Integration of Travel Context Information into Multimodal Platforms

PEDRO MIGUEL NOBRE DIOGO
(Licenciado)

Dissertação para obtenção do grau de mestre em Engenharia Informática e de Computadores

Orientadores:
Doutor José Manuel de Campos Lages Garcia Simão
Especialista Nuno António Fraga Juliano Cota

Júri:
Presidente: Doutor Nuno Miguel Machado Cruz
Vogais:
Doutor Nuno Miguel Soares Datia
Doutor José Manuel de Campos Lages Garcia Simão

Dezembro de 2024

Integration of Travel Context Information into Multimodal Platforms

PEDRO MIGUEL NOBRE DIOGO
(Licenciado)

Dissertação para obtenção do grau de mestre em Engenharia Informática e de Computadores

Orientadores:
Doutor José Manuel de Campos Lages Garcia Simão, ISEL
Especialista Nuno António Fraga Juliano Cota, ISEL

Júri:
Presidente: Doutor Nuno Miguel Machado Cruz, ISEL
Vogais:
Doutor Nuno Miguel Soares Datia, ISEL
Doutor José Manuel de Campos Lages Garcia Simão, ISEL

Dezembro de 2024

Acknowledgements

I would like to begin by expressing my deepest gratitude to my Advisors, Professor José Simão and Professor Nuno Cota. Their guidance and support throughout this work have been invaluable. Without their mentorship, this project would not have reached its full potential, and for that, I am truly thankful.

I also want to extend my heartfelt appreciation to the entire team at Solvit. Their generosity in providing the necessary resources, as well as their continuous support and collaboration, played a crucial role in the successful completion of this project. Working alongside such a dedicated team has been a privilege. Additionally, I would like to express my gratitude to the ATE - WP4.02 Plataforma Multimodal de Transportes project. If it wasn't for the funding of this project, this work would not have been possible.

Finally, I would like to express my profound thanks to my parents, my brother, my girlfriend and friends. Their unwavering and unconditional support, patience, and understanding have been a constant source of motivation for me. Their belief in me has made this journey possible, and I am forever grateful for their love and encouragement during every step of this process.

To all of you, my sincere thanks!

Statement of integrity

I declare that this dissertation is the result of my personal and independent research. Its content is original, and all sources listed in the bibliographic references were consulted and are duly mentioned in the text. I further declare that all scientific and technical references relevant to the development of the work are duly cited and included in the bibliographic references.

The author

Assinado por: **Pedro Miguel Nobre Diogo**
Num. de Identificação: 30637343
Data: 2024.12.19 15:00:48+00'00'

Lisbon, 19th of December, 2024

Integration of Travel Context Information into Multimodal Platforms

Resumo

Os transportes públicos oferecem soluções de mobilidade acessíveis, económicas e ambientalmente sustentáveis para milhões de pessoas em todo o mundo. Contribuem para a redução do congestionamento de trânsito e das emissões de carbono, garantindo o acesso equitativo a serviços essenciais. Os serviços de transporte público fiáveis e eficientes são necessários para alcançar estes objetivos. Para tal, os operadores de transportes públicos precisam de tomar decisões informadas relativamente aos seus serviços. Dados como o fluxo de passageiros podem ajudar a identificar horários de pico, rotas subutilizadas e áreas de crescente procura, permitindo ajustes ao serviço para melhor atender às necessidades dos passageiros. Embora os sistemas de bilhética inteligente sejam úteis para compreender os padrões de passageiros, não fornecem informações detalhadas sobre o número de passageiros que saem de um veículo. Esta informação pode ser útil na gestão de transferências de passageiros entre dois modos de transporte ou rotas, o que é crucial para manter a conectividade fluida e minimizar os tempos de viagem. Para resolver esta questão, propomos um sistema que recolhe dados de entrada e saída dos passageiros de autocarros através de sensores de contagem automática. Esses dados são posteriormente apresentados de forma visual num painel de controlo, para ajudar os operadores de transporte público a melhorar os seus serviços. Os resultados mostraram-se promissores, com o sistema a funcionar conforme esperado durante os testes num autocarro operacional. Os sensores selecionados alcançaram uma precisão de 93,75% para as contagens de saídas e 96,3% para as contagens de entradas. Embora seja necessário um ajuste adicional para maximizar a precisão, o sistema continua a ser uma ferramenta valiosa para melhorar a eficiência do transporte público e responder às necessidades dos passageiros.

Palavras chave

Transporte público; Plataformas multimodais; Contagem automática de passageiros; Arquitetura de sistemas

Integration of Travel Context Information into Multimodal Platforms

Abstract

Public transportation offers accessible, affordable, and environmentally friendly mobility solutions for millions of people worldwide. It contributes to a reduction of traffic congestion and carbon emissions and ensures equitable access to essential services. Reliable and efficient public transportation services are necessary to accomplish these goals. Therefore, public transportation operators need to make informed decisions regarding their services. Data such as passenger flow data can help identify peak travel times, underutilised routes, and emerging demand areas, enabling service adjustments to better cater to passengers' needs. While smart ticketing systems are useful for understanding passenger patterns, they do not provide detailed information on the number of passengers who have left the vehicle. This information can be helpful in managing passenger transfers between two different modes of transportation or routes, which is crucial to maintaining seamless connectivity and minimising travel times. To address this, we propose a system that collects entry and exit data from bus passengers through automatic passenger counter sensors. This data is later presented visually in a user-friendly dashboard to aid public transportation operators in improving their services. The results were promising, with the system performing as expected during field tests. The selected sensors achieved an accuracy of 93.75% for exit counts and 96.3% for entries. Although further calibration is needed to optimise accuracy, the system still proves to be a valuable tool for improving public transportation efficiency and addressing passengers' needs.

Keywords

Public transportation; Multimodal platforms; Automatic passenger counting; System architecture

Contents

List of figures

List of tables

Glossary

computer vision involves the development of algorithms and systems that enable computers to interpret and understand visual information from the world. It enables machines to analyse and make sense of images or videos, similar to how humans perceive and comprehend visual data. 9

coupling degree of dependency between different modules, components, or subsystems within a computer system or software application. 13

crowd-sourced obtaining information or opinions from a large group of people who submit their data via the Internet, social media, and smartphone apps. 2

distributed system a group of computer systems that are physically separated but are interconnected over the Internet that work together as a unified entity, sharing resources and processing tasks across multiple nodes. 5

geofencing technology that creates a virtual boundary or ”fence” around a specific geographic area using GPS, Wi-Fi, or cellular data. When a device, like a smartphone or a vehicle, enters or leaves this area, it triggers an action, such as sending a notification, alert, or performing a specific task. 27

machine learning a field of artificial intelligence that focuses on developing algorithms and models that enable computers to learn from data. Instead of being explicitly programmed for a specific task, machines learn patterns and make predictions or decisions based on the information they have encountered. 6

maintainability refers to the ease with which a product can be modified, updated, or repaired. 14

open-source a type of software or project whose source code is made available to the public. This means that anyone can view, modify, and distribute the code. 2

process an instance of a computer program that is being executed by a computer’s Central Processing Unit (CPU). It encompasses the set of instructions and tasks that a program carries out during its execution. 11

scalability ability of a system, application, or network to handle an increasing amount of work or growth in a seamless and efficient manner. 14

service refers to a discrete unit of software that runs a specific application. Each service typically represents a single component or part of a larger application. 23

sharding technique used to horizontally partition data across multiple servers or nodes in a distributed system. 16

Acronyms

ACID Atomicity, Consistency, Isolation, and Durability. 16

AMQP Advanced Message Queuing Protocol. 15

APC Automatic Passenger Counting. 5

API Application Programming Interface. 13

CAN Bus Controller Area Network Bus. 20

CORBA Common Object Broker Architecture. 13

DCOM Distributed Component Object Model. 13

DDB Distributed Database. 16

DFS Distributed File System. 16

DOP Dilution Of Precision. 28

GDOP Geometric Dilution Of Precision. 28

GDPR General Data Protection Regulation. 10

GNSS Global Navigation Satellite System. 20

gRPC Google Remote Method Invocation. 13

GTFS General Transit Feed Specification. 37

HDFS Hadoop Distributed File System. 16

HTTP Hypertext Transfer Protocol. 13

HVAC Heating, ventilation, and air conditioning. 7

MAC Media Access Control. 7

MQTT Message Queuing Telemetry Transport. 15

OBU On-Board Unit. 20

PLMN Public Land Mobile Network. 21

PoE Power over Ethernet. 21

PTSS Public Transportation Surveillance System. 9

REST Representational State Transfer. 13

RFID Radio-Frequency Identification. 5

RMI Remote Method Invocation. 13

RPC Remote Procedure Call. 12

SOA Service Oriented Architecture. 13

SOAP Simple Object Access Protocol. 13

TCP Transmission Control Protocol. 13

ToF Time of Flight. 10

URI Uniform Resource Identifiers. 13

WSDL Web Services Description Language. 13

XML Extensible Markup Language. 13

Chapter 1

Introduction

In this chapter, we introduce the project and its motivation, followed by a discussion of the objectives and contributions. Finally, we give an overview of the structure of the remaining document.

1.1 Background

As urban areas expand and change, the need for an efficient and reliable public transportation network becomes more evident in promoting sustainable development and urban resilience. Public transportation offers affordable and accessible travel options while reducing traffic congestion, environmental pollution, and energy consumption by lowering reliance on private vehicles. Additionally, effective public transportation services support economic growth by connecting people to jobs, educational institutions, and essential services [1, 2].

Planning and providing a transportation service isn't an easy task. When done successfully, it can encourage the public to stop relying so much on private vehicles. That is why public transport operators require reliable data about their services. Through this data, understanding and addressing key issues, such as optimising schedules and improving overall service quality, becomes an easier task. By analysing metrics such as the flow of passengers on various routes, public transport operators can make informed decisions. These data-driven strategies allow them to fine-tune routes, decide if a vehicle should be replaced with one better suited to demand, and enhance overall fleet efficiency.

Managing passenger transfers between two vehicles (i.e. a bus and a train) can benefit from this data as well by identifying patterns in passenger flow and peak times. It's not just the public transportation providers who benefit from this data; passengers can also use such information to their advantage. Access to occupancy data of a given bus or train, for example, allows individuals to stay informed about the status of routes relevant to them, making it easier to plan their trips and avoid congestion or overcrowded services. A well-aligned supply (services) with demand (passengers) improves the effectiveness of public transportation and contributes to a more sustainable and environmentally friendly urban landscape [3, 4].

1.2 Motivation

This dissertation results from a research project (ATE - P4.02 - Plataforma Multimodal) in the context of PRR [5] funds and “Alianças Verdes para a Inovação Empresarial”. Through this project, funding and resources were provided to contribute to develop public transportation services and empower public transport operators with comprehensive and actionable insights.

The literature on solutions that provide public transport operators with meaningful data about their services shows that the importance of accounting for the number of passengers who leave the vehicle is often overlooked. This information can be helpful in managing passenger transfers between two or more transportation methods. Commercially available applications like City Mapper [6] and Google Maps [7] allow the planning of a route with multiple public transportation methods if required. Despite this, they are passenger-focused, meaning that they offer insights to the end-users of the transportation service, not the provider. Although Google Maps supports vehicle occupancy data, its reliance on crowd-sourced information often results in inaccuracies or incomplete data. With that said, this project aims to address these gaps by providing transit operators with more reliable data, particularly regarding passenger disembarkation, to enhance the efficiency of public transportation systems.

1.3 Objectives

The main objective of this study is to evaluate the characteristics of the automatic passenger detection of different types of sensors, as well as to propose an architecture and implement a prototype that consolidates travel context information (i.e. passenger flow) in a dashboard-like interface. The objectives can be split into the following:

- Acquire an appropriate sensor for automatic passenger counting;
- Design a scalable and flexible architecture capable of handling various data sources;
- Perform evaluations of the prototype by deploying it in one or more operating buses.
- Prefer open-source technologies for the implementation;

1.4 Contributions

The work developed can be divided into several phases. First, existing solutions to estimate passenger occupancy and flow on board buses were evaluated to determine their feasibility and identify potential candidates for the technologies to be used in the implementation. Based on the insights gained from this evaluation, an architecture for the application was proposed to meet the specified requirements. Following this, sensors were acquired and set up to gather data, and a prototype of the proposed solution was implemented. Finally, practical field tests were conducted using a real bus to evaluate the effectiveness of the prototype and identify areas for improvement.

It's noteworthy to mention that a paper on the work done in this project, titled "Towards Integration of Travel Context Information into Multimodal Platforms", has been published and accepted. The paper was presented at the CENTERIS [8] 2024 conference, where it highlighted the advancements and insights gained from integrating travel context data into multimodal transportation platforms.

