 (visited on 07/15/2023).

[3] *What's new for 2020? | Euro NCAP*. en. URL: https://www.euroncap.com:443/en/vehicle-safety/safety-campaigns/2020-assisted-driving-tests/whats-new/ (visited on 12/19/2022).

[4] *Road traffic injuries*. en. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries (visited on 07/05/2023).

[5] *Preliminary 2021 EU Road Safety Statistics*. en. URL: https://transport.ec.europa.eu/news/preliminary-2021-eu-road-safety-statistics-2022-03-28_en (visited on 12/19/2022).

[6] https://www.latimes.com/people/emily-baumgaertner and https://www.latimes.com/people/russ-mitchell. *Car crash deaths have surged during COVID-19 pandemic. Here's why*. en-US. Section: World & Nation. Dec. 2021. URL: https://www.latimes.com/world-nation/story/2021-12-08/traffic-deaths-surged-during-covid-19-pandemic-heres-why (visited on 07/15/2023).

[7] Elena Constantinou et al. "Risky and aggressive driving in young adults: Personality matters". en. In: *Accident Analysis & Prevention* 43.4 (July 2011), pp. 1323–1331. ISSN: 0001-4575. DOI: 10.1016/j.aap.2011.02.002. URL: https://www.sciencedirect.com/science/article/pii/S0001457511000169 (visited on 12/19/2022).

[8] Lambros Lazuras et al. “Driving self-regulation and risky driving outcomes”. en. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 91 (Nov. 2022), pp. 461–471. ISSN: 1369-8478. DOI: 10.1016/j.trf.2022.10.027. URL: https://www.sciencedirect.com/science/article/pii/S1369847822002625 (visited on 12/19/2022).

[9] Ying Zhou, Weina Qu, and Yan Ge. “The role of trait emotional intelligence in driving anger: The mediating effect of emotion regulation”. en. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 88 (July 2022), pp. 281–290. ISSN: 1369-8478. DOI: 10.1016/j.trf.2022.05.024. URL: https://www.sciencedirect.com/science/article/pii/S136984782200119X (visited on 12/19/2022).

[10] *Aggressive Driving*. URL: https://www.penndot.pa.gov/about-us/media/Aggressive%20Driving/Pages/default.aspx (visited on 07/15/2023).

[11] Sebastian Zepf et al. “Driver Emotion Recognition for Intelligent Vehicles: A Survey”. en. In: *ACM Computing Surveys* 53.3 (May 2021), pp. 1–30. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3388790. URL: https://dl.acm.org/doi/10.1145/3388790 (visited on 07/10/2023).

[12] Mariam Hassib et al. “Detecting and Influencing Driver Emotions Using Psycho-Physiological Sensors and Ambient Light”. In: Aug. 2019, pp. 721–742. ISBN: 978-3-030-29380-2. DOI: 10.1007/978-3-030-29381-9_43.

[13] Eric R. Dahlen et al. “Driving anger, sensation seeking, impulsiveness, and boredom proneness in the prediction of unsafe driving”. en. In: *Accident Analysis & Prevention* 37.2 (Mar. 2005), pp. 341–348. ISSN: 0001-4575. DOI: 10.1016/j.aap.2004.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0001457504000995 (visited on 12/19/2022).

[14] Elena Constantinou et al. “Risky and aggressive driving in young adults: Personality matters”. en. In: *Accident Analysis & Prevention* 43.4 (July 2011), pp. 1323–1331. ISSN: 0001-4575. DOI: 10.1016/j.aap.2011.02.002. URL: https://www.sciencedirect.com/science/article/pii/S0001457511000169 (visited on 12/19/2022).

[15] Qian Zhang et al. “The effect of the emotional state on driving performance in a simulated car-following task”. en. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 69 (Feb. 2020), pp. 349–361. ISSN: 1369-8478. DOI: 10.1016/j.trf.2020.02.004. URL: https://www.sciencedirect.com/science/article/pii/S1369847819303948 (visited on 07/10/2023).

[16] Mitchell L. Cunningham and Michael A. Regan. “The impact of emotion, life stress and mental health issues on driving performance and safety”. In: *Road & Transport Research* 25.3 (Nov. 2020). Publisher: ARRB Group Ltd, pp. 40–50. DOI: 10.3316/informit.476816178722935. URL: https://search.informit.org/doi/abs/10.3316/INFORMIT.476816178722935 (visited on 07/10/2023).

[17] Ying Zhou, Weina Qu, and Yan Ge. “The role of trait emotional intelligence in driving anger: The mediating effect of emotion regulation”. en. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 88 (July 2022), pp. 281–290. ISSN: 1369-8478. DOI: 10.1016/j.trf.2022.05.024. URL: https://www.sciencedirect.com/science/article/pii/S136984782200119X (visited on 12/19/2022).

[18] Khalid Zaman et al. “Driver Emotions Recognition Based on Improved Faster R-CNN and Neural Architectural Search Network”. en. In: *Symmetry* 14.4 (Apr. 2022). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 687. ISSN: 2073-8994. DOI: 10.3390/sym14040687. URL: https://www.mdpi.com/2073-8994/14/4/687 (visited on 07/10/2023).

[19] Nuria Mateos-García et al. “Driver Stress Detection from Physiological Signals by Virtual Reality Simulator”. en. In: *Electronics* 12.10 (Jan. 2023). Number: 10 Publisher: Multidisciplinary Digital Publishing Institute, p. 2179. ISSN: 2079-9292. DOI: 10.3390/electronics12102179. URL: https://www.mdpi.com/2079-9292/12/10/2179 (visited on 07/10/2023).

[20] Mariya Tauqeer et al. “Driver’s emotion and behavior classification system based on Internet of Things and deep learning for Advanced Driver Assistance System (ADAS)”. en. In: *Computer Communications* 194 (Oct. 2022), pp. 258–267. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2022.07.031. URL: https://www.

sciencedirect.com/science/article/pii/S014036642200278X (visited on 07/10/2023).

[21] Silvia Ceccacci, Andrea Generosi, and Andrea Castellano. "Designing in-car emotion-aware automation". In: *European Transport/Trasporti Europei* (Dec. 2021), pp. 1–15. DOI: 10.48295/ET.2021.84.5.

[22] Nishant Mukund Pawar, Nagendra R. Velaga, and R. B. Sharmila. "Exploring behavioral validity of driving simulator under time pressure driving conditions of professional drivers". en. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 89 (Aug. 2022), pp. 29–52. ISSN: 1369-8478. DOI: 10.1016/j.trf.2022.06.004. URL: https://www.sciencedirect.com/science/article/pii/S1369847822001267 (visited on 12/19/2022).

[23] *Vehicle-Hardware-In-The-Loop system for ADAS prototyping and validation*. en-US. URL: https://ieeexplore.ieee.org/abstract/document/6893229 (visited on 07/10/2023).

[24] Miguel Villanueva Fernández. *Own source*.

[25] *Clemson Vehicular Electronics Laboratory: Data Acquisition for Competition Vehicles*. URL: https://cecas.clemson.edu/cvel/auto/AuE835_Projects_2009/callies_project.html (visited on 07/15/2023).

[26] Boris Antić et al. *The Influence of Aggressive Driving Behavior and Impulsiveness on Traffic Accidents*. English. Tech. rep. June 2018. URL: https://rosap.ntl.bts.gov/view/dot/36298 (visited on 06/26/2023).

[27] Emilia M. Szumska and Rafał Jurecki. "The Effect of Aggressive Driving on Vehicle Parameters". en. In: *Energies* 13.24 (Jan. 2020). Number: 24 Publisher: Multidisciplinary Digital Publishing Institute, p. 6675. ISSN: 1996-1073. DOI: 10.3390/en13246675. URL: https://www.mdpi.com/1996-1073/13/24/6675 (visited on 06/26/2023).

[28] Arash Jahangiri, Vincent Berardi, and Sahar Ghanipoor Machiani. "Application of Real Field Connected Vehicle Data for Aggressive Driving Identification on

Horizontal Curves". In: *IEEE Transactions on Intelligent Transportation Systems* PP (Dec. 2017), pp. 1–9. DOI: 10.1109/TITS.2017.2768527.

[29] R. Sinha, W. R. Lovallo, and O. A. Parsons. "Cardiovascular differentiation of emotions". eng. In: *Psychosomatic Medicine* 54.4 (1992), pp. 422–435. ISSN: 0033-3174. DOI: 10.1097/00006842-199207000-00005.

[30] J. A. Calvo et al. "Influence of vehicle driving parameters on the noise caused by passenger cars in urban traffic". en. In: *Transportation Research Part D: Transport and Environment* 17.7 (Oct. 2012), pp. 509–513. ISSN: 1361-9209. DOI: 10.1016/j.trd.2012.06.002. URL: https://www.sciencedirect.com/science/article/pii/S1361920912000582 (visited on 06/27/2023).

[31] *Detectores de sonido – Mouser España*. URL: https://www.mouser.es/c/sensors/audio-sensors/?_gl=1*dg5jw4*_ga*dW5kZWZpbmVk*_ga_15W4STQT4T*dW5kZWZpbmVk*_ga_1KQLCYKRX3*dW5kZWZpbmVk (visited on 06/27/2023).

[32] *Anger and aggressive driving all the rage on our roads*. en-US. Nov. 2022. DOI: 10.1016/j.aap.2014.06.021;. URL: https://lens.monash.edu/@politics-society/2022/11/21/1385266?slug=anger-and-aggressive-driving-all-the-rage-on-our-roads (visited on 07/16/2023).

