UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Back-end implementation for an automatized car parking

Autor:

D. Samuel Pilar Arnanz

Tutor:

Dr. D. Juan Carlos Aguado Manzano

D. Daniel Castaño Navarro

VALLADOLID, JULIO 2020

TRABAJO FIN DE GRADO

TÍTULO:	**Back-end implementation for an automatized car parking.**
AUTOR:	**D. Samuel Pilar Arnanz**
TUTOR:	**Dr. D. Juan Carlos Aguado Manzano,**
	D. Daniel Castaño Navarro
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Ignacio de Miguel Jiménez**
VOCAL:	**Dr. D. Ramón Durán Barroso**
SECRETARIO:	**Dr. D. Juan Carlos Aguado Manzano**
SUPLENTE:	**Dr. D. ª Noemí Merayo Álvarez**
SUPLENTE:	**Dr. D. Juan Blas Prieto**

FECHA:	13 de Julio de 2020
CALIFICACIÓN:	

Resumen del Trabajo Fin de Grado

El propósito de este Trabajo Fin de Grado es la implementación de un software back-end para el control de un sistema de carsharing. La característica principal de este sistema de carsharing es que consta de unos parkings donde el vehículo se aparca autónomamente. El software back-end, guardado en nuestro servidor, se encarga de gestionar el funcionamiento de los parkings, por ello, el back-end es capaz de comunicarse con toda la flota de coches. Además, también es capaz de comunicarse con la aplicación de carsharing para cuando un usuario solicita un coche. El back-end proporciona otra serie de servicios como controlar que dispositivos acceden al sistema, administración de la base de datos del sistema y gestión de fallos. Durante el desarrollo de este software, se probado su funcionamiento gracias a un simulador donde se comunicaba con otros elementos del sistema como son los coches o las aplicaciones móvil.

Abstract

The purpose of this Final Degree Project is the implementation of a back-end software for the control of a carsharing system. The main characteristic of this carsharing system is that it consists of parking areas where the vehicle is parked autonomously. The back-end software, stored on our server, is responsible for managing the operation of the parking areas, therefore, the back-end is able to communicate with the whole fleet of cars. In addition, it communicates with the carsharing application for when a user requests a car. The back-end provides another series of services such as controlling which devices access to the system, administration of the database and fault management. During the development of this software, its operation was tested thanks to a simulator where it communicates with other elements of the system such as cars or mobile applications.

Palabras clave

Back-end, Enter procedure, Exit procedure, FMEA, IoT, MariaDB, MQTT, Outside stage, parking, process architecture, relational database, RFID, Smart City, Standby stage, messaging, Twizy, TwizyLine.

Acknowledgements

First of all, I would like to thank Juan Carlos Manzano Aguado, tutor of this Final Project Degree, for the opportunity to participate in the TwizyContest, and also to thank all the time and help that he has dedicated to carry out the TwizyLine project.

I would like to thank my colleagues from TwizyLine, Ignacio Royuela González, Adrian Mazaira Hernández and Mario Martín Fernández, for all the endeavor made in the last 9 months. Also, I would like to thank Jairo Gurdiel González and Daniel Castaño Navarro, employees of Renault, for their involvement in the project and in the development of the Final Degree Project.

Finally, I would like to thank all my family for supporting and encouraging me in these last 4 years in the degree. Special dedication to my grandparents who unfortunately passed away during this last year.

INDEX

FIGURE INDEX

TABLE INDEX

1 Introduction

1.1 Contextualization and motivation of the Final Degree Project

Air pollution is one of the main hazards our planet faces. WHO (World Health Organization) has warned several times the irreparable damage that can cause a high level of pollution in the whole world. In 2015, WHO notified that air pollution kills around seven million people worldwide every year [1]. The most harmful pollutant particles floating in the air with the strongest evidence of health effects are particular matter, nitrogen dioxide (NO_2), and sulfur dioxide (SO_2) [2]. Once people are exposed to this kind of air, the long-term diseases are diverse, but mainly it can affect our brain, hearth, and lungs (Figure 1). In a more recent research article published in 2018, the authors estimated roughly 9 million people have premature deaths due to this problem [3].

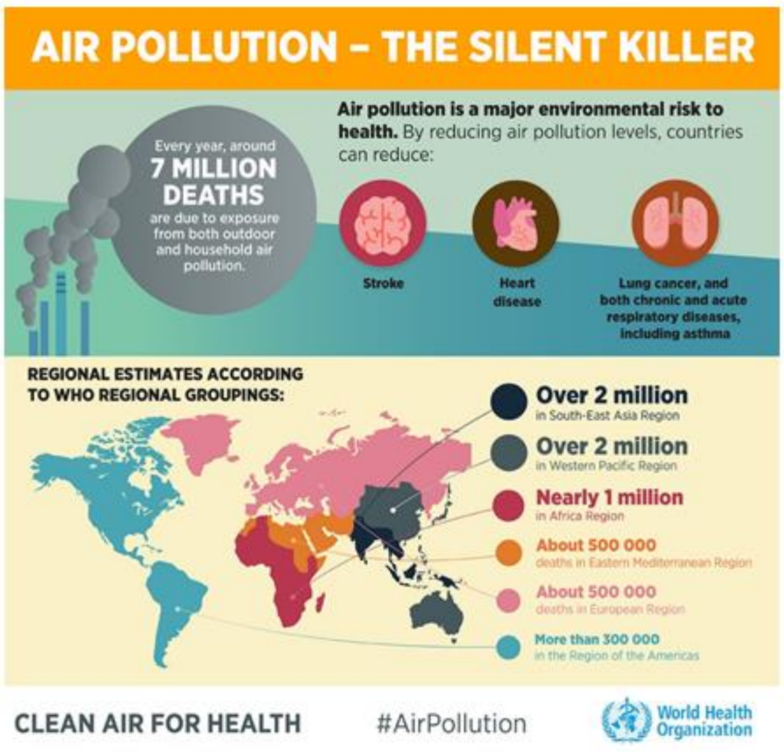

AIR POLLUTION – THE SILENT KILLER
Every year, around 7 MILLION DEATHS are due to exposure from both outdoor and household air pollution.
Air pollution is a major environmental risk to health. By reducing air pollution levels, countries can reduce:
Stroke
Heart disease
Lung cancer, and both chronic and acute respiratory diseases, including asthma
REGIONAL ESTIMATES ACCORDING TO WHO REGIONAL GROUPINGS:
Over 2 million in South-East Asia Region
Over 2 million in Western Pacific Region
Nearly 1 million in Africa Region
About 500 000 deaths in Eastern Mediterranean Region
About 500 000 deaths in European Region
More than 300 000 in the Region of the Americas
CLEAN AIR FOR HEALTH
#AirPollution
World Health Organization

Figure 1. Infographic of 2016 about how the air pollution hazards our health [4]

If we take a look at the map in Figure 1, it is plain clear that undeveloped countries from South Asia and Africa are the ones who suffer more this global crisis. However, the number of people affected in developed countries is also really high. Apart from the pollutants emissions with health consequences, the Greenhouse gasses emissions constitute also a huge challenge for the whole world. Since 1992 the United Nations organized a Climate Change Summit, it is yearly held to reach an agreement between different countries to apply policies to stop gas emissions. One of the main achievements of these gatherings is the Paris Agreement in 2015. It is a pact signed by almost 200 countries where each country assures that it will try to lower the emission of greenhouse-gasses until the previously agreed limit [5].

It is remarkable to notice that air pollution does not affect only humans. It contributes to climate change, which has multiple collateral consequences for flora and fauna, making more difficult for humans to live on Earth. One of its main effects can be seen in global warming (Figure 2) and the destruction of ecosystems.

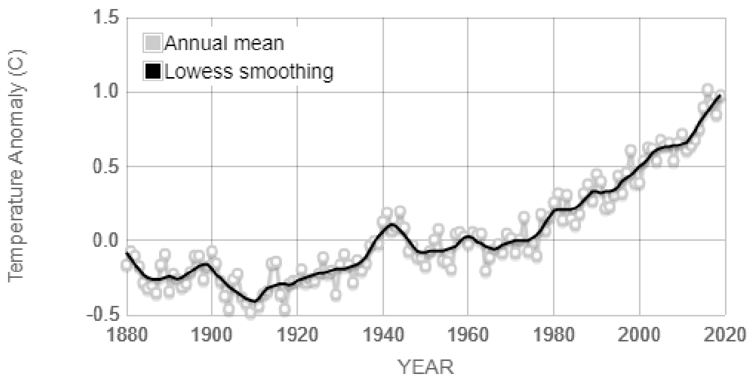


Figure 2. Global surface temperature relative to 1951-1980 average temperatures [6]

According to the WHO, most of the air pollution is man-made and is produced by the inefficient combustion of fossil fuels or biomass. These combustions are produced in the industry and vehicles. New ecological policies implanted are focused on these two big groups. For example, countries are supporting renewable energies, and gasoline and diesel cars are being forbidden to drive in big cities, where the concentration of a great amount of cars produces large quantities of particular matter and NO_2. So, as these pollutant vehicles could be no longer available to drive inside the cities in the future, one solution is the electric cars. The automobile sector has increased its investment in the research and development of hybrid and electric vehicles. However, their sales account for a significant percentage of the market only in some countries like Norway, Sweden or Finland [7]. Despite this, some research report states that its sales will rise in the next years, reaching 39% by 2030 (Figure 3) [8].

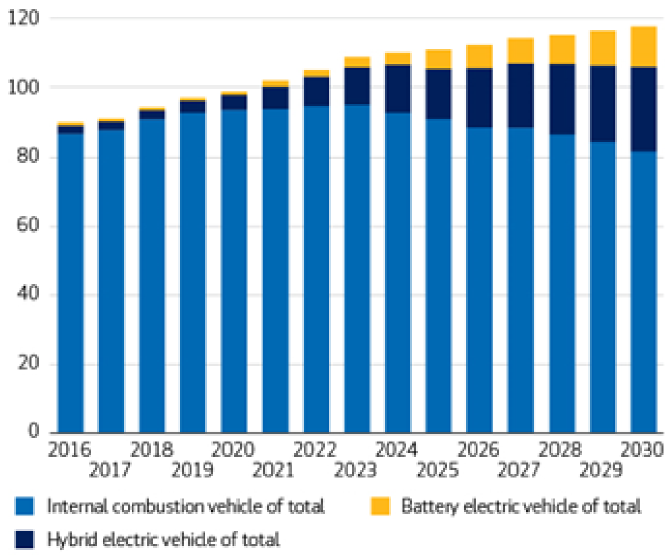


Figure 3. Hybrid and electric vehicle growth is expected to increase (millions of unit) [8].

Nowadays there are many government initiatives to fight air pollutant problem. So, many countries encourage their citizens to buy electric cars or to change their old pollutant car by a new electric one such as the plan MOVALT in Spain [9], and big cities like Madrid or Paris have launched their initiatives to regulate the traffic of polluting vehicles through the city center. For example, thanks to “Madrid central” the air pollution has been reduced by 32% [10].

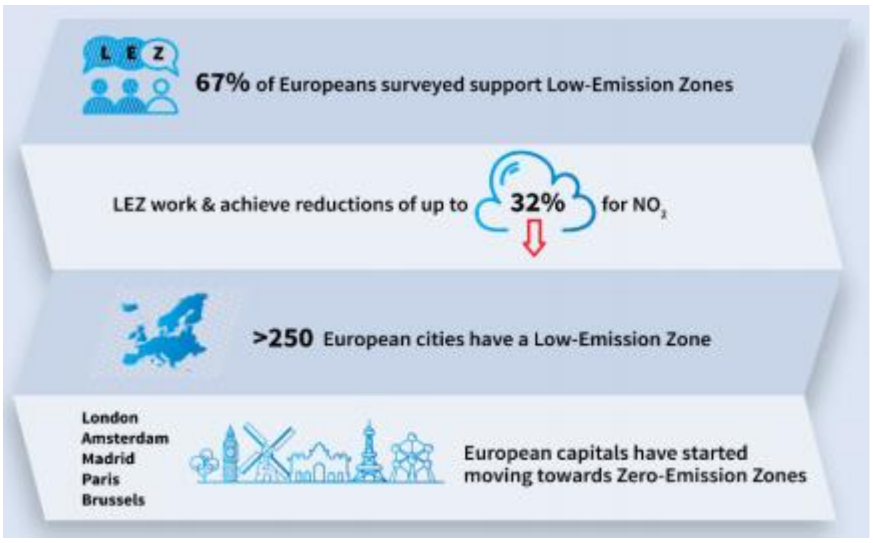

L E Z
67% of Europeans surveyed support Low-Emission Zones
LEZ work & achieve reductions of up to 32% for NO_2
>250 European cities have a Low-Emission Zone
London
Amsterdam
Madrid
Paris
Brussels
European capitals have started moving towards Zero-Emission Zones

Figure 4. Low-Emission Zones (LEZ) are becoming the aims of the main cities in Europe [10].

Cities are evolving to become eco-friendlier (Figure 4). They are trying to adapt to the new requirements of this ecological revolution. For example, one of the main drawbacks of electric cars is the few number of electric park stations where to charge them. So, cities are implementing steadily more of this class of service around the cities. This ecological revolution requires progress in technology.

In this context of global concern about the climate change plus the need of automobile companies to reinvent themselves to offer to the market new ecological cars is based this year the contest organized by Renault, called “TwizyContest”. “TwizyContest” is a competition between team groups from nine countries around the world. The challenge aims to suggest an idea to reduce the environmental footprint of the Olympic Games of Paris 2024. So the project must be innovative, ecological, frugal, and a durable solution. In addition, it was recommendable to use the model Renault Twizy, to provide it with new functions [11] [12].

The team made up of four students of the University of Valladolid will be the Spanish representatives in this event. The project that we have proposed is called “TwizyLine” (Figure 5). It is a carsharing system, with the advantage that the vehicles involved in this system have their own autonomous parking. Clients of our system do not need to worry about park place, which is huge convenient in an event such as the Olympic Games, and also for huge cities like Paris. Users just need to reach the parking and park it on the leaving zone. Everybody with a smartphone and downloading our application can take a Twizy from the pick-up zone of the parking [13].

Figure 5. Logotype and symbol of TwizyLine [13]

Then, the teamwork has marked four main technical goals to carry out this project:

- Mechanical changes in the Twizy: The mechanical structure of the Twizy must be modified to manage the longitudinal and transversal control. Its final aim is to achieve enough control over the car, to reach level 4 of autonomous driving.
- ECUs development: New ECUs for the Twizy, with new functionalities to process all the changes introduced to it.
- Back-end implementation of an autonomous parking: Enter and exit procedures must be fully controlled. Barrier and other elements of the parking are going also to be controlled. Furthermore, it must be able to manage incoming data from outside cars and smartphones.
- Front-end implementation in a smartphone application: Our system must have its own application. It must be available for most of the smartphones. It must be able to communicate with the server to ask for a car.

This Final Degree Project is in charge of explaining how the third main goal, the back-end implementation of the autonomous parking was developed. Thanks to this implementation the parking will be an infrastructure with its own brain to make decisions, and the most relevant of all, it is a connected infrastructure. Outside elements belonging to the same system as the parking are going to be able to interact with it. It is an approach to the intelligent buildings being part of a possible smart city of the future.

1.2 Objectives

The purpose of the document is to explain how the back-end implementation of the parking has been carried out. At the same time, it has also been developed by other members of the work team, the proper communication systems to connect the three main entities: car, parking, and smartphone application. On one hand, Ignacio Royuela González has developed a communication ECU, implemented in a humming board [14]. On the other hand, Adrián Mazaira Hernández has created a smartphone application, similar to the ones used for carsharing [15]. Furthermore, Mario Martín Fernández is in charge of the dissemination through social networks and the administrative management of the patent of the project [16].

The exact implementation of the software is highly dependent on the structure of the parking, this is, how the users and cars are going to interact with the infrastructure. Moreover, the back-end software must be able to control every place and event inside the parking. So, it is important to highlight that the first step was the description of the structure of the parking. Further on, it will be explained in detail.

The list of specific objectives included in this project is:

- Description of different zones and elements of the parking, such as the barrier and the turnstiles.
- Implementation of a server application to communicate with the parking devices, the whole fleet of cars and all smartphone application users.
- The software developed must be able to connect to a database, by using a standard database manager connection or implementing directly the database (being its own manager). Manager must be robust, to hold multiple connections at the same time.
- Design and creation of a database, where all the necessary information is stored.
- Design a complete control over the car, which is inside the parking. The parking system will decide which path has the car to follow to reach its park spot, or which car is the chosen to go out once a user asks for one through the application.
- Create a simple but effective back-up system if the software crashes.

1.3 Phases and methods

The realization of the project began with an in-depth analysis of the technologies required to make it done. The whole study is included in the document “TwizyLine-Technologies” [13], written by the work team. In that document, we include our proposal to implement “TwizyLine”, such as: how to manage the transversal and longitudinal control of the autonomous car, the infrastructure of the parking, the communications between the different elements involved in the project, security measures and the planning of the project (before the coronavirus outbreak) [13].

For the development of this Final Degree Project, it is particularly important the chapters about how the parking infrastructure and the communications system are. So the next phase was to know how the parking should work and which communication protocols the system does need. This second phase can be divided into the following steps:

- First, it was analyzed which kind of protocols are more suitable for our project.
- Once MQTT protocol was chosen, due to its properties, that makes it perfect for IoT projects, it was studied how to implement MQTT.
- Then, it was decided which programming language to use, which libraries were going to be necessary to work with python (the chosen language), to work with MQTT and parallel processing. To practice with these libraries, a mini-project was done, which consists in controlling a temperature and humidity sensor.
- The next step was to design flow diagrams to determine how the parking would act in each situation.
- Finally, all the software's logic was programmed.

The last phase was running the integration and validation tests. At this point, three members of the work team checked if the communications were well implemented. Therefore, this phase can also be divided into three parts:

- First: Check that the exchange of messages with the car's communications module was done correctly.
- Second: Check the correct functionality of the application thanks to the information provided by the server.
- Third: Full integration of the three parts, checking that the system works properly.

1.4 Means

This project has been carried out in two places, it was started in the laboratory 2L028 of the Higher Technical School of Telecommunications Engineers of the University of Valladolid. However, due to coronavirus spread across the whole country, the Spanish government decided to apply the alarm state status and the university closed. So the project had to be completed in the participants' houses, working in coordination and conducting integration tests remotely.

In order to be developed, it has been used 2 Raspberry's: a Raspberry Model B+; and another Raspberry Model B, which were placed in different locations. Furthermore, it was used a 8BitDo Zero controller.

2
State of the art

In the introductory chapter it has been seen that the objective of this Final Degree Project is to provide intelligence to an infrastructure, such as a parking, with the final goal of being able to communicate with other devices. This is one of the bases of what is called 'Smart city'. Millions of devices distributed throughout the city connected to the same network, to obtain and analyze this data in order to take action and make the city a more sustainable and ecological site.

This chapter is dedicated to study the state of the art of the current 'Smart cities'. Thus, it is required to research what technologies are behind this type of cities. Then, a selection will be made of the technologies that best fit our project to achieve its achievement.

2.1 Evolution of Smart cities and technologies related

“The 19th century was a century of empires, the 20th century was a century of nation-states, the 21st century will be a century of cities” Wellington E. Webb, former mayor of Denver (Colorado, Unites States of America). In July 2007, the urban population already surpassed the rural population in the world and the forecasts point out that in 2050 it will practically reach 70% [17]. Around 600 cities together gather the fifth part of the whole population, producing 60% of the GDP worldwide.

In this new situation, cities must deal with several problems. Forecasts indicate that urban spaces will become increasingly dense. They will have to face many difficulties related to the management of scarce resources, the provision of public services, the management of information, urban mobility and traffic, as well as to energy efficiency and in general to sustainability.

For a sustainable ecosystem, cities must evolve to manage better the resources to afford its citizens a high living standard. To this end, the concept of smart city has grabbed the attention of many governments of large cities around the world. The definition of smart city is a bit blurry, because of no city has already reached that state and nobody knows how far the smart cities can go. A smart city integrates the ICT (information and communications technology) with the physical infrastructures in a strategic purpose to provide efficient services and guarantee an optimum living standard for the citizenry. The idea behind this does not mean just to increase the social amenities, the services and the infrastructures, but it focuses on the efficient design of methods to optimize the available resources [18].

Certainly, Smart Cities are called to become one of the most powerful tools in public policy in the coming years. Integrating the use of information and communication technologies in the evolution of a city will not only lead to notable improvements in the provision of services, but it will itself constitute a sustainable path for economic and social development in the next decades of the economy of the cities and, therefore, of the economy of the countries.

Some of the most remarkable benefits that a smart city can produce are [19]:

- Reduce public expenditures: It decreases the capital used to provision and management of public services.
- Increase efficiency and quality of services: It is possible to manage resources more efficiently and improve the quality of the services provided.
- Provide support for decision making: Facilitates the identification of the requirements of the city and the approach of new services to offer support.
- Real-time information: It improves citizens' awareness of the environment in which they live by providing information in real-time and, at the same time, improves the transparency of the administration.

To sum up, a Smart City comes to support the development of cities, both in terms of improving their current problems and recognizing and controlling their future issues.

Some towns have already started their route to reach this hopeful city. Each year there are some ‘Smart City Expo’ around the world where the innovations are presented and analyzed. The last one should have been celebrated in Brazil during the 26th and 27th of March 2020, where the latest innovations would have been presented, but it was cancelled due to the coronavirus epidemic [20]. In the world, Barcelona was considered the most advanced city inside the smart city context, before New York City (USA) or London (Great Britain), according to the Juniper Research ranking. Another European city, which is likely to appear in the top positions, is Vienna (Austria) [21].

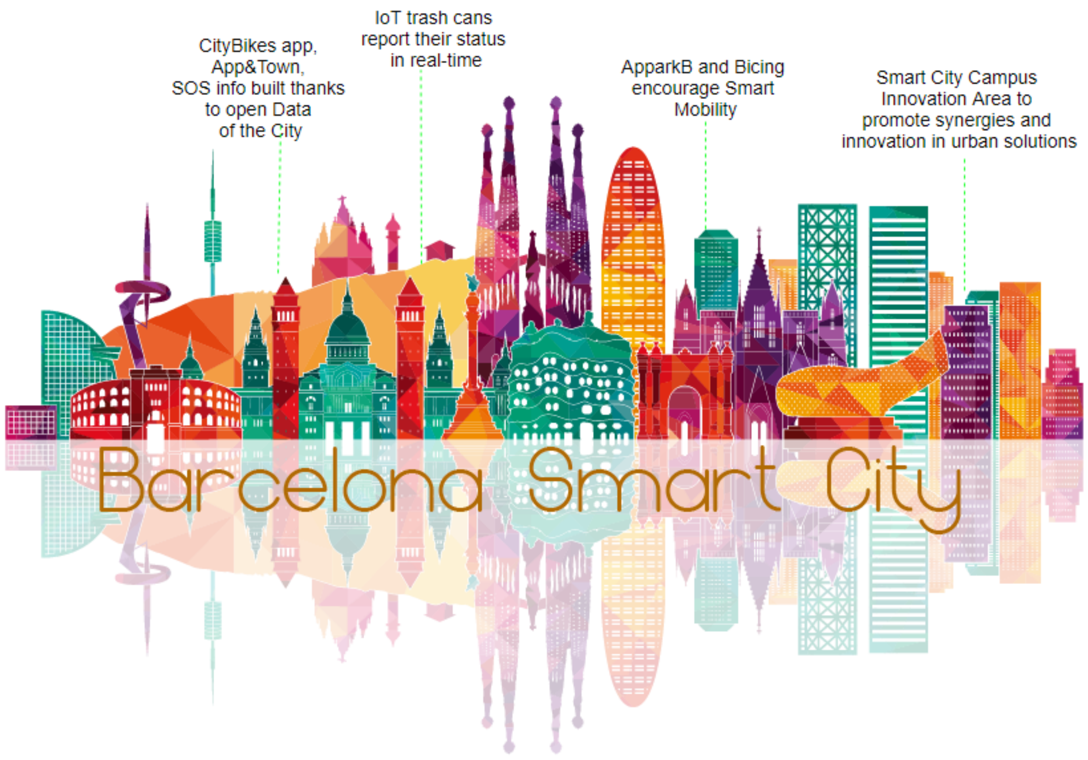

CityBikes app, App&Town, SOS info built thanks to open Data of the City
IoT trash cans report their status in real-time
ApparkB and Bicing encourage Smart Mobility
Smart City Campus Innovation Area to promote synergies and innovation in urban solutions
Barcelona Smart City

Figure 6. Barcelona approaches to reach the stage of Smart City [22]

According to Francesca Bria former CTO (Chief Technology Officer) for the city of Barcelona, the key of smart cities will be the digital democratization. So, the data coming from the devices produced by the people must be available for the people themselves and, moreover, that data should not be employed for commercial goals [23]. By raising the techno-literacy of the citizens, they will be able to understand how and where technology can best serve their interests. Accordingly, Barcelona has already implemented many people-oriented initiatives like those shown in Figure 6. Another relevant idea is the creation of the "Barcelona Digital City" website where the whole project and its technological achievements are collected [24[24] P]. That website is divided into 3 parts:

- Digital transformation: The technology and the data obtained from IoT sensors must be employed to provide better and more affordable services to citizens. Besides, another aim is to make government more transparent, participative and effective.
- Digital innovation: Support the use of the digital applications to battle social challenges and to promote the digital economy. FabLabs teach the necessary skills for innovation using new technologies. Projects like 'i.lab' develop an area for co-creation with the general public and the confluence of new technologies, such as open data and big data, robotics, artificial intelligence and the internet of things, to ensure sustainability and to analyze the possible environmental and social impact.
- Digital empowerment: The technological evolution must not interfere in the digital rights, gender equality and social inclusion. There are two remarkable projects following this philosophy, 'Vincles BCN', which aims to strengthen the social ties of elderly people by teaching them how to use new technologies; and 'Decidim Barcelona' is a social platform which tries to be the voice of the people living in Barcelona to take part or to suggest ideas related to digital innovations to be implemented in the city.

If we take a look at our region, there is a consulting firm located in Valladolid, called GEOCyL, which contributes to implanting territorial and environmental sustainability values, supported by the use of new geographic information technologies [25]. In 2017, they made a project called 'Mi ciudad inteligente' ('My Smart City'). This project was done by two members of GEOCyL, who travel across 30 different cities of Spain with Renault Zoe. This project had two objectives [26]:

- Analyze smart cities from a geographical perspective, observing how new smart policies and projects transform the habitability and morphology of our cities.
- Transmitting to the citizen the importance of new technological projects and social digitization, making use of the repercussion in the media, social networks.

Another interesting description of a smart city is related to technology. It is considered an urban space with smart infrastructures and networks with millions of sensors and actuators (Figure 7), which must also include people and their smartphones. A city able to listen and understand what is happening in it thanks to the data collected from all the elements that compound the whole town.

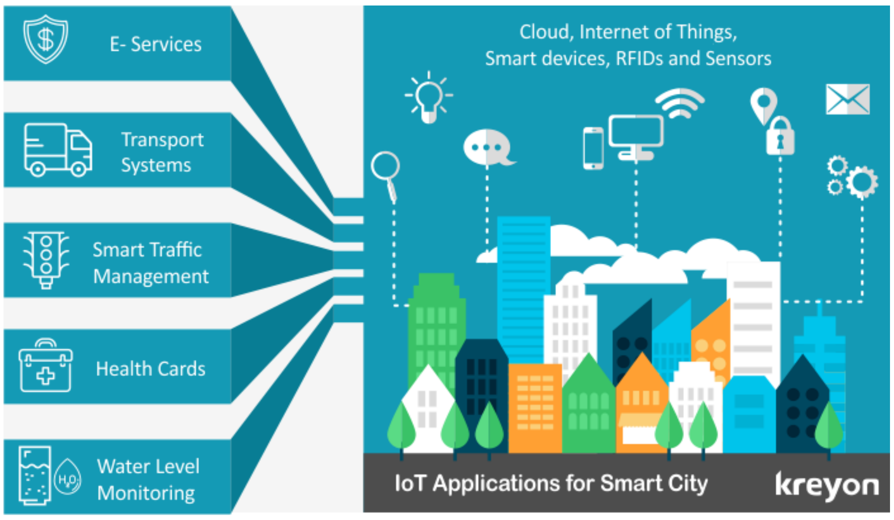


Figure 7. IoT applications for smart cities [25]

In a technological context, the smart city concept and the Internet of Things are two closely related terms. Both ideas have their basis in M2M (machine-to-machine) communications. Thanks to the applications and uses of that kind of M2M communications, IoT approaches what it is called to be the Internet of the future. Precisely, the Internet of the future will not only consist in the connection of more and more people, but ideally, everything can be connected, from devices to common objects that usually did not have this connectivity. This is the case of buildings, cars, household appliances and in general everything that could be managed or controlled. Without the slightest doubt, this new "digital reality" is going to require a new way of managing a house, a community, a city or even the economy of a country [19].

There are a wide variety of sensors to control whatever physical magnitude. Most of them already existed, but now thanks to the technological evolution, they are digitalized and connected to the Internet. So, it is possible to process all the data obtained in real-time and thus propose new services within the framework of the smart city. It is also relevant that these sensors can be reprogrammed remotely without the need for an operator to move to the place where it is installed to modified it. In this sense, for maintenance, the over the air programming (OTA) methodology is usually employed.

As an example, in recent years in Spain, all traditional electric energy meters have been changed by an electronic model that allows time discrimination and that has the capacity for remote management [28]. That change to smart meters is part of the smart grid. The smart grid is an electrical grid characterized by a variety of operation and energy measurement. It includes smart meters, smart appliances and renewable energy resources (Figure 8). The name of smart grid comes from 2007, when the USA government approved the "Energy Independence and Security Act of 2007 (EISA-2007)" [29]. The act aims at renewing the electrical infrastructure of the country and trying to employ renewable resources as a powerful mean to produce electricity. As a result, this concept of smart grid is related to the smart cities; it is even a part of the smart city.

