POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Sustainable Urban Mobility: Dimensioning Robotaxi Fleet and Public Transportation for Future Cities

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Filippo Carbonara**

Student ID: 992935
Advisor: Prof. Silvia C. Strada
Co-advisors: Ing. Antonio Pagliaroli, Prof. Sergio Savaresi
Academic Year: 2023-24

Abstract

Autonomous driving technology is set to revolutionize mobility in the coming years, opening new opportunities in the urban mobility and driving the shift from a private model of car ownership to a public shared one. This thesis aims to develop and analyze a multimodal urban mobility system through the use of innovative technology of robotaxis, autonomous electric vehicles shared among users. To achieve this goal, a simulator has built based on real travel data collected from GNSS devices, directly installed on customer vehicles of UnipolSai Group, addressing an optimization problem through Co-design theory. The simulator determines the optimal number of robotaxis needed to meet travel demands within the service area over a week, analyzing waiting times, travel times, and other parameters useful for evaluating the service. Two distinct cases are analyzed. In the first scenario, a system is designed to be competitive with current car-sharing companies. The results demonstrate that it would be significantly more efficient and economically cheaper than existing services. In the second scenario, we hypothesize that the circulation in Area B can be made only with robotaxis or public transports. In this future mobility model the number of robotaxis required would be 10 to 50 times fewer than the number of private cars currently in circulation. This would allow a urban regeneration, substituting parking zones with green spaces and useful areas for the community, in addition to the benefits of 100% green mobility. Economically, for most people, using robotaxis would be more convenient than owning a private car. It is hoped that this type of system will succeed in the coming years, following the example of cities like San Francisco and Wuhan, which are already experimenting with robotaxis, albeit not yet on a large scale.

Keywords: Urban mobility, Robotaxis, Autonomous vehicles, Milano, Shared mobility, Car-sharing

Abstract in lingua italiana

La guida autonoma è una tecnologia destinata a rivoluzionare la mobilità nei prossimi anni, aprendo nuove opportunità nella mobilità urbana e facilitando il passaggio da un modello di proprietà privata dell'automobile a uno condiviso pubblicamente. Questa tesi mira a sviluppare e analizzare un sistema di mobilità urbana multimodale mediante l'utilizzo dei robotaxi, veicoli elettrici autonomi condivisi tra gli utenti. Per realizzare questo obiettivo, è stato costruito un simulatore basato su dati di percorrenza reali ottenuti tramite dispositivi GNSS, installati direttamente sui veicoli dei clienti di UnipolSai Group, risolvendo un problema di ottimizzazione tramite la teoria del Co-design. Il simulatore determina il numero ottimale di robotaxi necessari per soddisfare le richieste di viaggio nell'area di servizio durante una settimana, analizzando tempi di attesa, di percorrenza e altri parametri utili a valutare il servizio. Sono stati analizzati due casi distinti. Nel primo, si è progettato un sistema concorrenziale rispetto alle attuali aziende di car-sharing. I risultati dimostrano che sarebbe notevolmente più efficiente ed economicamente competitivo rispetto alle compagnie esistenti. Nel secondo scenario, è stata ipotizzata una limitazione della circolazione delle auto private nell'area B della città, con una mobilità urbana composta esclusivamente da mezzi pubblici e soluzioni di mobilità condivisa. Il numero di robotaxi necessari sarebbe da 10 a 50 volte inferiore rispetto al numero di auto private attualmente in circolazione. Questo permetterebbe di riqualificare vaste aree destinate a parcheggi, creando spazi verdi e aree utili per la comunità, senza considerare i vantaggi portati da una mobilità 100% green. Anche a livello economico, per la maggior parte della popolazione, l'utilizzo dei robotaxi risulterebbe più conveniente rispetto all'auto privata. È auspicabile che questo tipo di sistema ottenga successo nei prossimi anni, seguendo l'esempio di città come San Francisco e Wuhan, che stanno già sperimentando l'uso di robotaxi, seppur non ancora su larga scala.

Parole chiave: Mobilità urbana, Robotaxi, Veicoli autonomi, Milano, Mobilità condivisa, Car sharing.

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

List of Figures **ix**

List of Tables **xiii**

1 Introduction **1**

1.1 Problem statement: understanding role and effects of urban, shared driverless mobility from data 4

1.2 State of the art 6

1.3 Main contributions 9

1.4 Structure of the thesis 10

2 Monotone co-design theory and adaptation to a future urban tri-modal transportation system **13**

2.1 Monotone co-design theory 13

2.1.1 Elements and functional relationships 14

2.1.2 Properties 14

2.1.3 Definition of optimization problems and their solution 17

2.2 Co-design of a future urban tri-modal transportation system 18

2.2.1 The interchanging transportation model 18

2.2.2 The optimization problem in the co-design framework 22

3 Description of robotaxis and their charging model **23**

3.1 On-demand autonomous vehicles (robotaxi) 23

3.1.1 Description and features 23

3.1.2 Costs and investments . . . 24
3.1.3 Parameters chosen for the model . . . 27
3.2 The robotaxi recharging infrastructure . . . 28
3.2.1 Features and costs . . . 28
3.2.2 Recharging model . . . 30
3.2.3 Placement of charging stations . . . 33

4 Data-driven experimental modeling of a robotaxi fleet co-designed with pedestrians and public transport 37
4.1 The real driving dataset for inferring people movement . . . 37
4.1.1 Data description and preprocessing . . . 39
4.1.2 Data spatial clustering . . . 42
4.2 The data-driven street network approximation . . . 45
4.2.1 On-demand autonomous mobility graph . . . 46
4.2.2 Walking and public transportation graphs . . . 48

5 Simulation of the tri-modal urban transportation system 53
5.1 Structure and operation of the simulator . . . 53
5.1.1 Introduction of the cache memory for distances . . . 59
5.1.2 One-layer robotaxi on-demand simulation . . . 61
5.1.3 Two layer on-demand robotaxi, pedestrians and public transport simulation . . . 62
5.2 Optimization problem design and convergence . . . 64

6 Results of the co-design of a urban on-demand robotaxi fleet cohabiting with walking and subway transportation 69
6.1 Simplified setting: one mobility layer . . . 69
6.1.1 Analysis of the resulting sizing of AVs fleet . . . 71
6.1.2 Economic analysis of service costs and pricing . . . 76
6.1.3 Study of the ratio between the robotaxi fleet size and the number of private cars replaced . . . 78
6.2 Interconnected mobility: two layers of robotaxi, walking and underground . 84
6.2.1 Cost and efficiency analysis of different AV fleets . . . 87
6.2.2 Economic analysis and study of the benefits of transitioning to shared autonomous mobility . . . 93

7 Conclusions and future developments 97
7.1 Main Contributions and Results . . . 99

7.1.1 Main Contributions 99
7.1.2 Results of one-layer and two-layer mobility model 101
7.2 Final remarks 102
7.3 Future developments 103

Bibliography **105**

A Appendix **109**

Acknowledgements **121**

List of Figures

1.1 Urban and rural population estimation in the world from 1940 until 2050. . 1
1.2 Transport emissions compared to total greenhouse gas emissions in EU. . . 3
1.3 Map of Milano with the 6 km circulation service area highlighted. 5
1.4 Levels of automation of an autonomous vehicle. 7
1.5 Percentage of fully electric cars sold in Italy compared to total cars. 8

2.1 Example of MDPI: Transducers. 15
2.2 Example of MDPI: Motor design. 15
2.3 Example of MDPI: Computer CPU. 15
2.4 Example of MDPI: Bin packing problem. 16
2.5 Three example of composition of different MDPIs. 16
2.6 Example of CDPI: Chassis design connected with motor design. 16
2.7 The ceiling function is Scott continuous. 18
2.8 The trimodal layers used in our model. 19
2.9 Scheme of AVs MDPI. 20
2.10 Scheme of the complete CDPI. 22

3.1 Revenues prediction for AVs in the next decade. They are expected to reach $300-400 billion by 2035, marking an increase of over 400% compared to 2022. The majority of these revenues (over 50%) will come from Level 4 automated vehicles, which are not yet in production. 25
3.2 A Cruise's robotaxi operating in San Francisco. 27
3.3 Example of a public recharging column. 29
3.4 Scheme of the process used for the recharging model. 32
3.5 The city of Milano divided in 500x500m blocks. 34
3.6 Placement of 2,000 charging columns in the corresponding charging stations. 35

4.1 Example of Unibox. 37
4.2 Some rows of the dataset. 38

4.3 Example of trips: only the red cases are completely inside the service are and kept in our analysis. The blue cases have start or end point outside, the black cases have both outside. 39
4.4 Distribution of the trips with respect to the service area. The trips used in the model are highlighted. 40
4.5 Distance, duration and average speed of the trips inside the 6 km radius. On the abscissa are reported the % of trips in the correspondent category. . 40
4.6 Distance-duration scatterplot of the trips inside the 6 km radius. 42
4.7 DBScan example with EPS=eps and MinPts=3. 44
4.8 Result of the DBScan applied to our dataset, with EPS=100m and MinPts=10. Every circle represent a different cluster, the radius of the circle is proportional to the cluster's size. 45
4.9 Image of the obtained graph for the city of Milan, every blue line corresponds to an edge. The red lines are the edges added later to connect all the components. 47
4.10 Paths of the 3 trips referred to in Table 4.2. 48
4.11 Representation of the 5 metro lines in the city of Milano. Only the stations inside the defined service coverage area were considered in the model. . . . 49
4.12 Paths of the 3 trips shown in Table 4.3. 51

5.1 Diagram representing the simulation process. 59
5.2 Evolution of the number of daily entries in the Area C of the city of Milano. 63
5.3 Example of optimization with Binary Search algorithm in a range from 0 to 50. A green value means that the constraints are satisfied, a red one that the constraints are not satisfied. The optimum value is obtained after 6 iterations. 66

6.1 Variability of the average customer waiting time for a robotaxi. Above are the results obtained for the entire week, with vertical red lines indicating the start and end of the weekend. Below is a focus on daily waiting times, specifically on 19/5/2022. 72
6.2 Variability of the 'missing call' percentage over the entire week. The vertical red lines indicate the start and end of the weekend. 73
6.3 Variability of available cars over time. Above are the results obtained for the entire week, with vertical red lines indicating the start and end of the weekend. Below is a focus on daily waiting times, specifically on 19/5/2022. 75
6.4 Cost allocation for the robotaxi service. 77

6.5 The three different service areas. In green the 3.5 km radius, in red the 6 km radius and in blue the 14 km radius. 80
6.6 Growth of the ratio between size of AVs fleet and number of substituted cars with respect to the radius of the service area. The orange lines indicate a 1/10 ratio, showing where the fleet size is significantly smaller compared to the total number of today cars as the service area expands. 83
6.7 Percentage of users that reach their destination walking or using underground, with respect to the fleet size of robotaxis used in the simulation. . 86
6.8 Trend of Δ time with respect to the fleet size of robotaxis used in the simulation.The function seems to approach a horizontal asymptote after the use of 1600 AVs. 87
6.9 Variation in costs for the robotaxi service and for public transportation with changes in the AVs fleet size. 90
6.10 Variation in costs for the robotaxi service and for public transportation with changes in the AVs fleet size. 90
6.11 Weekly data for the simulation obtained with 1600 AVs. 92
6.12 Comparision between the cost for the user per km of using robotaxis service or driving the private car. 94

List of Tables

1.1 Population projection for the city of Milano. 2

3.1 Features and costs for different models of full electric vehicles. 26
3.2 Power and associated cost for different types of charging columns available on the market. 30

4.1 Table reporting the mean, median, and standard deviation of the three variables of the trips within a radius of 6 km. 41
4.2 Duration of 3 sample different trips computed with Google Maps and our model (with average speed set at 16 km/h). 47
4.3 Duration of 3 different trips computed with Google Maps and our model. . 50

6.1 Results of the simulation with a service area of 6 km radius. 70
6.2 Results of the simulation with a service area of 6 km radius and the 'missing calls' threshold modified at 8 minutes. 70
6.3 All the costs incurred in the two one-layer simulations conducted. 77
6.4 Proposed rates for the robotaxi service. 78
6.5 Results of the simulation with 3.5 km, 6 km and 14 km radius in Scenario 1 are presented. The ratio between the robotaxi fleet and the number of private cars substituted by them is highlighted. The original private cars are those that the robotaxi service can substitute. 80
6.6 Results of the simulation with 3.5 km, 6 km and 14 km radius in Scenario 3. The ratio between the robotaxi fleet and the number of private cars substituted by them is highlighted. 81
6.7 Results of the simulation with different AVs fleet size. The 'Average Δ time' parameter indicates the differences (in mean) between the actual average travel time obtained using private cars and the average travel time with this robotaxi service. 85
6.8 All the costs incurred in the two-layer simulations conducted. 88
6.9 Total weekly costs of the two-layer simulation divided by categories. 89

1 | Introduction

According to projections, the urban share of the world population will grow to 6.5 billion, an increase of 65%, by 2050. Currently, more than half of the population resides in urban areas (58%) and this trends will continue to increase [1]. It is evident that such a significant numerical change will impact various aspects of urban life. Many cities already face multiple issues, including overcrowding, air quality, and urban mobility.

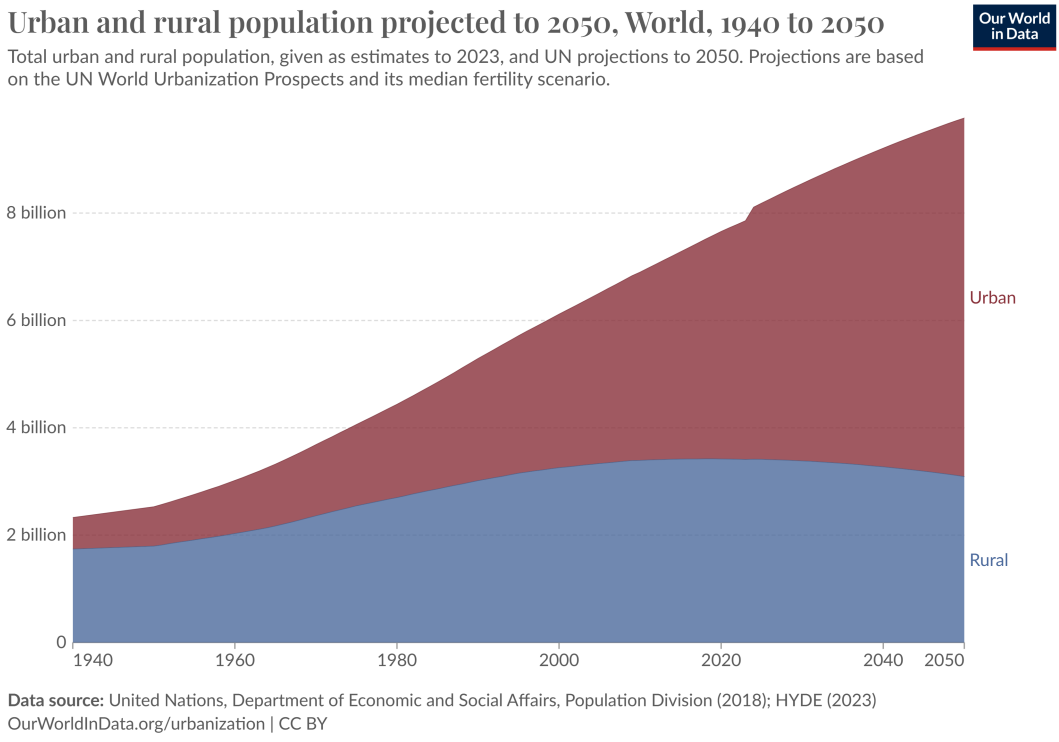


Figure 1.1: Urban and rural population estimation in the world from 1940 until 2050.

Focusing on the case study of the city of Milano, a 5.9% increase in population is expected over the next 15 years [2], translating to an increase of nearly 100,000 individuals. This growth will further exacerbate the already severe issues of private mobility and traffic congestion in the metropolis.

Year	Population	% increase relative to today
2024	1,368,738	-
2030	1,404,597	2.62%
2035	1,427,950	4.27%
2040	1,449,500	5.90%

Table 1.1: Population projection for the city of Milano.

Furthermore, Milano is situated in one of the areas with the poorest air quality in Europe, a condition partly attributable to the region's geography. This poor air quality has significant health implications for residents, contributing to higher incidences of cancer and respiratory issues. For years, local administrations have been striving to limit car usage within the urban area, starting with the introduction of Area C in 2012, followed by Area B in 2019, which restricts the circulation of the most polluting vehicles and heavy trucks [3]. In recent years, many multi-lane roads have been reduced to create space for bicycle lanes, and 30 km/h zones have been introduced. These measures have polarized public opinion. Some residents welcome the transition to a greener, more sustainable city that favors alternative mobility forms, while others are frustrated by the inconveniences these measures cause, such as increased urban traffic and circulation issues, mainly caused by the narrowing of the lanes.

While these changes may be challenging for some residents to accept, they represent a step towards more modern and environmentally sustainable mobility, aimed at ensuring greater health, safety, and efficiency in transportation for citizens. The transport sector is responsible for 25% of the EU's green house gas emissions, with a significant portion of them stemming from road transport (71%). The European Union mandates that by 2050, emissions, both CO_2 and polluting emissions like NO_X and PM, in this sector must be significantly reduced by more than 90% [4].

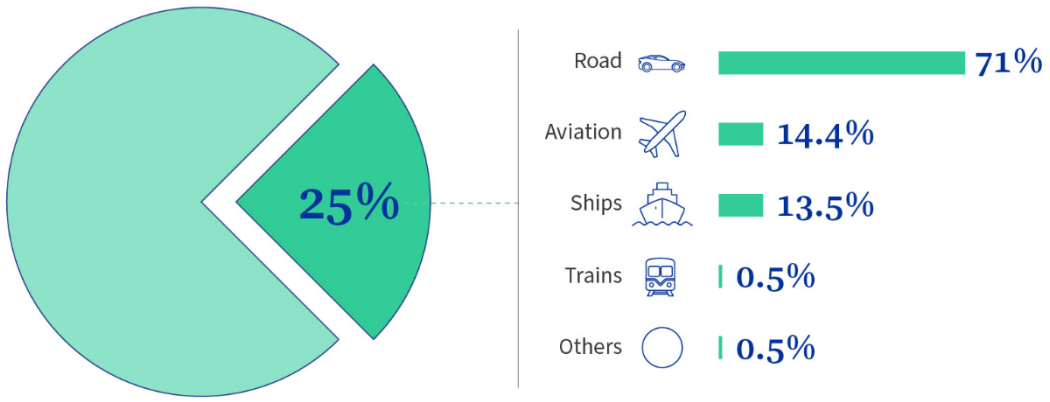


Figure 1.2: Transport emissions compared to total greenhouse gas emissions in EU.

Restrictions on private vehicle use can also positively impact urban mobility. Milano ranks fourth in Europe for time spent in traffic, with 137 hours per year spent for each vehicle. Additionally, it has been estimated that 20-30% of the time spent in cars is dedicated solely to searching for parking, amounting to about 30 minutes per person per day.

Solutions to this problem are multifaceted: they include the aforementioned regulations introduced in recent years, the establishment of zones with complete traffic bans, enhancement of public transportation, funding incentives for the use of electric or hybrid cars, and the implementation of more innovative and modern technologies such as robotaxis, i.e. fully electric autonomous vehicles (AVs).

This latter technology is already being widely tested and will likely become a common sight in our cities within a few years. On August 10, 2023, the city of San Francisco introduced a public service of this kind, with autonomous electric robotaxis for the transportation of citizens allowed to operate 24 hours.

Data on the impact of this service, where it is already in use, is not yet available. However, several studies suggest that it could lead to a significant reduction in the number of vehicles in circulation. This is because shared transportation allows a single vehicle to meet the needs of a large number of private citizens. Additionally, robotaxis would not spend a significant portion of their time parked like private cars. It is estimated that current vehicles spend more than 90% of their time parked. Moreover, it could result in reduced travel times and costs. The service would completely eliminate time spent searching for parking and reduce time spent in traffic, thanks to better traffic management facilitated by au-

tomation. Moreover, shared transportation would lower individual mobility costs, making car ownership non-essential for some citizen. This would also decrease fuel consumption due to both improved driving efficiency (4-10% less [5]) and a quicker transition to electric vehicles. Charging methods would also benefit: currently, owning a private electric car has disadvantages due to long charging times. Shared transportation would optimize this process by sending more vehicles for charging during periods of lower demand.

1.1. Problem statement: understanding role and effects of urban, shared driverless mobility from data

In this thesis, our aim is to address the issue of a new paradigm of urban mobility by studying the potential implementation of a robotaxi service in the city of Milano.

Our simulation endeavors to be as realistic as possible, built upon real data. These data are provided by the insurance company Unipol group, which has granted us access to a comprehensive database of trips made by their clients within the province of Milano.

Given this, the objective is to address several straightforward questions:

- How would an "on-demand" autonomous mobility servece (robotaxi) operate within the urban area of Milano?
- What would be the efficiency of the robotaxi service in terms of fleet size and waiting time for users?
- What would be the costs incurred both by users and by a provider of this service?
- What would be the advantages and disadvantages of this new type of autonomous "on-demand" urban mobility?
- How could a robotaxi service be designed and integrated into existing public transportation?

The work is therefore divided into two parts: the first involves constructing the data-driven simulated scenario, introducing a series of models and approximations that do not compromise its realism. The second part entails an analysis of the results to understand the effects of the service and the associated costs, thereby designing the optimal way to design it. The idea is to assume the perspective of the service provider, putting ourselves in the shoes of the municipality or a company that wishes to plan offer the service.

As mentioned earlier, the available data covers the entire province of Milano, but the proposed service may only operate within a much smaller urban area. For this reason, we concentrate on a 6 km coverage area, calculated from the city center, taken as Piazza del Duomo (Figure 1.3). The concept is to envision a city where, within what is currently the Area B, private vehicle circulation is prohibited, introducing the robotaxi service alongside the existing public transportation one.

Figure 1.3: Map of Milano with the 6 km circulation service area highlighted.

In addition to utilizing real data on private vehicles, another significant contribution of this thesis is the design and construction of a city graph to simulate the road network. The robotaxis then navigate these streets at a constant speed, simulating very realistically what would occur in reality. Concerning public transportation, real metro stations already in operation with their actual locations and train schedules from the ATM website [6] have been used. As mentioned earlier, surface public transportation vehicles have not been introduced because they would have required the implementation of an extensive network of starts and stops locations, without significantly altering the results of this study regarding the previously mentioned objectives.

1.2. State of the art

Autonomous driving has seen significant expansion in recent years, transitioning from a futuristic concept to a present-day reality. It's a technology with which we will become increasingly familiar in the years to come. These types of vehicles have long fascinated humans, with research into their development spanning many years. The first example of a driverless car dates back to 1925 when the Houdina Radio Control company in New York showcased a radio-controlled vehicle that took a demonstrative spin in the city. In more recent years, the first fully autonomous vehicle is a Mercedes-Benz van known as "VaMoRs", developed in Germany in 1986 [7].

Since then, investments of automotive manufacturers have become increasingly significant. Prototypes of autonomous vehicles have been developed by companies like Mercedes-Benz, General Motors, Toyota, Renault, Nissan, and others, including Google. Estimates suggest that there have been investments in the sector totaling hundreds of billions of dollars. California was the first state in the United States to authorize urban testing with fully automated vehicles without human drivers, starting from April 2, 2018 and it continues to be a pioneering state in this sector. On August 10, 2023, Alphabet's Waymo and General Motors' Cruise received approval from the California Public Utilities Commission to operate their fleets of robotaxis in the San Francisco area, thus competing with traditional taxi services and Uber [8]. Recently, Elon Musk, the billionaire entrepreneur and owner of Tesla, announced that his company would unveil its own line of robotaxis on August 8, 2024 [9]. These vehicles appear to have potential use in China, as the tech giant Baidu has shown strong interest. This company already offers a similar service in some cities in its home country and according to their estimates, they aim to serve more than 100 cities by 2030 [10]. A notable case is Wuhan, a city of 12 million inhabitants, where by the end of 2024, 1000 next-generation vehicles will be traveling, produced at lower costs, approximately $28,000, which is 50% less than previous prices [11].

Naturally, opposition to these introductions exists, ranging from taxi drivers fighting against the potential low-cost competition to more skeptical individuals who raise concerns about safety and road traffic. A 2021 survey by the AAA (American Automobile Association) revealed that 71% of respondents did not trust to ride in a fully autonomous car, with the percentage rising to 79% among women [12].

A major initial hurdle to overcome will be convincing the public, which is still largely reluctant to embrace this technology. However, data reveals a different reality. In the EU, there were 22,660 deaths in 2019, and 94% of fatal car accidents are attributable to human error. People have a high perception of safety when in cars, partly due to their

habitual use, and this perception may not be the same within a fully automated vehicle. Moreover, any potential road accident caused by such a vehicle would likely be met with significant public backlash. Consider the controversy sparked by an incident on October 2, 2023, when a Cruise's vehicle hit a woman: although the accident was actually caused by another human-driven vehicle, protests led to Cruise's temporary suspension within the state [13]. Distrust stems from various factors: limited understanding of the topic and associated technology, enjoyment derived from driving, or a desire for control over the vehicle and the external situation [14].

In addition to safety, another crucial aspect is legislation. It's important for countries to legislate on the issue of potential accidents caused by autonomous vehicles and on the behaviors and liabilities to be attributed. On July 14, 2022, Europe approved the circulation of autonomous vehicles on roads. It's now up to individual countries to adapt to the new directives. Italy, for its part, hasn't made many interventions in this regard. Currently, it's allowed for level 1-2 autonomous vehicles to circulate, which means vehicles with simple driver assistance systems (such as cruise control, ESC, brake supports, lane-keeping systems) and vehicles with partial autonomy that allow the driver to relinquish control in limited situations, such as constant speed stretches on highways [12]. In total, there are 6 levels of automation, represented in Figure 1.4 [15].

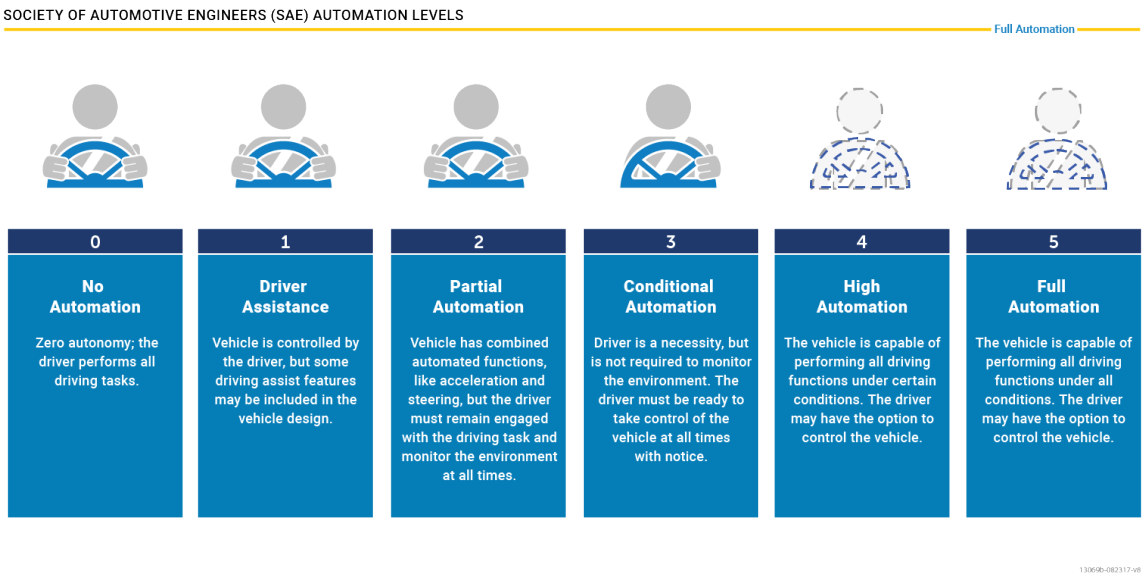


Figure 1.4: Levels of automation of an autonomous vehicle.

Furthermore, in our country, the "Codice della Strada", drafted in the pre-digital era,

defines vehicles as all machines circulating on the road driven by humans, thus excluding autonomous vehicles a priori. In 2018, the "Decreto Smart Road" introduced new rules for the experimentation of autonomous vehicles on public roads. The decree allows vehicles to circulate as long as there is a suitably trained supervisor present who can switch to manual mode at any time [16].

Another challenging issue for us Europeans to tackle is that of electric cars: the transition seems inevitable, but the predictions of recent years have not been met despite incentives. In Italy, in particular, sales did increase in 2023, but only 4.2% of the total vehicles sold were fully electric, compared to some predictions from previous years that had already placed us at 9% (Figure 1.5). In Europe, our country is among the worst performers in these rankings: the European average in 2023 was 14.6% of electric cars sold out of the total, with Norway even reaching 82.4%. Currently, only 0.3% of the cars in circulation in Italy are fully electric, well below the European average of 0.7% [17].

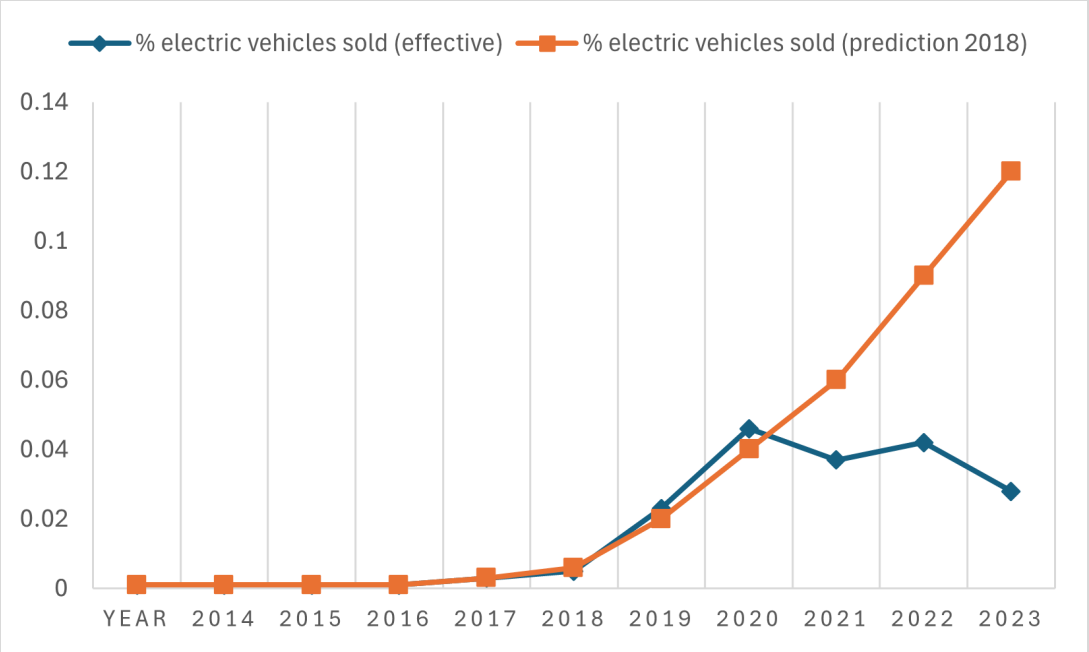


Figure 1.5: Percentage of fully electric cars sold in Italy compared to total cars.

However, a change will be necessary in the coming years, as also mandated by the new European regulations. The goal is to make the EU climate-neutral by 2050. To achieve this, emissions from the transport sector will need to be cut by 90%. But even in the shorter term, there are constraints. The "Fit for 55%" package includes cuts to be made by 2030 in all economic sectors, to ensure a fair and socially equitable transition and to maintain and strengthen EU innovation and competitiveness in this sector. For examples,

it aims to reform the EU ETS, a carbon market based on a system of emission allowances trading for high-energy-intensive industries and the energy production sector, making it more ambitious. This market was introduced in 2005 and it has already contribuited to a 41% reduction in emissions [4].

A system like the one studied in this thesis, based on shared and electric transportation, could facilitate the transition in our country. Private individuals are hesitant to switch to green vehicles, likely due to range anxiety, high costs and lack of trust in associated infrastructure, such as the limited availability of charging stations in urban centers or the slow charging process, which is not well-suited for short and frequent trips. On the other hand, the transition to electric vehicles in public transportation has long been underway, and users do not encounter any issues because they do not have to deal with these concerns: in Milano, the aim is to have a fleet of buses, trams, and metropolitan trains that are 100% green by 2030, with the percentage already exceeding 80% today. Shared electric mobility would shift the challenges of charging away from the individual to the service provider. With access to a large fleet and greater expertise, service providers would encounter fewer problems than private individuals in managing charging infrastructure. Consequently, users would simply enjoy a convenient service, which in most cases, is more economical than using private cars.