[33] Grigorios Kakkavas et al. "Design, Development, and Evaluation of 5G-Enabled Vehicular Services: The 5G-HEART Perspective". en. In: *Sensors* 22.2 (Jan. 2022). Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, p. 426. ISSN: 1424-8220. DOI: 10.3390/s22020426. URL: https://www.mdpi.com/1424-8220/22/2/426 (visited on 07/16/2023).

[34] Sreekrishna Pandi et al. "Joint Design of Communication and Control for Connected Cars in 5G Communication Systems". In: *2016 IEEE Globecom Workshops (GC Wkshps)*. Dec. 2016, pp. 1–7. DOI: 10.1109/GLOCOMW.2016.7848940.

[35] Fabio Giust et al. "Multi-Access Edge Computing: The Driver Behind the Wheel of 5G-Connected Cars". In: *IEEE Communications Standards Magazine* 2.3 (Sept. 2018). Conference Name: IEEE Communications Standards Magazine, pp. 66–73. ISSN: 2471-2833. DOI: 10.1109/MCOMSTD.2018.1800013.

[36] Fatma Raissi, Sami Yangui, and Frederic Camps. "Autonomous Cars, 5G Mobile Networks and Smart Cities: Beyond the Hype". In: *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. ISSN: 2641-8169. June 2019, pp. 180–185. DOI: 10.1109/WETICE.2019.00046.

[37] Dirk Hetzer et al. "5G connected and automated driving: use cases, technologies and trials in cross-border environments". In: *EURASIP Journal on Wireless Communications and Networking* 2021.1 (Apr. 2021), p. 97. ISSN: 1687-1499. DOI: 10.1186/s13638-021-01976-6. URL: https://doi.org/10.1186/s13638-021-01976-6 (visited on 07/10/2023).

[38] Hossam Elsayed, Bassem A. Abdullah, and Gamal Aly. "Fuzzy Logic Based Collision Avoidance System for Autonomous Navigation Vehicle". In: *2018 13th International Conference on Computer Engineering and Systems (ICCES)*. Dec. 2018, pp. 469–474. DOI: 10.1109/ICCES.2018.8639396.

[39] Till Hülnhagen et al. "Maneuver recognition using probabilistic finite-state machines and fuzzy logic". In: *2010 IEEE Intelligent Vehicles Symposium*. ISSN: 1931-0587. June 2010, pp. 65–70. DOI: 10.1109/IVS.2010.5548066.

[40] Mohammadreza Sarshar and Mahdi Rezaei. "A novel system for Advanced Driver Assistance Systems". In: *2010 IEEE International Systems Conference*. Apr. 2010, pp. 529–534. DOI: 10.1109/SYSTEMS.2010.5482319.

[41] Dave Elliott, Remco Polman, and Richard McGregor. "Relaxing Music for Anxiety Control". In: *Journal of Music Therapy* 48.3 (Oct. 2011), pp. 264–288. ISSN: 0022-2917. DOI: 10.1093/jmt/48.3.264. URL: https://doi.org/10.1093/jmt/48.3.264 (visited on 07/02/2023).

[42] Jonathan C. Smith and Carol A. Joyce. "Mozart versus New Age Music: Relaxation States, Stress, and ABC Relaxation Theory". In: *Journal of Music Therapy* 41.3 (Oct. 2004), pp. 215–224. ISSN: 0022-2917. DOI: 10.1093/jmt/41.3.215. URL: https://doi.org/10.1093/jmt/41.3.215 (visited on 07/02/2023).

[43] *RHYTHMJUJU FLOW - Live Music for Yoga - Marla Leigh - Frame Drums, World Percussion, Flutes, Composer, Educator Marla Leigh – Frame Drums, World Percussion, Flutes, Composer, Educator*. URL: https://marlaleigh.com/concert-drum-programs/yoga-studio-programs/rhythmjuju-flow-live-musicyoga/ (visited on 07/16/2023).

[44] Janet S Richmond et al. "Verbal De-escalation of the Agitated Patient: Consensus Statement of the American Association for Emergency Psychiatry Project BETA De-escalation Workgroup". In: *Western Journal of Emergency Medicine* 13.1 (Feb. 2012), pp. 17–25. ISSN: 1936-900X. DOI: 10.5811/westjem.2011.9.6864. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3298202/ (visited on 06/29/2023).

[45] A. Lazare, S. Eisenthal, and L. Wasserman. "The customer approach to patienthood. Attending to patient requests in a walk-in clinic". eng. In: *Archives of General Psychiatry* 32.5 (May 1975), pp. 553–558. ISSN: 0003-990X. DOI: 10.1001/archpsyc.1975.01760230019001.

[46] *De-escalation Skills Can Aid Frontline Workers | University of Nebraska Public Policy Center*. URL: https://ppc.unl.edu/de-escalation-skills-can-aid-frontline-workers (visited on 07/16/2023).

[47] Gary Duncan et al. "A Novel Simulation-Based Multidisciplinary Verbal De-escalation Training". en. In: *Cureus* 13.12 (Dec. 2021). Publisher: Cureus. ISSN: 2168-8184. DOI: 10.7759/cureus.20849. URL: https://www.cureus.com/articles/80214-a-novel-simulation-based-multidisciplinary-verbal-de-escalation-training (visited on 06/29/2023).

[48] Nathan Moore et al. "Designing Virtual Reality–Based Conversational Agents to Train Clinicians in Verbal De-escalation Skills: Exploratory Usability Study". EN. In: *JMIR Serious Games* 10.3 (July 2022). Company: JMIR Serious Games Distributor: JMIR Serious Games Institution: JMIR Serious Games Label: JMIR Serious Games Publisher: JMIR Publications Inc., Toronto, Canada, e38669. DOI: 10.2196/38669. URL: https://games.jmir.org/2022/3/e38669 (visited on 07/02/2023).

[49] Xiaoying Zhang et al. "Influences of Emotion on Driving Decisions at Different Risk Levels: An Eye Movement Study". In: *Frontiers in Psychology* 13 (2022). ISSN: 1664-1078. URL: https://www.frontiersin.org/articles/10.3389/fpsyg.2022.788712 (visited on 07/10/2023).

[50] *BeamNGpy — BeamNGpy documentation*. URL: https://beamngpy.readthedocs.io/en/latest/readme.html (visited on 12/04/2022).

[51] *ZeroMQ*. URL: https://zeromq.org/ (visited on 06/26/2023).

[52] *MQTT - The Standard for IoT Messaging*. URL: https://mqtt.org/ (visited on 06/26/2023).

[53] *What is MQTT*. URL: https://www.twilio.com/blog/what-is-mqtt (visited on 07/18/2023).

[54] *paho-mqtt: MQTT version 5.0/3.1.1 client class*. URL: http://eclipse.org/paho (visited on 07/11/2023).

[55] *Eclipse Mosquitto*. en. Jan. 2018. URL: https://mosquitto.org/ (visited on 07/12/2023).

[56] *eclipse-mosquitto - Official Image | Docker Hub*. URL: https://hub.docker.com/_/eclipse-mosquitto (visited on 07/12/2023).

[57] *Docker security*. en. July 2023. URL: https://docs.docker.com/engine/security/ (visited on 07/13/2023).

[58] *Deploy a Private 5G network in your campus*. en-US. URL: https://firecell.io/use-cases/deploy-a-private-5g-network-in-your-campus/ (visited on 07/12/2023).

[59] *RUTX50 - Industrial 5G Router*. URL: https://teltonika-networks.com/products/routers/rutx50 (visited on 07/12/2023).

[60] *Build Fuzzy Systems Using Fuzzy Logic Designer - MATLAB & Simulink - MathWorks España*. URL: https://es.mathworks.com/help/fuzzy/building-systems-with-fuzzy-logic-toolbox-software.html (visited on 12/19/2022).

[61] *Welcome to Python.org*. en. URL: https://www.python.org/ (visited on 12/19/2022).

[62] *Choose the best package - Snyk Open Source Advisor*. en. URL: https://snyk.io/advisor (visited on 07/03/2023).

[63] Marco S. Nobile. *simpful*. original-date: 2018-09-26T12:52:25Z. June 2023. URL: https://github.com/aresio/simpful (visited on 07/03/2023).

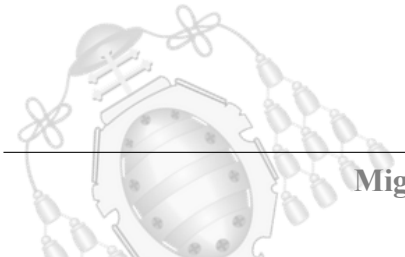
[64] Mohammad Sameer Ahmad and Christian Wagner. "JuzzyPy — A Python Library to Create Type—1, Interval Type-2 and General Type-2 Fuzzy Logic Systems". In: *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*. Dec. 2022, pp. 735–742. DOI: 10.1109/SSCI51031.2022.10022085.

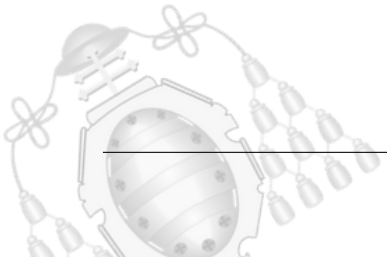
[65] *scikit-fuzzy: Fuzzy logic toolkit for SciPy*. URL: https://pypi.python.org/pypi/scikit-fuzzy (visited on 07/03/2023).

