Universidade do Minho
Escola de Engenharia
Departamento de Informática

Luís Francisco Mendes Lopes

Automated Driving: an approach to Automated Guided Vehicles in a controlled environment.

December 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Luís Francisco Mendes Lopes

Automated Driving: an approach to Automated Guided Vehicles in a controlled environment.

Master dissertation
Integrated Master in Informatics Engineering

Dissertation supervised by
Prof. Pedro Rangel Henriques
Nuno Viveiros Ponte

December 2022

AUTHOR COPYRIGHTS AND TERMS OF USAGE BY THIRD PARTIES

This is an academic work which can be utilized by third parties given that the rules and good practices internationally accepted, regarding author copyrights and related copyrights.

Therefore, the present work can be utilized according to the terms provided in the license bellow.

If the user needs permission to use the work in conditions not foreseen by the licensing indicated, the user should contact the author, through the RepositóriUM of University of Minho.

License provided to the users of this work

Attribution-ShareAlike
CC BY-SA
https://creativecommons.org/licenses/by-nc/4.0/

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Assinado por: **LUÍS FRANCISCO MENDES LOPES**
Num. de Identificação: 15861239
Data: 2022.12.12 09:56:14+00'00'

Luís Francisco Mendes Lopes

ACKNOWLEDGEMENTS

First of all I truly thank my mentor, Professor Pedro Rangel Henriques, and my company supervisor, Nuno Viveiros Ponte, for having guided, taught, and encouraged me on this journey that was the master's thesis. Without them none of this would be possible. I am also grateful to all of my friends, who not only helped me by debating ideas and possible solutions for the project, but also were always ready to assist and cheer me up during these five demanding years. Lastly, I also appreciate all my family members for their unconditional support throughout this academic path. This goes to my mother Francelina, my father Joaquim, my brother João, and my confidant who has stood by me since the beginning of my academic life, Inês. I want to thank them for being always present in all the moments provided by my academic life, for motivating me and being my foundation so that everything became possible.

ABSTRACT

The growing need for process optimisation is one of the main drivers of technological modernisation in companies. Process automation has been one of the main initiatives in this modernisation task. In many scenarios, the movement of any kind of vehicles or other moving objects requires human hands.

In this context, this Master's Thesis, included in the fifth year of the Integrated Master in Informatics Engineering in the University of Minho, reports the work developed in a business context with the help of Deloitte Technologys S.A. that welcomed the student in a curricular internship program. The aim is to develop an implementation of of a simulator of automated guided vehicles in a controlled environment, which can reveal itself as an initiative for the modernisation of the logistic process of moving mobile objects or vehicles.

In order to develop the Master's Project, a work methodology was outlined. This methodology is supported on four crucial stages to reach the proposed objectives: documentary, proposal, pre-development and development. The documentary analysis was made to develop a better understanding of the context and to study other developed projects and tools that could be necessary for the proposal and development. In the solution proposal we started by dividing the problem in several components, in a kind of "divide and conquer" method. During this process, a conceptual map was developed, which will be analysed in the document and will guide us to the first Conceptual Architecture Diagram. The next step is through the analysis of the Conceptual Architecture Diagram to study which modules will be out of the scope of what is necessary for the full functioning of the Master project, taking certain components as assumptions and substituting some modules by other solutions, so the development can be faster however in a way to not neglect the required functionality for the full operation of the solution. After these steps, the development of the solution will follow, putting into practice all the knowledge acquired in the previous stages.

The Master's Project culminated in five micro-services: Vehicle, Controller, Map, Vehicle/Sessions Micro-Service and Alert; and a web application that allows the user to check with a GUI data about vehicles, sessions, alerts and the map of the controlled environment. These micro-services and the web application working simultaneously allow having a vehicle simulator automatically driven in a controlled environment.

Keywords: Automated Guided Vehicles, Automated Driving, Path Calculation, Controlled Environment, Impact prediction

RESUMO

A crescente necessidade de otimização de processos é uma das principais motivações para a modernização tecnológica nas empresas. A automatização de processos tem sido uma das principais iniciativas nesta tarefa de modernização. Em muitos cenários, a movimentação de qualquer tipo de veículos ou outros objetos móveis requer mão humana.

Neste contexto esta Dissertação de Mestrado, incluída no quinto ano do Mestrado Integrado em Engenharia Informática na Universidade do Minho, relata o trabalho desenvolvido em contexto empresarial com a ajuda da Deloitte Technologys S.A que acolheu o aluno num programa de estágio curricular. É procurado desenvolver uma implementação de de um simulador de veículos automaticamente conduzidos em ambiente controlados, que se pode revelar uma iniciativa para a modernização do processo logístico de movimentação de objetos móveis ou veículos.

De forma a desenvolver o Projeto de Mestrado foi delineada uma metodologia de trabalho. Esta metodologia está apoiada em quatro etapas crucias para atingir os objetivos propostos: documentário, proposta e pré-desenvolvimento e desenvolvimento. A análise documental foi feita para desenvolver uma melhor compreensão do contexto e estudar outros projetos desenvolvidos, e ferramentas que poderão ser necessárias para a proposta e desenvolvimento. Na proposta de solução começou-se por repartir o problema em várias componentes, numa espécie de estratagema de "divide and conquer". No decorrer deste processo foi desenvolvido um mapa conceptual que será analisado no documento e guiar-nos-á ao primeiro Diagrama Conceptual de Arquitetura. A etapa seguinte passa por através da análise do Diagrama Conceptual de Arquitetura estudar quais módulos estarão fora do âmbito daquilo que é necessário para o pleno funcionamento do projeto de Mestrado, tomando certos componentes como pressupostas e substituindo alguns módulos por outras soluções, assim o desenvolvimento poderá ser mais célere não descorando de funcionalidades necessárias para o pleno funcionamento da solução. Após estas etapas, seguir-se-á o desenvolvimento da solução, pondo em prática todo o conhecimento adquirido nas etapas anteriores.

O projeto de Mestrado culminou em cinco micro-serviços: Veículo, Controlador, Mapa, Veículo/Sessões Micro-Serviço e Alerta; e uma aplicação web que permite o utilizador verificar com uma GUI dados sobre veículos, sessões, alertas e o mapa do ambiente controlado. Estes micro-serviços e a aplicação web em simultâneo permitem ter um simulador de veículos automaticamente conduzidos em ambiente controlado.

Palavras-Chave: Veículos Automaticamente Conduzidos, Condução Automatizada, Cálculo de Rota, Ambiente Controlado, Previsão de Impacto

CONTENTS

1 INTRODUCTION 1
1.1 Motivation 2
1.2 Objectives 2
1.3 Research Method 2
1.4 Document Structure 3
2 STATE OF THE ART 5
2.1 Automated Guided Vehicles 5
2.2 Environment Recognition and Mapping 5
2.3 Stationary system control and Monitoring 6
2.3.1 Network Communication Protocols 6
2.3.2 Security Protocols 8
2.3.3 Clock Synchronization 8
2.4 Summary 9
3 PROPOSED APPROACH 10
3.1 Environment 10
3.2 Map 11
3.3 Vehicle 12
3.4 Controller 13
3.5 Monitoring 15
3.6 Conceptual Architecture Diagram 16
3.7 Summary 17
4 PRE-DEVELOPMENT CONSIDERATIONS 18
4.1 Environment Recognition and Map 18
4.2 Monitoring 20
4.3 Learning Capabilities 21
4.4 Logistics and Conflict Management 21
4.5 Conceptual Diagram Update 21
4.6 Summary 22
5 DEVELOPMENT 23
5.1 Technologies Used 24
5.1.1 Spring 24
5.1.2 TCP and HTTP 25
5.1.3 Google Protocol Buffers 25

5.1.4 Angular 26
5.1.5 H2 Database Engine 26
5.1.6 Mapbox GL JS 26
5.2 General Project Structure and Conventions 27
5.3 Logical Data Model 28
5.4 AGV-MAP 30
5.5 AGV-ALR 31
5.6 AGV-VMS 32
5.7 AGV-CTL 33
5.7.1 "Cache" Implementation 34
5.8 AGV-VEH 34
5.9 AGV-FE 35
5.10 Vehicle-Controller Interaction 38
5.10.1 Registration and Handshake 38
5.10.2 TCP Data Exchange 42
5.11 Vehicle Impact and Off-Lane Monitoring 43
5.11.1 Vehicle Impact 44
5.11.2 Vehicle Off-Lane 45
5.12 Summary 46
6 SYSTEM OVERVIEW 48
7 CONCLUSION 51

LIST OF FIGURES

Figure 1 TCP Packet 7
Figure 2 UDP Packet 7
Figure 3 Initial conceptual map state. 10
Figure 4 Environment bracket of the conceptual map. 11
Figure 5 Map bracket of the conceptual map. 11
Figure 6 Layer bracket of the conceptual map. 12
Figure 7 Vehicle bracket of the conceptual map with characterization bracket open. 12
Figure 8 Communication bracket of vehicle bracket zoomed in. 13
Figure 9 Localization and State bracket of vehicle bracket open. 13
Figure 10 Controller bracket of the conceptual map. 14
Figure 11 Safety bracket of controller bracket open. 14
Figure 12 Logistics and Conflict Management bracket of controller bracket open. 15
Figure 13 Learning bracket of controller bracket open. 15
Figure 14 Monitoring bracket of the conceptual map. 15
Figure 15 Conceptual Architecture Diagram. 16
Figure 16 OSM extracted Map Without processing or filtering. 19
Figure 17 Revised Conceptual Diagram 22
Figure 18 Micro-service Architecture of the solution 23
Figure 19 AGV-CTL summarised structure 27
Figure 20 Logical Data Model 29
Figure 21 Web App Mockup. 35
Figure 22 AGV-FE summarised structure 36
Figure 23 Vehicle Controller Registration and Handshake General Sequence Diagram . 39
Figure 24 TCP Data Exchange Diagram between Controller and Vehicle 43
Figure 25 Diagram of how to find the four coordinates that form the rectangle of the vehicle 44
Figure 26 Schematic diagram of Off-Lane calculation 45
Figure 27 Vehicle intersecting with line and not line segment. 46
Figure 28 AGV-FE Web Page. 48
Figure 29 Close up right click action on a vehicle 49

Figure 30 Close up left click action on a vehicle and selecting "Set Destination" 50
Figure 31 Vehicle Off Lane Alert in AGV-FE 50
Figure 32 Project Gantt chart. 52

LIST OF LISTINGS

4.1 Excerpt from OSM file 19
4.2 GeoJSON Example 20
5.1 Data class from AGV-CTL 34
5.2 Excerpt from map creation method on AGV-FE 37
5.3 Excerpt from method CreateLayer() from AGV-FE 37
5.4 Method for Vehicle Registration from AGV-VEH 38
5.5 VehInfoDTO Class excerpt 39
5.6 VehTcpInfoDTO Class excerpt 40
5.7 Controller Gateway 41
5.8 Outbound Integration Flow creation 41
5.9 Inbound Integration Flow creation 41
5.10 Protobuff file used to structure vehicle state data 42

ACRONYMS

AES Advanced Encryption Standard.
AGV Automated Guided Vehicle.
API Application Programming Interface.

CRUD Create Read Update Delete.
CSS Cascading Style Sheets.

DAO Data Access Object.
DTO Data Transfer Object.

GPS Global Position System.

HTML HyperText Markup Language.
HTTP Hypertext Transfer Protocol.

I4.0 Industry 4.0.
IEEE Institute of Electrical and Electronics Engineers.
IoT Internet of Things.
IP Internet Protocol.
IPsec Internet Security Protocol.
ISO/OSI International Organization for Standardization/Open Systems Interconnection.

JOSM Java OpenStreetMap Editor.
JPA Java Persistence API.
JSON JavaScript Object Notation.

LiDAR Light Detection and Ranging.

MAC Media Access Control.

NTP Network Time Protocol.

OS Operating System.
OSM OpenStreetMap.

PCT Private Communications Technology.

REST Representational State Transfer.

RGB Red, Green and Blue.
RST Reset.

SLAM Simultaneous Localization and Mapping.
SQL Structured Query Language.
SSL Secure Sockets Layer.
SSL/TLS Secure Sockets Layer/Transport Layer Security.
SYN Synchronize.

TCP Transmission Control Protocol.
TLS Transport Layer Security.

UDP User Datagram Protocol.
URL Uniform Resource Locator.

XML Extensible Markup Language.

1 INTRODUCTION

Since the beginning of industrial revolution, the primary goal was to increase profits. Allied to the greed of profits, the higher demand of products guided companies to rise the manufacturing leading to an increase demand of natural resources, resulting in a high industrial waste.

This industrial developments has brought many threats to nature. The fast consume of resources, allied with carbon emissions from manufacturing, contributed to climate changes. Therefore, there is a urge to evolve manufacturing and production to become more sustainable and environmentally friendly. Leaving our future generations, natural resources that can fulfill their own needs (Bílgín, 2021).

Thus in 2011, Henning Kagermann, the head of the German National Academy of Science and Engineering unveiled a government-sponsored industrial initiative that was named Industry 4.0 (I4.0). This initiative initially was intended to represent Germany's strategic vision of the future, but rapidly was adopted all around the world.

This initiative, also known as fourth industrial revolution focus in manufacturing innovation, exploring resources like Internet of Things, artificial intelligence, cloud computing and other technologies (Boone, 2020). Lying on the core of I4.0, there are the "Smart Factories" or "Digital Factories". This Smart Factories take advantage of advanced manufacturing, as well technology to make their production more efficient using automatization, from production robots to transport devices (Horst and Santiago).

This transportation devices can be Automated Guided Vehicle (AGV), that are one of the optimizations in manufacturing or logistic systems (Mehami et al., 2018).

In many scenarios there are objects or many type of vehicles, such as drones, forklifts and any type of object with mobility capacities, that need to be moved through human hand. In spite of being one of the most needed change to modernize companies, factories and warehouses, this is an extreme challenge, due to the diversity of elements to be controlled, the differences in the environment as well as the fact that there is often more than one object to be moved. This can lead to several issues, such as *conflict management* - which consists in integrate all factors that can contribute to solve and prevent conflicts (Barkovic et al., 2008); *path selection* - that consists in vehicle planning with advance its movements and where to move through the environment to reach its destination (Mahdawy and Mougy, 2021); *impact prediction* and others, that need to be overcome.

1.1 MOTIVATION

Automating this tasks, mainly made by human hand, will not only reduce costs but also increase speed and decrease errors that can be created by human error. Warehouses from delivering companies can use this system to transport the parcels (Luo et al., 2011), improving and hastening their delivery process. But AGVs can have several uses besides industrial or warehouses appliances. In a research Pedan et al. (2017), simulated the use of AGV integration and concluded that this integration could save up to 345 minutes of a total of 1440 minutes, for day. This represents almost 25% of the day. AGV can be used in food, waste, clean or used laundry transportation. It can also be used to transport patients in beds or wheel chairs. In fact there are a lot of healthcare appliances that AVG can be used. And with the development of simple AGVs, we can lead to better warehouse, industrial, cities and healthcare logistic, gaining time and efficiency.

1.2 OBJECTIVES

As previously stated, the use of modern AGV can be a big step up in industry, as well in healthcare. In that way, this Master's Project main objective is the implementation of a simulator of Automated Guided Vehicle in a controlled environment. In order to achieve this objective, it is needed to:

- Characterize a controlled environment;
- Implement a software capable of controlling a vehicle.
- Implement communication between vehicles and central control.
- Simulate a controlled environment of vehicles.

1.3 RESEARCH METHOD

The methodology used to achieve the objectives will be the following:

- Bibliographic research of remote controlled vehicles, using the bibliography used above as starting point;
- Analysis of existing solutions, tools and programming approaches found in the research;

- Proposal of a software architecture to accomplish the objectives above.
- Choice of tools and programming languages to be used;
- Implementation of the solution;
- Test the developed solution.

If the feedback given by the tests done is not as expected, the steps listed will be iterated until the tests give a solid feedback.

1.4 DOCUMENT STRUCTURE

The document is structured on seven chapters: **Introduction**, **State of Art**, **Proposed Approach**,**Pre-Development Considerations** , **Development**, **System Overview** and **Conclusion**.

The **Introduction**, as the name suggests, will introduce in what context does the Project applies and the motivation to fulfill it. Besides that it will define the main goal and sub goals that are needed to materialize the main goal. It will also define a research methodology to be follow in this Master's project.

In the **State of The Art** chapter it will be discussed the different concepts of AGVs, what it is, when it was first implemented and its development through out the years, as well several technologies. More modern or ancient that are used to implement AGVs. This chapter will be useful to rearrange ideas of implementation and what technologies could be used.

After a brief introduction and a extended research, **Proposal Approach** chapter will, in a detailed way, describe what could be a solution and implementation for the problem, approaching how can the problem be solved by defining structures and how they connect, share and communicate to each other.

Before starting the development it is necessary to establish some points to facilitate and speed up the development. The chapter **Pre-Development Considerations** will address some components and modules of the project, which will not be developed because they are not strictly necessary, or whose subject matter is not so relevant to the solution.

After addressing the pre-development considerations, we will move on to the next phase of **Development**. This chapter will discuss how the solution was developed from the solution proposal, discussing frameworks, tools and languages used, as well as the projects that constitute the solution and how they interact with each other.

The chapter **System Overview** describes and contains images of the result of the developed web application. It also explains what information is possible to inspect and interactions between user and application.

Last but not least, there is the **Conclusion** chapter, that will close the dissertation with a synthesis of the most important ideas discussed in each chapter and directions for future work. Moreover, the chapter includes some considerations about the work done and the outcomes reached at the end of the project.

2 STATE OF THE ART

The concept of the Automated Guided Vehicle was created seven decades ago, since then there where a lot of development in the vision of AGVs and technologies used. In the following sections it will be discussed the origins of AGVs and the current state-of-the-art of both AGVs and technologies.