1.3. Main contributions

The goal of this thesis is to build a data-driven model of a tri-modal urban mobility network that predicts how an autonomous robotaxi service could operate alongside the main existing traditional public service. Indeed, predicting the costs of such an innovative service upfront is challenging, starting from the size of the definition of the fleet needed to manage such a large number of users.

For this reason, two different models have been created. In the first one, the simulation is carried out with a lower number of requests, aiming to satisfy a percentage of users above a certain threshold, in order to understand what would be the optimal number of autonomous vehicles to use for partially shared mobility, where private vehicles would still be in circulation. The idea is to implement a robotaxis service that is competitive with the car-sharing services currently operating in the city, comparing their efficiency. Desides, this allows us to calculate the number of private vehicles that shared mobility could replace. In this case public transports are not implemented. The second model makes stronger assumptions. It simulates a scenario in which private mobility is prohibited within Area B and is entirely replaced by shared mobility, consisting of robotaxis and

public transport. To achieve this, a much larger amount of user travel request data is utilized, aiming to obtain a number of trips that closely approximates the actual number of daily movements within the considered area of the city of Milano. This could be a very useful tool for studying the feasibility of the project, the effect that any slight modifications would have, its costs, and the advantages and disadvantages for citizens.

Other similar projects have been carried out in recent years, most of them focusing on different cities with diverse characteristics, such as the work by Provera [18] in Brescia and by Zardini & al. [19] in Washington. However, our model has additional features: first, this is a tri-modal model that integrates public and pedestrian transportation, providing a more realistic simulation and allowing us to study the changes that would occur in their usage as well. It can be easily expanded by introducing more detailed public transportation or a greater variety of micro-mobility options, without making significant modifications to the simulator's structure. Second, it utilizes a city graph that seeks to realistically reproduce the actual road network, thereby enhancing its realism. The difference from the project by Zardini & al. [19] lies in the fact that here, a model for electric vehicle charging is also implemented. When the battery is below a certain threshold, the autonomous vehicles will automatically go to designated charging stations, strategically positioned within the city based on the requests made.

For an organisation, it is also straightforward to derive the costs and efficiency of robotaxis from our model. A thorough analysis is conducted where, for each vehicle, the total kilometers traveled over an entire week, the number of calls, the total recharges, and the time spent at charging stations are displayed. It is also easy to observe how the request of robotaxis varies throughout different hours and days of the week. For users, the average travel time before and after the introduction of the service is shown, allowing for the study of any differences.

The thesis aims to be a useful study for an organisation, whether public or private, intending to provide the service in the future. It allows for the examination of the investments based on the number of requests to be satisfied.

1.4. Structure of the thesis

The thesis is structured into the following sections:

- In Chapter 2, the foundations of co-design theory are introduced, upon which the simulation construction is based. This theory allows us to build a complex problem step by step, starting from simpler sub-problems, and ensures the attainment of an

optimal value of resoursces to solve it. It is also explained how co-design theory can be applied to the case of urban mobility. The sub-problems are described along with their peculiar characteristics and connections, leading to the definition of the complete problem.

- In Chapter 3, the autonomous vehicles used in the simulation are detailed, highlighting their characteristics and costs. The charging model employed is then introduced, starting from its mechanism and proceeding to the study of costs and the positioning of charging stations.
- In Chapter 4, the dataset forming the basis of the work, containing real user data from Unipol group in the city of Milano, is introduced. Following this, various operations conducted on the data are explained, ranging from initial qualitative analysis to preprocessing and clustering. Finally, the urban graph used for the simulation is introduced, encompassing both the network for robotaxis and that for public transportation and pedestrians. The techniques employed to create it from real data and the assumptions made are elaborated upon.
- In Chapter 5, the operation of the tri-modal urban mobility simulator is examined in depth, detailing the mechanisms upon which it is based. Lastly, an analysis of its convergence is conducted.
- In Chapter 6, the results obtained are described. Firstly, those derived from a simplified simulation, without the introduction of public transportation and with a reduced dataset, are presented. Subsequently, public transportation is introduced, and a number of requests closer to that of a city utilizing only this mode of mobility, replacing private transportation, is used.
- In Chapter 7, conclusions and analyses of the obtained results are presented, along with suggestions for future developments that could expand upon this project.

2 | Monotone co-design theory and adaptation to a future urban tri-modal transportation system

In this chapter, we will describe the theory of co-design, addressed in [19], using some properties described in [20] and how can we adapt that to our case study, a future urban tri-modal transportation system.

After that, we will describe the application of this theory to our case, namely urban mobility. We will identify the various sub-problems along with their resources and functionalities, verify their monotonicity, and integrate them correctly.

2.1. Monotone co-design theory

Let's begin with an introduction to co-design theory, highlighting its characteristics, main properties, and areas of application.

For this reason, we will need to introduce a series of specific definitions to maintain the precise mathematical formality of the theory.

This will allow us to more easily demonstrate the existence of an optimal solution to our problem, which, being highly complex, is not trivial.

The idea lies in the fact that it's possible to start from simpler sub-problems, called design problems with implementation (DPIs), and obtain the complex problem, co-design problem with implementation (CDPI), simply by integrating them together. To ensure the existence of a solution, we just need to check the monotonicity of the DPIs and some simple properties of the CDPI.

2.1.1. Elements and functional relationships

As mentioned earlier, the elements that compose our problem are DPIs. However, to describe them, we need some definitions.

Definition 2.1.1 (Poset). *A partially ordered set (poset) is a tuple $\mathcal{P} = \langle P, \preceq_{\mathcal{P}} \rangle$, where P is a set and $\preceq$ a partial order.*

Definition 2.1.2 (Monotone map). *A map $f : \mathcal{P} \to \mathcal{Q}$ between to posets $\langle P, \preceq_{\mathcal{P}} \rangle$, $\langle Q, \preceq_{\mathcal{Q}} \rangle$ is monotone iff $x \preceq_{\mathcal{P}} y$ implies $f(x) \preceq_{\mathcal{Q}} f(y)$.*

Definition 2.1.3 (DPI). *A design problem with implementation (DPI) is a tuple:*

$$\langle \mathcal{F}, \mathcal{R}, \mathcal{I}, prov, req \rangle$$

where:

- *$\mathcal{F}$ is a poset, called functionalies space;*
- *$\mathbb{R}$ is a poset, called resources space;*
- *$\mathcal{I}$ is a set, called implementation space;*
- *the map $prov : \mathcal{I} \to \mathcal{F}$ maps an implementation to the functionality it provides;*
- *the map $req : \mathcal{I} \to \mathcal{R}$ maps an implementation to the resources it requires.*

Definition 2.1.4 (MDPI). *Given the posets $\mathcal{F}, \mathcal{R}$, representing functionalities and resources, respectively, we define a monotone design problem with implementation (MDPI) as a tuple $\langle \mathcal{I}_d, prov, reqs \rangle$, where $\mathcal{I}_d$ is the set of implementations, and prov and reqs are functions from $\mathcal{I}_d$ to $\mathcal{F}$ and $\mathcal{R}$, respectively:*

$$\mathcal{F} \overset{prov}{\leftarrow} \mathcal{I}_d \overset{reqs}{\to} \mathcal{R}$$

Furthermore, to each MDPI, we are able to associate a monotone map $\bar{d} : \mathcal{F}^{op} \to \mathcal{R}$, where $(.)^{op}$ reverses the order of a poset. The expression $\bar{d}(f*, r)$ returns the set of implementations $\mathcal{S} \subseteq \mathcal{I}_d$ for which the functionalities f are feasible with resources r.

2.1.2. Properties

In essence, an MDPI is nothing more than an object that receives resources as input and provides functionalities as output. The fundamental property is the monotonicity of the

associated map d, that we clarify in the following definitions:

Definition 2.1.5 (Functionalities to resources map). *Given an MDPI d, one can define a monotone map $h_d : \mathcal{F} \to \mathcal{R}$, mapping a functionality to the minimum amount of resources providing it. Dually, we can define $h'_d : \mathcal{R} \to \mathcal{F}$, mapping a resource to the maximum amount of functionalities $\mathcal{F}$.*

Possible examples of MDPI are:

- a transducers that bridges between different mediums (Figure 2.1).
- a motor for a robot (Figure 2.2).
- a computer CPU (Figure 2.3).
- The bin packing problem, that involves finding the minimum-sized container capable of accommodating a given number of objects, each with its own volume (Figure 2.4).

But we could continue this list with an infinite number of other examples.

Figure 2.1: Example of MDPI: Transducers.

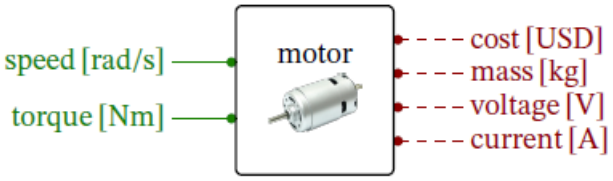


Figure 2.2: Example of MDPI: Motor design.

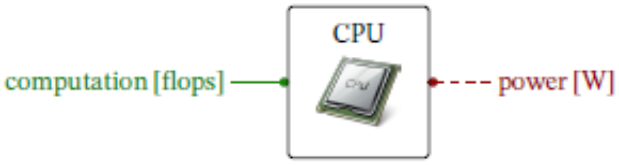


Figure 2.3: Example of MDPI: Computer CPU.

Figure 2.4: Example of MDPI: Bin packing problem.

The monotonicity of an MDPI can be formalized as follows:

Definition 2.1.6 (Monotonicity of MDPIs). *Consider an MDPI with* $\bar{d}(f^*, r) = \mathcal{S}$.

- *Consider* $f' \preceq f$. *Then* $\bar{d}(f'^*, r) = \mathcal{S}' \subseteq \mathcal{S}$.
- *Consider* $r' \succeq r$. *Then* $\bar{d}(f^*, r') = \mathcal{S}' \subseteq \mathcal{S}$.

To obtain a co-design problem, various MDPIs can be integrated together. This composition of MDPIs can occur in three ways: serial, parallel, or loop (Figure 2.5). The composition operation preserves monotonicity. We refer to the composition of MDPIs as a co-design problem with implementation (CDPI).

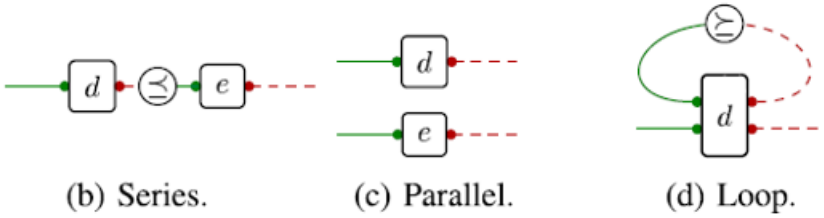


Figure 2.5: Three example of composition of different MDPIs.

In this way, starting from very simple sub-problems in which demonstrating monotonicity is straightforward, we can construct a complex problem that is also monotonic.

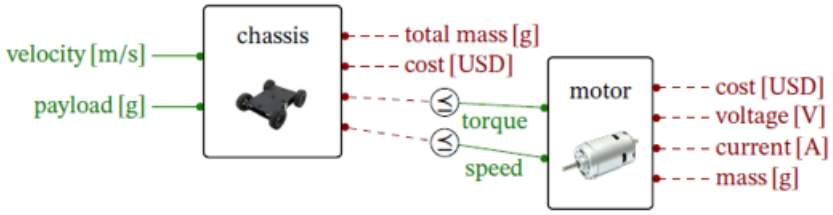


Figure 2.6: Example of CDPI: Chassis design connected with motor design.

2.1.3. Definition of optimization problems and their solution

Once the CDPI is created, the problem remains of ensuring that an optimal solution to the problem exists.

In essence, we can have two types of problems:

- given a set of functionalities $f \in \mathcal{F}$, we want to find the minimal set of resources $r \in \mathcal{R}$ that we need., in alternative we want to provide a certificate of infeasibility.
- given a set of resources $r \in \mathcal{R}$, we want to find the maximal set of functionalities $f \in \mathcal{F}$ that we are able to provide.

Ours will be the first case: given the functionalities that we need to achieve (for example, a certain minimum efficiency of the mobility service provided), we want to understand the minimum resources required to invest (such as the costs for purchasing the fleet of AVs and for the operation of public transportation).

Formalizing, given the maps h_d of the various sub-problems, we want to understand how to find the map h of the complete CDPI.

At this point, we first need to introduce some other definitions.

Definition 2.1.7 (Directed set). *In a poset $\mathcal{P} = \langle P, \preceq_{\mathcal{P}} \rangle$, we say that a set $\mathcal{S} \in \mathcal{P}$ is directed if each pair of elements in $\mathcal{S}$ has un upper bound. For each pair $x, y \in \mathcal{S}$, there exists $z \in \mathcal{S}$ such that $x \preceq z$ and $y \preceq z$.*

Definition 2.1.8 (Completeness). *A poset is a directed complete partial order (DCPO) if each of its directed subsets has a supremum (least of upper bounds). It is a complete partial order (CPO) if it also has a bottom.*

Definition 2.1.9 (Scott continuity). *A map $f : \mathcal{P} \to \mathcal{Q}$ between DPCOs is Scott continuous iff for each directed subset $\mathcal{S} \in \mathcal{P}$, the image $f(\mathcal{S})$ is directed, and*

$$f(\sup \mathcal{S}) = \sup f(\mathcal{S})$$

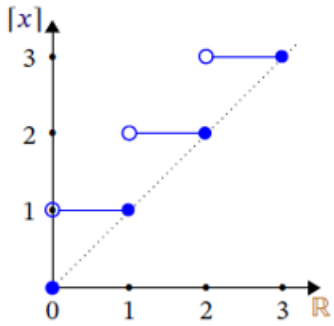


Figure 2.7: The ceiling function is Scott continuous.

Now if we have a map h that is Scott continuous and the posets are complete partial orders, we can rely on Kleene's fixed point theorem to find the solution. In essence, under these conditions, the existence of the optimum is ensured.

2.2. Co-design of a future urban tri-modal transportation system

In this section, we will begin by describing our case study, demonstrating how it is well-suited for the application of co-design theory. We will start by detailing the various MDPI components that comprise it, progressing towards the construction of the complete CDPI. We will verify how the solution is assured through this process.

2.2.1. The interchanging transportation model

To begin, it is necessary to understand how movements occur within our simulation. In a modern city like Milano, travel is facilitated by private vehicles, a network of surface public transportation, five metro lines, and a suburban train network. Additionally, there are options for light mobility, such as scooters, eco-scooters, and bicycles, as well as the possibility of traveling on foot.

In our simplified model, we will not account for all these options but will focus only on the main ones. Therefore, we will exclude the option of light mobility (except for walking) and disregard surface public transportation, resulting in a less efficient network compared to the current one. The analysis is focused on the network of autonomous vehicles replacing private ones, so this simplification does not significantly impact our results.

Layers definition

We have thus constructed a model with three different layers:

- On-demand autonomous mobility layer (AVs layer)
- Public transportation layer
- Walking layer

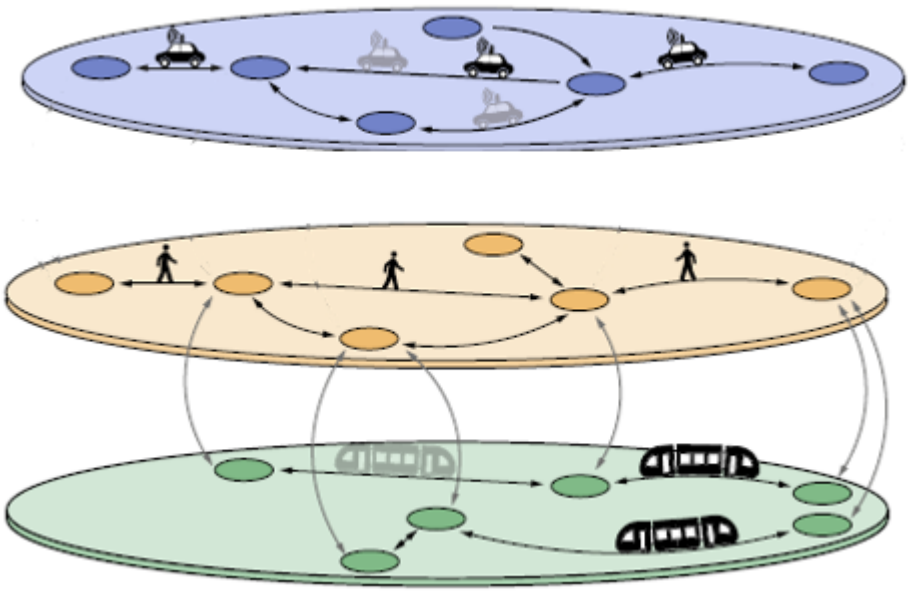


Figure 2.8: The trimodal layers used in our model.

In the context of co-design theory, each of these layers is composed of various integrated MDPIs. These three layers are then integrated with each other, thereby forming our CDPI.

Moreover, as illustrated in Figure 2.8, constructing the three separate layers is not sufficient. It is also necessary to connect them, allowing a user to, for instance, walk and use public transportation to travel from point A to point B.

Let us now examine in detail the composition of the different layers.

The in-layer network graph

First, let us analyze the characteristics of the three layers individually. Each of these will have associated costs and different travel times.

1. **On-demand autonomous mobility (AVs) layer**

   This layer consists of a series of edges, each representing a road segment connecting two points. Each edge will be associated with a weight, indicating the time required to traverse it, which depends on the speed of the AVs (described in Section 3.1.3). A certain number of vehicles will move here (to be optimized), incurring various costs: a cost for the purchase of each vehicle, a maintenance cost, and a charging cost. It is easy to demonstrate that this constitutes an MDPI: indeed, the more vehicles used, the higher all these costs will be (more initial purchases, more vehicles to maintain, more total kilometers traveled). In addition to these costs, there are also costs for charging stations; namely, the purchase cost of each station and the cost of renting the land. However, in our case, we have fixed the number of charging stations, so this does not fall under a variable cost and is not accounted for (Figure 2.9).

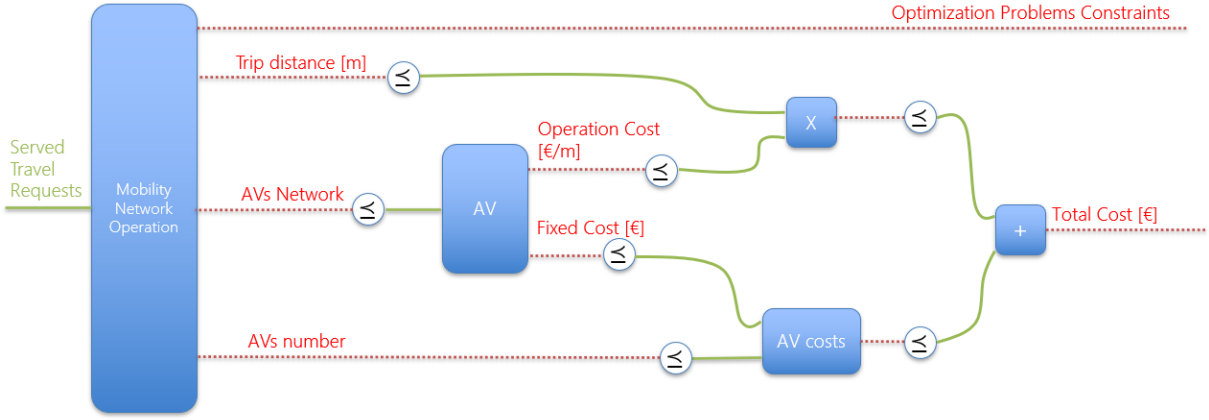


Figure 2.9: Scheme of AVs MDPI.

2. **Public Transportation layer**

   In this case, there are also a series of edges, but they only connect the stations of connected metro lines. The weight of each edge is again related to the duration of the journey from one station to another, based on real data from ATM obtained from [6], as detailed in Section 4.2.2. Additionally, stations served by two different metro lines are interconnected. The cost is calculated based on the number of users utilizing public transportation, as described in [21]. Specifically, we consider users who currently use private vehicles, so existing public transportation users will continue to use it, and additional costs will arise from these new users. Once again, monotonicity is evident: the more users utilize this service, the higher the costs will be.

3. **Walking layer**

   For the walking layer, the creation is simpler: the same edges as the AVs layer are used, but with different weight. Specifically, the time taken will be calculated based on the average walking speed of 4.8 km/h. As for the financial expenditure, it is null in this case.

The intra-layers network graph

The previously described layers alone are not sufficient to solve our problem. It is necessary to connect them together because users must be able to move not only using a specific method to reach their destination. This enhances the realism of the simulation. This connection is referred to as the intra-layers network graph.

In our case, we have decided to connect only the Public Transport and Walking layers, thereby reducing the complexity of the model. With these assumptions, a user can thus travel from point A to point B in two ways:

- using the AVs transportation service;;
- utilizing public transportation, walking or both of them;

hence, there is no option to use all three methods together, such as taking the metro for some stops and then continuing with autonomous vehicles.

The connection between the public transportation and walking layers is essential. Without it, it would not be possible to reach all destinations using only public transportation, as there is rarely a metro station corresponding to every possible starting and destination point, a correspondence that has been created in the other two graphs, as elaborated in Section 4.2.2. To create this connection, it was sufficient to link each station in the public transportation graph to the nearest vertices in the walking layer graph. The resulting edges have costs associated, like in the other cases, related to the travel time calculated based on pedestrian speed. Economically, there are obviously no expenses for traversing these segments.

The resulting model thus has two distinct layers:

- On-demand autonomous mobility layer
- Walking and Public transportation layer

2.2.2. The optimization problem in the co-design framework

The operation of the individual layers has now been clarified; it simply remains to describe how they are interconnected to constitute the CDPI.

The composition operation we employ in this case is of a parallel type: indeed, the essential and common resource across all layers is costs, which can be subsequently divided into fixed costs and operational costs. The former are associated with the purchase of the fleet, in our case, only the AVs fleet, and do not depend on their subsequent use. The latter, however, increase with the utilization of vehicles: some of these costs increase with the number of kilometers traveled by AVs, while others increase with the increased usage of metro lines. Thus, this is the connection that links these two layers. Our objective is to optimize costs (our resources), ensuring that minimal functionalities, namely a certain service efficiency, which we will describe later, are met.

The complete CDPI with all connections is described in Figure 2.10.

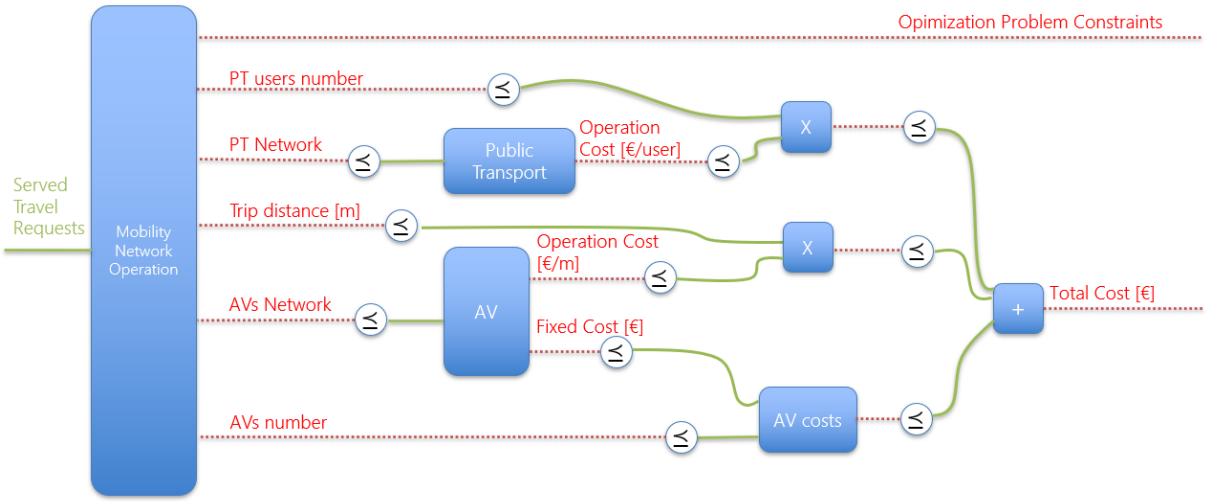


Figure 2.10: Scheme of the complete CDPI.

3 | Description of robotaxis and their charging model

This chapter contains a detailed description of the technology underlying the operation of robotaxis and their charging process. In our model, several parameters were selected for speed, battery capacity, costs, and other characteristics of these vehicles. The following pages explain the rationale behind these specific choices.

3.1. On-demand autonomous vehicles (robotaxi)

We will commence by delineating the autonomous electric vehicles selected for this simulation. This section will present the characteristics and improvements that have been made to these vehicles in recent times. This will enable a better understanding of the parameter choices that have been made in the simulation.

3.1.1. Description and features

The concept of autonomous vehicles (AVs) is fascinating because it revolutionises the way we think about mobility. An autonomous vehicle uses tools such as GPS, radar and cameras to send information to a central control system, which processes it to identify appropriate routes, obstacles and signposts.

This type of vehicle has many advantages, but also many doubts. First of all, there is comfort: the driver of the vehicle would become a mere passenger, and the time spent on the road could be used for other purposes, such as working on the computer or relaxing, watching a film or reading a book, for example. In addition, hours spent in traffic would be reduced through intelligent planning of autonomous traffic with communication between the various vehicles on the road, and the number of accidents would be reduced thanks to the electronic systems in these vehicles that optimise reaction times and detect obstacles or hazards that the human driver would not be able to detect in time, such as the blind spot detector. Estimates suggest that in Italy, it could be possible to reduce the current

annual death toll from 3,000 to just 330 victims, a significant improvement in terms of safety. Another benefit would be for the elderly or people with reduced mobility, who would be able to move around in these vehicles without assistance, whereas today they are often unable to drive conventional cars. The biggest questions concern problems that are currently difficult to solve. For example, how will the car react in a dangerous situation, will it better protect passengers or pedestrians and other cars? Another doubt concerns liability in the event of an accident, whether it will fall on the owner or the car manufacturer. Certainly, national regulations will need to address these important issues and provide clarity where there are gaps in the law. There are also doubts as to how these vehicles would function in adverse weather conditions: certain atmospheric phenomena, such as snow or rain, could interfere with the sensors fitted to the vehicle and cause many problems. The problem of human reluctance to use them also needs to be overcome. Certainly, at least in the first few years, people will have to be convinced of the benefits of these developments, and the initial reluctance of the masses to embrace this technology will have to be overcome. Finally, there is the issue of cost, which is still very high for the Level 3 and 4 automation vehicles currently already in production.

In any case, fully autonomous vehicles (Level 5) are not expected to be on the road until at least 2035. Until then, there will certainly be work to be done on all the more critical aspects of this technology.

3.1.2. Costs and investments

Today, there is still no vehicle on the road with full autonomy, i.e. Level 5. However, great progress has been made in recent years. Europe is a step behind other continents in this area, as only Level 2 autonomous cars are permitted in most member countries, including Italy. In China, the first car model with Level 4 autonomy, called JiYue Robotaxi 01, was unveiled in 2023. The robotaxis operating in San Francisco, owned by Waymo and Cruise, are also Level 4. Other car manufacturers such as Tesla, BMW, Mercedes and Honda have instead developed Level 3 automation systems and more advanced prototypes.

At present, production costs are still relatively high, with the cost of additional components alone for a Level 2 AV in the range of $1,500-$2,000, while for a higher level such as 3 or 4, the cost exceeds $5,000 per vehicle. However, investment is being driven by the potential players of the sector, and the former are estimated to reach $400 billion by 2035 (Figure 3.1).

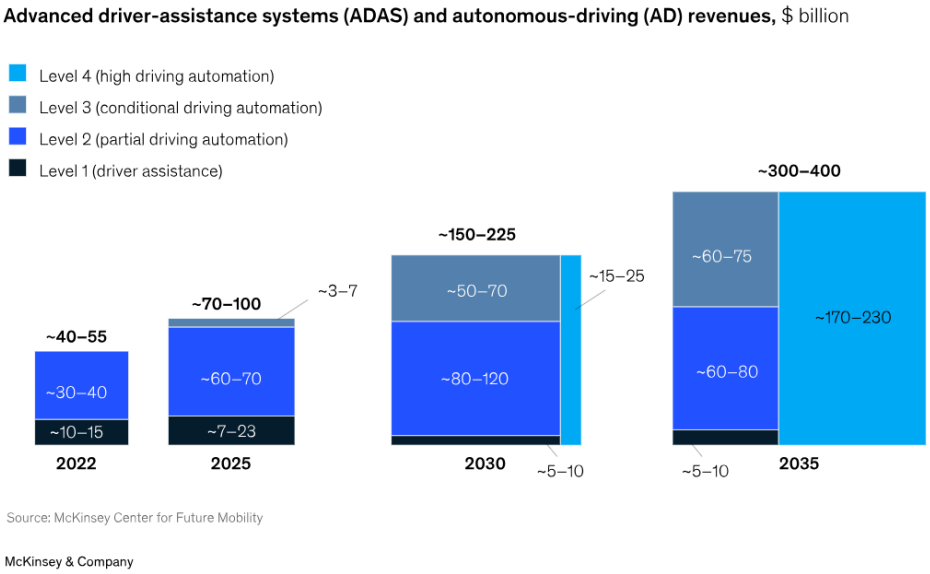


Figure 3.1: Revenues prediction for AVs in the next decade. They are expected to reach $300-400 billion by 2035, marking an increase of over 400% compared to 2022. The majority of these revenues (over 50%) will come from Level 4 automated vehicles, which are not yet in production.

Progress in electric vehicles is also remarkable. The global market is clearly expanding, with sales exceeding 15 percent of the global car fleet in 2023, rising to more than 50% in 2035 and 70% in 2040, thanks in part to the new fuel economy regulations that will drive growth. There have been significant improvements in vehicle performance: on average, the maximum charging power of the cars launched in 2023 reached 186 kW, an increase of 25 percent compared to the same figure in 2022. The average range of new cars will also exceed 470 km, which, combined with a wider distribution of charging points, could lead to greater customer confidence in relying on these vehicles. With the development of the next generation of batteries, the aim is to exceed the 1,000 km range [22].

The costs themselves are getting down, thanks in part to the many government incentives for buying this type of car. In Table 3.1 we show the costs and characteristics of some full electric cars.

Model	Battery Capacity [kWh]	Consumption [kWh/km]	Cost [€]
Abarth 500e	37.3	0.235	37,950
Audi Q8 e-tron	83.6	0.210	82,100
BMW i5	81.2	0.189	74,400
Chevrolet Bolt	66	0.176	29,870
Citroen e-C3	44	0.1375	23,900
Citroen e-C4	50	0.159	38,000
Fiat 500e	42	0.159	29,950
Fiat Topolino	5.4	0.075	9,890
Jaguar I-Pace	90	0.196	55,000
Hyundai Kona	64.8	0.175	42,000
Lancia Ypsilon	51	0.145	39,500
Mercedes EQE	90.6	0.191	77,300
Opel Astra	54	0.15	39,900
Peugeot e-208	50	0.12	35,500
Renault Megane	45	0.123	38,000
Tesla Model 3	72	0.149	43,470
Tesla Model S	100	0.136	97,000
VW ID.3	58	0.172	41,000

Table 3.1: Features and costs for different models of full electric vehicles.

A use case similar to the one described in this thesis, that can be used as a model, is that of the city of San Francisco. Since 10 August 2023, the Californian city has had two different fleets of robotaxis, managed by two different companies: Waymo, an autonomous vehicle service run by Google, and Cruise, owned by General Motors. This innovation has been controversial, but despite this, Cruise has stated its intention to expand the service to the cities of Phoenix and Austin in 2025. Waymo currently operates a fleet of around 200 Jaguar I-Pace vehicles, while Cruise has around 400 Chevrolet Bolt vehicles on the road. The cost of using the service is $5 per activation, plus $0.90 per mile and $0.90 per minute. Since, it takes about 15 minutes to drive 4 km in Milano, the price for such a trip would be about $13, namely €11.50. A taxi for the same distance costs between €15-20, so prices would be quite competitive for the market (Figure 3.2).