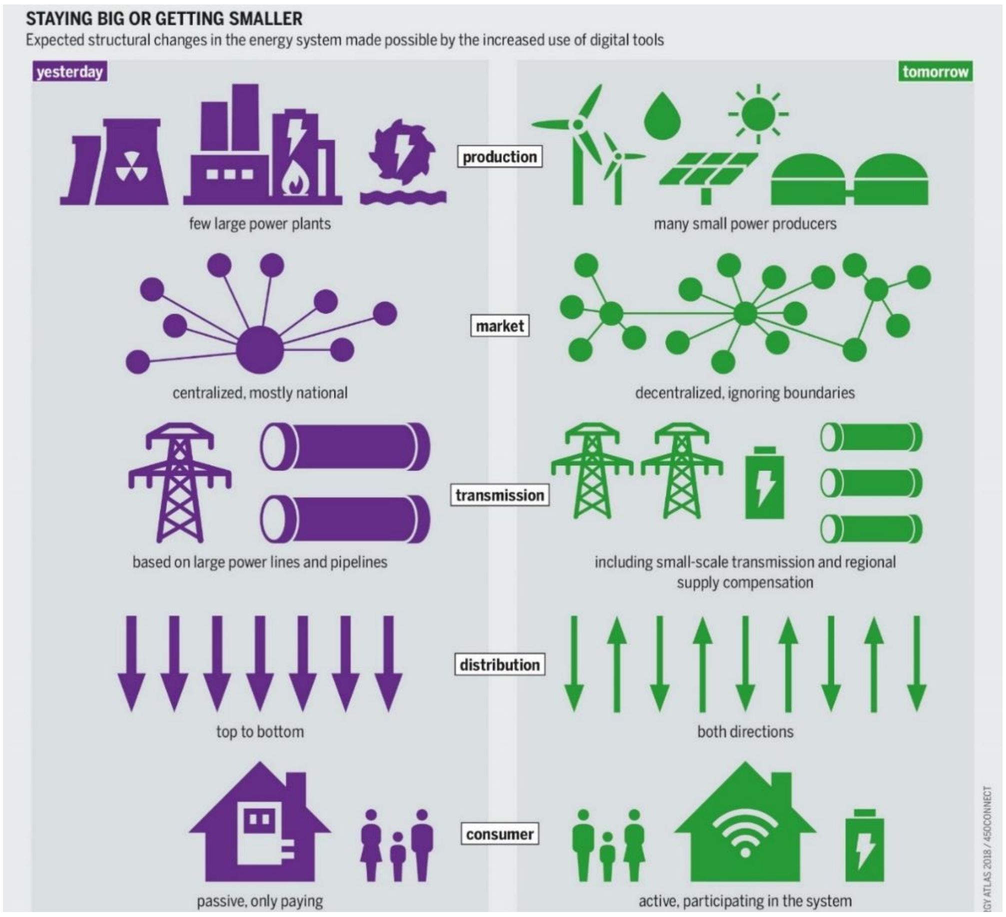

STAYING BIG OR GETTING SMALLER
Expected structural changes in the energy system made possible by the increased use of digital tools
yesterday
tomorrow
production
few large power plants
many small power producers
market
centralized, mostly national
decentralized, ignoring boundaries
transmission
based on large power lines and pipelines
including small-scale transmission and regional supply compensation
distribution
top to bottom
both directions
consumer
passive, only paying
active, participating in the system
RGY ATLAS 2018 / 450CONNECT

Figure 8. Traditional grid vs Smart Grid [30]

Another technology the use of which has been common lately is the RFID technology. RFID tags contain antennas to allow us to receive and respond to radio frequency requests from an RFID transmitter-receiver. This technology is very practical in inventory management, in the secure identification of assets (documentation, equipment ...), etc. The daily use of mobile devices, such as smartphones, PDAs, tablets, etc., has aided the use of different alternatives within the mobile marketing such as NFC technology. Concerning proximity sensors, the QR codes have been familiar in the last years, to obtain information about the object where they refer to.

Nowadays there is an increase in investment in research into this topic to help the society to overcome the coronavirus crisis. A clear example of an IoT which can be used soon is the product developed by Spanish company Libelium [31]. This company provides a device called the Meshlium Scanner. This device allows to detect smartphones, and in general, any gadget which works with Wi-Fi or Bluetooth interfaces (Figure 9). Thanks to this system, it can measure the number of people who are present in a defined area at a specific time, allowing the analysis of the evolution of the crowd and avoiding large gatherings. This system works also with cars. It can monitor the traffic, so it is possible to control the flow and congestion of vehicular traffic, leading to more efficient road systems in cities [32].

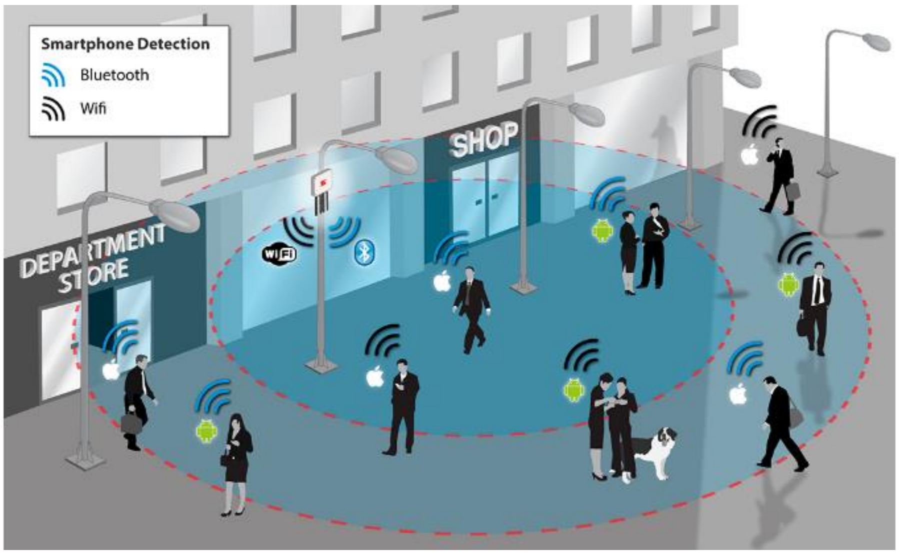


Figure 9. Meshlium Scanner to measure the number of people gather in a defined area [32]

2.2 Selection of technologies

As it was explained in the previous section, smart cities rely heavily on a type of technology called IoT. This technology is closely related to this Final Degree Project. Therefore, this section analyzes the characteristics of IoT projects and the technologies necessary for their development.

The IoT technological architecture is usually divided into layers, including the perception layer, the network layer and the application layer (Figure 10). The perception layer is made up of a network of sensors spread across a defined area, for example, distributed in a city. The network layer connects the perception layer to the application layer through a diverse range of communication technologies, such as Wi-Fi, satellite networks or Ethernet. The application layer provides a set of services for citizens or third party consumers [33].

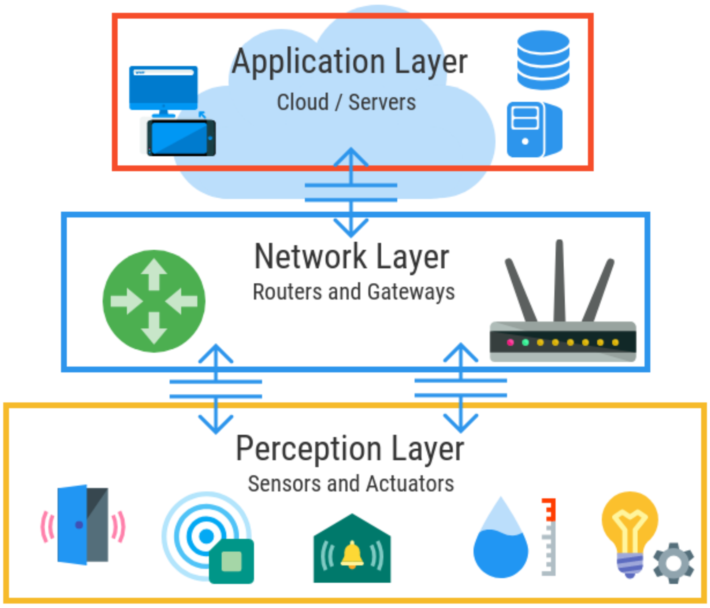


Figure 10. IoT technological architecture [34].

This Final Degree Project is focused on the application layer, especially on the back-end development. This back-end software is stored in the server. It is important to distinguish between back-end software and server, the server is the device where all the information about the system is saved and it provides services to its clients, and the back-end software is the algorithm used to manage all the operation done inside the server [35].

Having a look at the IoT projects architecture implemented in Barcelona, such as the IoT trash cans [36] or the smart Barcelona street lamps [37], the back-end software they use is called Sentilo (which means "sensor" in Esperanto). Sentilo is the part of the Barcelona Smart City architecture that is in charge of managing the information generated by the sensors placed across the city, providing accessibility and interoperability. It is estimated that Sentilo collects data from around 1800 sensors distributed in Barcelona city, registering more than 1.300.000 daily records in its databases [33].

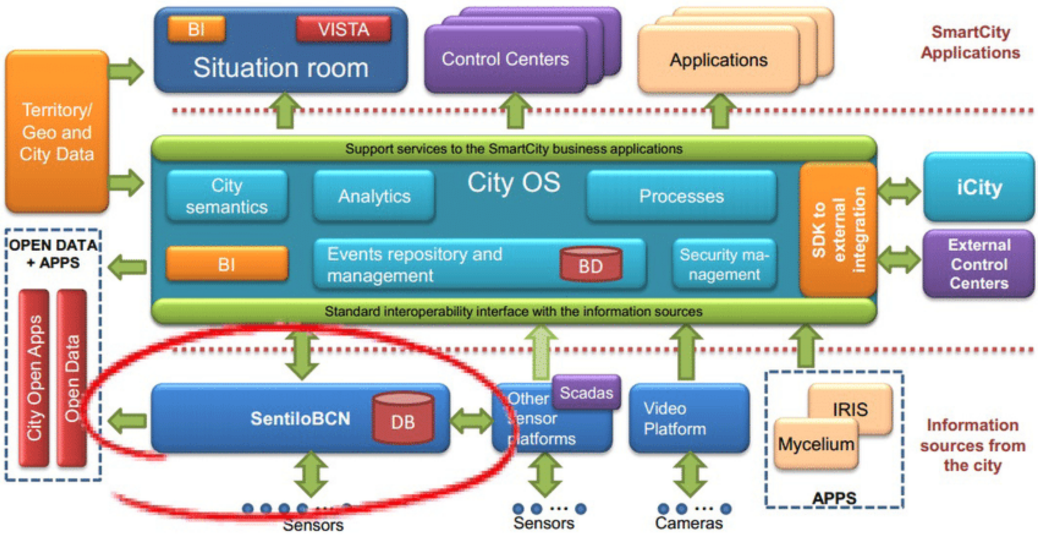


Figure 11. Barcelona Smart City Architecture [33].

Looking at the Information Sources (IS) layer from Figure 11 (botton part), it can be possible to see that Sentilo has its own database to store the information it is receiving. Furthermore, this whole layer works with the Middleware layer. The Middleware layer is a level between the application layer and the network layer, which main objective is to process, filter and make analytics procedures to get out the meaningful information as feeds for many applications [33].

TwizyLine needs its own back-end software, to make the functions similar to Sentilo, its database and the Middleware layer. After a search, the most relevant technological tools that can implement that kind of software and be used in IoT project are: Docker, Puppet, Django, NGINX and Grafana [38]. These software are created with C, Python, Ruby and Go. So, as it was desired to create this back-end software since the beginning, it was decided to choose one of these programming languages to develop the code. Below in Table 1, there is a comparison between these languages.

C	Python	Go	Ruby
Procedural programming	Multiple programming paradigms	Multiple programming paradigms	Object-oriented programming
Influenced in B and Fortran	Influenced in Perl and Java	Influenced in Python and C++	Influenced in Python and Perl
Supports paho MQTT	Supports paho MQTT	Supports paho MQTT	Do not support paho MQTT
Multiprocessing libraries	Multiprocessing libraries	No multiprocessing library or similar available	Multiprocessing libraries
Medium knowledge	High knowledge	Low knowledge	No previous knowledge

Table 1. Comparison between different programming languages [39][40][41].

As it can be seen in Table 1, Python provides the libraries required to work with MQTT (IoT protocol, later on this section it will be explained) and manage processes at the same time thanks to the multiprocessing library, along with the high knowledge of it, making it the best candidate to employ.

Python is a high level interpreted language. Python supports many programming paradigms, such as structured, object-oriented or functional programming [42]. Nowadays, Python is considered an embedded language. However, some years ago coding for embedded applications tilted heavily toward compiled languages, to avoid overhead of evaluating code on the fly on machines with limited processing power and memory. Many modern microcontrollers now have more than enough power to host a Python interpreter [43]. According to the ranking of the top programming languages 2019 made by the technological magazine "IEE Spectrum" edited by the IEE (Institute of Electrical and Electronics Engineers), Python was ahead (Figure 12) [44], repeating the same position as the previous year [43].

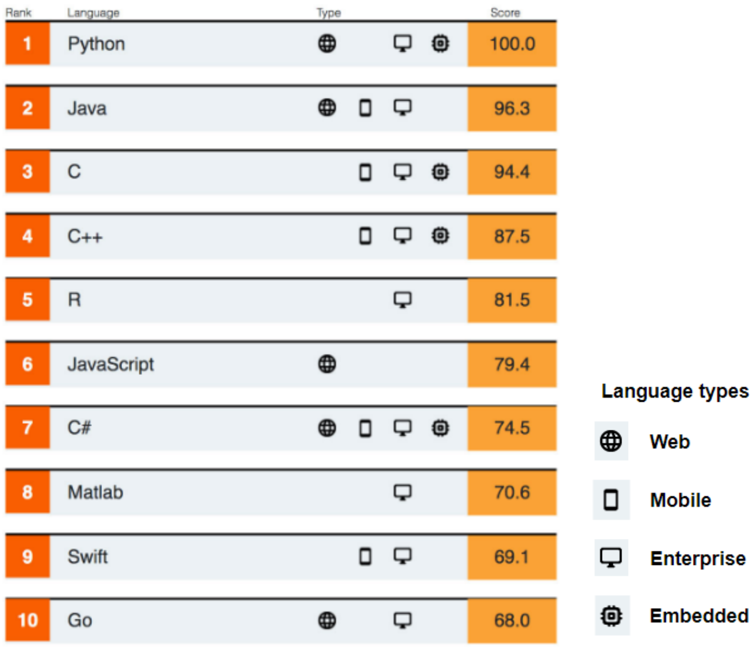


Figure 12. Ranking of most used programming languages worldwide in 2019 [44].

Finally, the programming language chosen to develop the back-end software was Python, but that was not the unique decision about the programming language. Python has two different versions:

Python 2 and Python 3. However, as Python 2 will be not being maintained past 2020, Python 3 was selected. In concrete, it will be used the Python 3.7 which was the latest version available.

Now that the programming language is defined, the next step is to choose which IoT protocol it is going to be implemented. One of the strongest points of this project is the communication of the parking with different devices. If we take a look at the protocol stack (Figure 13) of the parking back-end, we can see that three protocols are already chosen due to the characteristic of the software implemented. UDP (User Datagram Protocol) was consider but TCP (Transport Control Protocol) is more suitable. In the transport protocol, we need a reliable protocol that provides us with the option of introducing security protocols such as SSL/TLS (Secure Sockets Layer/Transport Layer Security). The physical layer depends on where the infrastructure of the parking is installed.

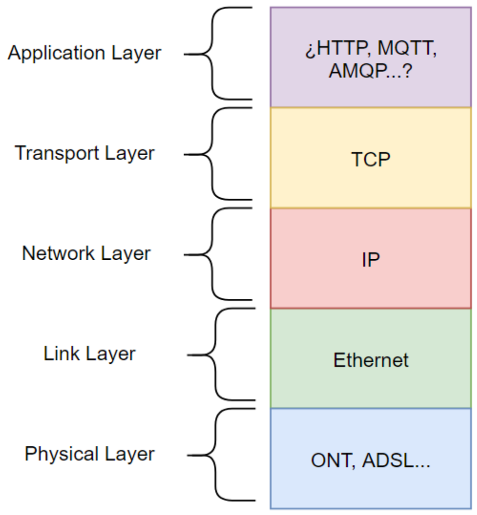


Figure 13. Protocol Stack, with the possible protocols to be implemented.

However, the variety of application protocols was larger. Finally, it was decided to focus on messaging protocols due to its common use in IoT projects. Other protocols, such as HTTP, were dismissed because of several reasons [45]:

- High network traffic: increased mainly because of large header size, not suitable for restricted networks.
- Synchronous protocol: the client waits for the server's answer. In the IoT world, synchronous communication has been a problem due to the large number of devices and the network is usually unreliable and with high latency.
- Half-duplex communication system: the client needs to initiate the connection. In IoT full-duplex connections are required, the devices and sensors are generally clients, which means that they cannot passively receive commands from the network.
- Point-to-point protocol: the client sends a request and the server answers. It is difficult and expensive to transmit broadcast messages to all networked devices, which is common in IoT applications.

The goal of the following section of this chapter is to compare the possible messaging protocols that can be used and analyze the best for our project. These protocols solve all the drawbacks presented by HTTP.

2.2.1 Messaging protocols

Messaging protocols are characterized by the transmission of telemetry between two or more devices. The most used messaging protocols are: MQTT (Message Queue Telemetry Transport), AMQP (Advanced Message Queue Protocol), CoAP (Constrained Application Protocol), XMPP (Extensible Messaging and Presence Protocol) and WrapUP [46]. So, the next step is to compare them, we will focus on the three first which are the most popular ones. Its main characteristics are shown in Table 2.

MQTT	AMQP	CoAP
TCP in the transport layer	TCP in the transport layer	UDP in the transport layer
Header size: from 2B	Header size: 8B	Header size: 4B
Simply	Complex	Complex
Multiple connections allowed	Multiple connections allowed	Multiple connections allowed
QoS implemented (3 levels)	QoS implemented (3 levels)	Not QoS, but it is implemented some kind of ACKs
Not compatible with HTTP	Compatible with HTTP	Compatible with HTTP
Based on publishing and subscribe messaging	Based on client/broker or client/server architecture	Based on client/server architecture
Design for IoT embedded systems	Design for a wealthier range of messaging circumstances	Design for IoT embedded systems
Ideal for minimum bandwidth networks	Low bandwidth networks	Low bandwidth networks

Table 2. Comparison of three different messaging protocols [47] [48].

According to the properties shown in Table 2, the best protocol for our project is MQTT. It is particularly designed for embedded systems, like the Raspberry; it uses TCP; header size is almost negligible; it can be used even if the network has limitations in bandwidth and the most important, its publish/subscribe architecture. Further on, this kind of architecture is explained more in detail, but basically it will help us to send messages just to single elements or groups of elements.

The other two protocols were dismissed due to the following reasons. AMQP provides us more functionalities that our project does not require, the header size is bigger than the MQTT header, so once we manage thousands of messages at the same time, the total sum of all the header can be significant, its use is more focus on websites. The main drawback of CoAP is that it is not supported by TCP, and the benefits that a publish/subscribe architecture provides us were better than the ones of a client/server architecture. For example, in MQTT a device can be at the same time publisher and subscriber whenever it needs. However, in the other system, the interchange of roles is not viable, and the server sends data only once the client has beforehand requested.

3
The autonomous TwizyLine parking

TwizyLine parking is thought to be built in cities. Its final aim is to provide a compact infrastructure for electric cars improving the mobility inside the city, especially for big events. It is specifically designed for Renault Twizy, however, the operating mode and the infrastructure to be used can be extrapolated for a wider range of electric cars.

Self-parking management can make TwizyLine parking one of the key components of the smart city of the future. But before explaining how this parking autonomously works, it is essential to analyze how its physical structure could be. Next sections focus on these topics. First, it is defined the architecture of the parking and then it is explained the control over everything concerning the parking.

3.1 Design of the circuit

The main feature of our parking is that vehicles are going to follow the line drawn on the floor. Therefore, the first dilemma to be solved is the circuit design. The proposed design must provide a series of characteristics such as: a parking zone and another to take out the vehicle; the largest possible capacity; implement a charging system in the parking lot; and ensure that the parking meets the ODD (Operating Design Domain) of the autonomous vehicle, which encompasses all the specific conditions under which a given drive automation system is designed to function, including driving modes [14].

Several ideas about the structure of the circuit were studied considering the previous requirements. One of the simplest designs is shown in Figure 14. This parking design was a circuit drawn in the floor by a magnetic tape that the Twizy has to follow reading that magnetic tape with a magnetic sensor. Thanks to the turns, it tries to solve the problem of the capacity of the parking. To solve the charging problem, there was an auxiliary path (draw in yellow in Figure 14), which provides access to charging stations (an employee should connect the Twizys manually) and repair zones.

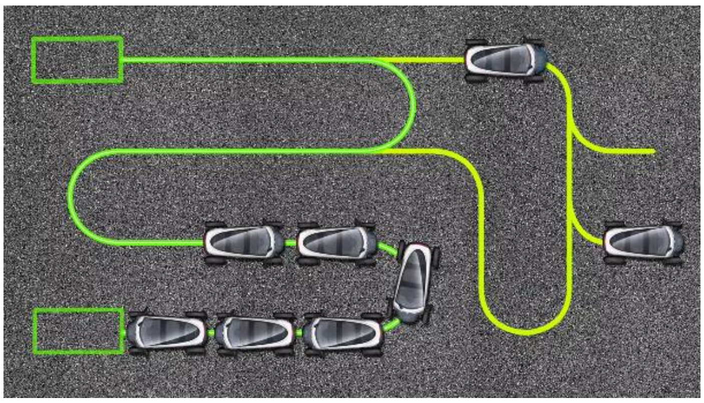


Figure 14. First idea for the parking structure [13].

However, this simple idea does not answer to several questions, which can be critical for the viability of the project. In a big city like Paris, the land is a scarce resource, so the infrastructure should optimize the space required to build it. Another dilemma is the charging system, if an employee is hired for each parking to plug the Twizy to the energy supply, the number of employees increased significantly taking into account that: there will be several parking areas; the parking areas are open most of the day and meanwhile it is open some employee must be there, thus more than one employee is required per day; in large parking with a high number of movements more than one employee is required at the same time. So, the project costs will increase due to the payment of their wages.

A better proposal that solves some of the problems of the first one is shown in Figure 15. First of all, there is an optimization of the space. In the previous proposal, the straight lines were separated exactly by the double of the turning radio of the vehicle (3,4 m, that is, 6,85 m [49]). However, in this proposal the straight lines of the circuit are only separated by the wide of a Twizy car plus the security band, that in total makes 2,4 m. In this design, there is no auxiliary path to charge the battery, because it is used an inductive charge system, which is deployed in the same path that the magnetic line. In order to know if the space is better optimized than the first design a study should be taken. However, this circuit has a great drawback, if a car breaks, it will block the whole parking. So, due to this major drawback this circuit was refused.

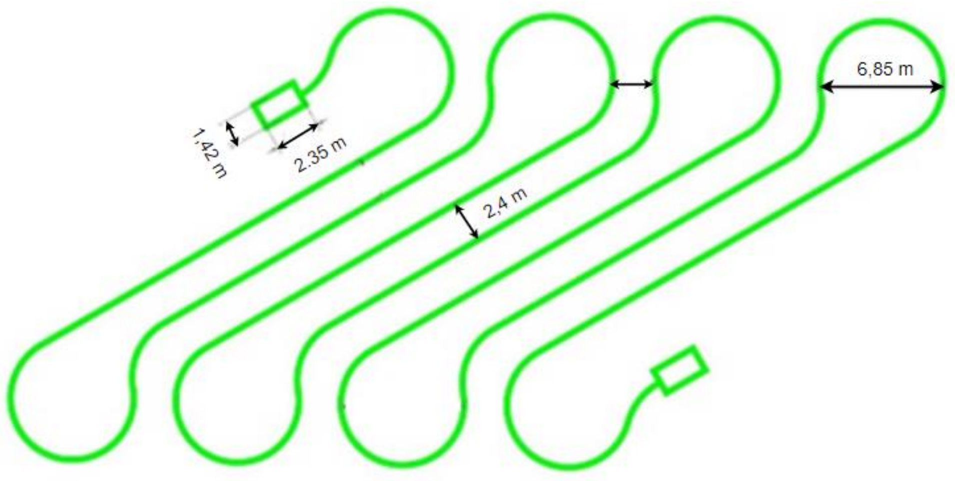


Figure 15. Variant of the TwizyLine parking circuit, with the appropriate measures of the Twizy [13].

Finally, the chosen design implementation is represented in Figure 16. It is made up of some rows where vehicles can park, and also there are two zones where to leave and take the Twizy. The parking of Figure 16 would be an example for small or medium parking. The number of columns or rows can be modified depending on the space available.

This new design allows to park or to take out any Twizy from different rows, so if one of them gets blocked, the parking can be still giving service to the clients. Furthermore, like the other two designs, it provides three different areas (leaving zone, inner circuit and pick-up zone) according to the requirements of the ODD [14], and it is implemented a charging system, which it is explained later in this section. In order to know if the capacity of this circuit is worthy we are going to make some calculations.

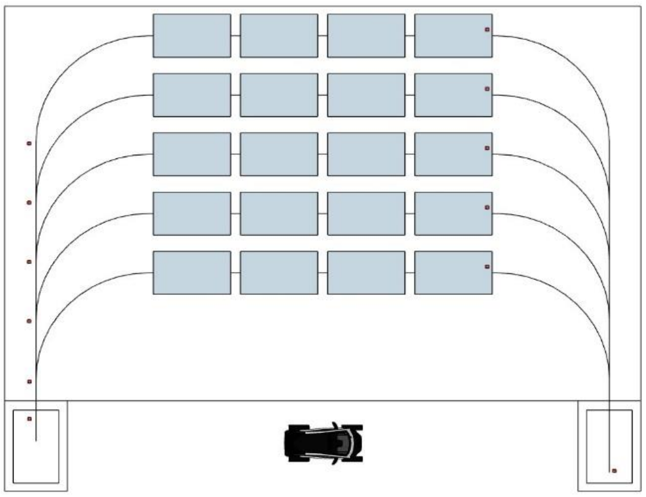


Figure 16. Final design for TwizyLine parking [13]

To get the optimum design, it has been made a research considering different parameters such as the length and width of the Twizy, safety distances between cars inside the parking or the turn radio of the Twizy. The objective of that research was to compare the density of vehicles per square meter of a TwizyLine parking from a conventional parking. The parking taken as reference is design following the urbanistic rules of Madrid (Figure 17) [50].

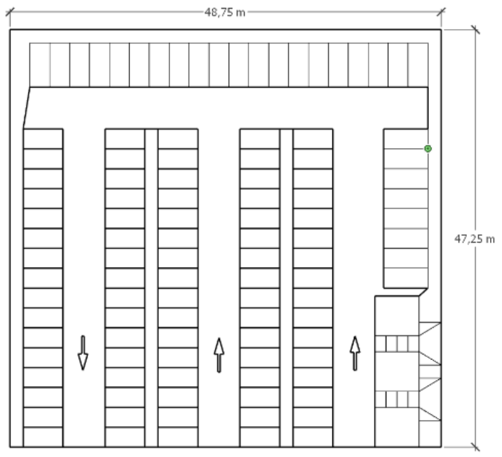


Figure 17. Conventional parking of Madrid [13]

The conventional parking shown in Figure 17 has a density of 0.0486 spot/m2. For a TwizyLine parking of 112 parking spots, the optimum distribution should be with 16 columns and 7 rows. As it can be seen in Figure 18, the density is better for that distribution, and it is achieved 0.1003 spot/m^2. The density has been duplicated in our parking.

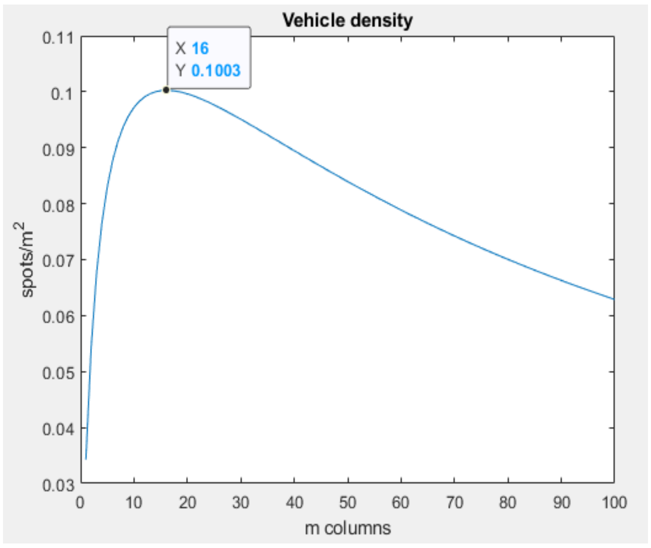


Figure 18. Optimization of the number of rows and columns for a TwizyLine parking[1].

To calculate that density, it has been supposed that the distance of security is 0.5 m and the distance of separation between cars of the same row is 0.3 m. These distances determine the exact density which our parking can have, however, these distances depend on how accurate our system is. The calculations made regarding the density of the parking varying these two parameters are shown in Figure 19. As it can be seen in the graph, if it is chosen 0.1 m for the distance of security and 0.2 m for the separation between cars of the same row, the density is 0.1456 spot/m^2, it has been almost triplicate the density of the conventional parking.

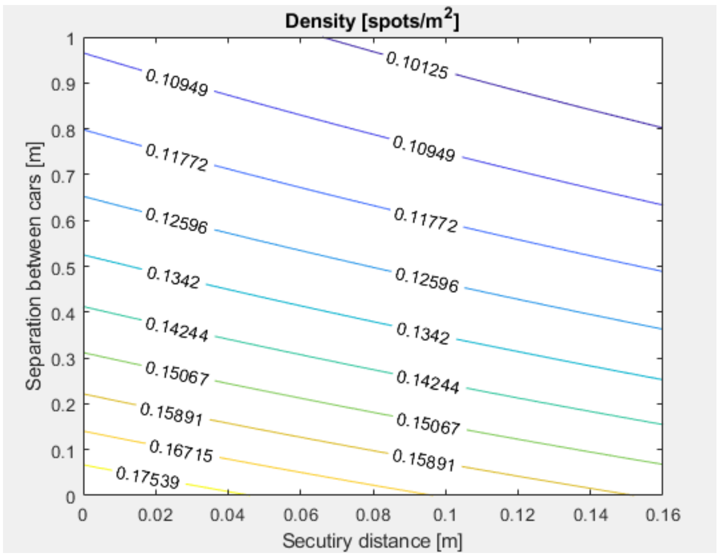


Figure 19. Densities of a TwizyLine parking depending on two parameters[2].

As this project aims at being implemented in the Olympic Games of Paris 2024, there is another design customized for big events. In this new design shown in Figure 20, there are more zones were to leave the parking and where to take the Twizy. One important advantage of this model is that if one car has any problem and stops moving, an employee can get out that Twizy from the line and once it

[1] For further information about how the graphic of Figure 18 have been calculated, see Annex I.
[2] For further information about how the graphic of Figure 19 have been calculated, see Annex I.

is repaired to introduce it on the row again without going out the parking. It can also be possible that a car can move, but it needs to be repaired some warnings, so the server can be able to send this car again to the beginning of the row, and to assign the next car to the client.

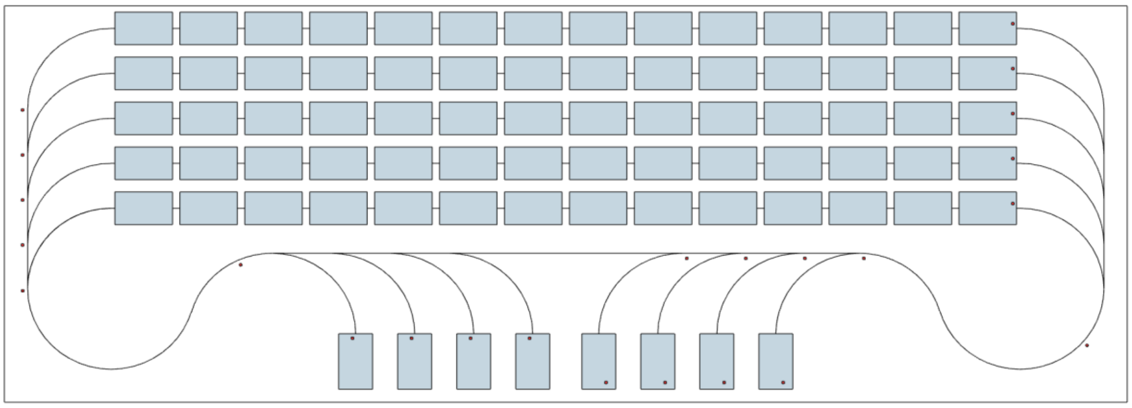
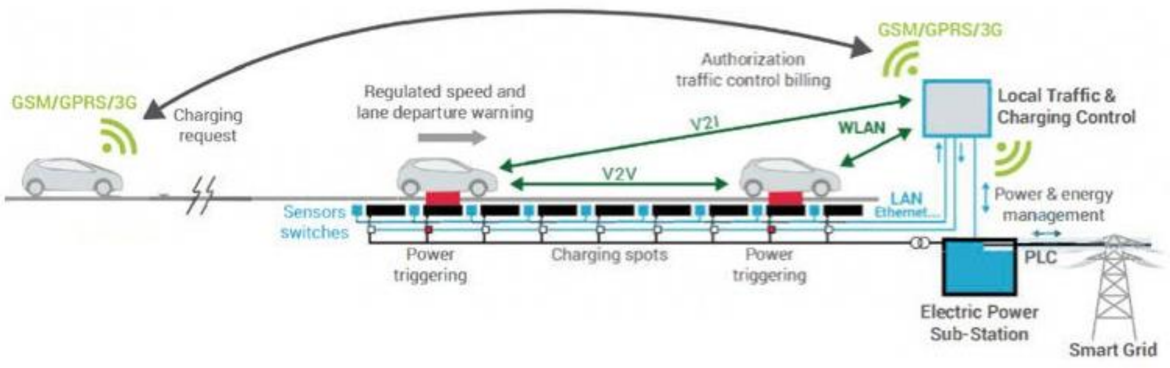


Figure 20. TwizyLine design for major events [13]

The chosen charging system is the same as in the previous one, it uses inductive charge. However, here each line can be designed with a different charging power. It does not matter the order, but in order to follow always the same rules, it has been defined the following assignation: the lines nearer to the entrance and exit zones should have less charge power, so, cars with high battery are going to be parked in those lines. In the other way around, those cars with lower batteries are going to be parked on the last lines, where the charge power is higher. The charge power of each line follows a staggered distribution. For example, for the design in Figure 16, as it has five rows and the maximum battery is 100%, each line is dedicated for a gap of length 20, the first line is dedicated to cars with battery between 0% and 19% (both inclusive), the next line for vehicles with 20% to 39% of battery and so on. This kind of distribution gives the parking high efficiency, to provide always the client an available Twizy with higher battery. In addition, it reduces the energy power supply it needs, if it is compared with a parking which has a high charge power in all of its rows. Renault has already carried out some research on wireless charging. Specifically, Renault, Qualcomm Technologies and Vedecom have joined to develop a charging system for electric vehicles meanwhile they are running over the circuit (Figure 21) [51]. This project is being developed in Satory (Versalles, Paris).