2.1 AUTOMATED GUIDED VEHICLES

It was in the beginnings of 1950's that AGVs were introduced. Follow by the idea of replacing an human hand driven tractor trailer for one that was automated, Barrett-Cravens implemented, in 1953, the first automated guided vehicle, that used an electrically conductive strip mounted in the ground to know its path. Europe entered this market in 1956, with the company EMI, the concept was quite similar, since the vehicle followed a colored strip on the floor that was recognized by an optical sensor (Ullrich, 2014). Nowadays the market for AGVs has grown, and there are plenty of appliances for this technology. Due to its high flexibility, AGVs are also used in the material flow, transporting goodies or products, both indoor and outdoor. The development of AGVs is towards higher functionalities, ditching physical guide-paths it gets more capabilities for navigation and free-moving (Ujvari, 2003). Modern AGVs uses a stationary system control, that have the tasks of **environment** recognition, in order to visualize and navigate, administrating orders, driving optimization, regulation and arrangement, as well the task of communication with the vehicles. (Schulze and Wullner, 2006)

2.2 ENVIRONMENT RECOGNITION AND MAPPING

Nowadays Maps are critical elements for different use cases, like Internet of Things (IoT), mobility and autonomous driving. Maps need to be accurate, global and have detailed attributes. This requirements are evolving maps and mapping, to an complex, high-tech and expensive process (Bonte and Hodgson, 2018). In fact, maps are the heart of autonomous driving as well automated drive, given by the fact that a map represents an **environment**, where we can see throughout **coordinates** the location of obstacles, **vehicles** and pathways.

In order to represent a environment, maps need layers that have different information. There can be various layers like infrastructural layers, static objects layers, pathway layers. There are many providers, like OpenStreetMap (OSM) [1], MapBox [2], Google Maps [3], Bing Maps [4] and many others, that have Application Programming Interface (API) to provide the client maps and other tools. Despite of providers having evolve, and offer different types of maps, the use of it would restrict the projects, because providers cannot provide indoor maps, neither locate a vehicle indoor. For an indoor use, there are technologies like Simultaneous Localization and Mapping (SLAM) and Perception-Driven approaches. SLAM can use several sensors such as ultrasonic sensors, laser scanners and Red, Green and Blue (RGB) Cameras, for estimating a robot's location and simultaneously construct a Map of the environment (Taheri and Xia, 2021). Perception-Driven is a grid-based and object-based representations of the environment, disassembling the complexity of e.g. an city, by classifying static objects from moving vehicles and road users. In addition there is more relevant information about the environment that can be used do determine road path, free parking spots, walls and obstacles (Rieken et al., 2015).

2.3 STATIONARY SYSTEM CONTROL AND MONITORING

2.3.1 *Network Communication Protocols*

A protocol defines the former,the order and the rules that messages should follow and exchange between entities in order to establish communication. The brain of an AGV is the computer installed that receives information and processes it. In order to communicate with the central, as referenced in Section 2.1, it is needed to create a computer network, using protocols (Kurose and Ross, 2012). There are several communication protocols, but for this purpose it will be studied only Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Zigbee.

Widely used in several applications, TCP is the today standard for network communication protocols. A TCP connection is a three-way handshake process, then the data is transmitted, followed by a four-way handshake process that is needed to close the connection (Kanmai, 2020). Kurose and Ross (2012) states that the design of this protocol guarantees a reliable data transfer service, ensuring the users that the data arrived to its destination without any data corruption. However there are some security problems in the design. In the establishing

1 More information at: https://www.openstreetmap.org/
2 More information at: https://www.mapbox.com/
3 More information at: https://developers.google.com/maps
4 More information at: https://www.bingmapsportal.com/

of the connection, a hacker could send a Reset (RST) packet, "Reset (RST) packet is used by a TCP sender to indicate that it will neither accept nor receive more data" (Kryszczuk and Richiardi, 2011), followed by an Synchronize (SYN) Packet, used to establish the first connection, making the recipient believe that the sender is making a new connection, and then proceed to send fake data. This can be avoided by filtering the senders Internet Protocol (IP) address, decoding or setting timeouts (Kanmai, 2020). Figure 1 shows an example of a TCP packet.

Source Port		Destination Port	
Sequence Number			
Acknowledgment Number			
Data Offset	Reserved	Flag	Window
Checksum		Urgent Pointer	
Options		Padding	
Data			

Figure 1: TCP Packet

Instead of being a connection oriented protocol, UDP is a transaction oriented protocol. The goal of this protocol is to offer a service of data transfer with a minimum of protocol mechanism, leaving a no-guarantee of uncorrupted data or even delivering to destination (Postel, 1980). The UDP packet has a section called Checksum, as confirmed in Figure 2, this is a 16-bit section that contains some information as the destination and source address. However a hacker can eavesdrop an UDP packet and create a new one by pretending that the source is the same as the eavesdropped packet. This technique is called spoofing and as success since UDP don't offer a way to check if the source is the source it seams to be (Olsen, 2003).

Source Port	Destination Port
Length	Checksum
Data	

Figure 2: UDP Packet

The protocols above discussed are insert in the Transport Layer of the International Organization for Standardization/Open Systems Interconnection (ISO/OSI) reference model. Zigbee introduces itself as the only complete IoT solution [5], offering a solution that goes all the way from the hardware to software. Built on top of the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 standard, Zigbee uses its proprietary way to

[5] https://zigbeealliance.org/solution/zigbee/

communicate, since it does not have IP address, but a uses Media Access Control (MAC) address. Zigbee was created for IoT uses-cases, being a low-cost and low-power-consumption solution (Alliance, 2015), what is a big concern in a world of battery power devices. It uses a symmetry cipher in order to maintain the confidentiality and integrity of the message, also deploys an Advanced Encryption Standard (AES) algorithm in order to guarantee the message source authentication. It also offers a message integrity check (Khanji et al., 2019).

2.3.2 *Security Protocols*

With the emerging of Web for everybody, there were many communication between client and server that contained sensitive data, e.g. usernames, passwords and credit-card numbers. For securing this sensitive data there were developed security protocols e.g. Internet Security Protocol (IPsec), Microsoft's Private Communications Technology (PCT) and Secure Sockets Layer/Transport Layer Security (SSL/TLS). The most adopted security protocol is TLS (Turner, 2014). TLS emerge as the sucessor of SSL 3.0, this protocol offers protection to eavesdropping, tampering and message forgery. It's divided in two layers: TLS Record Protocol - built on top of TCP, offers symmetric cryptography and message integrity check, conferring a private and reliable connection; TLS Handshake Protocol - it is a three-way handshake negotiation, using public-key, cryptography or asymmetric to confirm peer's identity, it also negotiates a shared secret that after will be used in TLS Record Protocol (Dierks and Allen, 1999).

2.3.3 *Clock Synchronization*

With the emerge of distributed applications, clock synchronization is a fundamental topic of research that will limit errors derived from inaccurate clocks. Every node have its own memory, Operating System (OS) and clock, resources that aren't shared, this require a method that can synchronize every node (Latha and Shashidhara, 2010). In an automated driving environment it is major that the clock from the vehicles are always synchronized, this can prevent collisions that could result from the clocks being inaccurate. E.g. If we have vehicle A going at a certain speed it will arrive at a X location at Y hours, but if a vehicle B, with an inaccurate clock, knows the vehicle A speed and path it will miss-calculate the Y hours, leading to a possible impact with both vehicles. For synchronizing clocks there are several methods e.g. through Global Position System (GPS) - although it offered high precision clock, the costs of hardware to accomplish are expensive; Christian method - this methods is a centralized server that provide time, answering to node's requests; Berkeley algorithm - it is also a centralized method, however the server does not provide the correct time, instead it collects all the times from the client, make an average and update the nodes with the new value (Latha and Shashidhara, 2010); Network Time Protocol (NTP) - this method is a protocol based in UDP, that as discussed in 2.2.1 offers a connection-less transport mechanism, NTP uses distributed clients and servers to offer a nanoseconds precision (Mills, 1992).

2.4 SUMMARY

The origins of the Automated Guided Vehicles and some current projects of implementation or simulation of them were studied during this chapter. In this way it is possible to form a better expectation of the benefits of AGVs. In order to complement the study of the subject, different technologies were approached and researched, ranging from communication protocols, security protocols, clock synchronization, and environment recognition methodologies, which together will help to formalize the proposed solution presented in the next chapter.

3 PROPOSED APPROACH

The goal of this Master's Project is to develop a solution capable of simulating automated guided vehicles in a controlled environment. Before designing an architecture of the solution, it is need to study and divide the problem, Automated Driving, into sub-problems. For that purpose, it was created a conceptual map, that will be discussed in the following sections to support decisions made do design the system architecture. As stated in Figure 3, automated driving can be divided in five different objects of study: **Environment**, **Map**, **Vehicle**, **Controller**, **Monitoring**.

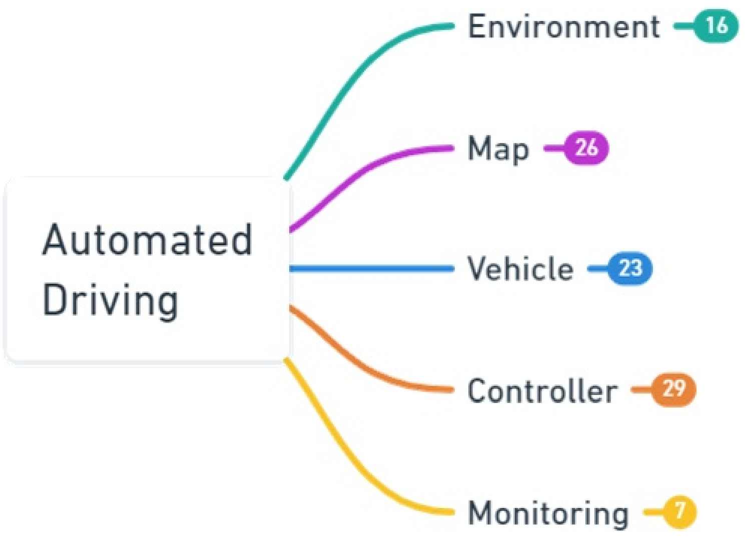


Figure 3: Initial conceptual map state.

3.1 ENVIRONMENT

The environment is where the vehicles are going to operate, being it health-care facilities, warehouses or factories. It is composed by objects, vehicles, walls, persons and everything that can be present in it. A way for the solution to know the environment is needed, therefore we can use sensors (e.g. cameras and Light Detection and Ranging (LiDAR), that can give us a recognition of the environment, locating objects and tracking if there is movement). This way its possible to classify the objects as moving or static objects, as we can see in Figure 4. Moving objects can be either vehicles, pedestrians or other non-identified objects that can move trough the environment.

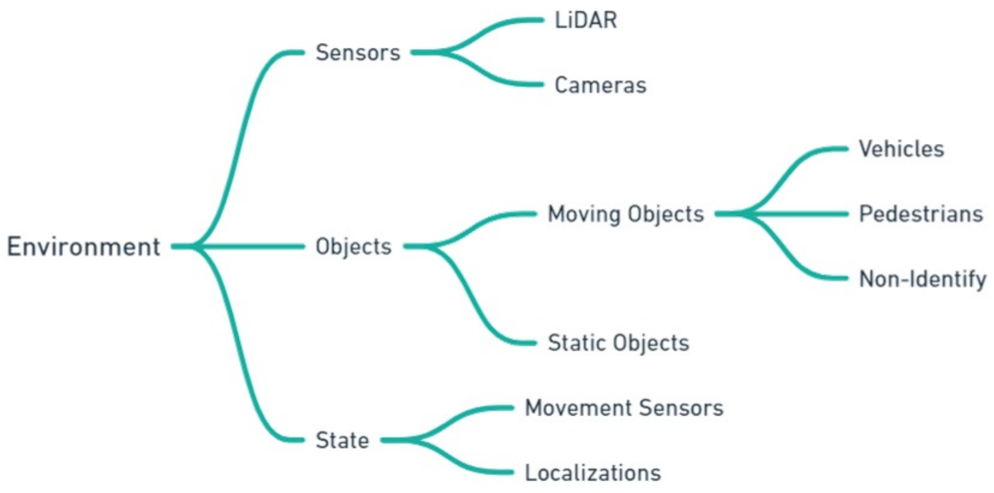


Figure 4: Environment bracket of the conceptual map.

3.2 MAP

As stated in Section 2.2, a map is a representation of an environment, and since it is a representation, it needs to store and display data about the localization of vehicles and objects. Therefore as stated in Figure 5, a map needs to have coordinates, that way we can assign a way to identify objects positions. When building a map, we can either use a base map, as in a file with a matrix, for example. Also there are several providers of maps and GPS, in Section 2.2 some were discussed, however in indoor situations the use of providers can limit the use of the solution, therefore environment recognition trough out sensors, as discussed in Section 3.1, is a more reliable solution.

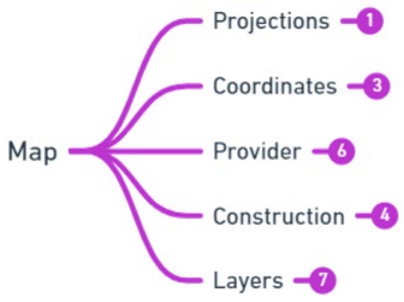


Figure 5: Map bracket of the conceptual map.

In Figure 6 is possible to see in detail layers that a map can settle. Using layers makes the information easier to be analysed by the solution. E.g. it can have a layer just for localization of vehicle parking, or specific zones that vehicles can move, and how the traffic should behave, following rules. Other layers could be used to display the localization from static objects. Last but not least, maps should have a layer of the infrastructure of the environment, representing walls, columns and sometimes doors or pass-troughs.

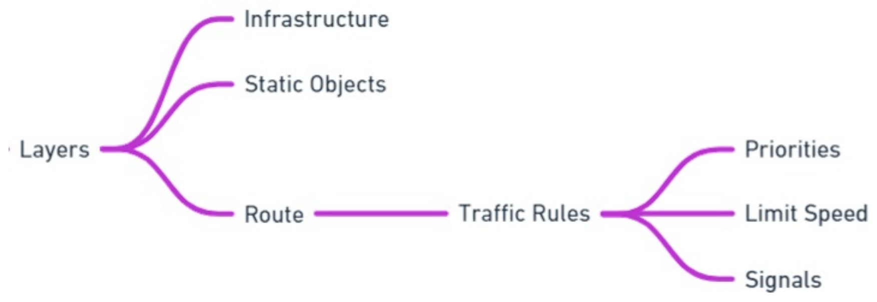


Figure 6: Layer bracket of the conceptual map.

3.3 VEHICLE

There are many aspects that are needed to be considered in a vehicle. This way, as seen in Figure 7, there are some characteristics that need to be known e.g. it's dimensions, in which directions it can move and top speed.

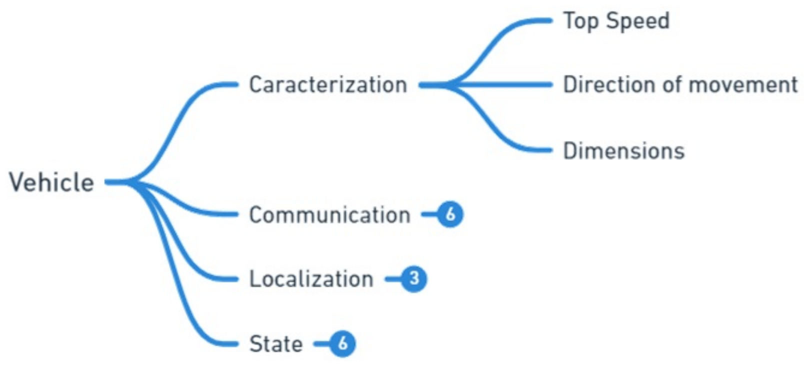


Figure 7: Vehicle bracket of the conceptual map with characterization bracket open.

Later, in Section 3.4 we will study the controller, this will be the brains of the operation. Therefore a vehicle needs to have modules so it can communicate with the controller or even other vehicles. This module will be used to receive orders and transmit the state of the vehicle or localization. In Figure 8 there are some technology examples that can be used to achieve this functionality.

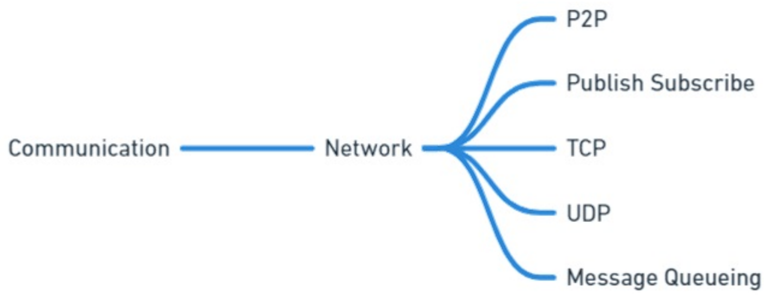


Figure 8: Communication bracket of vehicle bracket zoomed in.

The vehicle localization can be obtained in different ways and will differ based in the choice of technology used to map the environment. In Figure 9 there are the two main examples: the use of GPS localization, using external providers for the maps, or recurring to sensors. Also in Figure 9 there are some examples of information related to state of the vehicle, this information (e.g speed, steering angle and acceleration) can be use to calculate the path of the vehicle and it's future localization, this way adjusting and communicating to other vehicles there is room to avoid possible collisions.

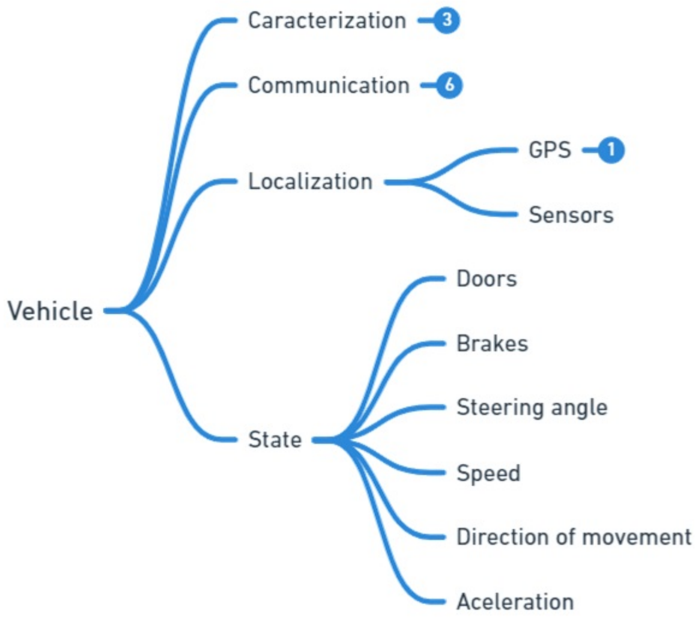


Figure 9: Localization and State bracket of vehicle bracket open.