1.5 Document Structure

Following the contextualisation of the topic of the problem that this dissertation is trying to solve, its significance, and its objectives, this chapter concludes by outlining the document's structure. The following chapters are organised in the following manner:

- **Chapter 2**: includes a review of the literature related to the dissertation topic, such as methods for counting passengers and various technologies to choose from when implementing the prototype;
- **Chapter 3**: presents the architecture of the proposed solution as well as the hardware selected for it;
- **Chapter 4**: goes into detail regarding the implementation and the technologies used for the prototype;
- **Chapter 5**: contains the results of the evaluation of the prototype;
- **Chapter 6**: outlines the conclusions and future work that can be done to improve the project.

Chapter 2

State of the Art and Related Work

In this chapter, we aim to identify gaps in the literature and establish a foundation for the dissertation's contribution. Firstly, in Section 2.1, various methods for determining passenger counts in public transport vehicles are examined. Moving forward, Section 2.2 explores different approaches for data to flow in a distributed system. Finally, Section 2.3 compares different types of technologies for storing the processed data.

2.1 Automatic Passenger Counting Systems

Automatic Passenger Counting (APC) are solutions designed to simplify and improve the collection of accurate and timely data on the number of passengers using public transportation. These solutions usually uses sensors strategically placed at vehicle entry and exit points that automatically record and analyse passenger flow (entries and exits).

The data collected by APC systems is useful for assessing the utilisation of public transportation services and serves as a foundation for optimising route planning, resource allocation, and overall operational efficiency within the public transport sector. By embracing technological advancements like APC, public transit operators can make data-driven decisions and improve service quality, resulting in better passenger experiences.

The research on APC systems and their related analysis is quite extensive, as we will see. At a higher level, these systems can be classified based on their degree of interaction with passengers, as mentioned in [9]. "Integrated" systems directly interact with passengers, whereas "independent" systems do not. Over the next subsections, we will review the different variations of both system types, explore their key advantages and limitations, and examine their impact on passenger data accuracy and operational efficiency.

2.1.1 RFID Sensors

Radio-Frequency Identification (RFID) sensors fit in the integrated systems category, as these sensors rely on an RFID reader that emits radio signals to a nearby RFID tag carried by a passenger. Upon receiving a signal, the tag gives feedback in the form of an identification code and/or a series of data stored in its memory. This allows for seamless identification and

tracking of the passengers, enabling efficient and automated processes such as determining the occupancy of a vehicle and accurate record-keeping of travel activities. The advantage of RFID compared to other technologies is that it is a cheap and easily maintainable solution.

The authors of [10, 11] propose a ticketing system based on RFID, where physical tickets are replaced with a smart card that stores all the passenger information. After boarding the vehicle, the passenger registers using the smart card (RFID tag), and based on the number of registrations, the number of passengers on board the vehicle is determined, along with the number of available seats. In [12], the authors propose a similar approach to the one described earlier, but instead of storing data locally on the bus, they transmit it to the cloud for further analysis.

The study described in [13] examined the effectiveness of utilising RFID for automatic passenger tracking. Forty volunteers participated by carrying two types of RFID cards (single-tag and dual-tag) in various items such as backpacks, wallets, pockets, and hands. The experiment involved a simulated bus door with four reading antennas, and different configurations, including setups with one and two rows of individuals passing through the portal, were assessed. The average recognition value for four scenarios amounted to 91% for single-tag cards and 82% for dual-tag cards.

2.1.2 Weight Sensors

Weight sensors are another approach used to estimate the number of passengers boarding or disembarking a public transport vehicle. According to [14], the authors explain that treadle mats (pressure-sensitive mats) can be found at bus entrances and exit points. The inner part of the mats is designed with a grid of one or more open metal alloy plates that function as band switches. When someone steps on the mat, the pressure causes the plate to deform and close the circuit, which sends a signal to the control system. This signal is interpreted as the presence of a passenger. Additionally, the data produced by the mat is relatively small in size since it consists of simple on-off signals, which can be easily processed by the processing unit receiving the data from the sensors. When confronted with highly crowded environments, treadle mats have the drawback of poor accuracy since more than one person can be standing on the mat simultaneously, which might be interpreted as only one passenger.

In [15], the authors suggest a different approach for a mass-based automated passenger counting method. The proposed method utilises the vehicle air ride suspension system, a widely adopted technology in buses. The system accurately counts passengers by measuring the pressure inside the airbag suspension, which correlates directly with the vehicle mass. Two algorithms were developed for detecting boarding events based on time-resolved vehicle mass data. Assuming an average passenger mass of 76 kg, the first algorithm demonstrated a -2.4% error. The second algorithm, utilising the vehicle mass time derivative to enhance event resolution, showed an increased overall error of -28.4%. A comparable solution was developed in [16], with the difference being that the algorithms mentioned before were substituted with machine learning algorithms. These algorithms analyse signal shapes and predict entries and

exits based on them. The system was integrated into four in-use transit buses in two metropolitan areas. The machine learning algorithms had 91% accuracy for a subset of the data used in training the algorithms. However, testing with the trained algorithms showed that the mass-based APC was 61% and 38% accurate in recording entry and exit events, respectively.

2.1.3 Infrared Sensors

Infrared sensors are a specific type of motion sensor that uses infrared radiation, which is a part of the electromagnetic spectrum. This radiation has wavelengths that are longer than those of visible light but shorter than microwaves. The way the sensor works depends on its type. For instance, active infrared sensors have two main independent components: a light-emitting diode and a receiver. These sensors are typically organised in an array (either vertically or horizontally) at a vehicle's entrance and exit points, parallel to each other. When a passenger enters the field of view of the emitter and interrupts the emitting light source, the receiver interprets this as a presence. To ensure that passengers cannot evade the infrared ray and to accurately assess the crossing direction, installing a minimum of two sensors on each door is necessary, leading to increased costs. One advantage of this solution is that it does not invade the passenger's privacy since no information related to them is recorded.

Despite the lack of literature about APC solutions that utilise infrared sensors, generic pedestrian counting solutions include infrared sensors as a commonly used method. The effectiveness of infrared technology in accurately detecting and counting individuals has made it a prevalent choice in diverse applications such as building automation and management. In [17], the authors propose a system that measures people's flow through doorways using infrared sensors to better regulate an Heating, ventilation, and air conditioning (HVAC) system in a building. Also, the study results conducted in [18] indicate that once calibrated, infrared sensors can provide continuous and accurate information on the distribution of service times at airport checkpoints. Finally, it's worth noting that infrared sensors are often combined with other types of sensors to increase accuracy in widespread pedestrian counting applications, as indicated in [15, 19].

2.1.4 Wi-Fi

Mobile devices and Wi-Fi networks have become an integral part of our daily lives. As such, nowadays, numerous public transport vehicles are equipped with Wi-Fi access points for passengers to connect to the Internet. When a device, for instance, a smartphone, wants to connect to a Wi-Fi network, it must first discover the available networks. When looking for an available Wi-Fi network, devices send out a signal called a probe request frame. This signal contains the device's unique identifier, called Media Access Control (MAC) address, to all the access points in the surrounding area. The access points then respond with their own signal, called a probe response frame, which includes information about their network and the respective MAC address. The smartphone collects all of this information and displays a list of available

Wi-Fi networks for the user to choose from. This is a simplified explanation of how our devices discover Wi-Fi networks.

Building on this concept, the authors of [20] propose a method to determine the number of bus passengers by analysing the Wi-Fi signatures of mobile devices (probe requests) carried inside a transport vehicle. The approach shows promising results, though some challenges need to be addressed, such as the need for a passenger to be on the bus across several stops before being counted among the other occupants.

A common issue in determining occupancy is ensuring that only the devices inside the targeted vehicle are counted. Devices outside can interfere with the results and lead to an overestimation. To overcome this problem, in [21], the authors propose an approach that combines the data from several low-cost Wi-Fi sensors placed in the vehicle. For a device to be counted, all the sensors must receive the same probe request that follows specific parameters defined by the authors. The findings of their study show that using four sensors concurrently leads to a maximum overestimation of 15%. Using three sensors also mitigates overestimation, albeit to a lesser degree. The mean overestimation across all three sensor configurations was calculated at 67%.

Despite the advantages of having free Wi-Fi networks in public transportation, some privacy concerns have been raised. As mentioned before, when a device connects to a network, its identifier is broadcasted, making it possible to track the device's owner. Mobile device manufacturers have developed techniques such as MAC address randomisation to address this concern. Apple first introduced this technique in the release of iOS 8.0 in September 2014. Since then, other manufacturers have started implementing this technique with their own approaches as described in [22].

While this grants anonymity to passengers, it poses challenges for Automatic Passenger Counting (APC) systems. Now, a single device can have multiple different MAC addresses, making it harder to have reliable data about passenger numbers. To overcome this issue, several studies have been published with the objective of developing a de-randomiser algorithm capable of understanding which MAC addresses are more likely to be traced back to the same device. An example of this is presented in [23], where the authors have developed an algorithm capable of analysing multiple probe request frames and identifying patterns within the frame content while considering the reception timestamps. To test the algorithm's efficacy, 21 devices were examined, and it was found that the algorithm successfully detected all randomised MAC addresses. A similar method was used in [24] to create a de-randomisation algorithm. The authors could group probe request frames from the same device by extracting features from them and using them in machine learning algorithms. Additionally, a heuristic was developed to handle pseudo-random MAC addresses, which only changed when the emitting device switched the Wi-Fi interface on and off. Tests conducted in a controlled environment achieved an accuracy of nearly 96%, while in a real-world scenario, an average accuracy of 75% was attained.

It is also important to consider that in APC, relying on Wi-Fi approaches may result in

underestimating passenger volumes due to the possibility of passengers having multiple Wi-Fi-capable devices (smartwatches, tablets, etc.) or Wi-Fi turned off to preserve device battery. However, according to [23], this issue can be addressed by conducting a statistical survey of a sample of bus passengers. By doing so, obtaining a scaling factor that accounts for the problems mentioned above is feasible.

2.1.5 Computer Vision

Over the last few years, the global landscape of public transportation has witnessed a considerable transformation marked by the fast growth of Public Transportation Surveillance System (PTSS) [25, 26]. This is due to the increasing concerns about public safety and increasing vandalism. Additionally, advancements in video surveillance technology, including high-resolution cameras, video analytics, and cloud-based storage solutions, have further accelerated the demand for PTSS.

As a result, multiple solutions have been developed to improve the surveillance capabilities of cameras with passenger detection and counting features. Computer vision-based methods, such as the one proposed in [27], introduced a video-based surveillance system that detects passenger movements on board by analysing their behavioural patterns.

Several other methods for passenger counting have been proposed, utilising cameras placed near the top of the vehicle's door to capture a downward-vertical view of passengers as they enter and exit. Depending on the camera placement, different approaches use different types of cameras and image processing algorithms. For instance, in [28], the authors propose a method where captured frames are initially split into multiple blocks, each classified based on its motion vector. These blocks are then analysed to determine whether they belong to the same moving object. The results demonstrated an average accuracy of 92% for this approach.

Besides regular surveillance cameras, stereoscopic cameras are also used in APC. For context, stereo vision is used in computer vision to perceive depth using binocular disparity – the slight differences in the perspectives of the left and right eyes. This process mimics how we perceive the world with two eyes, resulting in a three-dimensional perception of the environment. In a typical stereo-vision setup illustrated in Figure 2.1, two cameras have parallel optical axes with a baseline distance d between them. The cameras assume identical focal distances f, and there exists a region in the scene visible to both cameras. The image-formation process projects a scene point p onto homologous pixels p_l and p_r of the left and right camera sensors, respectively. Disparity, defined as the horizontal difference between homologous pixels, decreases as the point p moves farther from the cameras. Stereo vision aims to extract information about the scene using visual data from the two images, a challenging task due to unknown homologous pixel pairs.

The authors of [29] propose an APC method using stereo vision sensors. The algorithm generates markers representing passengers' heads when boarding or leaving the bus. These markers are then tracked in the image sequence to reconstruct the passengers' trajectories. Later on, a tracking zone composed of two lines of interest was defined in the cameras' field

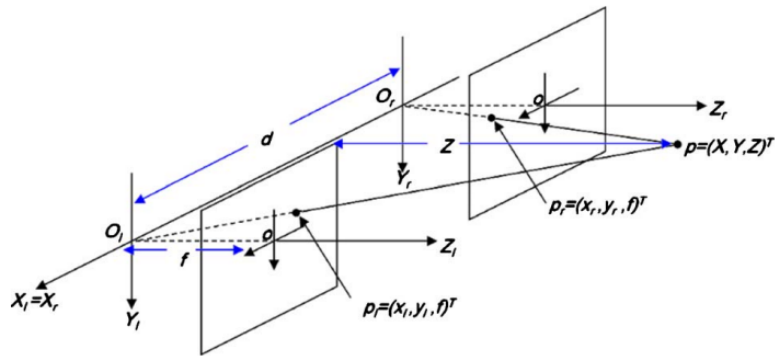


Figure 2.1 Geometric modelling of a binocular stereoscope. Extracted from [29].

of view, as illustrated in Figure 2.2. Depending on the line that a passenger crosses, they are considered to be either leaving or entering the bus. Ultimately, people were counted based on these trajectories. The proposed technique was validated through realistic experiments, demonstrating a counting accuracy of 99% and 97% on two large datasets featuring authentic scenarios.