[66] *fuzzylogic: Fuzzy Logic for Python 3*. URL: https://github.com/amogorkon/fuzzylogic (visited on 07/03/2023).

[67] *playsound: Pure Python, cross platform, single function module with no dependencies for playing sounds*. URL: https://github.com/TaylorSMarks/playsound (visited on 07/13/2023).

[68] *Agencia Tributaria: 3.5.4 Tabla de amortización simplificada*. URL: https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/folleto-actividades-economicas/3-impuesto-sobre-renta-personas-fisicas/3_5-estimacion-directa-simplificada/3_5_4-tabla-amortizacion-simplificada.html (visited on 07/03/2023).





A. Code

Inside this annex, all created or modified code appears complete. The first chart shows ”audio_output.py” code, which serves the function to reproduce the calming audio files from receiving the different risk values.

```
from playsound import playsound
import time
import paho.mqtt.client as mqtt
import argparse

# ============================================================
# -- TFG functions -------------------------------------------
# ============================================================

n_audios = 3
interval = 15           # Waiting interval, in seconds
iteration = 0
risk_percentage = 0
risk_threshold = False
accel_list = []

def mqtt_connect(ip):   # Function to connect to the broker and subscribe to
    the topics
    mqtt_client = mqtt.Client()                     # Create the client
    mqtt_client.connect(ip, 1883, 60)               # Connect the client to
   the broker
    mqtt_client.loop_start()                        # Execute loop for
   receiving messages
    mqtt_client.subscribe("output/risk/percentage") # Subscribe to
   precentage risk value
    mqtt_client.subscribe("output/risk/threshold")  # Subscribe to threshold
    alert
```

```
23      mqtt_client.on_message=on_message               # What to do when
    receiving a message
24      return mqtt_client
25
26  def on_message(client, userdata, message):  # Assigns the rx value to the
    program variables
27      global risk_percentage
28      global risk_threshold
29      if (message.topic == "output/risk/percentage"):
30          risk_percentage = round(float(message.payload.decode("utf-8")))
31      if (message.topic == "output/risk/threshold"):
32          risk_threshold = (message.payload.decode("utf-8") == "True")
33
34  # =============================================================
35  # -- main() ---------------------------------------------------
36  # =============================================================
37
38  def main():
39      global iteration
40
41      argparser = argparse.ArgumentParser(
42          description='Emotion based ADAS audio output')
43      argparser.add_argument(
44          '--mqttip',
45          metavar='M',
46          default='127.0.0.1',
47          help='IP of the MQTT server (default: 127.0.0.1)')
48      argparser.add_argument(
49          '--path',
50          metavar='P',
51          default='/home/pi/CARLA/tfg_carla_control_api/',
52          help='Path to the audio files folder (default: /home/pi/CARLA/
    tfg_carla_control_api/)')
53      args = argparser.parse_args()
54
55      try:
56          mqtt_client = mqtt_connect(args.mqttip)
```

```
    except:
        print("Could not connect to the broker")
        exit()

    clock_seconds = round(time.time()/30)

    while (True):
        time.sleep(1)
        # Wait till 30 seconds have passed to execute this
        if (round(time.time()/interval) != clock_seconds):
            clock_seconds = round(time.time()/interval)

            # If the threshold indicator is active, play an audio
            if (risk_threshold):
                # Each time play a different audio
                iteration = iteration + 1
                print("Playing audio", iteration % 3)
                if (iteration % n_audios == 1):
                    playsound(args.path + 'audio1.mp3', True)
                elif (iteration % n_audios == 2):
                    playsound(args.path + 'audio2.mp3', True)
                elif (iteration % n_audios == 0):
                    playsound(args.path + 'audio3.mp3', True)
            # If the threshold is deactivated, congratulate the driver
            else:
                if (iteration != 0):
                    iteration = 0
                    print("Congratulation audio")
                    playsound(args.path + 'good1.mp3', True)


if __name__=="__main__":
  main()
```

Next, appears the developed code for the Fuzzy Logic processing system.

```
import time
import paho.mqtt.client as mqtt
from fuzzylogic.classes import Domain, Set, Rule
from fuzzylogic.hedges import very
from fuzzylogic.functions import R, S
from fuzzylogic.classes import rule_from_table
import matplotlib.pyplot as plt
import pandas as pd
#import numpy as np
import statistics as st
import math
import argparse

# ============================================================
# -- Variables ---------------------------------------------
# ============================================================

hr = 70
n = 15                  # Interval of evaluation -> 30 seconds
accel_list = [0] * n    # List of received accelerations (each one is 1
    second)
accel_stdev = 0         # Should change name?
accel_exceeded = 0      # The count of events in the
emotion_list = [""] * n  # List of received emotions (each one is 1 second)
emotion_dict = {"Enfadado":0, "Asco":1, "Asustado":2, "Feliz":3, "Triste":4,
     "Sorpresa":5, "Neutral":6}
risk_percentage = 0     # Risk estimation as a percentage
risk_threshold = False  # Risk threshold is exceeded

# ============================================================
# -- TFG functions -----------------------------------------
# ============================================================

def mqtt_connect(ip):
    mqtt_client = mqtt.Client()
    mqtt_client.connect(ip, 1883, 60)
    mqtt_client.loop_start()
```

```
36      mqtt_client.subscribe("sensors/sim/#")
37      mqtt_client.subscribe("sensors/cam/emotion")
38      mqtt_client.subscribe("sensors/h10/#")
39      mqtt_client.on_message=on_message
40      print("Connection to broker successful")
41      return mqtt_client
42
43  def on_message(client, userdata, message):
44      global hr
45      global accel_stdev
46      global accel_list
47      global emotion_list
48      try:
49          print("received message ", message.payload.decode("utf-8"), " with
    topic ", message.topic)
50          if (message.topic == "sensors/sim/totalacc"):
51              update_list(accel_list, float(message.payload.decode("utf-8")),
    30)
52              accel_stdev = st.stdev(accel_list)
53
54          if (message.topic == "sensors/cam/emotion"):
55              update_list(emotion_list, message.payload.decode("utf-8"), 30)
56
57          if (message.topic == "sensors/h10/pulse"):
58              hr = int(message.payload.decode("utf-8"))
59      except:
60          print("Could not process incoming message")
61
62  def update_list(list, element, target_length):
63      global accel_list
64      global emotion_list
65      list.insert(0, element)
66      if (len(list)>target_length):
67          list.pop()
68
69  def count_mean_list(list, element):
70      return (list.count(element)/len(list))*100
```

```
71
72
73 # ================================================================
74 # -- rules() ---------------------------------------------------
75 # ================================================================
76
77
78 def rules(values, accel, heart, emotion):
79     risk = Domain("Risk", 0, 100)
80
81     accel.low = S(1, 2)
82     accel.high = R(1, 2)
83     heart.low = S(60, 100)
84     heart.high = R(60, 100)
85     emotion.low = S(5, 50)
86     emotion.high = R(5, 50)
87
88     risk.low = S(0, 35)
89     risk.high = R(65, 100)
90     risk.medium =  risk.low &  risk.high
91
92     #accel.low.plot()
93     #accel.high.plot()
94     #plt.show()
95     #heart.low.plot()
96     #heart.high.plot()
97     #plt.show()
98     #emotion.low.plot()
99     #emotion.high.plot()
100    #plt.show()
101    #risk.low.plot()
102    #risk.medium.plot()
103    #risk.high.plot()
104    #plt.show()
105
106    R1 = Rule({(accel.low, heart.low, emotion.low): risk.low})
107    R2 = Rule({(accel.high, heart.low, emotion.low): risk.medium})
```

```
108     R3 = Rule({(accel.low, heart.high, emotion.low): risk.medium})
109     R4 = Rule({(accel.high, heart.high, emotion.low): risk.high})
110     R5 = Rule({(accel.low, heart.low, emotion.high): risk.medium})
111     R6 = Rule({(accel.high, heart.low, emotion.high): risk.high})
112     R7 = Rule({(accel.low, heart.high, emotion.high): risk.medium})
113     R8 = Rule({(accel.high, heart.high, emotion.high): risk.high})
114 
115     #rules = sum([R1, R2, R3, R4, R5, R6, R6, R7, R8, R9, R10, R11, R12, R13
        , R14, R15, R16])
116     rules = sum([R1, R2, R3, R4, R5, R6, R6, R7, R8])
117     return rules
118 
119 
120 # ================================================================
121 # -- main() ------------------------------------------------------
122 # ================================================================
123 
124 def main():
125     global hr
126     global accel_stdev
127 
128     argparser = argparse.ArgumentParser(
129         description='Emotion based ADAS FL processing system')
130     argparser.add_argument(
131         '--mqttip',
132         metavar='M',
133         default='127.0.0.1',
134         help='IP of the MQTT server (default: 127.0.0.1)')
135     args = argparser.parse_args()
136 
137     try:
138         mqtt_client = mqtt_connect(args.mqttip)
139     except:
140         print("Could not connect to the broker")
141         exit()
142 
143 
```

```
    accel = Domain("Accel standard deviation", 0, 20)
    heart = Domain("Heart Rate", 0, 300)
    emotion = Domain("Panic emotion", 0, 100)

    vals = {accel: accel_stdev, heart: hr, emotion: 0}
    fl_rules = rules(vals, accel, heart, emotion)

    while(True):
        vals = {accel: accel_stdev, heart: hr, emotion: round(
    count_mean_list(emotion_list, "Enfadado"))}
        risk = fl_rules(vals)
        print("=>", risk)
        mqtt_client.publish("output/risk/percentage", payload=risk, qos=0,
    retain=False)
        if (risk > 55):
            mqtt_client.publish("output/risk/threshold", payload="True", qos
    =0, retain=False)
        if (risk < 45):
            mqtt_client.publish("output/risk/threshold", payload="False",
    qos=0, retain=False)
        time.sleep(1)

    mqtt_client.loop_stop()



if __name__=="__main__":
  main()
```