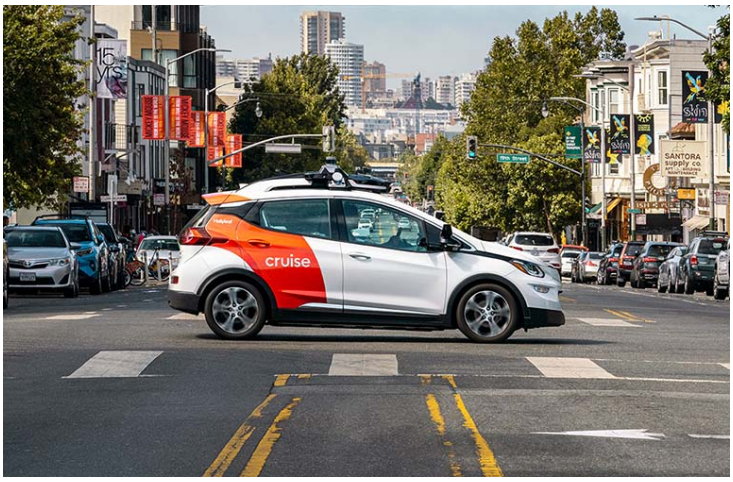


Figure 3.2: A Cruise's robotaxi operating in San Francisco.

Such services are also available in China, where fleets of robotaxis are already operating in several cities, including Beijing. Baidu, the company that manages them, has announced the arrival of the RT6, a sixth-generation robotaxi produced for only around €26,000, but with a very high level of technology and a top speed of 135 km/h. These vehicles will be tested in the city of Wuhan as early as 2024, and the plan is to offer the same service in more than 100 Chinese cities by 2030.

3.1.3. Parameters chosen for the model

Based on data on the purchase costs and features of full electric vehicles and the costs of introducing sensors and automation technology, we chose the following parameters for our model:

- **Cost:** €50,000 per vehicle
- **Battery Capacity:** 40 kW
- **Average Consumpion (C_c):** 0.1375 kWh/km
- **Average Speed:** 14-16 km/h

The idea is to use a compact vehicle, ideal for urban use with low fuel consumption. The capacity must be discreet, but not necessarily very high, because the distances to be covered to serve each customer are within a small area and therefore quite short. For the average speed of vehicles, two distinct cases were considered. In the simulation assuming that there are no more private vehicles in circulation, the average speed was set at 16 km/h, which is slightly higher than the current average speed in the service

area (Table 4.1). This assumption is based on the idea that autonomous vehicles, by communicating with each other, can better manage traffic situations and reduce the time spent in queues. In the other scenario, where private vehicles are still present on the roads, the average speed is set at 14 km/h, slightly lower than the current average speed, assuming that autonomous vehicles have lower peak speeds.

3.2. The robotaxi recharging infrastructure

One of the most important aspects of this thesis is the introduction of dynamic charging of AVs during the simulation. Indeed, this is one of the most complicated aspects to model, but still quite realistic. One of the biggest problems in electric mobility is that of charging modes and times. It is necessary to correctly calculate the number of columns needed to meet the requirements of all the vehicles and their location. If there are some mistakes in this implementation, a vehicle can be forced to travel a longer distance to get to a station and create a disruption: a vehicle would be unavailable for a longer period of time and a higher energy cost for having to travel a longer distance should be obtained.

A company planning such a service would also have to consider the number of times the vehicles would be recharged and the times at which they would be recharged. In fact, we would not want a service that had no vehicles available at certain times because they were all recharging at the columns. A way must therefore be found to ensure that as many charged vehicles as possible are available at peak times in order to meet customer demand as far as possible.

3.2.1. Features and costs

Charging modes for electric vehicles can be of different types and the construction of the columns (Figure 3.3) is subject to international technical standards. There are four types of charging [23]:

1. **Slow charging in a domestic environment:** Only permitted in domestic environments and from normal single-phase domestic sockets. Can only be used for bicycles, scooters and motorbikes, not electric cars.
2. **Slow charging with protection in a private environment:** Similar to the first mode, but in this case the power cable is equipped with a protection system to ensure safe charging. The power is up to 2.3kW and the charging times for cars are therefore very long.
3. **Slow charging in private and public environments:** These are real, private

or public charging stations, designed and installed for charging electric or plug-in cars, in which alternating current is supplied in a safe and smart way.

4. **Quick charging in public environments:** In this case, the current is continuous, the column is already equipped with a cable and allows the car to be recharged very quickly, in some situations even within a few minutes.

Figure 3.3: Example of a public recharging column.

In practice, public columns are therefore divided into two types:

1. **Slow recharging stations:** They can deliver up to 22kW of alternating power.
2. **Fast recharging stations:** With a continuos power greater than 22kW, they are in turn divided into:
   - **Fast:** Up to 100 kW
   - **Ultrafast or HPC:** Over 100 kW (up to 350 kW).

In Table 3.2 are the power ratings and associated costs for different types of public charging columns on the market. Given these prices and the level of charging efficiency achieved today, we have chosen the following parameters for the charging infrastructure in our model:

- **Power (P_c):** 50 kWh
- **Cost:** €0.50 per kWh
- **Efficiency (η_c):** 99%
- **Installation cost:** €3,500 per column

- **Soil rent:** €420 per column per year (€70 per m^2 per year)

In this way, we assume an existing level of power at a slightly lower cost than the average, imagining that the utility can obtain more favourable prices than the current private client. For the installation costs of the charging columns, we have used the prices of the quotations provided by Instapro [24], while for the rental of the land we have used the prices defined by the Canone Unico Patrimoniale in force from 2019, assuming that each charging station occupies about 6 m^2 net space [25].

Company	Power [kWh]	Cost [€/kWh]
A2A	-	0.34
	110	0.86
	150	0.95
Acea	22	0.58
	50	0.68
Duferco	50	0.62
	50	0.65
	>50	0.79
Enel X	-	0.40
	22	0.69
	<150	0.89
	150	0.99
Ionity	<50	0.49
	<350	0.79
Telepass	<49	0.70
	>100	0.80
Tesla	>150	0.61

Table 3.2: Power and associated cost for different types of charging columns available on the market.

3.2.2. Recharging model

Having defined all the costs associated with recharging AVs, we proceed to describe the recharging model, following the work described in [18], with some slight modifications.

We need first to define how the state of charge (SOC) of the battery changes during use. While the car is moving, the discharging process described by the equation takes place:

$$SOC(t + \Delta t) = SOC(t) + C_c \cdot d(\Delta t) \tag{3.1}$$

where SOC represents the state of charge of the battery (in kWh), C_c the average consumption (in kWh/km) and $d(\Delta t)$ the distance (in km) travelled in the time interval Δt.

The evolution of the SOC during the charging phase follows the equation:

$$SOC(t + \Delta t) = SOC(t) + \eta_c \cdot P_c \cdot \Delta t \tag{3.2}$$

with η_c and P_c representing the efficiency and power of the columns, respectively.

The process thus consists of 6 steps (Figure 3.4):

1. The AV is in service with a certain level of SOC, the vehicle status is 'AVAILABLE'.
2. At the end of a run, the SOC level becomes lower than a certain previously set threshold: $SOC(t) < low\_threshold$. The status of the vehicle becomes 'NOT AVAILABLE' and it will temporarily be unable to serve other customers.
3. At this point there are two possibilities:
   a) If, among the columns that can be reached in less than 20 minutes, none is free, then the vehicle will wait in its position for one to become free.
   b) If there are vacant columns that can be reached in less than 20 minutes, the vehicle will drive to the nearest of them.
4. The vehicle reaches the charging station and starts charging, its status remains 'NOT AVAILABLE'.
5. When the SOC level reaches another predetermined threshold, its state returns to 'AVAILABLE': $SOC(t) > high\_threshold$. The vehicle remains in the charging state until one of the following two cases occurs:
   a) A call is made to the recharging vehicle, which, being 'AVAILABLE', will be able to fulfil the request, freeing the column and heading for the start point.
   b) The SOC level reaches 100%, i.e. the vehicle is fully loaded. At this point we assume that the AV clears the column, remaining stationary in the same position while waiting for a call.
6. The robotaxi is now back in service and the procedure starts again from step 1.

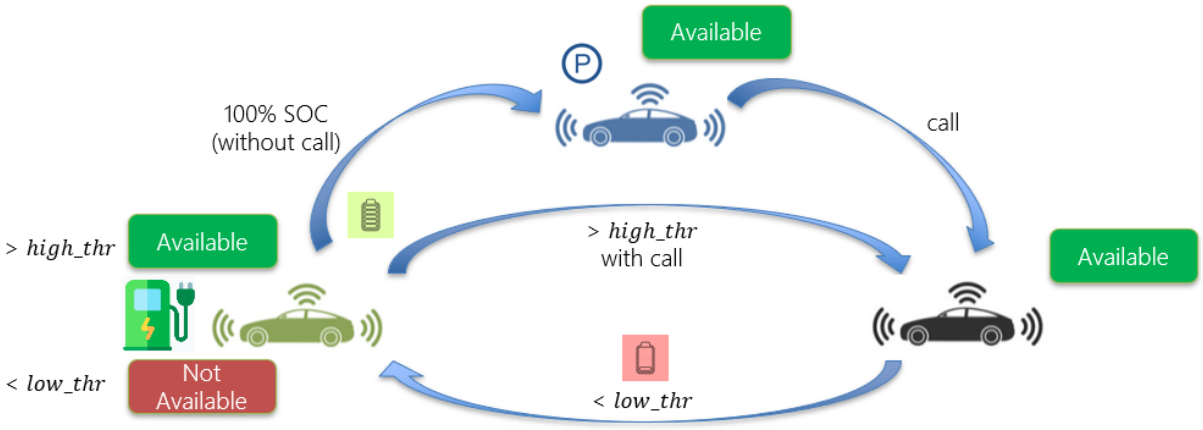


Figure 3.4: Scheme of the process used for the recharging model.

It remains to set *low_threshold* and *high_threshold*. In our simulation model, we have chosen to define separate values for each threshold: one for daytime and another for nighttime. Nighttime is defined as the period from 2:00 AM to 5:00 AM, during which we have set higher thresholds in both cases to account for increased vehicle charging demand at times when overall user demand is lower. For *high_threshold*, which represents the level of SOC for which the vehicle returns 'AVAILABLE', the choice is less important and more arbitrary: in our case we have chosen 60% SOC during the day and 90% during the night. For *low_threshold* there are more factors to consider. The case to be avoided is that the vehicle discharges during a trip, so this threshold must be high enough to allow any vehicle with a higher SOC to reach a new customer and take them to their destination before reaching the charging station. Below are the calculations made for the consumption of an AV in the worst-case scenario, bearing in mind that the maximum distance that can be covered is can be assumed about 12 km, with a coverage area radius of 6 km:

1. **At customer arrival:** Let us consider the worst case of 12 km to reach the client call.
2. **Distance travelled:** Let us consider the limiting case of a 12 km trip.
3. **Getting to the charging station:** Let us consider the limit case of 20 minutes to reach the charging station.

Taking into account the vehicle's average consumption, speed and battery capacity:

- **Consumption** (C_c) = 0.1375 kWh/km
- **Average Speed** = 14-16 km/h

- **Battery Capacity** = 40 kWh

We obtain the following SOC_i values after each phase mentioned before:

$$\begin{aligned} SOC_1 &= 12 \text{ km} \cdot 0.1375 \text{ kWh/km} = 1.65 \text{ kWh} \\ SOC_2 &= 12 \text{ km} \cdot 0.1375 \text{ kWh/km} = 1.65 \text{ kWh} \\ SOC_3 &= 0.33 \text{ h} \cdot 16 \text{ km/h} \cdot 0.1375 \text{ kWh/km} = 0.726 \text{ kWh} \\ SOC_{\text{tot}} &= 1.65 \text{ kWh} + 1.65 \text{ kWh} + 0.726 \text{ kWh} = 4.026 \text{ kWh} \end{aligned} \tag{3.3}$$

From these calculations we can see that the threshold of energy consumed by a single service task cannot exceed about 10% of the total battery capacity. We therefore set $low\_threshold$ to 15% for daytime and 40% for nighttime, to encourage recharging at times of lower demand. With these values, we never had an empty vehicle in any of our simulations.

3.2.3. Placement of charging stations

In addition to the charging process, the size and location of the charging stations themselves is also crucial.

In our model, we assumed that charging stations would be built in different parts of the city. The service area of the AVs was divided into square 'blocks', each square with a side of 500 m (Figure 3.5). Within each of these blocks there would be one charging station, which could contain a number of columns, which we describe below. This could indeed simplify the work of the provider and the municipality, as they would only have to use 'standardised' charging stations. For the sake of simplicity, these have been placed at a latitude and longitude corresponding to the centre of their reference block.

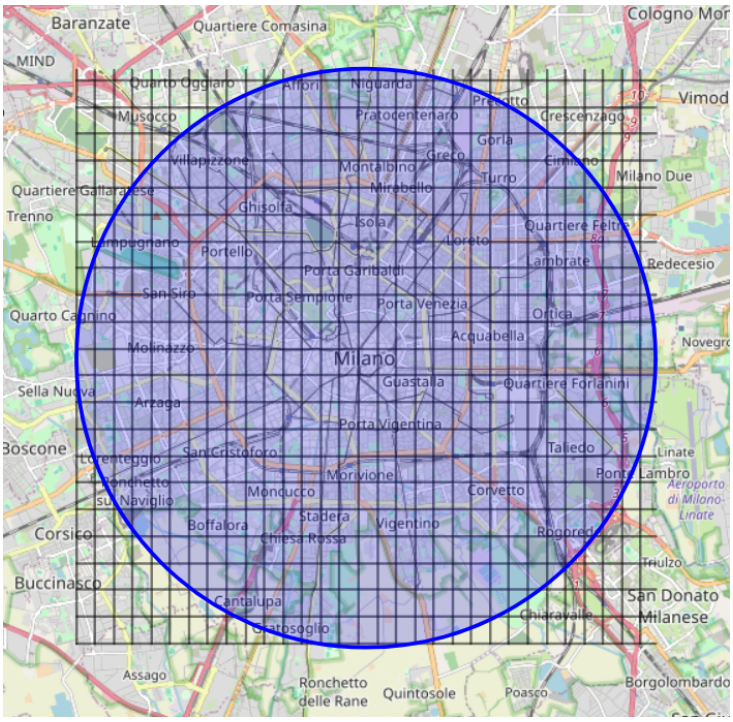


Figure 3.5: The city of Milano divided in 500x500m blocks.

The most useful and complex part is to define the total number of columns to be installed and how to distribute them among the various stations. As far as the total number of columns is concerned, we decided to set it at 2,000, based on projects that are in the approval phase and that envisage similar investments in the Milano area [26]. In the simulation with a more limited number of customers to serve, we reduced this number to 500, ensuring they are still distributed across the entire area but without significantly impacting service costs. From the results we obtained, we also noted that these numbers seemed sufficient to handle the fleets of AVs to be used for the service.

The next step is to work out how best to position the charging stations. Our aim is to manage the positioning based on real data. Basically, we count how many trips start or end in a particular block and allocate a proportional number of columns in that station. The more the block is in demand, the more columns are placed. In this way, it will be easier for an AV to find a free charging column nearby when it finishes a run, or there will be an AV that has finished charging nearby when a user sends a request. To give an example, suppose we have 10,000 trips in our dataset and we want to place 2,000 charging columns. From a certain block there are 100 trips, which is 1% of the total, so 20 columns will be placed in the corresponding charging station, i.e. 1% of 2,000. Below are the placement rules:

- In a cell 0 columns are allocated only if the proportion would allocate less than 0.25 columns.
- In a cell 1 column is allocated if the proportion would allocate from 0.25 to 1.25 columns.
- In a cell 2 columns are allocated if the proportion would allocate from 1.25 to 2.5 columns.
- If the proportion is greater than 2.5, in the cell are allocated a number of columns multiples of 5, rounding up the value.

A similar algorithm for placing charging points was used in [18], with the difference that there was a maximum of 20 columns per station. We have decided to remove this limit, as it would have been difficult to allocate all 2,000 columns in this way, and would have required a large number of columns to be installed in areas that are not heavily used. Applying this algorithm, in a single station never more than 40 columns are allocated, which is a reasonable number. In Figure 3.6, the placement of 2,000 charging columns, obtained using the described algorithm, is depicted.

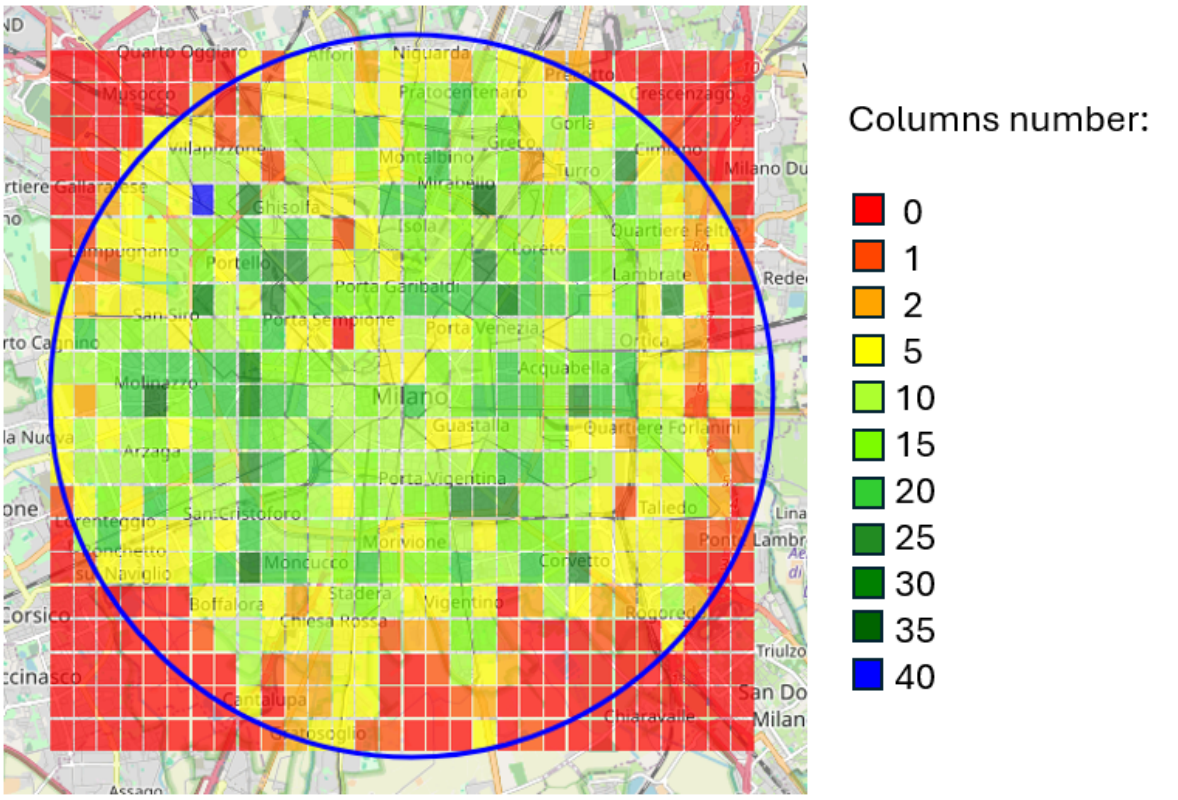


Figure 3.6: Placement of 2,000 charging columns in the corresponding charging stations.

4 | Data-driven experimental modeling of a robotaxi fleet co-designed with pedestrians and public transport

This chapter presents a comprehensive description and analysis of the data provided by the Unipol Group, which forms the basis of the model. In the second part, it is shown how, starting from the data, the urban network was created in which AVs, public transport, and pedestrians move within the simulation.

4.1. The real driving dataset for inferring people movement

One of the key themes of this thesis is to build the model based on real data. Thanks to Unipotech, part of Unipol Group, we were able to access their customers' travel data, collected anonymously from GNSS devices (commonly known as Unibox, Figure 4.1) installed in insured vehicles. The Unibox constantly collect data on vehicle behaviour and periodically sends mileage and time data to Unipol's servers.

Figure 4.1: Example of Unibox.

In total, we have a dataset consisting of about 17.5 million trips made by 20,000 different cars immatriculated in the province of Milano during 2022. Each trip is characterised by the following parameters (Figure 4.2):

- **Id_trip:** Unique identification series for each trip.
- **Time_start:** Trip start time (date, hour, minutes and seconds).
- **Time_end:** Trip end time (date, hour, minutes and seconds).
- **Latitude_start:** Unibox datum with starting latitude.
- **Latitude_end:** Unibox datum with starting longitude.
- **Longitude_start:** Unibox data with latitude of arrival.
- **Longitude_end:**Unibox data with longitude of arrival.
- **Trip_duration:** Time duration, expressed in seconds, of the trip considered.

Other filed such as Id_driver, Average_speed, Distance and others are present in the dataset but they were not exploited in the creation of our model.

id_trip	id_driver	time_start	time_end	latitude_start	longitude_start
30540f3d-3850-4b8c-80c0-3e3126141111	2	2022-01-10 11:28:59+00:00	2022-01-10 11:39:13+00:00	45.43649	9.086906
ed49e684-28c4-4a2d-9a25-49e51caef3a1	2	2022-01-11 08:29:41+00:00	2022-01-11 08:43:19+00:00	45.454876	9.099706
2f98e3d5-2fc5-4a76-91f1-ea3bae929ee6	2	2022-02-08 09:08:04+00:00	2022-02-08 09:22:50+00:00	45.45471	9.100254
81a9d324-7e25-474f-bf31-2b8efad5864b	2	2022-02-08 10:10:50+00:00	2022-02-08 10:23:23+00:00	45.430916	9.080309
d51545ad-2c79-4679-b066-717b540bd405	2	2022-02-17 09:13:39+00:00	2022-02-17 09:25:33+00:00	45.45492	9.100154
0851e414-47f2-41a8-8526-c4d499568b21	2	2022-02-17 11:12:13+00:00	2022-02-17 11:29:12+00:00	45.44623	9.039604
b4800b05-a58a-4a37-85fb-2bd4f66b7a88	2	2022-02-22 11:47:34+00:00	2022-02-22 11:56:02+00:00	45.454617	9.100073
b0e94986-a34d-4967-a440-27d373d37339	2	2022-03-03 08:42:48+00:00	2022-03-03 09:11:39+00:00	45.454895	9.10023
45f45801-d63f-41d9-8a53-c7fc4c62e2e7	2	2022-03-03 09:17:47+00:00	2022-03-03 09:28:50+00:00	45.481052	9.015874
6c88149c-9a83-4fa7-a337-774778ae7f0b	2	2022-03-03 10:53:18+00:00	2022-03-03 11:02:34+00:00	45.490498	9.035012
4a4caf1b-6870-40f5-8dd3-9553c1e508e2	2	2022-03-03 11:22:52+00:00	2022-03-03 11:30:55+00:00	45.489937	8.997806
22f028fa-c197-42e7-956e-f0234c38a935	2	2022-03-03 11:50:58+00:00	2022-03-03 12:11:03+00:00	45.481064	9.015894
04408b03-e39e-4d4b-8d29-14550038033b	2	2022-03-05 08:51:07+00:00	2022-03-05 09:14:53+00:00	45.454704	9.100376

longitude_start	latitude_end	longitude_end	distance	stop_duration_BT	trip_duration_BT	average_speed_trip_kmh
9.086906	45.454876	9.099706	4210	75028	614	24.68403909
9.099706	45.4351	9.093615	4070	2420685	818	17.91198044
9.100254	45.43091	9.080317	4290	2880	886	17.43115124
9.080309	45.45492	9.100154	5940	773416	753	28.39840637
9.100154	45.441544	9.041955	5720	6400	714	28.84033613
9.039604	45.454617	9.100073	6370	433102	1019	22.50441609
9.100073	45.453545	9.084192	1470	766006	508	10.41732283
9.10023	45.481052	9.015874	11020	368	1731	22.91854419
9.015874	45.490498	9.035012	5220	5068	663	28.3438914
9.035012	45.489986	8.997704	3650	1218	556	23.63309353
8.997806	45.481064	9.015894	2370	1203	483	17.66459627
9.015894	45.454704	9.100376	10570	160804	1205	31.57842324
9.100376	45.480927	9.015998	11110	5583	1426	28.04768583

Figure 4.2: Some rows of the dataset.

4.1.1. Data description and preprocessing

The only trips of interest to us, for building the model, are those taking place, starting and ending, within a 6 km radius. In fact, our AVs service can only satisfy the demand for these, as it cannot go beyond this radius (Figure 4.3).

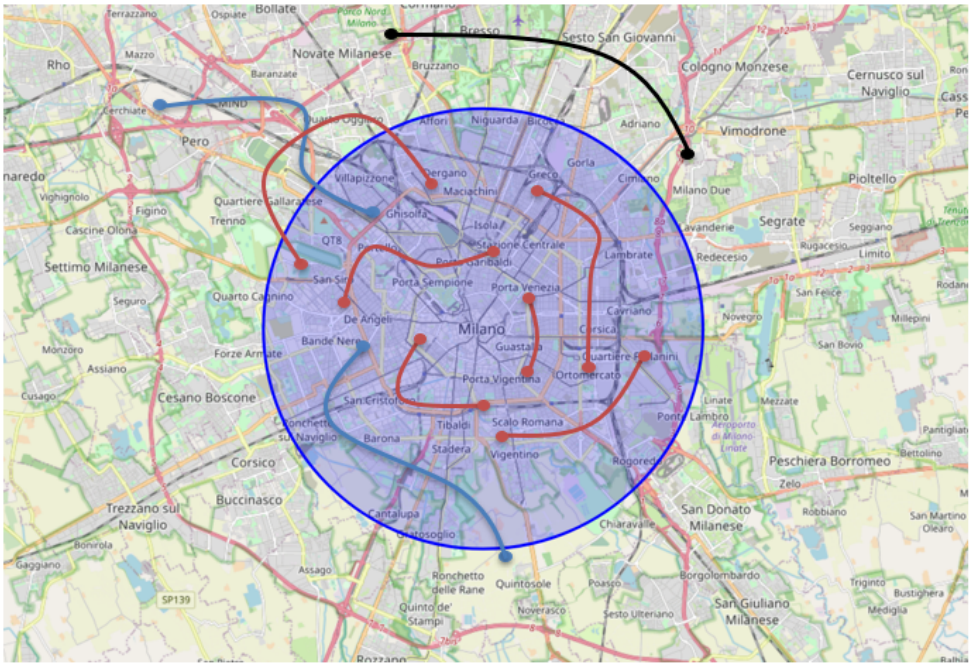


Figure 4.3: Example of trips: only the red cases are completely inside the service are and kept in our analysis. The blue cases have start or end point outside, the black cases have both outside.

A function was then created which, given latitude and longitude values as input, checked whether the point was within the service area. Only trips where both the start and end points were within the zone were kept, the others were discarded. This left about 8% of the original number of trips, as shown in Figure 4.4. As the data is taken from the whole province, only a minority of the trips are made within the urban area of the city.

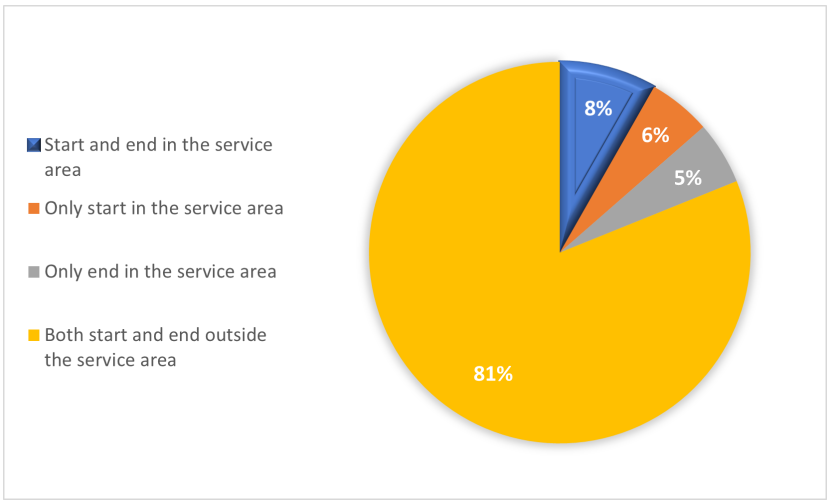
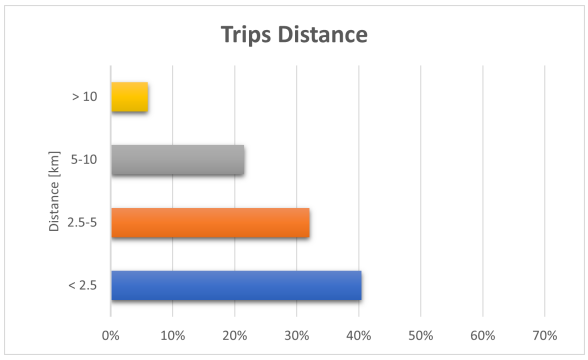
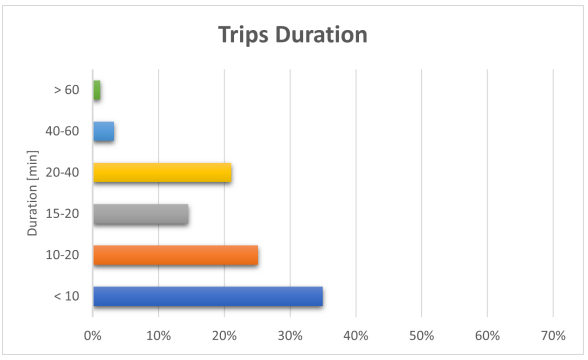


Figure 4.4: Distribution of the trips with respect to the service area. The trips used in the model are highlighted.

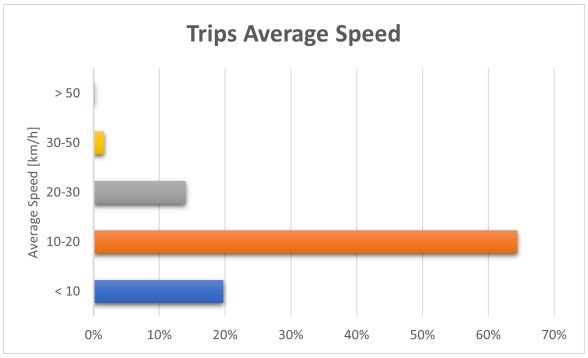
The distribution of times, distances and speeds of the remaining trips was analysed, obtaining the results shown in Figure 4.5.



(a) Distance.



(b) Duration.



(c) Average Speed.

Figure 4.5: Distance, duration and average speed of the trips inside the 6 km radius. On the abscissa are reported the % of trips in the correspondent category.

As might be expected, trip times, distances and speeds are generally not high, as the trips to be made are generally not very long and urban traffic does not allow for very fast trips, given the many interruptions caused by traffic lights and other cars. It can therefore be seen that the use of autonomous vehicles travelling at an average speed of 16 km/h would be an improvement in trip times, as this is a higher speed than the current average. This assumption is consistent with the fact that self-driving cars, while achieving lower peak speeds, can manage traffic very well, thus reducing queuing. The mean, median and standard deviation of distance, duration and speed are given in Table 4.1.

	Mean	**Median**	**Std Dev**
Avg Speed	14.91 km/h	14.66 km/h	3.17 km/h
Distance	3.87 km	3 km	2.89 km
Duration	15 min 24 sec	12 min 24 sec	9 min 34 sec

Table 4.1: Table reporting the mean, median, and standard deviation of the three variables of the trips within a radius of 6 km.

However, from the scatterplot shown in Figure 4.6, which represents the distributions of duration and distance in our data, we note the presence of numerous outliers, i.e. values very far from the averages, probably due to errors in the recordings made by the corresponding Unibox. In fact, the sensors installed inside them, are not always perfectly accured. These values have therefore been eliminated; in total they represent 1.6% of the total number of values and thus do not significantly affect the number of data remaining. Specifically, values with travel times greater than 1 hour or distances greater than 20 km were eliminated.