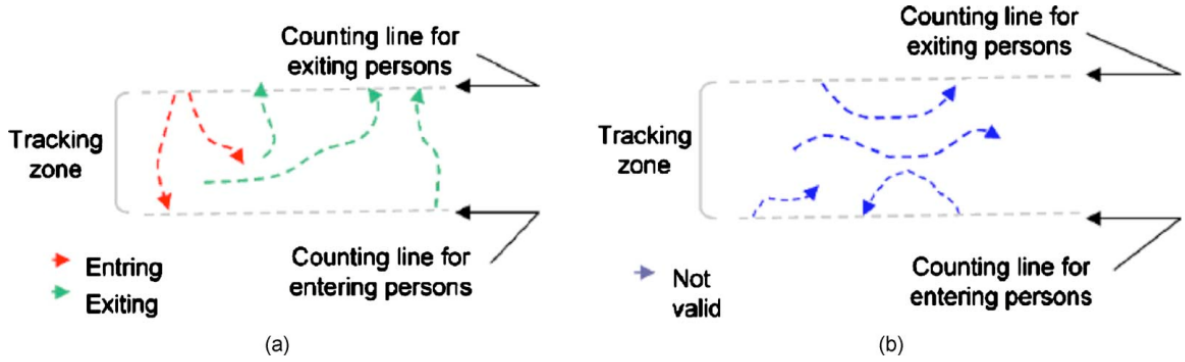


Figure 2.2 Examples of (a) valid and (b) invalid trajectories. Extracted from [29].

Despite the accuracy and value these types of systems bring to public transport operators, privacy concerns have also been growing, and ever since the General Data Protection Regulation (GDPR) was implemented in the European Union, data protection has been more prominent than ever. Thus, when implementing such systems in public transportation, the passengers' privacy has to be a priority. Some players in the market of APC products have contemplated this requirement and started developing solutions that use Time of Flight (ToF).

A ToF camera is a sensor that calculates the time it takes for light to travel from it to an object and back. By measuring this time, these cameras generate depth maps, as illustrated in Figure 2.3, enabling precise distance measurements and the creation of three-dimensional representations of the scene while preserving subject privacy. Another advantage of these sensors is their ability to function independently of ambient lighting, unlike stereoscopic sensors and other 2D cameras. Reflections, on the other hand, can pose challenges and affect the sensor's accuracy.

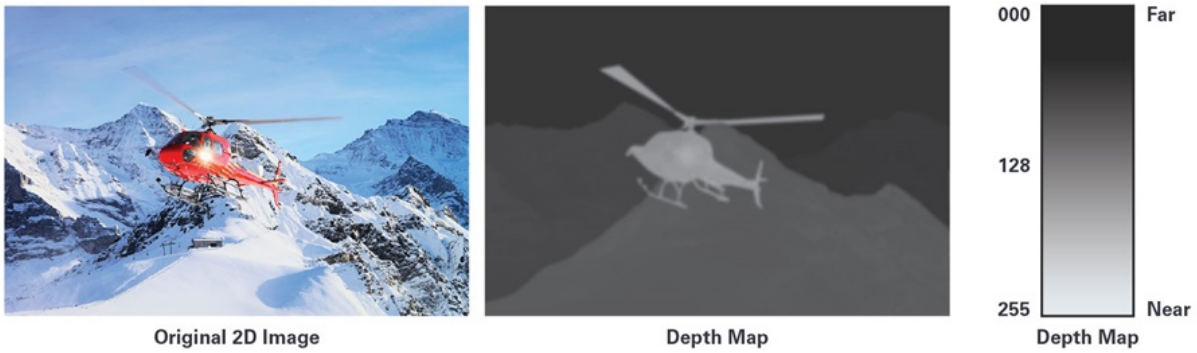


Figure 2.3 Example of a ToF depth map. Extracted from [30].

In [31], the authors have proposed a method for counting people using ToF sensors. The paper discusses the pre-processing of the images and provides a detailed description of the algorithms developed for processing 3D data to detect subjects. The experimental results are promising, with an accuracy rate ranging from 98.8% to 99.40%, depending on the resolution of the detection algorithm used.

Table 2.1 summarises the discussed APC methods up to this point.

2.2 Dataflow

When dealing with systems that receive data from different data sources, transmitting data between two processs lacking shared memory (e.g. sending data over the network) can be achieved in multiple ways. Dataflow influences compatibility, the connection between a process encoding the data and another decoding it. Ensuring both backward and forward compatibility is crucial, as it facilitates changes by allowing independent upgrades to different system components without requiring a simultaneous overhaul of the entire system. In [32], Kleppmann distinguishes between the following methods for how data flows between processes:

- Via Databases;
- Via Services;
- Via Asynchronous Messages.

2.2.1 Dataflow via Databases

In a database, data is encoded by the process that writes to it and decoded when the process reads from it. We can think that when a process writes data to the database, it writes a message to its future self when it eventually reads it. Thus, compatibility is important since the same process that wrote the data in the past may have changed in the future. We will not go into detail about dataflow via databases since it is not a feasible solution for guaranteeing compatibility, requiring additional mechanisms to be implemented. We will, however, discuss databases in Section 2.3 for data persistence.

Table 2.1 APC methods comparison.

APC Method	Pros	Cons	Studies and Results
RFID	High accuracy; Agnostic to environmental conditions.	Requires passengers to carry RFID tags; Costly if RFID tags have to be issued to every passenger; Privacy concerns.	[10], [11], [12], [13] with 82% to 91% accuracy.
Weight	Simple and low-cost implementation; No need for user interaction.	Accuracy affected by uneven distribution of passengers; Cannot differentiate between people, luggage, etc.	[14], [15] with error rates reduced by 2.4% and 28.4%, [16] with an accuracy of 61% and 38% for entry and exit counts.
Infrared	High accuracy; Agnostic to environmental conditions.	Low accuracy when passenger groups form; Calibration is needed for high accuracy.	[17] 93% accuracy in estimating the number of occupants in rooms, [18] with acceptable accuracy, [15], [19].
Wi-Fi	Uses passengers' devices to detect presence; Not expensive to implement.	Dependent on passengers having Wi-Fi-enabled devices; Limited accuracy due to multiple devices per person.	[20], [21] with a maximum overestimation of 15%, [23] with 94% accuracy in the dynamic test case, [24] with an accuracy of 75%.
Computer Vision	High accuracy; Can provide demographic information; Multiple solutions available on the market.	Expensive to install and maintain; Affected by ; Privacy concerns.	[27] , [28] with an average accuracy of 92%, [29] with an accuracy of 99% to 97%, [31] with an accuracy of 98.8% to 99.4%.

2.2.2 Dataflow via Services

A service is a unit of software that can be invoked through standardised procedures and can perform designated functions independently. Each service works in a heterogeneous environment, which includes different types of hardware, operating systems, and programming languages. Dataflow via services occurs when either other services or clients communicate with the service interface. This interface has to be previously defined and well-known to all the components that intend to use it. We can further distinguish between Remote Procedure Call

(RPC) services and Web services.

RPC services are widely used in distributed systems, the purpose of which is to replicate the local communication between the various components of an application. This abstraction is called location transparency, which allows a process to call a function from another process over the network. As stated before, the objects' interfaces containing the functions must be defined and known to make this possible. When implemented, this approach follows the Service Oriented Architecture (SOA) [33], resulting in a system with a low coupling level.

RPC systems, pioneered by SunRPC [34] in the 1970s, expanded over time with variations like Java Remote Method Invocation (RMI), Common Object Broker Architecture (CORBA), and Distributed Component Object Model (DCOM). Although these frameworks made significant contributions, they had certain limitations that hindered their widespread adoption. One of the biggest issues was their lack of compatibility with previous and newer versions. Additionally, these frameworks were too complex, as they were built on top of communication protocols like Transmission Control Protocol (TCP), making interoperability difficult due to a large number of specifications. Recent advancements in RPC focus on creating more versatile solutions that are agnostic to the programming language. An example of an RPC framework that supports multiple programming languages is Google Remote Method Invocation (gRPC) [35]. Initially developed by Google, it offers features such as bidirectional streaming and support for various platforms, using Protocol Buffers for serialisation.

Like RPC services, web services facilitate communication and data exchange between different services over the network, with the difference that Hypertext Transfer Protocol (HTTP) is used as the underlying protocol for communication. There are two main approaches to web services: Representational State Transfer (REST) and Simple Object Access Protocol (SOAP).

REST was introduced by Fielding et al. [36] as a design philosophy that relies on the principles of HTTP and on a stateless, client-server communication model, where the server holds the resources, and clients interact with these resources through well-defined, stateless operations. Resources are identified by Uniform Resource Identifiers (URI), and they can represent entities like data objects, services, or concepts. An Application Programming Interface (API) designed according to the principles of REST is called RESTful. By contrast, Simple Object Access Protocol (SOAP) is a protocol that relies on Extensible Markup Language (XML) for message formatting and typically operates over HTTP, despite avoiding using most HTTP features. The SOAP web service API is outlined using Web Services Description Language (WSDL), an XML-based language. WSDL facilitates code generation so that clients can interact with a distant service through local classes and method calls. This process involves encoding XML messages and subsequent decoding by the framework. While beneficial in statically typed languages, its utility is somewhat diminished in dynamically typed ones.

In [37], the authors compare RESTful versus SOAP web services in an indoor network composed of actuators. The authors' findings suggest that, upon initial examination, the REST architectural style is notably advantageous compared to SOAP in the context of the given system. This is evident in its ability to deliver fast query access and response times and reduce

CPU consumption and service load. The lightweight nature of REST is highlighted, leading to smaller message sizes, thereby requiring less parsing time and exhibiting lower latency. In [38], the authors also compare these two technologies based on metrics such as software maintainability, complexity, and others. The outcomes do not distinctly indicate a preferable choice for integrating systems between the two approaches. A thorough analysis of functional and non-functional requirements is necessary to make an informed decision regarding SOAP or REST. For instance, REST is a suitable choice when integrating two straightforward information systems. On the other hand, in scenarios where systems are intricate and demand additional security measures, SOAP emerges as the more fitting option, given its support for various standards.

2.2.3 Dataflow via Asynchronous Messages

Data transmission via asynchronous messages refers to sending and receiving data asynchronously through a message-oriented infrastructure. The element responsible for sending the messages is often referred to as the publisher, and the component that receives them is the consumer. In this approach, there is the need to have an element that acts as an intermediary, also known as a message broker, that relays the messages from the publisher to the consumer. This interaction scheme is known as Publish/Subscribe, or Pub/Sub [39]. Topics are abstractions representing logical channels or categories to which messages are published. Publishers publish messages on specific topics, and subscribers express interest in receiving messages from particular topics. Topics help categorise and organise the communication within a Pub/Sub system.

The benefits of Publish/Subscribe in distributed systems are as follows:

- Publishers and subscribers are decoupled in time, space, and synchronisation.
- Offers mechanisms for automatic message redistribution;
- One message can be sent to multiple recipients (fan-out pattern);
- Promotes horizontal scalability by removing all explicit dependencies between the interacting elements.

Message queues are another form of asynchronous communication between services. Unlike Pub/Sub, each message is only received by one consumer. In recent literature [40, 41], a variety of technologies leveraging Publish/Subscribe and message queue interaction schemes have been explored. Apache Kafka [42], a prominent example, is a distributed streaming platform well-known for its capability to process and handle large volumes of real-time data reliably. Kafka's publish-subscribe model facilitates the integration of diverse data sources, making it an ideal solution for real-time analytics, event sourcing, and log aggregation. Kafka clusters utilise Apache ZooKeeper [43] to maintain naming and configuration data, as well as to provide robust synchronisation and leader election functionalities within distributed systems.

Kafka's ecosystem extends through Kafka Connect and Kafka Streams, which improve its data integration and stream processing capabilities. Kafka Connect acts as a framework for building and running connectors, allowing seamless integration between Kafka and various external systems. It supports a broad range of connectors for databases, file systems, and cloud services, thus simplifying the movement of data between Kafka topics and different data sources.

Redis [44] is a data structure store that operates primarily in memory, serving as both a database and a cache. Redis can also function as a message broker, supporting Pub/Sub applications. Redis Pub/Sub enables publishers to broadcast messages to multiple channels, which is well-suited for scenarios where low latency is crucial. However, Redis Pub/Sub is an at-most-once messaging system without data persistence or acknowledgements, making it highly efficient but less reliable for scenarios requiring guaranteed message delivery and only supports a maximum message size of 512MB.