Below can be seen the modified code for simulator client.

```
#!/usr/bin/env python

# Copyright (c) 2019 Intel Labs
#
# This work is licensed under the terms of the MIT license.
```

```
# For a copy, see <https://opensource.org/licenses/MIT>.

# Allows controlling a vehicle with a keyboard. For a simpler and more
# documented example, please take a look at tutorial.py.

"""
Welcome to CARLA manual control with steering wheel Logitech G29.

To drive start by preshing the brake pedal.
Change your wheel_config.ini according to your steering wheel.

To find out the values of your steering wheel use jstest-gtk in Ubuntu.

"""

from __future__ import print_function


# ==============================================================
# -- find carla module ---------------------------------------
# ==============================================================


import glob
import os
import sys

try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass


# ==============================================================
```

```
43 # -- imports ------------------------------------------------
44 # ================================================================
45
46 import paho.mqtt.client as mqtt
47
48 import carla
49
50 from carla import ColorConverter as cc
51
52 import argparse
53 import collections
54 import datetime
55 import logging
56 import math
57 import random
58 import re
59 import weakref
60
61 if sys.version_info >= (3, 0):
62
63     from configparser import ConfigParser
64
65 else:
66
67     from ConfigParser import RawConfigParser as ConfigParser
68
69 try:
70     import pygame
71     from pygame.locals import KMOD_CTRL
72     from pygame.locals import KMOD_SHIFT
73     from pygame.locals import K_0
74     from pygame.locals import K_9
75     from pygame.locals import K_BACKQUOTE
76     from pygame.locals import K_BACKSPACE
77     from pygame.locals import K_COMMA
78     from pygame.locals import K_DOWN
79     from pygame.locals import K_ESCAPE
```

```
80      from pygame.locals import K_F1
81      from pygame.locals import K_LEFT
82      from pygame.locals import K_PERIOD
83      from pygame.locals import K_RIGHT
84      from pygame.locals import K_SLASH
85      from pygame.locals import K_SPACE
86      from pygame.locals import K_TAB
87      from pygame.locals import K_UP
88      from pygame.locals import K_a
89      from pygame.locals import K_c
90      from pygame.locals import K_d
91      from pygame.locals import K_h
92      from pygame.locals import K_m
93      from pygame.locals import K_p
94      from pygame.locals import K_q
95      from pygame.locals import K_r
96      from pygame.locals import K_s
97      from pygame.locals import K_w
98  except ImportError:
99      raise RuntimeError('cannot import pygame, make sure pygame package is
    installed')
100
101 try:
102     import numpy as np
103 except ImportError:
104     raise RuntimeError('cannot import numpy, make sure numpy package is
    installed')
105
106
107 # ==============================================================
108 # -- Global functions ----------------------------------------
109 # ==============================================================
110
111
112 def find_weather_presets():
113     rgx = re.compile('.+?(?:(?<=[a-z])(?=[A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|$)'
    )
```

```
114     name = lambda x: ' '.join(m.group(0) for m in rgx.finditer(x))
115     presets = [x for x in dir(carla.WeatherParameters) if re.match('[A-Z].+'
        , x)]
116     return [(getattr(carla.WeatherParameters, x), name(x)) for x in presets]
117 
118 
119 def get_actor_display_name(actor, truncate=250):
120     name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
121     return (name[:truncate - 1] + u'\u2026') if len(name) > truncate else
        name
122 
123 
124 # ==============================================================
125 # -- World ---------------------------------------------------
126 # ==============================================================
127 
128 
129 class World(object):
130     def __init__(self, carla_world, hud, actor_filter):
131         self.world = carla_world
132         self.hud = hud
133         self.player = None
134         self.collision_sensor = None
135         self.lane_invasion_sensor = None
136         self.gnss_sensor = None
137         self.camera_manager = None
138         self._weather_presets = find_weather_presets()
139         self._weather_index = 0
140         self._actor_filter = actor_filter
141         self.restart()
142         self.world.on_tick(hud.on_world_tick)
143 
144     def restart(self):
145         # Keep same camera config if the camera manager exists.
146         cam_index = self.camera_manager.index if self.camera_manager is not
        None else 0
```

```
147         cam_pos_index = self.camera_manager.transform_index if self.
    camera_manager is not None else 0
148         # Get a random blueprint.
149         blueprint = random.choice(self.world.get_blueprint_library().filter(
    self._actor_filter))
150         blueprint.set_attribute('role_name', 'hero')
151         if blueprint.has_attribute('color'):
152             color = random.choice(blueprint.get_attribute('color').
    recommended_values)
153             blueprint.set_attribute('color', color)
154         # Spawn the player.
155         if self.player is not None:
156             spawn_point = self.player.get_transform()
157             spawn_point.location.z += 2.0
158             spawn_point.rotation.roll = 0.0
159             spawn_point.rotation.pitch = 0.0
160             self.destroy()
161             self.player = self.world.try_spawn_actor(blueprint, spawn_point)
162         while self.player is None:
163             spawn_points = self.world.get_map().get_spawn_points()
164             spawn_point = random.choice(spawn_points) if spawn_points else
    carla.Transform()
165             self.player = self.world.try_spawn_actor(blueprint, spawn_point)
166         # Set up the sensors.
167         self.collision_sensor = CollisionSensor(self.player, self.hud)
168         self.lane_invasion_sensor = LaneInvasionSensor(self.player, self.hud
    )
169         self.gnss_sensor = GnssSensor(self.player)
170         self.camera_manager = CameraManager(self.player, self.hud)
171         self.camera_manager.transform_index = cam_pos_index
172         self.camera_manager.set_sensor(cam_index, notify=False)
173         actor_type = get_actor_display_name(self.player)
174         self.hud.notification(actor_type)
175
176     def next_weather(self, reverse=False):
177         self._weather_index += -1 if reverse else 1
178         self._weather_index %= len(self._weather_presets)
```

```
179         preset = self._weather_presets[self._weather_index]
180         self.hud.notification('Weather: %s' % preset[1])
181         self.player.get_world().set_weather(preset[0])
182 
183     def tick(self, clock):
184         self.hud.tick(self, clock)
185 
186     def render(self, display):
187         self.camera_manager.render(display)
188         self.hud.render(display)
189 
190     def destroy(self):
191         sensors = [
192             self.camera_manager.sensor,
193             self.collision_sensor.sensor,
194             self.lane_invasion_sensor.sensor,
195             self.gnss_sensor.sensor]
196         for sensor in sensors:
197             if sensor is not None:
198                 sensor.stop()
199                 sensor.destroy()
200         if self.player is not None:
201             self.player.destroy()
202 
203 # ==============================================================
204 # -- DualControl ---------------------------------------------
205 # ==============================================================
206 
207 
208 class DualControl(object):
209     def __init__(self, world, start_in_autopilot):
210         self._autopilot_enabled = start_in_autopilot
211         if isinstance(world.player, carla.Vehicle):
212             self._control = carla.VehicleControl()
213             world.player.set_autopilot(self._autopilot_enabled)
214         elif isinstance(world.player, carla.Walker):
215             self._control = carla.WalkerControl()
```

```
216             self._autopilot_enabled = False
217             self._rotation = world.player.get_transform().rotation
218         else:
219             raise NotImplementedError("Actor type not supported")
220         self._steer_cache = 0.0
221         world.hud.notification("Press 'H' or '?' for help.", seconds=4.0)
222 
223         # initialize steering wheel
224         pygame.joystick.init()
225 
226         joystick_count = pygame.joystick.get_count()
227         if joystick_count > 1:
228             raise ValueError("Please Connect Just One Joystick")
229 
230         self._joystick = pygame.joystick.Joystick(0)
231         self._joystick.init()
232 
233         self._parser = ConfigParser()
234         self._parser.read('wheel_config.ini')
235         self._steer_idx = int(
236             self._parser.get('G29 Racing Wheel', 'steering_wheel'))
237         self._throttle_idx = int(
238             self._parser.get('G29 Racing Wheel', 'throttle'))
239         self._brake_idx = int(self._parser.get('G29 Racing Wheel', 'brake'))
240         self._reverse_idx = int(self._parser.get('G29 Racing Wheel', '
    reverse'))
241         self._handbrake_idx = int(
242             self._parser.get('G29 Racing Wheel', 'handbrake'))
243 
244     def parse_events(self, world, clock):
245         for event in pygame.event.get():
246             if event.type == pygame.QUIT:
247                 return True
248             elif event.type == pygame.JOYBUTTONDOWN:
249                 if event.button == 0:
250                     world.restart()
251                 elif event.button == 1:
```

```
252                     world.hud.toggle_info()
253                 elif event.button == 2:
254                     world.camera_manager.toggle_camera()
255                 elif event.button == 3:
256                     world.next_weather()
257                 elif event.button == self._reverse_idx:
258                     self._control.gear = 1 if self._control.reverse else -1
259                 elif event.button == 23:
260                     world.camera_manager.next_sensor()
261 
262             elif event.type == pygame.KEYUP:
263                 if self._is_quit_shortcut(event.key):
264                     return True
265                 elif event.key == K_BACKSPACE:
266                     world.restart()
267                 elif event.key == K_F1:
268                     world.hud.toggle_info()
269                 elif event.key == K_h or (event.key == K_SLASH and pygame.
    key.get_mods() & KMOD_SHIFT):
270                     world.hud.help.toggle()
271                 elif event.key == K_TAB:
272                     world.camera_manager.toggle_camera()
273                 elif event.key == K_c and pygame.key.get_mods() & KMOD_SHIFT
    :
274                     world.next_weather(reverse=True)
275                 elif event.key == K_c:
276                     world.next_weather()
277                 elif event.key == K_BACKQUOTE:
278                     world.camera_manager.next_sensor()
279                 elif event.key > K_0 and event.key <= K_9:
280                     world.camera_manager.set_sensor(event.key - 1 - K_0)
281                 elif event.key == K_r:
282                     world.camera_manager.toggle_recording()
283                 if isinstance(self._control, carla.VehicleControl):
284                     if event.key == K_q:
285                         self._control.gear = 1 if self._control.reverse else
     -1
```

```
286                     elif event.key == K_m:
287                         self._control.manual_gear_shift = not self._control.
    manual_gear_shift
288                         self._control.gear = world.player.get_control().gear
289                         world.hud.notification('%s Transmission' %
290                                                ('Manual' if self._control.
    manual_gear_shift else 'Automatic'))
291                     elif self._control.manual_gear_shift and event.key ==
    K_COMMA:
292                         self._control.gear = max(-1, self._control.gear - 1)
293                     elif self._control.manual_gear_shift and event.key ==
    K_PERIOD:
294                         self._control.gear = self._control.gear + 1
295                     elif event.key == K_p:
296                         self._autopilot_enabled = not self.
    _autopilot_enabled
297                         world.player.set_autopilot(self._autopilot_enabled)
298                         world.hud.notification('Autopilot %s' % ('On' if
    self._autopilot_enabled else 'Off'))
299
300         if not self._autopilot_enabled:
301             if isinstance(self._control, carla.VehicleControl):
302                 self._parse_vehicle_keys(pygame.key.get_pressed(), clock.
    get_time())
303                 self._parse_vehicle_wheel()
304                 self._control.reverse = self._control.gear < 0
305             elif isinstance(self._control, carla.WalkerControl):
306                 self._parse_walker_keys(pygame.key.get_pressed(), clock.
    get_time())
307             world.player.apply_control(self._control)
308
309     def _parse_vehicle_keys(self, keys, milliseconds):
310         self._control.throttle = 1.0 if keys[K_UP] or keys[K_w] else 0.0
311         steer_increment = 5e-4 * milliseconds
312         if keys[K_LEFT] or keys[K_a]:
313             self._steer_cache -= steer_increment
314         elif keys[K_RIGHT] or keys[K_d]:
```

```
315             self._steer_cache += steer_increment
316         else:
317             self._steer_cache = 0.0