GSM/GPRS/3G
Authorization traffic control billing
GSM/GPRS/3G
Charging request
Regulated speed and lane departure warning
V2I
WLAN
Local Traffic & Charging Control
V2V
LAN Ethernet...
Sensors switches
Power & energy management
Power triggering
Charging spots
Power triggering
PLC
Electric Power Sub-Station
Smart Grid
Figure 21. Functioning of the project from Renault for wireless changing [52].

3.2 Elements and zones of the parking

In the final design of Figure 16, it can be possible to distinguish three zones in the parking which names are: the leaving zone, the inner circuit, and the pick-up zone. It is important to highlight that the whole parking must be delimited by a fence, except the entrance and the exit zones where there are barriers. This idea of delimiting the parking is not only due to safety for the cars, but also to have under control the area where the car is in autonomous mode to meet some legal and technical requirements (for more information check the Final Degree Project of Mario Martín Fernández [16] and the Final Degree Project of Ignacio Royuela González [14]).

In order to understand better the next sections, it is important to know that the Twizy can be in three different modes: autonomous, manual, and standby mode. Depending on the moment and in what place the Twizy is, it will change from one mode to another. For more details about the Twizy's configuration check Final Degree Project of Ignacio Royuela González [14].

The leaving zone is the place where clients have to park their Twizys. It is delimited by two barriers (barrier 1 and barrier 2). There is also a unidirectional turnstile (turnstile 1) to allow users who leave there their car to go outside. In the floor, there is an RFID tag which is used to notify the parking that there is a car in that position. In Figure 22, there is a representation of the leaving zone.

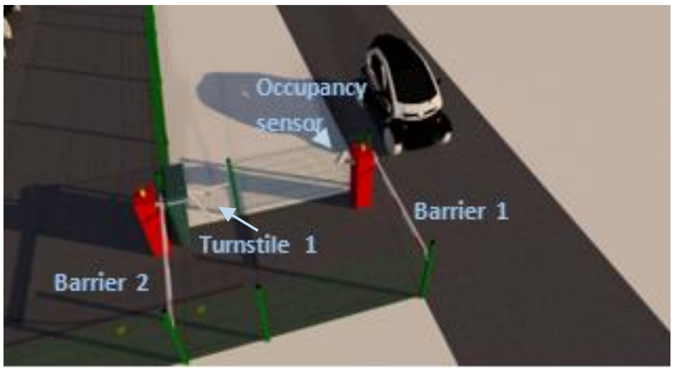


Figure 22. Leaving zone [13]

The inner circuit is made up of the magnetic tape which indicates the available paths for the Twizys. Furthermore, as it can be seen in Figure 22 drawn in yellow points, there are some RFID tags around the circuit. There are two types of RFID:

- Turn RFID tags: these tags indicate the beginning of each row. They are used to order the Twizy when to turn (Figure 23).
- Stop RFID tags: these tags are at the end of the row. They are used to notify the car that it has arrived at the end of the line and it needs to stop.

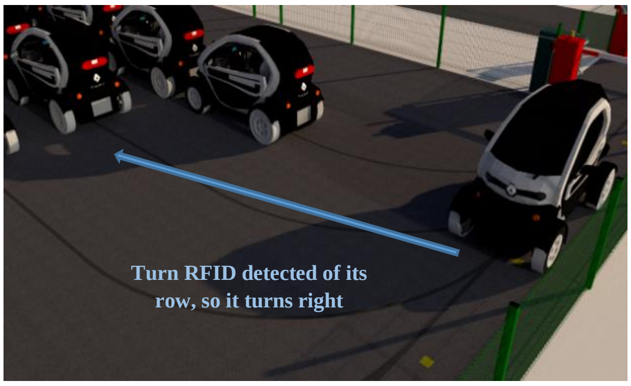


Figure 23. Car turns once it detects the turn RFID of its row where it is going to be parked [13].

The pick-up zone is the place where the clients are going to take its Twizy. Users are going to be able to enter this zone just once the car has already changed to manual mode. Like the leaving zone, the pick-up zone is delimited by two barriers (barrier 3 and 4), and there is another unidirectional turnstile (turnstile 2). Similarly to the previous turnstile, turnstile 2 is critical to meet legal requirements of this kind of projects. However, in this case, the device is blocked until three conditions are met: the car is stopped on the pick-up point, it is not in autonomous mode and the user who has asked for it has passed its smartphone near to an NFC tag or a QR code that should be next to the turnstile. There is also an RFID tag on the floor, to notify the car to stop and for the system to know that the car has reached this zone (Figure 24).

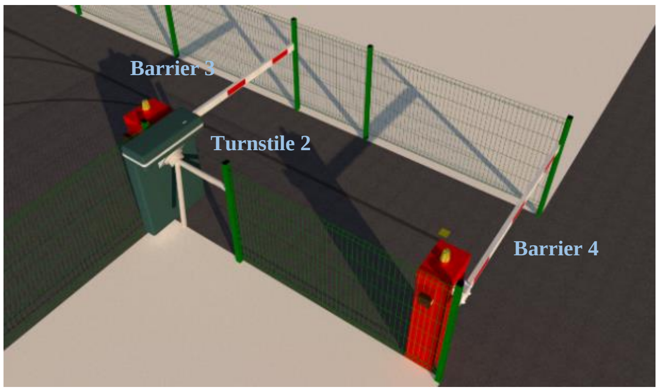


Figure 24. Pick-up zone [13].

3.3 Operation of the parking

This section describes how the parking must act in each situation. To help the explanation, the corresponding flow diagrams will be shown for each case. In the flow diagram appears the concrete name of the messages exchanged, which are explained in the section 4.4.

Specifically, a Twizy can be involved in 4 different states, regarding the parking: outside stage, enter procedure, standby stage and exit procedure. The relationship between them is represented in Figure 25.

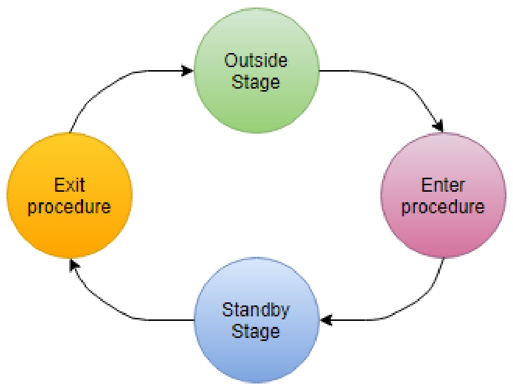


Figure 25. Diagram of the 4 possible states of a Twizy.

3.3.1 Outside stage flow diagram

First, it is going to be tackled the entrance procedure. To start during this state, it is relevant to know that the Twizy is sending all the time its GPS position and its battery level, so, the server knows if it is near some parking. In fact, this checking that a Twizy is near to a specific parking is only done thanks to compare coordinates of both, however, this system can be reinforced by adding other technologies to verify that the car is really near it, just in case the GPS position is not too accurate. If the vehicle is close to the leaving zone, the barrier 1 elevates. This stage is called ‘Outside Stage’ and its flow diagram is in Figure 26.

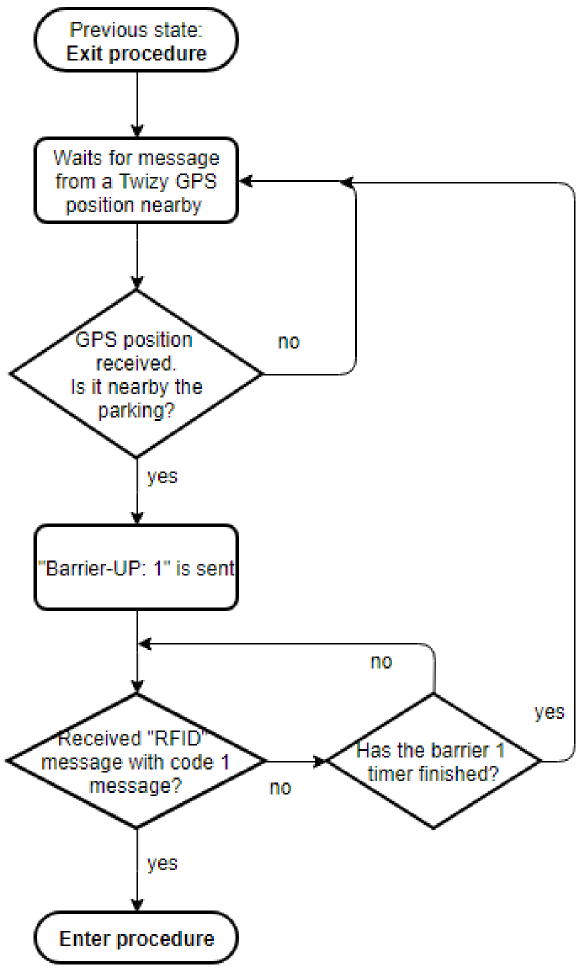

Previous state:
Exit procedure
Waits for message from a Twizy GPS position nearby
GPS position received. Is it nearby the parking?
no
yes
"Barrier-UP: 1" is sent
no
yes
Received "RFID" message with code 1 message?
no
Has the barrier 1 timer finished?
yes
Enter procedure

Figure 26. Outside stage flow diagram.

Once the car is over the leaving zone, the Twizy reads the RFID tag of that zone and sends a message to the server, saying that it has read that specific RFID. Now the back-end implantation must be able to process that data to know from which parking is that RFID. The barrier 1 of that parking must close, so no more Twizy enter to that zone, and thus, it is ensured that there is just one car in that place.

3.3.2 Enter procedure flow diagram

Now the system has to wait until the driver, and maybe another passenger, pass through the turnstile to verify that no humans are on the leaving zone. This system can be also reinforced with the use of occupancy sensors to detect that nobody has jumped over the turnstile or has illegally passed through the barrier 1.

When the car is alone in the leaving zone, barrier 2 opens, and the message to set the car in autonomous mode is sent. Once this message is acknowledged, another message with the path it needs to follow inside the parking is transmitted to the car to be parked autonomously. The battery level determines in which row the Twizy must be parked.

The barrier 2 closes just when the car reads the turn RFID to ensure that the barrier 2 will not collide with the vehicle. Since this point of the process, the leaving zone is enabled to service another Twizy. When the car reaches its position, the back-end sends him a message to pass to standby mode. This whole process is included in the 'Enter procedure' shown in the flow diagram of Figure 27.

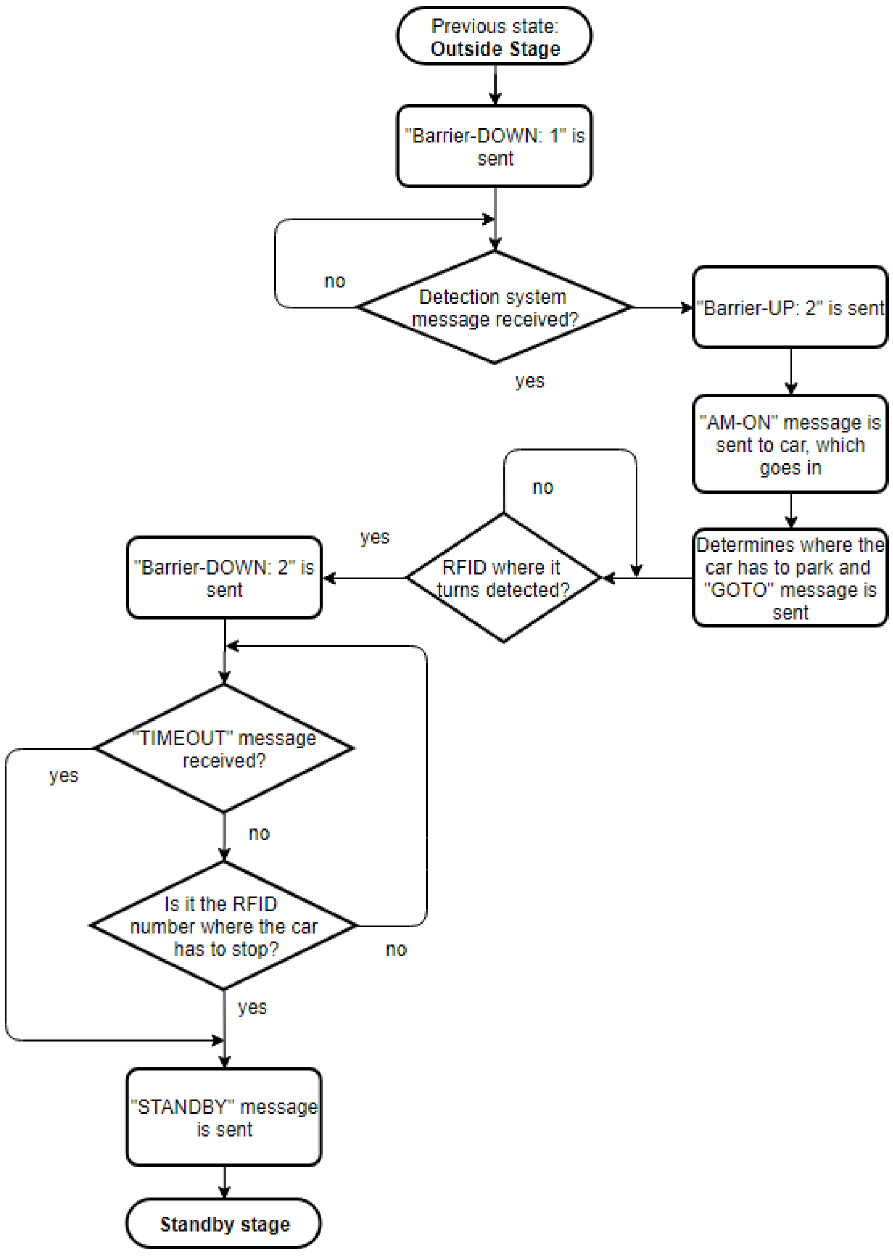

Previous state: Outside Stage
"Barrier-DOWN: 1" is sent
no
Detection system message received?
yes
"Barrier-UP: 2" is sent
"AM-ON" message is sent to car, which goes in
Determines where the car has to park and "GOTO" message is sent
no
RFID where it turns detected?
yes
"Barrier-DOWN: 2" is sent
"TIMEOUT" message received?
yes
no
Is it the RFID number where the car has to stop?
no
yes
"STANDBY" message is sent
Standby stage

Figure 27. Enter procedure flow diagram.

3.3.3 Standby stage flow diagram

Now it is going to be explained the state of 'Standby stage' (Figure 28). If someone wants to use our system must previously download the TwizyLine application to ask for a Twizy. Then, it is required to verify that the credentials of the user belong to the ones saves on our database. There will be a series of message exchange between the server and the application to know the client the Twizys available and to select from which parking does the client want the Twizy (for more details on this procedure, see the Final Degree Project of Adrian Mazaira [13]).

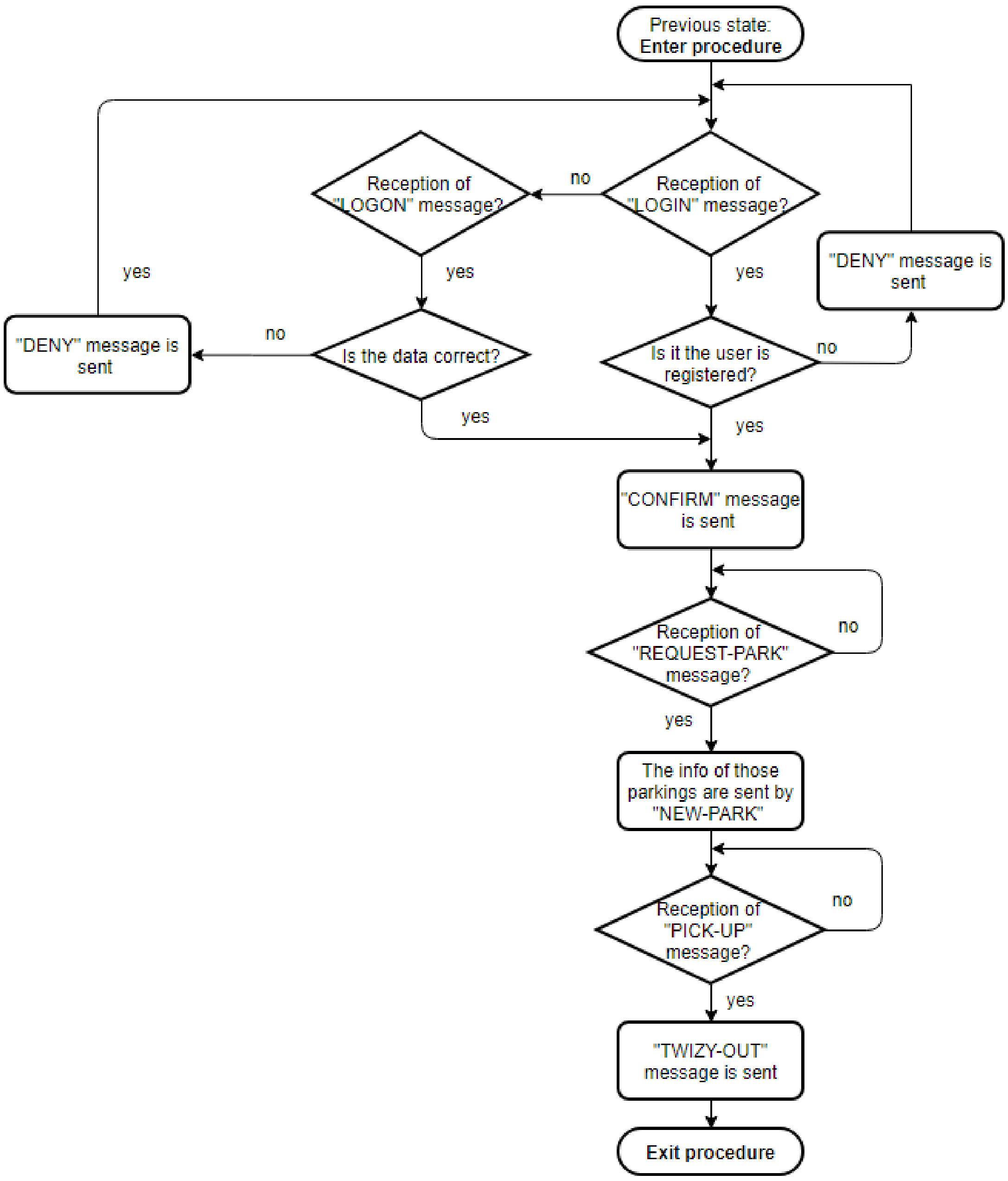

Previous state: Enter procedure
Reception of "LOGIN" message?
no
Reception of "LOGON" message?
yes
yes
"DENY" message is sent
Is it the user is registered?
no
Is the data correct?
no
"DENY" message is sent
yes
yes
yes
"CONFIRM" message is sent
Reception of "REQUEST-PARK" message?
no
yes
The info of those parkings are sent by "NEW-PARK"
Reception of "PICK-UP" message?
no
yes
"TWIZY-OUT" message is sent
Exit procedure

Figure 28. Standby stage flow diagram

3.3.4 Exit procedure flow diagram

Once a Twizy has been assigned to a client, the server starts running the 'Exit procedure' (Figure 29). This process has two main objectives: to move the selected Twizy to the pick-up zone in order to be taken by the client, and to move forward one spot the cars behind from the same row as the selected one to make more room for incoming vehicles. Clients can enter the pick-up zone through the turnstile to take their Twizy, only when the car is in manual mode as is described in section 3.2.

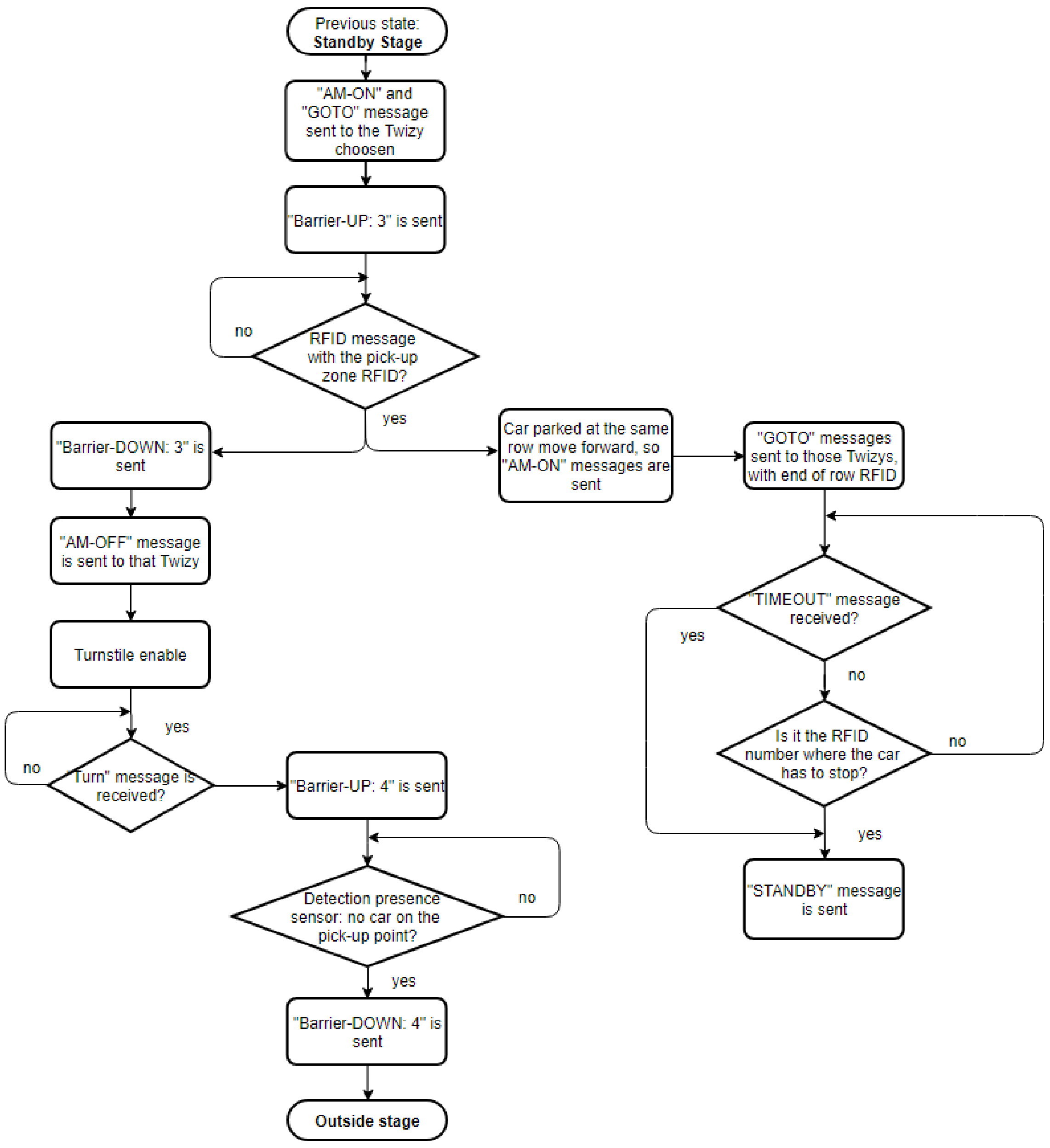

Previous state:
Standby Stage
"AM-ON" and "GOTO" message sent to the Twizy choosen
"Barrier-UP: 3" is sent
RFID message with the pick-up zone RFID?
no
yes
"Barrier-DOWN: 3" is sent
"AM-OFF" message is sent to that Twizy
Turnstile enable
"Turn" message is received?
yes
no
"Barrier-UP: 4" is sent
Detection presence sensor: no car on the pick-up point?
no
yes
"Barrier-DOWN: 4" is sent
Outside stage
Car parked at the same row move forward, so "AM-ON" messages are sent
"GOTO" messages sent to those Twizys, with end of row RFID
"TIMEOUT" message received?
yes
no
Is it the RFID number where the car has to stop?
no
yes
"STANDBY" message is sent

Figure 29. Exit procedure flow diagram.

Along this section, it can be seen in the flow diagrams that the messages exchanged between the back-end software and the rest of the elements are already predefined. Further on this document, in chapter 4.4 is explained more in detail all the messages.

4
Analysis of the back-end implementation

The back-end software will be in charge of managing all the operations carried out within the TwizyLine system. Some of the most notable tasks are: the simultaneous interconnections between different elements, especially vehicle-parking and application-parking communication; in addition to failure management, controlling which devices can connect to the back-end or administration of a database of system information.

The back-end is a critical point of the entire system since if it failed, the communication between the other elements among themselves would be impossible. Therefore, a back-up system must be provided. This chapter will be dedicated to explaining how this back-end has been implemented so that it fulfils its mission.

4.1 MQTT protocol

MQTT is a broker based publish and subscribe messaging protocol developed by IBM (International Business Machines Corporation). It stands out from other similar protocols because of being: open, lightweight, easy to implement and designed to be employed in constrained environments. For example, it is suitable to implement MQTT where the network is expensive, unreliable or has low bandwidth. Besides, it is ideal to run it on an embedded device with limited processor or memory resources.

Another main feature of MQTT is the publisher/subscriber architecture. In this protocol's architecture there are four key concepts (Figure 30):

- Broker: It is the service in charge of managing the network and sending the messages. All the clients connected to the broker send a periodic ping message to keep alive the connection.
- Publisher: Client who wants to send a message to a specific topic. It will be sent to the broker and then to the clients which are subscribed to the same topic.
- Subscriber: Client who received all the data sent to the topic subscribed.
- Topic: Depending on what kind of information the messages carry; they can be sent through one topic or another.

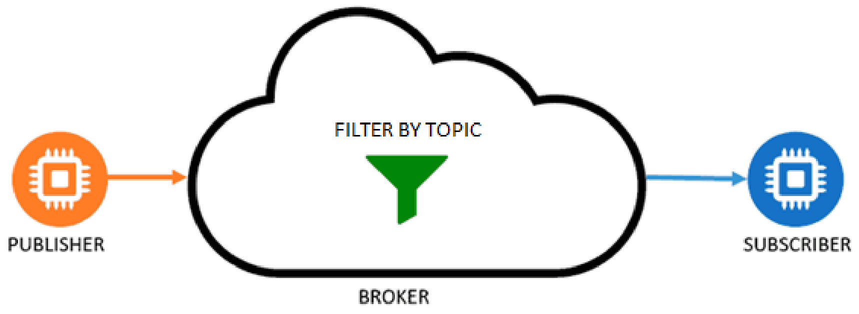


Figure 30. MQTT elements and how it works.

This kind of protocol is commonly used in IoT (Internet of Things). Thanks to the capacity of filtering the messages depending on the topic. For example, it could be possible to just send orders to the sensors outside a house or to a specific room. As a result, broadcast messages or more filters in receptors are not needed, decreasing the network traffic. That grouping of elements can be done applying wildcards to the topics. Wildcards can be used only by subscribers. There are two types of wildcards:

- One level wildcard '+': It can substitute just one level of the topic hierarchy. For example, if a subscriber is listening at topic '+/order/turn-ON', it receives all the messages published in topics, such as: 'car/order/turn-ON' or 'parking/order/turn-ON'.
- Multiple level wildcard '#': It can replace many levels of the topic hierarchy. It can just be used at the end of the topic. For example, if a device is subscribed at topic 'car/#', it received messages coming from publishers sending to topics like: 'car/order/stop' or 'car/info/battery'.

The MQTT packet format is shown in Figure 31. Each message is made up of three parts [53]:

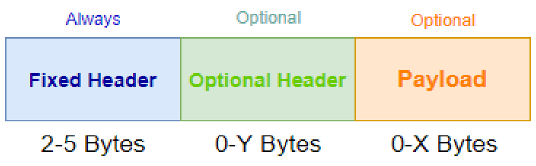


Figure 31. MQTT message format.

- Fixed header: It is part of all the messages. This field consists of the control field of 1-byte size and the packet length field of size variable between 1 and 4 bytes.
- Optional header: It is optional and it stores additional information for specific messages.
- Payload: It is optional; its size can be as long as the whole message size reach 256MB.

In the next explanations, the most important types of messages will be just stated, however, for further information it is included all the information about MQTT messages in Annex 1.

Another interesting property of MQTT is that three levels of QoS (Quality of Service) are implemented [53]:

- QoS level 0: At most once delivery. The message is sent following the best efforts of the underlying TCP/IP network. No acknowledgement is expected. The message can arrive at the server once or not at all.
- QoS level 1: At least once delivery. Now the message is answered by a PUBACK message. The publisher will resend the message if it is not acknowledging in a determined period. Duplicated messages can be sent to the subscriber.
- QoS level 2: Exactly once delivery. This is the highest level of messaging sending, for use when duplicated messages are not acceptable. In the Table 3 is shown the QoS level protocol flow.

Client	Message and direction	Server
QoS = 2 Message ID = x **Action:** Store message	PUBLISH $\longrightarrow$	**Actions:** • Store message ID • Publish message to subscribers
	PUBREC $\longleftarrow$	Message ID = x
Message ID = x	PUBREL $\longrightarrow$	**Actions:** Delete message ID
Action: Discard message	PUBCOMP $\longleftarrow$	Message ID = x

Table 3. QoS level 2 message flow[3] [53].

We must ensure that messages reach their destiny, so QoS level 1 or 2 can be useful for our project. In order to know which is the better QoS level, a study must be done to analyze how each of them affects to the network traffic as a future line to optimize the system. At the moment, in the exchange of messages, the QoS of level 2 will be used.

In our project, all the elements are at the same time publishers and subscribers. All of them need to send and receive data simultaneously. The broker is installed in the server where the back-end software is stored. Eclipse Mosquitto is the broker installed in the Raspberry. It is an open-source and lightweight MQTT broker and suitable for all devices, even for low power single board computers [54]. This broker provides the possibility of implement passwords to establish a connection with it. These passwords are included in a cypher file, in which there are written the username of the element and its password. In that way, it is guaranteed that just the elements of the system can connect to the broker. If there is not any kind of security to access the broker, a malicious device can connect to the broker easily, and manipulate the system.