Message Queuing Telemetry Transport (MQTT) is a protocol designed for low-bandwidth, high-latency, or unreliable networks and is particularly well-suited for IoT applications. MQTT follows the publish-subscribe model and is implemented by open-source solutions such as Mosquitto [45] and HiveMQ [46]. MQTT offers various Quality of Service (QoS) levels, ranging from 0 to 2, which define the guarantees for message delivery between the MQTT broker and clients. Its lightweight nature allows for messages up to 256MB in size, making it ideal for efficient communication in constrained environments.

ZeroMQ[47] is a high-performance messaging library designed for distributed and scalable applications. Unlike traditional message-oriented middleware, ZeroMQ operates at the socket level and supports multiple messaging patterns, including publish-subscribe, request-reply, and push-pull. Its design eliminates the need for a central broker, which reduces latency and improves scalability. While ZeroMQ offers simplicity and flexibility, it lacks built-in message persistence and durability features, which may be a consideration depending on the use case.

Lastly, RabbitMQ[48] is a robust message broker implementation based on the Advanced Message Queuing Protocol (AMQP). It excels in routing, queuing, and delivering messages between producers and consumers, supporting a variety of messaging patterns such as point-to-point, publish-subscribe, and request-reply. RabbitMQ's strengths include its message acknowledgement feature, which ensures reliable delivery, and its support for message persistence, allowing messages to be stored on disk for added durability.

2.3 Data Persistence

In the context of the problem that we are trying to solve, after the data from the sensors is processed, it needs to be persisted somewhere for either audit purposes or to be consumed by a dashboard or other third-party applications. Considering the distributed nature of the pretended solution, ideally, the technology used for data persistence should include aspects like scalability, fault tolerance and consistency.

We can consider two possible types of solutions: Distributed File Systems (DFS) and Distributed Databases (DDB). A notable example of a distributed file system is Hadoop Distributed File System (HDFS). It excels at storing large files, providing fault tolerance and scalability. However, when dealing with smaller files, such as sensor data, HDFS performance degrades due to its limitation in supporting random reads of small files [49]. On the other hand, distributed databases can handle smaller files and provide the same scalability and fault tolerance as distributed file systems. That is why only distributed databases will be considered and explored for the solution over the following subsections.

2.3.1 Apache Cassandra

Apache Cassandra [50] stands out as a powerful and scalable NoSQL database designed to manage extensive datasets across multiple servers while mitigating the risks associated with single points of failure. Cassandra adopts a peer-to-peer distributed architecture, treating all nodes (each node is an instance of Cassandra) equally in communication. Data is distributed and replicated across nodes, ensuring fault tolerance and continuous availability. Cassandra's wide-column store data model promotes flexible schema design, resembling a table structure with rows and columns. Cassandra is known for its horizontal scalability, allowing seamless expansion by adding more nodes. Its architecture ensures high availability, even in the event of node failures. Furthermore, the database excels in write operations, while read operations are slower. Only recently, support for Atomicity, Consistency, Isolation, and Durability (ACID) transactions was added, making it an even more reliable solution. Despite its strengths, Cassandra has limitations. Achieving consistency can pose challenges, requiring careful tuning for optimal performance.

2.3.2 MongoDB

MongoDB [51] stands out among NoSQL databases with its document-oriented approach. It stores data in flexible, JSON-like documents, similar to rows in a SQL table. Collections in MongoDB are similar to SQL tables, containing sets of similar documents. This approach offers flexibility in data modelling, making it adaptable to changing schema requirements. MongoDB ensures horizontal scalability through automatic sharding and supports replication with locks and an asynchronous master-slave model. In version 4.0, released in 2018, MongoDB added support for multi-document ACID transactions and further extended that support for distributed multi-document ACID transactions in version 4.2 in 2019. One of the drawbacks of this solution is the lack of support for join operations, making queries across multiple collections more complex and resource-intensive. Also notable is the maximum document size of 16 MB.

2.3.3 CrateDB

CrateDB [52] is a distributed SQL database designed to handle large amounts of real-time data. It utilises a shared-nothing architecture, meaning that each node in the cluster operates

independently, without shared memory or storage. This design allows for horizontal scalability, enabling the addition of nodes to the cluster as data needs grow. CrateDB uses a sharding mechanism to distribute data across nodes. Data is divided into smaller, manageable pieces called shards, and each node is responsible for a subset of these shards. This parallelisation improves query performance and ensures that the database can handle vast amounts of data seamlessly. It's noteworthy to mention that CrateDB does not support ACID transactions, but instead, it has atomic operations and eventual consistency at the row level.

2.3.4 Data Persistence Methods Comparison

A concise comparison of NoSQL engines was conducted in Lourenço et al.'s study [53]. They highlighted the most beneficial use case scenarios and quality attributes from a software engineer's perspective. Cassandra outperforms MongoDB in terms of availability, scalability, write performance, and stabilisation time. However, MongoDB excels in read performance, recovery time, and reliability. Consistency, durability, and maintainability show similar performance between the two. Additionally, in [54], the authors compare MongoDB and Cassandra in application processing time for various query types on data streams with heterogeneous generation rates. In most cases, Cassandra outperforms MongoDB, though exceptions exist depending on the data stream rate. Regarding CrateDB, no interesting comparative studies were found.

Chapter 3

System Architecture

Building on our findings during the review of the State of the Art and Related Work in Chapter 2, we will now shift our focus to the design of the proposed solution. We start by discussing the system requirements. Following that, the system's architecture and the decisions made throughout its conception are explained in detail. After that, an overview of the required hardware is provided.

3.1 Requirements

According to the defined objectives, our goal is to implement a prototype of a system that can gather travel context information and present it visually in an interactive dashboard. As such, the prototype should meet the following requirements:

- **Accuracy**: the system must process and present information to the end-user with an accuracy of at least 90%. This includes precise passenger counts, vehicle locations, and stop data;
- **Passenger counts**: the system must continuously track and report the exact number of passengers entering and exiting at each stop, ensuring real-time updates for end-users;
- **Live occupancy**: the system must calculate and display the current occupancy level of each vehicle in real-time, based on the most recent passenger counts.
- **Route information**: the system must provide end-users with detailed information about the vehicle's current route, including a list of all stops, the vehicle's current location;
- **Fault tolerance**: the prototype must include mechanisms to detect, log, and recover from errors or malfunctions, ensuring the system remains operational and reliable under normal and adverse conditions.

3.2 Architecture Overview

Considering the requirements above, a diagram for the proposed architecture of the solution can be found in Figure 3.1.

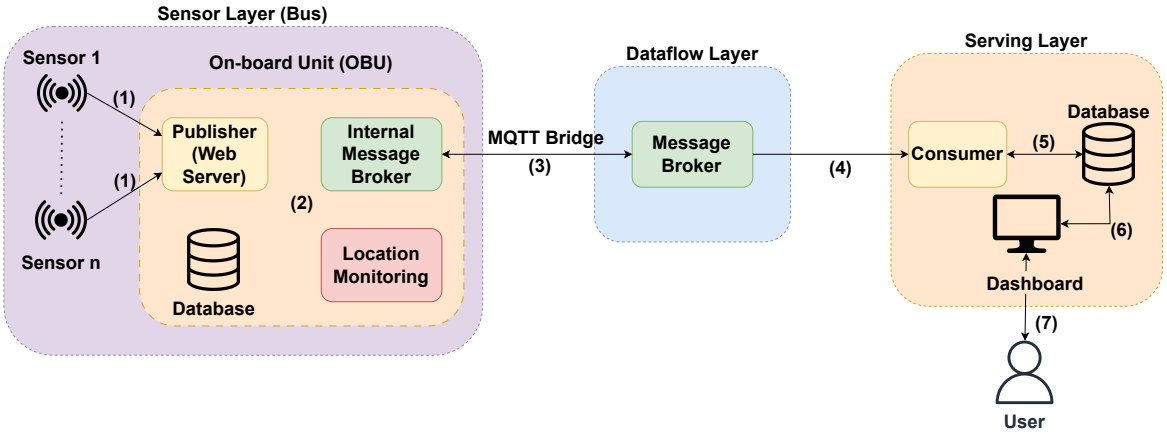


Figure 3.1 Proposed architecture of the solution.

The architecture is composed of the following layers:

- **Sensor Layer**: consists of all the sensors and equipment arranged on a bus, such as the Global Navigation Satellite System (GNSS)/GPS antenna, automatic passenger counter sensors, and the Controller Area Network Bus (CAN Bus), a unit that follows a vehicle standard designed for microcontrollers and other devices to communicate with each other. Additionally, as this project involves a remote environment (bus) that needs to communicate with a central server, an On-Board Unit (OBU), also called a gateway, will be required. This gateway will serve as a network bridge between the bus and the central platform, allowing the transmission of local data from the various sensors. An overview of all the required equipment inside the bus can be seen in Figure 3.2.

  In this layer, the information regarding passenger flow gathered from the sensors will be transmitted using HTTP over a TCP/IP connection to the gateway, more specifically to the publisher service operating on it (1). The sensor data will be processed and summarised within the gateway according to each bus stop. This data will then be correlated with information from the CAN Bus and the GNSS and stored in a database to provide a comprehensive overview of the vehicle's capacity, operations, and location (2).

- **Dataflow Layer**: consists of a message broker that plays a critical role in managing the data received from the sensor layer (3). Its primary responsibility is to obtain the information from the various buses and efficiently distribute it to the consumers within the serving layer (4). The message broker ensures that data is accurately and promptly delivered, facilitating seamless integration and real-time data processing. An MQTT bridge and a QoS level of 2 are required between the sensor and dataflow layers to ensure that when communication problems arise, the data will be stored and sent once it's possible.

- **Serving Layer**: composed of a consumer service that receives and stores the processed data from the dataflow layer into a dedicated database (5). Finally, the dashboard queries the database (6) and then displays the data to the end user (7).

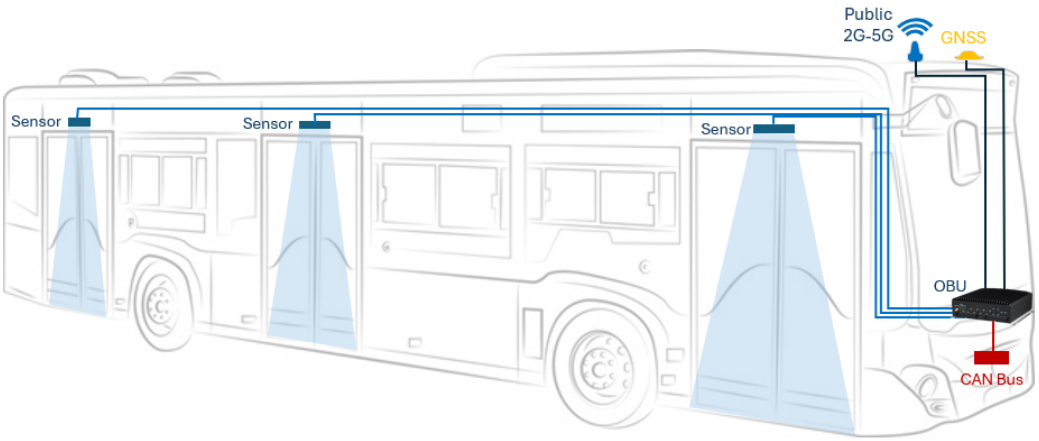


Figure 3.2 Schematic of the equipment inside the bus.

3.3 Hardware

After finalising the prototype's architecture, the next step was to select the necessary hardware based on the established architecture, which involved sourcing an APC sensor, a gateway, and antennas.

In Chapter 2, we explored different types of APC sensors. Our objective for this project is to develop a system capable of integrating sensor data into an existing platform instead of developing a new APC method. Given the numerous solutions readily available in the market and their accuracy, we opted to use stereoscopic cameras for passenger counting. We reached out to multiple companies and ended up acquiring one sensor from Hikvision [55] and another one from Xovis [56], depicted in Figure 3.3. Despite using the same principle of stereovision, the sensors differ in the actual processing algorithm applied to the images to detect and count passengers.

For the gateway, our partners at Solvit [57], who have had experience with these types of equipment, recommended the VTC 6222 in-vehicle computer from Nexcom [58]. As illustrated in Figure 3.4, the gateway has a slot for a SIM card to connect to a Public Land Mobile Network (PLMN) provider, four Power over Ethernet (PoE) ports that supply the sensors with power and Internet connectivity, built-in GPS module and CAN Bus support. Finally, two generic antennas were used for GPS and mobile network communications.

(a) Hikvision sensor **(b)** Xovis sensor

Figure 3.3 Acquired APC sensors.