3.4 CONTROLLER

The controller is the brain of the operations, with the input of the map and vehicle data, it gives the vehicle instructions with the path to it's destination. Having such an important role in the project, it has a lot of requirements. In Figure 10 there are approached overall requirements, (e.g. communication - as previously stated in Section 3.3), the vehicle needs to communicate to the controller, so there is also a need in the controller to have a way to receive and transmit data; Data persistence, clock synchronize - to guarantee that all vehicles have the same time; Safety, Logistics, Conflict Management and Learning capabilities.

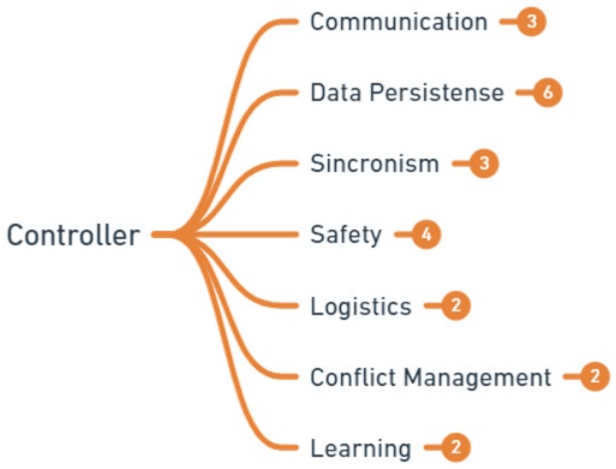


Figure 10: Controller bracket of the conceptual map.

One of the most critical aspects of all applications is its safety. Developers need to implement safety protocols or certificates, as mentioned in Figure 11, so that is service can be reliable. With the use of certificates and safety protocols, the unauthorized control of the vehicles is avoided, guaranteeing that the application can be trusted and have protections to attackers.

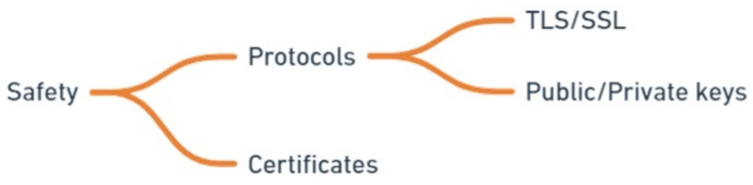


Figure 11: Safety bracket of controller bracket open.

Figure 12 shows both logistics and conflict management bracket zoomed in. They appear in the same figure since one complements the order. Since the vehicle is moved through input given by the controller, it needs to have a module to calculate paths and parking when vehicles are not used. This module will work together with the conflict management to assure that the vehicle will park in a empty spot and in is course respects priorities and intersections with other vehicles.

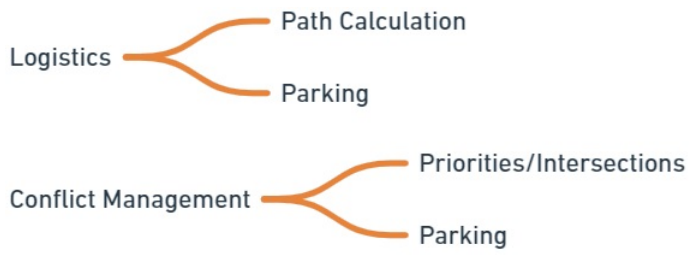


Figure 12: Logistics and Conflict Management bracket of controller bracket open.

In a project of this area it is important that the system is constantly evolving. As visible in Figure 13, this component have the responsibilities of learning and optimization of patch calculation and impact prediction, turning the controller a system that is always up-to-date.

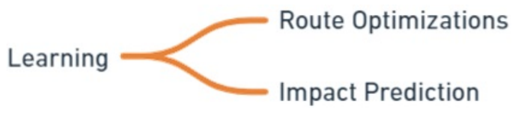


Figure 13: Learning bracket of controller bracket open.

3.5 MONITORING

Another very important aspect, is monitoring. Although the controller must implement an impact prediction system, there must be an external monitoring system, that alerts the systems administrator for possible impacts that could have happen, or if there is any problem regarding priorities in intersections. In addition, it must monitor the status of the sensors used so that if any sensor has an error the controller immediately stops the circulation of vehicles and, again, alert the system's administrator for the situation that is happening. Figure 14 is shown the bracket of the conceptual map for monitoring.

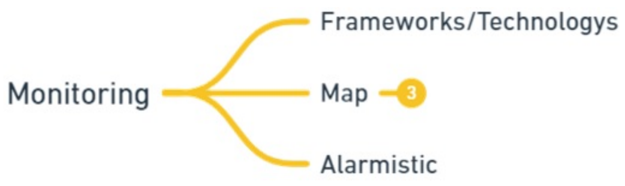


Figure 14: Monitoring bracket of the conceptual map.

3.6 Conceptual Architecture Diagram

Figure 15 shows a Conceptual Architecture Diagram of the system, it makes an approach of an implementation based in micro-services model. Derived from the conceptual map presented earlier, this diagram presents several services that together culminate in a possible solution to the initial problem.

At the top left box there is represented the environment, the environment connects to the Sensors which will make the environmental recognition, resulting in a Map in the bottom left box.

The Map acts as an input for the Controller. Being the central piece of the solution, it is the largest box, containing several services: logistics - to calculate path and where to park; Conflict Management - to deal with conflicts that could lead do impacts or traffic jam; Data Persistence - there are data that needs to be stored for future use; Learning - in order to optimize paths and impact prediction; and a communication service that make the controller capable of receiving data and sharing data with cars, front-end and other connected services.

Monitoring service is connected to Sensors, Map, Controller and Front-end. It constantly verifies the system status, verifying if all the sensors are working properly, if there hasn't been any impact or conflict. If any of the before happens, it should command the controller to stop any moving vehicle and alert the system administrator for the succeed through the front-end.

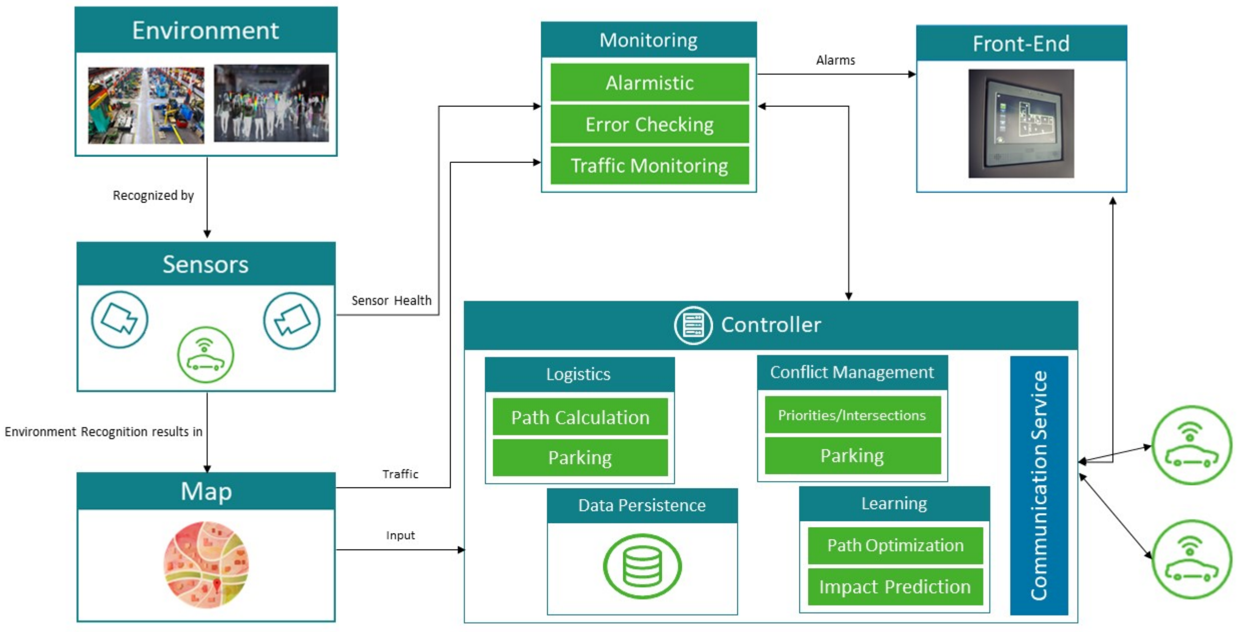


Figure 15: Conceptual Architecture Diagram.

3.7 SUMMARY

Throughout the Chapter the concept of Automated Driving was broken down into smaller concepts using a conceptual map. This way it was possible to conclude that there are five crucial points to fully achieve the concept, they are: Environment, Map, Vehicle, Controller and Monitoring. The goal is to create a simulator of Automatically Guided Vehicles in Controlled Environments. Being an controlled environment it is necessary that the system is familiar with the environment. Using cameras or providers such as OSM and Google it is possible to have some sort of environment recognition. This way it's possible to build a map of the environment that will be stored in the system. The vehicles will not be autonomous but automated, so they need a controller that has access to the map and communicates which trajectory they should follow without crashing into other vehicles or obstacles and without leaving the lane. So it is very important to monitor. Through the analysis of these concepts it was architected the diagram available in Figure 15.

4 PRE-DEVELOPMENT CONSIDERATIONS

After the pre-study carried out in Chapter 3 it is necessary to rethink the structure, take certain points for granted, and which modules will not be strictly necessary for the development of the solution. In this Chapter these modules will be discussed and studied in order to explain to the reader why they are not necessary and what alternatives will be used in some cases.

4.1 ENVIRONMENT RECOGNITION AND MAP

As written in Section 2.2, environment recognition can be achieve in multiple ways, being with RGB cameras, sensors and many other. However it also is a thorough process, evolving a lot of research, study and development. That said, it is safe to assume that environment recognition would make an interesting dissertation topic in itself, due to it's complexity. To replace this component, it was decided to follow a route where static maps are used. After an extensive research for map providers, OSM was the chosen provider to extract the different layers that composes and elaborate a map, being an free open source provider. Although the extracted map contains a lot of information, much of it is not necessary for the purpose of the solution. Thus, it was decided to create a micro-service, afterwards called AGV-MAP that receives the extracted OSM file and filters the information considered relevant. Listing 4.1 shows a part of the OSM file of the extracted map. Then, in figure 16, we see a graphical representation of this extracted map using a tool called Java OpenStreetMap Editor (JOSM)[1] without any filtering or processing done.

1 https://wiki.openstreetmap.org/wiki/JOSM

```xml
<?xml version="1.0" encoding="UTF-8"?>
        <osm version="0.6" generator="CGImap 0.8.6 (1601248 spike-07.
    openstreetmap.org)"
        copyright="OpenStreetMap and contributors" attribution=
        "http://www.openstreetmap.org/copyright"
        license="http://opendatacommons.org/licenses/odbl/1-0/">
        <bounds minlat="40.9199900" minlon="-8.5673900" maxlat="40.9229700"
        maxlon="-8.5639000"/>
        <node id="665907895" visible="true" version="4" changeset="113178563"
        timestamp="2021-10-31T03:00:17Z" user="SilverKnight"
        uid="1692086" lat="40.9240155" lon="-8.5650241"/>
        <node id="665907916" visible="true" version="4" changeset="113178563"
        timestamp="2021-10-31T03:00:17Z" user="SilverKnight"
        uid="1692086" lat="40.9240097" lon="-8.5647823"/>
        <node id="665908211" visible="true" version="5" changeset="31198168"
        timestamp="2015-05-16T10:51:04Z" user="boot_killer"
        uid="1428742" lat="40.9224294" lon="-8.5634625"/>
        <node id="665908239" visible="true" version="5" changeset="31198168"
        timestamp="2015-05-16T10:51:04Z" user="boot_killer"
        uid="1428742" lat="40.9225198" lon="-8.5635569"/>
```

Listing 4.1: Excerpt from OSM file

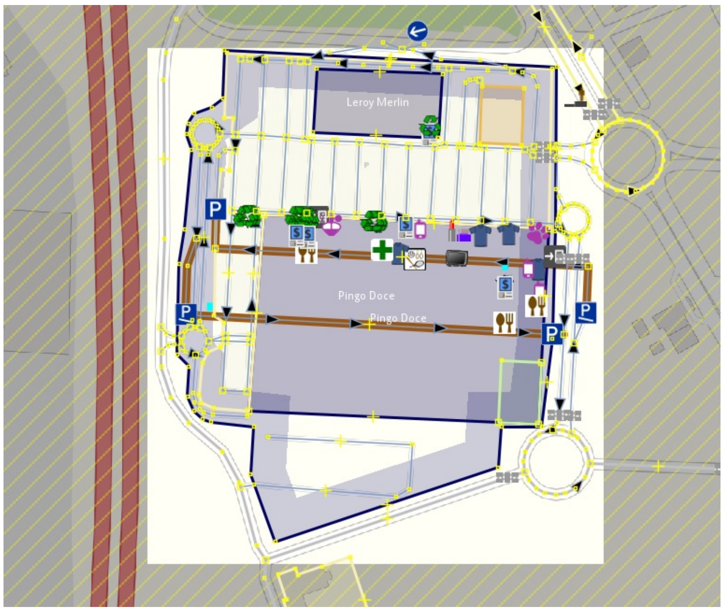


Figure 16: OSM extracted Map Without processing or filtering.

As we can see in Listing 4.1 the map extracted from OSM is an Extensible Markup Language (XML) file, however it was preferred to increase the human readability of the file and decrease its weight. For this, it was chosen to transform the extracted file into a () file, but not just any JSON file, but GeoJSON. Thus, in addition to the objectives previously mentioned, we ensured compatibility with the technology used for data persistence. A GeoJSON file is, as its name indicates, based on the JSON structure. It defines a proper structure to represent the map through objects that represents geographic features, their properties, and their spatial extents (Butler et al., 2016). Listing 4.2 shows an example of the structure of a GeoJSON file.

```
{ "type": "FeatureCollection",
  "features":
    [
        {
            "type": "Feature",
            "id": "way/53095200",
            "properties": {
                "uid": "12612838",
                "highway": "residential",
                "lane_markings": "no",
                "lit": "yes",
                "maxspeed:type": "PT:urban",
                "name": "Rua Germano da Silva Santos",
                "surface": "asphalt",
                "id": "way/53095200"
            },
            "geometry": {
                "type": "LineString",
                "coordinates": [
                    [
                        -8.5664846,
                        40.9200122
                    ]
                ]
            }
        }
    ]
}
```

Listing 4.2: GeoJSON Example

4.2 MONITORING

The monitoring of vehicles and the environment is quite important. However, it is also complex and depends on many factors. In the conceptual architecture diagram in Figure 15 it is possible to see that the sensors/cameras are connected to the monitoring module. However, as stated in Section 4.1 sensors will not be used or anything like that, so it is necessary to rethink the monitoring module. Further in Chapter 5 we will see that traffic monitoring will be done by the controller, and that there will be an independent alarm service for registering possible errors.

4.3 Learning Capabilities

The learning capabilities of a system are quite positive, especially in an Automated Vehicle system. It allows optimising routes, better preventing collisions, among others. However, implementing learning capabilities is something that is quite complex, involving a lot of study and also ends up being a little bit outside the intended with the development of the work. Since we want to create a simulator of the controlled environment, where the vehicles can operate, it is not necessary for the current case to follow that route. However, it is important to mention that for a system that would guide the vehicles, it would be an added value.

4.4 Logistics and Conflict Management

As with the module covered in Section 4.3, this module turns out to be a bit out of scope of what is intended for this paper. However, like the module Learning, Logistics and Conflict Management is something that would be very important for future development that would pass the goal of moving the vehicles in an intelligent and schematized way. To this end, it was necessary to define traffic rules that would be imposed by this module, such as, for example, at an intersection, it was necessary for Conflict Management to intervene so that vehicles could pass through the intersection without colliding.

4.5 Conceptual Diagram Update

After the considerations it is necessary to go back and update the Conceptual Architecture Diagram in order to outline which modules will need to be developed for the realisation of the solution. In order to think about development, the tools to be used are also considered. This is what the Figure 17 shows, an revised version of the Figure 15. It can be seen that there is already a hint of which programming languages are planned to be used, such as some text formats. In addition, are also represented some modules in lighter colour, this happens because they will not be developed

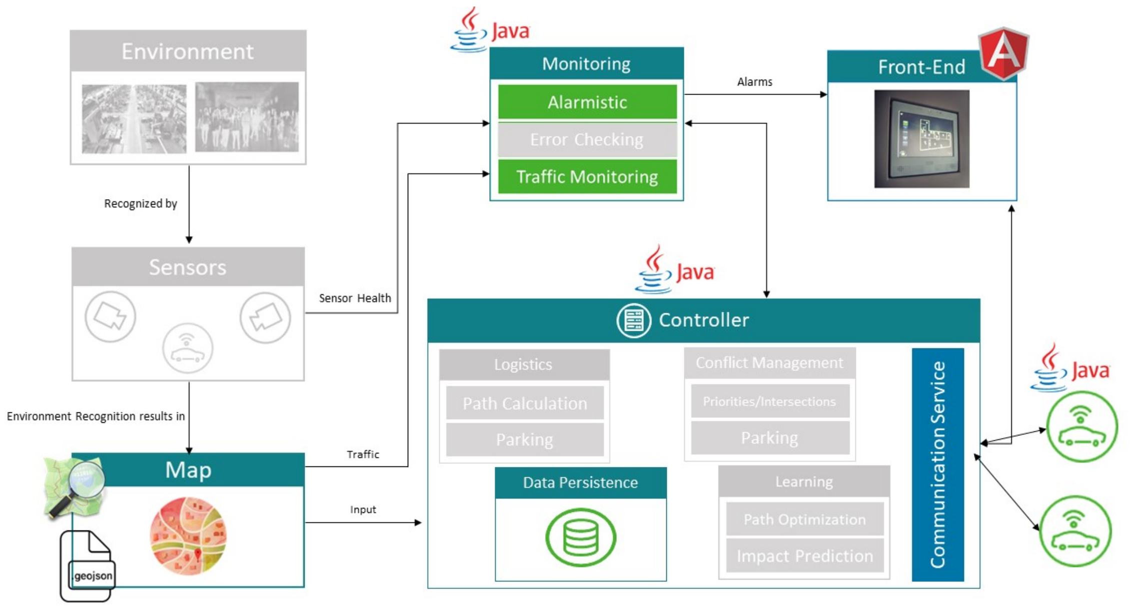


Figure 17: Revised Conceptual Diagram

4.6 SUMMARY

Upon the first iteration of the Conceptual Architecture Diagram it was necessary to pause and observe it to understand which modules would be out of the scope of what is necessary and decide those that will not be developed on account of the time available for the development of the Master's Work. Then, some decisions were taken. For the environment recognition it was decided to use OpenStreetMap as the environment provider, so it was possible to fill the absence of recognition sensors and create layers that resulted in the environment map. The learning capabilities, as well as logistics management or path calculation, despite being very interesting topics, are also very time consuming and end up being out of the scope of what is necessary for the development of the solution, being very interesting material for future work. After these considerations an updated Conceptual Architecture Diagram was drawn, as can be seen in Figure 17.