4.1.1 Implementing MQTT: Eclipse Paho MQTT Python library

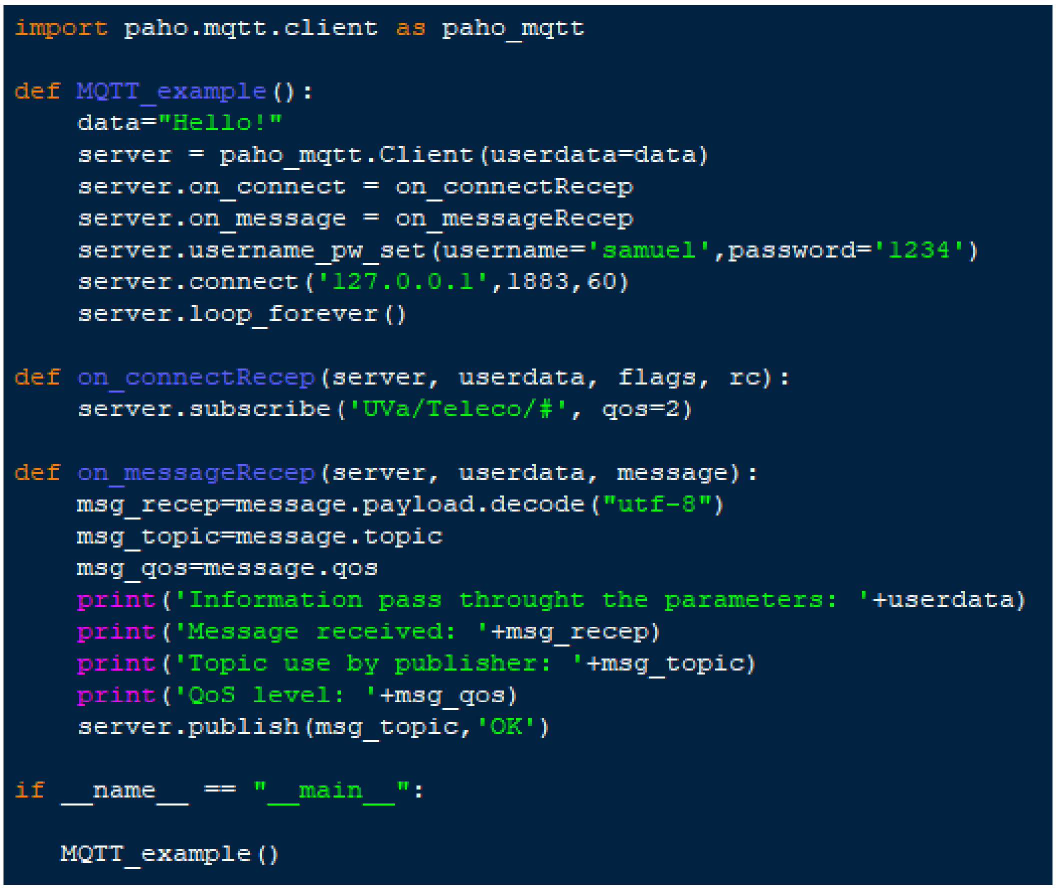
The Eclipse Paho project provides open-source software to implement MQTT message protocol, aimed at fresh and emerging applications for the IoT [55]. This section is dedicated to

[3] For further information about the messages exchange in a QoS of level 2 communication, see Annex II.

explaining the performance of the ‘Eclipse Paho MQTT Python’ client library. It will help us to understand how the processes were designed in the back-end development.

In Figure 32, there is an example of a simple python code. It will help us to describe its operation [56]:

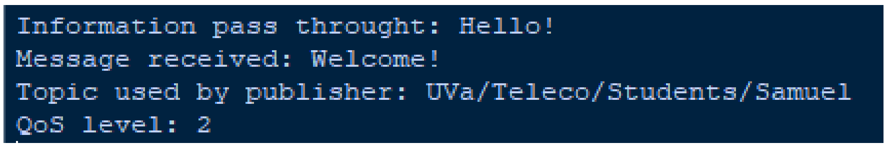
- First, it is created a client instance, to which a parameter is passed. The parameter ‘userdata’ will be available in the rest of the callbacks from the same client instance.
- The function ‘connect’ is in charge of connecting to the broker, its parameters are the IP address of the broker, the MQTT port (1883) and the timer of keep-alive (once it finishes, it disconnects from the broker).
- Due to the keep-alive timer, the function ‘loop_forever’ is used to reconnect again.
- The callback ‘on_connectRecep’ runs once the client receives a CONNACK from the broker. In this callback, the client subscribes to the topic ‘UVa/Teleco/#’. In that way, always that the client tries to reconnect to the broker it will keep subscribing to that topic.
- Finally, in the callback ‘on_messageRecep’, activated once a message arrives, it is processed all the data needed and it is sent back in the same topic ‘OK’.



```python
import paho.mqtt.client as paho_mqtt

def MQTT_example():
    data="Hello!"
    server = paho_mqtt.Client(userdata=data)
    server.on_connect = on_connectRecep
    server.on_message = on_messageRecep
    server.username_pw_set(username='samuel',password='1234')
    server.connect('127.0.0.1',1883,60)
    server.loop_forever()

def on_connectRecep(server, userdata, flags, rc):
    server.subscribe('UVa/Teleco/#', qos=2)

def on_messageRecep(server, userdata, message):
    msg_recep=message.payload.decode("utf-8")
    msg_topic=message.topic
    msg_qos=message.qos
    print('Information pass throught the parameters: '+userdata)
    print('Message received: '+msg_recep)
    print('Topic use by publisher: '+msg_topic)
    print('QoS level: '+msg_qos)
    server.publish(msg_topic,'OK')

if __name__ == "__main__":

   MQTT_example()
```

Figure 32. Example of a python code using paho MQTT.

If it is sent the message ‘Welcome!’ using the topic ‘UVa/Teleco/Students/Samuel’, it can be seen in the console of the IDLE from python the strings of Figure 33.



```
Information pass throught: Hello!
Message received: Welcome!
Topic used by publisher: UVa/Teleco/Students/Samuel
QoS level: 2
```

Figure 33. Console of the python IDLE.

4.2 TwizyLine failure management

In section 3.3 we have seen the ideal operation of the parking, however, due to some problems, some procedure may not correctly follow its flow of operation. The system must be prepared to detect and solve these situations. This section will be devoted to explaining what kinds of failures can occur, and how the system should act to repair them.

First, it has been followed the instructions given by the ISO 26262. These set of rules defined ‘HARA’. ‘HARA’ is a method to identify and categorize hazardous events of items and to specify safety goals, in order to avoid unreasonable risk. The difference between the following terms must be clear to define the kind of failures [57]:

- Harm: Physical damage to a living being.
- Risk: Probability of a certain event occurring along with the severity of the damage
- Safety: Process that avoids risks of unreasonable damage.
- Hazard: Power source of unreasonable risk of harm or security threat.

HARA method identifies possible hardware or software failures using FMEA. In order to apply the FMEA (Failure Mode and Effect Analyses) technique, a table must be created with all the possible failures, following these steps:

- Discuss with experts in the field.
- Determine the level of detail we want in the specific table.
- List the failure modes.
- For each failure mode, assign severity (from 1 to 10).
- For each failure mode, assign the probability of occurrence (from 1 to 10).
- Chances of detecting the fault.
- Calculate the risk priority number (RPN), which equals to: S (severity) x O (occurrence) x D (detection)
- RPN risk situations are ordered and addressed.

At the moment, the failure management system implements the four first points of the FMEA technique, however, in the next developments of the TwizyLine project the rest of the point will be introduced. In the fourth point, it has been defined two kinds of failure depending on the severity: error and warning. Then, it is needed to define what nomenclature it is going to be used.

- It is called ‘failure’ to any kind of possible issue that it is reported to the server. In this system, there two types of failure: warning and error.
- It is called ‘warning’ to those failures which suppose a very low risk, but it can provoke some damage to the element operation which is reporting the failure. Despite this, it is

notified to the administrator in case they decide to check the element to solve that warning.

- It is called ‘error’ to those failures which interrupt the element operation. This kind of failures suppose a high risk. If the error appears in a physical object, such as a car or a barrier, they must be stopped and then repaired in order to restart, once it has been reported to the administrators.

Once it is defined the kind of failure, it is relevant to classify the failures depending on which element reports it. There are four elements in this system, so there will be four sorts of failures in this classification:

- Parking failure: Parking can report problems about barriers, turnstiles, or issues in some procedure if they are not following the correct flow.
- Twizy failure: Twizys notify their failures to warn about them to the administrators. The difference between warning and error is very important here because it can determine if the car must stop or not. For further information, check the Final Degree Project of Ignacio Royuela González [14].
- Server failure: The server does not report itself its own failure. There is a log where all the operations done by the back-end software is saved. So, the administrator can be able to check what is happening at code level if it is noticed an anomalous behavior in some process.
- Application failure: Some operation asked by the application to the server can be denied, for example, because credentials of login are wrong or because the user has no permissions.

For the parking and Twizy failures, the way of proceeding is similar. Any failure is reported to the server, where the back-end software saves that report into its database and sends in real-time to the administration application. Administrators can ask for the no repaired failures reports any time whenever they want, through the application (for further information check Adrian Mazaira Hernández Final Degree Project [15]). If the report notified an error the administrator must fix it.

The difference of the application failures with the previous ones is that no reparation of the administrator is needed. In this case, the error does not mean to stop drastically the app [15]. The server failures are most difficult to notice. In this case, a back-up system is implemented to keep proving service to the clients if the back-end software crashed drastically (check 4.6.2 for more information about the back-up system).

In Annex III, there is a table with all the failures implemented that can be managed by the back-end.

4.3 Database managers and TwizyLine Database

To manage properly a large number of data simultaneously, it is required a warehouse where to store all that information. That is the aim of the database. The system has its own database, where all the relevant information about users, Twizys, parking and other elements is updated and saved. In the next two sections, it is exposed the software tool used to manage the database and the composition of the TwizyLine database.

4.3.1 Database managers

A Database Management System (DBMS) is a program (or set of programs) that allows users to perform the following operations on the database: specifying the structure, type and restrictions of the data; and enabling the insertion, updating, deletion and consultation of the data [58]. It is important to distinguish between the database and the DBMS. The first is the warehouse where the data is stored and the DBMS is the tool needed to access and manipulate that warehouse of information.

During the development of this software, three different DBMS were tried out to manage our database, which are: SQLite, MySQL and MariaDB. At the beginning SQLite was used, however, due to the large quantity of information and simultaneous connections, sometimes it crashed. MySQL and MariaDB are very similar, in fact, MariaDB is a fork of MySQL. Both are more scalable than SQLite, so they are suitable for projects of our characteristics. Below in Table 4, there is a comparison between these three DBMSs.

SQLite	MySQL	MariaDB
Open Source	Oracle	Open Source
RDBMS (relational DBMS)	RDBMS (relational DBMS)	RDBMS (relational DBMS)
Low scalability	High scalability	High scalability
No security services	Authentication and codification	Authentication and codification
Medium performance with few simultaneous connections	High performance	Higher performance than MySQL under the same test conditions
Installable	Compatibility issues with Raspbian 10 Buster	Installable
No further services	Database engines to provide more services to manage the database	It has the same database engines as MySQL, and also, it has other ones such as Aria (make more resistant against crashes)

Table 4. Comparison between three different databases [59][60][61].

Due to the research shown in Table 4, MariaDB was chosen to be our DBMS. On one hand, SQLite was replaced mainly because of the problems it has with scalability and security. In our project, a large quantity of data can be managed at the same time, and this data must be saved in a safe warehouse. On the other hand, MySQL has some problems of compatibility with the operating system installed in the server (Raspbian 10 Buster), and in addition, although MariaDB and MySQL are quite similar, MariaDB provides more services and better performance in the same network conditions [60]. It is relevant to notice that MySQL belongs to Oracle, but MariaDB is open-source [4].

[4] For further information about how the connection to the database is done with python, check Annex IV.

4.3.2 TwizyLine database

The database model that it is going to be used is the relational model. It is based on a grouping of data into interrelated two-dimensional tables. Databases should be normalized in order to avoid integrity issues and data redundancy. The process of normalization consists in a series of steps to decompose the data in different tables, in which its structure is considered to be optimum to be implemented and manage [58]. In each table, there is at least one primary key. The primary key is the attribute which identifies a single entity (one register in the database).

To create our database, it is important to identify what are: the entities, the table names; the attributes, the fields of each table; and the relations between the different entities. There are different types of relationships between tables: one to one, one to many, 0 to many… For example: a subject is taught by just one professor (relation 1 to 1), and many students attend to that subject (relation 1 to many entities). To understand better the TwizyLine database shown in Figure 34, the relationships types used are:

- —‖ : This symbol indicates 1 entity obligatory.
- —o+ : One or cero entities related.
- —o< : Cero or many entities.
- —|< : One or many entities.

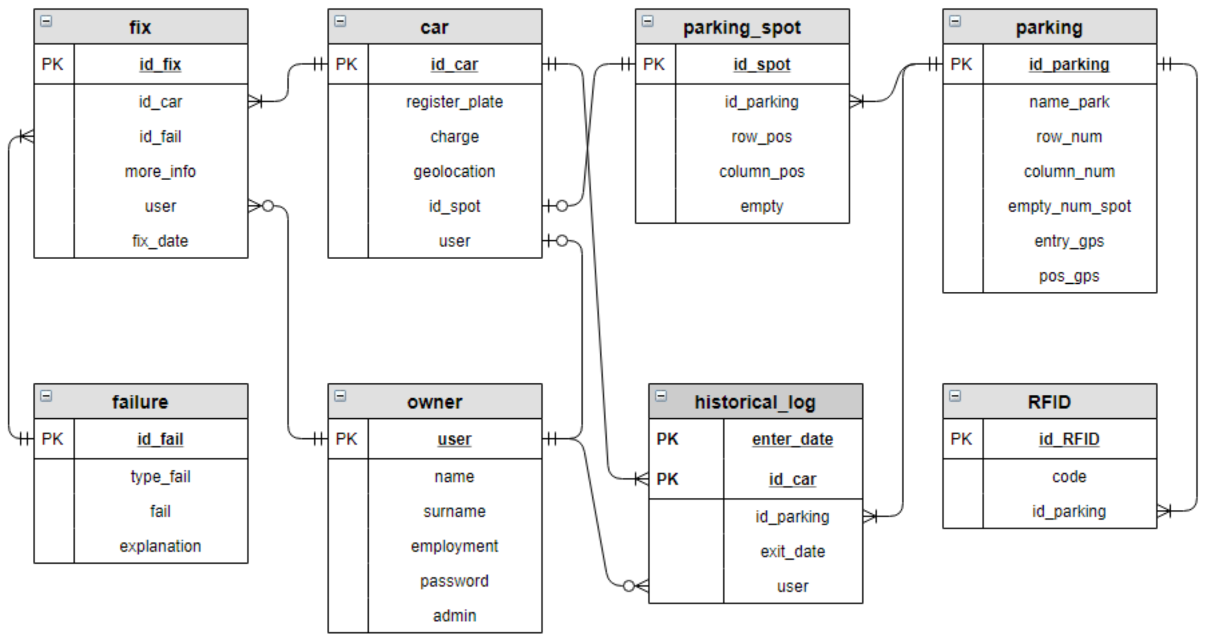


Figure 34. TwizyLine database.

The entities that made up our databases are:

- 'car': The whole fleet of Twizys is here saved. For each car is possible to know: its register plate, its position, if it is parked in some TwizyLine parking or which user is driving it.

- ‘parking’: All the parking areas are registered here. There are two GPS positions for each parking: one for the leaving zone to check the closeness of a car to it and another for the middle of the parking, to give that information to the application.
- ‘parking_spot’: Information related to all the parking spots from the TwizyLine parking.
- ‘RFID’: As RFID codes are large and sometimes difficult to deal with, the RFID tags from all the parkings are normalized to a model that can be extrapolated to any parking of any dimension. ‘id_RFID’ stores the real RFID code and the field ‘code’ stores the new codification given to deal with it. In order to differentiate between the new codes, the ‘parking_id’ is also saved. In Figure 35, there is an example of the numbering of the RFID tags (red number) for the TwizyLine parking that we will work with later on this document.

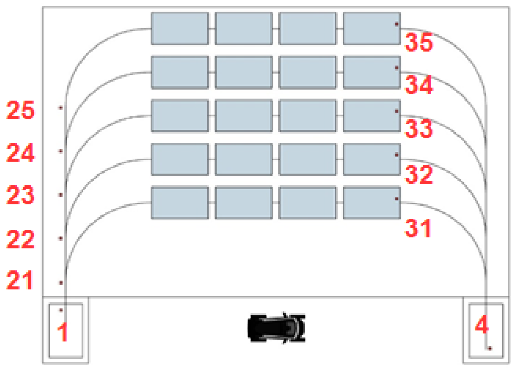


Figure 35. TwizyLine with the RFID tags numbered as in the database.

- ‘owner’: Table to register all the users of the application, especially relevant are the fields of ‘password’ and ‘admin’.
- ‘historical_log’: Thanks to this table, the system knows all the movements of all the Twizys. It is possible to know: in which parking it is and it was, when it has entered or go out from any parking, or the users who has driven it.
- ‘failure’: All the information about types of failure are store here (in section 4.2 this topic is treated in depth).
- ‘fix’: In this table, it is saved all the errors or warnings from the whole fleet of cars. Furthermore, once a car is repaired, the employee who has fixed it and the time when it has been fixed is also included

For further information of the TwizyLine database, such as the type of data stored in each field, the attributed length or properties of each attributed, it is provided in the Annex IV the code needed to create this database.

4.4 TwizyLine messages

Messages exchanged between the different elements are defined. Each TwizyLine message has a descriptive name and some of them have some attributes. These attributes are additional information to process correctly the message received. In this chapter, messages used by the back-end are going to be split up into three groups according to the elements involved in the exchange of messages.

In each table from the next sections, it is shown the publisher, the subscriber, the topic and the payload of every message. It is important to highlight that those levels of topics between brackets mean that level should be replaced for the real number. For example, topic '(id_car)/battery' means that this message is going to be sent in topic '1/battery' or '2/battery' or whatever id from any Twizy of our system next to '/battery'. As it was shown in chapter 4.1, when MQTT was explained, the subscriber must be listening in topic '+/battery' to receive the battery information from the whole fleet of Twizys.

4.4.1 Messages between the back-end and the Twizy

Mainly the back-end software sends orders to the Twizy and the messages sent by the car are information and acknowledgement to orders. Below in Table 5. Messages exchanges between the Twizy and the server., you can see the whole set of messages.

Publisher	Subscriber	Topic	Payload
Twizy	Server	(id_car)/battery	(0-100) OR -1
Twizy	Server	(id_car)/location	(latitude,longitude) OR No signal OR GPS not connected
Twizy	Server	(id_car)/info	CONNECT regist_plate
			TIMEOUT
			RFID (ID number)
			WRN (error code) [attribute]
			ERR (error code) [attribute]
			AM-ON OK
			AM-OFF OK
			CONTINUE OK
			PAUSE OK
			GOTO OK
			STANDBY OK
Server	Twizy	(id_car)/order	CONNECTED
			STANDBY
			AM-ON
			AM-OFF
			CONTINUE
			PAUSE
			GOTO (initialSpeed) (defaultFork) [T(RFID) [newSpeed]…] [V(RFID) newSpeed…] S(RFID)
			RESTART

Table 5. Messages exchanges between the Twizy and the server.

The meaning of each of these messages are:

- The position and the battery are sent periodically every 2 seconds by each car. If the battery is '-1', there has been some failure with the car. So, this failure is going to be notified to the administrators. The same situation happens with the value 'No signal' and 'GPS not connected', one leads to a warning and the other to an error, but both are going to be sent to the administrator.
- An RFID message is sent once a Twizy reads an RFID tag. This message is used by the car to send information about its position inside the parking.
- A 'TIMEOUT 'message is sent when the obstacle sensor detects that an object is in front of it.
- The first time a Twizy sends a 'CONNECT (register_plate)', its credentials are stored in the database.

- ‘WRN’ and ‘ERR’: These messages are used to notify the server the back-end that a failure occurs in the Twizy. Depending on the type of failure, it is sent ‘WRN’ for warning or ‘ERR’ for error.

The rest of the messages are going to be used to make the car move autonomously, most of them are going to be acknowledged to ensure that the car has received correctly the order, so this kind of message appears in the table with the same name but with an OK added at the end. Their meanings are:

- ‘AM-ON’ and ‘AM-OFF’: Autonomous mode activated or manual mode activated.
- ‘PAUSE’ and ‘CONTINUE’: Thanks to these orders, it is possible to stop and then to move the car whenever it is required meanwhile the car is still in autonomous mode. This gives us much flexibility, for example, to test the car the administrators or in large parking where the logic of its functioning needs these orders to provide the parking a service faster.
- ‘STANDBY’: The car passes to the low-cost energy status inside the parking. This low-cost energy status helps to charge faster the car, as it is almost not consuming much energy.
- ‘GOTO’: Once it is in autonomous mode, this message sends the route the vehicle must follow. This route is determined depending on the number of RFIDs and speed specified in this message. The speed is in km/h and the parameter “defaultFork” can be: ‘L’, which means to follow the left bifurcation by default; or ‘R’, which means to follow the right path by default. The last RFID indicated with an ‘S’means where the car must stop.
  For example, “GOTO 5 L T26 V27 3 S28” message, means that the Twizy start with 5 km/h as its initial speed and it follows the left bifurcation of the path by default, however, when it reaches the RFID number 26, it turns to the right bifurcation. Then, when the vehicle reaches the RFID 27 its speed is changed to 3 km/h and finally when it reads the RDIF 28 it stops. For further information and examples of this message, check the Ignacio Royuela’s Final Degree Project [14].
- ‘RESTART’: This message is used to restart a car once it has been stopped due to an error.

4.4.2 Messages between the back-end and the application

Depending on which kind of user access to the application some information will be restricted. Administrators have messages to ask the server for more information. That is why the messages sent between the back-end and the application is going to be divided into 3 groups.

Publisher	Subscriber	Topic	Payload
Application	Server	log	LOGIN/user/password
Application	Server	log	LOGON/user/name/surname/employment/password
Server	Application	log	CONFIRM/username
Server	Application	log	DENY/username/ErrCode
Server	Application	(user)/app	ADMIN/(0 o 1)

Table 6. Message exchange between the application and the server during the log process.

The messages to start the session in the application are described in the table above (Table 6). ‘LOGIN’ is used for a user who has already register previously and ‘LOGON’ for a new user to create an account. ‘LOGIN’ can be denied if the credentials are wrong and the ‘LOGON’ can be denied also if the username already exists. If everything is correct and the log message is confirmed, it is sent to that specific user application a 0 if it is a normal user or a 1 if it is an admin. This information is going to be processed by the app to determine which session should start [15].

Publisher	Subscriber	Topic	Payload
Application	Server	(user)/server	REQUEST-PARKS
Server	Application	(user)/app	NEW-PARK/id_parking/name_park/latitud/longitud/Twizys disponibles/Plazas disponibles
Server	Application	app	INFO-PARK/id_parking/Twizys disponibles/Plazas disponibles
Application	Server	(user)/server	PICK-UP/id_parking
Server	Application	(user)/app	TWIZY-OUT /id_vehicle

Table 7. Message exchange between the application and the server required to take out a Twizy.

The messages exchanged in a session to pick up a Twizy from our parking are described in Table 7. The user application asks the back-end for information about the status of the parking (‘REQUEST-PARKS’), and this information is periodically updated each 30 seconds (‘INFO-PARK’). Once the user wants to take a Twizy from a TwizyLine parking, a ‘PICK-UP’ message is sent. The id number of the car which is going to go out is sent to the app. So, the user knows which car has assigned to him.

Publisher	Subscriber	Topic	Payload
ApplicationAdmin	Server	(user)/admin_server	REQUEST-ERRORS
Server	ApplicationAdmin	admin o (user)/admin	ERR/id_fix/(id_vehicle o id_parking) /failure/explanation/error_date/more_info
ApplicationAdmin	Application	(user)/admin_server	ERR-FIX/id_fix
ApplicationAdmin	Server	(user)/admin_server	ASK-PARK/id_parking
Server	ApplicationAdmin	(user)/admin	PARK/num_row/num_column/ [(row_pos;column_pos;id_car)/...]
Server	ApplicationAdmin	(user)/admin	PARK-ERR
ApplicationAdmin	Server	(user)/admin_server	ASK-CAR/id_car
Server	ApplicationAdmin	(user)/admin	CAR/[enter_date;id_parking;exit_date;user/…]
Server	ApplicationAdmin	(user)/admin	CAR-ERR

Table 8. Message exchange between the application of an administrator and the server.

Messages shown in Table 8 are messages available just for administrators. They can ask for: the log of errors, the status of a parking or information about a Twizy. If the employee, who is log as an administrator fixes a vehicle, he must send to the back-end that it has been repaired a specific car through the app (‘ERR-FIX’). ‘PARK-ERR’ and ‘CAR-ERR’ are used to notify the application that the parking or the Twizy which is asking for information is not on the TwizyLine database.

4.4.3 Miscellaneous messages

There are other messages that the back-end uses that are not sent to any application or Twizy. For example, MQTT is employed to communicate with the four barriers and with the turnstile. There are two goes down for barrier 1:

- ‘B_DOWN-RFID’ means that the barrier 1 must close after a car has entered the leaving zone.

- 'B_DOWN' is used for barrier 1, once a Twizy passes near a parking and due to its close GPS position to the entry of the parking the barrier has to be up, but it does not go in the leaving zone, so the timer of barrier 1 expires and it closes.

This differentiation is especially relevant for debugging applications, to know exactly what is happening in the parking. For the turnstile, there are two messages one to enable the turnstile 2 once the car is already in manual mode and the other to notify the back-end that the turnstile number 1 or 2 has turned. Below in Table 9, there are these messages.

Publisher	Subscriber	Topic	Payload
Server	Parking barrier	barrier/(id_parking)/(1:4)	B_UP#(id_car)
Server	Parking barrier	barrier/(id_parking)/1	B_DOWN-RFID#(id_car)
Server	Parking barrier	barrier/(id_parking)/ (1:4)	B_DOWN#(id_car)
Server	Turnstile	turn_enable/(id_parking/2	ENABLE_TURN_2
Turnstile	Server	turn/(id_parking)/(1 o 2)	OK
Controller	Server	init_remote_server	INIT REMOTE SERVER
MQTT process	MQTT process	MQTT/enter/(S oT)/(id_car)	RFID (number) (id_car) (id_process)

Table 9. Messages exchange between the server and other elements of the TwizyLine system.

There are two more messages that they are going to be explained more in detail in the next chapters. One is used to initialize the back-end implementation remotely through a controller and the other is used to communicate inner processes of the back-end software.

4.5 TwizyLine processing architecture

As it has been explaining along this document, the back-end implementation must be able to control different elements and to manage many more operations of the system simultaneously. At the software level, this parallel processing can be done in two ways: using threads or multiprocessing. It is important to differentiate both elements:

- Multiprocessing: It is created multiple processes. Each process provides the resources to execute a program. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution.
- Threading: A thread is an entity within a process that can be scheduled for execution. The program began with a main thread that later on it splits up into various threads. All threads of a process share its virtual address space and system resources. Besides, each thread maintains exception handlers, thread-local storage, a scheduling priority, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled [62]. It is lightweight compared with a process. They are not interruptible or killable.

The back-end software implemented uses multiprocessing due to the following advantages: separate memory space, eliminates most needs for synchronization primitives, child processes are killable or interruptible, it takes advantage better of multiple CPUs and python multiprocessing includes more useful abstractions than python threading [63].

One of the main advantages that we are going to take advantage of is the possibility of killing child process. This is very important to be running just the necessary processes in the system, to reduce the overall CPU memory employed. In addition, this action should be done correctly, otherwise, our system could be slowed due to the accumulation of defunct processes occupying uselessly space in memory. These defunct processes are also known as ‘zombie processes’. In the following section is shown how the processes used to control the TwizyLine system are organized in the back-end software.

For further information of the code developed, it is possible to read it on the repository of GitHub from the GCO (Group of Optical Communications) from the University of Valladolid [64]. The file is called ‘hidraserverMySQL.py’ and it is inside the folder ‘Server’. The name of that file comes from: ‘hidra’, it is the name of the back-end (equivalent to ‘Sentilo’ from the Barcelona project shown in section 2.2; server, because it is where the back-end is running; and ‘MySQL’ because it uses the syntax of MySQL, which is used to work with MariaDB.

4.5.1 Process diagram of the back-end implementation

To create this structure of processes the multiprocessing python library was used. This library provides a FIFO piping system to send data between two different processes. To organize the process architecture, the processes are classified depending on when they are born. Thus, the architecture is divided into levels (Figure 36). Those processes belonging to level 0 or level 1 are going to be never killed meanwhile the back-end software is running. Those processes belonging to levels below 1 are going to be killed once they finished to do their task.

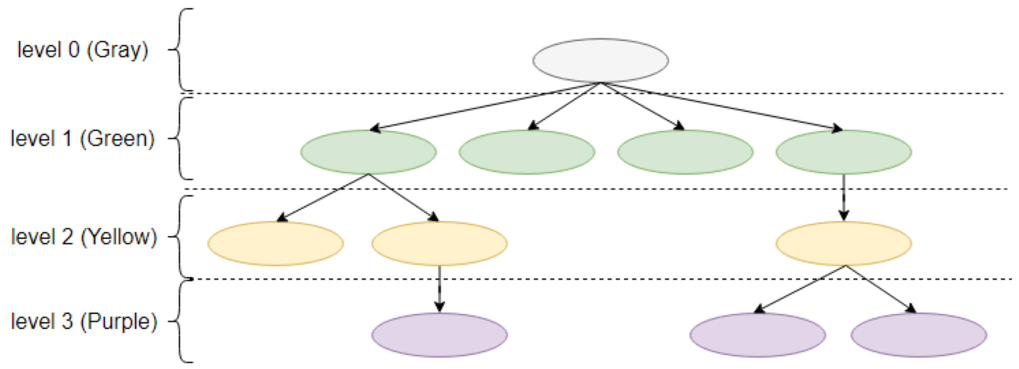


Figure 36. Example of the process architecture with 4 levels.

At the beginning, in the back-end software developed there is just the main process (level 0) which launches other 12 processes (level 1). So, always 13 processes are running at the same time, except in the initial start-up of the software. The list of processes launched is shown in Figure 37. In the next sections, it is going to be explained the function of each process and when each message shown in section 4.4 is sent.

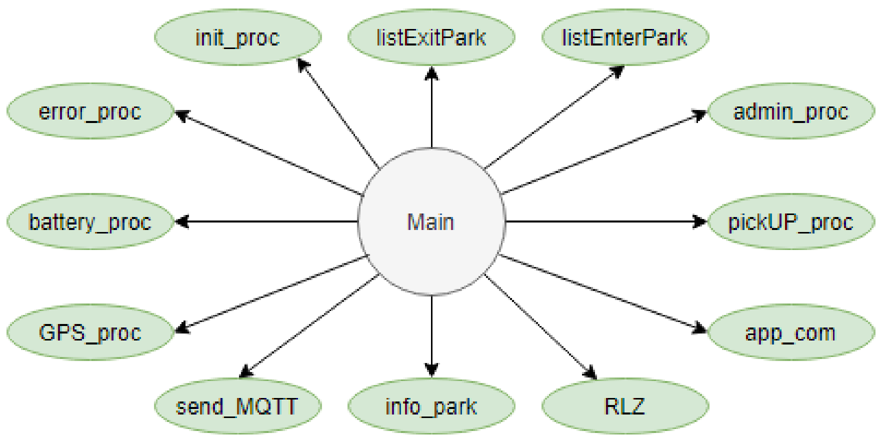


Figure 37. Main processes (level 0 and 1) from the TwizyLine architecture.