318         self._steer_cache = min(0.7, max(-0.7, self._steer_cache))
319         self._control.steer = round(self._steer_cache, 1)
320         self._control.brake = 1.0 if keys[K_DOWN] or keys[K_s] else 0.0
321         self._control.hand_brake = keys[K_SPACE]
322 
323     def _parse_vehicle_wheel(self):
324         numAxes = self._joystick.get_numaxes()
325         jsInputs = [float(self._joystick.get_axis(i)) for i in range(numAxes
    )]
326         # print (jsInputs)
327         jsButtons = [float(self._joystick.get_button(i)) for i in
328                      range(self._joystick.get_numbuttons())]
329 
330         # Custom function to map range of inputs [1, -1] to outputs [0, 1] i
    .e 1 from inputs means nothing is pressed
331         # For the steering, it seems fine as it is
332         K1 = 1.0  # 0.55
333         steerCmd = K1 * math.tan(0.8 * jsInputs[self._steer_idx])
334 
335         K2 = 1.6  # 1.6
336         throttleCmd = K2 + (2.05 * math.log10(
337             -0.7 * jsInputs[self._throttle_idx] + 1.4) - 1.2) / 0.92
338         if throttleCmd <= 0:
339             throttleCmd = 0
340         elif throttleCmd > 1:
341             throttleCmd = 1
342 
343         brakeCmd = 1.6 + (2.05 * math.log10(
344             -0.7 * jsInputs[self._brake_idx] + 1.4) - 1.2) / 0.92
345         if brakeCmd <= 0:
346             brakeCmd = 0
347         elif brakeCmd > 1:
348             brakeCmd = 1
349 
```

```
350         self._control.steer = steerCmd
351         self._control.brake = brakeCmd
352         self._control.throttle = throttleCmd
353 
354         #toggle = jsButtons[self._reverse_idx]
355 
356         self._control.hand_brake = bool(jsButtons[self._handbrake_idx])
357 
358     def _parse_walker_keys(self, keys, milliseconds):
359         self._control.speed = 0.0
360         if keys[K_DOWN] or keys[K_s]:
361             self._control.speed = 0.0
362         if keys[K_LEFT] or keys[K_a]:
363             self._control.speed = .01
364             self._rotation.yaw -= 0.08 * milliseconds
365         if keys[K_RIGHT] or keys[K_d]:
366             self._control.speed = .01
367             self._rotation.yaw += 0.08 * milliseconds
368         if keys[K_UP] or keys[K_w]:
369             self._control.speed = 5.556 if pygame.key.get_mods() &
    KMOD_SHIFT else 2.778
370         self._control.jump = keys[K_SPACE]
371         self._rotation.yaw = round(self._rotation.yaw, 1)
372         self._control.direction = self._rotation.get_forward_vector()
373 
374     @staticmethod
375     def _is_quit_shortcut(key):
376         return (key == K_ESCAPE) or (key == K_q and pygame.key.get_mods() &
    KMOD_CTRL)
377 
378 
379 # ==============================================================
380 # -- HUD -----------------------------------------------------
381 # ==============================================================
382 
383 
384 class HUD(object):
```

```
385     risk_percentage = 0
386     risk_threshold = False
387 
388     def __init__(self, width, height):
389         self.dim = (width, height)
390         font = pygame.font.Font(pygame.font.get_default_font(), 20)
391         font_name = 'courier' if os.name == 'nt' else 'mono'
392         fonts = [x for x in pygame.font.get_fonts() if font_name in x]
393         default_font = 'ubuntumono'
394         mono = default_font if default_font in fonts else fonts[0]
395         mono = pygame.font.match_font(mono)
396         self._font_mono = pygame.font.Font(mono, 12 if os.name == 'nt' else
    14)
397         self._notifications = FadingText(font, (width, 40), (0, height - 40)
    )
398         self.help = HelpText(pygame.font.Font(mono, 24), width, height)
399         self.server_fps = 0
400         self.frame = 0
401         self.simulation_time = 0
402         self._show_info = True
403         self._info_text = []
404         self._server_clock = pygame.time.Clock()
405 
406     def on_world_tick(self, timestamp):
407         self._server_clock.tick()
408         self.server_fps = self._server_clock.get_fps()
409         self.frame = timestamp.frame
410         self.simulation_time = timestamp.elapsed_seconds
411 
412     def tick(self, world, clock):
413         self._notifications.tick(world, clock)
414         if not self._show_info:
415             return
416         t = world.player.get_transform()
417         v = world.player.get_velocity()
418         a = world.player.get_acceleration()
419         c = world.player.get_control()
```

```
420         heading = 'N' if abs(t.rotation.yaw) < 89.5 else ''
421         heading += 'S' if abs(t.rotation.yaw) > 90.5 else ''
422         heading += 'E' if 179.5 > t.rotation.yaw > 0.5 else ''
423         heading += 'W' if -0.5 > t.rotation.yaw > -179.5 else ''
424         colhist = world.collision_sensor.get_collision_history()
425         collision = [colhist[x + self.frame - 200] for x in range(0, 200)]
426         max_col = max(1.0, max(collision))
427         collision = [x / max_col for x in collision]
428         vehicles = world.world.get_actors().filter('vehicle.*')
429         self._info_text = [
430             'Server:  % 16.0f FPS' % self.server_fps,
431             'Client:  % 16.0f FPS' % clock.get_fps(),
432             '',
433             'Vehicle: % 20s' % get_actor_display_name(world.player, truncate
    =20),
434             'Map:     % 20s' % world.world.get_map().name.split('/')[-1],
435             'Simulation time: % 12s' % datetime.timedelta(seconds=int(self.
    simulation_time)),
436             '',
437             'Speed:   % 15.0f km/h' % (3.6 * math.sqrt(v.x**2 + v.y**2 + v.z
    **2)),
438             u'Heading:% 16.0f\N{DEGREE SIGN} % 2s' % (t.rotation.yaw,
    heading),
439             'Location:% 20s' % ('(% 5.1f, % 5.1f)' % (t.location.x, t.
    location.y)),
440             'GNSS:% 24s' % ('(% 2.6f, % 3.6f)' % (world.gnss_sensor.lat,
    world.gnss_sensor.lon)),
441             'Height:  % 18.0f m' % t.location.z,
442             'Acceleration:  % 8.0f m/s^2' % (round(a.length())),      #
    Acceleration
443             'Risk %%: % 19.0f %%' % self.risk_percentage,           # Risk %
444             '']
445         if isinstance(c, carla.VehicleControl):
446             self._info_text += [
447                 ('Throttle:', c.throttle, 0.0, 1.0),
448                 ('Steer:', c.steer, -1.0, 1.0),
449                 ('Brake:', c.brake, 0.0, 1.0),
```

```
450                 ('Reverse:', c.reverse),
451                 ('Hand brake:', c.hand_brake),
452                 ('Manual:', c.manual_gear_shift),
453                 ('RISK:', self.risk_threshold), # Risk pilot, activated with
    threshold message
454                 'Gear:        %s' % {-1: 'R', 0: 'N'}.get(c.gear, c.gear)]
455         elif isinstance(c, carla.WalkerControl):
456             self._info_text += [
457                 ('Speed:', c.speed, 0.0, 5.556),
458                 ('Jump:', c.jump)]
459         self._info_text += [
460             '',
461             'Collision:',
462             collision,
463             '',
464             'Number of vehicles: % 8d' % len(vehicles)]
465         if len(vehicles) > 1:
466             self._info_text += ['Nearby vehicles:']
467             distance = lambda l: math.sqrt((l.x - t.location.x)**2 + (l.y -
    t.location.y)**2 + (l.z - t.location.z)**2)
468             vehicles = [(distance(x.get_location()), x) for x in vehicles if
     x.id != world.player.id]
469             for d, vehicle in sorted(vehicles):
470                 if d > 200.0:
471                     break
472                 vehicle_type = get_actor_display_name(vehicle, truncate=22)
473                 self._info_text.append('% 4dm %s' % (d, vehicle_type))
474 
475     def toggle_info(self):
476         self._show_info = not self._show_info
477 
478     def notification(self, text, seconds=2.0):
479         self._notifications.set_text(text, seconds=seconds)
480 
481     def error(self, text):
482         self._notifications.set_text('Error: %s' % text, (255, 0, 0))
483 
```

```
484     def render(self, display):
485         if self._show_info:
486             info_surface = pygame.Surface((220, self.dim[1]))
487             info_surface.set_alpha(100)
488             display.blit(info_surface, (0, 0))
489             v_offset = 4
490             bar_h_offset = 100
491             bar_width = 106
492             for item in self._info_text:
493                 if v_offset + 18 > self.dim[1]:
494                     break
495                 if isinstance(item, list):
496                     if len(item) > 1:
497                         points = [(x + 8, v_offset + 8 + (1.0 - y) * 30) for
 x, y in enumerate(item)]
498                         pygame.draw.lines(display, (255, 136, 0), False,
points, 2)
499                     item = None
500                     v_offset += 18
501                 elif isinstance(item, tuple):
502                     if isinstance(item[1], bool):
503                         rect = pygame.Rect((bar_h_offset, v_offset + 8), (6,
 6))
504                         pygame.draw.rect(display, (255, 255, 255), rect, 0
if item[1] else 1)
505                     else:
506                         rect_border = pygame.Rect((bar_h_offset, v_offset +
8), (bar_width, 6))
507                         pygame.draw.rect(display, (255, 255, 255),
rect_border, 1)
508                         f = (item[1] - item[2]) / (item[3] - item[2])
509                         if item[2] < 0.0:
510                             rect = pygame.Rect((bar_h_offset + f * (
bar_width - 6), v_offset + 8), (6, 6))
511                         else:
512                             rect = pygame.Rect((bar_h_offset, v_offset + 8),
 (f * bar_width, 6))
```

```
513                         pygame.draw.rect(display, (255, 255, 255), rect)
514                     item = item[0]
515                 if item:  # At this point has to be a str.
516                     surface = self._font_mono.render(item, True, (255, 255,
    255))
517                     display.blit(surface, (8, v_offset))
518                 v_offset += 18
519         self._notifications.render(display)
520         self.help.render(display)
521
522
523 # ==============================================================
524 # -- FadingText ----------------------------------------------
525 # ==============================================================
526
527
528 class FadingText(object):
529     def __init__(self, font, dim, pos):
530         self.font = font
531         self.dim = dim
532         self.pos = pos
533         self.seconds_left = 0
534         self.surface = pygame.Surface(self.dim)
535
536     def set_text(self, text, color=(255, 255, 255), seconds=2.0):
537         text_texture = self.font.render(text, True, color)
538         self.surface = pygame.Surface(self.dim)
539         self.seconds_left = seconds
540         self.surface.fill((0, 0, 0, 0))
541         self.surface.blit(text_texture, (10, 11))
542
543     def tick(self, _, clock):
544         delta_seconds = 1e-3 * clock.get_time()
545         self.seconds_left = max(0.0, self.seconds_left - delta_seconds)
546         self.surface.set_alpha(500.0 * self.seconds_left)
547
548     def render(self, display):
```

```
549         display.blit(self.surface, self.pos)
550 
551 
552 # ==============================================================
553 # -- HelpText ------------------------------------------------
554 # ==============================================================
555 
556 
557 class HelpText(object):
558     def __init__(self, font, width, height):
559         lines = __doc__.split('\n')
560         self.font = font
561         self.dim = (680, len(lines) * 22 + 12)
562         self.pos = (0.5 * width - 0.5 * self.dim[0], 0.5 * height - 0.5 *
    self.dim[1])
563         self.seconds_left = 0
564         self.surface = pygame.Surface(self.dim)
565         self.surface.fill((0, 0, 0, 0))
566         for n, line in enumerate(lines):
567             text_texture = self.font.render(line, True, (255, 255, 255))
568             self.surface.blit(text_texture, (22, n * 22))
569             self._render = False
570         self.surface.set_alpha(220)
571 
572     def toggle(self):
573         self._render = not self._render
574 
575     def render(self, display):
576         if self._render:
577             display.blit(self.surface, self.pos)
578 
579 
580 # ==============================================================
581 # -- CollisionSensor -----------------------------------------
582 # ==============================================================
583 
584 
```

```
585 class CollisionSensor(object):
586     def __init__(self, parent_actor, hud):
587         self.sensor = None
588         self.history = []
589         self._parent = parent_actor
590         self.hud = hud
591         world = self._parent.get_world()
592         bp = world.get_blueprint_library().find('sensor.other.collision')
593         self.sensor = world.spawn_actor(bp, carla.Transform(), attach_to=
    self._parent)
594         # We need to pass the lambda a weak reference to self to avoid
    circular
595         # reference.
596         weak_self = weakref.ref(self)
597         self.sensor.listen(lambda event: CollisionSensor._on_collision(
    weak_self, event))
598 
599     def get_collision_history(self):
600         history = collections.defaultdict(int)
601         for frame, intensity in self.history:
602             history[frame] += intensity
603         return history
604 
605     @staticmethod
606     def _on_collision(weak_self, event):
607         self = weak_self()
608         if not self:
609             return
610         actor_type = get_actor_display_name(event.other_actor)
611         self.hud.notification('Collision with %r' % actor_type)
612         impulse = event.normal_impulse