(a) Front

(b) Back

Figure 3.4 Acquired gateway.

Chapter 4

Implementation

In Chapter 3, we discussed the design of the architecture for the prototype. This chapter will focus on the implementation of that architecture. We will present and explain the developed prototype while introducing the selected technologies and providing reasons for their inclusion.

4.1 Sensor Layer

Software-wise, the sensor layer has been programmed in Python, a high-level, interpreted programming language known for its simplicity, readability, and versatility. It is a popular choice for developing prototypes because of its extensive libraries, which simplify the development process by reducing the amount of code needed. However, Python also has some drawbacks. As an interpreted language, it can be slower in execution compared to compiled languages like C or C++, which may be a concern for applications requiring high performance or real-time processing.

Python's memory usage is also relatively high, which can be limiting in environments with constrained resources. Additionally, while increasing flexibility, Python's dynamic typing can lead to runtime errors that would be caught at compile-time in statically typed languages. Therefore, we may consider using a different language for the final version of the system we are developing.

The services that are responsible for gathering and processing all the sensor information will be executed inside the gateway/OBU. Knowing that this prototype may run in various vehicles, we opted to use Docker [59], a platform for developing, shipping, and running applications inside lightweight containers. Docker containers package the application and its dependencies in a standardised unit, ensuring consistency across different environments. This allows the prototype to be easily deployed and executed on any vehicle, regardless of its underlying operating system or hardware configuration. Docker also provides isolation between services, reducing conflicts and making managing and updating individual components easier without affecting the entire system.

To manage multiple Docker containers/services, we used Docker Compose, a tool that allows us to define and run multi-container Docker applications using a single YAML file. With

Docker Compose, we can specify all the services, networks, and volumes needed for the application in one place, making it straightforward to start, stop, and configure the entire stack. This approach improves development efficiency by enabling easy orchestration of services, ensuring that all necessary components are up and running correctly, regardless of the environment in which they are deployed. The Docker Compose file used for the sensor layer can be found in Appendix A.

Figure 4.1 illustrates the interactions between the four services specified in the docker-compose file for this layer.

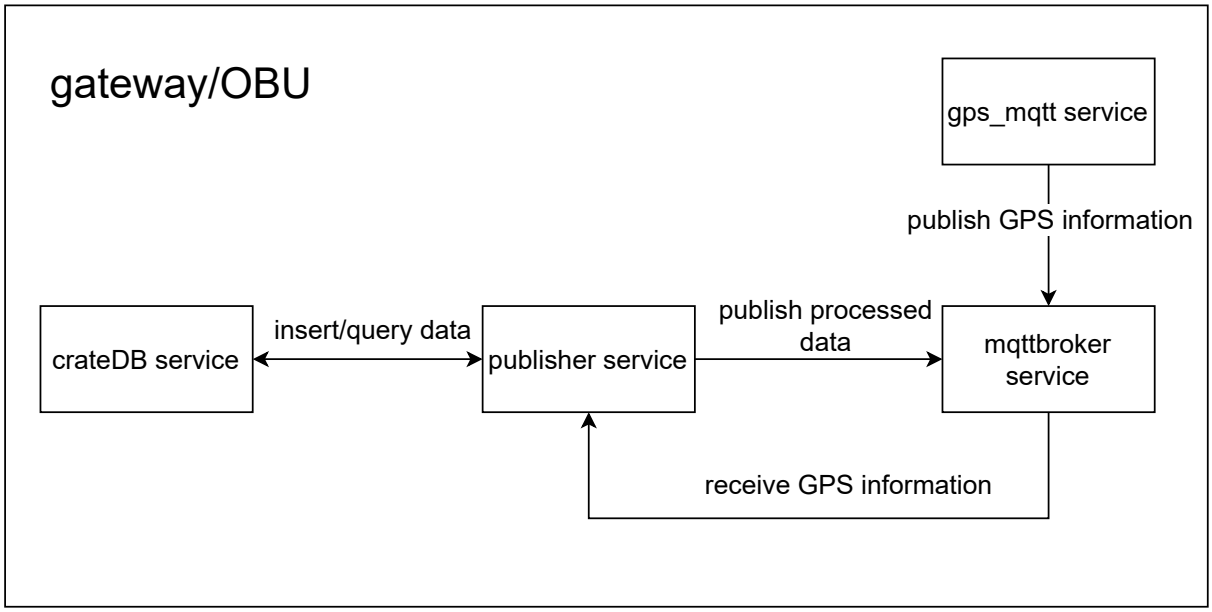


Figure 4.1 Interactions between the gateway services.

Starting with the *mqttbroker* service, as the name suggests, functions as an MQTT broker to facilitate message communication. For this, we used a Mosquitto image for the container because it supports configuring bridges between different brokers while maintaining a lightweight footprint, making it suitable for our use case.

Following the *mqttbroker* service, we have the *gps_mqtt* service, which is responsible for processing GPS signals containing information such as the coordinates, the bus speed and other important metrics. This service receives the raw GPS data (in decimal degrees) from the gateway and then publishes it through the previously mentioned MQTT broker. It's important to note that this communication is strictly internal to the gateway; hence, all messages containing the GPS information are published within the gateway environment and are not exposed outside.

Moving on to *crateDB* service, it serves as the local database for storing and managing data collected from various sources, such as GPS coordinates, sensor readings, and other relevant information. A CrateDB image was used as it is particularly well-suited for handling time-series data, making it ideal for efficiently storing and querying large volumes of real-time data generated by the system. Additionally, its distributed architecture ensures high availability and scalability, allowing the database to handle increased loads as incoming data grows.

Finally, the *publisher* service is responsible for receiving various types of data, such as GPS and sensor information and processing it. Figure 4.2 shows the interactions between the following Python modules that form this service:

- **publisher.py**: this module is the entry point for the publisher service and is responsible for initialising all the other modules. As mentioned, the sensors send a message containing the information in Listing 4.1 via HTTP requests [1]. To handle these requests, we utilised Flask [60], a lightweight and flexible web framework for Python that is used for building web applications and APIs;
- **mqttclient.py**: acts as an MQTT client that subscribes to the *internal/gps* topic that contains the GPS messages published by the *gps_mqtt* service [2]. Once the sensor data has been processed and is ready to be sent, the client publishes a message to the local MQTT broker [3] that gets relayed to Solvit's MQTT broker located at the dataflow layer, ensuring that the data is communicated effectively into the broader system;
- **dbclient.py**: this module implements a driver that facilitates communication with the local database managed by the *crateDB* service. When data is received by the *publisher.py* module, the **dbclient.py** stores this data in the database [4] according to the data model in Table 4.1. The stored data is later queried for further processing [5].
- **geofencemonitor.py**: in the requirements, we specified that the prototype must include information about the bus stops along a given route. Therefore, when the publisher service is initialised, data about the bus stops on the scheduled route is fetched and added to the system. Afterwards, the module monitors the bus's location by comparing the received coordinates with the bus stop data [6]. If it detects that the bus has left a bus stop, a thread is created to publish a message[7] containing the passenger flow data from that bus stop as shown in Listing 4.2. We discuss this module in greater detail and its importance in Section 4.1.1.

Listing 4.1 Unprocessed sensor data structure.

```
<EventNotificationAlert version="2.0"
xmlns="http://www.isapi.org/ver20/XMLSchema">
    <ipAddress>1.2.3.4</ipAddress>
    <portNo>5000</portNo>
    <protocol>HTTP</protocol>
    <macAddress>aa:bb:cc:dd:ee:ff</macAddress>
    <channelID>1</channelID>
   <dateTime>2024-09-24T15:37:00+01:00</dateTime>
    <activePostCount>1</activePostCount>
    <eventType>PeopleCounting</eventType>
    <eventState>active</eventState>
```

```
    <eventDescription>PeopleCounting alarm</eventDescription>
    <channelName>TEST</channelName>
    <peopleCounting>
        <statisticalMethods>timeRange</statisticalMethods>
        <TimeRange>
            <startTime>2024-09-24T15:36:00+01:00</startTime>
            <endTime>2024-09-24T15:37:00+01:00</endTime>
        </TimeRange>
        <enter>14</enter>
        <exit>9</exit>
    </peopleCounting>
    <childCounting>
        <enter>0</enter>
        <exit>0</exit>
    </childCounting>
</EventNotificationAlert>
```

Listing 4.2 Processed sensor data structure.

```
{
    "stop_id": 174287,
    "route_id": 13B,
    "enter_count": 16,
    "exit_count": 9,
    "onboard_bus": 10,
    "bus_identifier": BF93SD,
    "lat": 12.345678,
    "lon": -1.2345678,
    "timestamp": "2024-08-25T14:58:23+00:00"
}
```

4.1.1 Optimisations

As data is being sent by the sensors every minute, the gateway must store it in a local database before sending it to an external one. At this rhythm, the storage would be used up quickly, leading to inefficiencies and potential data loss. Two options were considered to address this issue. The first option involves changing the rhythm of data transmission from the sensors, which would reduce the granularity and real-time nature of the data. This approach would decrease the amount of data generated and stored, thereby conserving storage space. However, it would also result in less detailed data, which might limit the precision and timeliness of passenger count information.

The second option involves discarding events that do not report passenger counts. This

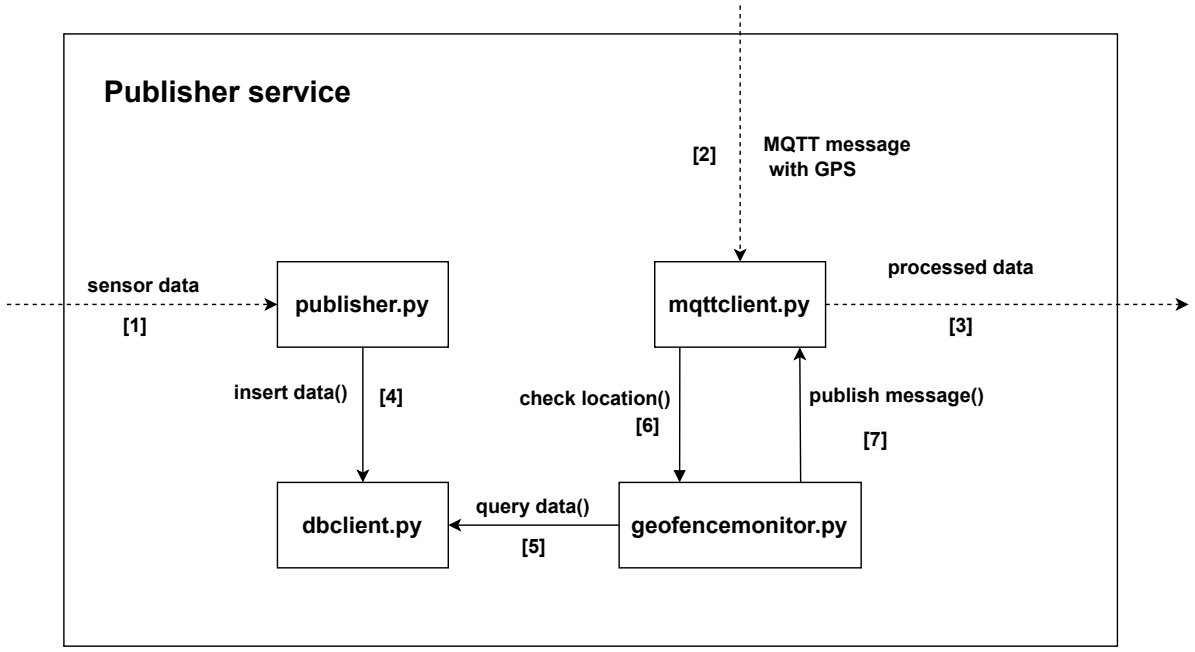


Figure 4.2 Interactions between the Python modules of the publisher service.

approach is based on the inference that if no data is reported, it implies that the passenger count is zero. By discarding these events, we can significantly reduce the volume of data stored and transmitted while maintaining the integrity and usefulness of the recorded data. This approach ensures that only relevant data is retained, optimising storage usage and maintaining the system's efficiency.

When data is sent from the gateway to the dataflow layer, costs are associated with this. The gateway inside the bus communicates with the exterior via a mobile network, so outbound communications should be minimised. It wouldn't be sustainable to send data continuously, so some optimisations were made.

Figure 4.3 illustrates a geofencing algorithm that was implemented in the *geofencemonitor.py* module that determines when a bus enters and leaves a region of interest (i.e., the perimeter around a bus stop). In the context of a bus, during the "dwell" period inside the geofence perimeter, the bus will arrive at a stop, and passengers will leave and enter the bus. That information is stored and is only sent when the bus leaves the geofence perimeter. In the current implementation, the radius of this perimeter is configurable and depends on the route that the bus is taking. It's important to remember that the radius of the stops can't overlap, which can be a problem when they are close to each other.