4.5.2 Miscellaneous processes

There are some processes of level 1, that do not belong to any specific procedure. These processes are the following ones:

- 'send_MQTT' process is employed by all the processes which want to send a message through MQTT protocol. There is one queue called "q_sendMQTT", which is used by other processes to send the topic and the message to 'send_MQTT' process.
- The process called 'init_proc' is in charge of adding to the table 'car' from the database new Twizys to send them the connection message ('CONNECTED'), to acknowledge the Twizy that has been connected to the TwizyLine system.
- The battery level sent periodically with a message from the car is stored in the database by the process called 'battery_proc'.
- The failure management is controlled by the 'error_proc' process. It is able to store in the database incoming failure messages ('ERR' or 'WRN') from the car in the 'fix' table, registering not only the type of error but also the time when that failure has appeared. Besides, it sends the failure in real-time to those administrators who are connected in that moment to the application.

4.5.3 Processes involved in the outside stage

There are three main processes involved in the outside stage: 'GPS_proc', 'listEnterPark' and 'RLZ'.

Like 'battery_proc', the 'GPS_proc' is in charge of listening to the periodical message of the GPS position from the whole fleet of cars and add that information to the database to the table 'car'. This process is also responsible to rise the barrier 1 if the car gets close to any parking. Obviously, before rising it, it is checked if the parking is full or if the barrier 1 has been already gone up to avoid repetition of messages sent.

The process 'GPS_proc' adds the id of the parking that opens barrier 1 to a list stored in the process 'listEnterPark' through a queue. Therefore, in 'listEnterPark' those parking that already have their barrier 1 opened are stored. In this way, as it was said in the previous paragraph, 'GPS_proc' checks in this list that it is not already raised and thus we avoid repetition of messages. Once barrier 2 is closed, this process should be communicated through another queue to remove the parking id from the list, and allow new vehicles to enter the leaving zone.

The process which determines when an enter procedure began is the ‘RLZ’ process. Its words mean ‘RFID from Leaving Zone’, because this process is listening to all the messages from cars with data of the RFID with code 1 of the parking (check Figure 38, where RFID with code 1 is on the leaving zone). Once it arrives at the server, ‘RLZ’ process closes barrier 1 from that parking and the ‘entry’ process is launched. ‘entry’ process is a child process from ‘RLZ’.

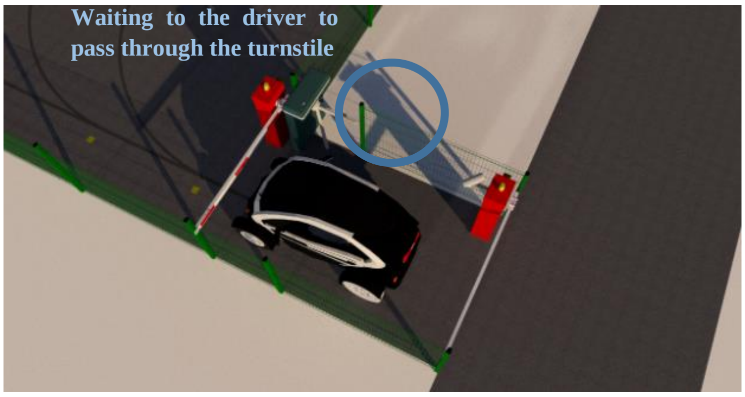


Figure 38. Twizy placed in the leaving zone with the driver inside the car.

4.5.4 Processes involved in the enter procedure

Now, we are going to analyze the processes and the functionalities of the back-end software involved in the enter procedure (Figure 27, flow diagram), that are, ‘entry’and ‘entMQTT’.

Then, first in the ‘entry’ process the system is waiting for the turn message to know that the driver is out the leaving zone. It is subscribed to topic ‘turn/(id_parking)/1’. Then, once the driver passes through the turnstile, the following operations take place in the ‘entry’ process:

- ‘AM-ON’ message is sent to the Twizy, so the vehicle turns to autonomous mode.
- Barrier 2 arises.
- Its parking spot is calculated. Then, a ‘GOTO’ message is sent
- Tables ‘parking’, ‘parking_spot’, ‘car’ and ‘historical_log’ from the database are updated.
- ‘entMQTT’ process is launched. It is a child process of ‘entry’. The turn RFID of the GOTO and the car id are parameters passed to ‘entMQTT’ process.
- ‘entry’ process keeps waiting for the MQTT message from the ‘entMQTT’ process.

Before continuing explaining the ‘entMQTT’ process, it is remarkable to explain how the ‘entry’ process has calculated the parking spot of the Twizy. The algorithm used to determine to which spot must be sent the Twizy consist of the following steps: first, its battery level determines its spot, those vehicles with low battery level are sent to the last rows where the charge power is higher, and the other way around, those vehicles with high battery level are sent to closest rows where charge power is lower. In order to calculate exactly the row, the total possible battery 100% is divided into the number of rows, in this way it is known the gap of battery levels to each row can go.

For example, for a parking with 3 rows, the closest row to the leaving zone will be parked cars with 100% to 67% of battery, in the middle row from 66% to 34% and in the further one from 33% to 1%. However, maybe the ideal row is full, so, another position must be calculated. As the rows with higher charge power are more valuable because they charge faster a vehicle, the algorithm tries to park it on the row closer to the entry. The last option is always the last row. The possibilities that a car can be sending 0% or -1 due to a warning to the back-end software are also treated, those cars are sent to the closer row possible to make easier the employer to repair it if it is required. Next, there are some examples of how this algorithm works.

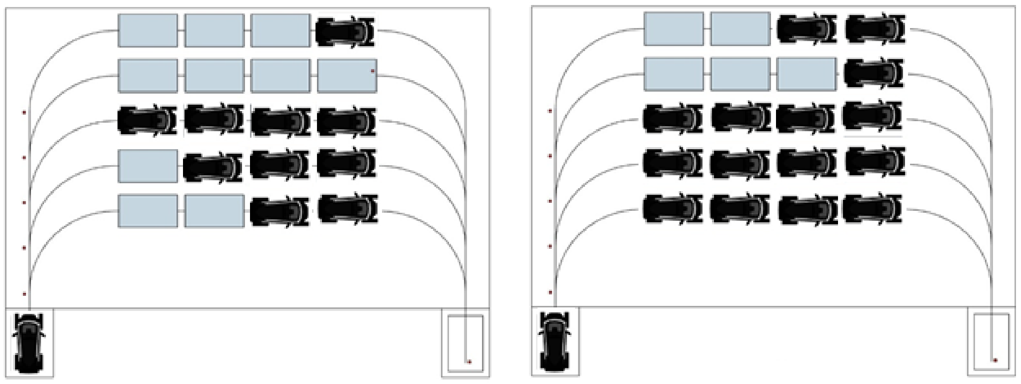


Figure 39. Parking situation A (left), parking situation B (right).

In the situation A from Figure 39, the car which battery level is of 50% should be parked in the third row, however, there are no empty spots. So, the next row to be checked where it can be parked is the second row. In this row, there is one empty spot so the Twizy is sent to spot in row 2 and column 1. In situation B from Figure 39, the car which battery level is 25%, it should be parked in the second row but there is no space. So, the system checks row 1, but it is full too. Then, row 3 is checked and there are no spots free again. Finally, it is check row 4 and there the Twizy can be parked. So, it goes to the spot in row 4 and column 3.

Coming back to the ‘entryMQTT’ process lauched by the ‘entry’ process, now the Twizy is driving autonomously towards its spot. Once the car reads the turn RFID from the row of its spot, it sends an RFID message to the server saying that has read that RFID and it turns. The process in charge of listening to that message is the child process called ‘entMQTT’, which is subscribed to the topic ‘(id_car)/info’. The mission of this process is to receive the RFID message with the RFID number and from the car ordered by the father process. Once it listens to the correct RFID from the proper Twizy, ‘entMQTT’ notifies through MQTT to the father process that it has been received using the topic ‘MQTT/enter/T/(id_car)’ (Table 9, from the section miscellaneous messages).

Now, ‘entry’ process order to go down barrier 2, the parking is removed from the list of block parking of the ‘listEnterPark’ process using a queue (as it was explained in the previous section) and a new ‘entMQTT’ process is launched. This time the stop RFID of the GOTO message is passed and the car id to the child process. It will work as previously with two differences: the first is that now it is listening for the stop RFID or a 'TIMEOUT' message (Figure 40) sent by that Twizy, and the second is that the MQTT topic to notify its father process about the reception of the message is 'MQTT/enter/S/(id_car)' (Table 9).

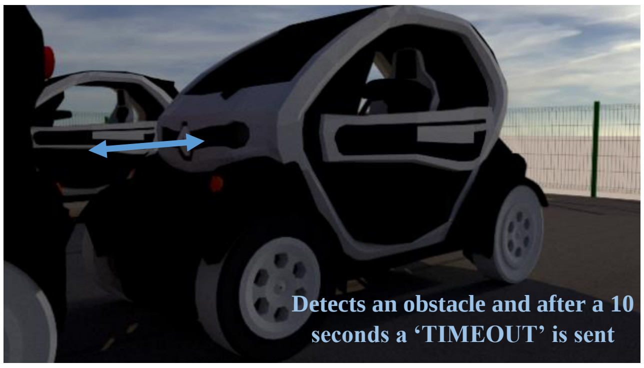

Detects an obstacle and after a 10 seconds a 'TIMEOUT' is sent

Figure 40. Twizy stops due to detect an obstacle in front of it.

Finally, when the 'entry' process received the notification of the second 'entMQTT', it sends a 'STANDBY' message to that car and the list of child processes below the level of 'RLZ' process involved in this process are killed. Below there is an image to illustrate this architecture. Processes with yellow background are of level 2 and the ones with purple background are from level 3, and they are going to be killed to avoid overload in the CPU memory once the whole process is finished (Figure 41).

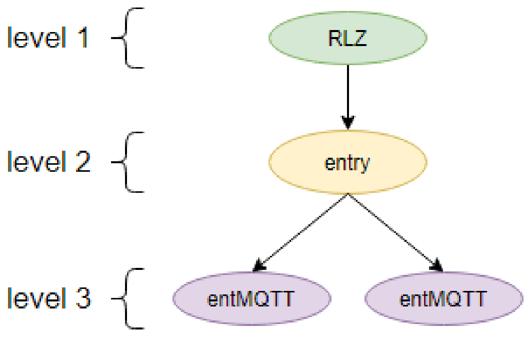


Figure 41. Schema processes involved in the enter procedure.

4.5.5 Processes involved in the communication with the TwizyLine applications

There are seven main processes involved in the communication with the TwizyLine applications: 'app_com', 'admin_proc', 'pickUP_proc', 'info_park', 'exit', 'forwarding' and 'forwarding_TO_standby'.

To start a session in the smartphone application, it must communicate with the 'app_com' process (level 1 process). In this process, messages from Table 6 are exchanged. It is used to say the application if the user who wants to enter is an administrator or a normal user. Furthermore, it is checked if the credentials from a LOGON message are correct, if not, an error message is sent to the app.

If the user is an administrator, the set of messages from Table 8 can be exchanged. The process ‘admin_proc’ (level 1 process) is in charge of interacting with the sessions of administrators. It can: restart a car once it has been repaired, send the information from any Twizy or parking, and provide a list of the failures from cars that are not fixed.

There are another two processes that communicates also with the application, which are ‘pickUP_proc’ and ‘info_park’ (both level 1 processes). ‘info_park’ is sending periodically updated information about the parking status, ‘INFO-PARK’ is message used (check Table 7). ‘pickUP_proc’ is subscribed to topic ‘(user)/server’ and it can receive two messages:

- ‘REQUEST-PARKS’ (Table 7) provides information about the status of the parking at the moment when this process detects that this message is received.
- ‘PICK-UP’ (Table 7) message is managed here. To process this message means first that the software must check if the pick-up point is free in order to avoid large queueing or collisions.
  The list of parking which its pick up zone is occupied is saved in a list stored in the ‘listExitPark’ process (level 1). Its operation is the same as the ‘listEnterPark’, once the exit procedure the parking is added to the list through a queue and once the Twizy leaves the pick-up zone the parking is removed from that list.
  Secondly, if the pick-up point is free it must determine which Twizy must go out. And at the end, the ‘exit’ process is launched (the next section talks about this process).

The algorithm programmed to determine which parking must go out consists of the Twizys available to move towards the pick-up point are those parked in the last column. So, the algorithm orders to go out the Twizy with higher battery. In case there are two Twizys with the same battery level in the last column, the one with is parked the further row is the chosen to go out. This is done to make more room in those rows with higher charge power.

4.5.6 Processes involved in the exit procedure

The processes involved in the exit procedure are the ‘pickUP_proc’, ‘exit’, ‘forwarding’, ‘forwarding_TO_standby’, and ‘listExitPark’.

The first process involved in the exit procedure is the ‘pickUP_proc’ that has been in the previous section explained. This process launched the ‘exit’ process (level 2 process). In this last process three main operations are made: send ‘AM-ON’, after the GOTO message to the pick-up zone to the Twizy chosen to leave the parking, and at the same time it orders to elevate the barrier 3. In this point, the ‘exit’ process is subscribed to the topic ‘(id_car)/info’, because it is waiting to the message with the RFID with code 4 of the parking from the Twizy moving out.

Once the Twizy reaches the pick-up zone and sends the RFID message, the barrier 3 is ordered to go down, the ‘AM-OFF’ message is sent to put the car in manual mode, the child process ‘forwarding’ is launched and the turnstile 2 is enabled. Now ‘exit’ process waits for the message from the turnstile 2 of the parking, subscribing to the topic ‘turn/(id_parking)/2’. When this message is received, the barrier 4 is elevated and the database is updated. When the Twizy leaves the pick-up zone, the barrier 4 goes down and the parking is removed from the list of block parking of the 'listExitPark' process (level 1 process, explained in the previous section). Finally, the 'entry' process auto-kills itself.

The child process 'forwarding' (level 3 process) is in charge of moving forward one spot those cars parked in the same row as the Twizy that has gone out. If there was no car behind the Twizy gone, this process just updates its parking spot to set it as free. If there were cars behind the Twizy gone, this process sends two messages to those vehicles, one to put them in autonomous mode ('AM-ON') and another one to send a 'GOTO' message to move them. In this process, the status of the parking spots is updated in the table of 'parking_spot' from the TwizyLine database. Besides, it is launched one child process for each Twizy moved forward called 'forwarding_TO_standby' and finally this process is deleted. As its name says, the new child process is in charge of sending the 'STANDBY' message to those cars, once it is received its respective end of row RFID message or the 'TIMEOUT' message by the Twizy. At the end of each 'forwarding_TO_standby' process, it is deleted.

Below it is shown a diagram (Figure 42) of the process architecture for this procedure. Those processes which do not belong to level 0 or 1 must be deleted, as it has been explained in this section.

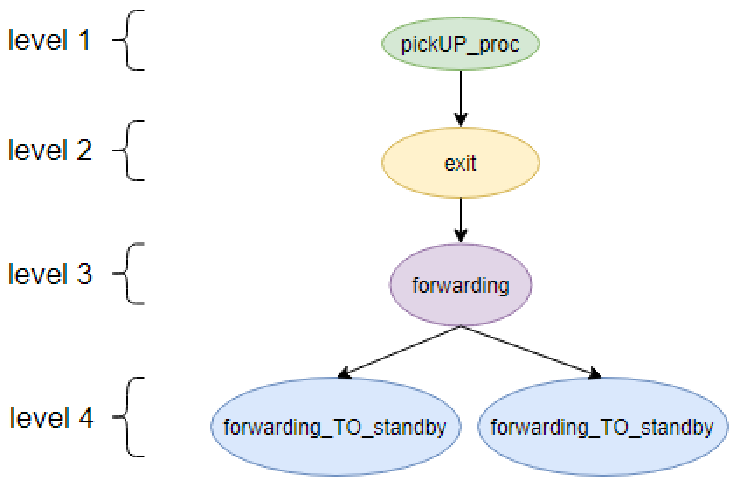


Figure 42. Schema processes involved in the exit procedure.

4.6 Implementation of more functionalities for the system

To add more functionalities to this system and to make after tests in order to test how the back-end software works, it is used an 8Bit Zero Controller to simulate MQTT messages and additional python programs to give more redundancy to the system. Next section aims at explaining how this implementation have been developed.

4.6.1 8Bit Zero Controller

8Bit Zero Controller is a controller with several buttons and it connects through Bluetooth to other devices. This device is connected to a Raspberry Pi B. A python code was programmed in that raspberry to read the different signals sent from each button of the controller. Each button sends a message through using the MQTT protocol when is pressed. Like the code used for the back-end software, this code is also stored in the GitHub repository from the GCO of the University of Valladolid [64]. The file is called ‘turnstile.py’ and it is inside the folder ‘turnstile’. Below in Figure 43, it can be seen what each button does.

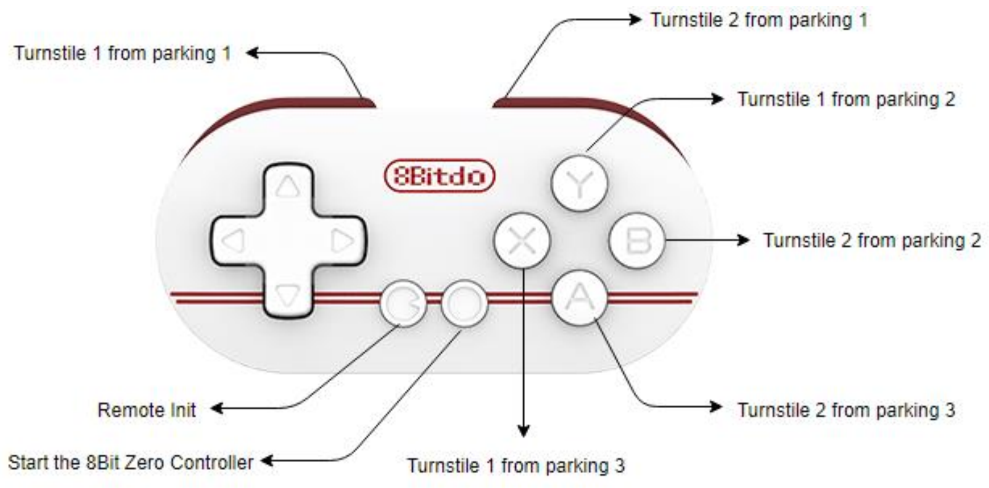

Turnstile 1 from parking 1
Turnstile 2 from parking 1
Turnstile 1 from parking 2
8Bitdo
Turnstile 2 from parking 2
Remote Init
Turnstile 2 from parking 3
Start the 8Bit Zero Controller
Turnstile 1 from parking 3

Figure 43. 8Bit Zero Controller and the function implemented in each button.

As it can be seen in Figure 43, the controller will be used for two functions: simulate different parking and to simulate the ‘INIT REMOTE SERVER’ message. It is worth it to notice that the turnstile messages are asynchronous, they are sent once the user passes through the turnstile. It is different from the barriers which movement is synchronous with the back-end software flow.

The ‘remote init’ button is going to be explained better in the next section, but its goal is to start the back-end software remotely whenever the administrator wants.

4.6.2 Back-up system

As it has been said previously, the back-end software is a critical point of the system. So, if this system is implemented in real life, it must be provided with redundancy systems to be always working. In this section, it is explained how a simple but effective back-up system is implemented.

At first, it is launched two python programs called ‘init_remote.py’ and ‘self_watchdog.py’. Their codes are also stored in the GitHub from the GCO [64] and they should be running in the raspberry which is working as a server (different from the one used to communicate with the 8Bit Zero Controller). The first one aims to listen to the ‘INIT REMOTE SERVER’ message from the controller. Once this message is received, it launches ‘hidraserverMySQL.py’. Always when ‘hidraserverMySQL.py’ starts running, its first action is to delete whatever process related to previous executions of itself. This will help us to delete possible defunct process running due to a bad disconnection of the server, such as power outage of the raspberry. The other file called ‘self_watchdog.py’, as its name says, is like a watchdog. Its goal is to check every 10 seconds if the back-end software is running, if not, a ‘hidraserverMySQL.py’ is launched. In Figure 44, there is the flow diagram for this back-up system.

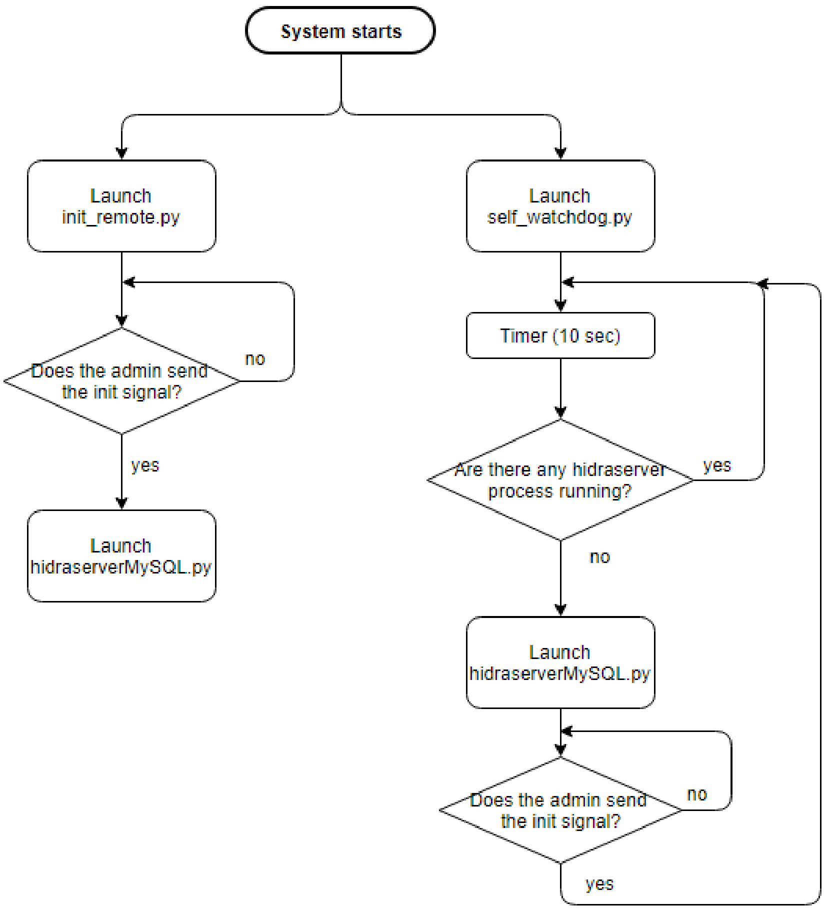

System starts
Launch init_remote.py
Launch self_watchdog.py
Does the admin send the init signal?
no
yes
Launch hidraserverMySQL.py
Timer (10 sec)
Are there any hidraserver process running?
yes
no
Launch hidraserverMySQL.py
Does the admin send the init signal?
no
yes

Figure 44. Flow diagram of the back-up system.

5 Integration and test

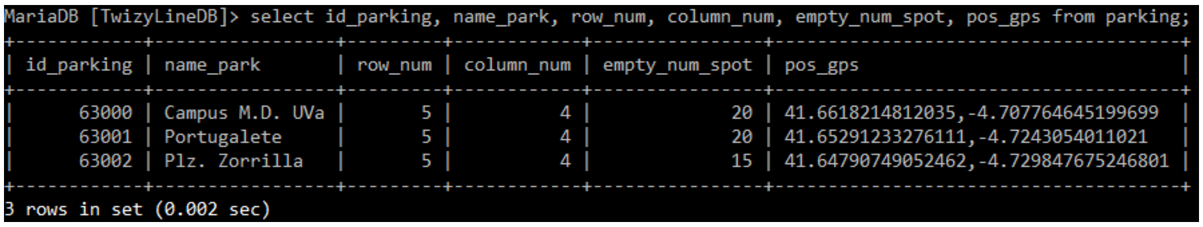
A test simulator has been created to integrate 3 different parts of the TwizyLine system. The simulator will serve first to visualize both Twizys and parking lots as in the real world thanks to Google Earth, and second, it will imitate the operation of the vehicles. For more information about the simulator and how it has been implemented, check the Final Degree Project by Ignacio Royuela González [14].

The following sections of this chapter will deal with the parking operation procedures seen in other chapters. It will also demonstrate how the failure management system works. It focuses on the physical operation, and will not go into so much detail in explaining the communication between the different elements, that is, the exchange of messages as it was already done in sections 3.3 and 4.4.

5.1 Initial conditions

To know the context in which the tests are carried out in the following sections, in this section the initial conditions of both the simulator and the database will be exposed.

Starting with the simulator, this represents 3 different parking areas located at: Campus Miguel Delibes, Plaza Zorrilla and Plaza. Portugalete. This information must coincide with that of the database, as it is seen in Figure 45.



```
MariaDB [TwizyLineDB]> select id_parking, name_park, row_num, column_num, empty_num_spot, pos_gps from parking;
+------------+-----------------+---------+------------+----------------+---------------------------------------+
| id_parking | name_park       | row_num | column_num | empty_num_spot | pos_gps                               |
+------------+-----------------+---------+------------+----------------+---------------------------------------+
|      63000 | Campus M.D. UVa |       5 |          4 |             20 | 41.6618214812035,-4.707764645199699   |
|      63001 | Portugalete     |       5 |          4 |             20 | 41.65291233276111,-4.7243054011021    |
|      63002 | Plz. Zorrilla   |       5 |          4 |             15 | 41.64790749052462,-4.729847675246801  |
+------------+-----------------+---------+------------+----------------+---------------------------------------+
3 rows in set (0.002 sec)
```

Figure 45. TwizyLine parking areas information in the database.

In Figure 45, we see that there are 5 cars on the Miguel Delibes Campus, and that the other two parking areas are empty. Regarding the database, it is also interesting to comment that 4 users have already registered.

For the following tests, the parking located in Plaza Zorrilla will be used. In Figure 46, it can be seen how the parking look like in the simulator. The four red points are the four barriers closed. If some barrier is open, it is shown a green point in later figures.

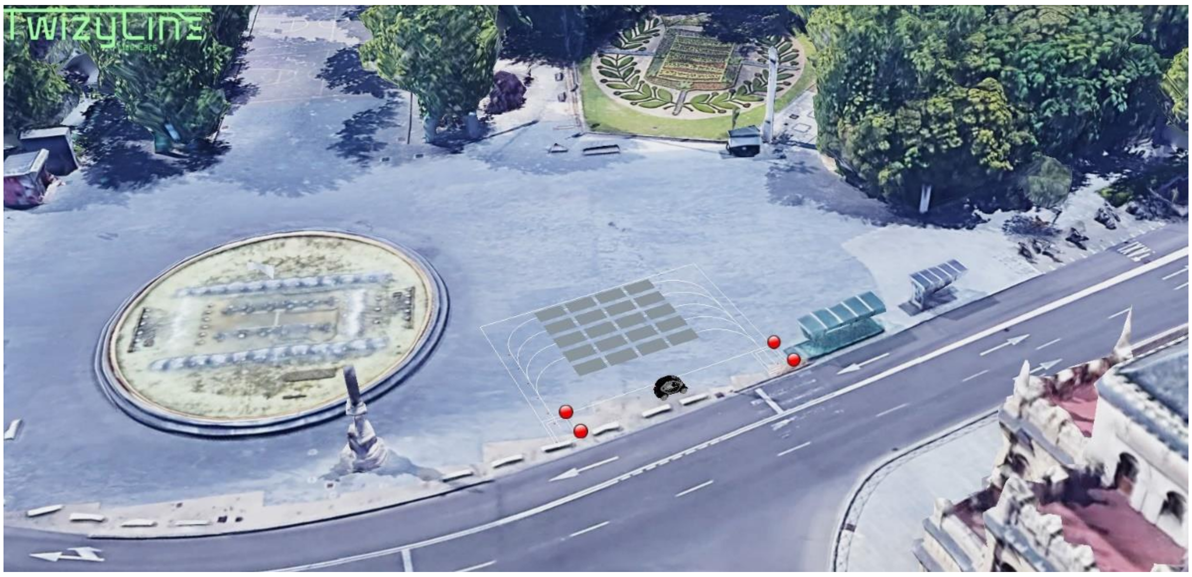
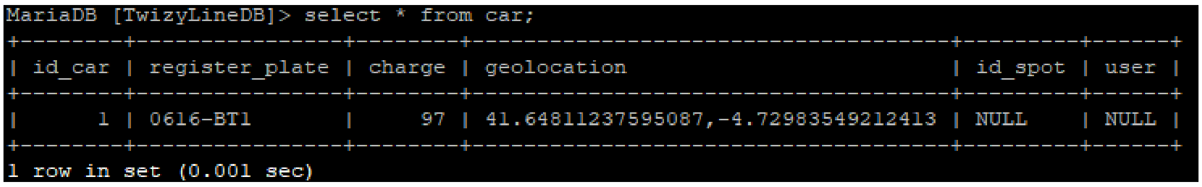


Figure 46. TwizyLine parking located in Plaza Zorrilla.

5.2 Outside stage

Any car that is outside any parking will be on the outside stage. So, the first time a Twizy is connected, it is in this state. At the initial moment, when TwizyLine starts, a Renault operator will put the vehicles in the parking lots. The operator should only leave them in the leaving area, and then the system will already be in charge of parking them as we have already explained in previous chapters. In Figure 47, we see what the first Twizy connected to the database would be like, being outside any parking.



```
MariaDB [TwizyLineDB]> select * from car;
+--------+----------------+--------+-------------------------------------+---------+------+
| id_car | register_plate | charge | geolocation                         | id_spot | user |
+--------+----------------+--------+-------------------------------------+---------+------+
|      1 | 0616-BT1       |     97 | 41.64811237595087,-4.72983549212413 | NULL    | NULL |
+--------+----------------+--------+-------------------------------------+---------+------+
1 row in set (0.001 sec)
```

Figure 47. Table 'car' from the TwizyLine database

As we see in this Figure 47, the fields 'id_spot' and 'user' are 'NULL', because it is supposed that at the initial moment they are carried by an operator as it is said before. So, once you enter a parking lot for the first time, it will be assigned an 'id_spot', that is, the spot where it is parked, or the name of the user who drives it if it is taken out. It is talked more about this in the following sections.

Returning to the outside stage and taking into account the flow diagram shown in section 3.3 (Figure 26), we will see what happens in the simulator. So, we placed in the TwizyLine parking from Plaza Zorrilla where the tests are going to be taken. As it was shown in the previous section, there are 5 vehicles in that parking. The cars have the arrangement shown in Figure 48 in that parking.