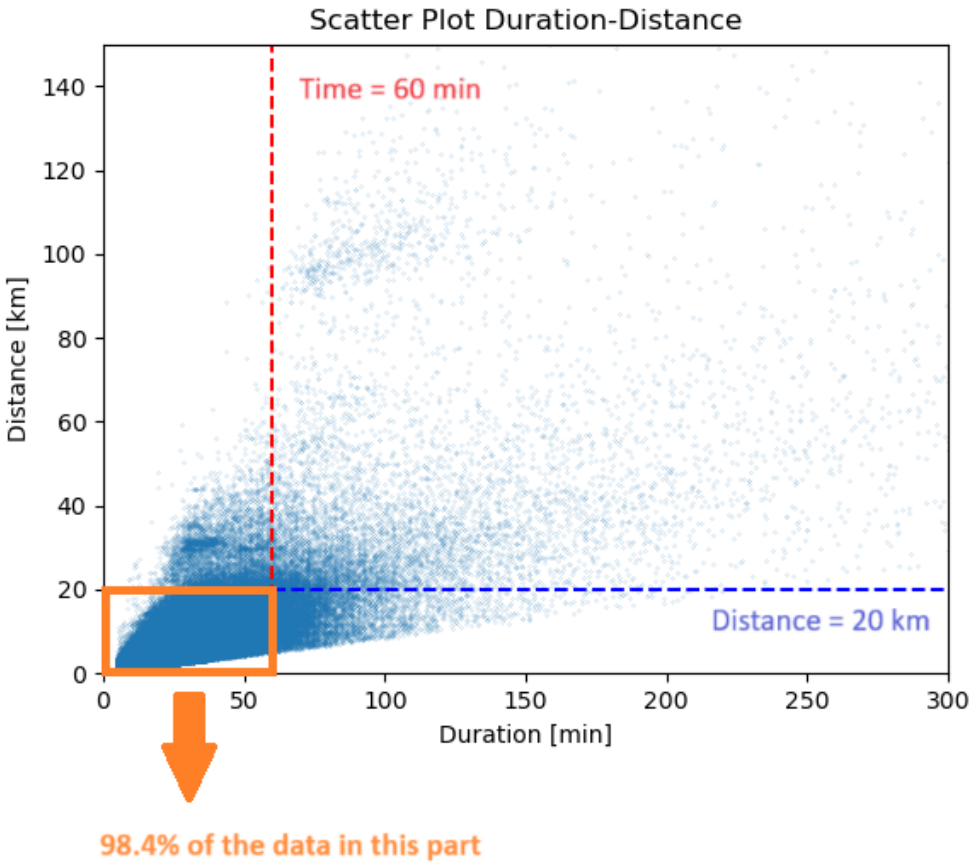


Figure 4.6: Distance-duration scatterplot of the trips inside the 6 km radius.

We continue to remove these outliers as well as they could greatly alter the operation and validation of our model. It seems unlikely that such trips in a space as small as an area of 113 km^2 could have such large distances or durations. In fact, the Unibox is not very accurate in recording trips, and could even combine two successive trips if they are only a short time apart.

After removing these elements as well, our dataset turns out to contain about 1.4 million trips, all within the service area and containing sensible values.

4.1.2. Data spatial clustering

Another issue to address in data preprocessing is the positioning of the starting and ending points of trips. When aiming to utilize a city graph to navigate our vehicles, we require a limited number of starting and ending points, which will constitute the nodes of our graph. Given the initial data, we actually obtain more than 2 million different points,

which would constitute an unsustainable number of nodes for a graph.

The idea is to aggregate nearby nodes, merging them into a single node with a position equal to the centroid of the aggregated nodes. To achieve this, the DBScan algorithm (density-based clustering algorithm) was used, which groups 'densely grouped' data points into a single cluster. DBScan takes two input values:

- **EPS:** the cluster radius, which is the distance within which points will be aggregated.
- **MinPts:** minimum number of points in a single cluster.

This algorithm classifies the points into 3 categories:

- **Core Point:** A point that has at least MinPts within its EPS neighborhood, excluding itself. There can be multiple core points within a cluster.
- **Border Point:** A point that is not a core point but is in the neighborhood of a core point.
- **Noise Point:** A point that is neither a core point nor a border point.

For each point x, its neighbors within the radius EPS are calculated:

- If the number of neighbors is equal to or greater than MinPts, then x is a core point and a cluster c is created. For each x' neighbor of x, proceed as follows:
  - ⋆ If x' meets the requirements to be a core point, then it becomes one, and the same procedure used for x is applied to x'.
  - ⋆ Otherwise x' is a border point.
- Otherwise x is initially labeled as a noise point and remains so if it does not fall within the neighborhood of another core point.

In Figure 4.7, a simple example of DBScan functionality is illustrated.

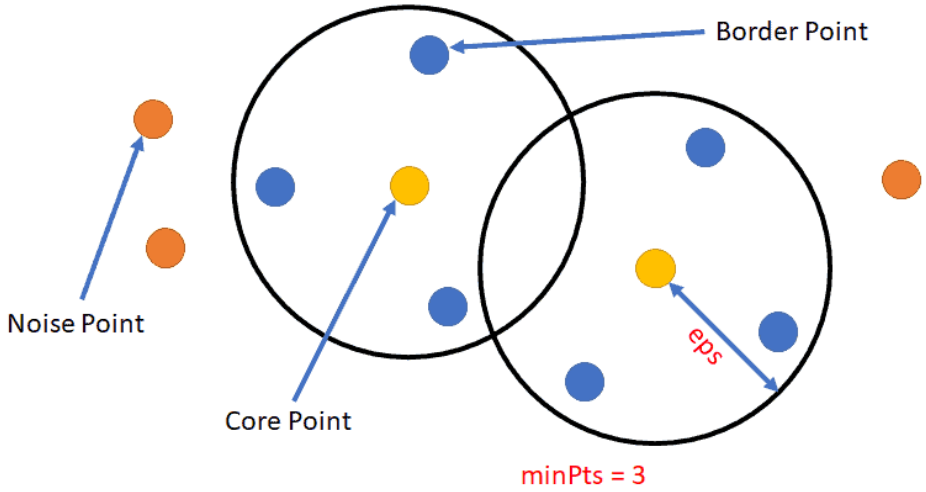


Figure 4.7: DBScan example with EPS=eps and MinPts=3.

The difficulty lies in finding the right working parameters for spatial clustering. A value that is too small for EPS leads to the formation of many clusters, while one that is too large creates a sort of "chain reaction" in clustering, resulting in too few clusters in densely populated areas. A MinPts value that is too small leads to the formation of many small, insignificant clusters on the periphery, while a large number eliminates a considerable number of points. After several attempts, we achieved good results by setting EPS=100 m and MinPts=10. This resulted in 9814 clusters. The latitude and longitude of these clusters were recalculated using the centroid of the constituent points, and each cluster was assigned a name from '1' to '9814'. Within the dataset, the latitude of the starting and ending points was replaced with that of their respective clusters, and the cluster name was added. In Figure 4.8, the obtained result is shown.

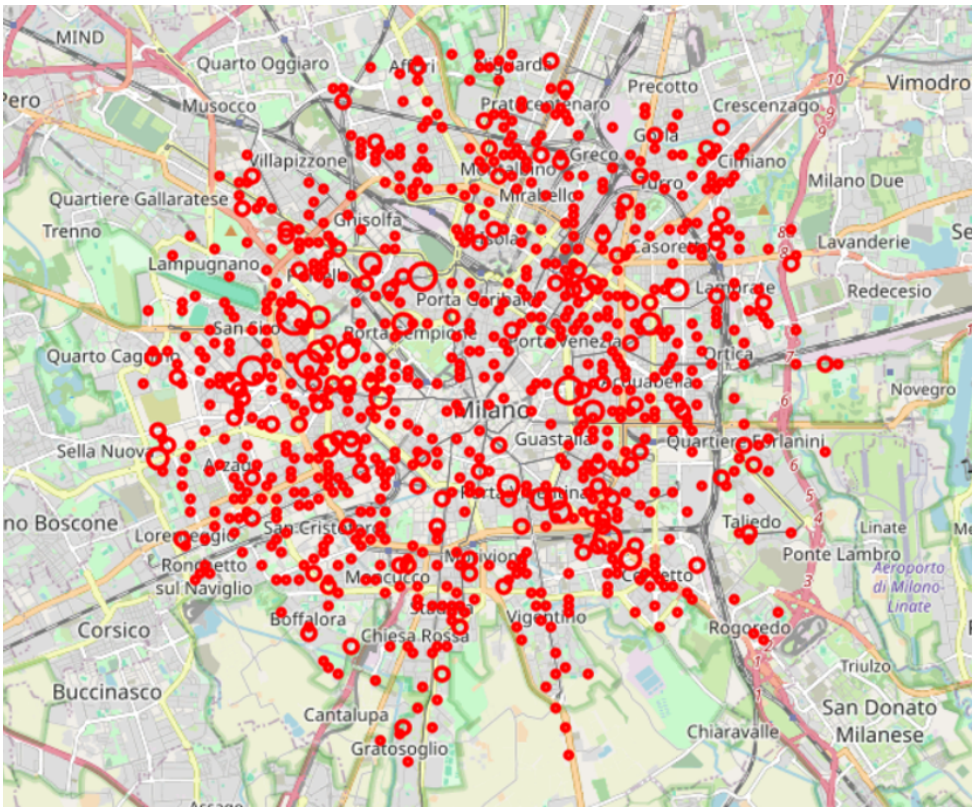


Figure 4.8: Result of the DBScan applied to our dataset, with EPS=100m and MinPts=10. Every circle represent a different cluster, the radius of the circle is proportional to the cluster's size.

As a final step, trips that had the same starting and ending points after DBScan were eliminated.

4.2. The data-driven street network approximation

As anticipated, one of the most significant contributions of our model is the creation of a realistic city graph, within which users can move both being AVs, public transportation, and on foot. This is extremely useful as it allows for more realistic mobility simulations and the future possibility of integrating other realistic features, such as congestion, i.e., a limit on the number of cars allowed to pass through a road (simulating slowdowns due to traffic), or changes in travel speeds based on road segments, making urban mobility more realistic, imposing, for example, lower travel speeds in city centers. Further innovations could be brought to the construction of the graph, aiming to create it with increasingly accurate techniques, such as those discussed in [27], where localization techniques utilizing common sensors in smartphones are introduced to create highly precise graphs.

For our work, we have introduced two different intersecting graphs: the first is the one traversed by robotaxis, and the second is the one that users could traverse autonomously using public transportation and walking.

4.2.1. On-demand autonomous mobility graph

The On-demand autonomous mobility graph (AVs graph), traversed by robotaxis, is entirely derived from the latitude and longitude data available in the dataset. It was previously explained how DBScan was utilized, resulting in the formation of 9814 distinct nodes. In addition to these, nodes corresponding to charging stations were also added. These stations must be accessible by AVs when they need to recharge, so they must be present in the graph and reachable. They were introduced with their latitude and longitude coordinates, and a name was added for each of them with values ranging from 'c0' to 'c483'.

Starting from this complete information on nodes with 'name', 'latitude', and 'longitude', it is possible to proceed to create the graph. The other essential component to add is computing the edges along with their respective weights. However, not all vertices are directly connected to each other; it wouldn't be realistic to connect two points at opposite ends of the city with a single road. Therefore, a function has been created to generate edges that only connect nodes within a distance of 150 meters, assigning them a weight corresponding to the AVs travel time in seconds. This travel time is calculated by dividing the distance in meters by the speed, which varies between 14 and 16 km/h depending on the scenario analyzed, and converting the result to m/s.

The choice of a 150-meter distance is arbitrary but closely approximates the typical length of a single road segment in the city. Moreover, it prevents the creation of overlapping roads, contributing to a realistic map. The final step is to create a fully connected graph. Initially, some peripheral areas remained disconnected from the rest of the city because there were no points within 150 meters nearby, making it impossible for users in those areas to reach most other nodes in the graph. Due to the circular structure of Milan city, a very large central component containing almost all nodes emerged, along with several small disconnected peripheral components. To connect these components, a simple method was employed: we gradually increased the maximum distance to connect nodes by 50 meters at a time until a single peripheral component was connected to the already connected component. This process results in a fully connected graph in which the AVs can operate (Figure 4.9).

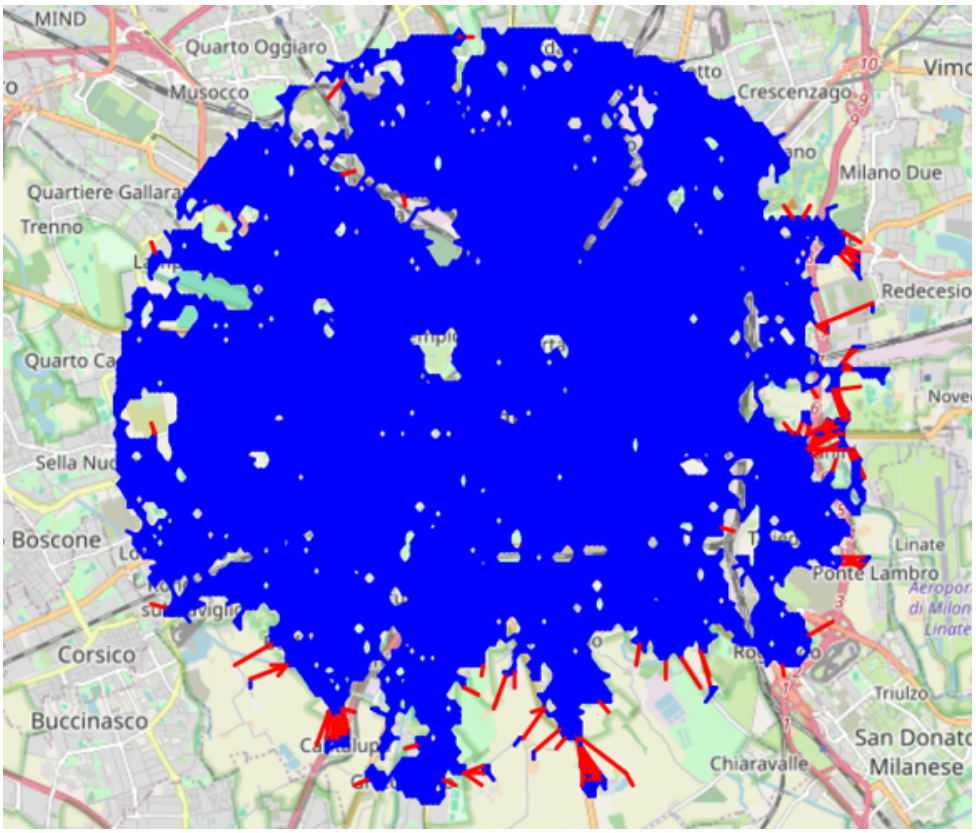
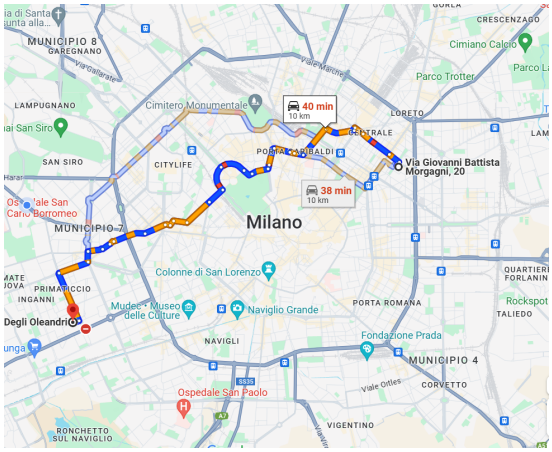


Figure 4.9: Image of the obtained graph for the city of Milan, every blue line corresponds to an edge. The red lines are the edges added later to connect all the components.

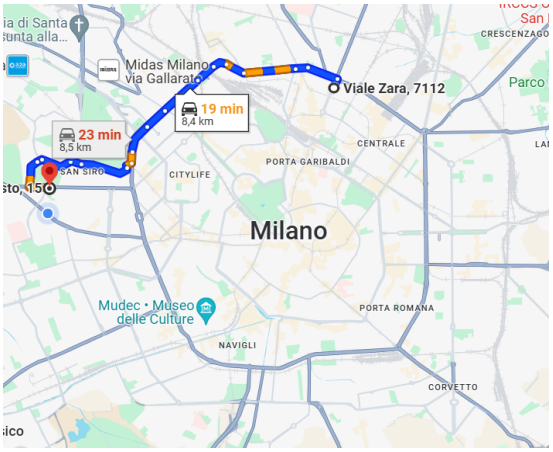
In Table 4.2, calculations of travel times between nodes in the AVs graph are reported. These are obtained both using our model and Google Maps. The positions of the nodes are visible in Figure 4.10.

	Google Maps	Model
Trip 1	38 min	28 min 41 sec
Trip 2	19 min	24 min 57 sec
Trip 3	42 min	30 min 26 sec

Table 4.2: Duration of 3 sample different trips computed with Google Maps and our model (with average speed set at 16 km/h).



(a) Trip 1.



(b) Trip 2.

(c) Trip 3.

Figure 4.10: Paths of the 3 trips referred to in Table 4.2.

We can observe differences in the calculated travel times. Particularly, trips traversing multiple congested segments are faster using our model, while those on smoother roads are slower. This is entirely understandable: AVs perform better than human-driven vehicles in congested roads, optimizing time spent in queues. Conversely, they perform worse on less congested roads, where speeds are higher.

4.2.2. Walking and public transportation graphs

For the graph representing the subway and walking nodes of mobility, the construction is slightly more complex and occurs in multiple stages.

1. Firstly, the Walking layer is created. This is the simplest step, as it utilizes the same network previously constructed for the robotaxis, with the only modification being

the adjustment of the travel speed on road segments from 14-16 km/h to 4.8 km/h, which is assumed the average walking speed. The assumption is that pedestrians move only on sidewalks adjacent to the roads, which therefore have the same lengths and directions as the roads themselves.

2. The second step is the creation of the Public Transportation layer. Each node of every subway and suburban train station within the service area was created one by one, including not only the name but also the corresponding latitude and longitude (Figure 4.11).

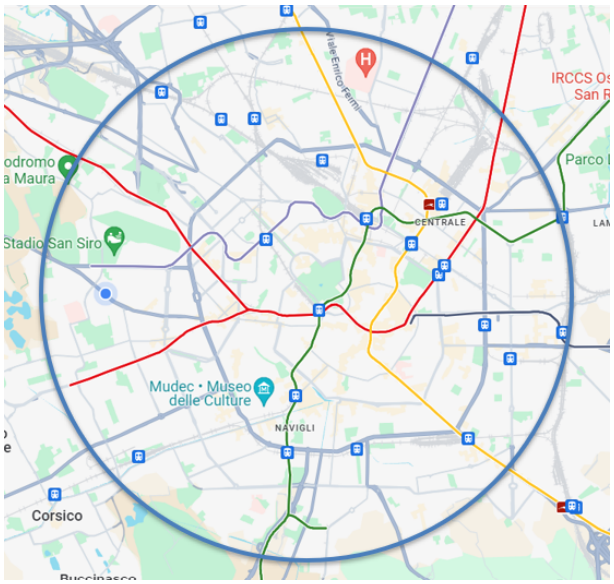


Figure 4.11: Representation of the 5 metro lines in the city of Milano. Only the stations inside the defined service coverage area were considered in the model.

If particular stations serving as interchange locations for multiple subway lines, a single node was created for each line. For example, nodes 'duomo1' and 'duomo3' were created, both with the same latitude and longitude coordinates. For each edge created, the weight represents, as before, the travel time. The edges can be of three different types:

- **Edges that connect two consecutive metro stations on the same lines:** For these edges, the travel time was set to 90 seconds, which is the time for a train to travel from one station to the next.
- **Edges that connect to consecutive suburban stations on the same lines:** In this case, the travel time is not fixed but has been calculated on a case-by-case basis by consulting the actual schedules of ATM available on the

website [6].

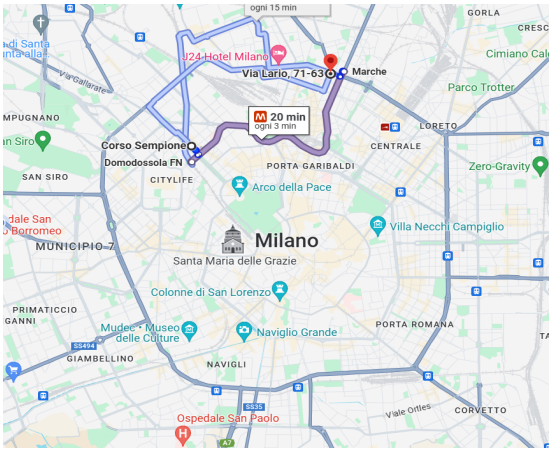
- **Edges within an interchange station:** Here, the weight represents the time for transferring between two different types of metro or between a metro and a suburban train. In either case, the calculated time is 2 minutes plus the waiting time, which is 3 minutes for the metro and 8 minutes for suburban trains.

3. The final step is to create the intra-layers network graph. Each node in the Public Transportation layer must be connected to the Walking layer. Otherwise, it would be impossible to reach public transportation within the simulation since the starting and ending points belong only to nodes in the other two graphs. It was decided to connect each node corresponding to a station to the 4 nearest nodes in the Walking layer to simulate, to some extent, the exits of the subway. To traverse the station-exit route, the distance in meters (divided by the speed of 4.8 km/h converted to m/s) plus 1 minute was used. However, for the reverse journey, the timing is different: the waiting time for the train must also be taken into account. Therefore, the distance in meters plus the waiting time (always 3 minutes for the metro and 8 for suburban lines) was used.

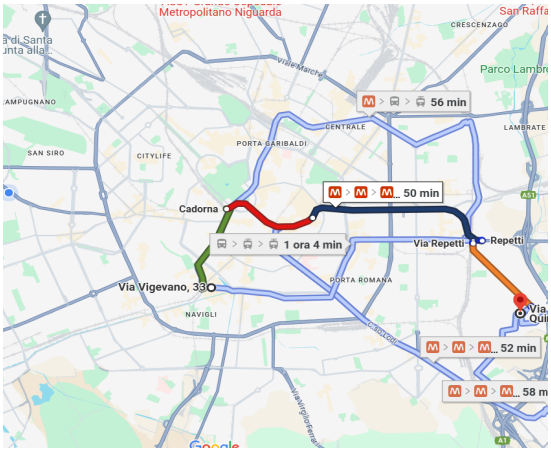
After completing these steps, a single graph is obtained in which both public transportation and walking routes can be used. Below, we provide some examples of trips, comparing the travel time calculated by Google Maps with that of our model (Table 4.3). We note that travel times are very accurate when using only the subway. The only significant simplification made was not accounting for any differences in public transportation schedules between day and night, always using daytime frequencies.

	Google Maps	**Model**
Trip 1	20 min	22 min 2 sec
Trip 2	50 min	59 min 14 sec
Trip 3	7 min	6 min 1 sec

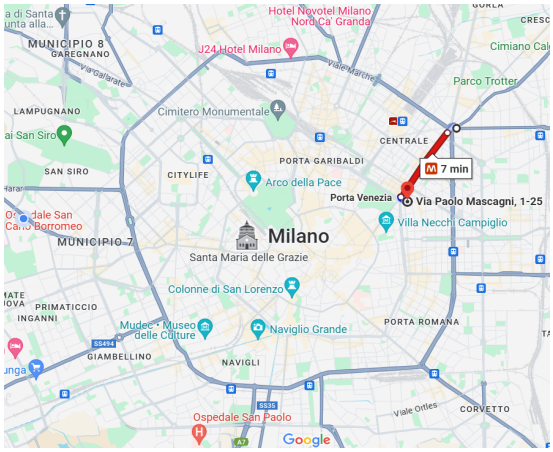
Table 4.3: Duration of 3 different trips computed with Google Maps and our model.



(a) Trip 1.



(b) Trip 2.



(c) Trip 3.

Figure 4.12: Paths of the 3 trips shown in Table 4.3.

5 | Simulation of the tri-modal urban transportation system

In Chapter 3 and Chapter 4 all the elements necessary for the construction of the new mobility system have been introduced. These include the hypotheses set for speed, consumption and purchase cost of the robotaxis, as well for the recharging infrastructure, and the set-up of the layers on the basis of which the simulation takes place.

This chapter will describe the mechanism by which the simulator operates, the approximations that have been made, the computational cost and how convergence is achieved.

5.1. Structure and operation of the simulator

The simulator is programmed in Python. The complete code can be found in Appendix A, at the end of this thesis. In this section, we will highlight and detail all the functionalities one by one, describing their logic and purpose.

Simulator Class

First, a Simulator Class is introduced. This class takes the following inputs:

- The dataset consisting of all trips ordered by Time_start.
- The dataset composed of all latitude and longitude coordinates of the square 'blocks' where the charging stations are placed, and the correspondent number of trips start and end in that 'block', as previously explained in Section 3.2.3.
- The number of AVs to be used in the simulation.
- The number of charging columns to be used in the simulation.
- The AVs' layer street graph described by nodes and edges.
- The Walking and Public Transportation graph.

- A cache memory with the precomputed and saved distances between nodes of the AVs' layer street graph. It is usefull to strongly reduce the computational time.
- The 'simulation_type' parameter, setted to '1' if the one-layer model is to be simulated, and '2' for the two-layer model.

This class initialises all the variables that will be needed later in the process. It also contains a series of parameters that can be modified according to the requirements of the user. These include the $low\_threshold$ and $high\_threshold$ set for both night and day for AVs battery cycle, the battery capacity of the AVs, their average consumption, and the power of the charging stations. A number of dataframes that will be used for the final results are also initialised, in particular $final\_df$, which contains, for each vehicle, the total time spent travelling, the overhead time (i.e. the time spent travelling but with no customer on board), the time resting, the total number of calls received, the time spent recharging and the total number of battery charges made.

Once the Simulator Class has been defined, we proceed to the actual simulation, which comprises several interconnected parts.

Initialize Recharging Stations

This function initiates the charging stations, determining the precise number of columns for each station. The calculation is straightforward and involves a proportion being calculated between the demand for trips corresponding to the square 'block' in which the station is located and the total demand for trips in the entire city. Subsequently, the number of columns to be installed in each station is calculated based on the total number of inlets to be installed, which was provided as input to the class simulator. The process is described in detail in Section 3.2.3.

The columns created at each station can be categorised into two distinct lists: those designated as 'available columns' and those designated as 'occupied columns'. Initially, all of the columns are designated as 'available columns'.

Initialize Cars

This function is used to initialize the AVs. The function takes as input only the number of AVs to initialize. Again, two distinct lists are created, 'free cars' and 'busy cars'. All AVs are initially placed in the first list, and each AV contains a series of essential variables for the simulation:

- A unique index for each vehicle.

- A node indicating the current position of the vehicle. At the start of the simulation, the vehicles are, for simplicity, placed at the position of the first equal number of customers calling the AVs. For example, if we use 1000 AVs, they will initially be placed at the starting position of the first 1000 trips to be served. This is obviously an unrealistic simplification, but does not have a significant effect on the simulation, as it loses all influence after the AVs have served customers in the first iterations. Obviously, in this way, the first customers will have an AV immediately available without any waiting time. In order to prevent this from affecting the analysis of the results, the first 2000 trips for each simulation have been excluded from the calculation, so that the counting of certain parameters essential for assessing the efficiency of the model does not take this assumption into account.
- A parameter called 'busy until' that indicates the time until which the vehicle is unavailable. It is initially set to the simulation start time, so that all vehicles are available.
- Various parameters for the final results, such as travel time, overhead, stationary time, number of calls, recharge time, and number of recharges. These are all initialized to 0.
- The State of Charge (SOC) of each AV initially set to 100%.
- A parameter 'charging status' that has a value of 'True' if the vehicle is charging or 'False' otherwise. Initially, it is set to 'False'.
- A parameter 'last check recharge' indicating the time at which the last check on the vehicle's SOC was performed.

Check Free Vehicles

This function takes as input the time when the actual request of a customer occurs. The idea is to check the status of all vehicles at that specific moment, updating the 'free cars' and 'busy cars' lists, and recalculating their SOC. An iteration loop is carried out over all robotaxis, proceeding through various steps:

- First, it is checked whether the AV is charging, updating the SOC state according to equation (3.2). If the state of charge reaches 100%, then the corresponding charging column is added to the 'available columns' list using the function **'Free Charging Stations'**.
- It is checked whether the 'busy until' parameter of the vehicle is greater than the time when the travel request occurs. If so, the vehicle is not available and is added

to the 'busy cars' list.

- If the vehicle is charging, its updated SOC status is checked to see if it has exceeded the threshold *high_threshold*. If this condition is met, the vehicle is added to the 'free cars' list; otherwise, it is added to the 'busy cars' list.
- If the vehicle is not charging, its SOC is checked to see if it is greater than *low_threshold*. If so, it is added to the 'free cars' list; otherwise, it is added to the 'busy cars' list. If the SOC level is below the threshold and there are available charging stations, the vehicle is sent to the nearest available charging station using the **'Send to Charging Stations'** function.

Free Charging Stations

This function takes as input the robotaxi that needs to release the charging station and the timestamp when the event occurs.

- First, the function checks which node the vehicle is at, specifically the node corresponding to the charging station it needs to release.
- The number of 'available columns' at the corresponding station is increased by 1, and the number of 'occupied columns' is decreased by 1.
- The SOC level of the AV is updated based on the timestamp corresponding to the release of the charging station.

Send to Charging Stations

It takes as input the vehicle to send to the charging station and the timestamp.

- Stations that have a number of 'available columns' greater than 0 are selected and placed into a list.
- If the list is empty (meaning no station has available columns), nothing happens and the vehicle remains where it is.
- If the list is not empty, then iterate over all stations with available columns and check the distance of each station from the current position of the AV. Select the nearest charging station, also saving the time required to reach it by traversing the AV graph.
  - If the selected station is more than 20 minutes away in terms of travel time, then nothing happens and the vehicle remains where it is.

  - Otherwise, if the selected station is within 20 minutes of travel time, the vehicle is sent to that station. The following updates occur:
    * Update the number of 'available columns' and 'occupied columns' of the station.
    * Update the AV position.
    * Update the travel time.
    * Update all related parameters, including the SOC level once the vehicle arrives at the charging station according to equation (3.1).

Free Car Available

This function is used when the 'free cars' list is not empty, allowing a service to be initiated. It takes as input the timestamp when the travel request occurs, the starting position, and the ending position of the trip to be completed.

- For all vehicles in the 'free cars' list, the distance between their current position and the trip request location is calculated by traversing the AV graph. The nearest available vehicle to the trip's starting point is selected, and the time required to reach this point is calculated.
- The minimum time required to reach the trip's destination from the customer's starting point is calculated by traversing the AV graph.
- At this point, two distinct processes occur depending on whether the simulation is in one-layer or two-layer mode, as described in Section 5.1.2 and Section 5.1.3.
- At the end of the process, if an AV has been used, the following operations are performed:
  - Update the AV position.
  - Update the 'busy until' parameter of the vehicle with the timestamp when the vehicle will have arrived at its destination and will become available again.
  - Update the travel time.
  - Update all related parameters, including the SOC level, once the vehicle arrives at the destination node.
  - If the vehicle was charging prior to the call, the charging station is released using the function **'Free Charging Station'**.

Simulation

This part is the core of the model mobility simulation. It logically combines the previously described functionalities as follows:

1. First, the charging stations and AVs are initialized using the functions **'Initialize Recharging Stations'** and **'Initialize Cars'**.
2. Iterate over each row of the dataset containing all the trips to be fulfilled:
    - The timestamp of the trip request, the starting position, and the ending position of the trip are saved.
    - The function **'Check Free Vehicles'** is called with the timestamp of the request as input, which returns the 'free cars' list.
    - If the 'free cars' list is not empty, then the function **'Free Car Available'** is called with the starting and ending nodes of the trip as input.
    - Otherwise, two different processes are performed in the one-layer and two-layer cases, as described in Section 5.1.2 and Section 5.1.3.
3. After iterating through all the trips in the dataset (employed as if they where robotaxi demanded routes), the results are saved.

At the end of the simulation, the results are checked to see if they satisfy certain specific constraints. If they do not, some input parameters are updated and a new simulation is run. The operation of the simulator is depicted in Figure 5.1.

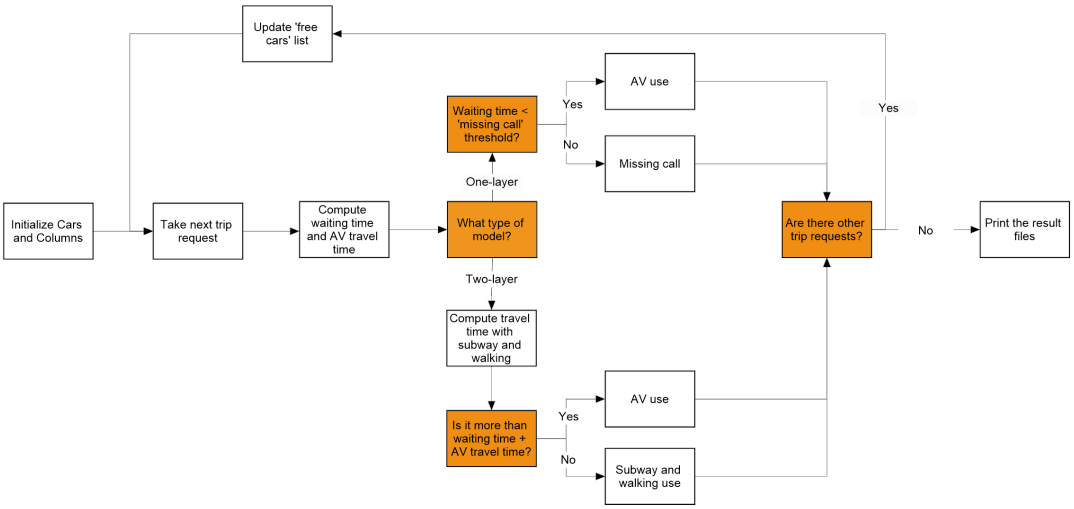


Figure 5.1: Diagram representing the simulation process.