5

DEVELOPMENT

From the Conceptual Architecture Diagram and even before the conceptual map that was analysed and explained in the previous chapter, we can divide the solution into modules. There are several ways to approach and develop an application, however, in order to maintain modularity, independence between functions and greater ease of expansion, the development model will be based on micro-services. We will then need a service that stores the environment characterization, **AGV-MAP**; a service that stores the alerts, **AGV-ALR**; a service that will store the vehicles and routes/sessions, **AGV-VMS**; a service that will control the vehicles, **AGV-CTL**; a service for the visualization, a front-end of the solution, **AGV-FE**; and by the vehicle itself, **AGV-VEH**. In Figure 18, the relationships and interactions between services are summarised. It should be noted that the micro-services within the boxes, despite not having a connection represented by an arrow, are grouped within a box because they can communicate with each other. The **AGV-CTL**, being a more complete micro-service, is represented in a box by itself, however, it also establishes a connection with the other micro-services, so it is possible to see a small connection between boxes.

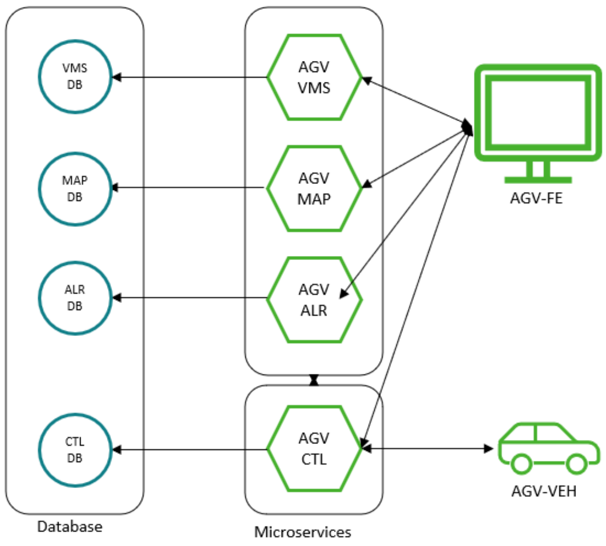


Figure 18: Micro-service Architecture of the solution

For the development of the solution it is necessary to define tools and programming languages to be used. The choice has to be thought out according to the model that will be followed. Due to the large online support and documentation , added to the student's familiarity with this language, Java was chosen for the development of the micro-services. With the help of Spring, a framework focused on speed, simplicity and productivity, it will be less difficult to create Representational State Transfer (REST) APIs and the project itself. However, Java is not the best option for creating web applications, for this purpose there are several options for front-end development, from Vue, React, to Node JavaScript. Although, Angular was chosen to develop the front-end, due to the familiarity of those involved in the development of the solution, as well as the ease of use.

5.1 TECHNOLOGIES USED

Chosen the programming languages for the development of the solution, some tools are necessary to help completing the functionality of the solution. In this section the technologies used will be addressed.

5.1.1 *Spring*

Spring is an open source framework that accelerates the development process in java, focused on speed, simplicity and productivity. Within this framework there are several tools that proved to be useful for the development of the final solution. These tools are: Spring Boot, Spring Data Java Persistence API (JPA) and Spring Integration.

As decided earlier, the solution will be developed in a micro-services view, developing several micro-services that in synchronization conjugate into a whole, without creating dependencies on each other. This way we can treat a micro-service as a piece of the solution and it can be changed without needing to affect the others, or with minor modifications to the others. For this purpose, we chose to use the previously mentioned tool, **Spring Boot**, which allows to rapidly create projects. It contains all the predefined configurations for the creation of a REST-API server, allowing the programmer to start development right away.

As in almost every project developed today there is information that is necessary and sometimes just interesting to keep. Thus it is necessary to use a system for data persistence to be fulfilled. Databases are used for this. To simplify their use, the Spring framework has another tool that we will use, **Spring Data JPA**. This tool reduces the effort required to execute queries or simply the effort required to connect to the database. As a developer it's only needed to define the entities that will be store, usually this entities are defined as Data Access Object (DAO) in the project, and later an Repository interface class that extends an repository programming model that have all the Create Read Update Delete (CRUD) general methods, saving a lot of time to the developer. It is in this class that the programmer will define methods, if needed, that follow a naming pattern that allow a parser, through natural language analysis, to understand what is intended and fetch the information from the database [1].

In the next section the communication protocols used will be discussed. However, still within the Spring framework, there is a tool that will allow us to implement TCP communication, since communication via Hypertext Transfer Protocol is already handled by Spring Boot. So, this tool that will allow us to implement communication in a Spring project is **Spring Integration**, with the use Integration we can create integrationflows and gateways that will create and establish the TCP connections between vehicles and controller and send the packets. Later in Section 5.7 and 5.8 the use and purpose of Integration Flows and gateways will be more explored and explained.

5.1.2 *TCP and HTTP*

As already mentioned, the solution envisaged consists of a set of micro-services. For the interconnection of these micro-services it is necessary to have communication between all of them, for this the HTTP protocol is used, the micro-services will provide endpoints of various types that receive these requests to exchange information. However this is not the only communication protocol that will be used. Although the HTTP protocol is fast, for the communication between vehicle and controller, something is needed to ensure that the communication is really happening and that the information has reached the destination. For this the TCP protocol is used. This protocol allied to the tool mentioned next in Section 5.1.3, Google Protocol Buffers, will be the solution to exchange information about the vehicle status and possible trajectory updates. On the other hand before establishing TCP communication between controller and vehicle, HTTP will also be used for registration and first contact between both. Later on, in Section 5.10 these nuances will be explained.

5.1.3 *Google Protocol Buffers*

As mentioned in the previous section, the TCP protocol will be used for communication between vehicles and the controller. Thus, it was necessary to define a standard that would structure the information flowing between them. This way Protocol Buffers were chosen, which are mechanisms for serialization of structured, platform-neutral and language-neutral information. For this purpose the **Google Protocol Buffer** [2] was chosen. Designed by Google, it is compatible with several languages, be it Java, Python, C++ and others.

1 https://spring.io/blog/2011/02/10/getting-started-with-spring-data-jpa/
2 https://developers.google.com/protocol-buffers

Therefore, keeping the line of reasoning of micro-services, in which services are independent, different languages can be used for different purposes.

5.1.4 *Angular*

For a more graphic visualization of the solution it was necessary to develop a front-end design. Nowadays, in order to increase the compatibility of applications between devices, tools are used that allow us to create web-based applications, that is, applications that run in the browser. Thus, whether it's a fixed computer, a laptop, tablet or cell phone of any operating system, it is possible to access the application. For this purpose we used the **Angular** framework, which allows us to quickly and easily create web projects. This development tool is built on TypeScript [3], and allows us to develop scalable projects compatible with a variety of libraries

5.1.5 *H2 Database Engine*

As mentioned before, data persistence is very important. Whether it is for the need to store certain information, such as vehicle characteristics, or the interest in storing data also, for example, about the position of the vehicles so that in the future it will be possible to review their route. This way it is necessary to use a database. For this purpose, and thinking about the practicality and speed of configuration, the **H2 Database Engine** [4] was used. An open-source, Structured Query Language (SQL), fast database that has a web-based console. This database is very easy to configure, allowing for different uses. It can be restarted each time the project is run, usually for testing reasons. It can also keep the information persistent and store it in an encrypted form.

5.1.6 *Mapbox GL JS*

One of the components of our solution is a web application that allows us to visualise the environment in which the vehicles are circulating, in Section 5.9 section this component will be addressed. To achieve this visualization we used the Mapbox GL JS [5] tool which is a client-side library developed in JavaScript that allows us to build maps in web pages. This tool allows us to add the layers that characterize environments, add markers and pop-ups to maps, as well as define custom styles to maps. As we can see in the section above this tool

3 TypeScript is a strongly typed programming language that builds on JavaScript, more in: https://www.typescriptlang.org/

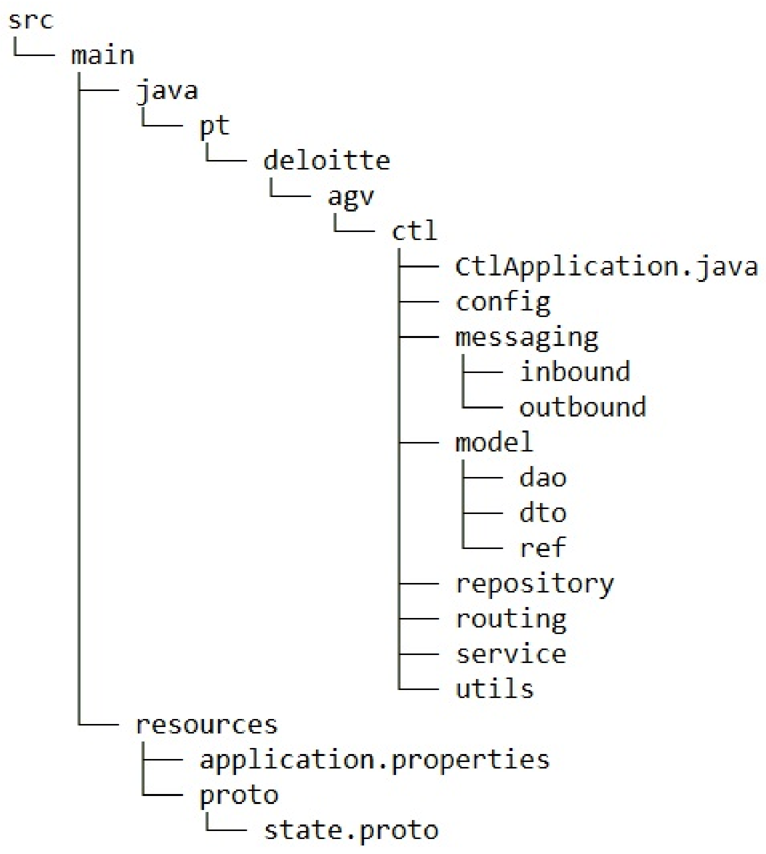
4 https://www.h2database.com/html/history.html

5 https://docs.mapbox.com/mapbox-gl-js/guides/

proves to be very useful and facilitating in this process of viewing in real time the map and the vehicles moving.

5.2 General Project Structure and Conventions

In order to achieve homogeneity and organization of projects, some rules for structuring and naming of projects were defined, which all will follow, except for AGV-FE since it uses a different development platform. However, even this project maintains some similarities. Figure 19 is the structure of the AGV-Controller, that is the project with an higher number of packages in comparison. Not all the packages are used in all projects.



```
src
└── main
    ├── java
    │   └── pt
    │       └── deloitte
    │           └── agv
    │               └── ctl
    │                   ├── CtlApplication.java
    │                   ├── config
    │                   ├── messaging
    │                   │   ├── inbound
    │                   │   └── outbound
    │                   ├── model
    │                   │   ├── dao
    │                   │   ├── dto
    │                   │   └── ref
    │                   ├── repository
    │                   ├── routing
    │                   ├── service
    │                   └── utils
    └── resources
        ├── application.properties
        └── proto
            └── state.proto
```

Figure 19: AGV-CTL summarised structure

- **config** - This package holds classes needed for extra configuration at project startup, or classes that contain methods that react when an exception occurs.
- **messaging** - This package stores the necessary classes for configuring TCP services, storing the gateways and classes for creating Integrations Flows, which are required for communication between CTL and VEH.

- **model** - Consisting of two subpackages Data Access Object and Data Transfer Object (DTO), this package stores the model classes for creating, manipulating and storing objects. For example, the classes inside the DAO subpackage correspond to the entries in the tables of a database.
- **repository** - As said in Section 5.1.1, for the simplification of the interaction with the database, a Spring tool was used: Spring Data . Thus this package stores the classes that contain the CRUD methods for the different database tables.
- **routing** - This package stores the controllers that are classes contain the endpoints. So when an HTTP request comes in, it is filtered by the endpoints and when it matches it executes some method defined by the services classes.
- **service** - In the previous point we have already given you a hint of what this package stores. Here are the classes with the methods that end up being the features. They can be methods that just insert new entries in the database, as more complex as for example assign a destination to the vehicle.
- **utils** - This class stores more miscellaneous classes, for example, widely used in almost all projects is the Mapper class, which contains methods that receives objects of type DTO and transforms them into objects of type DAO for insertion into the database.

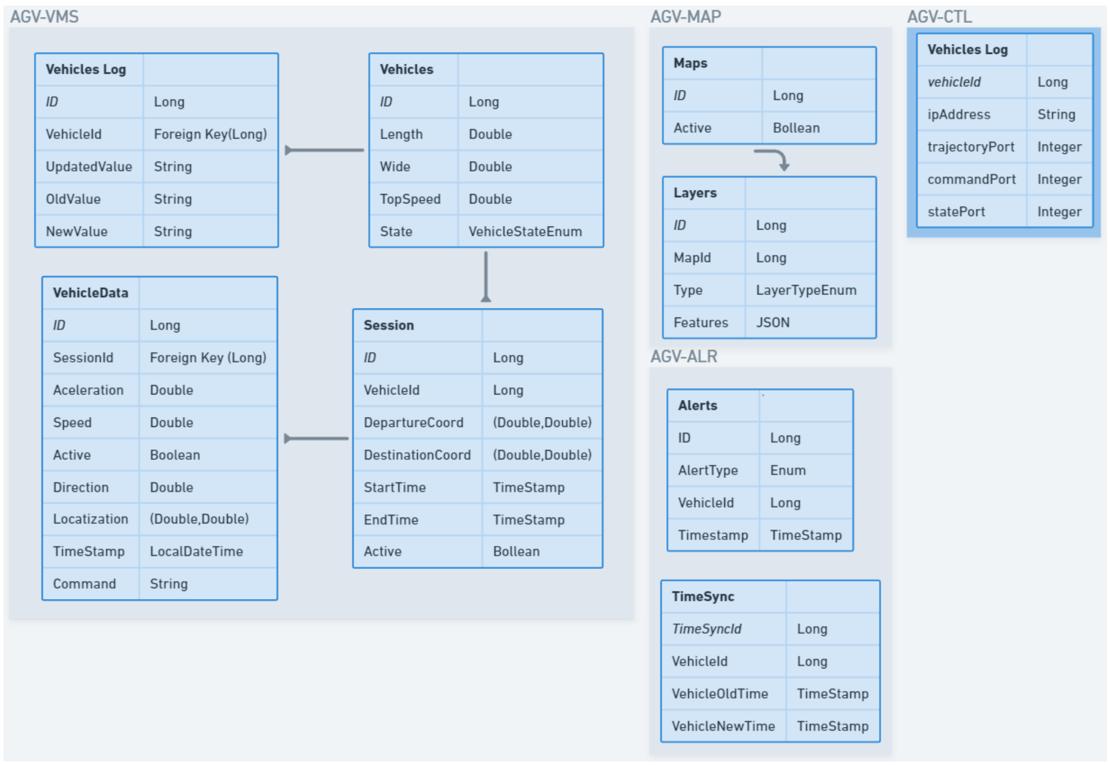
However, there are some that are not present in all projects because no functionalities or classes that justify their use were required.

Besides the structure, the names also follow existing conventions, classes must follow a Camel Case pattern, methods follow a Mixed Case pattern, always starting with a lowercase letter and must be verbs. The variables must also follow the Mixed Case pattern. In the case of Lower Case packages, all letters are lowercase (Microsystems, 1997).

Also on the endpoints some conventions have been established. Endpoints must always be Lower Case and words must always be separated by a slash. However, if strictly necessary they can be separated by underscore.

5.3 LOGICAL DATA MODEL

As with any software used these days, there is data that needs to be persistent and accessible at all times. Several of the projects that culminate in this solution, are CRUD projects or projects that need a support for data. To achieve data persistence, to be able to access data quickly and at any time, one of the solutions to use will be a database. As mentioned in Section 5.1.5 a Structured Query Language database was used, this language was developed for storing, manipulating and retrieving data from relational databases. Thus it is necessary to organise and structure the data that we want to be persistent. In Figure 20 a logical data model is presented, this logical model represents tables that will contain the information that is needed to store.