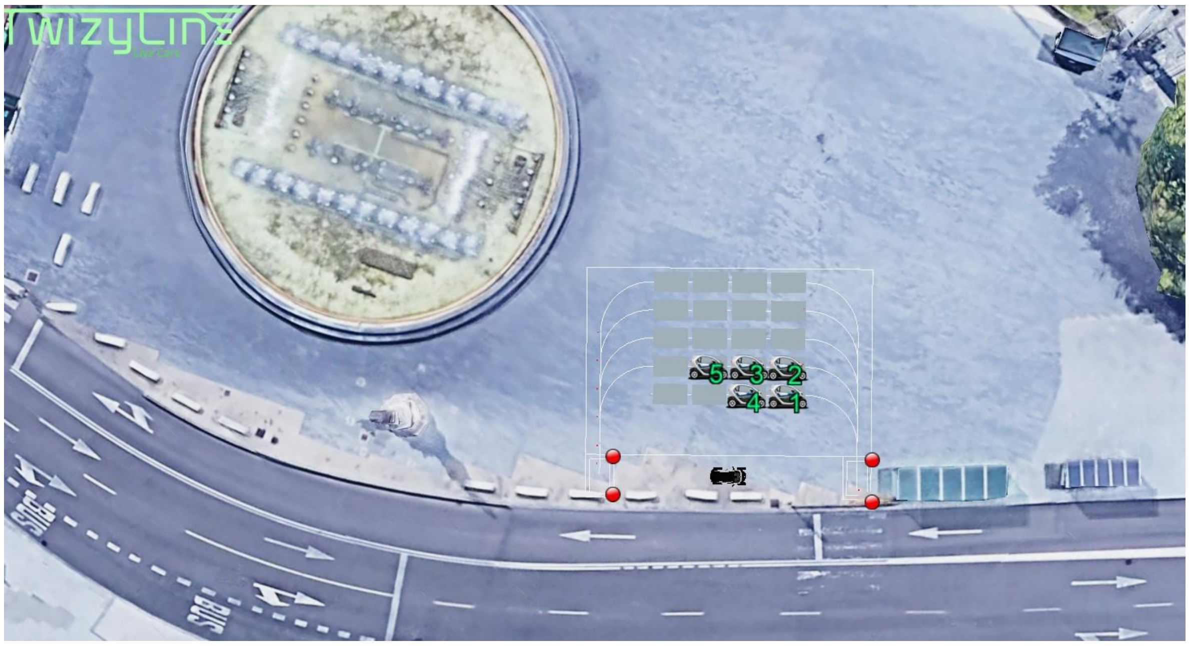


Figure 48. Current status of a TwizyLine parking.

We will assume that car number 6 wants to park in that parking. Then, once you get close enough, barrier 1 is opened (Figure 49).

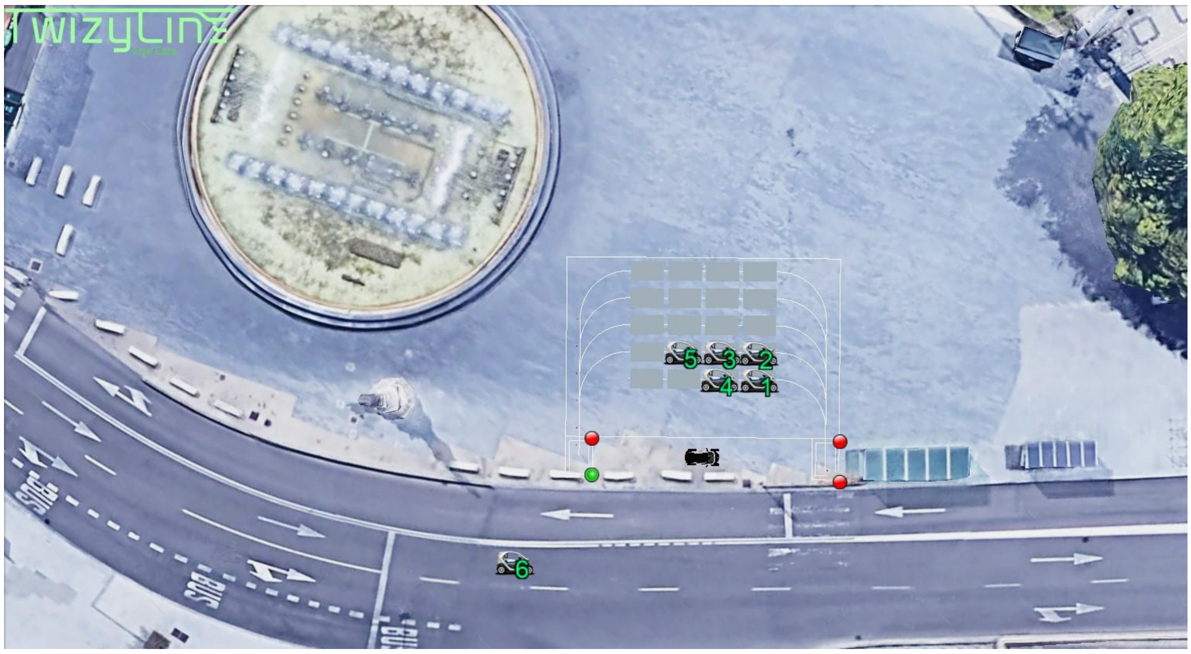


Figure 49. Barrier 1 opens (green point) because a Twizy is near.

As the barrier 1 is up now, it will allow the car to enter the leaving zone. When the vehicle reads the RFID of the leaving zone, it will send an RFID message (which was explained in section 4.4) and the system detects that there is a car in that zone and will close barrier 1 (Figure 50). From this point, the entry procedure would begin.

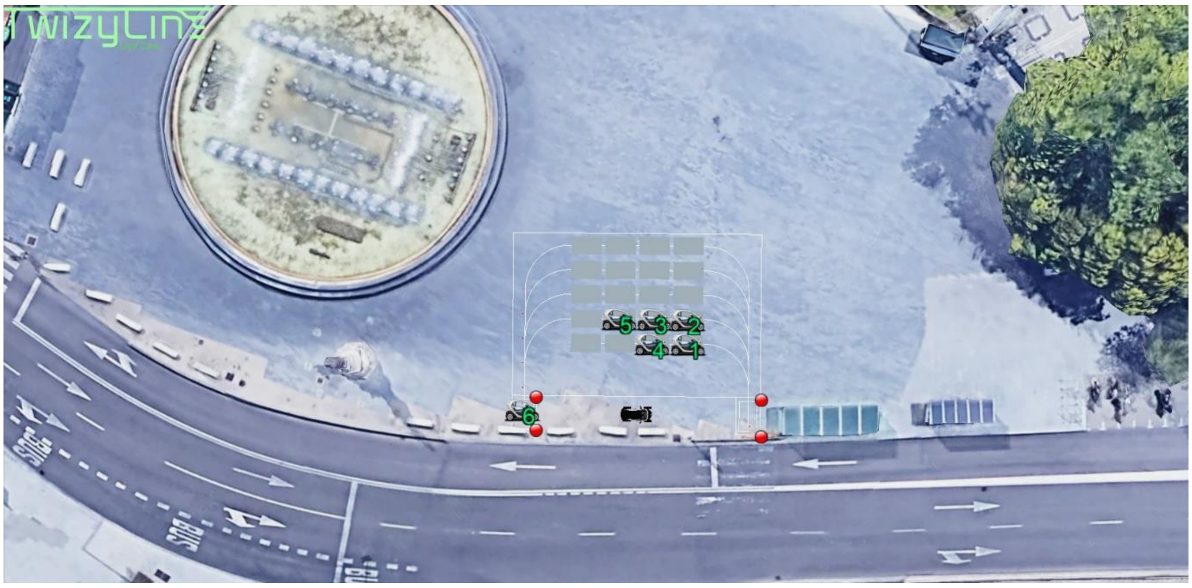


Figure 50. The Twizy number 6 is in the leaving zone and the system has already detected it.

Regarding this point, there is a small variant. If the car wants to go in that parking but there is already one in the leaving area. To simulate this case, we have used another TwizyLine parking located in Plaza Portugalete.

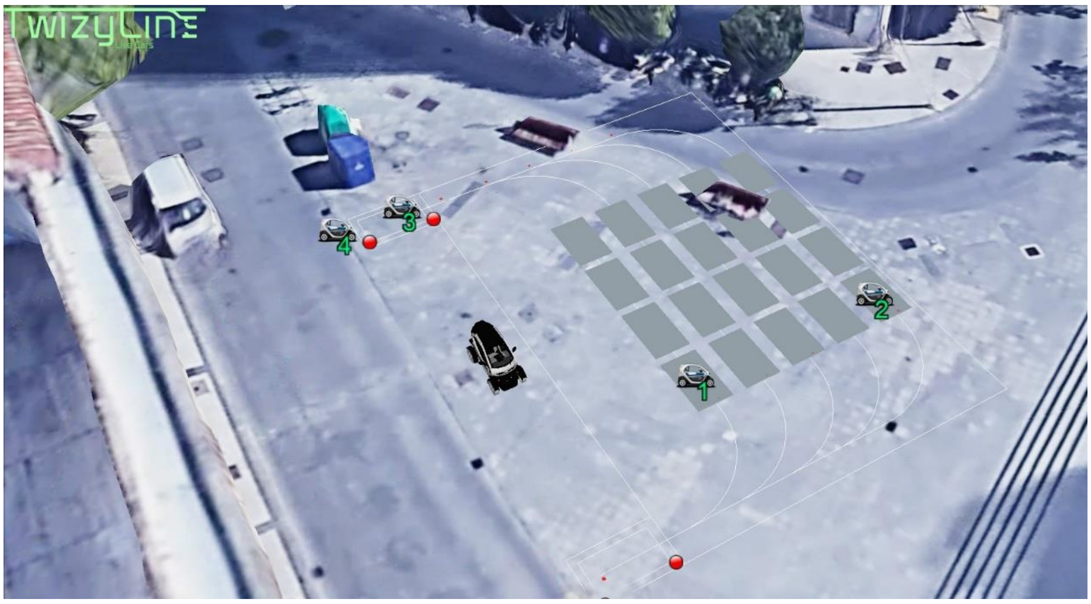


Figure 51. TwizyLine parking from Plaza Portugalete. Barrier 1 does not open because the leaving zone is occupied.

As we see in Figure 51, barrier 1 does not open because it detects that the leaving zone is already occupied. Once this car enters and barrier 2 closes, it will enable barrier 1 to open again (Figure 52).

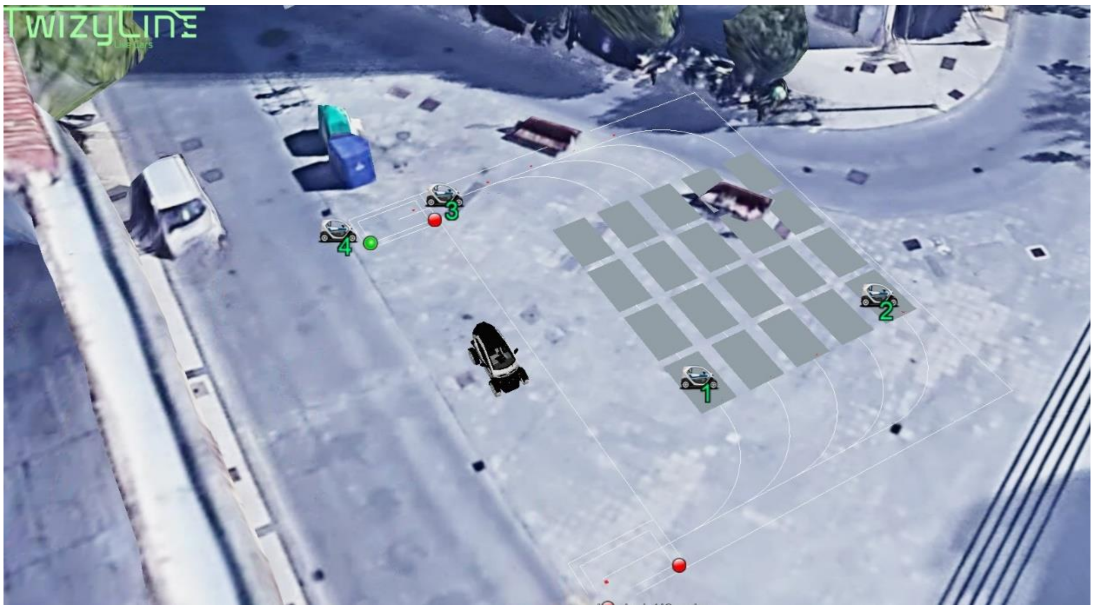


Figure 52. Barrier 1 is enabled to open now, as the leaving zone is empty and the barrier 2 is closed.

5.3 Enter procedure

Continuing the previous example of Twizy number 6 from the section 5.2, now it is started the enter procedure. Now the system is waiting for the detection system to assure that the driver has gone out the leaving zone through the turnstile. To imitate this action, it is clicked on the button 'X' of the 8Bit Controller, which is in charge of simulating the message sent by the turnstile to the server (section 4.6.1). Now the barrier 2 is opened, and the car number 6 drives autonomously to its parking spot (Figure 53).

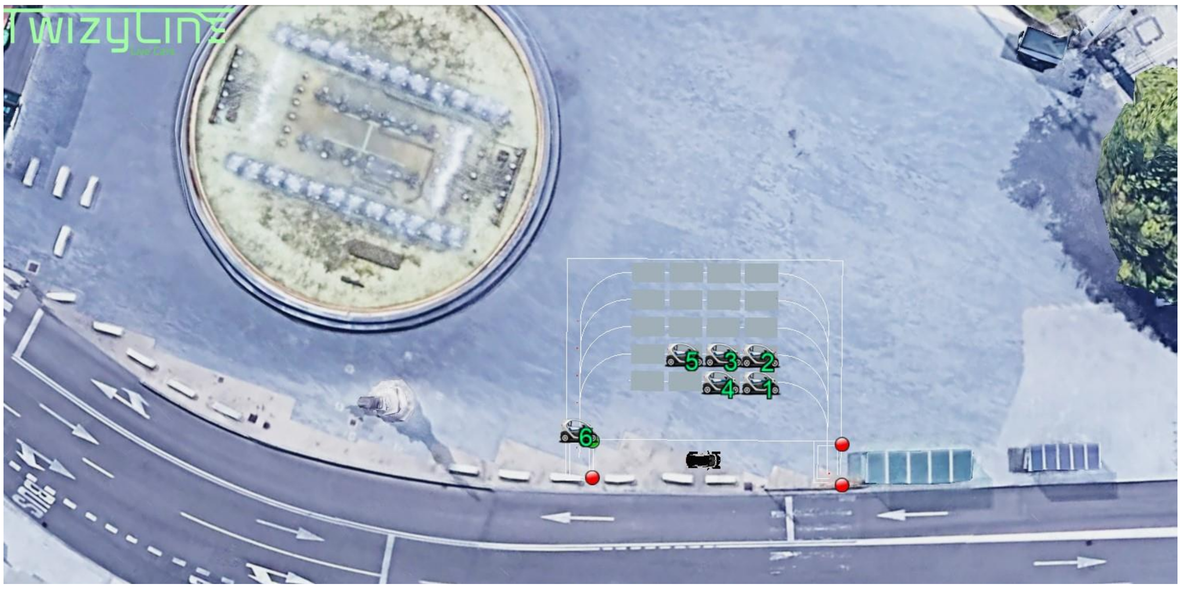


Figure 53. Barrier 2 opens, and the Twizy moves autonomously towards its spot.

Once the car has reached the turn RFID of the row of its spot, the barrier 2 closes (Figure 54). At this point, the back-end software can manage another entrance from the leaving zone.

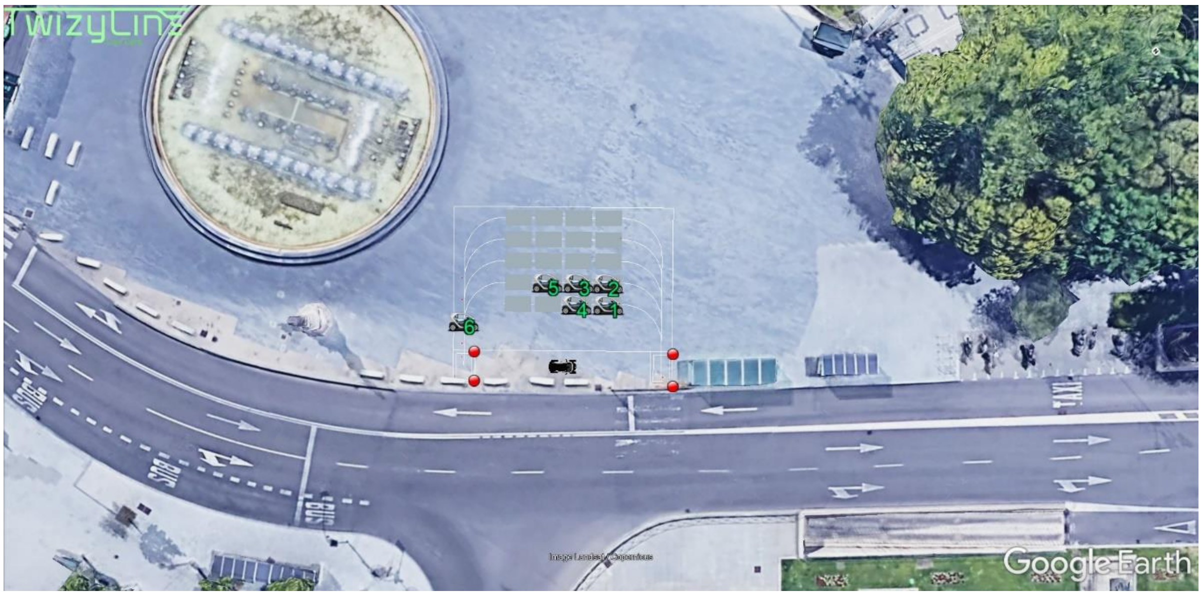
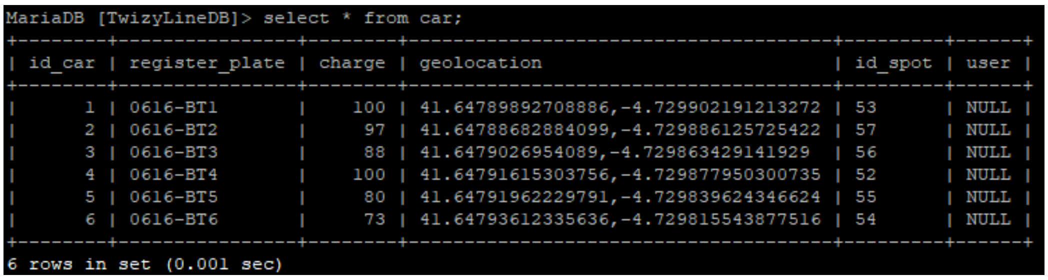


Figure 54. The barrier 2 closes as the Twizy number 6 reaches the turn RFID.

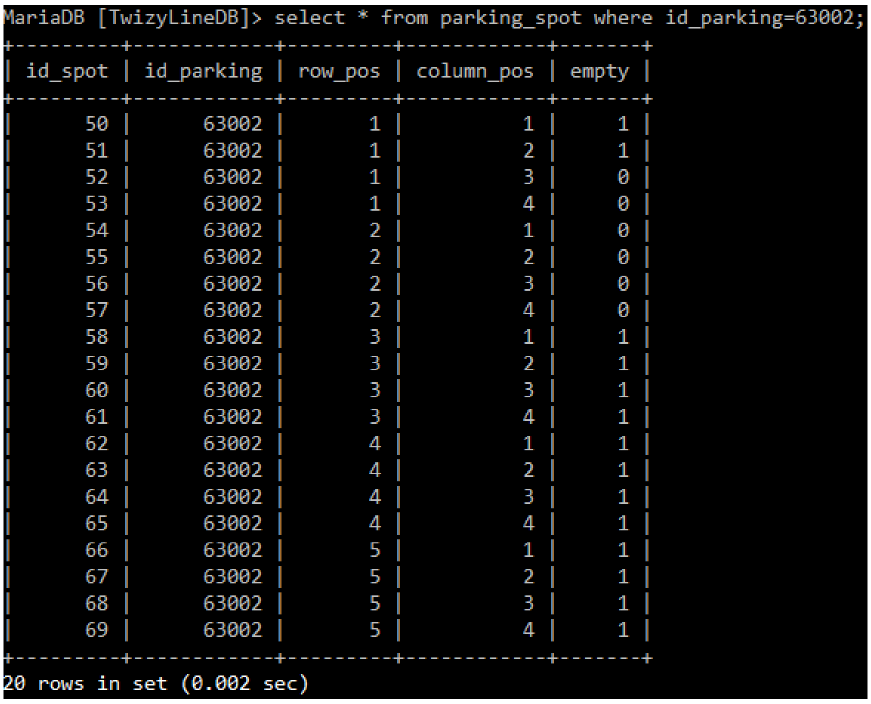
As Twizy number 6 has 73% of battery, the back-end software sent it to the second row, it is possible to see its battery in the table ‘car’ of the TwizyLine database (Figure 55).



```
MariaDB [TwizyLineDB]> select * from car;
+--------+----------------+--------+---------------------------------------+---------+------+
| id_car | register_plate | charge | geolocation                           | id_spot | user |
+--------+----------------+--------+---------------------------------------+---------+------+
|      1 | 0616-BT1       |    100 | 41.64789892708886,-4.729902191213272  | 53      | NULL |
|      2 | 0616-BT2       |     97 | 41.64788682884099,-4.729886125725422  | 57      | NULL |
|      3 | 0616-BT3       |     88 | 41.6479026954089,-4.729863429141929   | 56      | NULL |
|      4 | 0616-BT4       |    100 | 41.64791615303756,-4.729877950300735  | 52      | NULL |
|      5 | 0616-BT5       |     80 | 41.64791962229791,-4.729839624346624  | 55      | NULL |
|      6 | 0616-BT6       |     73 | 41.64793612335636,-4.729815543877516  | 54      | NULL |
+--------+----------------+--------+---------------------------------------+---------+------+
6 rows in set (0.001 sec)
```

Figure 55. Table 'car', Twizy id equals to 6 has 73% of battery.

In Figure 56 and Figure 57, it is shown how the information recorded in the database matches correctly with the status of the parking. In the Figure 56, it is the information about the spots of parking with id equals to 63002 (which is the one of Plaza Zorrilla), and it can be seen the occupied spots in the field ‘empty’. In Figure 57, it is the final result of the entrance procedure of the Twizy number 6.



```
MariaDB [TwizyLineDB]> select * from parking_spot where id_parking=63002;
+---------+------------+---------+------------+-------+
| id_spot | id_parking | row_pos | column_pos | empty |
+---------+------------+---------+------------+-------+
|      50 |      63002 |       1 |          1 |     1 |
|      51 |      63002 |       1 |          2 |     1 |
|      52 |      63002 |       1 |          3 |     0 |
|      53 |      63002 |       1 |          4 |     0 |
|      54 |      63002 |       2 |          1 |     0 |
|      55 |      63002 |       2 |          2 |     0 |
|      56 |      63002 |       2 |          3 |     0 |
|      57 |      63002 |       2 |          4 |     0 |
|      58 |      63002 |       3 |          1 |     1 |
|      59 |      63002 |       3 |          2 |     1 |
|      60 |      63002 |       3 |          3 |     1 |
|      61 |      63002 |       3 |          4 |     1 |
|      62 |      63002 |       4 |          1 |     1 |
|      63 |      63002 |       4 |          2 |     1 |
|      64 |      63002 |       4 |          3 |     1 |
|      65 |      63002 |       4 |          4 |     1 |
|      66 |      63002 |       5 |          1 |     1 |
|      67 |      63002 |       5 |          2 |     1 |
|      68 |      63002 |       5 |          3 |     1 |
|      69 |      63002 |       5 |          4 |     1 |
+---------+------------+---------+------------+-------+
20 rows in set (0.002 sec)
```

Figure 56. Table 'parking_spot', showing just the information of those registers with id equals to 63002.

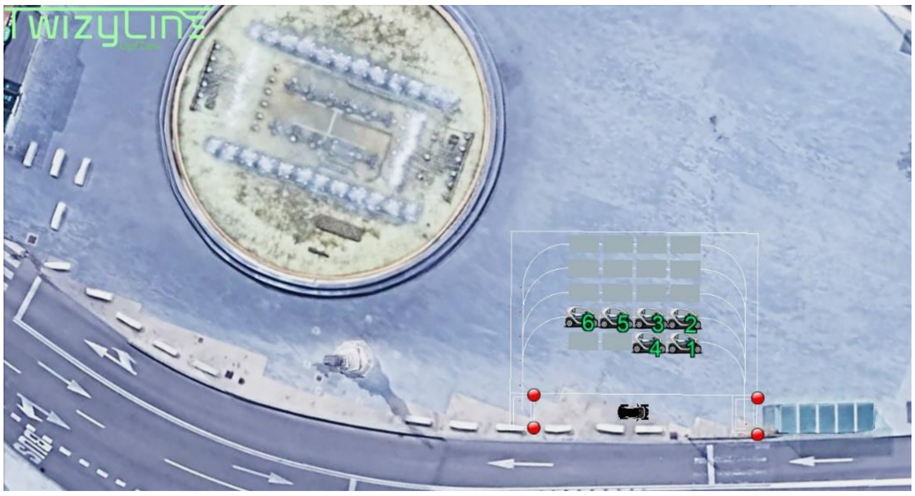
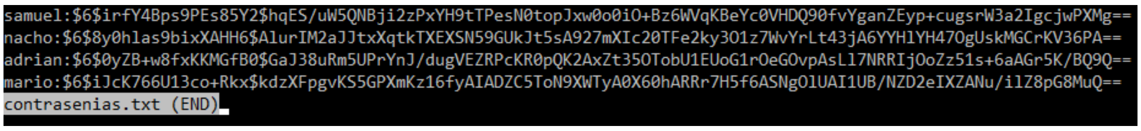


Figure 57. Status of TwizyLine parking of Plaza Zorrilla.

5.4 Standby Stage

Standby stage is the state in which the vehicle is inside the parking in standby mode. In this stage, the Twizy just moves if the exit procedure starts. The most important point about this stage is that the communication between the application and the server is correct, and the data shown in the application matches with the information stores in the database.

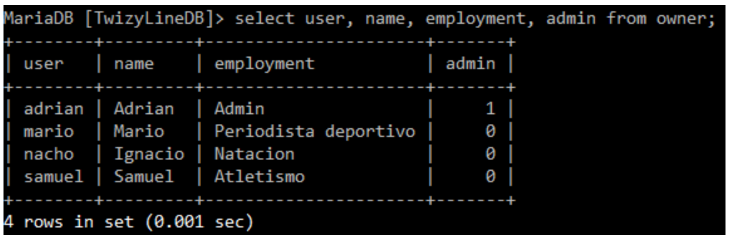
First, the user must log in to the application. If it is already registered, the user and password of the client who wants to access to the system is compared with the file ‘contrasenias.txt’. Inside this file, it is the user and the password cyphered (Figure 58). It is stored in the server.



```
samuel:$6$irfY4Bps9PEs85Y2$hqES/uW5QNBji2zPxYH9tTPesN0topJxw0o0iO+Bz6WVqKBeYc0VHDQ90fvYganZEyp+cugsrW3a2IgcjwPXMg==
nacho:$6$8y0hlas9bixXAHH6$AlurIM2aJJtxXqtkTXEXSN59GUkJt5sA927mXIc20TFe2ky3O1z7WvYrLt43jA6YYH1YH47OgUskMGCrKV36PA==
adrian:$6$0yZB+w8fxKKMGfB0$GaJ38uRm5UPrYnJ/dugVEZRPcKR0pQK2AxZt35OTobU1EUoG1rOeGOvpAsL17NRRIjOoZz51s+6aAGr5K/BQ9Q==
mario:$6$iJcK766U13co+Rkx$kdzXFpgvKS5GPXmKz16fyAIADZC5ToN9XWTyA0X60hARRr7H5f6ASNgO1UAI1UB/NZD2eIXZANu/ilZ8pG8MuQ==
contrasenias.txt (END)
```

Figure 58. File 'contrasenias.txt' where it is the username and the password cypher of the TwizyLine clients.

If the client has no account, it is checked if the username matches with any of the previous usernames used, which are stored in the database (Figure 59). If it is not equal as any other it is accepted that user, if not it is refused.



```
MariaDB [TwizyLineDB]> select user, name, employment, admin from owner;
+--------+---------+---------------------+-------+
| user   | name    | employment          | admin |
+--------+---------+---------------------+-------+
| adrian | Adrian  | Admin               |     1 |
| mario  | Mario   | Periodista deportivo |     0 |
| nacho  | Ignacio | Natacion            |     0 |
| samuel | Samuel  | Atletismo           |     0 |
+--------+---------+---------------------+-------+
4 rows in set (0.001 sec)
```

Figure 59. Table 'owner' of the TwizyLine database.

To continue testing the system, it is logged with the user ‘samuel’. So, in Figure 60 it is seen in the interface of the application once the user has chosen the parking from Plaza Zorrilla, that in this parking there are 6 cars available and 14 spots free. And in Figure 61, there is part of the table ‘parking’ of the TwizyLine database, showing that in the parking of Plaza Zorrilla there are 14 free spots.

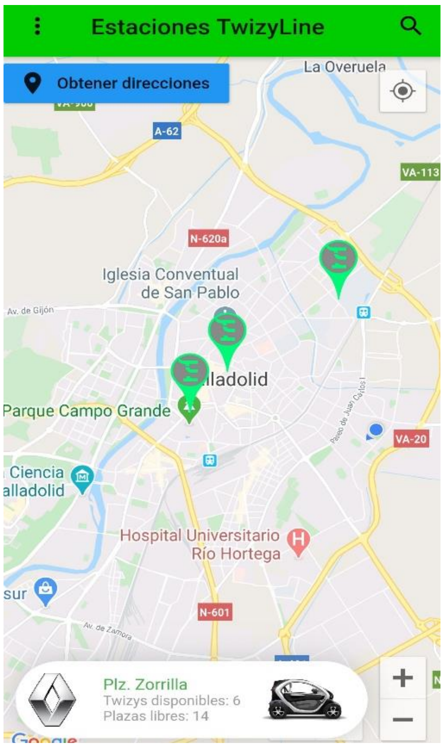
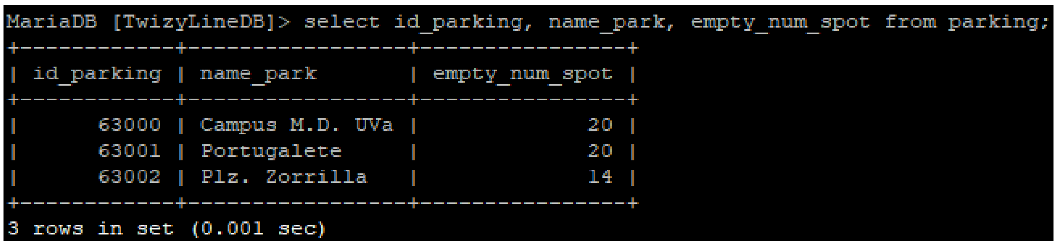


Figure 60. TwizyLine application, pressing the parking of Plz. Zorrilla.



```
MariaDB [TwizyLineDB]> select id_parking, name_park, empty_num_spot from parking;
+------------+-----------------+----------------+
| id_parking | name_park       | empty_num_spot |
+------------+-----------------+----------------+
|      63000 | Campus M.D. UVa |             20 |
|      63001 | Portugalete     |             20 |
|      63002 | Plz. Zorrilla   |             14 |
+------------+-----------------+----------------+
3 rows in set (0.001 sec)
```

Figure 61. 3 fields of table 'parking' of the TwizyLine database.

Then the user presses the button at the bottom where it is the information of TwizyLine parking, and after the application receives the Twizy selected to that client. The chosen Twizy starts the exit procedure which is explained in the next section. This Twizy selected for the client is the number 2 (Figure 57), because according to the back-end algorithm if all the Twizys in the last column have the same battery 100%, the Twizy parked in the farthest row from the pick-up zone is chosen. Regarding these last sentence 2 relevant notes:

- As it has been explained, this selection way is due to the charging system proposed in the parking explained in section 4.5.5.
- Although in Figure 55 vehicles 2 and 1 has different battery levels, meanwhile testing has been waiting until both charge to the maximum battery level.

5.5 Exit procedure

Continuing with the example of the previous section, now the Twizy number 2 of the parking of Plaza Zorrilla is in autonomous mode and following the orders sent by the back-end software the Twizy drives towards the leaving zone. Before this happens, the barrier 3 opens (Figure 62).