5.1.1. Introduction of the cache memory for distances

The main problem encountered with this model is the high computational cost. It is necessary to compute the minimum path connecting two nodes of a graph several times, which is a rather expensive algorithm in the case of such large networks with more than 10,000 nodes and more than 30,000 edges. The most efficient algorithm for these calculations is Dijkstra's algorithm, which has a computational cost of $O(E + VlogV)$ for each iteration, where V is the number of vertices (or nodes) in the graph and E is the number of edges. Since we have to repeat the minimum path calculation several times for each trip request of our simulator, the computational cost is significant:

- For each trip request, the algorithm is used N times to calculate the path between each AV in the 'free cars' list and the point where the customer's request occurs in the AVs graph, in order to send the nearest free car to the starting point. Since all vehicles in the 'free cars' list are checked, N can be up to the total number of AVs in our simulation, in case in that specific timestamp all vehicles are available.
- For each trip request, the algorithm is used once to calculate the shortest path between the request point and the destination point in the AVs graph, to calculate the time in which the AV will reach its destination once it arrives at the customer.
- In the case of two-layer model, the algorithm is used once for each trip request to calculate the shortest path between the request point and the destination point in

the Walking and Public Transportation graph. This is used to calculate the time it would take the customer to use public transport instead of the robotaxi service, to see which option is more convenient for them.

- This process is repeated for each trip request present in the simulation, which includes those made over an entire week.

In light of these considerations, we obtain the following complexity estimate:

$$O(T * N * (E + VlogV))$$

Where:

- $T \rightarrow$ number of customer calls to fulfill.
- $N \rightarrow$ number AVs available at each timestamp.
- $E \rightarrow$ number of edges of the graph.
- $V \rightarrow$ number of vertices of the graph.

In a simulator-based optimization problem, this entire procedure might itself be iterated multiple times until the obtained result satisfies the imposed constraints, with input parameters being adjusted at each simulation iteration.

The complexity is too high: for a single simulation with more than 500,000 weekly requests and more than 1000 AVs, the total computation time exceeds 10 days, on a server with two Intel(R) Xeon(R) Silver 4214 processors, each with 12 cores. This is an unsustainable timeframe for conducting in-depth analyses.

The most efficient solution to this problem is to create a cache that has already been calculated and stored, with all the minimum distances between all possible pairs of nodes. This way, in the course of the simulation, all you have to do is to load this cache as a dictionary into the code, and whenever you need to calculate a minimum path, simply look up the value in the dictionary. Since the cost of accessing a dictionary entry is $O(1)$, the total cost is reduced to:

$$O(cache\_computation + T * N)$$

Complexity is thus converted from the temporal to the spatial context, i.e. we use a lot more memory than before, but the computational time is much smaller in return.

However, computation of the cache is a very expensive process. The number of possible

combinations of different nodes is in fact $N*(N-1)/2$, and Dijkstra's algorithm must be run once for each combination. However, by parallelizing the code on a virtual machine with 60 CPUs, we were able to compute it in a reasonable amount of time (less than a day).

In our model, the cache was computed only for the AVs graph, so we employ Dijkstra's algorithm once for each trip request (to find the minimum path from the request point to the arrival point by public transport), resulting in a total computation cost of:

$$O(cache\_computation + T*(N+E+VlogV))$$

Thanks to this, the computational cost has been reduced from several days to 2-3 hours for a single simulation. In the future, by creating a second cache that also stores distances for the Walking and Public Transportation graph, the simulation time can be further reduced.

5.1.2. One-layer robotaxi on-demand simulation

Two different types of simulation have been carried out. The first consideres the hypothesis of using only the robotaxi service to get to the desired destination, thus excluding the option of using public transport or walking. The idea is to simulate a robotaxi service alongside private transport, which remains significant. Therefore, within this scenario, a less extensive dataset of trips was used, with about 30,000 trips generated from the actual trips of 4,000 different private cars between 16 and 22 May 2022. These figures are in line with those of a normal car-sharing service, which in fact in 2019 recorded a number of around 17,000 daily trips made by 6 different companies, with a total of just under 3,000 trips per company per day. In this simulation, our service aims to cater for more than 4,000 trips per day, a number higher than the average number of providers of similar services. As private vehicles coexist with AVs, we have chosen to assume an average speed of 14 km/h for the robotaxi service in this case, slightly lower than that of normal vehicles (calculated at 14.91 km/h, as shown in Table 4.1). The idea is that autonomous vehicles have a lower top speed than conventional vehicles and therefore move at a slightly lower average speed in such a situation.

The structure of the code follows the one described above, with the majority of functions remaining unchanged. When a robotaxi is called from the customer, three different cases can occur:

1. If the nearest free AV is less than a certain threshold away, initially set at 5 minutes,

then the customer will wait for it and will be taken to their destination.

2. If the nearest free AV is more than a certain threshold away, initially set at 5 minutes, the customer will refuse the ride and our service will register a 'missing call'.

3. If there are no free AVs, the customer will not use the service and a 'missing call' will be recorded.

The only changes are therefore in the **'Free Car Available'** and **'Simulation'** functions:

- In the **'Free Car Available'** function, once the distance to the nearest free vehicle has been calculated, it is evaluated whether this is more or less than 5 minutes: in the first case a 'missing call' is recorded, in the second case the vehicle goes to the customer and serves the ride, at the end of which its parameters will be updated (node it is at, 'busy until' parameter, time spent travelling, SOC level, etc.).
- In the **'Simulation'**, in case the 'free cars' list is empty, a 'missing call' is recorded and the next 'call' from of the dataset is considered.

Once the simulation is complete on all trips in the dataset, the percentage of 'missing call' obtained, the percentage of vehicle overhead time and the average waiting time for customers are checked. All these parameters represent the constraints of our optimization problem:

- A 'missing call' percentage less than 5%.
- An average overhead percentage of all AVs less than 30%.
- An average waiting time less than 3 minutes.

And we want to obtain the minimum number of AVs that satisfies these assumptions. To do this, we check the value of these variables at the end of the process and, if they do not meet the requirements, we repeat the simulation by modifying one or more inputs.

5.1.3. Two layer on-demand robotaxi, pedestrians and public transport simulation

This second simulation has a different objective from the previous one. First, the Walking and Public Transportation layer is also introduced and linked. The number of trips in the model is much larger: data from 20 different weeks were merged, chosen so that there were no particular holidays or summer months that could alter the flow of traffic in the city. In this way, we obtained a data set of 559,144 trips, corresponding to the movements of about

82,000 different real vehicles. The idea is to simulate a service that is used in the central area of Milano, but where private mobility has been (at least partially) eliminated. For this reason, the AVs will travel at an average speed of 16 km/h, which is slightly higher than the current speed, as they are better able to cope with traffic congestion. With these figures, we simulate the service of more than 80,000 daily trips, a number that is still lower than the actual number of trips that take place in the city of Milano, but which is closer to the real figure. The number of trips was chosen by checking the number of daily entries in Area C of the city, as reported in Figure 5.2 [28].

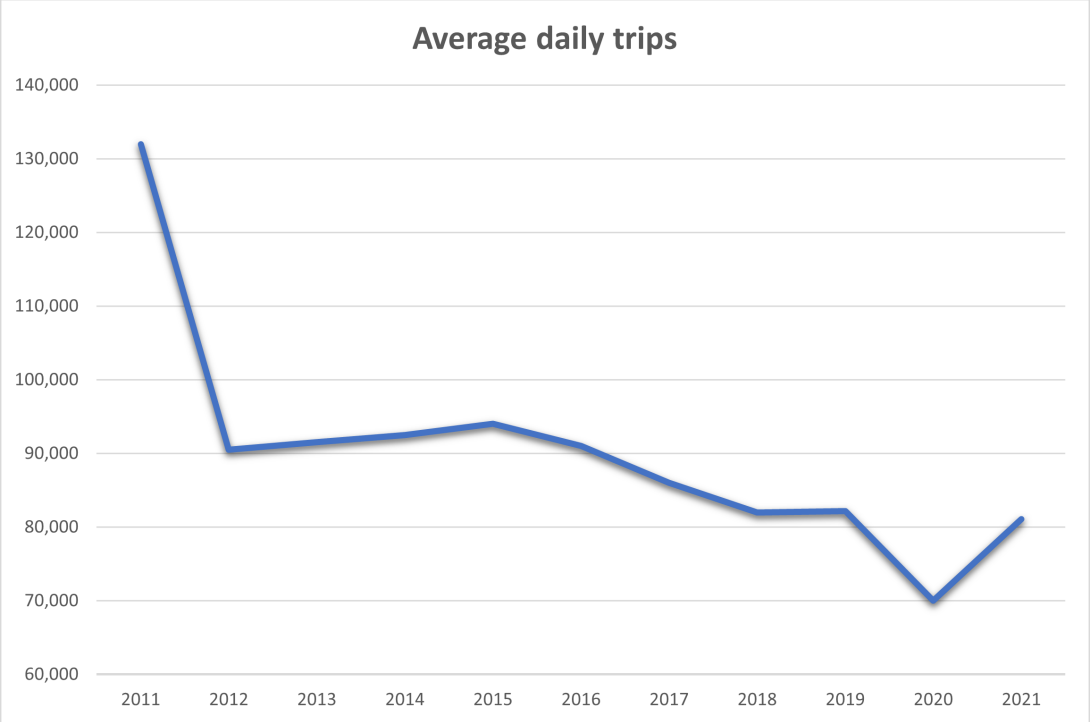


Figure 5.2: Evolution of the number of daily entries in the Area C of the city of Milano.

Again, as with the first simulation described, only slight changes are made to the code in the functions **'Free Car Available'** and **'Simulation'**. There are three cases that can occur:

- The time to reach the arrival point by public transport is shorter than the time it takes using AV, including the waiting time; in this case the user will use public transport.
- The time to reach the arrival point by public transport is longer than the time it takes with the use of AV, including the waiting time; the user will then use the sharing service.

- There are no free AVs and the user will have to use public transport.

The following changes are introduced into the code:

- Within the **'Free Car Available'** function, after calculating the distance between the point of request and the nearest free car, and the distance between the point of request and the point of arrival, the time that would be taken by public transport or on foot is calculated by calculating the minimum distance in the Walking and Public Transportation graph. If this is less than the sum of the waiting time and the walking time by car, public transport is used, otherwise the closest vehicle is used and the associated parameters are updated at the end of the trip.
- In the **'Simulation'** function, if the 'free cars' list is empty, the minimum path in the walking and public transportation graph will be calculated, forcing the user to use public transport to get around.

In this case, once the simulation is complete, we do not want to impose any particular constraints. In fact, the objective is to simulate the service with fleets of AVs of different sizes, starting from 1000 up to 2000 vehicles, and to assess how the efficiency of the service varies as investments and costs increase. The idea is to provide a company that wants to implement this sharing service with a tool that allows it to hypothesise investments based on the type of service it wants to provide, and to study the possible impact of the service following the introduction of various changes, such as the type of AVs in circulation, the speed or type of recharging, or the number of columns installed.

5.2. Optimization problem design and convergence

The two case studies analysed in this thesis have been described. With reference to the one-layer model, first we need to introduce a legenda:

- $N = AVs\ total\ number$
- $S = Recharging\ stations\ toal\ number$
- $N_s = Number\ of\ AVs\ in\ every\ recharging\ station$
- $C_s = Capacity\ of\ every\ recharging\ station$
- $L_n = AVs\ battery\ level$
- $T_n = AVs\ battery\ threshold$

The simulator-based optimization problem, can be formulated as:

$$\min_{N} \quad N$$

$$\text{s.t.} \quad x_1, x_2, x_3, x_4 \quad \text{holds}$$

Where:

- $x_1 = AV\ graph\ constraints$; that is, all vehicle movements must occur via AVs layer street graph.
- $x_2 = N_s \leq C_s \quad \forall s \in S$; that is, for each station, the number of AVs recharging must be less than the total capacity of the station.
- $x_3 = L_n \geq T_n \quad \forall n \in N$; that is, for each AV available, the battery level must be more than the threshold.
- $x_4 = missing\ call\ percentage \leq 5\%$

For the two-layer model, we simply run a simulation without any constraints, save the results, and then compare the data from different simulations with different numbers of AVs. In fact, we run simulations by setting the number of AVs as a variable input, starting from a fleet size of 1000 and going up to 2000. For each simulation, we want to collect the data and evaluate the effectiveness of the service offered, to understand what the optimal number of AVs is to serve the area in question. Then, the results are analyzed one by one and conclusions are drawn. There is no need to worry about convergence.

Returning to the one-layer model, we have shown in Chapter 2 that the problem under consideration is a CDPI composed of interconnected MDPIs. Therefore, we only need to check that the map h connecting the poset of resources to that of functionalities is Scott continuous to ensure the existence of the optimal value. In our case, the poset of resources is the number of AVs, i.e. the set $\mathbb{N}$ of natural numbers; the functionalities to be satisfied are the travel requests received at a given time, i.e. the set $\mathbb{R}$ of real numbers. Both sets in question are clearly complete directed set, so two DCPOs. Given any subset of AVs, they will produce a set of satisfied travel requests, also a directed set. This satisfies the sufficient conditions for Scott continuity.

Having demonstrated this, an algorithm must be constructed to achieve the optimum. Since one must simply optimise on the number of AVs, a simple Binary Search algorithm (Figure 5.3) was chosen:

1. Let's start by setting the optimal number of AVs that satisfy the constraints to zero.
2. We choose a range of values within which to search for the optimal number of AVs

that satisfy the constraints.

3. We calculate the midpoint (approximate integer) of our range and use it as the value for the simulation. Two things can happen at this stage:
   - If the chosen value does not satisfy the constraints, then we modify the range, setting the initial value as the value itself increased by 1 and leaving the final value of the range unchanged. We will search for good ones using more AVs than now.
   - If the value satisfies the constraints, we modify the final value of the range by decreasing the value itself by one. We also update the optimal number of AVs with the value obtained. We have already found a value that satisfies the constraints, so we will see if it is possible to satisfy them by using fewer AVs.
4. Start again from step 1 with the new range thus calculated, until it contains only 1 element. In that case, the optimal value obtained will be reported.

If, at the end of the algorithm, the optimal number of AVs is still zero, it means that, within the chosen range, no value solves the constraints, otherwise I have obtained the optimum.

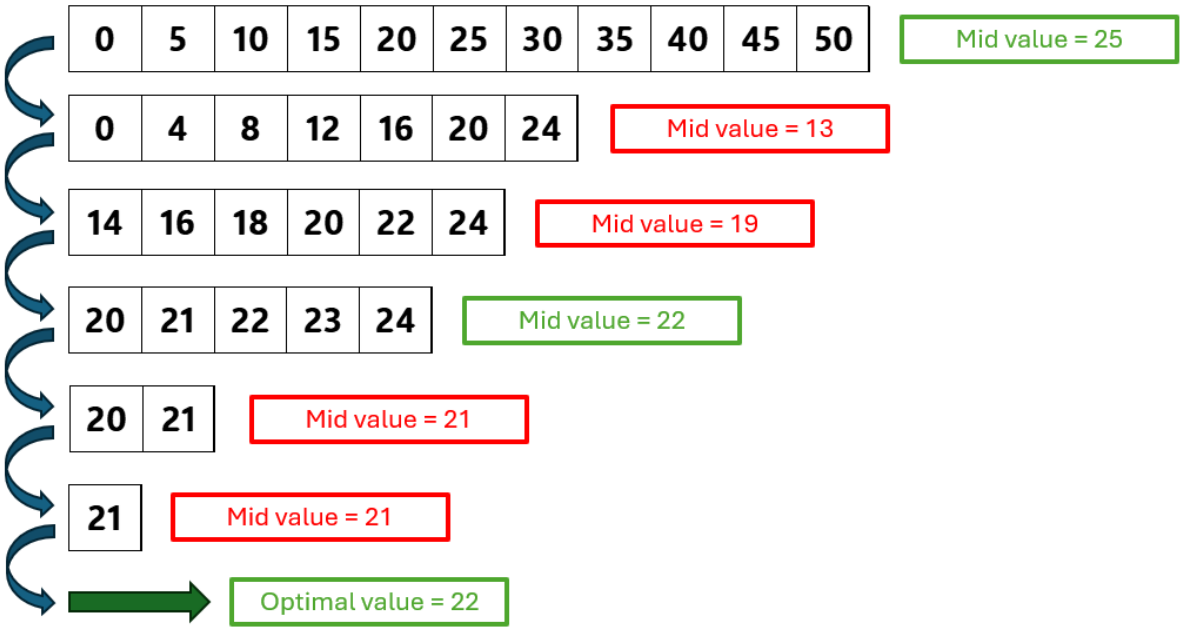


Figure 5.3: Example of optimization with Binary Search algorithm in a range from 0 to 50. A green value means that the constraints are satisfied, a red one that the constraints are not satisfied. The optimum value is obtained after 6 iterations.

The complexity of the Binary Search algorithm is logarithmic $O(\log_2 n)$, where n is the length of the search range. Since in the case of the one-layer model, using the cache, one no longer has to solve Dijkstra's expensive algorithm, running a code simulation is very fast and has a cost of $O(T * N)$, where T is the number of customer trips to fulfil and N is the number of AVs in the simulation. Finding the optimum therefore has a total cost given by:

$$O(\log_2(R) * T * N)$$

R is the length of the range, N is, in the worst case, the maximum value of the range. In our case, considering a database of 29,521 trips and an initial range of AVs [50, 1000], we have $R = 950$, $N = 1000$, and we obtain the optimum after 11 simulations, for a total time of less than 4 minutes.

6 | Results of the co-design of a urban on-demand robotaxi fleet cohabiting with walking and subway transportation

The following chapter presents all the results of the model. The two frameworks described and simulated, one-layer and two-layer networks, are analysed, verifying the effectiveness of the services and the costs involved, and introducing a possible tariff for customers who decide to use those ways of moving instead of buying a private car. Meanwhile, an analysis of the ratio of required AVs to private cars currently in use will be carried out to assess how much vehicles on the road could be reduced.

6.1. Simplified setting: one mobility layer

In this section, the results obtained with the one-layer model are presented. This model uses only robotaxis for transportation, aiming to emulate the operation of an autonomous taxi service. The service area radius is set at 6 km, intending to cover the entire urban area of Milano, corresponding to the current Area B. The number of installed charging columns is also fixed at 500. Managing a limited number of robotaxis, this number of columns ensures that the investment required for the construction and maintenance of charging stations is not overly burdensome. Additionally, it allows for the distribution of stations across the entire service area, ensuring that vehicles can always find a nearby charging point regardless of their location. The goal is to determine the optimal number of AVs required to meet the following constraints:

- A 'missing call' percentage less than 5%. A 'missing call' occurs when the waiting time for the customer is greater than a threshold, set at 5 minutes.
- An average overhead driven distance percentage for of all AVs less than 30% of the

overall distance.

- An average waiting time less than 3 minutes.

In Table 6.1, the optimal results obtained from the model simulator are shown, with the robotaxi speed set at 14 km/h.

Radius	AVs number	Missing calls	Threshold	Overhead %	Avg distance traveled per AV
6 km	**187**	4.93%	5 min	14%	60.48 km per day

Avg waiting time	Avg daily calls	Avg daily time in service	Original private cars	Ratio
105 sec	22 per AV	4h 19 min	4099	1/22

Table 6.1: Results of the simulation with a service area of 6 km radius.

The ratio indicates the relationship between the optimal number of robotaxis found and the number of original private vehicles. We note that the reduction in the number of vehicles on the road is over 95%, and despite this, the service remains highly efficient. However, the service could be further improved: for instance, the daily distance traveled by a single robotaxi is not very high, around 60 km, as is the average number of trips served by each AV per day (22). Increasing these values would allow for a reduction in the price of a single trip for a user, as we will see later. To achieve this, we can try to relax some constraints.

By slightly modifying the constraint on 'missing call', i.e., setting the threshold value to 8 minutes, we obtain the results shown in Table 6.2.

Radius	AVs number	Missing calls	Threshold	Overhead %	Avg distance traveled per AV
6 km	**118**	4.85%	8 min	22%	105.18 km per day

Avg waiting time	Avg daily calls	Avg daily time in service	Original private cars	Ratio
174 sec	34 per AV	7h 30 min	4099	1/34

Table 6.2: Results of the simulation with a service area of 6 km radius and the 'missing calls' threshold modified at 8 minutes.

The results of the study enable a number of very interesting considerations. In terms of the customer experience, the service was found to be less efficient, with an increase in the average waiting time of over a minute, from 105 seconds to 174 seconds. Furthermore, the overhead costs were found to have increased, rising from 14% to 22%. In fact, AVs

will spend proportionately more time reaching customer calls, increasing the time spent on the road with no one in the vehicle. Conversely, we are able to make better use of our robotaxis by having them travel almost twice as many kilometres per day and by having each of them serve more than 50% more calls per day than before (from 22 to 34). This also allows us to make the most of the potential of the batteries of our AVs. Each has a range of approximately 290 km on a full charge, and an average battery can withstand approximately 1200 recharges before needing to be substituted. With a simple multiplication, we obtain:

$$battery\ life\ in\ km \approx 1200 * 290\text{km} \approx 348,000\text{km}$$

In this way, we obtain, the number of years of battery life:

$$battery\ life\ in\ years \approx \frac{battery\ life\ in\ km}{avg\ yearly\ distance\ traveled\ per\ AV}$$

Assuming that over the course of the year, the robotaxis travel the same distance as observed during the studied week, this results in:

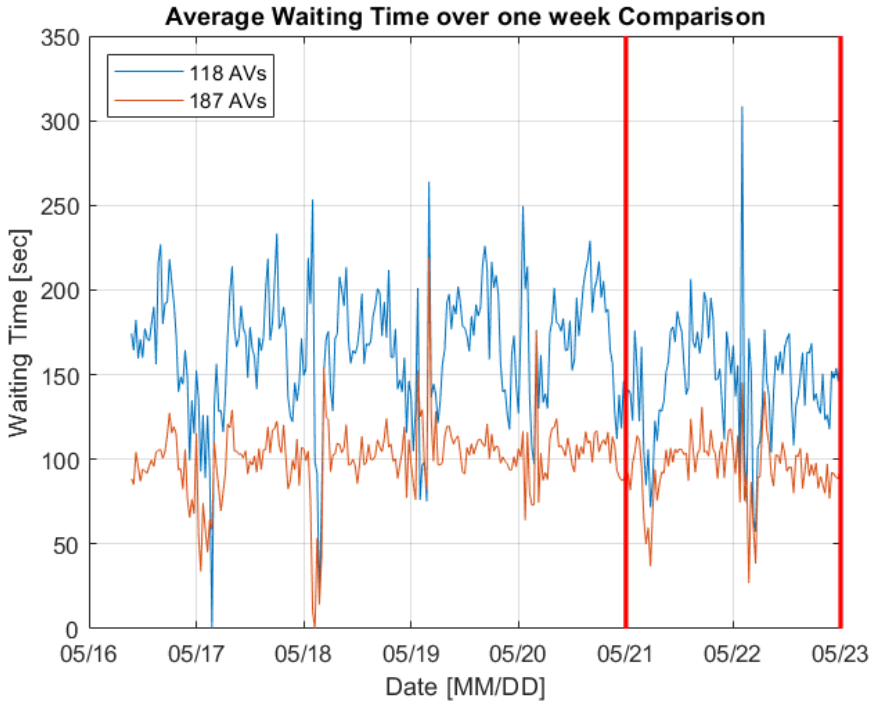
- In the simulation with a 5 minute threshold, the average distance driven in one year is 22,017 km, and the battery life approximately 15 years.
- In the simulation with a 8 minute threshold, the average distance driven in one year is 38,286 km, and the battery life approximately 9 years.

The lifespan of a robotaxi, however, does not depend only on this factor. Vehicle wear and tear must be considered, along with the fact that the battery gradually loses performance over the years regardless of its usage. For this reason, it is presumed that the fleet should be renewed at least every 10 years in any case. So the second simulation demonstrates a significant reduction in resource optimization. In the former case, the AVs would not be utilised to their full capacity. In the latter case, however, they would be. This can also result in significant cost savings, which will be analysed in detail later.

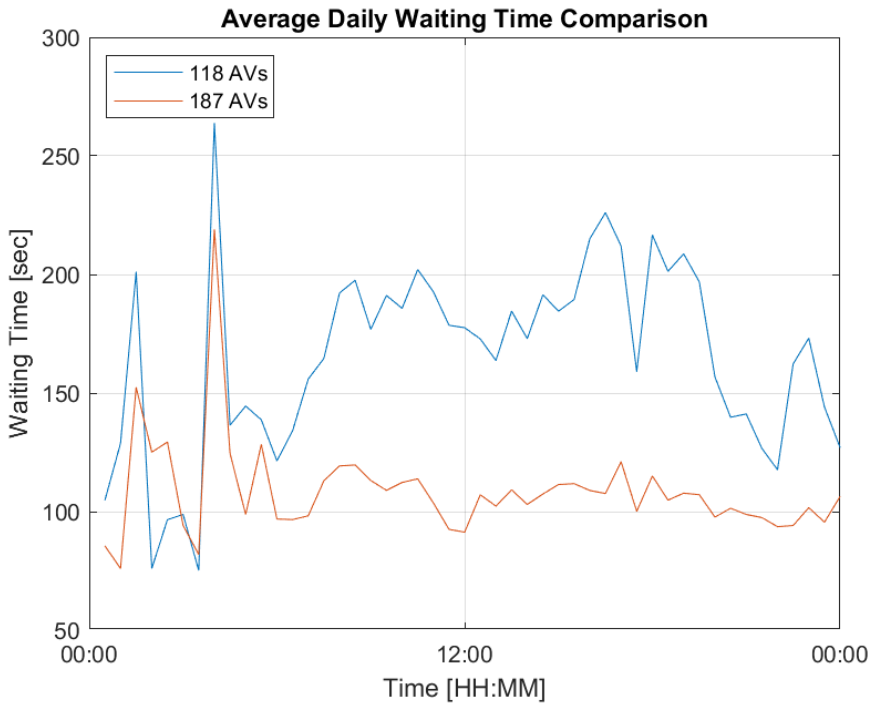
6.1.1. Analysis of the resulting sizing of AVs fleet

To discuss the effectiveness of the service, it is necessary to visualize how customer requests are fulfilled throughout the entire week and on a particular day, checking for any service disruptions at specific times.

First, let's examine how the average waiting time varies throughout the days of the week in the two cases analyzed as depicted in Figure 6.1.



(a) Weekly data from 16/5/2022 until 22/05/2022.



(b) Daily data of 19/5/2022.

Figure 6.1: Variability of the average customer waiting time for a robotaxi. Above are the results obtained for the entire week, with vertical red lines indicating the start and end of the weekend. Below is a focus on daily waiting times, specifically on 19/5/2022.

It can be observed that the only marginal differences over the course of the week are between weekdays and holidays. There is a slight decrease in waiting times over the weekend, which is attributed to the lower demand for trips. The simulation with the fleet of 118 robotaxis reveals a peak in waiting time close to the beginning of the night-time hours. This is attributed to an excess of robotaxis being sent to the charging stations, with a shortage of available vehicles to serve customer trips. This phenomenon is most pronounced on Saturday evenings, when the number of people to be catered for is at its highest. Consequently, a minor alteration to the charging process of the AVs could be beneficial, particularly on days when there is a greater concentration of nightlife.

On a daily basis, there is a discernible increase in demand at specific times, such as in the morning when many individuals are commuting to work and in the late afternoon when the working day draws to a close. In general, the fleet of 187 AVs is better equipped to accommodate customer demand at all times of the day, resulting in a more constant waiting time for service at all times.

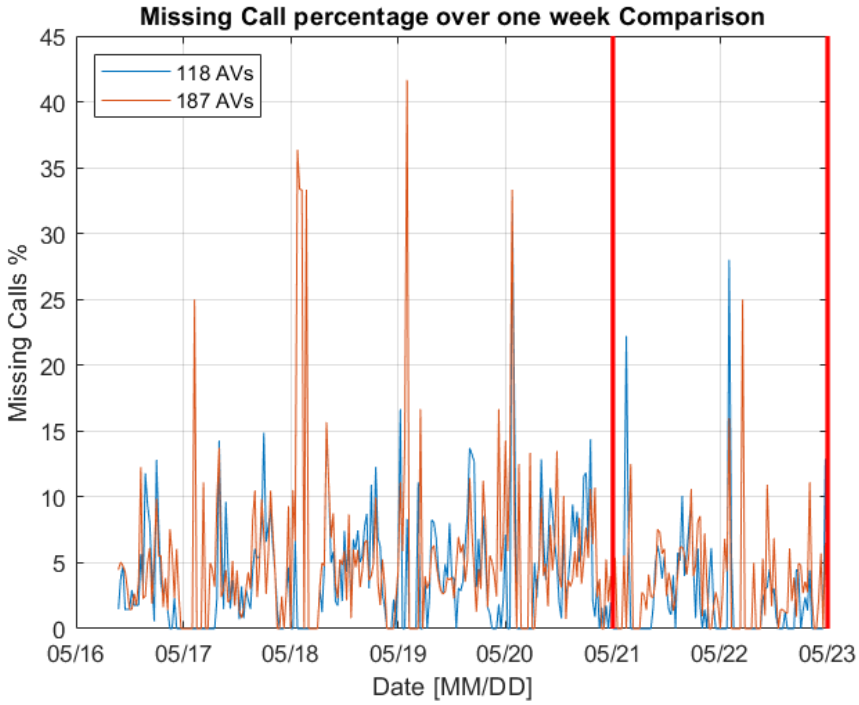


Figure 6.2: Variability of the 'missing call' percentage over the entire week. The vertical red lines indicate the start and end of the weekend.