613         intensity = math.sqrt(impulse.x**2 + impulse.y**2 + impulse.z**2)
614         self.history.append((event.frame, intensity))
615         if len(self.history) > 4000:
616             self.history.pop(0)
617 
618 
```

```
619 # ==============================================================
620 # -- LaneInvasionSensor ---------------------------------------
621 # ==============================================================
622
623
624 class LaneInvasionSensor(object):
625     def __init__(self, parent_actor, hud):
626         self.sensor = None
627         self._parent = parent_actor
628         self.hud = hud
629         world = self._parent.get_world()
630         bp = world.get_blueprint_library().find('sensor.other.lane_invasion'
    )
631         self.sensor = world.spawn_actor(bp, carla.Transform(), attach_to=
    self._parent)
632         # We need to pass the lambda a weak reference to self to avoid
    circular
633         # reference.
634         weak_self = weakref.ref(self)
635         self.sensor.listen(lambda event: LaneInvasionSensor._on_invasion(
    weak_self, event))
636
637     @staticmethod
638     def _on_invasion(weak_self, event):
639         self = weak_self()
640         if not self:
641             return
642         lane_types = set(x.type for x in event.crossed_lane_markings)
643         text = ['%r' % str(x).split()[-1] for x in lane_types]
644         self.hud.notification('Crossed line %s' % ' and '.join(text))
645
646 # ==============================================================
647 # -- GnssSensor -----------------------------------------------
648 # ==============================================================
649
650
651 class GnssSensor(object):
```

```
652     def __init__(self, parent_actor):
653         self.sensor = None
654         self._parent = parent_actor
655         self.lat = 0.0
656         self.lon = 0.0
657         world = self._parent.get_world()
658         bp = world.get_blueprint_library().find('sensor.other.gnss')
659         self.sensor = world.spawn_actor(bp, carla.Transform(carla.Location(x
    =1.0, z=2.8)), attach_to=self._parent)
660         # We need to pass the lambda a weak reference to self to avoid
    circular
661         # reference.
662         weak_self = weakref.ref(self)
663         self.sensor.listen(lambda event: GnssSensor._on_gnss_event(weak_self
    , event))
664 
665     @staticmethod
666     def _on_gnss_event(weak_self, event):
667         self = weak_self()
668         if not self:
669             return
670         self.lat = event.latitude
671         self.lon = event.longitude
672 
673 
674 # ==============================================================
675 # -- CameraManager -------------------------------------------
676 # ==============================================================
677 
678 
679 class CameraManager(object):
680     def __init__(self, parent_actor, hud):
681         self.sensor = None
682         self.surface = None
683         self._parent = parent_actor
684         self.hud = hud
685         self.recording = False
```

```
686         self._camera_transforms = [
687             carla.Transform(carla.Location(x=-5.5, z=2.8), carla.Rotation(
    pitch=-15)),
688             carla.Transform(carla.Location(x=1.6, z=1.7))]
689         self.transform_index = 1
690         self.sensors = [
691             ['sensor.camera.rgb', cc.Raw, 'Camera RGB'],
692             ['sensor.camera.depth', cc.Raw, 'Camera Depth (Raw)'],
693             ['sensor.camera.depth', cc.Depth, 'Camera Depth (Gray Scale)'],
694             ['sensor.camera.depth', cc.LogarithmicDepth, 'Camera Depth (
    Logarithmic Gray Scale)'],
695             ['sensor.camera.semantic_segmentation', cc.Raw, 'Camera Semantic
     Segmentation (Raw)'],
696             ['sensor.camera.semantic_segmentation', cc.CityScapesPalette,
697                 'Camera Semantic Segmentation (CityScapes Palette)'],
698             ['sensor.lidar.ray_cast', None, 'Lidar (Ray-Cast)']]
699         world = self._parent.get_world()
700         bp_library = world.get_blueprint_library()
701         for item in self.sensors:
702             bp = bp_library.find(item[0])
703             if item[0].startswith('sensor.camera'):
704                 bp.set_attribute('image_size_x', str(hud.dim[0]))
705                 bp.set_attribute('image_size_y', str(hud.dim[1]))
706             elif item[0].startswith('sensor.lidar'):
707                 bp.set_attribute('range', '50')
708             item.append(bp)
709         self.index = None
710 
711     def toggle_camera(self):
712         self.transform_index = (self.transform_index + 1) % len(self.
    _camera_transforms)
713         self.sensor.set_transform(self._camera_transforms[self.
    transform_index])
714 
715     def set_sensor(self, index, notify=True):
716         index = index % len(self.sensors)
717         needs_respawn = True if self.index is None \
```

```
718             else self.sensors[index][0] != self.sensors[self.index][0]
719         if needs_respawn:
720             if self.sensor is not None:
721                 self.sensor.destroy()
722                 self.surface = None
723             self.sensor = self._parent.get_world().spawn_actor(
724                 self.sensors[index][-1],
725                 self._camera_transforms[self.transform_index],
726                 attach_to=self._parent)
727             # We need to pass the lambda a weak reference to self to avoid
728             # circular reference.
729             weak_self = weakref.ref(self)
730             self.sensor.listen(lambda image: CameraManager._parse_image(
    weak_self, image))
731         if notify:
732             self.hud.notification(self.sensors[index][2])
733         self.index = index
734 
735     def next_sensor(self):
736         self.set_sensor(self.index + 1)
737 
738     def toggle_recording(self):
739         self.recording = not self.recording
740         self.hud.notification('Recording %s' % ('On' if self.recording else
    'Off'))
741 
742     def render(self, display):
743         if self.surface is not None:
744             display.blit(self.surface, (0, 0))
745 
746     @staticmethod
747     def _parse_image(weak_self, image):
748         self = weak_self()
749         if not self:
750             return
751         if self.sensors[self.index][0].startswith('sensor.lidar'):
752             points = np.frombuffer(image.raw_data, dtype=np.dtype('f4'))
```

```
753             points = np.reshape(points, (int(points.shape[0] / 4), 4))
754             lidar_data = np.array(points[:, :2])
755             lidar_data *= min(self.hud.dim) / 100.0
756             lidar_data += (0.5 * self.hud.dim[0], 0.5 * self.hud.dim[1])
757             lidar_data = np.fabs(lidar_data) # pylint: disable=E1111
758             lidar_data = lidar_data.astype(np.int32)
759             lidar_data = np.reshape(lidar_data, (-1, 2))
760             lidar_img_size = (self.hud.dim[0], self.hud.dim[1], 3)
761             lidar_img = np.zeros(lidar_img_size)
762             lidar_img[tuple(lidar_data.T)] = (255, 255, 255)
763             self.surface = pygame.surfarray.make_surface(lidar_img)
764         else:
765             image.convert(self.sensors[self.index][1])
766             array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
767             array = np.reshape(array, (image.height, image.width, 4))
768             array = array[:, :, :3]
769             array = array[:, :, ::-1]
770             self.surface = pygame.surfarray.make_surface(array.swapaxes(0,
    1))
771         if self.recording:
772             image.save_to_disk('_out/%08d' % image.frame)
773
774
775 # ==============================================================
776 # -- TFG functions -------------------------------------------
777 # ==============================================================
778
779 risk_percentage = 0
780 risk_threshold = False
781 clock_seconds = 0
782 accel_list = []
783
784 def mqtt_connect(ip):   # Function to connect to the broker and subscribe to
     the topics
785     mqtt_client = mqtt.Client()                    # Create the client
786     mqtt_client.connect(ip, 1883, 60)              # Connect the client to
    the broker
```

```
787     mqtt_client.loop_start()                        # Execute loop for
    receiving messages
788     mqtt_client.subscribe("output/risk/percentage") # Subscribe to
    precentage risk value
789     mqtt_client.subscribe("output/risk/threshold")  # Subscribe to threshold
     alert
790     mqtt_client.on_message=on_message               # What to do when
    receiving a message
791     return mqtt_client
792
793 def on_message(client, userdata, message):  # Assigns the rx value to the
    program variables
794     global risk_percentage
795     global risk_threshold
796     if (message.topic == "output/risk/percentage"):
797         risk_percentage = round(float(message.payload.decode("utf-8")))
798     if (message.topic == "output/risk/threshold"):
799         risk_threshold = (message.payload.decode("utf-8") == "True")
800
801
802 # ============================================================
803 # -- game_loop() ---------------------------------------------
804 # ============================================================
805
806
807 def game_loop(args):
808     # Needed variables inside the game loop
809     global risk_percentage
810     global risk_threshold
811     global clock_seconds
812     global accel_list
813
814     pygame.init()
815     pygame.font.init()
816     world = None
817
818     try:
```

```
819         client = carla.Client(args.host, args.port)
820         client.set_timeout(2.0)
821 
822         # MQTT broker connection function
823         try:
824             mqtt_client = mqtt_connect(args.mqttip)
825         except:
826             print("Couldn't connect to MQTT broker")
827             exit()
828 
829         display = pygame.display.set_mode(
830             (args.width, args.height),
831             pygame.HWSURFACE | pygame.DOUBLEBUF)
832 
833         hud = HUD(args.width, args.height)
834         world = World(client.get_world(), hud, args.filter)
835         controller = DualControl(world, args.autopilot)
836 
837         clock = pygame.time.Clock()
838         while True:
839             clock.tick_busy_loop(60)
840             if controller.parse_events(world, clock):
841                 return
842             world.tick(clock)
843             world.render(display)
844             pygame.display.flip()
845 
846             # Gets the acceleration value for each fps
847             a = world.player.get_acceleration()
848             tot_accel = a.length()
849             accel_list.append(tot_accel)
850 
851             # Send average acceleration value each second
852             if (round(pygame.time.get_ticks() / 1000) != clock_seconds):#
    Executes every second
853                 accel_avg = sum(accel_list)/len(accel_list)             #
    Calculate the average acceleration
```

```
854                 accel_list = []                                         #
    Empty the list
855                 # Send the acceleration value in a MQTT message
856                 mqtt_client.publish("sensors/sim/totalacc", payload=
    accel_avg, qos=0, retain=False)
857                 clock_seconds = round(pygame.time.get_ticks() / 1000)   #
    Waits for a new second
858
859             # Update HUD with received values from MQTT
860             hud.risk_percentage = risk_percentage
861             hud.risk_threshold = risk_threshold
862
863     finally:
864
865         if world is not None:
866             world.destroy()
867
868         pygame.quit()
869
870
871 # ==============================================================
872 # -- main() ----------------------------------------------------
873 # ==============================================================
874
875
876 def main():
877     argparser = argparse.ArgumentParser(
878         description='CARLA Manual Control Client')
879     argparser.add_argument(
880         '-v', '--verbose',
881         action='store_true',
882         dest='debug',
883         help='print debug information')
884     argparser.add_argument(
885         '--host',
886         metavar='H',
887         default='127.0.0.1',
```

```
888         help='IP of the host server (default: 127.0.0.1)')
889     argparser.add_argument(
890         '-p', '--port',
891         metavar='P',
892         default=2000,
893         type=int,
894         help='TCP port to listen to (default: 2000)')
895     argparser.add_argument(
896         '-a', '--autopilot',
897         action='store_true',
898         help='enable autopilot')
899     argparser.add_argument(
900         '--res',
901         metavar='WIDTHxHEIGHT',
902         default='1280x720',
903         help='window resolution (default: 1280x720)')
904     argparser.add_argument(
905         '--filter',
906         metavar='PATTERN',
907         default='vehicle.*',
908         help='actor filter (default: "vehicle.*")')
909     argparser.add_argument(
910         '--mqttip',
911         metavar='M',
912         default='127.0.0.1',
913         help='IP of the MQTT server (default: 127.0.0.1)')
914     args = argparser.parse_args()
915 
916     args.width, args.height = [int(x) for x in args.res.split('x')]
917 
918     log_level = logging.DEBUG if args.debug else logging.INFO
919     logging.basicConfig(format='%(levelname)s: %(message)s', level=log_level
    )
920 
921     logging.info('listening to server %s:%s', args.host, args.port)
922 
923     print(__doc__)
```

```
924
925     try:
926
927         game_loop(args)
928
929     except KeyboardInterrupt:
930         print('\nCancelled by user. Bye!')
931
932
933 if __name__ == '__main__':
934
935     main()
```