It is also necessary to consider scenarios in which the bus is in areas that are prone to deteriorating the GPS signal received. The accuracy of the received coordinates is related to the quality of the GPS signal; thus, if the reception is poor, we may have situations where the system "thinks" that the bus has entered or exited a bus stop incorrectly. This can lead to false notifications or inaccurate tracking, impacting the reliability of the service. To minimise this issue, a rolling window has been implemented. It takes the last nine observed coordinates plus the current coordinates and calculates the mean to eliminate potential outliers.

Table 4.1 Data model of the data stored inside the gateway.

Name	Meaning
ip_address	IP address of the sensor (either IPv4 or IPv6).
sensor_identifier	For Hikvision sensors represents the MAC address.
lat	Latitude.
lon	Longitude.
start_timestamp	Timestamp of the start of the counting.
end_timestamp	Timestamp of the end of the counting.
enter_count	Number of people who crossed the line in the forward direction of the line.
exit_count	Number of people who crossed the line in the backward direction of the line.
bus_identifier	Identifier of the bus (license plate).

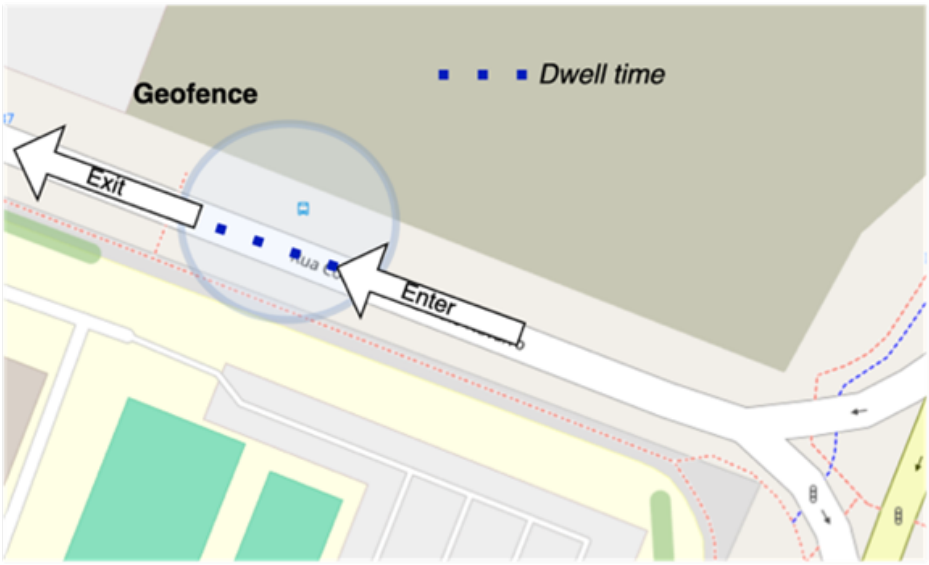


Figure 4.3 Geofencing algorithm representation.

The message containing the GPS information also holds the various Dilution Of Precision (DOP) values as represented in Listing 4.3. In practice, these values represent the error due to the positions of the satellites in the sky. The closer they are to the horizon, the more atmosphere and obstacles the signals have to pass through to reach the receiver. Upon receiving the GPS messages from the MQTT broker, the system also checks this value. It discards messages that have a Geometric Dilution Of Precision (GDOP) (combination of the various DOP values) bigger than five, which is when the GPS signal is considered to be of “moderate” quality according to [61].

Listing 4.3 GPS mssage data structure.

```
{
    "time":"2024-08-25T14:58:23+00:00",
    "lat":12.345678,
```

```
    "lon":-1.2345678,
    "alt":87.354,
    "speed":0.0648,
    "xdop":0.58,
    "ydop":0.5,
    "vdop":1.05,
    "tdop":0.72,
    "hdop":0.76,
    "gdop":1.48,
    "pdop":1.3
}
```


4.2 Dataflow and Serving Layers

Regarding the MQTT broker of the dataflow layer, Solvit authorised us to use their broker to publish messages. The only configuration needed was to set up a bridge connection in the MQTT broker running on the gateway to connect to Solvit's broker.

Similar to the sensor layer, the serving layer is defined within a Docker Compose file containing three services: the *consumer*, which receives the data published to the MQTT broker located at the dataflow layer and stores it in a database managed by the *crateDB* service according to the data model shown in Table 4.2.

Table 4.2 Data model of the processed bus stop data.

Name	Meaning
stop_id	Identifier of the bus stop.
lat	Latitude.
lon	Longitude.
timestamp	Timestamp of the message.
enter_count	Number of people who crossed the line in the forward direction of the line.
exit_count	Number of people who crossed the line in the backward direction of the line.
onboard_bus	Number of people onboard the bus.
bus_identifier	Identifier of the bus.

Finally, the *grafana* service, which, as the name suggests, consists of a Grafana Docker image to display the data in a dashboard. We opted to use Grafana since it provides a free, robust, ready-to-use dynamic dashboard with support for CrateDB as a data source.

Chapter 5

Evaluation and Results

In this chapter, we assess the prototype by focusing on the accuracy of the automatic passenger counting sensors through both controlled and field tests. Furthermore, we showcase the dashboard that the transportation operators can use to visualise real-time passenger data, including entry and exit counts, bus stop metrics, and overall vehicle occupancy.

5.1 Sensor Evaluation

Our project heavily relies on the accuracy of the sensor data to establish passenger patterns that can help public transport operators optimise their services. Therefore, it is important to ensure that the sensor readings are as precise as possible.

After acquiring the sensors mentioned in Section 3.3, we conducted tests in a controlled environment, more specifically at the entrance points of a building. The objective was to evaluate the sensor's performance based on hypothetical scenarios that could occur in a bus. Based on the results of Table 5.1, we have recorded three separate attempts, labelled as #1, #2, and #3, with each result formatted as entry count/exit count. Looking at the results, the sensors behave comparably in the first test case. However, in the second and last test cases, we can observe that sensor B outperformed sensor A. These two scenarios are particularly significant as they are prone to occurring and could result in counting inaccuracies. In total, sensor A has produced seven accurate results, while Sensor B has yielded ten. Considering this, Sensor B has been the one used for the prototype.

Additional field tests were conducted in collaboration with a public transportation operator located in the Porto Metropolitan Area, Portugal. The hardware was installed on a bus currently in operation and has been collecting real data for two months. Figure 5.1 shows an image captured from the live feed of the sensor, where we can see the passenger counts (0 entries and 334 exits) in the top-right corner. The red rectangle is automatically generated by the sensor and indicates the possible area where the counts will be considered. The blue rectangle is user-defined and specifies the area from which we want the sensor to count within the red area. Finally, the yellow line defines the line of interest and the direction for recording entries and exits. The arrow extending from the line indicates the direction of entries. Since this camera

Table 5.1 Results of the controlled environment tests on the APC sensors.

Test case	Expected	Sensor A results			Sensor B results		
		#1	#2	#3	#1	#2	#3
3 people are crossing at the same time in each direction.	3 entry / 3 exit	3/2	3/3	2/3	2/3	3/3	3/3
1 or more people standing on the line to see how the sensor reacts.	0 entry / 0 exit	1/2	2/1	0/2	1/0	0/0	0/0
3 people cross the line simultaneously in different directions (i.e., 1 enters, 2 exits).	1 entry / 2 exit	1/2	1/2	1/2	1/2	1/2	1/2
2 people standing on the line, the other 1 exiting.	0 entry / 1 exit	0/1	0/1	0/1	0/0	0/0	0/0
1 person goes back and forth 4 times in a short period of time.	0 entry / 0 exit	1/1	1/1	1/1	0/0	0/0	0/0

is positioned at one of the exits of the bus, the arrow is pointing inward so that when passengers cross it, the sensor interprets it as an exit.

Table 5.2 displays the results of the passenger counts for ten bus stops during field tests. We compared the counts from the sensors (observed row) against the recordings of the sensor images (expected row). That way, using ground truth, we could validate the accuracy of the sensor data by comparing it directly with the manually recorded passenger counts. During the observed stops, a total of 54 passengers entered the bus. The sensors reported 52 passengers, giving us an accuracy rate of approximately 96,3%. As for the exits, 16 passengers left the bus, and the sensors counted 15, resulting in an accuracy rate of 93,75%. During the analysis of the sensor footage, it was observed that the primary factor affecting sensor accuracy was the height of the passenger. As a result, additional sensor calibration will be necessary to minimise this issue. We also noticed that at certain times, the bus driver would leave and enter the bus multiple times between different routes. This leads to inaccurate passenger counts, as the system may register the driver's movements as additional entries or exits, skewing the data.

Table 5.2 Results of the field tests on the APC sensors.

Counts	Bus stop									
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
Expected (entries/exits)	2/2	5/2	1/5	1/0	36/0	1/2	3/0	1/2	2/1	2/2
Observed (entries/exits)	2/2	5/2	1/4	1/0	34/0	1/2	3/0	1/2	2/1	2/2

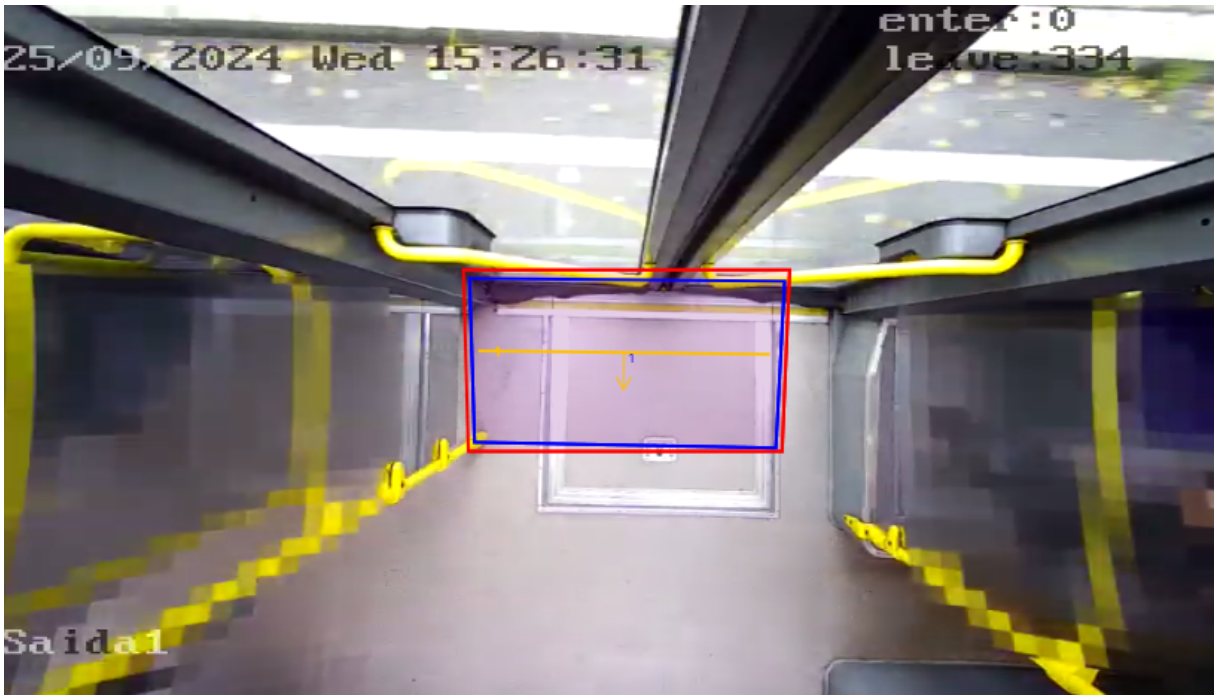


Figure 5.1 Image from the sensor in field tests.

5.2 System Evaluation

Tests were carried out to assess the system's fault tolerance in cases where the mobile network was insufficient to establish communication between the bus and the dataflow layer. The results were promising, as we configured an MQTT bridge between the local broker at the bus and the one in the dataflow layer. When the connection was lost and a message was sent from the bus, the message was successfully queued and stored locally. Once the network was restored, the system automatically transmitted the stored messages to the server without data loss, ensuring smooth and reliable communication even in intermittent network conditions. In more extreme cases where communication is completely disrupted for an extended period, the system also stores the messages in the local database that can be retrieved at a later time.

The developed Grafana dashboard is composed of different panels with relevant information to satisfy the requirements set in Section 3.1. The data that is stored by the consumer service is fetched by Grafana according to user-defined SQL queries. Furthermore, the usage of variables inside Grafana allows the data to be filtered by the bus stop number or by bus, providing fine-grained results while allowing for more flexible queries.