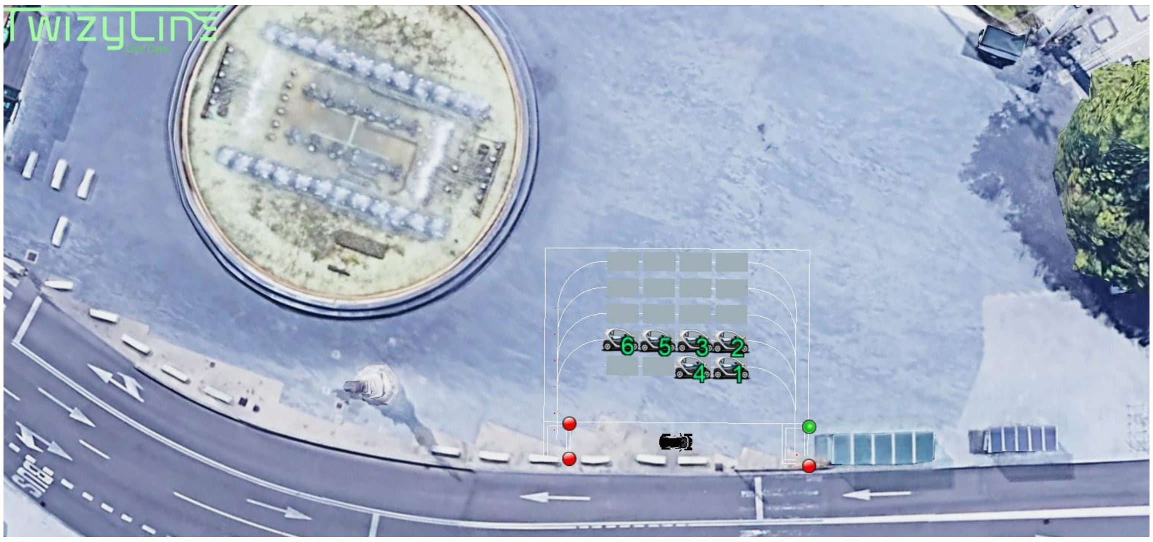


Figure 62. Barrier 3 opens (green point) because the Twizy number 2 must drive to the pick-up zone.

Once the car number 2 is over the RFID of the pick-up zone, barrier 3 closes, the turnstile is enabled and the vehicles parked in the same row as the number 2, Twizys number 6, 5 and 3 are going to move forward one spot (Figure 63).

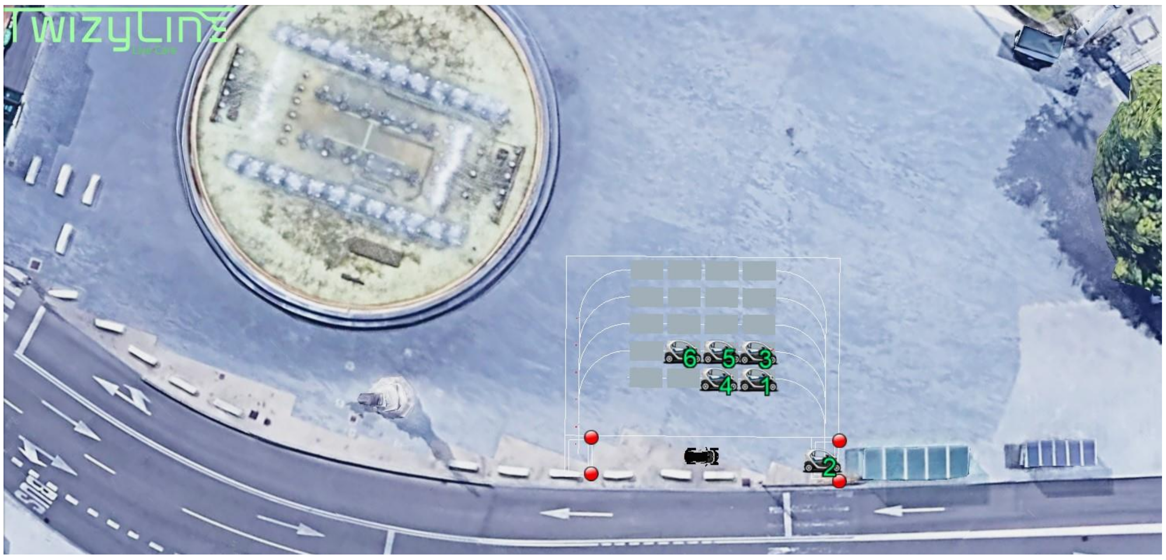


Figure 63. Twizy number 2 assigned to one client and cars number 6, 5 and 3 has been moved forward one spot.

Now the system is waiting until the client passes through the turnstile to take its Twizy. This message of the turnstile is simulated pressing the button ‘A’ of the 8Bit Controller. Once it is pressed, the barrier 4 opens and the client drive its Twizy (Figure 64). Barrier 4 is closed with the car already outside the pick-up zone.

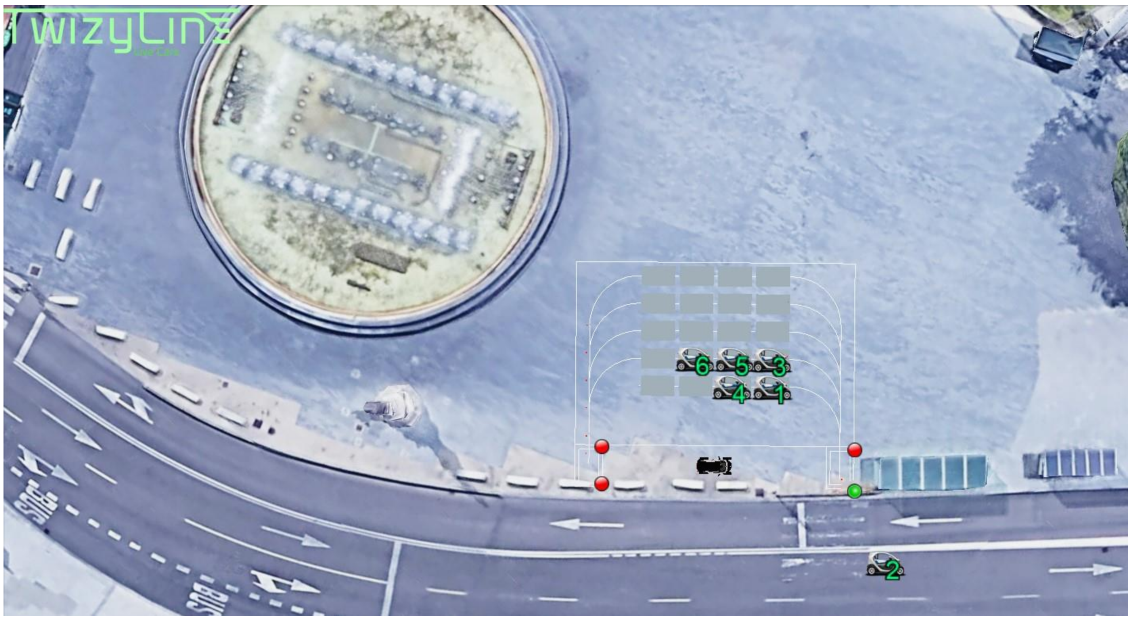
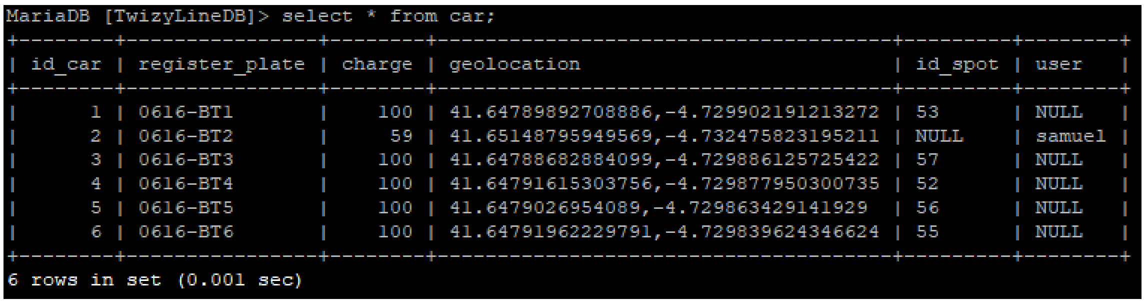


Figure 64. Twizy number 2 leaves the parking, now it is being used by a client.

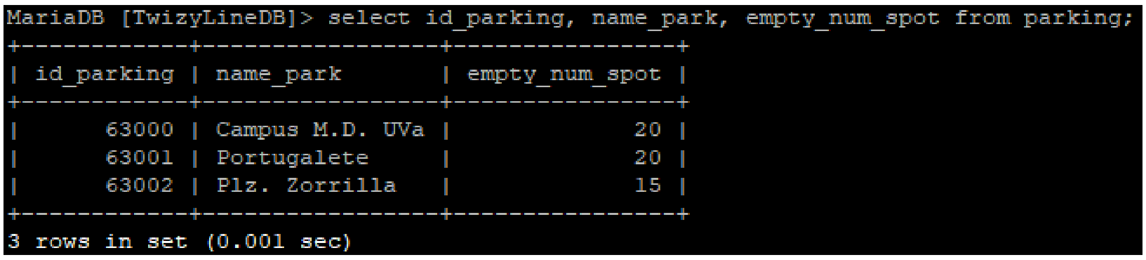
Taking a look to the TwizyLine database, it can be seen that the car number 2 is used by user ‘samuel’, because in the previous section it has been said that it has been logged as ‘samuel’ (Figure 65).



```
MariaDB [TwizyLineDB]> select * from car;
+--------+----------------+--------+---------------------------------------+---------+--------+
| id_car | register_plate | charge | geolocation                           | id_spot | user   |
+--------+----------------+--------+---------------------------------------+---------+--------+
|      1 | 0616-BT1       |    100 | 41.64789892708886,-4.729902191213272 | 53      | NULL   |
|      2 | 0616-BT2       |     59 | 41.65148795949569,-4.732475823195211 | NULL    | samuel |
|      3 | 0616-BT3       |    100 | 41.64788682884099,-4.729886125725422 | 57      | NULL   |
|      4 | 0616-BT4       |    100 | 41.64791615303756,-4.729877950300735 | 52      | NULL   |
|      5 | 0616-BT5       |    100 | 41.6479026954089,-4.729863429141929  | 56      | NULL   |
|      6 | 0616-BT6       |    100 | 41.64791962229791,-4.729839624346624 | 55      | NULL   |
+--------+----------------+--------+---------------------------------------+---------+--------+
6 rows in set (0.001 sec)
```

Figure 65. Table 'car' from the database once the Twizy is being driven by user 'samuel'.

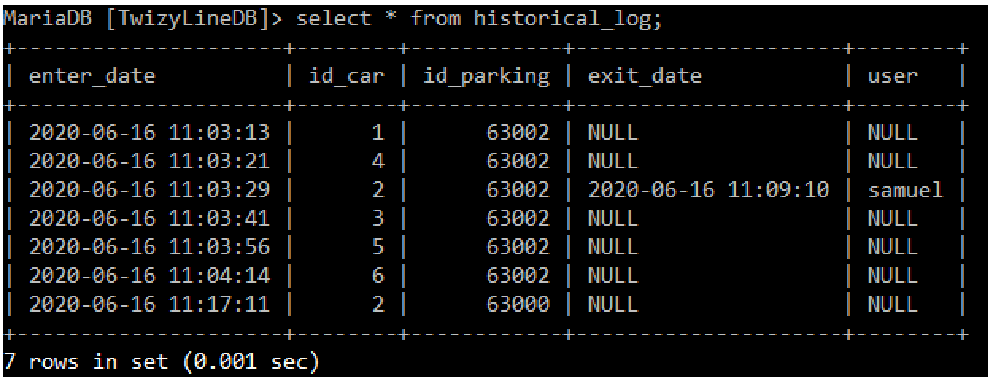
In the table ‘parking’ of the database, it is possible to observe that there are now 15 free spots and not 14 as previously (Figure 66).



```
MariaDB [TwizyLineDB]> select id_parking, name_park, empty_num_spot from parking;
+------------+-----------------+----------------+
| id_parking | name_park       | empty_num_spot |
+------------+-----------------+----------------+
|      63000 | Campus M.D. UVa |             20 |
|      63001 | Portugalete     |             20 |
|      63002 | Plz. Zorrilla   |             15 |
+------------+-----------------+----------------+
3 rows in set (0.001 sec)
```

Figure 66. Table 'parking' of the TwizyLine database.

Finally, it is relevant to highlight the table ‘historical_log’, which works as a fleet management for the administrators, looking at that table in Figure 67, it is possible to know the enter and exit time of the Twizy number 2 from the parking 63002 (Plaza Zorrilla), and it has been driven by user ‘samuel’ until it has been left in parking 6300 (Campus Miguel Delibes).

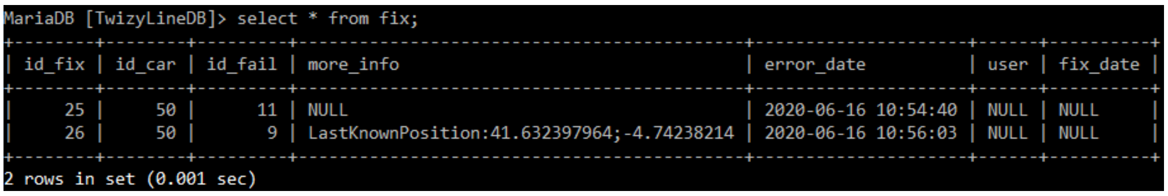


```
MariaDB [TwizyLineDB]> select * from historical_log;
+---------------------+--------+------------+---------------------+--------+
| enter_date          | id_car | id_parking | exit_date           | user   |
+---------------------+--------+------------+---------------------+--------+
| 2020-06-16 11:03:13 |      1 |      63002 | NULL                | NULL   |
| 2020-06-16 11:03:21 |      4 |      63002 | NULL                | NULL   |
| 2020-06-16 11:03:29 |      2 |      63002 | 2020-06-16 11:09:10 | samuel |
| 2020-06-16 11:03:41 |      3 |      63002 | NULL                | NULL   |
| 2020-06-16 11:03:56 |      5 |      63002 | NULL                | NULL   |
| 2020-06-16 11:04:14 |      6 |      63002 | NULL                | NULL   |
| 2020-06-16 11:17:11 |      2 |      63000 | NULL                | NULL   |
+---------------------+--------+------------+---------------------+--------+
7 rows in set (0.001 sec)
```

Figure 67. Table 'historical_log' of the TwizyLine database.

5.6 Failure management

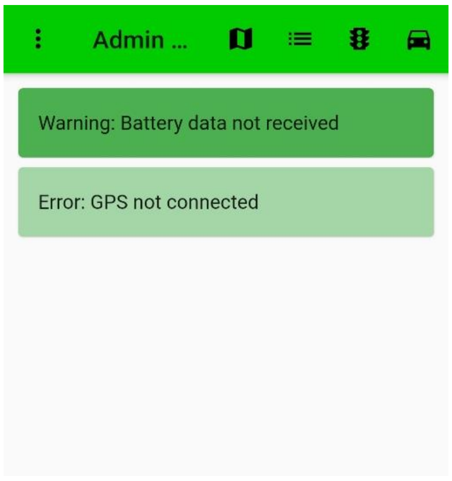
The last function it has been tested is the failure management. In this part, the Twizy number 50 has reported 2 failures. This Twizy number 50 has the particularity that it is the one which implements the humming boards developed by Ignacio Royuela in its Final Degree Project [14]. So, in order to check that the failures are reported and then saved in the database correctly, we look into the database (Figure 68).



```
MariaDB [TwizyLineDB]> select * from fix;
+--------+--------+---------+---------------------------------------------+---------------------+------+----------+
| id_fix | id_car | id_fail | more_info                                   | error_date          | user | fix_date |
+--------+--------+---------+---------------------------------------------+---------------------+------+----------+
|     25 |     50 |      11 | NULL                                        | 2020-06-16 10:54:40 | NULL | NULL     |
|     26 |     50 |       9 | LastKnownPosition:41.632397964;-4.74238214 | 2020-06-16 10:56:03 | NULL | NULL     |
+--------+--------+---------+---------------------------------------------+---------------------+------+----------+
2 rows in set (0.001 sec)
```

Figure 68. Table 'fix' from TwizyLine database.

In table ‘fix’, it can be seen the car which reports the failure, the kind of failure and when it has occurred. This information is sent to the administrator to warn them about these failures (Figure 69). If it is logged with user ‘adrian’ to the application, we logged as administrator (check Figure 59, ‘adrian’ is the only administrator available).

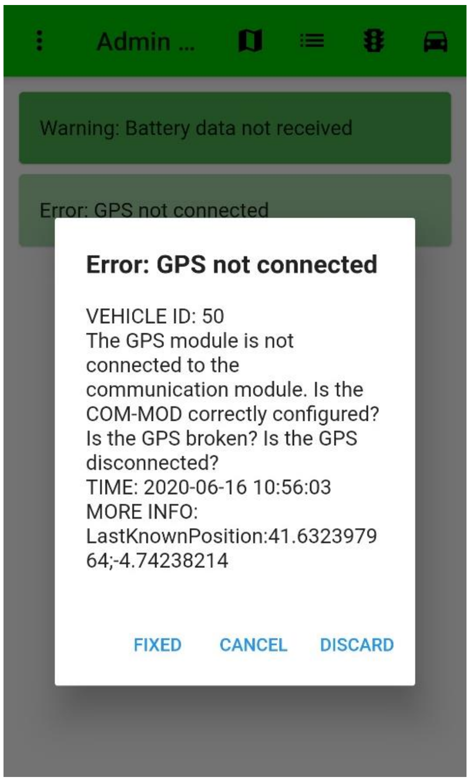


Warning: Battery data not received

Error: GPS not connected

Figure 69. Administrator application interface of error management.

Administrators can select what to do with these failures (Figure 70). For more information about the administrator account of the application check Adrian Mazaira Hernandez’s Final Degree Project [15].



Warning: Battery data not received

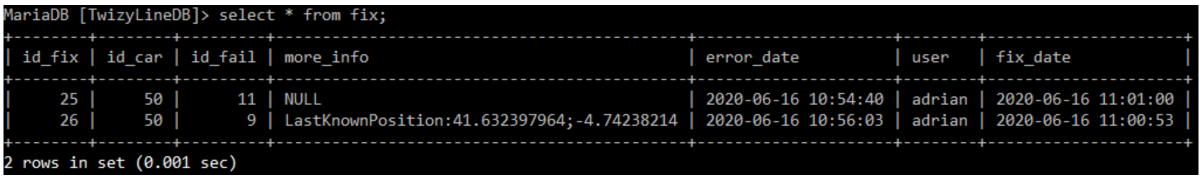
Error: GPS not connected

Error: GPS not connected

VEHICLE ID: 50
The GPS module is not connected to the communication module. Is the COM-MOD correctly configured? Is the GPS broken? Is the GPS disconnected?
TIME: 2020-06-16 10:56:03
MORE INFO:
LastKnownPosition:41.632397964;-4.74238214

Figure 70. An error 'GPS not connected' of Twizy 50 has been received by an administrator.

If the failure is an error, it must be repaired. Once it is repaired, the administrator pressed to ‘FIXED’ and then a ‘RESTART’ message is sent to the Twizy and the database is updated to know who has fixed that error and when it has been made (Figure 71).



```
MariaDB [TwizyLineDB]> select * from fix;
+--------+--------+---------+------------------------------------------+---------------------+--------+---------------------+
| id_fix | id_car | id_fail | more_info                                | error_date          | user   | fix_date            |
+--------+--------+---------+------------------------------------------+---------------------+--------+---------------------+
|     25 |     50 |      11 | NULL                                     | 2020-06-16 10:54:40 | adrian | 2020-06-16 11:01:00 |
|     26 |     50 |       9 | LastKnownPosition:41.632397964;-4.74238214 | 2020-06-16 10:56:03 | adrian | 2020-06-16 11:00:53 |
+--------+--------+---------+------------------------------------------+---------------------+--------+---------------------+
2 rows in set (0.001 sec)
```

Figure 71. Table 'fix' of the TwizyLine database updated with the new information.

6 Conclusions and future lines

6.1 Conclusions

Considering the objectives defined in section 1.2 of this Final Degree Project, it can be concluded that it has been implemented a software able to manage most of the TwizyLine system functionalities. Thanks to this development, it has been approached those intelligent infrastructures which form the smart city of the future, a field that as it has been seen in the state of the art is booming.

First of all, the development of the entire TwizyLine project has served to learn what it means to manage a project of such magnitude. Regarding work, it has been important the teamwork, to contribute ideas, to reach agreements, and carry out the project in general.

About this Final Degree Project, before the beginning of the back-end software implementation, an intensive research and after study have been made to choose properly the technologies that are going to be employed. MQTT, python and MariaDB were the best candidates, and thus the three of them were installed in the raspberry which works as a server for our system.

In the coding phase, it is paid special attention to four main points:

- Controlling which devices can connect to the TwizyLine system.
- Multiple simultaneous processing, the back-end software must be able to manage different operations of the system at the same time.
- Access and management of the database, the software coded have to be able to make queries to the database which is a warehouse of data from the TwizyLine system.
- Communicate with external elements connected to the same system, either in real-time or after a fact.

Once the back-end software was completed, it was the time of the testing phase. In this phase, this Final Degree Project along with the one from Adrian Mazaira Hernández and Ignacio Royuela González were put into operation to check the performance of the whole system in a simulator, which aims to imitate the functioning of TwizyLine in real life.

It is necessary to emphasize that thanks to the realization of this Final Degree Project, new knowledge has been acquired and other topics studied in the degree have been studied in depth. This project also promotes the introduction of technology in cities as a mechanism to help society be greener by encouraging the use of electric cars. Besides, it is remembered that this TFG is part of the TwizyLine project that will be submitted to the TwizyContest, and its final aim is to be implemented in real-life in Paris during the Olympic Games of 2024.

6.2 Future lines

TwizyLine is an innovative project, which can be still improved and developed many more functionalities. About the back-end software presented in this Final Degree Project, it tries to cover most of the functionalities of the ideal parking, however, some are left. About the parking, it could be interesting to implement some of the devices that are used to control the entrance and exit of the vehicles.

New lines of development:

- Research about people detection cameras, to ensure that nobody is on the leaving zone, before it is in autonomous mode.
- Implement a detection sensor to detect the vehicles going out of the pick-up zone.
- Detection of errors from the parking, in this way, errors such as the barrier or the turnstile do not work properly can be notified to the administrator.
- Research about inductive charge systems for electric vehicles.
- Research about current ways to provide more redundancy to our server and try to implement them.
- Provide more sensors to the inner circuit to have more control over the position of the Twizy, such as an RFID in each spot.
- Implement the security layer of TLS/SSL to provide to the communications more confidentiality.
- Analyze the behaviour of the network traffic depending on the QoS level used in the exchange of messages.
- Improve the TwizyLine failure management system.

6.3 Awards received and other merits

This Final Degree Project is part of the TwizyLine project. The project has been awarded at national level and officially chosen to represent Spain in the international contest TwizyContest (Figure 72) in December 2020.

2020 TWIZY'CONTEST
Off-the-Street

Figure 72. Logotype of the international contest TwizyContest [11].

Furthermore, TwizyContest have been awarded with the ‘Premios Prometeo’ (Figure 73), organized by the FunGe UVa (General Foundation of the University of Valladolid). ‘Premios Prometeo’ program supports projects from students of the University of Valladolid with the aim to get a market-oriented prototype. In addition, this award helps us to file a patent application of our project.

prometeo

Figure 73. Logotype of the awards ‘Premios Prometeo’.

7 Bibliography

[1] World Health Organization (WHO), “How air pollution is destroying our health“, 2018. Available online: https://www.who.int/en/air-pollution/news-and-events/how-air-pollution-is-destroying-our-health

[2] World Health Organization (WHO), “Ambient air pollution: Pollutants”, 2017. Available online: https://www.who.int/airpollution/ambient/pollutants/en/

[3] Maureen L. Cropper et al, “Global estimates of mortality associated with long-term exposure to outdoor fine particulate matter”, 2018. Available online: https://www.pnas.org/content/115/38/9592

[4] World Health Organization (WHO), “Air pollution – The silent killer”, 2017. Available online: https://www.who.int/airpollution/infographics/en/

[5] World Wildlife Fund (WWF), “París: Un acuerdo histórico que necesita concentarse en acción”. Available online: https://www.wwf.es/nuestro_trabajo/clima_y_energia/cumbres_del_clima/

[6] NASA’s Goddart institute for Space Studies (GISS), “Global land-ocean temperature index”, 2020. Available online: https://climate.nasa.gov/

[7] Statista, “Los automóviles eléctricos, todavía en minoría en Europa”, 2019. Available online: https://es.statista.com/grafico/16084/proporcion-de-coches-electricos-sobre-matriculaciones-totales/

[8] Jonathan Guantt, “Global Emissions Standards Driving Hybrid and Electric Vehicle Growth”, 2019. Available online: https://www.aegonassetmanagement.com/us/thought-leadership/blog/credit-research/global-emissions-standards-driving-hybrid-and-electric-vehicle-growth/

[9] Ministerio de transición ecológica y reto de demográfico, “Plan MOVALT vehículos: Una manera de hacer Europa”, 2018. Available online: https://www.idae.es/ayudas-y-financiacion/para-movilidad-y-vehiculos/plan-movalt-vehiculos

[10] Transport & Environment, “Low Emission Zones are a success-but they must now move to zero-emission mobility”, 2019. Available online: https://www.transportenvironment.org/sites/te/files/publications/2019_09_Briefing_LEZ-ZEZ_final.pdf

[11] Twizy Contest 2020: Off-the-Street, 2019. Available online: https://www.twizycontest.com/

[12] Twizy Contest Coordination Team, Groupe Renault, “TwizyContest2020: Rules and conditions”, 2019.

[13] Adrian Mazaira Hernández, Samuel Pilar Arnanz and Ignacio Royuela González, “TwizyLine: Technologies”, version 1 2020.

[14] Ignacio Royuela González, “Four level autonomous vehicle for an automatized parking”, 2020.

[15] Adrian Mazaira Hernández, “Front-end implementation for an automatized car parking”, 2020.

[16] Mario Martín Fernández, “Plan de Comunicación de TwizyLine: plataforma de carsharing con aparcamiento autónomo”, 2020.

[17] Department of Economic and Social Affairs, United Nations, “68% of the world population projected to live in urban areas by 2050, says UN”, 2018. Available online: https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html

[18] Sriharsha Mallapuram et al., “Smart City: The State of the Art, Datasets, and Evalution Platforms”, 2017. Available online:

https://www.researchgate.net/publication/319875757_Smart_City_The_State_of_the_Art_Datasets_and_Evaluation_Platforms

[19] Fundación Telefónica, "Smart cities: Un primer paso hacia la Internet de las cosas", 2011. Available online: https://www.fundaciontelefonica.com/cultura-digital/publicaciones/101/

[20] Smart City Expo World Congress, "Editions abroad in 2020", 2020. Available online: http://www.smartcityexpo.com/en/the-event/editions-abroad

[21] Urban Hub, "Inteligencia: identificación de la ciudad más 'inteligente' ", 2016. Available online: https://urban-hub.com/es/sustainability/identificacion-de-la-ciudad-mas-inteligente/

[22] MIMEISA Asset Management, "Impulso creative y generación de talent; el sello Barcelona", 2020. Available online: https://mimeisa.com/blog/impulso-creativo-y-generacion-de-talento-el-sello-barcelona/

[23] Francesca Bria, former CTO of Barcelona interview at the Barcelona Smart City World Congress, 2017. Available online: https://www.youtube.com/watch?v=VAZrlVlUPwE&feature=emb_title

[24] Platform Barcelona Digital City, Ajuntament de Barcelona, 2020. Available online: https://ajuntament.barcelona.cat/digital/en

[25] GEOCyL Consultoría Ambiental y Territorial, 2011. Available online: https://www.geocyl.com/

[26] Mi ciudad inteligente, "Mi ciudad inteligente 2017", 2017. Available online: https://www.miciudadinteligente.info/mi-ciudad-inteligente/

[27] Kreyon, "IoT Applications for Smart City", 2016. Available online: https://www.kreyonsystems.com/Blog/iot-applications-for-smart-cities/

[28] Red Eléctrica de España, "Contadores inteligentes", 2013. Available online: https://www.ree.es/es/red21/redes-inteligentes/contadores-inteligentes

[29] United States Environmental Protection Agency (EPA), "Summary of the Energy Independence and Security Act", 2007. Available online: https://www.epa.gov/laws-regulations/summary-energy-independence-and-security-act

[30] Wikipedia, Smart Grid "Staying_big_or_getting_smaller.jpg", 2018. Available online: https://en.wikipedia.org/wiki/Smart_grid#/media/File:Staying_big_or_getting_smaller.jpg

[31] Innovadores by Inndux, "Covid-19: Este Sistema IoT ayuda a la política a detector movimientos de personas y vehículos.", 2020. Available online: https://innovadores.larazon.es/es/covid-19-este-sistema-iot-ayuda-a-la-policia-a-detectar-movimientos-de-personas-y-vehiculos/

[32] Libelium, Smartphone Detection, 2020. Available online:
http://www.libelium.com/products/meshlium/smartphone-detection/

[33] Amir Sinaeepourfard, Jordi García et al., "Estimating Smart City Sensors Data Generation: Current and Future Data in the City of Barcelona", 2016. Available online: https://www.researchgate.net/figure/Barcelona-Smart-City-IT-architecture_fig1_304580274

[34] Adam Calihman, NetBurner, "Architectures in the IoT Civilization", 2019 .Available online: https://www.netburner.com/learn/architectural-frameworks-in-the-iot-civilization/

[35] Oxford Learner's Dictionaries. Available online: https://www.oxfordlearnersdictionaries.com/definition/english/back-end_1

[36] Fira Barcelona "firanews", "Smart bins at the IoTSWC", 2018. Available online: http://firanews.firabarcelona.com/en/another-look/smart-bins-at-the-iotswc/

[37] GOOD Magazine, "GOOD LOOK: Barcelona Street Lamps", 2010. Available online: https://www.youtube.com/watch?time_continue=71&v=52YKJ31pde0&feature=emb_logo

[38] Tomasz Tarczynski, "IoT backend architecture", 2018. Available online: https://es.slideshare.net/TomaszTarczyski1/iot-backend-architecture

[39] Eclipse Paho, "Eclipse Paho Downloads", 2016. Available online: https://www.eclipse.org/paho/downloads.php

[40] Ruby, “Acerca de Ruby”, 2006. Available online: https://www.ruby-lang.org/es/about/

[41] Wikipedia, “Go (lenguaje de programación)”, 2009. Available online: https://golang.org/doc/

[42] Python.org, the official home of the Python Programming Language. Available online: https://www.python.org/

[43] IEEE Spectrum, “The 2018 Top Programming Languages”, 2018. Available online: https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages

[44] IEEE Spectrum, “Top Programming Languages 2019”, 2019. Available online: https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019

[45] Michael Yuan, IBM, “Conociendo MQTT”, 2017. Available online: https://www.ibm.com/developerworks/ssa/library/iot-mqtt-why-good-for-iot/index.html

[46] Chris Pietschmann, Build5Nines, “Top 5 IoT Messaging Protocols”, 2020. Available online: https://build5nines.com/top-iot-messaging-protocols/

[47] Pick Data, “MQTT vs CoAP, la batalla por ser el major protocol IoT”, 2019. Available online: https://www.pickdata.net/es/noticias/mqtt-vs-coap-mejor-protocolo-iot

[48] Sanjay Aiyagari et al, “AMQP Advanced Message Queuing Protocol: Protocol Specification”, 2008. Available online: https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

[49] Groupe Renault, “Renault Twizy: Plug into the positive energy”, 2018. Available online: https://renaultfuturcar.com/wp-content/uploads/2018/01/Twizy.pdf

[50] Ayuntamiento de Madrid, “Compendio-2019 de las Normas Urbanísticas del Plan General de Ordenación Urbana de Madrid de 1997. (actualizado 16-12-2019)”. Available online: http://www.madrid.org/cs/Satellite?blobcol=urldata&blobheader=application%2Fpdf&blobheadername1=Content-Disposition&blobheadervalue1=filename%3D05_Normas+Urban%C3%ADsticas.pdf&blobkey=id&blobtable=MungoBlobs&blobwhere=1352917079100&ssbinary=true

[51] Groupe Renault, “Electric vehicles, towards dynamic wireless charging?”, 2017. Available online: https://group.renault.com/en/news-on-air/news/electric-vehicles-towards-dynamic-wireless-charging/

[52] Hibridos y eléctricos, “Qualcomm y Renault realizan la carga inalámbrica dinámica (de 20 kW a 100km/h) de coches eléctricos”, 2017. Available online: https://www.hibridosyelectricos.com/articulo/tecnologia/qualcomm-renault-realizan-carga-dinamica-20-kw-100-km-h-coches-electricos-inalambricos/20170520173750014258.html

[53] OASIS, “MQTT Version 5.0, OASIS Standard”, 2019. Available online: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[54] Eclipse Foundation, “Eclipse Mosquitto: An open source MQTT broker”. Available online: https://mosquitto.org/

[55] Eclipse, “Eclipse Paho”, 2014. Available online: https://www.eclipse.org/paho/

[56] Python Package Index, “paho-mqtt 1.5.0”, 2013. Available online: https://pypi.org/project/paho-mqtt/

[57] Krzysztof Czarnecki, Waterloo Intelligent System Engineering (WISE) Lab, “On-Road Safety of Automated Driving System (ADS). Taxonomy and Safety Analysis Methods”, 2018. Available online: https://www.researchgate.net/publication/326546852_On-Road_Safety_of_Automated_Driving_System_ADS_-_Taxonomy_and_Safety_Analysis_Methods

[58] Míriam Antón Rodríguez, Laboratorio de Desarrollo de Sistemas Telemáticos impartido en la E.T.S.I.T de la Universidad de Valladolid, “Base de Datos Relacionales”, 2019.