In continuation, the configuration file for the steering wheel appears.

```
1  [G29Pedal]
2  steering_wheel = 0
3  throttle = 3
4  brake = 2
5  sensitivity = 0.5
6  k1 = 2
7  k2 = 1.6
8
9  [G29WheelButton]
10 # Circle
11 handbrake = 2
12 # Right flap
13 reverse = 4
14
15 [G29 Racing Wheel]
16 steering_wheel = 0
17 clutch = 3
18 throttle = 1
19 brake = 2
20 handbrake = 4
21 reverse = 5
```

In the next chart, the modified code from Ignacio's TFG to detect emotions using just a webcam appears.

```
#import keyboard
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.python.keras.models import load_model
import os
import time
import _thread
import paho.mqtt.client as mqtt

### CONFIGURACIÓN ###
# Umbral de confianza de las respuestas (0-0.99)
ACCURACY = 0.10
# Se muestran las precisiones (True) de lo contrario (False) // Multi: mostrar una o varias caras
ACC_ACTIVATED = False
MULTI = True
# Archivo de modelo a utilizar
model='./modelos/modelo_CNN.hdf5'
CNN_model = tf.keras.models.load_model(model)

# Diccionario con las emociones
emotion_dict = {0: "Enfadado", 1: "Asco", 2: "Asustado", 3: "Feliz", 4: "Triste", 5: "Sorpresa", 6: "Neutral"}
color_dict = {0: (96, 100, 216), 1: (203, 3, 129), 2: (20, 20, 255), 3: (0,255, 87), 4: (244, 59, 69), 5: (184, 59, 244), 6: (199, 244, 59)}
emotion_string = ""
acc=""
state = True

# FPS Variables
prev_frame_time = 0
new_frame_time = 0
```

```
31
32 # Se activa la cámara
33 cap = cv2.VideoCapture(0)
34
35 ################################
36
37 # Some needed variables are defined
38 emotion_list = []
39 clock_seconds = 0
40 mqtt_broker_ip = "127.0.0.1"
41
42 # MQTT broker connection
43 try:
44     client = mqtt.Client()
45     client.connect(mqtt_broker_ip, 1883, 60)
46     print("Broker connection successful")
47 except:
48     print("Could not connect to the broker")
49
50
51 while(state):
52     ret, frame = cap.read()
53
54     # Si no hay imagen, se termina el bucle
55     if not ret:
56         print('Error, no image')
57         time.sleep(2)
58         break
59
60     haarcascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml
       ')
61     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
62     faces = haarcascade.detectMultiScale(gray, 1.3, 5)
63
64     for (x, y, w, h) in faces:
65         new_frame_time = time.time()
66         roi_gray = gray[y:y + h, x:x + w]
```

```
67
68          # Adaptacion dimensión
69          image = cv2.resize(roi_gray, (48, 48))
70          np_image_array = image.astype('float32') / 255.0
71          img_ready = np.expand_dims(np.expand_dims(np_image_array, -1), 0)
72
73          # Inferencia
74          result = CNN_model.predict(img_ready)
75          emotion_number = int(np.argmax(result))
76
77          if(np.max(result) > ACCURACY):
78              # Actualización panel led
79              if(emotion_dict[emotion_number] != emotion_string):
80                  emotion_string = emotion_dict[emotion_number]
81                  #client.publish("sensores/cam/emotion", emotion_string, qos
    =0, retain=False)
82
83          else:
84              if(emotion_dict[emotion_number] != emotion_string):
85                  emotion_string="¿?"
86                  emotion_string = "¿?"
87
88          #Cálculo de FPS
89          fps = 1/(new_frame_time-prev_frame_time)
90          prev_frame_time = new_frame_time
91          fps = int(fps)
92          color = color_dict[emotion_number]
93
94          #Cálculo de resultados
95          if(ACC_ACTIVATED):
96              acc = "  "+str(int(np.max(result)*100))+"%"
97          if(MULTI):
98              cv2.putText(frame, emotion_string+acc , (x+30,y-30), cv2.
    FONT_HERSHEY_SIMPLEX,  1, color, 2)
99              cv2.putText(frame, "FPS: "+str(fps) , (230,450), cv2.
    FONT_HERSHEY_SIMPLEX,  1, (0,0,0), 2)
100         else:
```

```
            cv2.rectangle(frame,(225,460),(590,420), (0,0,0),-1)
            cv2.putText(frame, emotion_string+acc+"FPS: "+str(fps) ,
    (230,450), cv2.FONT_HERSHEY_SIMPLEX,  1, color, 2)

        # Calculate most viewed emotion over 1 second and send results
        emotion_list.append(emotion_number)
        if (round(time.time()) != clock_seconds):
            clock_seconds = round(time.time())
            max_emotion = max(set(emotion_list), key = emotion_list.count)
            emotion_list.clear()
            client.publish("sensors/cam/emotion", emotion_dict[max_emotion],
     qos=0, retain=False)



    # Commented to operate over command line
    #cv2.imshow('Video', cv2.resize(frame,(1080,720)))
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
print("Desconectando... hasta la proxima :)")
```