Looking at the panels themselves, Figure 5.2 illustrates a panel containing an interactive map that displays the route that the bus is performing, marked by the blue icons. The green icon represents the last stop that the bus has been to. Each point contains information about the passenger flow (entries and exits), the stop and route identifier, and the location of the bus stop. Figure 5.2 displays similar information in a table format. Finally, Figures 5.4 to 5.7 address the other requirements, such as the live occupancy of the vehicle and the passenger counts in various time scales (daily, weekly and monthly).

Based on the various panels, we can also verify that the results are relatively good, in the sense that the total entries match the total exits, with an error of around 2.3% based on the total counts shown in Figure 5.5. The bar charts of passengers per week and month are also an indicator of the overall consistency of the data.

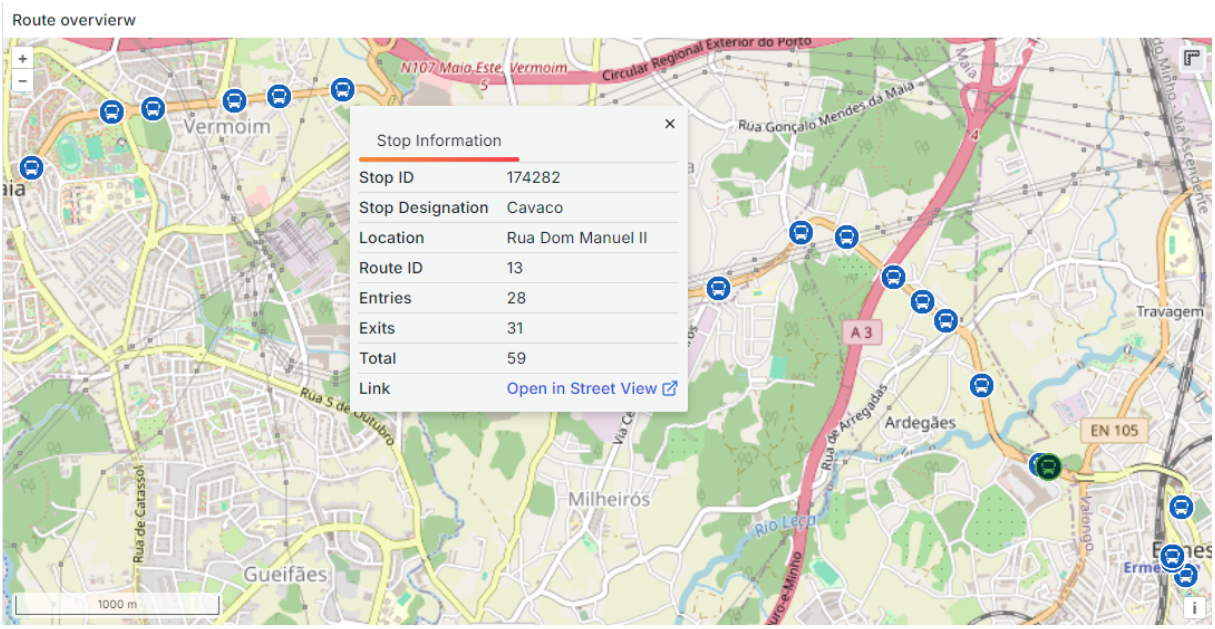


Figure 5.2 Dashboard panel of the route overview map.

Data per stop

Route ID	Stop ID	Stop Designation	Location	Entries	Exits	Total
13	174287	Sobreiro	Avenida Dom Manuel II	569	686	1255
13	174265	Ermesinde (Estação)	Largo da Estação	328	404	732
13	174266	Estação Ermesinde	Largo da Estação	139	156	295
13	174276	Monte Penedo	Rua Doutor Joaquim Nogueira dos Santos	140	145	285
13	174275	Primavera	Avenida Dom João I	108	145	253
13	174277	Rio	Rua Doutor Joaquim Nogueira dos Santos	142	101	243
13	174286	Maia (Estádio)	Avenida Dom Manuel II	133	50	183

Figure 5.3 Dashboard panel of the bus stop table.

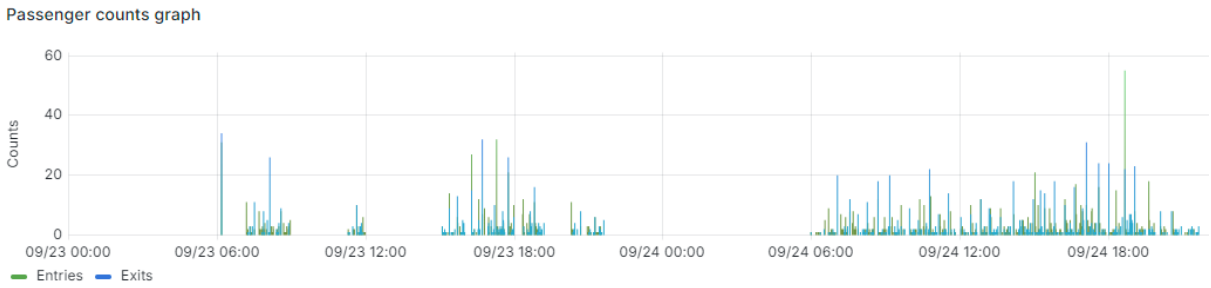
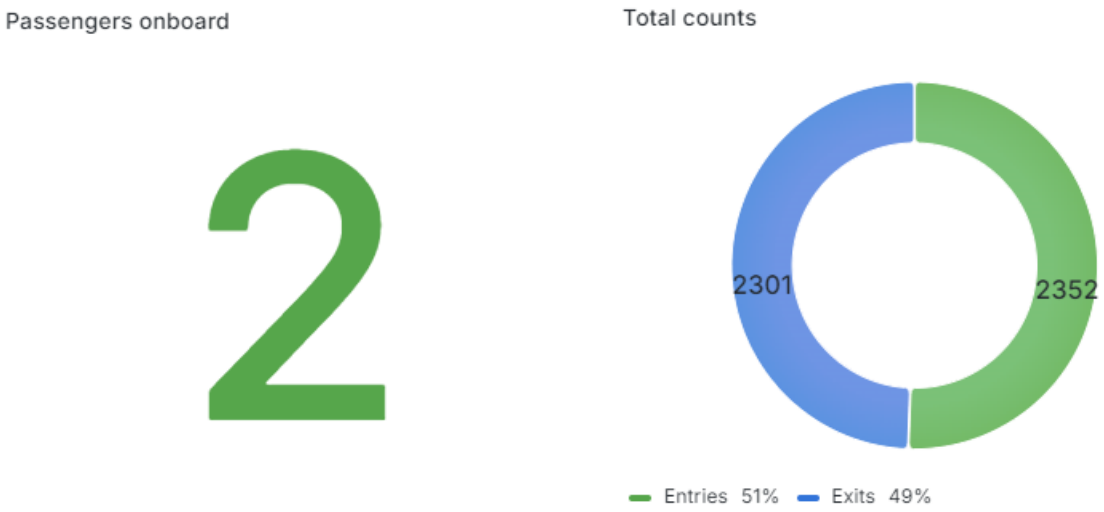


Figure 5.4 Dashboard panel of the time series passenger flow data.



(a) Passengers onboard **(b)** Total counts

Figure 5.5 Dashboard panels of the passengers onboard and total counts.

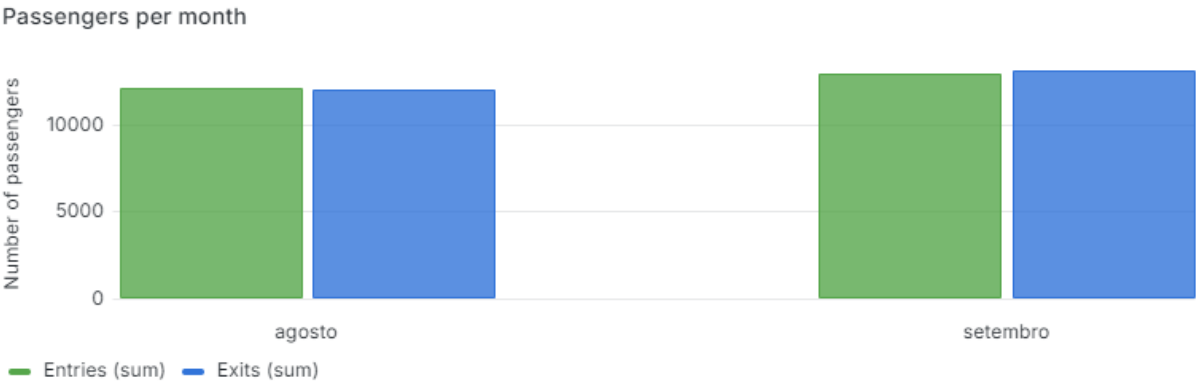


Figure 5.6 Dashboard panel of the passengers per month.

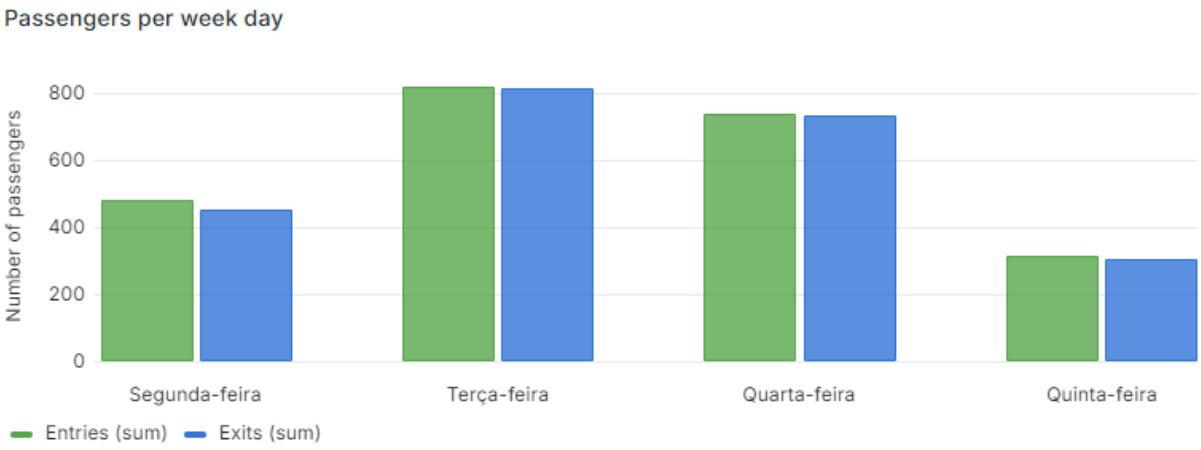


Figure 5.7 Dashboard panel of the passengers per week.

Chapter 6

Conclusions and Future Work

This thesis highlights the critical role of integrating Automatic Passenger Counting systems with cutting-edge sensor technologies to improve public transportation networks. The research involved the successful development of a prototype system that captures real-time passenger flow data and integrates it into an accessible dashboard for transportation operators. The system was built using a combination of hardware, including stereoscopic cameras and an on-board unit, and supported by software developed with open-source technologies. Together, these components enable real-time monitoring and provide valuable insights into passenger behaviour and transportation trends.

Field testing of the prototype demonstrated that the selected sensors met the necessary accuracy standards for tracking passenger entries and exits. Furthermore, the system's overall performance was robust, operating without major issues or disruptions. The success of this project suggests that the chosen combination of hardware and software offers a practical and reliable solution for passenger flow monitoring in real-world public transportation settings.

Ultimately, this project contributes to the growing demand for accurate and reliable data in public transportation systems, particularly as cities continue to expand their multimodal transit networks. By providing real-time data on passenger movements, the developed system offers transportation operators an essential tool for optimising service efficiency, improving passenger experiences, and supporting future urban mobility planning. The findings reinforce the importance of incorporating advanced technology into transportation systems to meet the evolving needs of modern cities.

Despite meeting the initial requirements, several improvements can be made to improve the robustness and utility of the system for transportation operators. One area for future development is testing the system with a larger fleet of buses operating concurrently, which would help assess the scalability and performance limits of the architecture. This would provide valuable insights into how the system handles increased data flow and whether any optimisations are necessary to maintain performance under heavier loads. Fortunately, we are in the process of deploying the sensor layer hardware in more buses, including a transportation operator located in the Lisbon metropolitan area. In addition, integrating more detailed data from the CanBus and the transportation operator General Transit Feed Specification (GTFS) data, such

as average bus speeds, route context information, and fuel efficiency, would offer transportation operators a more comprehensive view of fleet operations. These metrics could help in understanding vehicle performance and optimising operational efficiency. Another important feature to consider for future work is implementing real-time alerts for issues such as delays, mechanical problems, or route deviations. These alerts would allow operators to respond more quickly to disruptions, improving service reliability and customer satisfaction. Finally, security considerations must be prioritised to ensure the system is robust against potential cyber threats. Implementing better security measures will safeguard the system against unauthorised access and data breaches, ensuring the reliability and integrity of the solution for transportation operators.