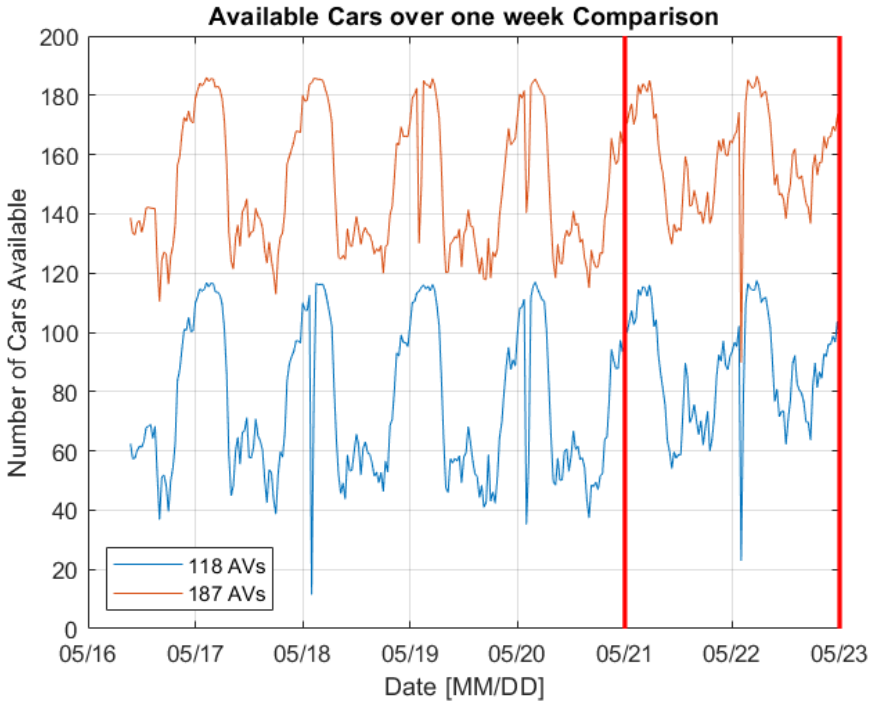
In Figure 6.2, the percentage of 'missing calls' during the week is displayed instead. As previously observed, the availability of vehicles at night is insufficient to meet demand. The fleet of 187 robotaxis is observed to register a higher percentage of 'missing calls' at

these times. This may initially appear to be a contradiction, but it is not. In fact, in this case, a more restrictive threshold has been set at 5 minutes, which means that the low number of AVs in circulation is often unable to arrive in this limited time to serve a customer who is far away. Conversely, with a threshold of 8 minutes, the probability of the service reaching him is increased. In this instance, two distinct modifications could be postulated:

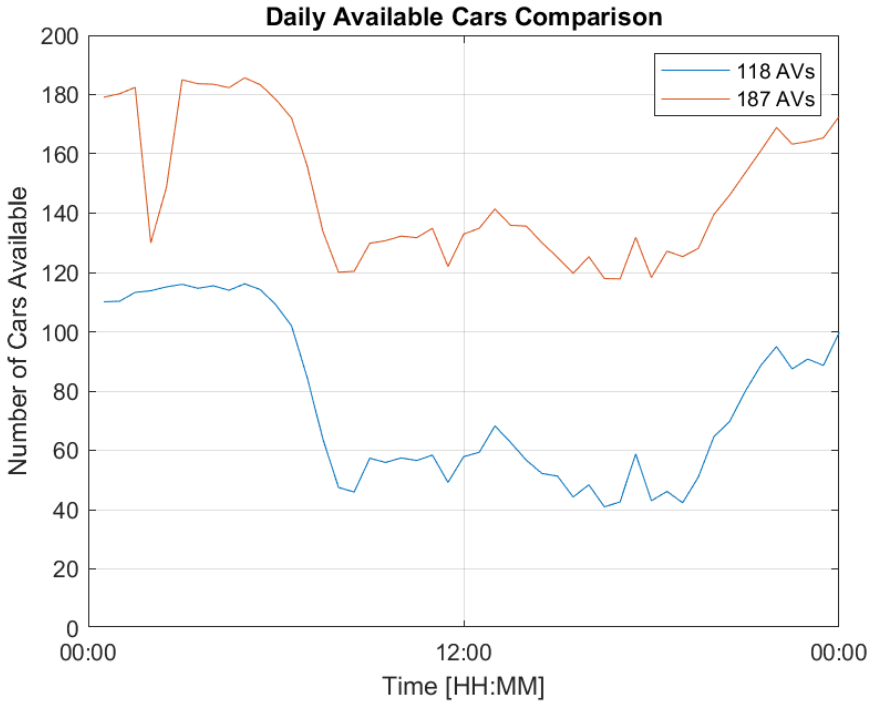
- Adjust the charging mechanism for overnight hours, aiming to ensure a minimum number of AVs in circulation.
- Increase the threshold set for 'missing calls' during night-time hours. It is reasonable to assume that customers are willing to wait a bit longer in these cases to use the service.

However, since there are significantly fewer requests during night-time hours, the number of 'missing calls' generated does not significantly impact the overall percentage.

Finally, in Figure 6.3, we present a visual representation of the number of vehicles available to users over the course of a week and a simulation day. We can clearly see that the fleet of 187 robotaxis is far too large for the demands we get from customers. Indeed, the number of available vehicles rarely falls below 120 in each considered timeframe. The utilisation of the fleet of 118 AVs is more optimal. This is due both to the greater use of the vehicles both to the bigger number of recharges that they undergo. Consequently, at certain times, there are approximately 40 available vehicles, which represents approximately the 30% of the total. This results in an optimal balance between the number of vehicles available to users and the utilisation of those vehicles by customers.



(a) Weekly data from 16/5/2022 until 22/05/2022.



(b) Daily data of 19/5/2022.

Figure 6.3: Variability of available cars over time. Above are the results obtained for the entire week, with vertical red lines indicating the start and end of the weekend. Below is a focus on daily waiting times, specifically on 19/5/2022.

6.1.2. Economic analysis of service costs and pricing

The robotaxi service provider has to make a substantial initial investment, which includes:

- Installation of 500 charging stations, costing €3,500 each, with a lifespan of approximately 7 years.
- Soil rent, costing approximately €420 per charging column annually.
- Purchase of AVs, costing approximately €50,000 each with a lifespan equal to the battery duration (or 10 years if the lifespan exceeds this threshold).

In addition, the following operational costs must be considered:

- A cost of €0.50 per kWh for vehicle charging.
- An annual maintenance cost for AVs of approximately €750, excluding charging expenses [29].

Therefore, we obtain the following costs:

- Fixed costs:

$$columns\ cost = \frac{columns\ number * columns\ installation\ cost}{columns\ life\ in\ years * 52}$$

$$soil\ cost = \frac{columns\ number * yearly\ soil\ rent}{52}$$

$$AVs\ cost = \frac{AVs\ number * AVs\ purchase\ cost}{AVs\ life\ in\ years * 52}$$

- Operational costs:

$$recharging\ cost = energy\ cost * weekly\ distance\ (in\ km) * average\ consumpion\ (in\ kWh/km)$$

$$maintenaince\ cost = \frac{AVs\ number * yearly\ maintenaince\ cost}{52}$$

In Table 6.3 the values of all the above mentioned parameters are reported, for the two considered simulated models. The AVs life has been calculated considering the minimum value between 10 years and the battery lifespan based on the average kilometers traveled by the robotaxis.

AVs number	Columns number	Columns installation cost	Energy cost
187	500	€3,500	€0.50 per kWh
118	500	€3,500	€0.50 per kWh

AVs number	AVs purchase cost	AVs life in years	AVs weekly distance
187	€50,000	10	79,117 km
118	€50,000	9	86,880 km

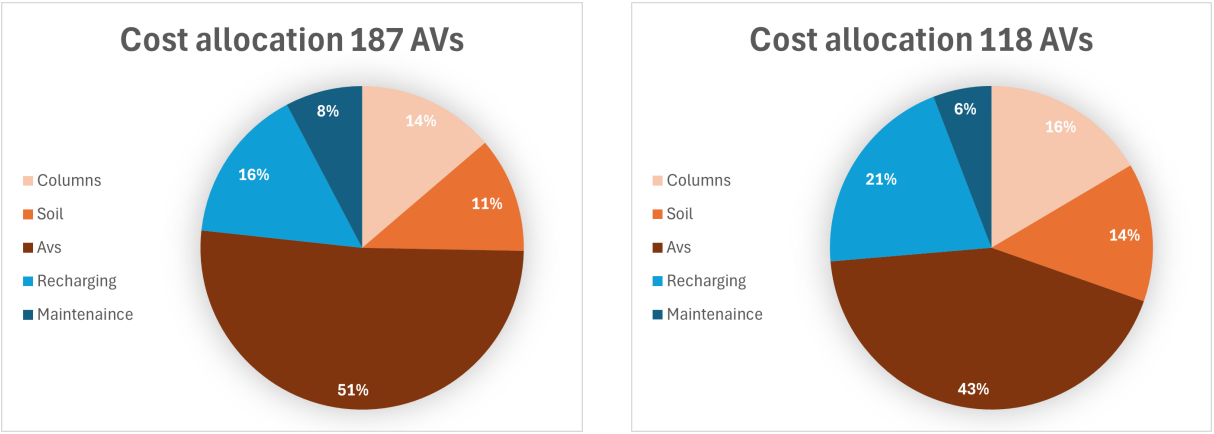
AVs number	Yearly soil rent	Average consumpion	Maintenaince cost
187	€420 per column	0.1375 kWh/km	€750 per year
118	€420 per column	0.1375 kWh/km	€750 per year

Table 6.3: All the costs incurred in the two one-layer simulations conducted.

By plugging these parameters into the previous equations, we obtain the following costs:

- For the first model with 187 AVs, a total of €34,967.43 of weekly costs.
- For the second model with 118 AVs, a total of €29,127.91 of weekly costs.

Figure 6.4 depicts the breakdown of costs relative to the total. We observe that in the second simulation, the costs associated with vehicle purchases decrease notably, from 51% to 43% of the total.



(a) 187 AVs (b) 118 AVs.

Figure 6.4: Cost allocation for the robotaxi service.

It is assumed that the total revenues obtained from the service are at least 50% higher than the listed costs, as the provider may have to make other expenditures, such as personnel costs. It can be observed that customers undertaking shorter trips incur a greater overhead time than those undertaking longer trips. Consequently, it is logical to include two distinct types of costs for the users:

- Unlock fee, namely the cost charged to the user at the beginning of the trip.
- Usage fee, namely the cost charged for the actual use of the vehicle, which varies depending on the trip duration.

In the case-study, during a week are registered:

- 28,000 trips made by customers.
- about 67,840 km traveled with passengers on board the vehicle, corresponding to 4846 hours and 50 minutes of travel time.

We propose the following rate schedule:

AVs number	Unlock Fee	Usage Fee (per min)	Weekly Revenue
187	€0.75	€0.11/min	€52,989.10
118	€0.75	€0.08/min	€44,264.80

Table 6.4: Proposed rates for the robotaxi service.

The prices listed in Table 6.4 would result in revenues that are more than 50% higher than the service costs previously analysed. In comparison to the average rates of the actual car sharing services, which range from €0.19/min to €0.29/min and unlock fee of €1, the proposed rates would be considerably lower. In the former case, the usage fee would decrease by over 40%, while in the latter case, it would decrease by over 55%. This is without considering the additional time and convenience advantages of not having to search for or park the vehicle. It is evident that by utilising the robotaxi fleet size of the second model, which optimises the potential of the AVs' batteries, it is possible to achieve more favourable pricing for users.

6.1.3. Study of the ratio between the robotaxi fleet size and the number of private cars replaced

This section describes an in-depth study of the reduction in the number of vehicles on the road that a robotaxi service would bring. The fleet of AVs would in fact replace a much

larger number of private cars, with numerous benefits in terms of consumption and space occupied. Today, vehicles spend much of their time parked, occupying public space and not making the best use of the resources used to produce them.

Obviously, the strongest the constraints we want our service to satisfy, the more AVs we will need, and thus the worse the gain in terms of this ratio. To show how this parameter can vary, we have used our one-layer model to examine two scenarios with different constraints on three different radius areas (3.5 km, 6 km and 14 km, Figure 6.5). Here the two scenarios are described:

1. In Scenario 1, we set the following constraints:
    - A required 'missing call' percentage less than 5%, it is considered a 'missing call' if the nearest available AVs is more than 5 minutes away from the location of the call.
    - An avarage overhead percentage of all AVs less than 30%.
    - An average waiting time less than 3 minutes.
2. In Scenario 2, we relax some constraints:
    - A 'missing call' percentage less than 5%, it is considered a 'missing call' if the nearest available AVs is more than 20 minutes away from the location of the call.
    - An avarage overhead percentage of all AVs less than 30%.
    - An average waiting time less than 5 minutes.

In Scenario 1, we have more restrictive constraints, which are well-suited for small areas like that of our case study, but not as much for larger areas with a 14 km radius. It is difficult to cover such a large area in a short time to serve a customer, and it would require a very large number of vehicles.

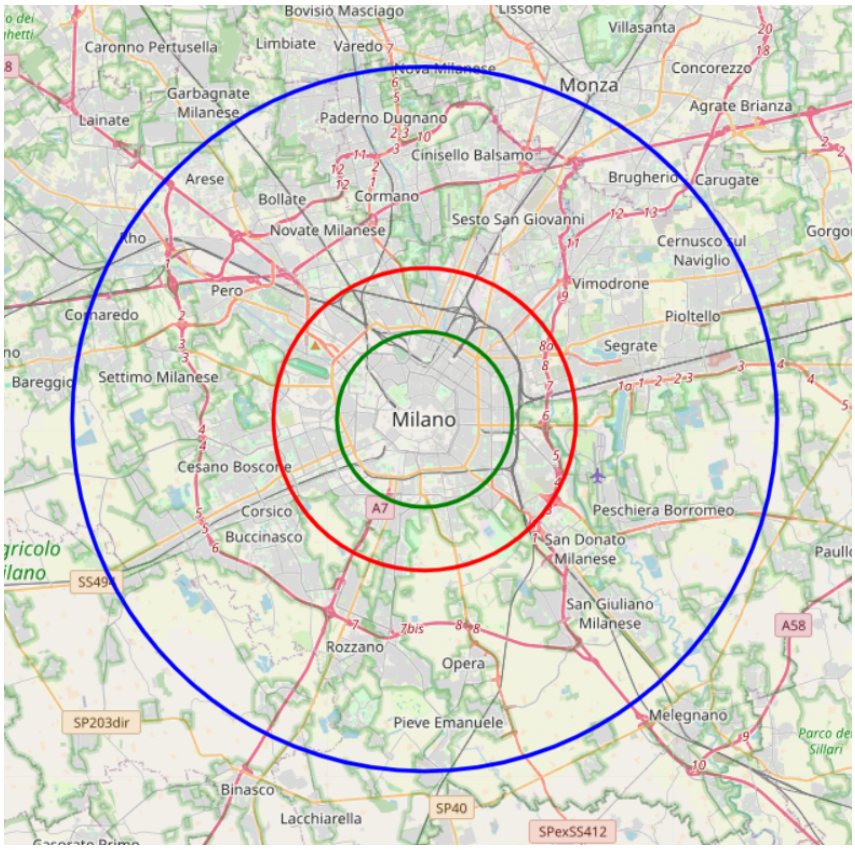


Figure 6.5: The three different service areas. In green the 3.5 km radius, in red the 6 km radius and in blue the 14 km radius.

Table 6.5 presents the results obtained for the Scenario 1.

Radius	AVs number	Missing calls	Avg wait	Overhead %	Original private cars	Ratio
3.5 km	106	4.96%	102 sec	13%	2601	**1/25**
6 km	187	4.93%	105 sec	14%	4099	**1/22**
14 km	4648	4.99%	83 sec	5.9%	28560	**1/6**

Table 6.5: Results of the simulation with 3.5 km, 6 km and 14 km radius in Scenario 1 are presented. The ratio between the robotaxi fleet and the number of private cars substituted by them is highlighted. The original private cars are those that the robotaxi service can substitute.

From these numbers, it is clear how, as the service radius increases, the number of AVs required to satisfy the same constraints grows much more rapidly compared to the number of vehicles replaced. In fact, the ratio between these two values increases with the radius. This result is not surprising, as such a service may perform better in a small, densely populated urban area where the number of trips to serve is high but within more limited

spaces. This setup makes it easier for one of the AVs to be in close proximity to the customer's request point and to reach that area quickly, thus avoiding a 'missing call'. This applies to both the simulation with a 3.5 km radius service area and that with a 6 km radius: they both serve a relatively small and densely populated urban area that meets these characteristics. The same cannot be said for the third considered radius, 14 km, where a much larger area is served, including various non-urban and sparsely populated spaces surrounding the city. Moreover, the 14 km/h speed used in the simulation is quite limiting, considering that travel speeds are generally much higher in peripheral areas. It is logical that using robotaxis in such a vast area is less efficient if the same service standards are to be maintained. We observe that the strongest constraints are too stringent for this larger radius: to ensure an AV arrives within 5 minutes to customers, a very large fleet is necessary, which nearly eliminates the advantage of using a car-sharing service, aimed at significantly reducing the number of vehicles on the road.

Furthermore, it is observed that the average percentage of AVs overhead is much lower as the service area increases. This is because AVs will need to fulfill much longer travel requests, remaining in service without carrying any passengers for longer periods. However, to maintain a waiting time of less than 5 minutes and avoid a 'missing call', the distances traveled to reach customers do not increase in the same proportion.

In Table 6.6, the results for Scenario 2 are included.

Radius	AVs number	Missing calls	Avg wait	Overhead %	Original private cars	Ratio
3.5 km	41	0.6%	253 sec	29%	2601	**1/63**
6 km	115	3.37%	236 sec	26%	4099	**1/36**
14 km	2600	4.99%	256 sec	18%	28560	**1/11**

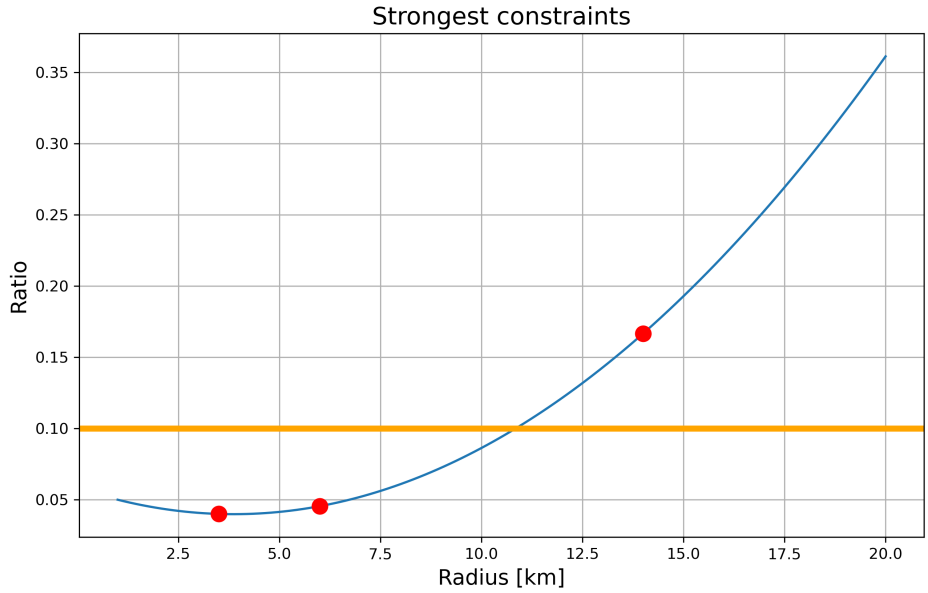
Table 6.6: Results of the simulation with 3.5 km, 6 km and 14 km radius in Scenario 3. The ratio between the robotaxi fleet and the number of private cars substituted by them is highlighted.

It is noted that by relaxing the constraints, the service performs more appropriately for the 14 km radius, ensuring a relatively small ratio between AVs and private cars replaced, which results in a significant reduction in vehicles on the road. Conversely, the relaxed constraints are not impactful enough for the urban radii of 3.5 km and 6 km. In these cases, assuming a customer is willing to wait 20 minutes for an AV's arrival is unrealistic and would lead to dissatisfaction with the service. It is observed that in these two scenarios, it is no longer 'missing calls' but rather the percentage of vehicle overhead that becomes the most limiting constraint. AVs, being able to reach customers within extended times up

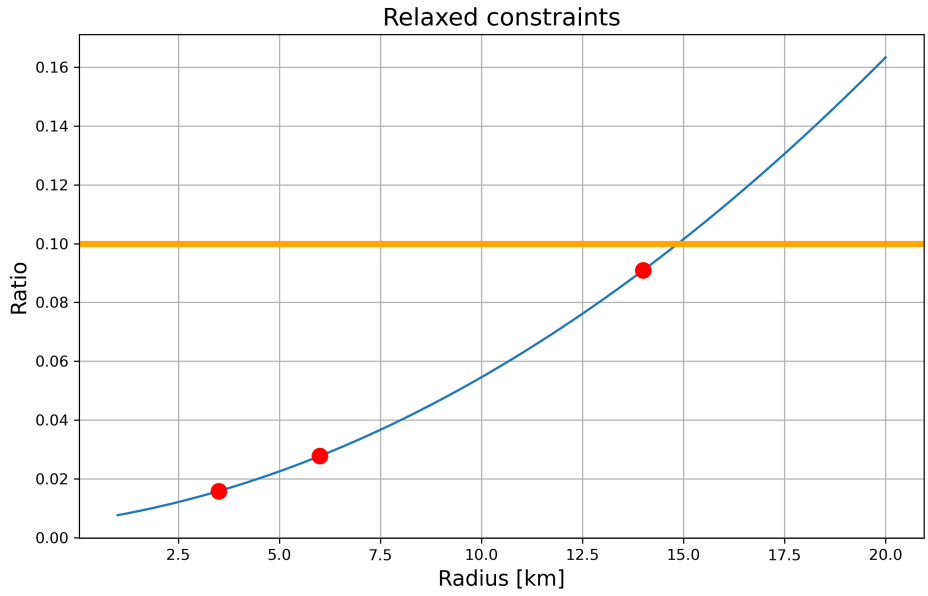
to 20 minutes, may spend most of their time traveling to customers rather than serving them, thereby wasting a substantial amount of their battery energy in a futile way.

From this analysis, it is understood that this ratio, although very important, cannot remain constant in any robotaxi service but depends on several variables, ranging from the geographical layout of the service area to specific efficiency choices of the service itself that we aim to satisfy. Therefore, it is crucial for a provider considering investment in this sector to plan ahead for the appropriate location and type of service implementation that best fits the chosen territory.

In Figure 6.6, it is observed that, with the chosen constraints for the robotaxi service fixed, the ratio between the number of AVs and the number of private cars replaced appears to exhibit quadratic growth as the service area radius increases.



(a) Scenario 1.



(b) Scenario 2.

Figure 6.6: Growth of the ratio between size of AVs fleet and number of substituted cars with respect to the radius of the service area. The orange lines indicate a 1/10 ratio, showing where the fleet size is significantly smaller compared to the total number of today cars as the service area expands.

A considerable reduction in the number of vehicles in circulation is achieved with a 1/10

ratio; higher values do not offer significant advantages compared to the current private mobility. As observed in Figure 6.6, this ratio corresponds to two different service areas in the two studied cases:

- In Scenario 1, it is achieved for an area with a radius of approximately 11 km.
- In Scenario 2, it is achieved for an area with a radius of approximately 15 km.

However, we note that in our case study with a service area radius of 6 km, the results are excellent, with a ratio of 1/22 between AVs and private cars in Scenario 1, obtained with very limiting constraints that force to have a really efficient service. This outstanding value is also due to the favorable urban layout where we are implementing the service. Milano is indeed a city well-suited for such a service, given its compact size and a large number of vehicles currently in circulation. These short distances can be well served by a robotaxi service, which performs less effectively in areas where urban areas become too extensive or roads are less congested.

6.2. Interconnected mobility: two layers of robotaxi, walking and underground

In this section, we analyse the results obtained for the two-layer model. The rationale behind this model differs from that of the previous one, which aimed to implement a robotaxi service similar to those currently in use. The objective of this model is to develop a robotaxi service that, additionally to walk and underground, replaces private urban mobility. As before, the service area of the AVs has a radius of 6 km, corresponding to Area B of the city. In this case, the walking and public transport layer is also included. Consequently, users will be able to move around within the specified area using either the robotaxi service or underground and walking. It should be noted that only the existing metro service is included as public transport for sake of simplicity. In the service area, 2000 charging columns are installed for the AVs to use, based on projects that envisage similar investments in the Milano area. The objective is to conduct a series of simulations for this model, varying the size of the robotaxi fleet, in order to ascertain the optimal number of AVs required to ensure a service with high efficiency standards and to analyse the associated behavior.

In Table 6.7 the results obtained from the 6 simulations are shown. The speed of the robotaxis was set at 16 km/h, slightly higher than the current average speed of cars in the same area of interest. Indeed, only autonomous vehicles would circulate on the streets, capable of intelligent communication among themselves to optimize travel times

by avoiding congestion and traffic.

AVs number	Public transport use %	Avg waiting time	Avg Δ time
1000	33%	340 sec	+43 sec
1200	16%	220 sec	-118 sec
1400	6.2%	133 sec	-233 sec
1600	2%	87 sec	-292 sec
1800	1.6%	75 sec	-305 sec
2000	1.3%	67 sec	-313 sec

AVs number	Overhead %	Avg distance per AV	Avg daily calls
1000	35%	228.36 km per day	53 per AV
1200	27%	202.05 km per day	56 per AV
1400	19%	167.20 km per day	53 per AV
1600	13.5%	140.28 km per day	49 per AV
1800	11.9%	122.50 km per day	44 per AV
2000	10.8%	108.94 km per day	39 per AV

AVs number	Avg daily time in service	Original private cars	Ratio
1000	14h 16 min	81,980	1/82
1200	12h 37 min	81,980	1/68
1400	10h 27 min	81,980	1/58
1600	8h 46 min	81,980	1/51
1800	7h 39 min	81,980	1/46
2000	6h 48 min	81,980	1/41

Table 6.7: Results of the simulation with different AVs fleet size. The 'Average Δ time' parameter indicates the differences (in mean) between the actual average travel time obtained using private cars and the average travel time with this robotaxi service.

It can be observed that all variables decrease as the AVs fleet size increases, although this decline is not uniform. Upon reaching a fleet size of 1600 AVs, the average waiting time begins to exhibit a slower decline, indicating that further investment in the number of robotaxis does not result in significant improvements in service efficiency. The 'Public transport use %' indicates the proportion of users who were not served by robotaxis but reached their destination independently, either by walking or using the subway, because

it was the fastest method or because no vehicles were available. It can be observed that, with 1600 AVs, the percentage of individuals utilising public transport is only 2%, which is notably low (Figure 6.7). The travel time for users of robotaxis, in comparison to those currently utilising private vehicles, is demonstrably reduced when a fleet of 1200 AVs is in operation. Considering that the average travel time with private vehicles is slightly over 15 minutes, we understand that using robotaxis can achieve a reduction of up to 33% (Figure 6.8). This figure represents a significant indicator of the potential of this kind of service. The enhanced traffic management capabilities of robotaxis and the elimination of the need to search for parking spaces would significantly accelerate urban travel for citizens. It is interesting to note that the number of 'Average daily calls' initially appears to remain constant despite an increase in the number of robotaxis in use. This is because, with a limited number of AVs, it is not possible to meet the demand from users who are reliant on public transport. Consequently, the number of requests served is lower and the daily rides of a single vehicle do not increase in comparison to the utilisation of larger fleets.

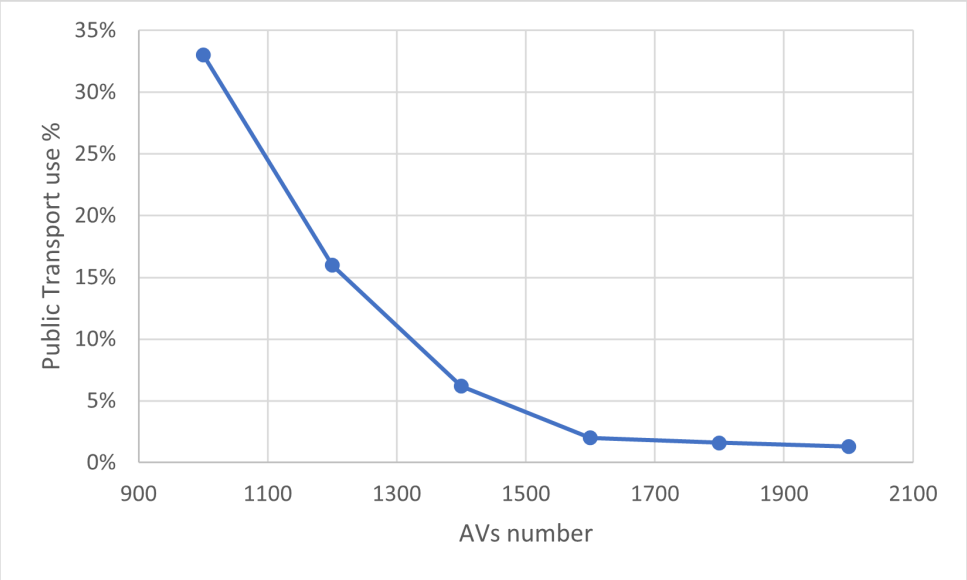


Figure 6.7: Percentage of users that reach their destination walking or using underground, with respect to the fleet size of robotaxis used in the simulation.

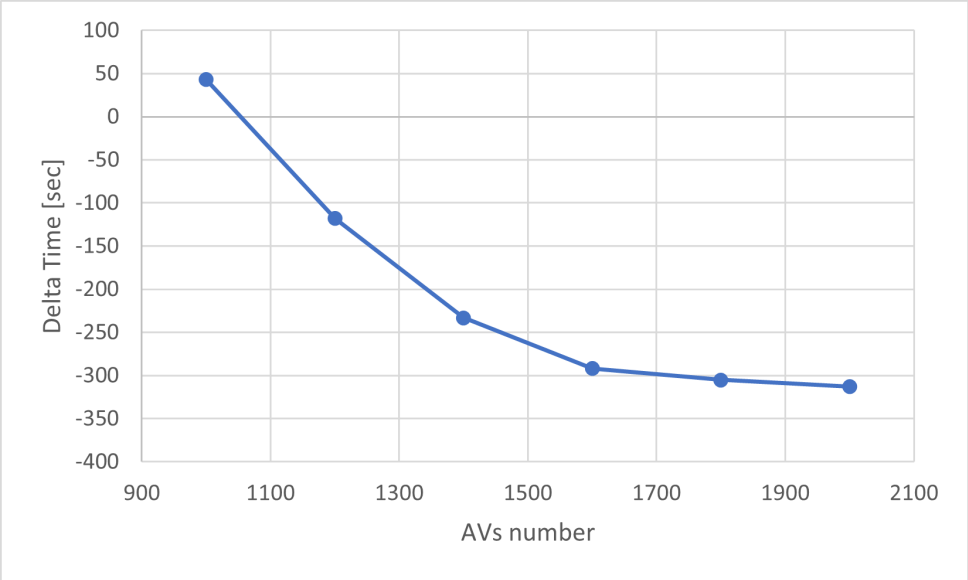


Figure 6.8: Trend of Δ time with respect to the fleet size of robotaxis used in the simulation.The function seems to approach a horizontal asymptote after the use of 1600 AVs.

6.2.1. Cost and efficiency analysis of different AV fleets

A provider who is considering investing in a robotaxi service such as the one described must not only evaluate the efficiency of the service, but also the costs involved. In addition to offering convenience to citizens, the use of AVs must also be cost-effective enough to persuade them to abandon use of private cars.

Similar to that shown in Section 6.1.2, costs are calculated for the two-layer model for each of the simulations performed. In Table 6.8, the parameters used to calculate them are shown.

AVs number	Columns number	Columns installation cost	Energy cost
1000	2000	€3,500	€0.50 per kWh
1200	2000	€3,500	€0.50 per kWh
1400	2000	€3,500	€0.50 per kWh
1600	2000	€3,500	€0.50 per kWh
1800	2000	€3,500	€0.50 per kWh
2000	2000	€3,500	€0.50 per kWh

AVs number	AVs purchase cost	AVs life in years	AVs weekly distance
1000	€50,000	4	1,598.55 km per AV
1200	€50,000	5	1,414.33 km per AV
1400	€50,000	6	1,170.40 km per AV
1600	€50,000	6	981.94 km per AV
1800	€50,000	6	857.52 km per AV
2000	€50,000	6	762.57 km per AV

AVs number	Yearly soil rent	Average consumpion	Maintenaince cost
1000	€420 per column	0.1375 kWh/km	€750 per year
1200	€420 per column	0.1375 kWh/km	€750 per year
1400	€420 per column	0.1375 kWh/km	€750 per year
1600	€420 per column	0.1375 kWh/km	€750 per year
1800	€420 per column	0.1375 kWh/km	€750 per year
2000	€420 per column	0.1375 kWh/km	€750 per year

Table 6.8: All the costs incurred in the two-layer simulations conducted.

It can be observed that, in each of the analysed cases, the robotaxis weekly distance traveled is high, resulting in a good optimization of the resources used to produce them. The battery is almost completely consumed before wear forces the renewal of the vehicle fleet. In any case, the maximum lifespan of a vehicle was set at 6 years, given the high utilisation of the fleet and the consequent need for frequent updates due to wear and tear.

Table 6.9 presents the total weekly costs of the simulations, disaggregated by category. An additional cost of €0.39 per user was calculated for public transport following consultation of the ATM data in [30]. Public transport is already operational, and the users in our model represent only an increase in service utilization compared to today, which would lead to a proportional increase in costs for the proper functioning of public transportation.

AVs number	Columns cost	Soil cost	AVs cost	Fixed costs
1000	€19,230.77	€16,153.85	€240,384.60	€275,769.20
1200	€19,230.77	€16,153.85	€230,769.20	€266,153.80
1400	€19,230.77	€16,153.85	€224,358.97	€259,743.59
1600	€19,230.77	€16,153.85	€256,410.26	€291,794.87
1800	€19,230.77	€16,153.85	€288,461.54	€323,846.15
2000	€19,230.77	€16,153.85	€320,512.82	€355,897.44

AVs number	Recharging cost	AVs Maintenaince cost	Operational costs
1000	€109,900.20	€14,423.08	€124,323.30
1200	€116,681.82	€17,307.69	€133,989.50
1400	€112,650.70	€20,192.31	€132,842.99
1600	€108,013.21	€23,076.92	€131,090.13
1800	€106,117.52	€25,961.54	€132,079.06
2000	€104,853.80	€28,846.15	€133,699.96

AVs number	Total Service cost	Public Transport cost	Total Investement
1000	€400,092.50	€71,943.30	€472,035.80
1200	€400,143.36	€34,814.60	€435,024.96
1400	€392,586.58	€13,516.62	€406,103.20
1600	€422,885.00	€4,360.20	€427,245.20
1800	€455,925.21	€3,488.16	€459,413.37
2000	€489,597.39	€2,834.13	€492,431.52

Table 6.9: Total weekly costs of the two-layer simulation divided by categories.