Vehicles Log	
ID	Long
VehicleId	Foreign Key(Long)
UpdatedValue	String
OldValue	String
NewValue	String

Vehicles	
ID	Long
Length	Double
Wide	Double
TopSpeed	Double
State	VehicleStateEnum

VehicleData	
ID	Long
SessionId	Foreign Key (Long)
Aceleration	Double
Speed	Double
Active	Boolean
Direction	Double
Locatization	(Double,Double)
TimeStamp	LocalDateTime
Command	String

Session	
ID	Long
VehicleId	Long
DepartureCoord	(Double,Double)
DestinationCoord	(Double,Double)
StartTime	TimeStamp
EndTime	TimeStamp
Active	Bollean

Maps	
ID	Long
Active	Bollean

Layers	
ID	Long
MapId	Long
Type	LayerTypeEnum
Features	JSON

Vehicles Log	
vehicleId	Long
ipAddress	String
trajectoryPort	Integer
commandPort	Integer
statePort	Integer

Alerts	
ID	Long
AlertType	Enum
VehicleId	Long
Timestamp	TimeStamp

TimeSync	
TimeSyncId	Long
VehicleId	Long
VehicleOldTime	TimeStamp
VehicleNewTime	TimeStamp

Figure 20: Logical Data Model

It is possible to see that the database is represented by sections, this is because as shown in the architecture presented in Figure 18 each service has its own database. The sections referring to AGV-VMS, AGV-MAP and AGV-ALR are part of the main data model. The AGV-CTL section is only used to maintain the persistence of certain data stored in the service cache, improving the interaction between vehicle and controller. Later in Section 5.7.1 we will deepen this issue of cache, after in Section 5.10 we will deepen the importance of cache and persistence for the interaction vehicle-controller.

The AGV-VMS section consists of 4 tables: Vehicle, Vehicle Log, Sessions and Vehicle Data. The first table stores the vehicle data, either physical data such as length and width, or limitations such as maximum speed, as well as the vehicle status. Next, the Vehicle Log stores any changes that may have occurred to the parameters mentioned above. Something that will also be important to store for analysis of trajectories and destinations is the Sessions, this table stores data such as the coordinates of departure, destination and timestamps that each occurred. Associated with these sessions are the Vehicle Data, this table stores at each timesync set, in this case will be 1 second, the state of the vehicle at that instant of time, as its location.

The AGV-MAP section consists of two tables, the Maps themselves, as of the layers associated with the maps. Each row of the Layers table, constitutes a layer of a map, that layer has its type associated, to which map it belongs and a string of GeoJSON type that is the text type chosen to represent the environment as stated in Section 4.1

The AGV-ALR section consists of two unrelated tables. This service only stores date and provides a service so that, if necessary, in the future, it can synchronize the clocks between services and vehicles, something that at this stage of the project will not be necessary, due to the use of the same machine for the tests and development of the solution. The information stored are alerts, alerts generated by the system when the vehicle disconnects, leaves the road or collides with another vehicle or obstacle.

5.4 AGV-MAP

As already mentioned, the environment is quite important in such a problem. Thus maps are needed to represent them. AGV-MAP is a REST project, developed with the purpose of storing and providing the various maps that we may want to use.

As mentioned in Section 4.1 it was necessary to develop an algorithm that was able to read and filter a map extracted from the OpenStreetMap (OSM). It was here that this algorithm was developed. AGV-MAP contains an endpoint that receives HTTP POST requests whose body contains the XML resulting from the OSM extraction. Before starting to work on the algorithm itself, some steps were necessary first. Firstly, to simplify the processing of the received XML string, model classes were created that describe the structuring of the string. So instead of reading it line by line, which would be inefficient, we can use JAXBContext to perform XML unmarshaling, which consists in converting it into a java object. This java object contains several methods that will facilitate the filtering. For example we can check all the nodes of a way without having to read the whole file.

After converting the XML file, we proceed to the filtering of two layers, Road and Buildings. These are the most relevant layers for the case, although it is quite complete the OSM information about parking is tiny, not being possible to create a layer through the received XML. Generally speaking, both layers are filtered in the same way, but with minor nuances. Both buildings and roads are represented by ways containing associated nodes.

So the first step is to retrieve all the ways from the OSM. Then we will check if it is building or road respectively. Such information is possible to retrieve using the tags that each way has, if it contains "building" it is naturally a building, if it contains "highway" it is a road. Next we will retrieve all the nodes of a way, again due to the conversion from XML to object, this step will also be easy, because each way object has a list of nodes. These Nd are nothing less than objects that contain the references (which will serve as ids) of the nodes. The resulting list of nodes is sent as an argument to a method called **getGeometry** which in each case will return the Feature type equivalent to the layer type in GeoJSON. In the case of buildings it will be Polygon, in the case of roads it will be a LineString. Then the Feature is created and added to the FeatureCollection. This FeatureCollection is then transformed into a JSON string to create a LayerDAO that contains the map id to which that layer belongs, the layer type and the Feature Collection itself, after which that LayerDAO is saved as an entry in the **Layers** table.

This algorithm is executed when the project receives a request for the endpoint "/map/save". In the next list we can see all the endpoints that AGV-MAP contains and which functionalities they handle:

- POST "/map/save" - Receives a OMS XML file, converts it to GeoJSON and stores it in database.
- GET "/map/active" - Returns the id of the active map.
- GET "/map/change/active/mapId" - Change the active map, receiving in the mapId as a Path Variable.
- GET "/map/mapId" - Returns a MapDTO of the given id. This MapDTO contains also the list of Layers associated to the map.
- PUT "/layer/add" - Receives a LayerDTO, convert to LayerDAO and insert into the database.
- GET "/layer/all/type" - Returns all layers of a type.
- GET "/layer/type/layerType" - Returns the specified layer type from the active map.
- GET "/layer/mapId/type" - Returns the specified layer type from the map with the given id.
- GET "/layer/layerId" - Returns the layer with the given id.

5.5 AGV-ALR

The nomenclature AGV-ALR comes from AGV-Alert. This project is also a CRUD micro service with the purpose of receiving the alerts generated by the other services and storing them. It also has endpoints that allow the synchronisation of clocks, when requested it is stored in the database which was the vehicle that requested and the previous time of the vehicle.

- POST "/alr/create" - Receives an alert from the controller and creates a database entry with the alert.

- GET "/alr/all" - Returns all alerts stored.
- GET "/alr/recent" - Returns the most recent alert according to timestamp.
- Post "/time" - Receives the timestamp from a machine and returns its own, storing the interaction in the database. (Clock synchronisation)
- GET "/time/all" - Returns all interactions resulting from clock synchronisation.

5.6 AGV-VMS

The name of this project AGV-VMS stands for AGV-Vehicle Micro Service. Having deciphered the nomenclature of this project, we can already foresee its purpose. It was necessary to create this project in order to store information about the vehicle, as well as the sessions done by the same. These sessions store the entire history of the vehicle from the moment it was given a destination until the moment it reaches it. Initially this concept of vehicle and session would be handled by different services, but the relationship and dependency between them is such that after it was decided to merge into a single project.

The AGV-VMS ends up being almost a true CRUD project definition. It makes available the methods to create Sessions, Vehicles and other inherent objects, as well as recover and update.

This project provides the following endpoints:

- POST "/session/create" - Receives a SessionDTO to be stored.
- GET "/session/all" - Returns a list of all sessions.
- GET "/session/id" - Returns the session with the given id.
- PUT "/session/veh/data" - Inserts the vehicle state received in body.
- GET "/session/stop/veh/vehicleId" - Stops active session from the vehicle with the given id, if exists.
- GET "/session/veh/last/coord/vehId" - Returns the last known coordinate from the vehicle with the given id.
- GET "/session/veh/current/state/vehId" - Returns the last known state from the vehicle with the given id.
- GET "/session/from/veh/vehId" - Returns the active session from the vehicle, if exists.
- GET "/veh/all" - Returns all vehicles.
- GET "/veh/vehId" - Return the vehicle with the given id.

- GET "/veh/logs/id" - Returns all logs from the vehicle with given id.
- GET "/veh/change/state" - Changes the state from the vehicle (e.g. Offline, Error, Paused etc..)
- DELETE "/veh/id" - Deletes the vehicle with the given id from the system.
- POST "/veh/registration" - Inserts the vehicle in the system.

5.7 AGV-CTL

The AGV-CTL is the key design of the solution. The nomenclature comes from an abbreviation for Controller. As already mentioned, almost all status, information and data pass through the controller, the controller is then the centrepiece.

It is the controller that establishes communication with the vehicle, as mentioned in Section 5.1.2, this communication is done through the TCP protocol. This is where the use of the Spring Integration tool comes in. With the help of this tool, we are able to open ports so that the vehicle can periodically send its status. In addition, we have also managed to establish communication for sending the trajectory correction to the vehicle. This functionality is stored in the package **messaging**. Later on this vehicle-controller interaction will be better explained, detailing all the components and communications involved for the registration and exchange of information between actors.

Besides the communication with the vehicle, the controller also serves as an information gateway to the front-end. Providing almost all the information and endpoints that AGV-FE needs, from the registered vehicles, to providing the methods to assign a destination to the vehicle, to stop the session, or to know the last known coordinates of the vehicles.

While the vehicles are in motion it is necessary to monitor them. The controller has a system implemented that when the vehicle is in an active session, each time a new state is received, the position and direction of the vehicle is analysed so that it can be determined whether it has hit another vehicle or obstacle. In addition to this monitoring, it is also checked if the vehicle has left the limits of the road. In Section 5.11 it will be discussed how these features were developed.

Although much of the information is stored in the AGV-VMS, there is some data that is not. An example is when the vehicle has no active session, although the vehicle continuously sends the status, it is not stored by the AGV-VMS because there is no session. So it is necessary to resort to a kind of cache implemented in the AGV-CTL that contains the last state sent by the vehicle.

5.7.1 *"Cache" Implementation*

So what is this kind of cache implemented in the AGV-CTL? There is some data, whether volatile objects or not, that is required to be accessed in different classes or methods. This way it was thought to create a class with @Repository annotation, that would work as a kind of cache. The annotation allows this class to be @Autowired to other classes, allowing quick access to data. This class that will work as a cache was nicknamed "Data", as we can see in Listing 5.1, this class contains 3 objects that are classes that provide methods to insert in the cache or retrieve data from it.

```
@Repository
public class Data {
    private @Autowired VehTcpInfoRep rp;
    private VehTcpInfo vehTcpInfo;
    private TrajectoryScheduleMap trajectoryScheduleMap;
    private VehicleState vehicleStateMap;
    ...
}
```

Listing 5.1: Data class from AGV-CTL

There is also another variable, this time @Autowired, which links the Data class to the VehTcpInfoRep class. This class is a repository that uses Spring Data JPA, to store the TCP, HTTP and IP address information of a registered vehicle. This is the only information that besides being cached is also stored persistently, so in case of sudden controller failure, the information about the ports and IPs of the vehicles are not lost, preventing them from having to re-register.

The trajectoryScheduleMap class stores a map whose key is the vin of the vehicle and the value a ScheduledFuture<?> which consists of a runnable that periodically corrects the trajectory of the vehicle. For logistic and exception handling reasons it is necessary to have quick access to these scheduled tasks. In Section 5.10 the importance of being able to access these tasks will be developed in depth.

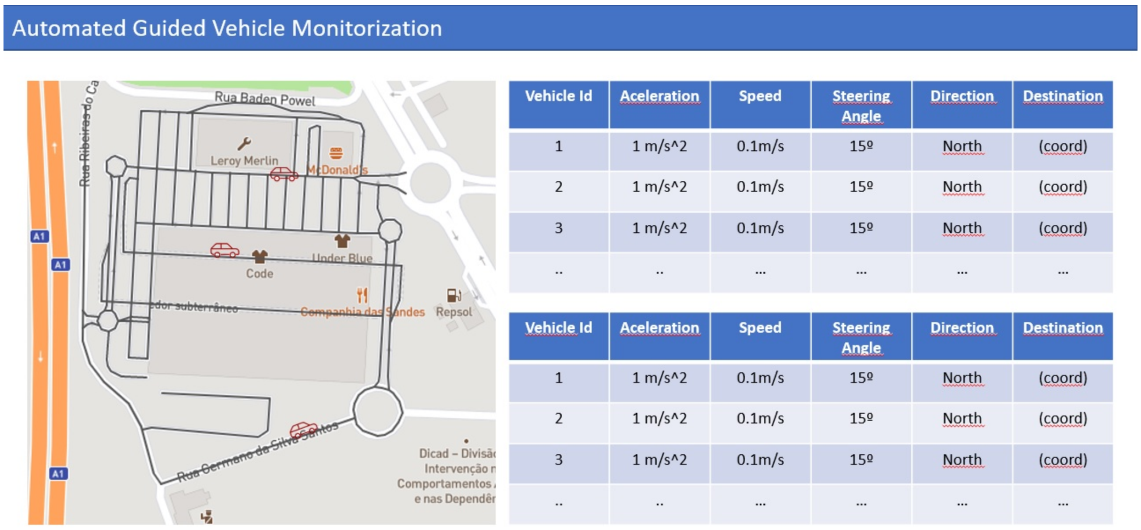
The vehicleStateMap class stores a map whose key is the vehicle id and value is the last known state of the vehicle. This map only keeps information about the last known state of the vehicle, if there is no active session for the same vehicle.

5.8 AGV-VEH

Establishing communication only with the controller, this is how the AGV-Vehicle, nomenclature AGV-VEH, fits into the final solution, being the "moving part". As it follows the general structure and as it also uses, naturally, the TCP protocol, it includes packages like messaging and routing. As explained in Section 5.1.2 - "TCP and HTTP", the vehicle uses both communications to integrate into the project. It provides a route with endpoint **"veh/command/{command}"** so that, in future work, it can send commands like Stop and others. In the TCP part, it includes classes to generate Integration Flows, Transformers and Gateway.

5.9 AGV-FE

As the goal is to develop a simulator of Automatically Driven Vehicles in a controlled environment, it is necessary a platform, in this case web app, that at least is able to show a map representation, the live location of the vehicles. Besides these requirements the developed application also shows some data, such as the status of the vehicles and sessions already performed. In Figure 21 we can see a mockup of the intended appearance and organization of the application.


Automated Guided Vehicle Monitorization
Rua Baden Powel
Rua Ribeira do Ca
Leroy Merlin
McDonald's
Code
Under Blue
dor subterrâneo
Companhia das Sandes
Repsol
A1
Rua Germano da Silva Santos
Dicad – Divisã Intervenção n Comportamentos e nas Dependê

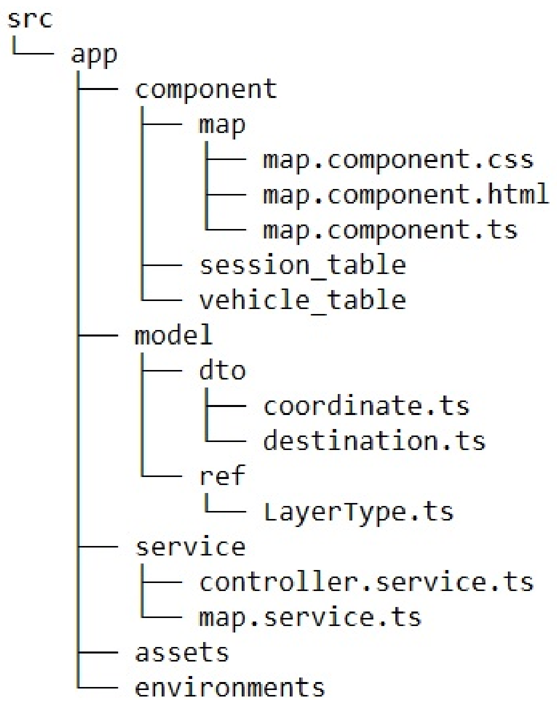
Vehicle Id	Aceleration	Speed	Steering Angle	Direction	Destination
1	1 m/s^2	0.1m/s	15º	North	(coord)
2	1 m/s^2	0.1m/s	15º	North	(coord)
3	1 m/s^2	0.1m/s	15º	North	(coord)
..	..	...	...	...	...

Vehicle Id	Aceleration	Speed	Steering Angle	Direction	Destination
1	1 m/s^2	0.1m/s	15º	North	(coord)
2	1 m/s^2	0.1m/s	15º	North	(coord)
3	1 m/s^2	0.1m/s	15º	North	(coord)
..	..	...	...	...	...

Figure 21: Web App Mockup.

In Section 5.2, a structure that almost all projects of the solution follow is described, as all projects except the AGV-FE were developed using the same tools and programming language it is possible to generalize the structure. However, AGV-FE was developed with the Angular tool in TypeScript, so it has its own structure. Next, in Figure 22 is shown an summarised structure of the AGV-FE.

In this tool pages are defined as components. A component is normally composed of 3 files, let's exemplify using the component for the map representation, this component is constituted by **map.component.css**, **map.component.html** and **map.component.ts**. The type



```
src
└── app
    ├── component
    │   ├── map
    │   │   ├── map.component.css
    │   │   ├── map.component.html
    │   │   └── map.component.ts
    │   ├── session_table
    │   └── vehicle_table
    ├── model
    │   ├── dto
    │   │   ├── coordinate.ts
    │   │   └── destination.ts
    │   └── ref
    │       └── LayerType.ts
    ├── service
    │   ├── controller.service.ts
    │   └── map.service.ts
    ├── assets
    └── environments
```

Figure 22: AGV-FE summarised structure

of file already reveals the purpose of each one, the skeleton of the page, or part of the page as the case presented, is defined in the HyperText Markup Language (HTML) file, the style of the page will be defined in the Cascading Style Sheets (CSS) file and finally in the TS file, TypeScript, the methods necessary for the operation of the page are defined.

In the application it is necessary to make requests for data, following the line of the previous example, it is necessary to ask the AGV-Map for the constituent layers of our environment. For this, service classes are created that hold the methods that will be later called on the components to make the requests.

As in the previous projects, as there is information exchange, it is also necessary to define the models for the objects, hence there is a directory called model, where the Data Transfer Object are defined.