Annex A

Docker Compose File for the Serving Layer

```
services:

  cratedb02:
    container_name: crateDB
    image: crate:latest
    ports:
      - "4202:4200"
      - "5431:5432"
    volumes:
      - ${LOCAL_DATA_PATH_02}:/data
    healthcheck:
      test: [ "CMD", "curl", "-f", "cratedb02:4200" ]
      interval: 15s
      timeout: 10s
      retries: 3
    deploy:
      replicas: 1
    restart: always
    environment:
      - CRATE_HEAP_SIZE=2g

  mqttbroker:
    image: eclipse-mosquitto
    hostname: mqttbroker
    container_name: mqttbroker
    restart: always
    ports:
```

```
    - "1883:1883"
    - "9001:9001"
  volumes:
    - ${LOCAL_MQTT_BROKER_CONFIG}:/mosquitto/config/mosquitto.conf
  healthcheck:
    test: [ "CMD", "mosquitto_sub", "-t", "$$SYS/#", "-C", "1",
       "-i", "healthcheck", "-W", "3" ]
    interval: 30s
    timeout: 10s
    retries: 5
    start_period: 20s

gps_mqtt:
  restart: always
  container_name: gps_mqtt
  environment:
    gateway_id: ${GATEWAY_ID}
  extra_hosts:
    - host.docker.internal:host-gateway
  image: registry.solvit.pt/ate/gps-mqtt
  depends_on:
    mqttbroker:
      condition: service_healthy

publisher:
  restart: always
  image: publisher
  container_name: publisher
  ports:
   - "5000:5000"
  build:
    context: ./
    dockerfile: Dockerfile
  env_file:
    - .env
  depends_on:
    cratedb02:
      condition: service_healthy
    mqttbroker:
        condition: service_healthy
```

Bibliography

[1] U. I. des Transports Publics, “Public transport benefits - mobility for (y)eu - benefits for all,” 2024. [Online]. Available: https://cms.uitp.org/wp/wp-content/uploads/2022/01/Public-Transport-Benefits-Mobility-for-YEU-Benefits-for-all.pdf

[2] A. P. T. Association, “Public transportation facts,” 2024. [Online]. Available: https://www.apta.com/news-publications/public-transportation-facts/

[3] B. Creations, “Public transport bus schedule optimisation,” 2024. [Online]. Available: https://medium.com/\spacefactor\@m{}comms%5F17126/public-transport-bus-schedule-optimisation-90acb47ff315

[4] A. Abuaisha and S. Abu-Eisheh, “Optimization of urban public transportation considering the modal fleet size: A case study from palestine,” *Sustainability*, vol. 15, no. 8, 2023.

[5] “Plano de recuperação e resiliência,” Accessed on: December 18, 2024. [Online]. Available: https://recuperarportugal.gov.pt/

[6] “Citymapper,” 2024, Accessed on: December 18, 2024. [Online]. Available: https://citymapper.com/

[7] “Google maps,” Accessed on: December 18, 2024. [Online]. Available: https://www.google.com/maps/

[8] “Centeris,” Accessed on: December 18, 2024. [Online]. Available: https://centeris.scika.org/

[9] I. Grgurević, K. Juršić, and V. Rajič, “Review of automatic passenger counting systems in public urban transport,” in *5th EAI International Conference on Management of Manufacturing Systems*. Cham: Springer International Publishing, 2022, pp. 1–15.

[10] M. Ferreira, C. Marte, J. De Medeiros, C. Sakurai, and C. Fontana, “Rfid for real time passenger monitoring,” in *Recent Researches in Telecommunications, Informatics, Electronics and Signal Processing*, 2013, pp. 170–175.

[11] D. Baskaran, M. Pattumuthu, B. Priyadharshini, P. Akram, and S. Sripriya, “Rfid based smart bus using embedded system,” in *International Journal of Engineering Research & Technology, COCODANTR*, 2016, pp. 1–7.

[12] I. G. Susrama Mas Diyasa, I. Yuniar Purbasari, A. Setiawan, and S. Winardi, “Smart passenger information system based on iot,” in *2019 TRON Symposium (TRONSHOW)*, 2019, pp. 1–5.

[13] C. Oberli and D. Landau, “Performance evaluation of uhf rfid technologies for real-time passenger tracking in intelligent public transportation systems,” in *2008 IEEE International Symposium on Wireless Communication Systems*, 2008, pp. 108–112.

[14] I. Pinna and B. Dalla Chiara, “Automatic passenger counting and vehicle load monitoring,” *Ingegneria Ferroviaria*, vol. 65, pp. 101–138, 02 2010.

[15] A. Kotz, D. Kittelson, and W. Northrop, “Novel vehicle mass-based automated passenger counter for transit applications,” *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2536, pp. 37–43, 08 2015.

[16] W. Northrop, “Development of a mass-based automated passenger counter,” 2019, report for transit IDEA project 84.

[17] H. Mohammadmoradi, S. Munir, O. Gnawali, and C. Shelton, “Measuring people-flow through doorways using easy-to-install ir array sensors,” in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2017, pp. 35–43.

[18] D. Bauer, M. Ray, and S. Seer, “Simple sensors used for measuring service times and counting pedestrians: Strengths and weaknesses,” *Transportation Research Record*, vol. 2214, no. 1, pp. 77–84, 2011.

[19] I. Amin, A. Taylor, F. Junejo, A. Al-Habaibeh, and R. Parkin, “Automated people-counting by using low-resolution infrared and visual cameras,” *Measurement*, vol. 41, no. 6, pp. 589–599, 2008.

[20] T. Myrvoll, J. Håkegård, T. Matsui, and F. Septier, “Counting public transport passenger using wifi signatures of mobile devices,” in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6.

[21] U. Mehmood, I. Moser, P. Jayaraman, and A. Banerjee, “Occupancy estimation using wifi: A case study for counting passengers on busses,” in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 165–170.

[22] E. Fenske, D. Brown, J. Martin, T. Mayberry, P. Ryan, and E. Rye, “Three years later: A study of mac address randomization in mobile devices and when it succeeds,” *Proceedings on Privacy Enhancing Technologies*, vol. 2021, pp. 164–181, 07 2021.

[23] M. Nitti, F. Pinna, L. Pintor, V. Pilloni, and B. Barabino, “iabacus: A wi-fi-based automatic bus passenger counting system,” *Energies*, vol. 13, no. 6, 2020.

[24] M. Uras, E. Ferrara, R. Cossu, A. Liotta, and L. Atzori, “Mac address de-randomization for wifi device counting: Combining temporal- and content-based fingerprints,” *Computer Networks*, vol. 218, p. 109393, 2022.

[25] M. Gain, “Public transportation surveillance system market research report provides thorough industry overview, which offers an in-depth analysis of product trends,” 2024. [Online]. Available: https://www.linkedin.com/pulse/public-transportation-surveillance-system-market-research-report/

[26] A. Soehnchen, “Video surveillance in public transportation,” 2024. [Online]. Available: https://www.masstransitmag.com/safety-security/article/12171666/video-surveillance-in-public-transportation

[27] B. C. Chee, M. Lazarescu, and T. Tan, “Detection and monitoring of passengers on a bus by video surveillance,” in *Proceedings of the 14th International Conference on Image Analysis and Processing*, ser. ICIAP ’07. USA: IEEE Computer Society, 2007, p. 143–148.

[28] C.-H. Chen, Y.-C. Chang, T.-Y. Chen, and D.-J. Wang, “People counting system for getting in/out of a bus based on video processing,” in *2008 Eighth International Conference on Intelligent Systems Design and Applications*, vol. 3, 2008, pp. 565–569.

[29] T. Yahiaoui, L. Khoudour, and C. Meurie, “Real-time passenger counting in buses using dense stereovision,” *Journal of Electronic Imaging*, vol. 19, no. 3, 2010.

[30] C. Slattery and Y. Shida, “Adi tof depth sensing technology: New and emerging applications in industrial, automotive markets, and more,” 2019. [Online]. Available: https://www.analog.com/en/analog-dialogue/articles/adi-tof-depth-sensing-technology-enabling-new-and-emerging-applications-beyond-consumer.html

[31] M. Stec, V. Herrmann, and B. Stabernack, “Using time-of-flight sensors for people counting applications,” in *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2019, pp. 59–64.

[32] M. Kleppmann, *Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O’Reilly Media, 2017. [Online]. Available: https://books.google.pt/books?id=BM7woQEACAAJ

[33] N. Niknejad, W. Ismail, I. Ghani, B. Nazari, M. Bahari, and A. R. B. C. Hussin, “Understanding service-oriented architecture (soa): A systematic literature review and directions for further investigation,” *Information Systems*, vol. 91, p. 101491, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437920300028

[34] A. D. Birrell and B. J. Nelson, “Implementing remote procedure calls,” *ACM Trans. Comput. Syst.*, vol. 2, no. 1, p. 39–59, 2 1984.

[35] K. Indrasiri and D. Kuruppu, *gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes*. O’Reilly Media, 2020. [Online]. Available: https://books.google.pt/books?id=883LDwAAQBAJ

[36] R. T. Fielding and R. N. Taylor, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, UNIVERSITY OF CALIFORNIA, IRVINE, 2000, aAI9980887.

[37] S. Malik and D.-H. Kim, “A comparison of restful vs. soap web services in actuator networks,” in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 753–755.

[38] J. Tihomirovs and J. Grabis, “Comparison of soap and rest based web services using software evaluation metrics,” *Information Technology and Management Science*, vol. 19, 12 2016.

[39] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The many faces of publish/subscribe,” *ACM Comput. Surv.*, vol. 35, no. 2, p. 114–131, 6 2003. [Online]. Available: https://doi.org/10.1145/857076.857078

[40] S. Celar, E. Mudnic, and Z. Seremet, *State-Of-The-Art of Messaging for Distributed Computing Systems*. DAAAM International Vienna, 01 2016, pp. 0298–0307.

[41] N. Suri, M. R. Breedy, K. M. Marcus, R. Fronteddu, E. Cramer, A. Morelli, L. Campioni, M. Provosty, C. Enders, M. Tortonesi, and J. Nilsson, “Experimental evaluation of group communications protocols for data dissemination at the tactical edge,” in *2019 International Conference on Military Communications and Information Systems (ICMCIS)*, 2019, pp. 1–8.

[42] “Apache kafka,” Accessed on: December 18, 2024. [Online]. Available: https://kafka.apache.org/

[43] “Zookeeper,” Accessed on: December 18, 2024. [Online]. Available: https://zookeeper.apache.org/

[44] “Redis,” Accessed on: December 18, 2024. [Online]. Available: https://redis.io/

[45] “Eclipse mosquitto,” Accessed on: December 18, 2024. [Online]. Available: https://mosquitto.org/

[46] “Hivemq,” Accessed on: December 18, 2024. [Online]. Available: https://www.hivemq.com/

[47] “Zeromq,” Accessed on: December 18, 2024. [Online]. Available: https://zeromq.org/

[48] “Rabbitmq,” Accessed on: December 18, 2024. [Online]. Available: https://rabbitmq.com/

[49] X. Liu, J. Han, Y. Zhong, C. Han, and X. He, “Implementing webgis on hadoop: A case study of improving small file i/o performance on hdfs,” in *2009 IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–8.

[50] A. S. Foundation, “Apache cassandra,” 2024. [Online]. Available: https://cassandra.apache.org/%5F/index.html

[51] “Mongodb,” Accessed on: December 18, 2024. [Online]. Available: https://www.mongodb.com/

[52] “Cratedb,” Accessed on: December 18, 2024. [Online]. Available: https://cratedb.com/

[53] J. Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, “Choosing the right nosql database for the job: a quality attribute evaluation,” *Journal of Big Data*, vol. 2, p. 18, 08 2015.

[54] G. Baruffa, M. Femminella, M. Pergolesi, and G. Reali, “Comparison of mongodb and cassandra databases for spectrum monitoring as-a-service,” *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 346–360, 2020.

[55] “Hikvision,” Accessed on: December 18, 2024. [Online]. Available: https://www.hikvision.com/

[56] “Xovis,” Accessed on: December 18, 2024. [Online]. Available: https://www.xovis.com/

[57] “Solvit,” Accessed on: December 18, 2024. [Online]. Available: https://solvit.pt/

[58] “Nexcom,” Accessed on: December 18, 2024. [Online]. Available: https://www.nexcom.com/

[59] “Docker,” Accessed on: December 18, 2024. [Online]. Available: https://www.docker.com/

[60] “Flask,” Accessed on: December 18, 2024. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/

[61] O. Isik, J. Hong, I. Petrunin, and A. Tsourdos, “Integrity analysis for gps-based navigation of uavs in urban environment,” *Robotics*, vol. 9, no. 3, 2020. [Online]. Available: https://www.mdpi.com/2218-6581/9/3/66