[59] Digital Ocean, “SQLite vs MySQL vs PostgreSQL: A comparison of relational database management systems”, 2019. Available online: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

[60] Digital Guide IONOS, “MariaDB vs. MySQL”, 2020. Available online: https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/mariadb-vs-mysql/

[61] SQLite, “Documentation”, 2020. Available online: https://www.sqlite.org/docs.html

[62] Stackoverflow, 2008. Available online: https://stackoverflow.com/questions/200469/what-is-the-difference-between-a-process-and-a-thread

[63] Stackoverflow, 2010. Available online: https://stackoverflow.com/questions/3044580/multiprocessing-vs-threading-python

[64] GCO Investigation Group – Universidad de Valladolid, repository “GCOdeveloper/Twizycontest”, 2020.

Annex I

The objective of this annex is to explain how the graphics of Figure 18 and Figure 19 have been obtained. These graphics show how can be optimized the TwizyLine parking, in order to get a higher density of spots/m2 if it is compared with a conventional parking.

The parameters used to calculate them are the ones shown in Figure 74.

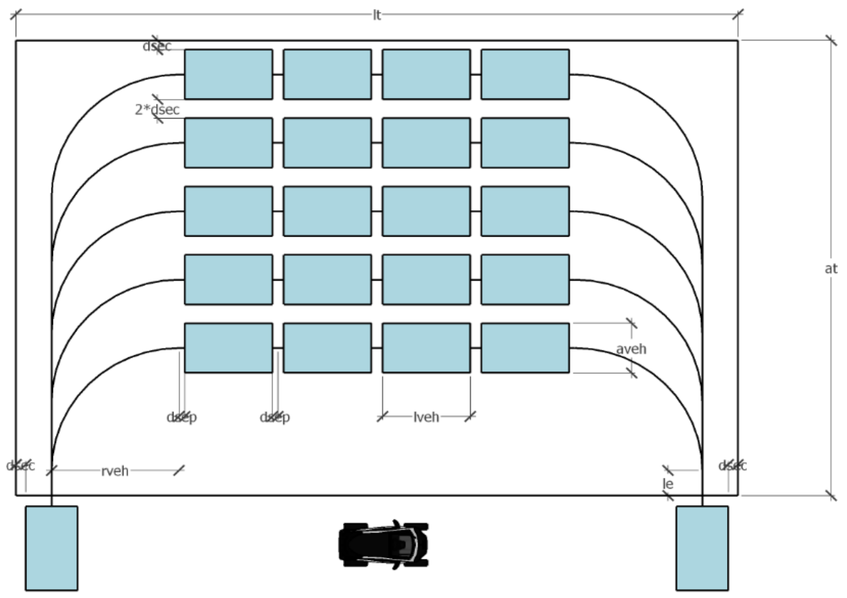


Figure 74. TwizyLine parking with its measurements [13].

It has been designed some MATLAB scripts to make the proper operations. The parameters shown in Figure 74 and then used in the MATLAB scripts means:

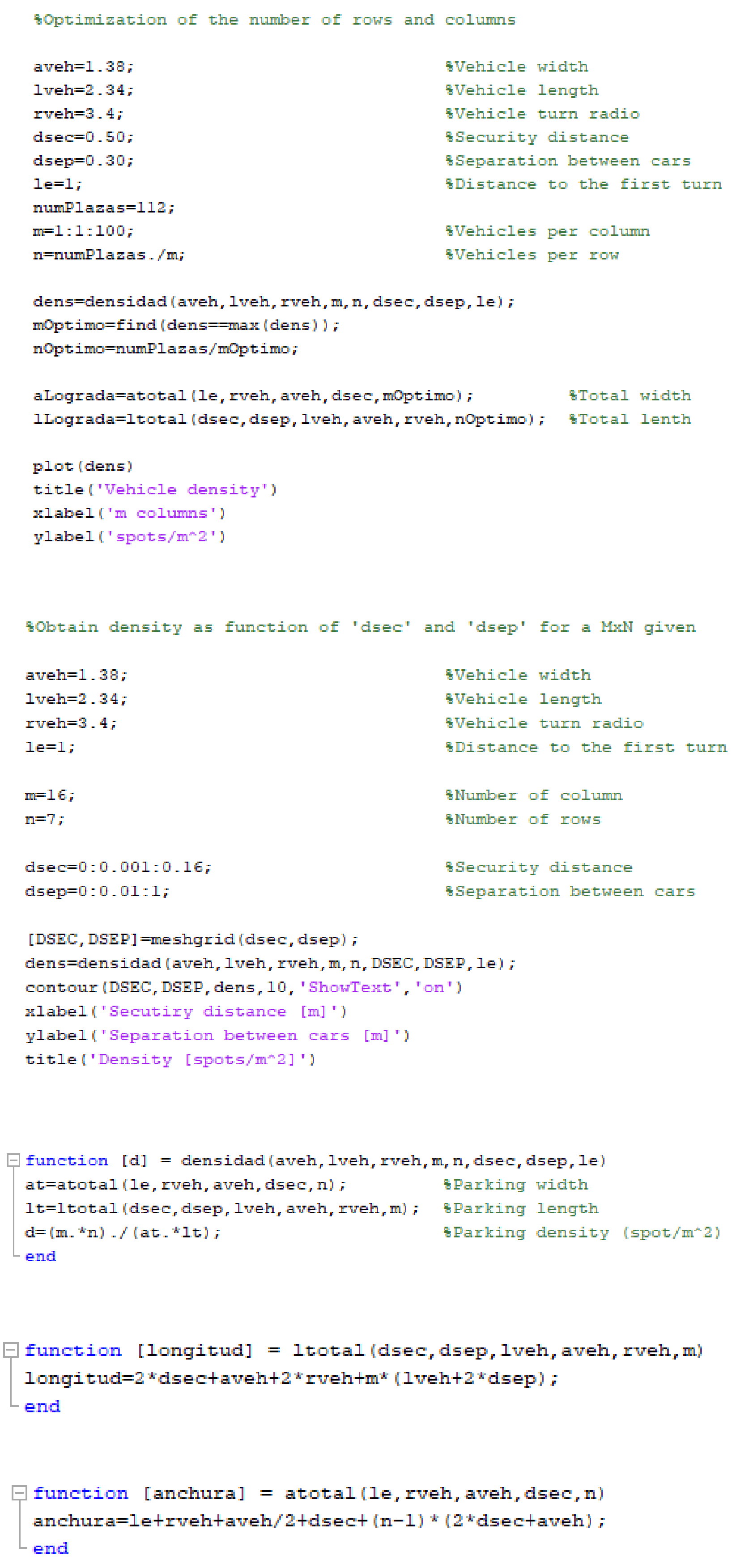
$$lt \ = \ total\ site\ leght$$
$$at \ = \ total\ site\ width$$
$$dsec \ = \ safety\ distance$$
$$dsep \ = \ distance\ between\ parked\ vehicles\ in\ the\ same\ row$$
$$lveh \ = vehicle\ lenght$$
$$aveh \ = \ vehicle\ width$$
$$rveh \ = \ vehicle\ turning\ radius$$
$$le \ = \ distance\ to\ the\ first\ turn$$
$$m \ = \ vehicles\ per\ row$$
$$n \ = \ number\ of\ rows$$

The total site length and the total site width can be obtained using the following equation:

$$at = le + rveh + \frac{aveh}{2} + dsec + (n-1) * (2 * dsec + aveh)$$

$$lt = 2 * dsec + aveh + 2 * rveh + m * (lveh + 2 * dsep)$$

Now it is shown the next figures the MATLAB scripts (Figure 75) used to calculate those graphics applying the notation and equation above explained.



```
%Optimization of the number of rows and columns

aveh=1.38;                              %Vehicle width
lveh=2.34;                              %Vehicle length
rveh=3.4;                               %Vehicle turn radio
dsec=0.50;                              %Security distance
dsep=0.30;                              %Separation between cars
le=1;                                   %Distance to the first turn
numPlazas=112;
m=1:1:100;                              %Vehicles per column
n=numPlazas./m;                         %Vehicles per row

dens=densidad(aveh,lveh,rveh,m,n,dsec,dsep,le);
mOptimo=find(dens==max(dens));
nOptimo=numPlazas/mOptimo;

aLograda=atotal(le,rveh,aveh,dsec,mOptimo);          %Total width
lLograda=ltotal(dsec,dsep,lveh,aveh,rveh,nOptimo);   %Total lenth

plot(dens)
title('Vehicle density')
xlabel('m columns')
ylabel('spots/m^2')



%Obtain density as function of 'dsec' and 'dsep' for a MxN given

aveh=1.38;                              %Vehicle width
lveh=2.34;                              %Vehicle length
rveh=3.4;                               %Vehicle turn radio
le=1;                                   %Distance to the first turn

m=16;                                   %Number of column
n=7;                                    %Number of rows

dsec=0:0.001:0.16;                      %Security distance
dsep=0:0.01:1;                          %Separation between cars

[DSEC,DSEP]=meshgrid(dsec,dsep);
dens=densidad(aveh,lveh,rveh,m,n,DSEC,DSEP,le);
contour(DSEC,DSEP,dens,10,'ShowText','on')
xlabel('Secutiry distance [m]')
ylabel('Separation between cars [m]')
title('Density [spots/m^2]')



function [d] = densidad(aveh,lveh,rveh,m,n,dsec,dsep,le)
at=atotal(le,rveh,aveh,dsec,n);          %Parking width
lt=ltotal(dsec,dsep,lveh,aveh,rveh,m);   %Parking length
d=(m.*n)./(at.*lt);                      %Parking density (spot/m^2)
end


function [longitud] = ltotal(dsec,dsep,lveh,aveh,rveh,m)
longitud=2*dsec+aveh+2*rveh+m*(lveh+2*dsep);
end


function [anchura] = atotal(le,rveh,aveh,dsec,n)
anchura=le+rveh+aveh/2+dsec+(n-1)*(2*dsec+aveh);
end
```

Figure 75. MATLAB code required to represent graphics of Figure 18 and Figure 19.

Annex II

In this annex, it is shown a table with the main properties of all the MQTT messages.

Message	Code	Description
CONNECT	0x10	Connect request
CONNACK	0x20	Connect acknowledgement
PUBLISH	0x30	Publish message
PUBACK	0x40	Publish acknowledgement (QoS 1)
PUBREC	0x50	Publish received (QoS 2 delivery part 1)
PUBREL	0x60	Publish released (QoS 2 delivery part 2)
PUBCOMP	0x70	Publish complete (Qos 2 delivery part 3)
SUBSCRIBE	0x80	Subscribe request
SUBACK	0x90	Subscribe acknowledgement
UNSUBSCRIBE	0xA0	Unsubscribe request
UNSUBACK	0xB0	Unsubscribe acknowledgement
PINGREQ	0xC0	PING request
PINGRESP	0xD0	PING response
DISCONNECT	0xE0	Disconnect notification

Table 10. MQTT type of messages [53].

Annex III

This Final Degree Project is focused on dealing with the Twizy, application and server failures. It is relevant to highlight that this project is part of a bigger project called TwizyLine, so in the future, more failures can be added if it is needed. Below it is the Table 11 with all the failures implemented stored in the database (in section 4.3.2 is shown the full TwizyLine database). In this table, COM-MOD refers to the communications module of the Twizy and the CON-MOD refers to the control module of the Twizy both implemented in the Final Degree Project of Ignacio Royuela González [14].

Source	Code	Hexadecimal code	Failure type	Failure name	Explanation
Twizy	1	0x001	Error	CON_STATUS not received	The CON_STATUS frame is not being received. Is the CON-MOD on? Is the CAN-Bus connection of both modules correct? The Twizy cannot change from one mode to another.
Twizy	2	0x002	Error	MOD-CON does not respond CON_STATUS signal	CON-MOD does not change status when ordered to do so. Is the CON-MOD correctly configured? Connection failed between server and Twizy.
Twizy	3	0x003	Error	MOD-CON does not respond to PAUSE signal	When sending a PAUSE or CONTINUE command from the COM-MOD, the CON-MOD does not respond correctly as the PAUSE signal does not change. High-risk inside the parking. Is the CON-MOD correctly configured?
Twizy	4	0x004	Error	GOTO-ACK not received multiple times	CON-MOD does not send an ACK when it has to do so. Is the CON-MOD correctly configured?
Twizy	5	0x005	Warning	Malformed GOTO frame received from Communication Module	CON-MOD has received a GOTO message but its information is not in the proper way as described in section 4.4.

Twizy	6	0x006	Error	Unexpected CON-MOD frame	CON-MOD is sending out signals that don't make sense right now. Is CON-MOD well configured? Problems in the control of the vehicle, powerful hazards that can turn into risks.
Twizy	9	0x009	Error	GPS not connected	The GPS module is not connected to the communication module. Is the COM-MOD correctly configured? Is the GPS broken? Is the GPS disconnected? Twizy can be stolen.
Twizy	10	0x00A	Warning	GPS is not receiving signal for a long time	The GPS module is connected but it is not receiving any signal from the satellites for a long time. Is the GPS broken? Is the car just in a place where there is no mobile coverage?
Twizy	11	0x00B	Warning	Battery data not received	The CAN message informing about the battery charge on the vehicle's CAN-bus is not detected. Is there a problem in the vehicle?
Twizy	20	0x014	Error	Connection refused - Not authorized	The MQTT broker is rejecting the connection due to no authentication. Is the client sending the correct authentication data?
Twizy	21	0x015	Error	Connection refused - Bad username or password	The MQTT broker is rejecting the connection due to a bad authentication. Is the client sending the correct username and password?
Twizy	25	0x019	Error	Server client is not responding	The server is not responding with the message CONNECTED when called. Is the server running?
Twizy	26	0x01A	Warning	Unexpected MQTT message	The server is sending an incorrect message. Is the server well configured?
Twizy	27	0x01B	Error	Lost connection with MQTT server	Ping responses from the server are not being received. Has CON-MOD lost the

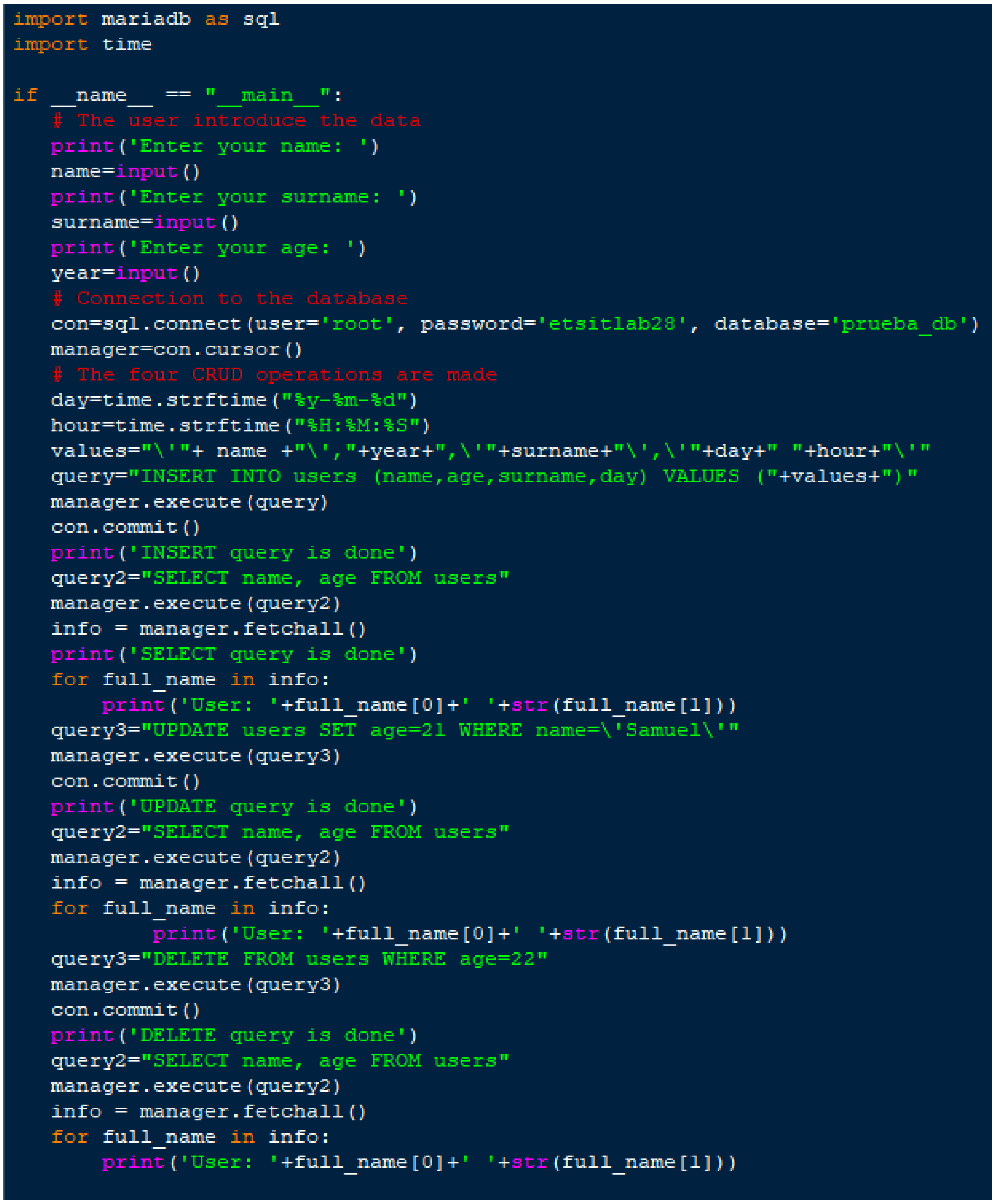
					internet connection? Has the MQTT broker stopped?
Twizy	129	0x081	Error	COM_STATUS not received	The COM_STATUS frame is not being received. Is the COM-MOD on? Is the CAN-Bus connection of both modules correct?
Twizy	130	0x082	Error	RFID-ACK not received	The COM-MOD has not received an ACK to the RFID message. Is it the connection with the server established? Has been the RFID message sent properly? Is there any problem in the connection between CON-MOD and COM-MOD?
Twizy	131	0x083	Error	CON-ERR-ACK not received	CON-MOD has sent an error message to the communication module but the COM-MOD has not responded with the corresponding ACK.
Twizy	132	0x084	Warning	Unexpected CON-MOD frame	A frame, which has arrived at the CON-MOD, has not the proper format. Is there any problem in the connection between CON-MOD and COM-MOD?
Server	257	0x101	Error	Username is not unique	The username passed on the LOGON is not valid because it is already in use
Server	258	0x102	Error	Parking busy. Try again to get your Twizy	In that moment the origin Park is busy, so the user must try again.
Server	259	0x103	Warning	That parking id does not exist	Administrator is asking for information about a parking which does not exist.
Server	260	0x104	Warning	That car id does not exist	Administrator is asking for information about a Twizy which does not exist.
Server	266	0x10A	Error	AM-ON OK not received	The server does not receive the AM-ON OK from the Twizy. Is the COM-CON well configured?

Server	267	0x10B	Error	AM-OFF OK not received	The server does not receive the AM-OFF OK from the Twizy. Is the COM-CON well configured?
Server	268	0x10C	Error	CONTINUE OK not received	The server does not receive the CONTINUE OK from the Twizy. Is the COM-CON well configured?
Server	269	0x10D	Error	PAUSE OK not received	The server does not receive the PAUSE OK from the Twizy. Is the COM-CON well configured?
Parking	513	0x201	Error	Barrier does not work properly	The parking reports an error in the performance of some of its barriers. Parking must stop its service.
Parking	514	0x202	Error	RFID tag has been damaged	The parking reports an error in the performance of some of its RFID tags. Parking must stop its service.
Parking	515	0x203	Error	Turnstile do not work properly	The parking reports an error in the performance of some of its turnstile. Parking must stop its service.
Parking	516	0x204	Error	Magnetic tape has been damaged	The parking reports an error in the circuit. Parking must stop its service.

Table 11. TwizyLine failures implemented in the back-end software.

Annex IV

Annex IV aims to show the reader how to connect to the TwizyLine database using python. It is especially relevant the library ‘mariadb’ which is in charge of connecting to our database. Below there is a code of example (Figure 76). In this example, it is done the four operations CRUD: create (“INSERT”), read (“SELECT”), update (“UPDATE”) and delete (“DELETE”).

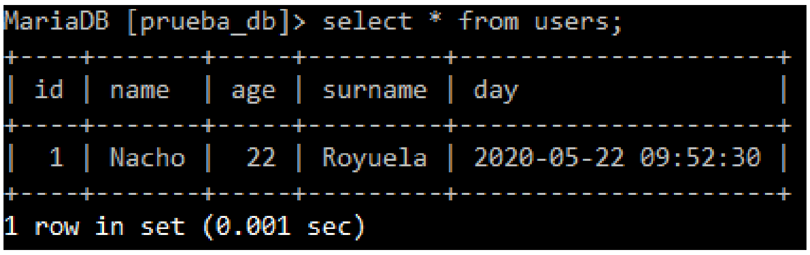


```python
import mariadb as sql
import time

if __name__ == "__main__":
   # The user introduce the data
   print('Enter your name: ')
   name=input()
   print('Enter your surname: ')
   surname=input()
   print('Enter your age: ')
   year=input()
   # Connection to the database
   con=sql.connect(user='root', password='etsitlab28', database='prueba_db')
   manager=con.cursor()
   # The four CRUD operations are made
   day=time.strftime("%y-%m-%d")
   hour=time.strftime("%H:%M:%S")
   values="\'"+ name +"\',"+year+",\'"+surname+"\',\'"+day+" "+hour+"\'"
   query="INSERT INTO users (name,age,surname,day) VALUES ("+values+")"
   manager.execute(query)
   con.commit()
   print('INSERT query is done')
   query2="SELECT name, age FROM users"
   manager.execute(query2)
   info = manager.fetchall()
   print('SELECT query is done')
   for full_name in info:
       print('User: '+full_name[0]+' '+str(full_name[1]))
   query3="UPDATE users SET age=21 WHERE name=\'Samuel\'"
   manager.execute(query3)
   con.commit()
   print('UPDATE query is done')
   query2="SELECT name, age FROM users"
   manager.execute(query2)
   info = manager.fetchall()
   for full_name in info:
           print('User: '+full_name[0]+' '+str(full_name[1]))
   query3="DELETE FROM users WHERE age=22"
   manager.execute(query3)
   con.commit()
   print('DELETE query is done')
   query2="SELECT name, age FROM users"
   manager.execute(query2)
   info = manager.fetchall()
   for full_name in info:
       print('User: '+full_name[0]+' '+str(full_name[1]))
```

Figure 76. Python code to connect and to make CRUD operations in mariaDB database.

In this code, first, it is asked for your name, surname and age. Then, it is connected to the database using the method connect from the mariadb library. In this method, it is required to introduce the user who accesses to the database, its password and the name of the database to which we want to access.

This code works with the table ‘users’ form ‘prueba_db’ database. At the beginning, there is just the data shown in Figure 77 stored in that table.



```
MariaDB [prueba_db]> select * from users;
+----+-------+-----+---------+---------------------+
| id | name  | age | surname | day                 |
+----+-------+-----+---------+---------------------+
|  1 | Nacho |  22 | Royuela | 2020-05-22 09:52:30 |
+----+-------+-----+---------+---------------------+
1 row in set (0.001 sec)
```

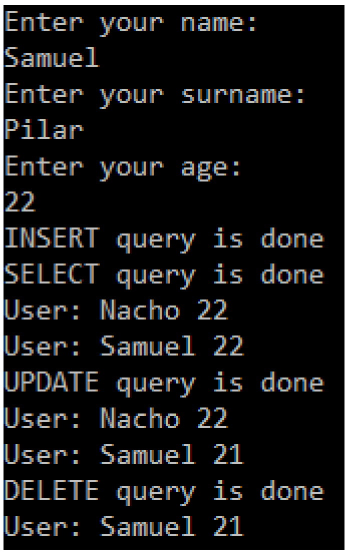
Figure 77. Table 'users' from ‘prueba_db’ database.

First, it is made the query “INSERT” to add to the database the data introduced previously plus the hour when that data was introduced through the keyboard.

Then, the query “UPDATE” is made. Its goal is to change the field ‘age’ from table ‘users’ to those registers which field ‘name’ equals to ‘Samuel’.

And finally the query “DELETE” is made, to remove those register from table ‘users’ whose registers have the field ‘age’ equals 22.

After each of these three queries, a "SELECT" query is made to see the contents of the 'users' table and the result is printed in console (Figure 78).

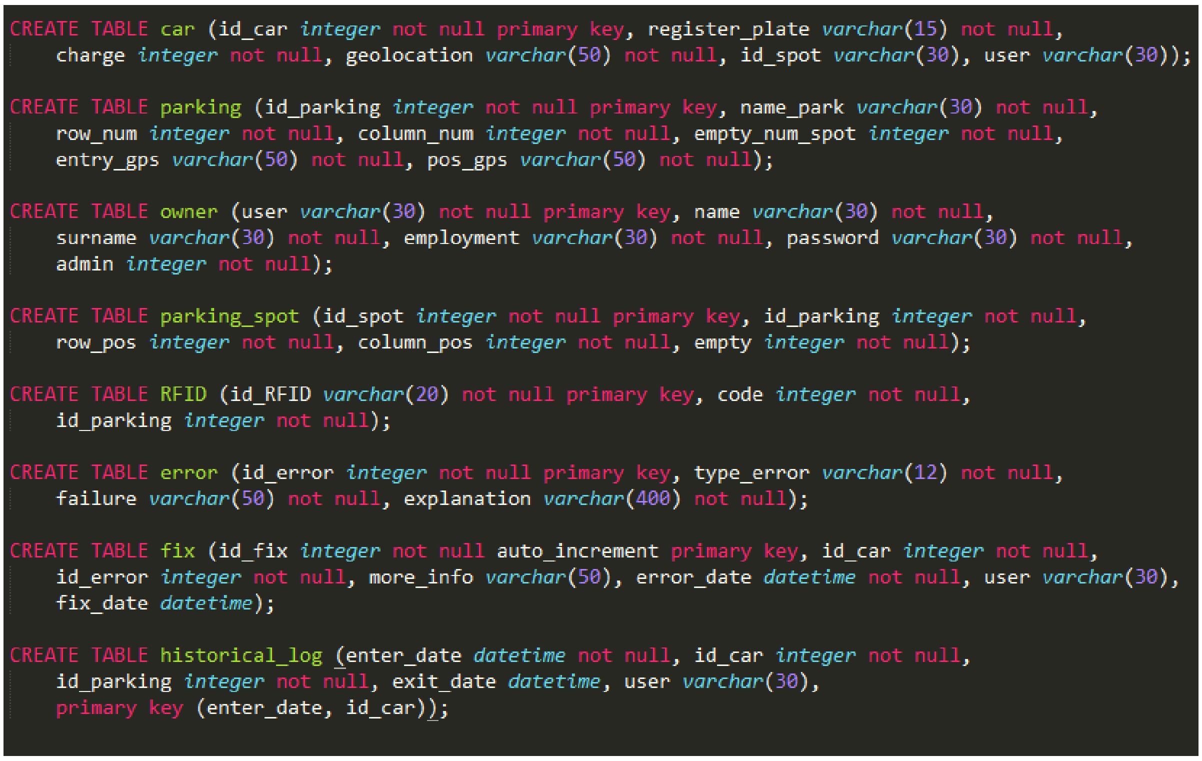


```
Enter your name:
Samuel
Enter your surname:
Pilar
Enter your age:
22
INSERT query is done
SELECT query is done
User: Nacho 22
User: Samuel 22
UPDATE query is done
User: Nacho 22
User: Samuel 21
DELETE query is done
User: Samuel 21
```

Figure 78. Information shown in console after running the code of Figure 76.

Annex V

In this Annex V there is the code written in SQL syntax to create the TwizyLine database (Figure 79). The name of the database is ‘TwizyLineDB’ and it is made up of 8 tables.



```sql
CREATE TABLE car (id_car integer not null primary key, register_plate varchar(15) not null,
    charge integer not null, geolocation varchar(50) not null, id_spot varchar(30), user varchar(30));

CREATE TABLE parking (id_parking integer not null primary key, name_park varchar(30) not null,
    row_num integer not null, column_num integer not null, empty_num_spot integer not null,
    entry_gps varchar(50) not null, pos_gps varchar(50) not null);

CREATE TABLE owner (user varchar(30) not null primary key, name varchar(30) not null,
    surname varchar(30) not null, employment varchar(30) not null, password varchar(30) not null,
    admin integer not null);

CREATE TABLE parking_spot (id_spot integer not null primary key, id_parking integer not null,
    row_pos integer not null, column_pos integer not null, empty integer not null);

CREATE TABLE RFID (id_RFID varchar(20) not null primary key, code integer not null,
    id_parking integer not null);

CREATE TABLE error (id_error integer not null primary key, type_error varchar(12) not null,
    failure varchar(50) not null, explanation varchar(400) not null);

CREATE TABLE fix (id_fix integer not null auto_increment primary key, id_car integer not null,
    id_error integer not null, more_info varchar(50), error_date datetime not null, user varchar(30),
    fix_date datetime);

CREATE TABLE historical_log (enter_date datetime not null, id_car integer not null,
    id_parking integer not null, exit_date datetime, user varchar(30),
    primary key (enter_date, id_car));
```

Figure 79. Code in SQL syntax to create the TwizyLine database.

Some fields have special attributes; which meanings are:

- ‘primary key’: This attribute determines the field where is stored the unique identifier for each register. There can be more than one primary key.
- ‘not null’: These fields cannot be empty or with a ‘null’ values.
- ‘auto_increment’: The value of this field in each register is selected automatically. It is written in the table the last value assigned to that field plus 1. It is used for ‘integer’ fields.