'Total Service cost' contains the sum of 'Fixed costs' and 'Operational costs'. 'Total Investment' is the sum of 'Total Service cost' and 'Public Transport cost'.

In the Figure 6.9 the cost trend is depicted as the number of AVs in the fleet changes. The 'Total Investment' decreases as the fleet size increases from 1000 to 1400 AVs. In fact, the 'Total Service cost' remain almost constant, while those related to public transportation decrease due to fewer users. Beyond this threshold, the 'Total Investment' starts to increase, as the number of AVs to purchase and maintain becomes more substantial.

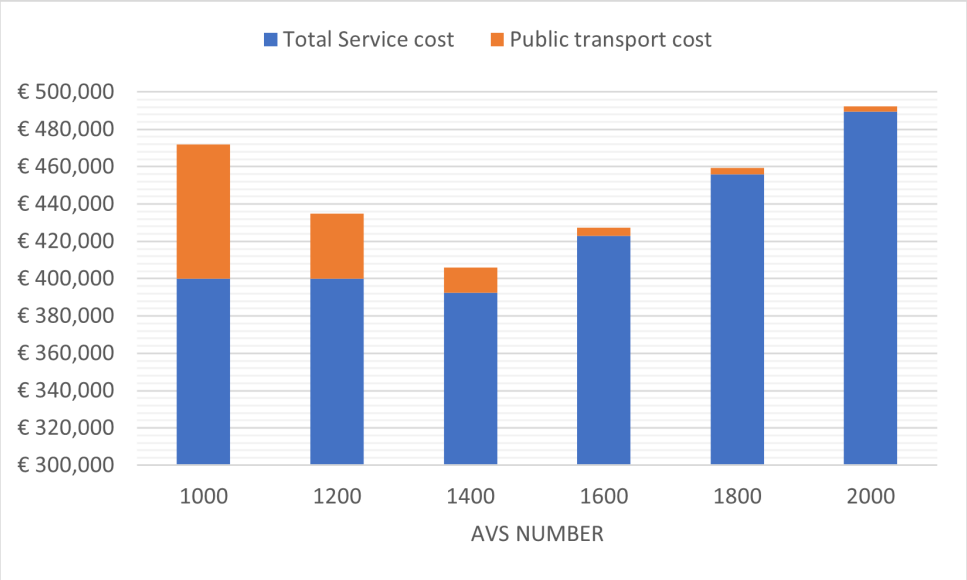


Figure 6.9: Variation in costs for the robotaxi service and for public transportation with changes in the AVs fleet size.

An optimal value of the AVs fleet size turns out to be of 1600 vehicles, which provides an excellent balance between cost and efficiency, guaranteeing not only an excellent average wating time (less than 90 seconds), but also a limited percentage of AVs overhead (13.5%). Let us show in Figure 6.10, how the costs related to this choice are distributed in the number of robotaxis. It can be seen that the cost for underground funcitoning is negligible (about 1%).

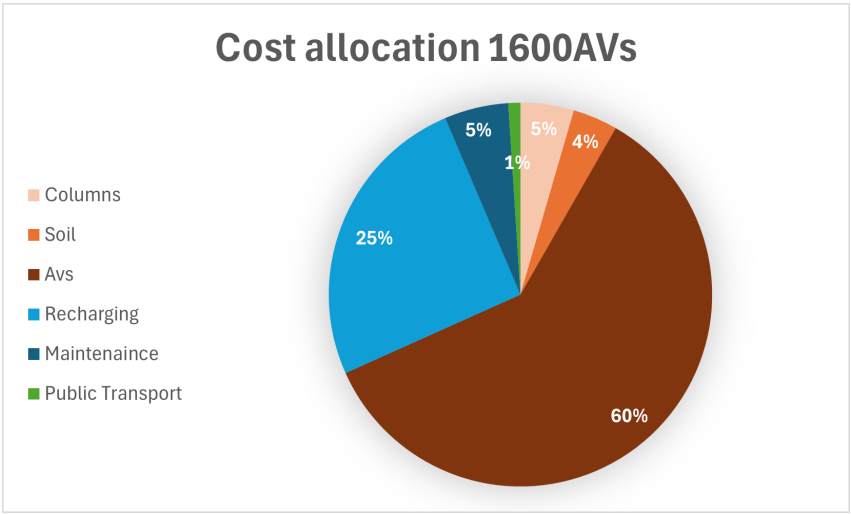
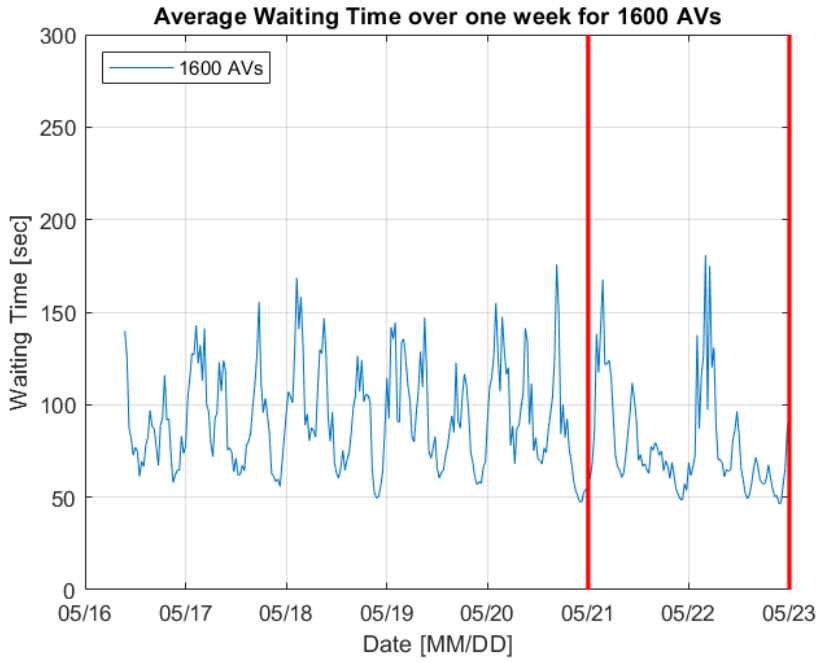


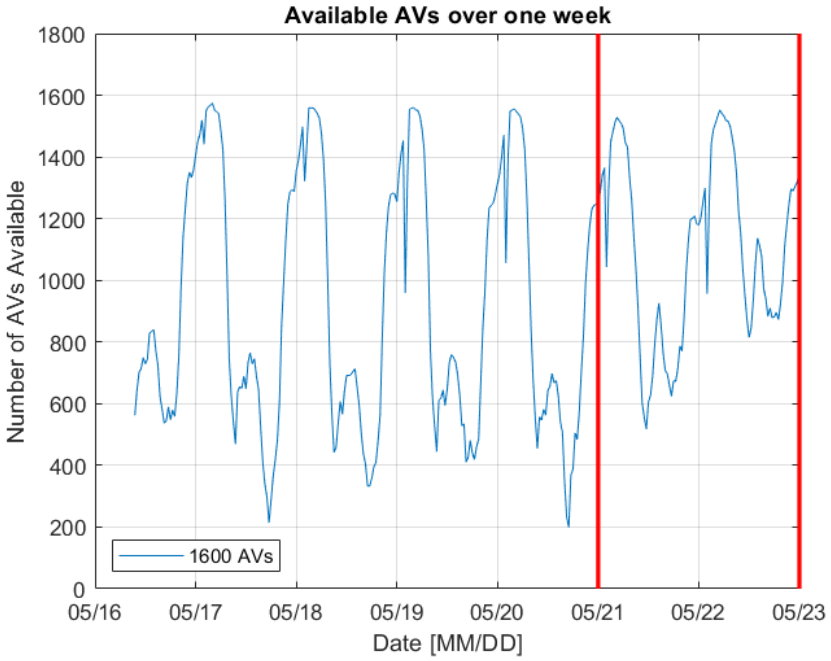
Figure 6.10: Variation in costs for the robotaxi service and for public transportation with changes in the AVs fleet size.

The variations in average waiting time and available vehicles over the course of a one-week simulation with regard to the utilisation of 1600 AVs in the two-layer model, i.e. the number considered optimal by our analysis, are shown in Figure 6.11.

It is notable that this robotaxi fleet is able to provide rides with minimal waiting times (on average less than three minutes at any time during the week under examination) and makes effective use of its resources (at certain times, less than 400 AVs remain available, representing 25% of the total). This impressive efficiency is further evidenced by an optimal ratio between robotaxis in use and private vehicles replaced. Each AV replaces approximately 50 cars previously in circulation, a ratio that would result in a notable gain in public land currently occupied by parking spaces. These spaces could be reused in a more beneficial and sustainable manner, potentially creating green areas or recreational spaces.



(a) Average waiting time.



(b) Average available cars.

Figure 6.11: Weekly data for the simulation obtained with 1600 AVs.

6.2.2. Economic analysis and study of the benefits of transitioning to shared autonomous mobility

Finally, it is necessary to analyse the cost of this service for the users who utilise it. In contrast to the one-layer scenario, this service is now conceived as a public service, implemented with the objective of ensuring more effective, safe, and sustainable urban mobility. Consequently, the service is designed with the objective of generating a lower profit than would be the case with a purely private investment.

In this instance, we may consider a very low unlock fee, aimed at preventing requests when they are not needed and covering the overhead cost, set at **€0.15**. Regarding the usage-based fee, it is calculated based on the distance traveled, allowing it to be subsequently compared with the current cost per kilometer of a private car. In this case, it is assumed that revenue will be 30% higher than costs, as other expenses, including personnel costs, must also be taken into account.

This yields a usage fee of **€0.35 per km**, equal to **€0.09 per minute**, for the robotaxi use. This would result in service revenues in excess of €560,000 per week.

In conclusion, it is necessary to evaluate the economic impact of such a service on citizens. One might inquire whether it is economically rational for users to relinquish their private vehicles and transition to fully shared mobility.

For purposes of comparison, consider a petrol car with the following characteristics:

- **Car Purchase:** €20,000
- **Urban consumption:** 18 km/L
- **Maintenaince cost:** €750 per year
- **Fuel price:** €1.80/L

To obtain a cost per km of the private car, we must calculate the depreciation of the vehicle over time. The most common depreciation models in the literature [31], follow an exponential trend, given by the following function:

$$V(j) = V(0) * \alpha(k)^j$$

Where $V(0)$ is the car purchase, $V(j)$ is the economic value of the vehicle on the market after j years and $\alpha(k)$ is a parameter that depends on the annual distance traveled k by the vehicle in question. In general α assumes values varying between 0.8 and 0.86, based

on the number of kilometers driven annually k. As k increases, α decreses, because higer usage leads to a lower market value of the vehicle.

Having done so, let us assume a time horizon of 6 years, equal to the lifespan of an AVs in the simulation with 1600 robotaxis. We calculate the following costs:

$$car\ depreciation = V(0) - V(6)$$

$$fuel\ cost = \frac{yearly\ average\ traveled\ distance\ (km) * 6}{urban\ consumpion} * fuel\ price$$

$$maintenaince\ cost = yearly\ maintenaince\ cost * 6$$

$$total\ cost = car\ depreciation + fuel\ cost + yearly\ maintenaince\ cost$$

$$cost\ per\ km = \frac{total\ cost}{yearly\ traveled\ distance\ (km) * 6}$$

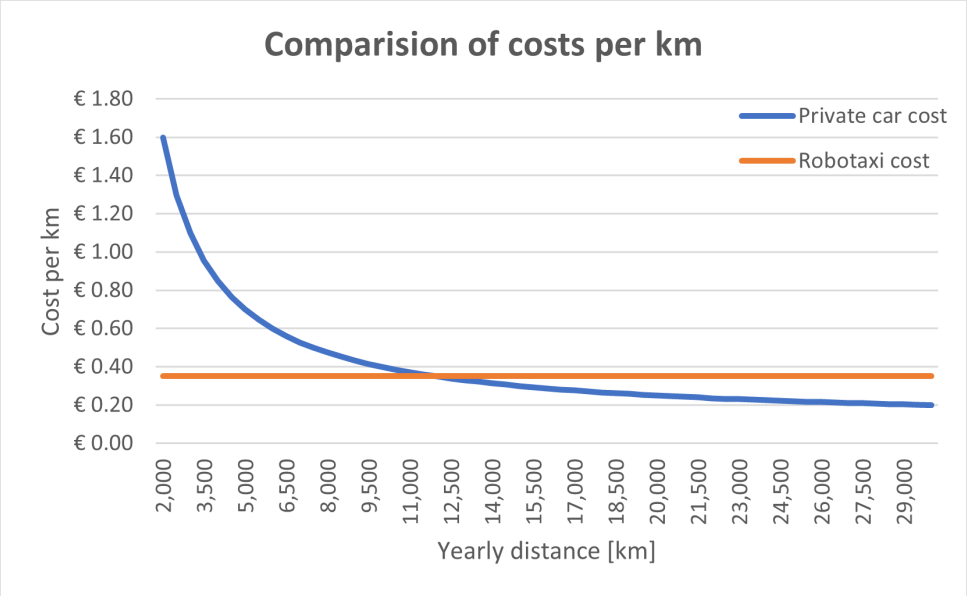


Figure 6.12: Comparision between the cost for the user per km of using robotaxis service or driving the private car.

As illustrated in Figure 6.12, the cost per km of a private vehicle exhibits a notable decline as the average distance travelled annually increases. The utilisation of robotaxis would result in savings for all those who utilise their vehicle for less than 11,500 km per year, while for longer distances, a private vehicle becomes a more cost-effective option. Given that the italian average yearly usage is 10,712 km per car [32], it would be advantageous to dispose of the majority of these vehicles and instead utilise an autonomous taxi service.

The advantages of this approach include both the speed of travel and the level of comfort experienced by passengers, as well as an economic benefit.

7 | Conclusions and future developments

This chapter summarizes the work conducted in this thesis, highlighting its most important and innovative contributions, the results achieved, and possible future developments.

This thesis aimed to answer the following questions:

- How would an "on-demand" autonomous mobility servece (robotaxi) operate within the urban area of Milano?
- What would be the efficiency of the robotaxi service in terms of fleet size and waiting time for users?
- What would be the costs incurred both by users and by a provider of this service?
- What would be the advantages and disadvantages of this new type of autonomous "on-demand" urban mobility?
- How could a robotaxi service be designed and integrated into existing public transportation?

By designing and operating a co-design model and optimizing it with a simulator-in-the-loop approach, which is based on real travel data from Unipol Group insurance customers black boxes, we were able to get the answers we were looking for.

First, we demonstrated how the urban mobility scenario can be framed as a problem based on Co-design Theory, thereby ensuring the achievement of a uniform and sistematic way of describing urban mobility context.

Subsequently, we defined how the autonomous mobility on-demand service would work, deciding on the type of vehicles to be used, the layout of the charging stations, the charging mechanism itself, and the associated costs. All these choices were made by reviewing the technologies currently available on the market and selecting the most appropriate ones based on the features we consider essential for the development of the service and the associated costs.

We then built the simulator for our model, which takes as input the dataset with the actual trips made by real drivers over a specified period (from 16 May to 22 May 2022), the number of charging stations and AVs to be used in the simulation, and the type of model to be used (one-layer or two-layer). The output allows us to assess the efficiency of the service and calculate its costs. It provides parameters such as the waiting time for each user request, the total travel duration to the destination, the number of available robotaxis at each time instant, the average overhead distance, the time spent in service for each vehicle, the total kilometers traveled, and much more. We have developed two different models, the first called one-layer model, where travel requests can only be served by robotaxis, and the second called two-layer model, where the desired destination can also be reached by walking or using the subway. In the case of the one-layer model, the output also provides the percentage of 'missing calls' produced, while in the two-layer model, it provides the percentage of trips that used public transportation instead of the robotaxi service.

Developing an ad hoc simulator, we were able to calculate how many vehicles our service would need in two different cases: one where we simulated a volume of trips similar to that of a current car-sharing service, and a second model where we assumed full shared mobility, where robotaxis completely substitute the private cars in the service area. This scenario also considered the possibility of utilizing public transportation or walking as alternative modes of travel. In the first case (one-layer model), we solved a true optimization problem, using a Binary Search algorithm, by determining the minimum number of robotaxis needed to achieve good service efficiency, defined as a 'missing call' percentage below 5%, an average overhead of less than 30% relative to the total kilometers traveled, and an average waiting time of under 3 minutes. In the second model (two-layer), we performed six different simulations with a variable number of AVs ranging from 1000 to 2000, analyzing all the efficiency indicators obtained and the associated implementation costs. This allowed us to identify the robotaxi fleet size that provides the best balance between expenses incurred and customer satisfaction.

Finally, the results obtained were analyzed, identifying the main advantages of a robotaxis service and the potential limitations. Additionally, a possible pricing model was developed to compare the current costs of maintaining a private car with the expenses users would incur if they adopted autonomous on-demand shared mobility.

7.1. Main Contributions and Results

7.1.1. Main Contributions

In this section, the most important and innovative contributions of this thesis are listed and described.

The introduction of the recharging model

In Chapter 3, the recharging mechanism used within the simulator and how the distribution of charging stations within the city is carried out have also been described. This is also a very important contribution because, if one wants to simulate an electric mobility service, this aspect cannot be overlooked. AVs need to spend a considerable amount of time recharging, a detail that cannot be neglected. During our simulations, we also observed how this can impact the efficiency level of the service, as nighttime disruptions were caused by an excessive number of vehicles at the charging stations. With a model of this type, a provider can understand how to better avoid these issues by exploring alternative solutions. Furthermore, the economic impact of leasing the land and constructing the charging stations represents a significant portion of the total investment required to implement a robotaxi service (approximately 10% according to the results obtained) and cannot be ignored.

Processing of a dataset composed by real trips

In Chapter 4, one of the most important aspects of this work was described, namely that the models developed were entirely built using a dataset of real trips made by UnipolSai customers with their private vehicles within the province of Milano in 2022. This represents a significant advantage over other articles in the literature, which attempt to simulate the operation of a similar service but rely on data from either taxi companies or existing sharing vehicles. The sample we used is much more comprehensive, with a significantly higher number of data points, and accurately represents the movements of the city's residents. Taxis or sharing vehicles trips, currently used for similar analysis, in fact, are not evenly distributed throughout the urban area, being more concentrated in central zones where this type of service is more prevalent, and particularly near stations or points of interest heavily used by tourists. The data we used, on the other hand, come from daily trips made by citizens, the same ones that a fully shared mobility system should be able to replace.

The introduction of public trasportation and walking mobility

Another innovative aspect was the introduction, within the two-layer model, of an alternative mobility method to robotaxis, namely the possibility of reaching destinations on foot and via the currently operating subway lines. Assuming the introduction of restrictions and bans on private car circulation, we must envision a public robotaxi service that aims to adequately serve all these movements. However, this service will certainly operate alongside the existing forms of sustainable mobility, such as public transportation and walking, which will not be eliminated. When implementing this new service, we must not forget this possibility, which represents a viable alternative for customers who wish to use it. In the model we built, this option is included: a user can always choose the quickest mode for reaching their destination, and the service provider must ensure both options are available, bearing the respective costs.

The urban graph construction

In Chapter 4, the construction of the urban graph was described. It consists of two distinct elements:

- The On-demand autonomous mobility graph, where the robotaxis circulate and move, which aims to emulate the layout of the city's streets.
- The Walking and Public Transportation graph, where users can move independently. This graph is composed of the same road network as the AVs graph, but with reduced speeds to simulate walking. Additionally, it includes the existing subway and suburban train stops, along with their connections and travel times.

The introduction of the urban graph is crucial, as it increases the realism of the simulation dynamics. The construction of the graph can later be improved by expanding the number of stations and the public transport network or by refining the road structure, to achieve dynamics that closely mimic real-life scenarios.

Costruction of the urban simulator model

In Chapter 5, the functioning of the simulator employed for the model was described. Taking as input a dataset of travel requests concentrated within a specific week, it dynamically simulates travel from the starting point to the desired destination, serving it either through the fleet of robotaxis in use or via the subway service. The logic employed is straightforward: the customer always chooses the most convenient method by comparing the travel duration across the different options provided to them. Once the simulator

is used, the obtained results are analyzed and studied. If these results do not satisfy the problem's constraints, the simulator inputs are modified, and a new run is executed. This process is repeated until an optimal solution is obtained, using an optimization method called simulator-in-the-loop.

7.1.2. Results of one-layer and two-layer mobility model

In this section, the main results obtained are presented, first with the one-layer model and subsequently with the two-layer model.

One-layer model

In the one-layer case, various scenarios were analyzed. First, we fixed the service area of the robotaxis to a circular area with a radius of 6 km, centered on Piazza del Duomo, and modified the constraints related to the problem, initially keeping them more restrictive and then slightly relaxing them. It was observed how crucial the choice of constraints is, as they must be carefully considered to ensure that the service is both efficent and economical for customers. For example, imposing too tight a time frame for a robotaxi to arrive at the request point, would require an excessively high number of AVs, thereby negating the benefits of reduced vehicle numbers and shared mobility, and increasing the cost of the service for customers. In any case, we have seen how such a service could be significantly more economical compared to the car-sharing services currently operating in the city, with costs potentially dropping to as low as €0.08/min, while still ensuring excellent efficiency standards.

Secondly, still using the one-layer model, we demonstrated how the ratio between the number of private vehicles currently in use and the number of robotaxis needed to replace them varies greatly depending on the size of the area where the service is to be implemented. The larger the area, the more the number of AVs must increase disproportionately compared to the increase in private vehicles to maintain the same efficiency standards. We showed that to serve an area with a 14 km side length, the service must be expected to be significantly less performant, with much longer waiting times, at least in peripheral areas. Otherwise, the number of AVs required would become excessively high, effectively nullifying the advantage of reducing the number of circulating vehicles.

Two-layer model

In the two-layer model, we observed that a robotaxi service operates even more effectively when implemented on a larger scale, envisioning an almost complete ban on private cars

within a 6 km radius of the city center. Excellent performance values were achieved by reducing the number of circulating vehicles by about 50 times, serving the trips of over 80,000 private vehicles with just 1600 robotaxis. Despite this drastic reduction in the number of cars, the service performance remained impeccable: the average waiting time was less than one and a half minutes, and the average travel time was reduced by nearly 30% compared to using private vehicles. This improvement is attributed to both the better traffic management by autonomous vehicles, which ensures significantly fewer traffic jams, and the advantage of eliminating the need to search for parking, a factor that considerably impacts the time required to reach the final destination in a city like Milano. The costs for the service would also be extremely low. Assuming it is a public service and that, in addition to expenses related to robotaxis, it must also cover additional costs due to the increase in users using the subway for travel, an usage fee of €0.35/km was hypothesized. This cost is lower than what most Italians currently spend on their personal cars, thus ensuring significant savings for the majority of the population, especially urban residents who do not typically undertake long trips with their own vehicles.

7.2. Final remarks

The studies scenarios including on-demand autnomous mobility are feasible and the advantages would be enormous in terms of environmental sustainability, since urban mobility would become 100% green, and above all in terms of freeing up areas that are completely useless to the community, reserved only for parking cars. We are often unaware of the fact that our cities are not built on the scale of the citizen, but on the scale of the car; it is difficult even to estimate the amount of public land taken up by car parks. Reducing the number of cars by a factor of 50 would therefore be a huge opportunity to create new meeting places, green spaces or services that would benefit the whole community.

After all, environmental protection and global warming, as well as new European regulations that will come into force in the next few years, are forcing us to move more and more towards electric mobility. However, it is very difficult to imagine that the entire fleet of cars on the road today can be replaced by this type of car. Firstly, despite government incentives, electric cars cost more than many families can afford. Secondly, replacing all the vehicles in the world today would not only be utopian, it would also be unfeasible in terms of raw materials, as there would not be enough elements to build all the batteries. Moreover, shared mobility is the only way to fully exploit the capacity of the batteries themselves, to fully exhaust their life before the car is too worn out to drive. Autonomous driving therefore also represents a great opportunity to facilitate the transition to elec-

tric mobility, as robotaxis will finally make it possible to reach sufficient mileage per car, which is the key enabling feature for a complete transition to full electric cars. So far, these have been "niche" services that do not have a significant impact on the number of cars on the road. The problem lies at the root: if we have too few vehicles, it becomes difficult to use them because it is hard to find an available vehicle nearby; if we increase the number of vehicles, we lose the benefits of shared mobility, which should serve to significantly reduce the number of cars. Autonomous sharing is the only way to overcome this obstacle, as the customer no longer has to worry about getting to the car, but the car goes to the customer.

7.3. Future developments

This thesis and the mobility system optimizer with simulator-in-the-loop developed should be a valuable starting point for future projects, we list below some possible lines of research:

- To introduce bus and tram lines within public transport would make the option of using public transportation more realistic and comprehensive. Currently, the model provides accurate times only for subway transport, while it makes significant errors for routes where surface vehicles are more efficient.
- To differentiate waiting times for public transport between day and night. In the current model, waiting times remain the same regardless of time of day, whereas in reality, there are significant differences. Many lines either do not operate or have greatly extended waiting times during nighttime hours, and the subway is replaced by less efficient night buses. Incorporating these distinctions would lead to more realistic results.
- To introduce a cache for the walking and public transport layer in the simulator, this cache will be a dictionary storing the distances between every pair of nodes. In this way, it will no longer be necessary to compute these distances every time within the simulator, significantly reducing computational costs and enabling more comprehensive studies of larger service areas or scenarios involving a higher volume of data. Currently, not having this cache forces the use of the Dijkstra algorithm at least once for each trip, preventing the time required for a single simulation from falling below 2-3 hours in the case of 559,144 trips to be serviced. By implementing this cache, simulation times could be reduced to just a few minutes, facilitating the development of optimization problems where numerous diverse simulations are necessary using this model. This enhancement would allow for the minimization

of total costs for service implementation, considering both robotaxis and public transport, while satisfying constraints that ensure high standards of efficiency.

- Once the simulator has been sped up, it becomes feasible to simulate an even larger number of daily trips equivalent to those occurring in the city of Milano. The current number of daily trip requests used, although high, is still insufficient to simulate the actual volume of private car movements within the city. By improving simulation speed, it will be easier to execute simulations using datasets containing an even greater number of trips.
- To introduce the possibility of traveling to the final destination using both robotaxis and public services during the trip. Currently, our model does not accommodate this option, whereas in reality, it could sometimes be the fastest choice.
- To improve the recharging mechanism during the night, ensuring that a certain number of vehicles remain available at all times to meet demand at any time of the day. Current observations highlight a limitation in the implemented charging model, where overnight charging often leads to a disproportionately high number of vehicles at charging stations, resulting in too few AVs circulating to efficiently fulfill trip requests during these hours.
- To study a mechanism for rebalancing robotaxis during off-peak hours so that they are more concentrated in areas of highest demand when needed. This could lead to further improvement in the service offered, thereby reducing waiting times for robotaxis.
- To improve the construction of the city graph, making it more and more similar to the real network of roads, with methods like those in Bedogni et al [27].

Bibliography

[1] Urbanization, Our World in Data. https://ourworldindata.org/urbanization.

[2] Statistiche Sperimentali: Previsioni comunali della popolazione. https://demo.istat.it/app.

[3] Area B - Comune di Milano. https://www.comune.milano.it/aree-tematiche/mobilita/area-b.

[4] Una mobilità pulita e sostenibileper un'UE climaticamente neutra - Consilium. https://www.consilium.europa.eu/it/policies/clean-and-sustainable-mobility/#goals, 2024.

[5] James M Anderson, Kalra Nidhi, Karlyn D Stanley, Paul Sorensen, Constantine Samaras, and Oluwatobi A Oluwatola. *Autonomous vehicle technology: A guide for policymakers*. Rand Corporation, 2014.

[6] Giromilano ATM, Azienda Trasporti Milanesi. https://giromilano.atm.it.

[7] Andrea Bressa. La vera storia dell'auto autonoma | Wired Italia. https://www.wired.com/story/robotaxis-cruise-waymo-san-francisco/.

[8] Cruise and W.

[9] Elon Musk announces Tesla will unveil a 'robotaxi' on August 8 | CNN Business. https://edition.cnn.com/2024/04/05/business/elon-musk-tesla-robotaxi-august-8/index.html, author = Valdes-Dapena, Peter, year = 2024.

[10] China's Baidu to dispatch robotaxis in 100 cities by 2030 - Nikkei Asia. https://asia.nikkei.com/Business/China-tech/China-s-Baidu-to-dispatch-robotaxis-in-100-cities-by-2030,, .

[11] La cinese Baidu lancia il robotaxi di sesta generazione. https://www.motorisumotori.it/la-cinese-baidu-lancia-il-robotaxi-sesta-generazione-28-350-
.

[12] PAOLO BUFFO. Veicoli a guida autonoma.

[13] Ricardo Cano. Cruise crash in SF changed robotaxi policy. https://www.sfchronicle.com/projects/2024/cruise-sf-collision-timeline/.

[14] Silvia Bruzzone, Pierluigi Cordellieri, Paolo Perego, Marianna Martini, Mirna Begnini, Stefano Crisci, Massimiliano Nicotra, Nicole Scala, Daniele Bilanzuoli, Marco Pollastri, et al. Aspetti etici, sociali e di sicurezza della guida autonoma.

[15] Road to 2030: the future of autonomous vehicles (AVs). https://www.cubictelecom.com/blog/self-driving-cars-future-of-autonomous-vehicles-automotive-vehicles-2030/, .

[16] Auto a guida autonoma: a che punto è la normativa in Italia e in Europa - Economyup. https://www.economyup.it/automotive/auto-a-guida-autonoma-a-che-punto-e-la-normativa-in-italia-e-in-europa/.

[17] Statistiche auto elettriche in Italia | AutoElettrica101. https://www.autoelettrica101.it/statistiche.php.

[18] Stefano Provera. Mobility as a service: Data-driven operational and economic feasibility of an autonomous urban car-sharing, 2022.

[19] Gioele Zardini, Nicolas Lanzetti, Andrea Censi, Emilio Frazzoli, and Marco Pavone. Co-design to enable user-friendly tools to assess the impact of future mobility solutions. *IEEE Transactions on Network Science and Engineering*, 10(2):827–844, 2022.

[20] Andrea Censi, Jonathan Lorand, and Gioele Zardini. Applied compositional thinking for engineers, 2022.

[21] Azienda Trasporti Milanesi S.p.A. Relazione Annuale Integrata Gruppo ATM, 2023. URL https://www.atm.it/it/IlGruppo/Financial_information/Documents/relazione%20Annuale%20Integrata%20Gruppo%20ATM%202023.pdf.

[22] Auto elettrica, gli ultimi dati per capire come sta andando veramente. https://www.qualenergia.it/articoli/auto-elettrica-ultimi-dati-capire-come-andando-veramente/, .

[23] Colonnine di ricarica auto elettriche | Quattroruote.it. https://www.quattroruote.it/guide/viaggio-e-rifornimento/guida-alle-colonnine-per-la-ricarica-elettrica.html.

[24] Costo installazione colonnina auto elettrica. https://www.instapro.it/elettricita/prezzi-costo/colonnina-ricarica-auto-elettrica.

[25] Canone unico patrimoniale per occupazione di suolo pubblico e attività: regole ed eccezioni. https://www.informazionefiscale.it/canone-unico-patrimoniale-occupazione-suolo-normativa-tariffe-pagamento.

[26] A2A: con le city plug si apre nuova era della ricarica elettrica urbana | la Repubblica. https://www.repubblica.it/dossier/economia/transizione-sostenibile/2024/01/16/news/a2a_con_le_city_plug_si_apre_nuova_era_della_ricarica_elettrica_urbana-421886498/.

[27] Luca Bedogni and Andrea Ercolessi. Analisi ed identificazione di percorsi stradali tramite sensori inerziali. 2019.

[28] Analisi dei dati mobilità Area C. https://datamobility.it/magazine/comunicare-con-dati-giochiamo-coi-numeri-area-c/.