As mentioned before, the main goal of this web application is to be able to visualize the map and the vehicles moving, to achieve this result a tool called Mapbox GL JS from the company Mapbox was used. In Section 5.1.6 it is already stated that this tool contains a platform for developers that allows us to create and customize the layout and information we want to show on the map. Through the methods and API provided by Mapbox GL JS, we can intuitively display a map on the front-end. In Listing 5.2 line 2 is made the creation of an object of type Mapbox and defined that the class map property will be this object. When creating the object we can define several options, in this case we define that the container where the map will be rendered is an element of type **div** called map, the style of the map can be found in that (), which coordinates the map should be centered by and the zoom. After creating the object, it is necessary to load the layers and after that the vehicles, something that proves easy using again the methods provided by the tool. In Listing 5.3 on line 2 it is possible to see that it is only necessary to execute this.map.addLayer(..) to add the layer, with no other worries because Mapbox GL JS is GeoJSON compatible. For the vehicles markers are added on the map, it involves a bit more programming as it is necessary to create a new element for the marker to be visible and frequently update the coordinates of the marker, so that it is possible to see the vehicle in motion.

```
buildMap(){
    this.map = new Mapbox({
        container: 'map',
        style: "mapbox://styles/chico2911/cl585y75c006q14pnjsuu252d",
        zoom:17,
        center: [-8.5655, 40.9216]
    });
    ...
}
```

Listing 5.2: Excerpt from map creation method on AGV-FE

```
1   createLayer(lType: string,lData: string){
      if (!this.map.getSource(lType)) {
3       this.map.addSource(lType, {
            type: 'geojson',
5           data: lData,
        });
7     }
      switch (lType) {
9       case LayerType.ROAD:
            this.map.addLayer({
11              id: LayerType.ROAD,
                type: 'line',
13              source: LayerType.ROAD,
                paint: { 'line-color': '#c7d3ec', 'line-width': 5 }
15          });
            break;
17      ...
        }
19  }
```

Listing 5.3: Excerpt from method CreateLayer() from AGV-FE

5.10 VEHICLE-CONTROLLER INTERACTION

The communication between the vehicle and the controller is something quite important for the whole operation of the solution. That is why it is so important to explain how it works and how the different technologies were used together to achieve the final result.

Previously we already had a clue that this communication was established in an initial way using HTTP requests and in a following phase using TCP requests.

5.10.1 *Registration and Handshake*

Figure 23 is a sequence diagram that gives us an overview of how the first communication is then carried out. The communication is essentially divided in two moments, the registration and the handshake, following the presented order.

The registration is then the first phase to be executed, it is necessary to ensure that the vehicle is registered in the system and if not registered it is not possible to execute the handshake, thus placing a small protection against possible systems that are impersonating vehicles. Both registration and handshake occur automatically, so that no human intervention is required when a new vehicle is added.

After the initial boot of any Spring application , it fires an ApplicationReadyEvent. And it is through this event that the registration and handshake process begins. A method of the RegisterVehicle class, belonging to the config package, contains an EventListener that reacts to the ApplicationReadyEvent event. This method in turn executes two other methods, reinforcing the separation of the processes. The method is present in Listing 5.4.

```
@EventListener(ApplicationReadyEvent.class)
public void vehicleRegistationInSystem() {
    //First let's registe the vehicle in VMS (All requests are made to CTL)
    registationInSystem();
    //If Vehicle is registed, let's proceed to "handshake" with CTL
    handshakeWithCTL();
}
```

Listing 5.4: Method for Vehicle Registration from AGV-VEH

The first method to be executed is `registationInSystem()`, this method consists of a loop that is executed until the response received is positive. In this loop a HTTP POST request is sent to the controller through the route **"ctl/veh/registration"**. In this request is sent an object of the VehInfoDTO type containing the structure presented in Listing 5.5

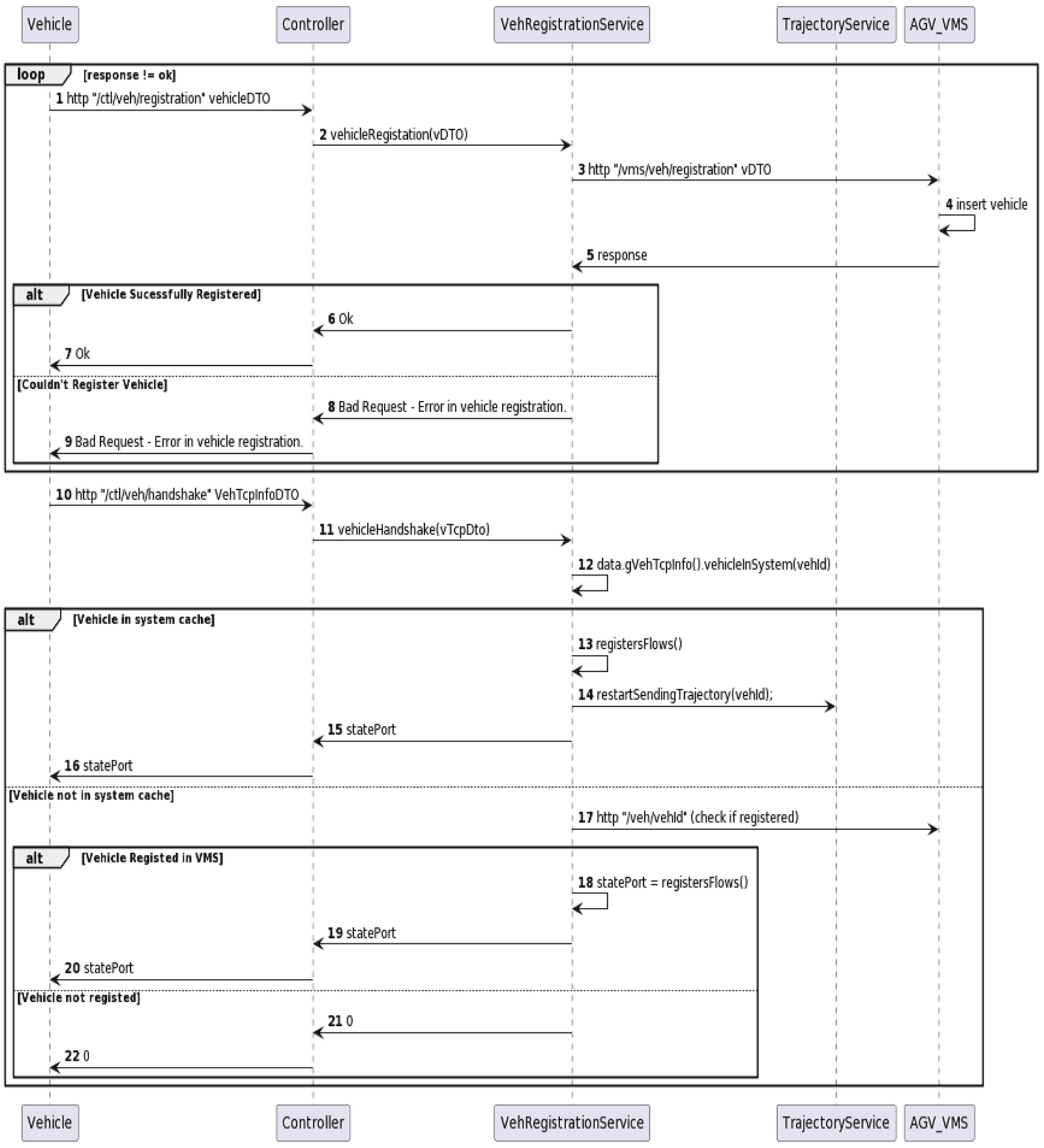

Vehicle
Controller
VehRegistrationService
TrajectoryService
AGV_VMS
loop [response != ok]
1 http "/ctl/veh/registration" vehicleDTO
2 vehicleRegistation(vDTO)
3 http "/vms/veh/registration" vDTO
4 insert vehicle
5 response
alt [Vehicle Sucessfully Registered]
6 Ok
7 Ok
[Couldn't Register Vehicle]
8 Bad Request - Error in vehicle registration.
9 Bad Request - Error in vehicle registration.
10 http "/ctl/veh/handshake" VehTcpInfoDTO
11 vehicleHandshake(vTcpDto)
12 data.gVehTcpInfo().vehicleInSystem(vehId)
alt [Vehicle in system cache]
13 registersFlows()
14 restartSendingTrajectory(vehId);
15 statePort
16 statePort
[Vehicle not in system cache]
17 http "/veh/vehId" (check if registered)
alt [Vehicle Registed in VMS]
18 statePort = registersFlows()
19 statePort
20 statePort
[Vehicle not registed]
21 0
22 0

Figure 23: Vehicle Controller Registration and Handshake General Sequence Diagram .

```
public class VehInfoDTO {
    private Long vehicleId;
    private Double length;
    private Double wide;
    private Double topSpeed;
}
```

Listing 5.5: VehInfoDTO Class excerpt

After receiving the request, the controller basically forwards it to the AGV-VMS, sending an HTTP request with the same object. This request basically as explained in Section 5.6 inserts the vehicle into the database. If the AGV-VMS responds successfully, that response is forwarded to the vehicle, otherwise it receives a bad gateway status. If the vehicle is already registered, the controller receives a success message as well so that the vehicle can proceed to the handshake phase. This first iteration is demonstrated between steps 1 to 9 in the sequence diagram of Figure 23.

The handshake process, as we can see in step 10 of Figure 23, starts, again, with an HTTP POST request to the controller via the **"ctl/veh/handshake"** route. Similar to what happens in the registration process, here also an object is sent in the request. This object is of type VehTcpInfoDto and contains information about the IP of the vehicle, open port to receive trajectory corrections and commands, in Listing 5.6 is shown an excerpt of the model class.

```java
public class VehTcpInfoDTO {
    private Long vehicleId;
    private String ipAddress;
    private Integer trajectoryPort;
    private Integer commandPort;
    ...
}

```

Listing 5.6: VehTcpInfoDTO Class excerpt

After receiving the request, the controller will check if it already contains the cached information of the Vehicle, and if so, a method is executed to create, if necessary, the flows and gateways. This method will be discussed later. If the vehicle is not registered in the cache, we move on to the second verification step, which is to check whether the vehicle is in the AGV-VMS database. If so, the method previously mentioned for the creation of flows and gateways is executed. In both of the previous scenarios the controller sends to the vehicle the port that will be available to receive the TCP communication with the state of the vehicle. If none of the previous scenarios occurs, an error, Internal Server Error, is sent to the vehicle as a response.

In the method mentioned above, `registerFlows()` there are four moments that are quite important, the creation of the first Integration Flow used to establish TCP communication with the vehicle regarding possible trajectory corrections. The next moment is the creation of the gateway that will use the connection created by the Integration Flow to provide the methods to the service to send the trajectory adjustments. In Listing 5.7 an excerpt of the class that constitutes a gateway is shown. The third moment consists of saving the vehicle's

TCP information in our cache and to guarantee data persistence and that in case of controller failure, the TCP data of the registered vehicles are not lost. Finally, the fourth and last moment is the creation of the Integration Flow that will allow us to open a port that will listen for TCP requests with updates on the vehicle status.

```
public interface ControllerGateway {

    String send(String payload);

    @Gateway
    CompletableFuture<String> sendAsync(String payload);
}
```

Listing 5.7: Controller Gateway

In the referred moments we can see that two Integration Flows are created for different functions, in the Listings 5.8 and 5.9 we can see the methods that create the Integration Flows respectively with the moments. The Listing 5.8 receives the IP and the port that the vehicle has listening, so we create a connection that will send data. In the Listing 5.9 the opposite happens, an Integration Flow is created that will open a port in the system to receive TCP requests. In line 6 of Listing 5.8 we see that we are transforming the object into a string, using a method from the existing one. However on line 5 of Listing 5.9 we see that a `.transform(this.statusTrans)` is called, this method uses a StatusTransformer class that transforms the vehicle state from string to object of type State, forward this type will be address.

```
1 public IntegrationFlow buildFlow(String flowId,String vehicleId,String
  ipAdress,Integer port) {
      IntegrationFlow flow =  f -> f
3             .handle(Tcp.outboundGateway(Tcp.netClient(ipAdress,port)
              .serializer(TcpCodecs.crlf())
5             .id(vehicleId)))
              .transform(Transformers.objectToString());
7     this.flowContext.registration(flow).id(flowId).autoStartup(true).register
  ();
      return flow;
9 }
```

Listing 5.8: Outbound Integration Flow creation

```
1 public IntegrationFlow buildFlow(Integer port) {
      IntegrationFlow flow =  IntegrationFlows.from(Tcp.inboundGateway(Tcp.
  nioServer(port)
3     .deserializer(TcpCodecs.crlf())
      .get()))
```

```
        .transform(this.statusTrans)
        .get();
        this.flowContext.registration(flow).id(String.valueOf(port)).register();
        return flow;
    }
```

Listing 5.9: Inbound Integration Flow creation

5.10.2 *TCP Data Exchange*

After receiving the response to the HTTP request, the vehicle will proceed in a similar manner to the controller, create Integration Flows which will open a port to listen for the trajectory corrections sent by the controller and create a Gateway and Integration Flow to send the status, periodically, to the controller through the port that the Controller has designated open to receive requests. After this phase of knowledge and registration carried out in HTTP, we move on to the exchange of information between Vehicle and Controller. This is where the use of the tool mentioned in Section 5.1.3, Google Protocol Buffer, comes in. Using this tool it is possible to write a file with the .proto typology that enables to structure the information. Google Protobuff provides a compiler that through the file creates a "special generated source code" [6] that provides methods to easily read and write objects with the structure that we draw. In Listing 5.10 you can view the file `.proto` developed. In line 3 it is designated for which package the source code should be generated at the moment of the solution compilation.

```
1     syntax = "proto2";
      package pt.deloitte.agv.ctl.model;
3     message Coordinate {
          required double latitude = 1;
5         required double longitude = 2;
      }
7     message State {
          required int64 id = 1;
9         required double aceleration=2;
          required double speed=3;
11        required double steering_Angle=4;
          required Coordinate localization=5;
13        required Coordinate destinationCoord=6;
      }
```

Listing 5.10: Protobuff file used to structure vehicle state data

6 https://developers.google.com/protocol-buffers

At this point it is necessary to ask why the use of this tool. Structuring the data using a multi-language tool, being compatible with Java, C++, Objective-C, Python, Ruby and others. It allows a greater compatibility and exchange in the programming languages that were used to develop both the vehicle and the controller. JSON could have been used, however, taking Google's word for it, Protobuffer is smaller and faster, plus it's not human readable.

Then, continuing after the handshake, a Scheduler is created on the vehicle side that is executed in a cycle within a defined time period of 1 second. This task consists of constantly sending status updates to the controller via the gateway and integration flow created at the time of handshake. These TCP requests are asynchronous and contain objects of the State type, which are created using the aforementioned tool.

However as Figure 24 suggests, and also as already mentioned, the controller is prepared to send possible trajectory corrections that need to be made to the vehicle. This trajectory

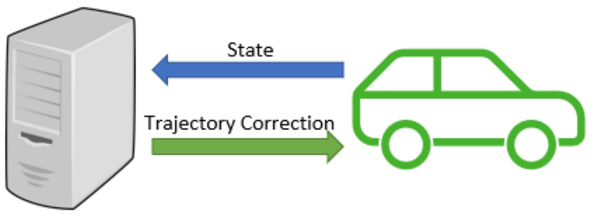


Figure 24: TCP Data Exchange Diagram between Controller and Vehicle

correction sending process only happens when the user, via the AGV-FE defines a new destination for the vehicle. The AGV-FE will then inform the Controller of the vehicle's new destination, which in turn initiates a new session and begins sending trajectory adjustments to the vehicle. This occurs in a similar way to the status sending process. A Scheduler is created, which will be stored in the "cache", that every second will send the supposed trajectory correction. It is important to emphasize that this Scheduler is kept in the cache so that in case of disconnection with the vehicle the Task is killed and the attempt of sending information to the vehicle is stopped. Although the sending of trajectory corrections is then available, this module will be part of a work to be developed.

5.11 VEHICLE IMPACT AND OFF-LANE MONITORING

As the main objective is to develop a simulator that allows the visualisation of the vehicles, it is also necessary to develop a system that monitors the vehicles and informs the user that a vehicle has hit another vehicle or an obstacle or if the vehicle is not at the limit of the road.

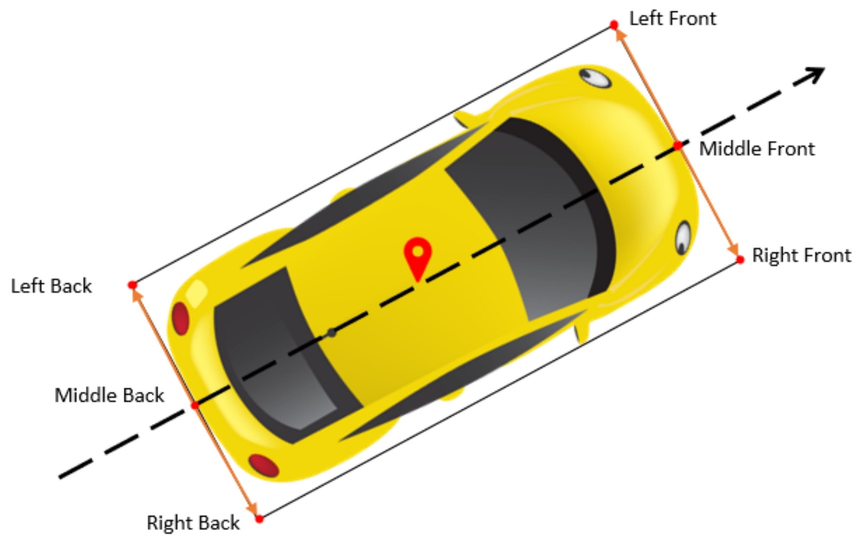


Figure 25: Diagram of how to find the four coordinates that form the rectangle of the vehicle

5.11.1 *Vehicle Impact*

To monitor whether the vehicle has hit another vehicle or an obstacle such as a building, it is needed first to find the four coordinates that outline the vehicle, forming a rectangle. The Figure 25 shows a vehicle and the method found to calculate the four coordinates.

With the vehicle's coordinate, length and direction, we can find the front midpoint of the vehicle. This is possible using a method defined on the Movable Type Scripts [7] website, this method has been converted from JavaScript to Java and have as input a coordinate, a distance and the bearing, returning the coordinate resulting from the translation of the input coordinate. After finding the front mid-coordinate, by removing 90 degrees to the direction of the car and using the width of the vehicle we are able to locate the front left coordinate. Similarly but by adding 90 degrees to the direction of the car we were able to calculate the front right coordinate. To find the rear points we follow the same steps as above, however we take 180 degrees off the direction of the vehicle to turn around and find the rear coordinate.