Finally, the SMIOT developed code used for the cardiac band appears.

```
import time
import pexpect
import subprocess
import paho.mqtt.client as mqtt

def hexStrToInt(hexstr):
  val = int(hexstr[0:2], 16) + (int(hexstr[3:5], 16) << 8)
  if ((val&0x8000) == 0x8000): # treat signed 16bits
    val = -((val^0xffff) + 1)
  return val
```

```
11
12 def main():
13   try:
14     command = "sudo hciconfig hci0 reset"
15     strBytes = pexpect.run(command)
16     # escaneo los dispositivos Bluetooth LE con hcitool
17     command = "sudo stdbuf -oL timeout -s INT 5s hcitool lescan"
18     strBytes = pexpect.run(command)
19     result = strBytes.decode("utf-8")
20     pos = result.find("Polar") # busca la Polar
21
22     if(pos == -1): # si no encuentra la Polar
23       print("Fail connecting Bluetooth to polar")
24
25     while(pos == -1): # mientras no la encuentre
26       time.sleep(10)
27       # se reintenta cada 10 segundos
28       strBytes = pexpect.run(command)
29       result = strBytes.decode("utf-8")
30       pos = result.find("Polar") # busca la Polar
31
32     mac = result[pos-18:pos-1] # extrae la MAC
33     print(mac)
34
35     client = mqtt.Client()
36     client.connect("127.0.0.1", 1883, 60)
37     print("Connected to MQTT broker")
38   except:
39     print("Fail in h10, couldn't connect to broker")
40
41   while(pos != -1):
42     try:
43       #comprobamos la bateria restante
44       command = "sudo gatttool -t random -b "+ mac +" --char-read -a 0x003d"
45       strBytes = pexpect.run(command)
46       result = strBytes.decode("utf-8")
47       battery = str(int(result[-5:-3],16)) # paso a decimal
```

```
48      client.publish('sensors/h10/battery', payload=battery, qos=0, retain=
    False)
49
50      command = "sudo gatttool -t random -b "+ mac +" --char-write-req --
    handle=0x0011 --value=0100 --listen"
51      while True:
52          child = pexpect.spawn(command)
53          child.expect("Notification handle = 0x0010 value: ",timeout=10)
54          child.expect("\r\n", timeout = 10)
55          pulse = int(child.before[2:5], 16)
56          client.publish('sensors/h10/pulse', payload=pulse, qos=0, retain=
    False)
57          if(len(child.before)>15):
58            rr = hexStrToInt(child.before[12:17])
59            client.publish('sensors/h10/rr', payload=rr, qos=0, retain=False
    )
60          time.sleep(2)
61
62   except:
63      time.sleep(0.3)
64      command = "sudo hciconfig hci0 reset"
65      try:
66        strBytes = pexpect.run(command)
67        # escaneo los dispositivos Bluetooth LE con hcitool
68        command = "sudo stdbuf -oL timeout -s INT 5s hcitool lescan"
69        strBytes = pexpect.run(command)
70        result = strBytes.decode("utf-8")
71        pos = result.find("Polar") # busca la Polar
72        child = pexpect.spawn(command)
73        if(pos == -1):
74          print("Fail connecting to polar, program")
75        while(pos == -1): # si no encuentra la Polar
76          time.sleep(10)
77          # se reintenta cada 10 segundos
78          strBytes = pexpect.run(command)
79          result = strBytes.decode("utf-8")
80          pos = result.find("Polar") # busca la Polar
```

```
81
82          mac = result[pos-18:pos-1] # extrae la MAC
83        except:
84          print("Fail in h10, program")
85 if __name__=="__main__":
86   main()
```