[29] Costi manutenzione auto annuali in Italia. https://www.automobile.it/magazine/manutenzione-auto/costi-manutenzione-auto-9014.

[30] Bilancio Consolidato ATM. https://risultati2021.atm.it/files/12_BILANCIO_minisito_def.pdf.

[31] Kenneth Lebeau, Philippe Lebeau, Cathy Macharis, and Joeri Van Mierlo. How expensive are electric vehicles? a total cost of ownership analysis. In *2013 World Electric Vehicle Symposium and Exhibition (EVS27)*, pages 1–12. IEEE, 2013.

[32] Gli italiani e l'auto, i numeri. https://finanza.lastampa.it/News/2022/09/30/gli-italiani-e-lauto-da-parco-circolante-a-km-percorsi-i-numeri/MTIxXzIwMjItMDktMzBfVExC, 2022.

A | Appendix

In this appendix, the complete code described Chapter 5 is reported.

```
import pandas as pd
import networkx as nx
import numpy as np
from tqdm import tqdm


class Simulator:
    def __init__(self, path, columns_path, avs_speed, car_number, recharging_station_number, graph1, graph2, cache, simulation_type):
        # In columns path ho un dataset con nodi colonnine e 'points' corrispondenti
        # Set speed for an autonomous vehicle
        self.L3VehicleSpeed = avs_speed # km/h
        self.PedestrianSpeed = 4.8 # km/h

        # Simulation data import
        self.path = path
        self.df = pd.read_csv(self.path)
        self.df['time_start'] = pd.to_datetime(self.df['time_start'])
        self.df = self.df.sort_values(by='time_start')
        self.df = self.df.reset_index()
        self.columns_path = columns_path
        self.columns_df = pd.read_csv(self.columns_path)
        self.graph1 = graph1
        self.graph2 = graph2
        self.cached_avs = cache
        self.cached_public_walking = {}
        self.simulation_type = simulation_type

        self.every_station_occupied = False

        self.final_df = pd.DataFrame(columns=['index', 'node',
                        'busy_until', 'travel_time', 'overhead',
                        'dead_time', 'tot_calls', 'time_waiting_free_station',
                        'time_recharging', 'tot_charges'])

        self.car_number = car_number
        self.recharging_station_number = recharging_station_number
        self.recharging_vehicles = 0

        # Dataset of the fictitious vehicles (updated overtime)
        self.free_cars = {}
        self.busy_cars = {}
```

```
    # Max time the user is willing to wait for the car to arrive.
    self.USER_WAIT_FOR_CAR = 5 # [min]
    # Max time to reach the nearest free station
    self.MAX_TIME_TO_STATION = 20 # [min]

    # Parameters Car
    self.battery_capacity = 40 # in kw
    self.efficiency_battery = 0.99
    self.consumption = 0.1375 # in kwh/km
    self.recharging_power = 50 # in kw

    self.threshold_send_recharge_day = 15
    self.threshold_car_free_day = 60
    self.threshold_send_recharge_night = 40
    self.threshold_car_free_night = 90
    self.threshold_send_recharge = self.threshold_send_recharge_day
    self.threshold_car_free = self.threshold_car_free_day
    self.night = False

    # List of all the user wait time
    self.user_wait_time = []
    # List of the number of free vehicles at each iteration
    self.free_cars_v = []
    # List of the number of vehicles in recharge at each iteration
    self.recharging_vehicles_list = []
    # List of all the skipped calls (if simulation_type == one)
    self.skipped_calls = []
    # List of all the time avs fail the use (if simulation_type == two)
    self.no_avs_use = []
    # List of all public/walking use (if simulation_type == two)
    self.public_walking_use = []
    # List of all previous time
    self.previous_time = self.df['trip_duration_BT']
    # List of all current time for all trips
    self.current_time = []
    # List of all time with public transport (if simulation_type == two)
    self.public_walking_time = []
    # List of all time with AVs (no waiting considered)
    self.avs_time = []
    # List of all time start
    self.time_start = self.df['time_start']
```

```
def InitializeRechargingStations(self):
    columns_number = self.recharging_station_number
    columns_df = self.columns_df
    max_starting_points = columns_df['points'].max()

    self.charging_stations = pd.DataFrame(columns=['node', 'available_stations', 'occupied_stations'])
    self.charging_stations['node'] = columns_df['name']
    self.charging_stations['available_stations'] = columns_df['points'] / max_starting_points
    self.charging_stations['occupied_stations'] = 0
    # scaling facto is useful to make the total number of columns installed equal to columns_tot
    # operation /5 and *5 round the number to the nearest multiple of 5
    current_sum = self.charging_stations['available_stations'].sum()
    scaling_factor = (columns_number / 5) / current_sum

    condition_1 = self.charging_stations['available_stations'] * scaling_factor < 0.05
    condition_2 = (self.charging_stations['available_stations'] * scaling_factor >= 0.05)
    & (self.charging_stations['available_stations'] * scaling_factor < 0.25)
    condition_3 = (self.charging_stations['available_stations'] * scaling_factor >= 0.25) \
                  & (self.charging_stations['available_stations'] * scaling_factor <= 0.5)

    self.charging_stations.loc[condition_1, 'available_stations'] = 0
    self.charging_stations.loc[condition_2, 'available_stations'] = 1
    self.charging_stations.loc[condition_3, 'available_stations'] = 2

    self.charging_stations.loc[~(condition_1 | condition_2 | condition_3), 'available_stations'] = \
                np.round(self.charging_stations['available_stations'] * scaling_factor).astype(int) * 5
```

```
def FreeChargingStation(self, used_car, **kwargs):
    time_req = kwargs.get('time_req', None)
    node=used_car[0]
    selected_station = self.charging_stations['node']
    self.charging_stations.loc[self.charging_stations['node'] == selected_station, 'available_stations'] += 1
    self.charging_stations.loc[self.charging_stations['node'] == selected_station, 'occupied_stations'] -= 1
    self.recharging_vehicles -= 1
    self.every_station_occupied = False
    if time_req != None:
        # Now update the charge until the last moment
        time_delta = (time_req - used_car[11]).total_seconds()
        used_car[6] = self.newSOC(used_car[6], True, delta_time=time_delta)
        # Update tot time recharging
        used_car[9] = used_car[9] + time_delta
        used_car[11] = time_req
        used_car[12].append(used_car[6])
        used_car[13].append(used_car[11])
    # Car no longer recharging
    used_car[7] = False

    return used_car


def SendToChargingStation(self, chosen_vehicle, actual_time):
    # Find the closest charging station available
    nodes = self.charging_stations.loc[self.charging_stations['available_stations'] > 0, 'node'].tolist()

    if len(nodes) == 0:
        self.every_station_occupied = True
        return [], False

    vehicle_pos = chosen_vehicle[0]
    time_distance = float('inf')
    selected_station = None

    for id_station in nodes:
        if (str(vehicle_pos), str(id_station)) in self.cached_avs:
            shortest_path = self.cached_avs[(str(vehicle_pos), str(id_station))]
        elif (str(id_station), str(vehicle_pos)) in self.cached_avs:
            shortest_path = self.cached_avs[(str(id_station), str(vehicle_pos))]
        else:
            # Compute node distance
            shortest_path = nx.shortest_path_length(self.graph1, source=str(vehicle_pos), target=str(id_station), weight='weight')
```

```
        # if the distance is less than the minimum distance computed
        if shortest_path < time_distance:
            time_distance = shortest_path  # Update the minimum distance
            selected_station = id_station  # Update the station ID

    if time_distance / 60 < self.MAX_TIME_TO_STATION:

        self.charging_stations.loc[self.charging_stations['node'] == selected_station, 'available_stations'] -= 1
        self.charging_stations.loc[self.charging_stations['node'] == selected_station, 'occupied_stations'] += 1
        self.recharging_vehicles += 1

        # Now update all datas of the car
        newSoc = self.newSOC(chosen_vehicle[6], charging=False, tot_distance = time_distance)
        newtime = actual_time + pd.to_timedelta((time_distance), unit='s')
        chosen_vehicle[12].append(newSoc)
        chosen_vehicle[13].append(newtime)

        updated_vehicle = [selected_station, chosen_vehicle[1],
                           chosen_vehicle[2] + time_distance, chosen_vehicle[3] + time_distance,
                           chosen_vehicle[4], chosen_vehicle[5], newSoc, True, chosen_vehicle[8], chosen_vehicle[9],
                           chosen_vehicle[10] + 1, newtime,
                           chosen_vehicle[12], chosen_vehicle[13]]

        return updated_vehicle, True

    else:
        return [], False
```

```
def InitializeCars(self, N_sharcars):
    self.free_cars = {}

    # 0,        1,         2,          3,        4,        5,         6,               7,
    # Node, busy_until, travel_time, overhead, dead_time, tot_calls, state_of_charge, charging_status,
    # 8,                          9,              10,          11,                12,            13.
    # time_waiting_free_station, time_recharging, tot_charges, last_check_recharge, soc_evolution, time_soc_change

    for i in range(N_sharcars):
        self.free_cars.update({i: [self.df.iloc[i]['cluster_start'],
                                   self.df.iloc[i]['time_start'], 0, 0, 0, 0, 100, False,
                                   0, 0, 0, self.df.iloc[i]['time_start'], [100], [self.df.iloc[i]['time_start']]]})

    self.busy_cars = {}


def FreeCarAvailable(self, row, request_time, request_pos):
    time_distance_to_user = float('inf')  # Set initial minimum distance
    chosen_vehicle = None  # Initialize car ID

    # Compute distance for each free car
    for car_id, car_data in self.free_cars.items():
        node = car_data[0]  # Actual car position
        # Check if the shortest_path is in the cache
        if (str(request_pos), str(node)) in self.cached_avs:
            shortest_path_avs = self.cached_avs[(str(request_pos), str(node))]
        elif (str(node), str(request_pos)) in self.cached_avs:
            shortest_path_avs = self.cached_avs[(str(node), str(request_pos))]
        else:
            # Compute node distance
            shortest_path_avs = nx.shortest_path_length(self.graph1, source=str(request_pos), target=str(node), weight='weight')

        # if distance is less then the minimum distance previously computed
        if shortest_path_avs < time_distance_to_user:
            time_distance_to_user = shortest_path_avs  # Update minimum distance
            chosen_vehicle = car_id  # Update car ID

    # This car has to go to the user
    dt_to_user = time_distance_to_user
    dt_avs = 10000

    start_point = request_pos
    end_point = row['cluster_end']
```

```
    # Now the user has the car and can use it
    if (str(request_pos), str(end_point)) in self.cached_avs:
        dt_avs = self.cached_avs[(str(request_pos), str(end_point))]
    elif (str(end_point), str(request_pos)) in self.cached_avs:
        dt_avs = self.cached_avs[(str(end_point), str(request_pos))]
    else:
        dt_avs = nx.shortest_path_length(self.graph1, source=str(start_point), target=str(end_point), weight='weight')

    self.avs_time.append(dt_avs)

    # if one-layer model is selected
    if self.simulation_type == 1:

        if dt_to_user / 60 > self.USER_WAIT_FOR_CAR: # If the user has to wait too much, the call is dropped
            self.skipped_calls.append(1)
            self.user_wait_time.append(None)
            self.current_time.append(None)
        else:
            self.user_wait_time.append(dt_to_user)
            self.skipped_calls.append(0)

            self.current_time.append(dt_avs + dt_to_user)
            # Now update the car dataset
            used_car = self.free_cars[chosen_vehicle]

            if used_car[7] == True: # Charging = True --> I free the charging stations
                used_car = self.FreeChargingStation(used_car, time_req=request_time)
                dead_time = 0
```

```
            else:
                # This car was waiting for a call, save the dead time
                dead_time = (request_time - used_car[1]).total_seconds()

            self.free_cars.pop(chosen_vehicle)
            new_soc = self.newSOC(used_car[6], charging = False, tot_distance = dt_avs + dt_to_user)
            self.busy_cars.update({chosen_vehicle: [row['cluster_end'],
                                                    request_time + pd.to_timedelta((dt_to_user + dt_avs), unit = 's'),
                                                    used_car[2] + dt_avs + dt_to_user,
                                                    used_car[3] + dt_to_user,
                                                    used_car[4] + dead_time,
                                                    used_car[5] + 1,
                                                    new_soc,
                                                    False,
                                                    used_car[8], used_car[9], used_car[10], used_car[11],
                                                    used_car[12], used_car[13]]})

    if self.simulation_type == 2:

        # Compute distance in walking and public transport graph
        if (start_point, end_point) in self.cached_public_walking:
            dt_public_walking = self.cached_public_walking[(start_point, end_point)]
        else:
            dt_public_walking = nx.shortest_path_length(self.graph2, source=str(start_point), target=str(end_point), weight='weight')
            self.cached_public_walking[(start_point, end_point)] = dt_public_walking

        self.public_walking_time.append(dt_public_walking)

        if dt_public_walking < dt_avs:
            self.public_walking_use.append(1)
            self.no_avs_use.append(None)
            self.user_wait_time.append(None)
            self.current_time.append(dt_public_walking)

        elif dt_public_walking < dt_avs + dt_to_user:
            self.public_walking_use.append(1)
            self.no_avs_use.append(1)
            self.user_wait_time.append(None)
            self.current_time.append(dt_public_walking)
```

```
         else:
            self.public_walking_use.append(0)
            self.user_wait_time.append(dt_to_user)
            self.no_avs_use.append(0)
            self.current_time.append(dt_avs + dt_to_user)
            # Now update the car dataset
            used_car = self.free_cars[chosen_vehicle]

            if used_car[7] == True: # Charging = True --> I free the charging stations
                used_car = self.FreeChargingStation(used_car, time_req=request_time)
                dead_time = 0

            else:
                # This car was waiting for a call, save the dead time
                dead_time = (request_time - used_car[1]).total_seconds()

            self.free_cars.pop(chosen_vehicle)
            new_soc = self.newSOC(used_car[6], charging = False, tot_distance = dt_avs + dt_to_user)
            self.busy_cars.update({chosen_vehicle: [row['cluster_end'],
                                                    request_time + pd.to_timedelta((dt_to_user + dt_avs), unit = 's'),
                                                    used_car[2] + dt_avs + dt_to_user,
                                                    used_car[3] + dt_to_user,
                                                    used_car[4] + dead_time,
                                                    used_car[5] + 1,
                                                    new_soc,
                                                    False,
                                                    used_car[8], used_car[9], used_car[10], used_car[11],
                                                    used_car[12], used_car[13]]})
```

```
def CheckFreeVehicles(self, request_time):
    all_vehicles = {**self.free_cars, **self.busy_cars}
    self.free_cars = {}
    self.busy_cars = {}
    for vehicle in all_vehicles.items():
        param_car = vehicle[1]
        # Update the SOC of charging vehicles
        if param_car[7] == True and param_car[11] < request_time:
            # If the vehicle is in charge, and it has arrived to the charging station
            time_delta = (request_time - param_car[11]).total_seconds()
            if time_delta > 900: # Update battery level first time after 30 minutes, to decrease computational effort
                param_car[11] = request_time # Update the new last_check_recharge
                param_car[6] = self.newSOC(param_car[6], True, self.L3VehicleSpeed, delta_time = time_delta)
                param_car[9] = param_car[9] + time_delta # Time spent recharging, in seconds
                param_car[12].append(param_car[6])
                param_car[13].append(param_car[11])
                if param_car[6] == 100: # The car is completely charged, leave the charging station
                    param_car = self.FreeChargingStation(param_car)

        if param_car[1] > request_time:
            # Vehicle is being used, not free
            self.busy_cars.update({vehicle[0]: param_car})
        else:
            if param_car[7] == True:
                # Vehicle is recharging
                if param_car[6] > self.threshold_car_free:
                    # Soc greater than 60%/90%, now the car becomes free
                    self.free_cars.update({vehicle[0]: param_car})
                else:
                    self.busy_cars.update({vehicle[0]: param_car})
            else:
                # Vehicle is not recharging
                if param_car[6] > self.threshold_send_recharge:
                    # SOC greater than lower threshold, vehicle is free
                    self.free_cars.update({vehicle[0]: param_car})
                else:
                    if self.every_station_occupied == True:
                        # Car is busy, leave like this
                        self.busy_cars.update({vehicle[0]: param_car})
```

```
                    else:
                        # I can send the car to recharge
                        updated_vehicle, available = self.SendToChargingStation(param_car, request_time)
                        if available == True:
                            # There's at least one station free not too far
                            # This is needed to deal with the limit case of the car going to occupy the last available station
                            # Update time waiting to be sent to charging station
                            updated_vehicle[8] = updated_vehicle[8] + (request_time - param_car[1]).total_seconds()
                            self.busy_cars.update({vehicle[0]: updated_vehicle})
                        else:
                            # The previous car occupied the last station or free stations too far, no more free
                            self.busy_cars.update({vehicle[0]: param_car})
```

```
def newSOC(self, old_soc, charging, *args, **kwargs):
    # This function gives as output the new State of Charge of the car in percentage
    delta_time_in_sec = kwargs.get('delta_time', None)  # in seconds
    distance_in_sec = kwargs.get('tot_distance', None)  # in km
    if distance_in_sec is not None:
        distance_in_km = distance_in_sec * (self.L3VehicleSpeed / 3.6) / 1000
    else:
        distance_in_km = 0
    new_soc = 0
    if charging == True:
        charge_battery = self.efficiency_battery * self.recharging_power * delta_time_in_sec / 3600
        pct_charge = charge_battery / self.battery_capacity
        new_soc = old_soc + pct_charge * 100
        if new_soc > 100:
            new_soc = 100
    else:  # I'm using the car -> discharging
        pct_used_power = self.consumption * distance_in_km / self.battery_capacity
        new_soc = old_soc - pct_used_power * 100
        if new_soc < 0:
            new_soc = 0
            print('Car battery dead, call the assistance')

    return new_soc


def createData(self, N):
    v0, v1, v2, v3, v4, v5, v6, v7, v8, v9 = [], [], [], [], [], [], [], [], [], []
    self.evolution_soc = []
    self.time_soc = []
    for item in {**self.busy_cars, **self.free_cars}.items():
        v0.append(item[0])
        v1.append(item[1][0])
        v2.append(item[1][1])
        v3.append(item[1][2])
        v4.append(item[1][3])
        v5.append(item[1][4])
        v6.append(item[1][5])
        v7.append(item[1][8])
        v8.append(item[1][9])
        v9.append(item[1][10])
        self.final_df.loc[len(self.final_df)] = [item[0],item[1][0],item[1][1],item[1][2],item[1][3],\
                                                 item[1][4],item[1][5],item[1][8],item[1][9],item[1][10]]
        self.evolution_soc.append(item[1][12])
        self.time_soc.append(item[1][13])
```

```python
def Simulation(self, N_car):
    # Start simulation
    self.InitializeRechargingStations()
    self.InitializeCars(N_car)

    users_wait_time, free_cars, skipped_calls = [], [], []

    info_col = ['time_start', 'cluster_start', 'cluster_end']

    for index, row in tqdm(self.df[info_col].iterrows(), total = len(self.df)):

        # When and where the call has been received
        request_time = row['time_start']
        request_pos = row['cluster_start']

        #### night change in recharge policy
        if self.night == False and request_time.hour >= 0 and request_time.hour < 3: #now midnight has arrived
            self.threshold_car_free = self.threshold_car_free_night
            self.threshold_send_recharge = self.threshold_send_recharge_night
            self.night = True

        if self.night == True and request_time.hour >= 3: # morning glory
            self.threshold_car_free = self.threshold_car_free_day
            self.threshold_send_recharge = self.threshold_send_recharge_day
            self.night = False
```

```python
        # Find the free vehicle at this moment in time
        self.CheckFreeVehicles(request_time)
        self.free_cars_v.append(len(self.free_cars))
        self.recharging_vehicles_list.append(self.recharging_vehicles)

        # Look for the cars and see if one is free
        if len(self.free_cars) > 0:
            # Free vehicles are available, find the closest to the user
            self.FreeCarAvailable(row, request_time, request_pos)
        else:
            if self.simulation_type == 1:
                # If there are not AVs --> missing call
                self.skipped_calls.append(1)
                self.user_wait_time.append(None)
                self.current_time.append(None)

            if self.simulation_type == 2:
                # If there are not AVs --> use public transport
                start_point = request_pos
                end_point = row['cluster_end']
                if (start_point, end_point) in self.cached_public_walking:
                    dt_public_walking = self.cached_public_walking[(start_point, end_point)]
                else:
                    dt_public_walking = nx.shortest_path_length(self.graph2, source=str(start_point), target=str(end_point), weight='weight')
                    self.cached_public_walking[(start_point, end_point)] = dt_public_walking

                # Controllo se la grandezza di cached supera il limite
                if len(self.cached_public_walking) > self.MAX_CACHED_PUBLIC_WALKING_SIZE:
                    # Rimuovo il valore più vecchio
                    oldest_key = min(self.cached_public_walking, key=self.cached_public_walking.get)
                    del self.cached_public_walking[oldest_key]

                self.public_walking_time.append(dt_public_walking)
                self.avs_time.append(None)
                self.public_walking_use.append(1)
                self.no_avs_use.append(1)
                self.user_wait_time.append(None)
                self.current_time.append(dt_public_walking)

    self.createData(N_car)
```

```
        if self.simulation_type == 1:
            return self.final_df, self.user_wait_time, self.free_cars_v, self.recharging_vehicles_list, self.skipped_calls, \
            self.current_time, self.previous_time, self.time_start, self.evolution_soc, self.time_soc

        if self.simulation_type == 2:
            return self.final_df, self.user_wait_time, self.free_cars_v, self.recharging_vehicles_list, self.no_avs_use, \
            self.public_walking_use, self.current_time, self.public_walking_time, self.avs_time, self.previous_time, \
            self.time_start, self.evolution_soc, self.time_soc


    def SetupEarlyStop(self, EARLY):
        self.df = self.df[:EARLY]


# Funzioni utili

def evaluation_opt_function1(df, user_wait, skipped_calls, previous_time, max_frac_skipped_calls, max_perc_overhead, max_avg_wait):

    frac_skipped_calls = sum(skipped_calls)/len(skipped_calls)
    perc_overhead = df['overhead'].sum()/df['travel_time'].sum()
    total_travel_time = df['travel_time'].sum()
    filtered_wait = [value for value in user_wait if value is not None]
    avg_wait = sum(filtered_wait[3000:]) / (len(filtered_wait)-3000) /60 # Wait in minutes
    # First 3000 values does not count to avoid biased results caused by cars inizialization
    print('Skipped calls: ', frac_skipped_calls)
    print('Average wait: ', avg_wait)
    print('Overhead %: ', perc_overhead)

    if frac_skipped_calls > max_frac_skipped_calls or perc_overhead > max_perc_overhead or avg_wait > max_avg_wait:
        return -1 # constraints violated

    else:
        return 1

def evaluation_opt_function2(df, user_wait, no_avs_use, public_walking_use, current_time, previous_time):

    current_time = np.array(current_time)
    previous_time = np.array(previous_time)

    # First 3000 values does not count to avoid biased results caused by cars inizialization
    filtered_current = current_time[3000:]
    filtered_previous = previous_time[3000:]
    mean_current = filtered_current.mean()
```

```
    mean_previous = filtered_previous.mean()
    frac_no_avs_use = no_avs_use.count(1)/(no_avs_use.count(0)+no_avs_use.count(1))
    frac_public_walking_use = sum(public_walking_use)/len(public_walking_use)
    perc_overhead = df['overhead'].sum()/df['travel_time'].sum()
    total_travel_time = df['travel_time'].sum()
    filtered_wait = [value for value in user_wait if value is not None]
    avg_wait = sum(filtered_wait[3000:]) / (len(filtered_wait)-3000) / 60 # wait in minutes
    print(f'No used AVs percentage: {frac_no_avs_use}')
    print(f'Used public transport (or walking) percentage: {frac_public_walking_use}')
    print(f'Overhead percentage: {perc_overhead}')
    print(f'Average waiting time: {avg_wait}')
    print(f'Actual average travel time: {mean_current}')
    print(f'Previous average travel time: {mean_previous}')

    return 1


def run_simulation(car_number, columns_number, avs_speed, simulation_type):
    sim = Simulator(path, columns_path, avs_speed, car_number, columns_number, G1, G2, cache_avs, simulation_type)
    sim.SetupEarlyStop(1000000)
    return sim.Simulation(car_number)
```

```
# Set parameters
avs_speed = float(input('Insert AVs speed in km/h: '))
sim_type = None
while sim_type not in [1, 2]:
    sim_type = int(input('Insert 1 for one-layer simulation, 2 for two-layer simulation: '))
if sim_type == 1:
    max_frac_skipped_calls = float(input('Insert the max percentage of skipped calls accepted (for 5% insert 0.05): '))
    max_avg_wait = float(input('Insert the max avg waiting time for the optimization problem (in minutes): '))
    max_perc_overhead = float(input('Insert the max overhead percentage (for 50% insert 0.50): '))
    range1 = int(input('Insert min range value for binary search: '))
    range2 = int(input('Insert max range value for binary search: '))
    range_avs = [range1, range2]
if sim_type == 2:
    car_number = int(input('Insert number of AVs for the simulation: '))
columns_number = int(input('Insert columns number for the simulation: '))

# Upload graphs data
df_vertices = pd.read_csv('...Insert vertices path in csv format')
print(df_vertices.head())
df_vertices_layers = pd.read_excel(r'...Insert vertices related to underground stations path in csv format')
print(df_vertices_layers.head())
df_edges = pd.read_csv(r'...Insert edges path in csv format')
print(df_edges.head())
df_edges_toadd = pd.read_csv(r'...Insert edges to connect the cgraph components path in csv format')
print(df_edges_toadd.head())
df_edges_layers = pd.read_csv(r'...Insert edges between underground stations path in csv format')
print(df_edges_layers.head())
df_edges_layers_connected = pd.read_csv(r'...Insert edges that connect underground stations with streets path in csv format')
print(df_edges_layers_connected.head())

# Create graphs
G1 = nx.Graph()
G2 = nx.DiGraph()

# Adding nodes
G1.add_nodes_from(df_vertices['name'].unique())
G2.add_nodes_from(df_vertices['name'].unique())
G2.add_nodes_from(df_vertices_layers['name'].unique())
```

```
# Adding edges
for i in range(len(df_edges)):
    node1 = str(df_edges.iloc[i]['node1'])
    node2 = str(df_edges.iloc[i]['node2'])
    G1.add_edge(node1, node2, weight=(df_edges.iloc[i]['avs'] * avs_speed) / 16)
    G2.add_edge(node1, node2, weight=df_edges.iloc[i]['public_walking'])
    G2.add_edge(node2, node1, weight=df_edges.iloc[i]['public_walking'])

for i in range(len(df_edges_toadd)):
    node1 = str(df_edges_toadd.iloc[i]['node1'])
    node2 = str(df_edges_toadd.iloc[i]['node2'])
    G1.add_edge(node1, node2, weight=(df_edges_toadd.iloc[i]['avs'] * avs_speed) / 16)
    G2.add_edge(node1, node2, weight=df_edges_toadd.iloc[i]['public_walking'])
    G2.add_edge(node2, node1, weight=df_edges_toadd.iloc[i]['public_walking'])

for i in range(len(df_edges_layers)):
    node1 = str(df_edges_layers.iloc[i]['node1'])
    node2 = str(df_edges_layers.iloc[i]['node2'])
    G2.add_edge(node1, node2, weight=df_edges_layers.iloc[i]['public_walking'])
    G2.add_edge(node2, node1, weight=df_edges_layers.iloc[i]['public_walking'])

for i in range(len(df_edges_layers_connected)):
    node1 = str(df_edges_layers_connected.iloc[i]['node1'])
    node2 = str(df_edges_layers_connected.iloc[i]['node2'])
    G2.add_edge(node1, node2, weight=df_edges_layers_connected.iloc[i]['public_walking'])
```

```
print(G1)
print(nx.is_connected(G1))
components = list(nx.connected_components(G1))
print(f"We have {nx.number_connected_components(G1)} unconnected components")

print(G2)

# Upload column dataset
columns_path = r'...Insert columns path in csv format'

# Upload trips dataset
path = r'Insert trips path in csv format'

# Upload cache
df_cache = pd.read_csv(r'...Insert cache path in csv format')
# Trasform df_cache into a dictionary
cache_avs = df_cache.set_index(['node1', 'node2'])['time_seconds'].to_dict()

df_merged = pd.read_csv(path)
print(len(df_merged))

if __name__ == '__main__':
    if sim_type == 1:
        min_car_number = range_avs[0]
        max_car_number = range_avs[1]

        # Binary search algorithm
        while min_car_number <= max_car_number:
            mid = (min_car_number + max_car_number) // 2
            print(mid)

            result = run_simulation(mid, columns_number, avs_speed, sim_type)
            df = result[0]
            user_wait = result[1]
            free_cars = result[2]
            recharging_vehicles = result[3]
            skipped_calls = result[4]
            current_time = result[5]
            previous_time = result[6]
            time_start = result[7]
            evolution_soc = result[8]
```

```
            time_soc = result[9]
            val_function = evaluation_opt_function1(df, user_wait, skipped_calls, previous_time, max_frac_skipped_calls, \
                           max_perc_overhead, max_avg_wait)

            if val_function == -1: # conditions not satisfied --> search into bigger car numbers a solution
                min_car_number = mid + 1
            if val_function == 1: # conditions not satisfied --> search into lower car numbers a better solution
                car_number = mid
                max_car_number = mid - 1

        result = run_simulation(car_number, columns_number, avs_speed, sim_type)
        df = result[0]
        user_wait = result[1]
        free_cars = result[2]
        recharging_vehicles = result[3]
        skipped_calls = result[4]
        current_time = result[5]
        previous_time = result[6]
        time_start = result[7]
        evolution_soc = result[8]
        time_soc = result[9]
        val_function = evaluation_opt_function1(df, user_wait, skipped_calls, previous_time, max_frac_skipped_calls, \
                       max_perc_overhead, max_avg_wait)
```

```
        if val_function == -1:
            print('NO VALUE IN THE RANGE THAT SATISFIES THE CONSTRAINTS')
        if val_function == 1:
            print('OPTIMAL VALUE OBTAINED: ', car_number)

            # Create a new DataFrame
            new_df = pd.DataFrame({
                'user_wait': user_wait,
                'free_cars': free_cars,
                'recharging_vehicles': recharging_vehicles,
                'skipped_calls': skipped_calls,
                'current_time': current_time,
                'previous_time': previous_time,
                'time_start': time_start
                })

            soc_df = pd.DataFrame({
                'evolution_soc': evolution_soc,
                'time_soc': time_soc
                })

            df.to_csv(f'final_df_one_layer_{car_number}AV.csv')
            new_df.to_csv(f'results_df_one_layer_{car_number}AV.csv')
            soc_df.to_csv(f'soc_df_one_layer_{car_number}AV.csv')


    if sim_type == 2:
        result = run_simulation(car_number, columns_number, avs_speed, sim_type)
        df = result[0]
        user_wait = result[1]
        free_cars = result[2]
        recharging_vehicles = result[3]
        no_avs_use = result[4]
        public_walking_use = result[5]
        current_time = result[6]
        public_walking_time = result[7]
        avs_time = result[8]
        previous_time = result[9]
        time_start = result[10]
        evolution_soc = result[11]
        time_soc = result[12]
```

```
        val_function = evaluation_opt_function2(df, user_wait, no_avs_use, public_walking_use, current_time, previous_time)

        # Create a new DataFrame
        new_df = pd.DataFrame({
            'user_wait': user_wait,
            'free_cars': free_cars,
            'recharging_vehicles': recharging_vehicles,
            'no_avs_use': no_avs_use,
            'public_walking_use': public_walking_use,
            'current_time': current_time,
            'public_walking_time': public_walking_time,
            'avs_time': avs_time,
            'previous_time': previous_time,
            'time_start': time_start
            })

        soc_df = pd.DataFrame({
            'evolution_soc': evolution_soc,
            'time_soc': time_soc
            })

        df.to_csv(f'final_df_two_layer_{car_number}AV.csv')
        new_df.to_csv(f'results_df_two_layer_{car_number}AV.csv')
        soc_df.to_csv(f'soc_df_two_layer_{car_number}AV.csv')
```

Acknowledgements

A heartfelt thank you goes to my supervisor, Prof. Silvia Carla Strada, for her availability and expert guidance throughout these months of work.

I am deeply grateful to Ing. Antonio Pagliaroli, who always provided invaluable advice during challenging times, and to Prof. Sergio M. Savaresi.

I also thank the entire DEIB-mOve department for providing me with all the necessary materials and spaces to conduct this study.