Checking whether two polygons intersect is a bit complex. To do this we used the `java.awt.geom` library from Java which provides objects such as Area and Path2D which in turn contain methods that check the intersection. Having already obtained the four coordinates of the vehicle, it is possible to create a Path2D with which the Area representing the vehicle is created. As we want to see if the vehicle has crashed into a building, it is necessary to ask AGV-MAP for the layer referring to the buildings. Then each Feature is transformed into a Path2D and again into an Area. Using the predefined method is possible to execute `featureArea.intersects(vehicleArea).isEmpty()`, if the result is false it means

7 http://www.movable-type.co.uk/scripts/latlong.html

that the Area of the vehicle intersects with the building, therefore crashed. Triggering a warning for the AGV-ARL. Using the method described above, it is also possible to check whether one vehicle has hit another. By calculating the two rectangles containing the vehicles and confirming that their area intersects we can then calculate whether the vehicles have crashed.

5.11.2 *Vehicle Off-Lane*

It is also important that the vehicle stays within its lane. To confirm that this is the case, a system has been developed that monitors the vehicle's coordinate and calculates whether it is within a specified radius, Figure 26 is a schematic diagram that allows in a visual manner to understand the thought developed.

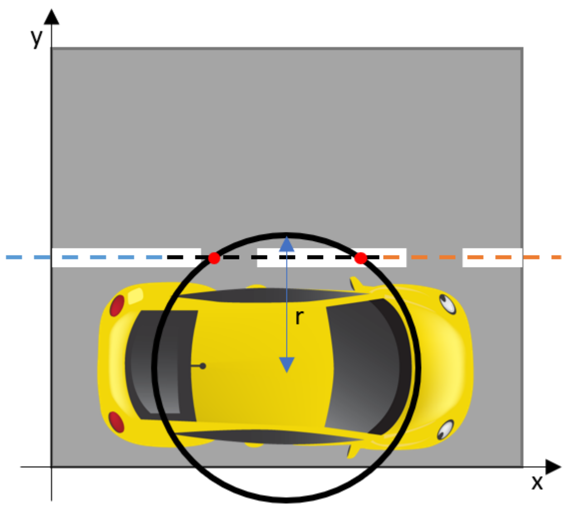


Figure 26: Schematic diagram of Off-Lane calculation

The dashed line in the centre of the rectangle, represents the line representing a road on the map's constituent road layer. That is, the roads on the map are represented by a line passing through the centre of them.

To verify that the vehicle is always in the lane, mathematical equations were used. As can be seen in Figure 26 to confirm that the vehicle is in its lane, the resulting circumference from the centre of the vehicle must intersect the line segment. This is because roads are made up of several straight line segments. A line segment is represented by two points. Knowing these two points and using the Fundamental Equation of the Line, $m = \frac{(Px-Bx)}{(Py-By)}$, and the Equation of the Line, $y = mx + n$, you can calculate a line that passes through that line segment. The equation of the circumference is $(x-h)^2 + (y-k)^2 = r^2$ where (h, k) is the centre of the circumference and r is the radius. Having the equation of the line passing through the line segment and the equation of the circle it is possible to calculate if they intersect, solving the system:

$$\begin{cases} y = mx + n \\ (x-h)^2 + (y-k)^2 = r^2 \end{cases} \tag{1}$$

Solving this system will result in a quadratic function $ax^2 + bx + c = 0$. Using $\Delta = b^2 - 4ac$ it is possible to calculate whether the line is secant, tangent or external to the circle, if $\Delta > 0$, $\Delta = 0$ or $\Delta < 0$ respectively.

The Controller then has a method that on each line segment constituting a road, will calculate through the aforementioned method whether the vehicle is within the lane, until it finds the intersection. However in addition to checking whether the vehicle intersects the line passing through the line segment it is necessary to check whether the points resulting from the intersection are between the line segment. Figure 27 gives a better understanding of the problem. The line segment is represented by the thicker line and the line passing through that line segment is represented by the thinner line.

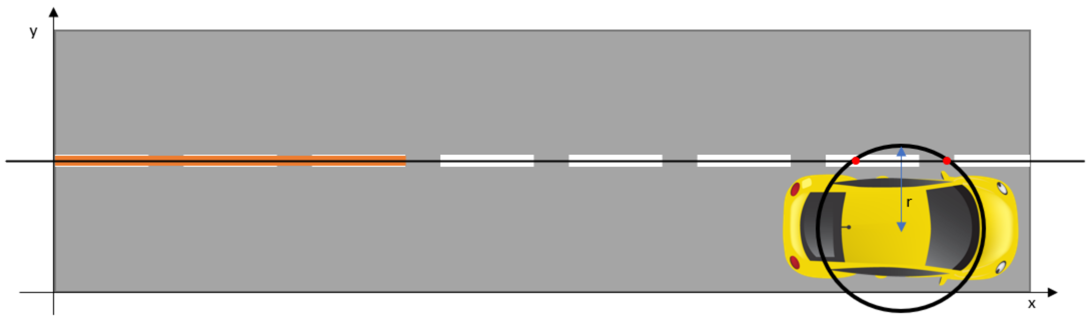


Figure 27: Vehicle intersecting with line and not line segment.

Although the line passing through the line segment is secant to the circle, the vehicle is not parallel to the line segment. It is therefore necessary for the intersection points to be between the points that define the line segment. Given that P and B are the points defining the line segment and C is a point of intersection, it is necessary that the following conditions are true: $(((Px \geq Cx \geq Bx) \parallel (Px \leq Cx \leq Bx))\&\&((Py \geq Cy \geq By) \parallel (Py \leq Cy \leq By)))$

5.12 SUMMARY

With the help of the Conceptual Architecture Diagram it is possible to decompose the solution into several modules. An architecture of micro-services was followed for the implementation of the different modules, in this way it is possible to achieve greater expandability and create greater independence between functionalities. Six services were developed: AGV-CTL, AGV-VEH, AGV-FE, AGV-VMS, AGV-MAP and AGV-ALR, where the AGV-FrontEnd is a web application and the remaining projects are micro-services that communicate among themselves.

The development of the micro-services was accelerated by the use of Spring framework, using technologies such as TCP and HTTP for communication between micro-services. These micro-services contain CRUD functionalities, which require a database, which was done using the H2 Database Engine. In order to organise the information stored in these services a database structure was created, described in Section 5.3. In Figure 20 the logical data model designed to meet the needs of the solution is provided.

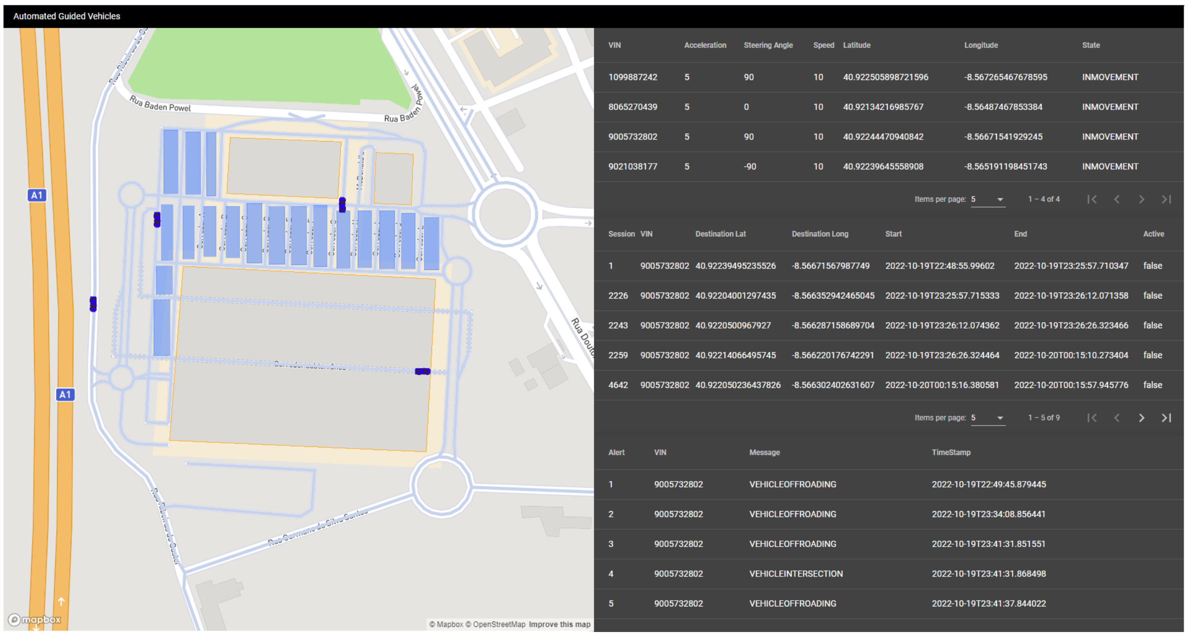
The AGV-MAP is a map management micro-service, where the maps of the environments to be simulated and the layers belonging to each map are stored. AGV-ALR implements an alert and clock synchronisation service. When a vehicle impacts or drifts out of the traffic lane, an alert is created and stored in this system, which can be later consulted through AGV-FE. It is necessary to store information about the vehicles present in the system, as well as the sessions and data corresponding to them. To achieve this, the AGV-VMS micro-service was developed.

The AGV-CTL is the central controller of the solution. It is through this micro-service that all the information flows, from the vehicle's speed, location, acceleration and direction as well as the new coordinates to assign to the vehicles and some others AGV-FE inputs to pause or stop sessions. In order to automate the process of vehicle discovery, methods for vehicle registration and handshake were developed, so the human hand is not needed to add a new vehicle. This first contact between vehicle and controller was described in more detail in Section 5.10. Being the Controller, it is in constant communication with the vehicles, constantly receiving vehicle status updates and being able to send trajectory corrections to them. This communication is developed in TCP, and to this end the Spring Integration tool was used, which enables these communication bridges to be quickly created. To achieve a good compatibility with different programming languages and since it also has better performance than other tools of the same concept was used Google Protobuffer to standardize the objects exchanged between vehicle and controller. In this micro-service are also implemented methods for monitoring the vehicle, which generate alerts on the AGV-ALR, so that the user through the AGV-FE is notified that a vehicle collided or drifted off the lane. The implementation of these methods was described in Section 5.11.

6

SYSTEM OVERVIEW

In the Chapter 5 a web application is discussed that allows to see in a more graphical way the positions of the vehicle, trajectories travelled and other relevant information. Throughout this chapter there will be several pictures and explanations on how to operate the application. Figure 28 shows the final web application front-end that consists of a map and three tables. The image in Figure 28 shows that four vehicles are active, and that they are moving, according to the first table on the right. In the following tables it is possible to check the finished and ongoing sessions, as well as the alerts generated from the monitoring discussed in Section 5.11.



Automated Guided Vehicles
Rua Baden Powel
Rua Baden Powel
A1
A1
Rua Douto
Rua Germano da Silva Cortes
© Mapbox © OpenStreetMap Improve this map
mapbox

VIN	Acceleration	Steering Angle	Speed	Latitude	Longitude	State
1099887242	5	90	10	40.922505898721596	-8.567265467678595	INMOVEMENT
8065270439	5	0	10	40.92134216985767	-8.56487467853384	INMOVEMENT
9005732802	5	90	10	40.92244470940842	-8.56671541929245	INMOVEMENT
9021038177	5	-90	10	40.92239645558908	-8.565191198451743	INMOVEMENT

Items per page: 5 1 – 4 of 4

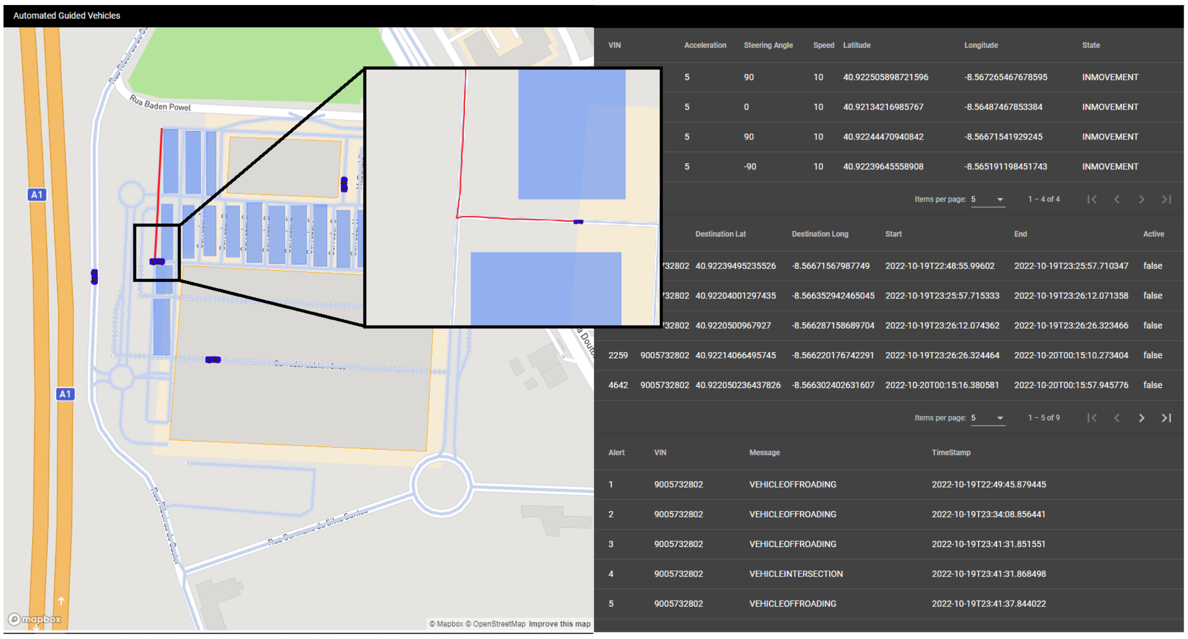
Session	VIN	Destination Lat	Destination Long	Start	End	Active
1	9005732802	40.92239495235526	-8.56671567987749	2022-10-19T22:48:55.99602	2022-10-19T23:25:57.710347	false
2226	9005732802	40.92204001297435	-8.566352942465045	2022-10-19T23:25:57.715333	2022-10-19T23:26:12.071358	false
2243	9005732802	40.9220500967927	-8.566287158689704	2022-10-19T23:26:12.074362	2022-10-19T23:26:26.323466	false
2259	9005732802	40.92214066495745	-8.566220176742291	2022-10-19T23:26:26.324464	2022-10-20T00:15:10.273404	false
4642	9005732802	40.922050236437826	-8.566302402631607	2022-10-20T00:15:16.380581	2022-10-20T00:15:57.945776	false

Items per page: 5 1 – 5 of 9

Alert	VIN	Message	TimeStamp
1	9005732802	VEHICLEOFFROADING	2022-10-19T22:49:45.879445
2	9005732802	VEHICLEOFFROADING	2022-10-19T23:34:08.856441
3	9005732802	VEHICLEOFFROADING	2022-10-19T23:41:31.851551
4	9005732802	VEHICLEINTERSECTION	2022-10-19T23:41:31.868498
5	9005732802	VEHICLEOFFROADING	2022-10-19T23:41:37.844022

Figure 28: AGV-FE Web Page.

It is important to be able to see the trajectory that a vehicle has travelled, this way we can see its starting point and also analyse the path it has travelled. Figure 29 is a close up of what happens when you right click on a vehicle. Right-clicking on the vehicle makes the front-end ask the controller for the vehicle's coordinates, drawing a red line representing the path travelled. The vehicle is in active session when it is in blue or yellow, representing the state `INMOVEMENT` or `PAUSED`, respectively.


Automated Guided Vehicles
Rua Baden Powel
A1
A1
mapbox
© Mapbox © OpenStreetMap Improve this map

VIN	Acceleration	Steering Angle	Speed	Latitude	Longitude	State
	5	90	10	40.922505898721596	-8.567265467678595	INMOVEMENT
	5	0	10	40.92134216985767	-8.56487467853384	INMOVEMENT
	5	90	10	40.92244470940842	-8.56671541929245	INMOVEMENT
	5	-90	10	40.92239645558908	-8.565191198451743	INMOVEMENT

Items per page: 5 1 – 4 of 4

		Destination Lat	Destination Long	Start	End	Active
	732802	40.92239495235526	-8.56671567987749	2022-10-19T22:48:55.99602	2022-10-19T23:25:57.710347	false
	732802	40.92204001297435	-8.566352942465045	2022-10-19T23:25:57.715333	2022-10-19T23:26:12.071358	false
	732802	40.9220500967927	-8.566287158689704	2022-10-19T23:26:12.074362	2022-10-19T23:26:26.323466	false
2259	9005732802	40.92214066495745	-8.566220176742291	2022-10-19T23:26:26.324464	2022-10-20T00:15:10.273404	false
4642	9005732802	40.922050236437826	-8.566302402631607	2022-10-20T00:15:16.380581	2022-10-20T00:15:57.945776	false

Items per page: 5 1 – 5 of 9

Alert	VIN	Message	TimeStamp
1	9005732802	VEHICLEOFFROADING	2022-10-19T22:49:45.879445
2	9005732802	VEHICLEOFFROADING	2022-10-19T23:34:08.856441
3	9005732802	VEHICLEOFFROADING	2022-10-19T23:41:31.851551
4	9005732802	VEHICLEINTERSECTION	2022-10-19T23:41:31.868498
5	9005732802	VEHICLEOFFROADING	2022-10-19T23:41:37.844022

Figure 29: Close up right click action on a vehicle

Let's just focus on the left side of the screen, the map. On the map you can see the vehicles moving and as already mentioned by clicking on the vehicle you can see the path travelled. It was also given a hint that the colour of the vehicle has different meanings. When the vehicle is in blue it means it is moving, in yellow it will be in pause, in red it will be in error state. In Figure 30 it is possible to see the action of left-clicking on a vehicle, be the vehicle any states, this action reveals a menu that allows us to set a destination for that vehicle, pause session and end session. Selecting the "Set Destination" option, the mouse turns into a cross-hair where clicking on the map the destination coordinates are sent to create a new session and set the vehicle in motion. Although the coordinates are sent, as the end goal of the project is to create a simulator of a controlled environment, these coordinates are collected only to be used in future work.

As stated in section 5.11 it is necessary, to alert the user when a vehicle has hit an obstacle or has gone out of its lane. The controller is constantly monitoring if something of the above has happened, if it has, it immediately stops the vehicle, changing its status to `ERROR` and generating an alert to the **AGV-ALR**. The **AGV-FE** in turn is constantly checking for any new alerts, so informing the user is a snap. In Figure 31 you can see a vehicle red-flagged because it has left the lane, in the centre at the top you can see the browser alert that is generated if the incident has occurred less than 30 seconds ago. All other alerts are available in the table on the bottom left.

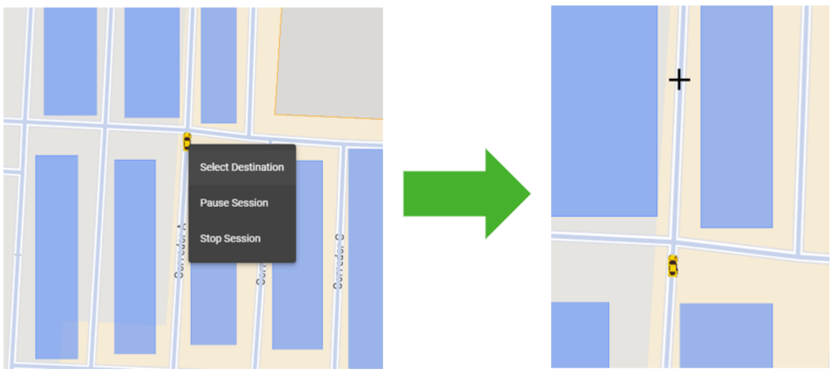
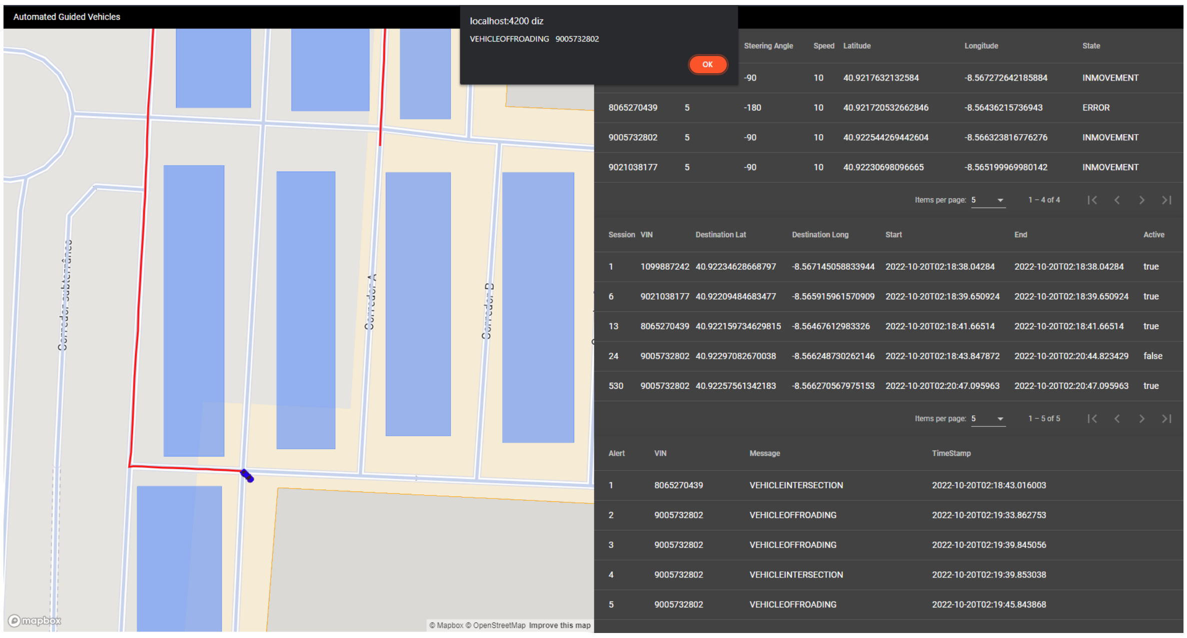


Figure 30: Close up left click action on a vehicle and selecting "Set Destination"


Automated Guided Vehicles
Corredor subterrâneo
Corredor A
Corredor D
© Mapbox © OpenStreetMap Improve this map

localhost:4200 diz

VEHICLEOFFROADING 9005732802

OK

		Steering Angle	Speed	Latitude	Longitude	State
		-90	10	40.9217632132584	-8.567272642185884	INMOVEMENT
8065270439	5	-180	10	40.921720532662846	-8.56436215736943	ERROR
9005732802	5	-90	10	40.922544269442604	-8.566323816776276	INMOVEMENT
9021038177	5	-90	10	40.92230698096665	-8.565199969980142	INMOVEMENT

Items per page: 5 1 – 4 of 4

Session	VIN	Destination Lat	Destination Long	Start	End	Active
1	1099887242	40.92234628668797	-8.567145058833944	2022-10-20T02:18:38.04284	2022-10-20T02:18:38.04284	true
6	9021038177	40.92209484683477	-8.565915961570909	2022-10-20T02:18:39.650924	2022-10-20T02:18:39.650924	true
13	8065270439	40.922159734629815	-8.56467612983326	2022-10-20T02:18:41.66514	2022-10-20T02:18:41.66514	true
24	9005732802	40.92297082670038	-8.566248730262146	2022-10-20T02:18:43.847872	2022-10-20T02:20:44.823429	false
530	9005732802	40.92257561342183	-8.566270567975153	2022-10-20T02:20:47.095963	2022-10-20T02:20:47.095963	true

Items per page: 5 1 – 5 of 5

Alert	VIN	Message	TimeStamp
1	8065270439	VEHICLEINTERSECTION	2022-10-20T02:18:43.016003
2	9005732802	VEHICLEOFFROADING	2022-10-20T02:19:33.862753
3	9005732802	VEHICLEOFFROADING	2022-10-20T02:19:39.845056
4	9005732802	VEHICLEINTERSECTION	2022-10-20T02:19:39.853038
5	9005732802	VEHICLEOFFROADING	2022-10-20T02:19:45.843868

Figure 31: Vehicle Off Lane Alert in AGV-FE

7 CONCLUSION

As Master's Project I decided to accept a challenge from Deloitte Technology S.A., which welcomed me in a 9-month curricular internship program. This internship program allowed me to develop the work of the dissertation in a business context, an experience that became very enriching and allowed a development and gain of knowledge that would not otherwise be possible. Throughout the document, the work developed over about a year was described, full of challenges and difficulties overcome.

Chapter 2 presents the outcome of bibliographic reading and studying of Automated Guided Vehicles and possible technologies that may be used in its implementation. It states the beginnings of AGV since its first implementation, back in 1953, using electrically conductive strips mounted in the ground. Nowadays's with the development of mapping and recognition technologies and wireless protocols there isn't the need of a physical connection to AGVs, making them more free to move and with a larger scope of utilization.

Chapter 3 describes an approach to a solution. This chapter starts to decomposing, through the use of a mind map, the problem of vehicle automation in smaller problems, and those smaller problems in even more fragment problems, like in a "divide and conquer" solution. Decomposing the problem gave a general perception and a better understanding of the constituents of automated driving. The problem was divided in five main aspects: **Environment**,**Map**,**Vehicle**, **Controller**, **Monitoring**, in Figure 15 it's presented a conceptual architecture diagram that make a clear overview of what single aspect represents, and how they relate to each other.

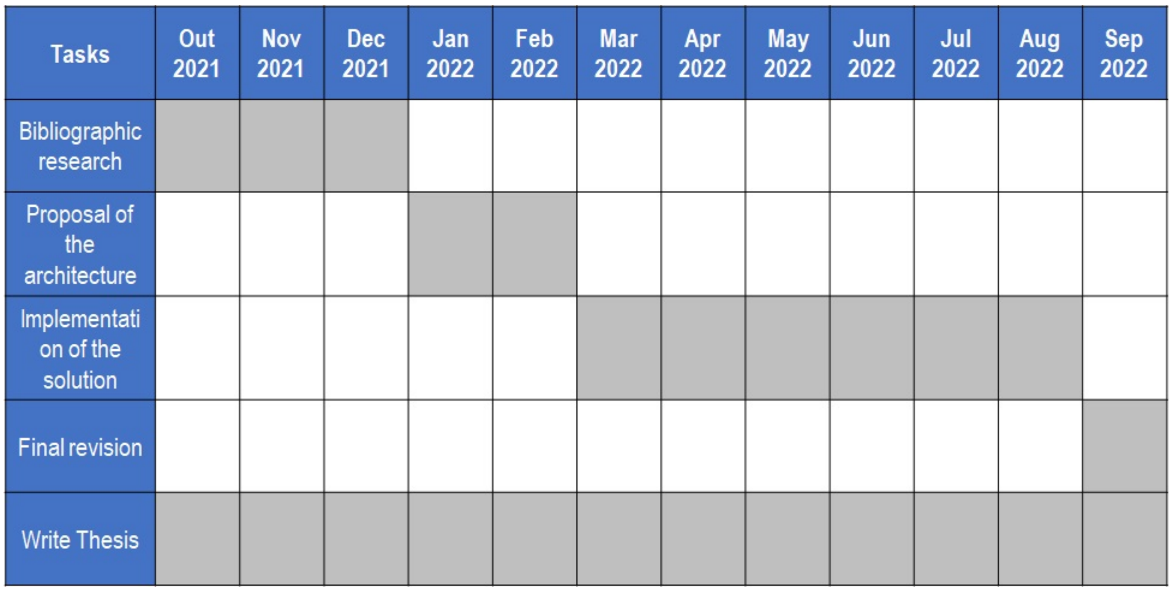
Prior to start with the development itself, it was necessary to consider some assumptions and modules that would not fit in the scope of the required. In this way, Chapter 4 approaches the modules that were not developed, explaining the reasons and solutions designed in order not to disregard the functionalities of the simulator. Solutions for environment representation and monitoring were presented here. Following these considerations, an updated Conceptual Architecture Diagram was presented, which is updated according to what was discussed in the chapter.

After having thought about these considerations, the Development phase took place, which was described in Chapter 5. The development was the most extensive phase of the Master's

Project, including the choice of programming languages, tools and frameworks. In this phase five micro-services were developed: AGV-MAP, AGV-ALR, AGV-CTL, AGV-VMS and AGV-VEH; as well as a web application, AGV-FE. These micro-services work simultaneously depending on each other for the simulator's full operationality. The AGV-CTL micro-service is the central one, through which all the data of the vehicles are transmitted, and it is also here that the monitoring of the vehicles is performed, achieving a warning system alerting the user through the AGV-FE when a vehicle drifts from its lane or collides with an obstacle.

Chapter 6 presented the web application resulting from the Development, in this chapter was described possible interactions of the user with the web application and what information is possible to obtain using it, from sessions, trajectories and alerts. It ends up being a general presentation of the system that constitutes the simulator, where it is possible not only to see the information mentioned above, but also to see the environment where the vehicles circulate and the different layers that constitute the "controlled environment", it is also possible to see in real time the location of the vehicles.

In order to better organise the available time for the development of the Master's Project, a schedule was developed during the Dissertation Proposal stage, presented by the Gantt diagram in Figure 32. It is possible to state with great satisfaction that, in general, the schedule was followed, revealing itself to be a huge help to fulfil the proposed objectives.



Tasks	Out 2021	Nov 2021	Dec 2021	Jan 2022	Feb 2022	Mar 2022	Apr 2022	May 2022	Jun 2022	Jul 2022	Aug 2022	Sep 2022
Bibliographic research	▨	▨	▨									
Proposal of the architecture				▨	▨							
Implementati on of the solution						▨	▨	▨	▨	▨	▨	
Final revision												▨
Write Thesis	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

Figure 32: Project Gantt chart.

In this Master Project, a solution for a simulator of automatically guided vehicles in a controlled environment was developed. This project could be a foundation for future works involving the correction of the vehicle trajectory. Allied to the monitoring functionality it would be interesting to develop a system that calculates the optimal path for the vehicle to follow, using the simulator warnings to adjust the trajectory. Using the division of layers of the environment, and with a parking layer present, a solution could be developed that after the vehicle session is over, guides it to the car park, again involving not only the trajectory correction but also the parking space management.

BIBLIOGRAPHY

ZigBee Alliance. ZigBee Specificarion. *Standard, Oct*, 2015. ISSN 0040-8905.

Drazen. Barkovic, Fachhochschule (Pforzheim), and Sveuciliste Josipa Jurja Strossmayera (Osijek). Ekonomski Fakultet. *Interdisziplinare Managementforschung IV : fourth interdisciplinary symposium, Porec, Croatia, June 1-3, 2007 = Interdisciplinary management research IV*. Faculty of Economics, Jospi Juraj Strossmayer Univ, 2008. ISBN 9789532530445.

Dominique Bonte and James Hodgson. The future of Maps. *ABI Reseach*, 2018. URL https://www.here.com/sites/g/files/odxslz166/files/2019-01/THEFUTUREOFMAPS.pdf.

Laura Boone. *Industry 4.0 (Fourth industrial revolution)*. Salem Press Encyclopedia, 2020. Accessible at https://search.ebscohost.com.

H. Butler, M. Daly, A. Doyle, Sean Gillies, T. Schaub, and Stefan Hagen. The GeoJSON Format. RFC 7946, August 2016. URL https://www.rfc-editor.org/info/rfc7946.

Elif Bílgín. Industry 4.0 and sustainable supply chain. *M U Iktisadi ve Idari Bilimler Dergisi*, 7 2021. ISSN 1300-7262. doi: 10.14780/muiibd.960306.

T. Dierks and C. Allen. The tls protocol version 1.0. RFC 2246, RFC Editor, 01 1999. URL https://www.rfc-editor.org/rfc/rfc2246.txt.

Johannes Horst and Fernando Santiago. What can policymakers learn from germany's industrie 4.0 development strategy?

Zhi Kanmai. Tcp/ip protocol security problems and defenses. In *2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI)*, pages 117–120, 2020. doi: 10.1109/ICHCI51889.2020.00033.

Salam Khanji, Farkhund Iqbal, and Patrick Hung. ZigBee Security Vulnerabilities: Exploration and Evaluating. *2019 10th International Conference on Information and Communication Systems, ICICS 2019*, pages 52–57, 2019. doi: 10.1109/IACS.2019.8809115.

Krzysztof Kryszczuk and Jonas Richiardi. *Springer Encyclopedia of Cryptography and Security*, pages 104–110. 01 2011.

James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012. ISBN 0132856204.

C A Latha and H L Shashidhara. Clock synchronization in distributed systems. In *2010 5th International Conference on Industrial and Information Systems*, pages 475–480, 2010. doi: 10.1109/ICIINFS.2010.5578658.

Jian Luo, Huayi Yin, Bo Li, and Changqing Wu. Path planning for automated guided vehicles system via interactive dynamic influence diagrams with communication. In *2011 9th IEEE International Conference on Control and Automation (ICCA)*, pages 755–759, 2011. doi: 10.1109/ICCA.2011.6137906.

Ahmed El Mahdawy and Amr El Mougy. Path planning for autonomous vehicles with dynamic lane mapping and obstacle avoidance. volume 1. SciTePress, 2021. ISBN 9789897584848. doi: 10.5220/0010342704310438.

Jasprabhjit Mehami, Mauludin Nawi, and Ray Y Zhong. Smart automated guided vehicles for manufacturing in the context of industry 4.0. *Procedia Manufacturing*, 26:1077–1086, 2018. ISSN 2351-9789. doi: https://doi.org/10.1016/j.promfg.2018.07.144. URL `https://www.sciencedirect.com/science/article/pii/S2351978918308205`. 46th SME North American Manufacturing Research Conference, NAMRC 46, Texas, USA.

Inc. Sun Microsystems. *Code Conventions for the Java Programming Language*, 1997. URL `http://www.oracle.com/technetwork/java/codeconventions-150003.pdf`.

David L. Mills. Network time protocol (version 3) specification, implementation and analysis. RFC 1305, RFC Editor, 03 1992. URL `https://www.rfc-editor.org/rfc/rfc1305.txt`.

Camilla Olsen. Security issues relating to the use of udp. SANS Institute, 2003.

Marko Pedan, Milan Gregor, and Dariusz Plinta. Implementation of automated guided vehicle system in healthcare facility. *Procedia Engineering*, 192:665–670, 2017. ISSN 1877-7058. doi: https://doi.org/10.1016/j.proeng.2017.06.115. URL `https://www.sciencedirect.com/science/article/pii/S1877705817326619`. 12th international scientific conference of young scientists on sustainable, modern and safe transport.

J. Postel. User Datagram Protocol. RFC 768, August 1980. URL `https://rfc-editor.org/rfc/rfc768.txt`.

Jens Rieken, Richard Matthaei, and Markus Maurer. Toward perception-driven urban environment modeling for automated road vehicles. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 731–738, 2015. doi: 10.1109/ITSC.2015.124.

Lothar Schulze and Alexander Wullner. The approach of automated guided vehicle systems. In *2006 IEEE International Conference on Service Operations and Logistics, and Informatics*, pages 522–527, 2006. doi: 10.1109/SOLI.2006.328941.

Hamid Taheri and Zhao Chun Xia. Slam; definition and evolution. *Engineering Applications of Artificial Intelligence*, 97:104032, 2021. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2020.104032. URL `https://www.sciencedirect.com/science/article/pii/S0952197620303092`.

Sean Turner. Transport layer security. *IEEE Internet Computing*, 18(6):60–63, 2014. doi: 10.1109/MIC.2014.126.

Sandor Ujvari. Simulation in automated guided vehicle system design. 2003. URL `www.dmu.ac.uk~DEMONTFORT`.

Günter Ullrich. *Automated Guided Vehicle Systems: A Primer with Practical Applications*. Springer Publishing Company, Incorporated, 2014. ISBN 3662448130.