Gil Lopes Teixeira

# Aplicação Móvel para Sistemas Inteligentes de Transporte com Integração de Comunicações Veiculares

# Mobile Application for Intelligent Transportation Systems with Vehicular Communications Integration

**Universidade de Aveiro**
**2024**

**Gil Lopes Teixeira**

# Aplicação Móvel para Sistemas Inteligentes de Transporte com Integração de Comunicações Veiculares

# Mobile Application for Intelligent Transportation Systems with Vehicular Communications Integration

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor José Alberto Fonseca, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, do Doutor João Miguel Pereira de Almeida, Investigador do Instituto de Telecomunicações, e com a colaboração do Engenheiro Emanuel Sousa Vieira, Investigador do Instituto de Telecomunicações.

Este trabalho é financiado pela União Europeia / Next Generation EU, através do Programa de Recuperação e Resiliência (PRR) [Projeto n.º 29: Route 25]. Este trabalho foi também realizado no âmbito duma parceria com a A-to-Be - Mobility Technology.

Este trabalho é financiado por fundos nacionais através da FCT - Fundação para a Ciência e a Tecnologia, I.P., no âmbito do projeto 10.54499/UIDB/50008/2020, com o identificador DOI, `https://doi.org/10.54499/UIDB/50008/2020`.

Dedico este trabalho ao meus pais, Augusto Teixeira e Graça Lopes, que de forma incansável me suportaram durante o mestrado. Dedico-o também ao André Patacas que me introduziu ao projeto e me ajudou em aspetos críticos da implementação com as tecnologias usadas.

**o júri / the jury**

presidente / president

Prof. Doutor Ilídio Fernando de Castro Oliveira
professor auxiliar, Universidade de Aveiro

vogais / examiners committee

Doutora Ana Neves Vieira da Silva
engenheira, A-to-Be - Mobility Technology

Prof. Doutor José Alberto Gouveia Fonseca
professor associado aposentado, Universidade de Aveiro

**agradecimentos / acknowledgements**

Agradeço toda a ajuda, orientação e disponibilidade dos meus orientadores, José Alberto Fonseca e João Almeida. Agradeço também a dedicação e companheirismo dos meus colaboradores e amigos: Emanuel Vieira e David Rocha!

**Palavras Chave**

Sistemas Inteligentes de Transporte (ITS), Comunicações Veiculares, Interface Homem-Máquina, Aplicação Móvel, Expo, React Native

**Resumo**

O crescimento exponencial da população e, consequentemente, do número de utilizadores da estrada e veículos rodoviários tem tido um impacto significativo no congestionamento do tráfego e na segurança rodoviária. Os acidentes rodoviários são responsáveis pela morte de cerca de 1,2 milhões de pessoas em todo o mundo todos os anos, mas muitos poderiam ter sido evitados. Os Sistemas de Transporte Inteligentes (ITS) tentam mitigar estes problemas integrando uma vasta gama de dados de sensores e técnicas de processamento que fornecem informações em tempo real aos utilizadores das estradas e aos gestores de tráfego. As comunicações veiculares são um componente chave destes sistemas, pois permitem que os veículos comuniquem entre si e com a infraestrutura rodoviária para obter informações contextualmente relevantes. Esta dissertação apresenta o desenvolvimento de uma aplicação móvel multi-plataforma projetada para interagir com veículos ou com um servidor central de partilha de informação, facilitando a monitorização e a interação dentro do ambiente ITS. A aplicação integra recolha e partilha de dados de sensores e pode ser usada para criar e visualizar eventos que alertam os utilizadores sobre potenciais perigos, como acidentes, engarrafamentos, obras rodoviárias e condições climatéricas adversas em tempo-real, o que pode ajudar a prevenir acidentes e reduzir congestionamentos. As avaliações de usabilidade identificaram potenciais desafios de eficácia da aplicação entre as populações mais idosas e termos técnicos desconhecidos para utilizadores comuns, enquanto que as avaliações de desempenho revelaram problemas com o processamento de dados em tempo-real e escalabilidade, o que leva o dispositivo a ficar lento. Os conhecimentos obtidos através destas avaliações informaram melhorias na aplicação e forneceram orientação para trabalhos futuros, contribuindo para os esforços contínuos para melhorar a segurança rodoviária e aliviar o congestionamento através do avanço de sistemas de transporte inteligentes e tecnologias de comunicação veicular.

**Keywords**

Intelligent Transportation Systems (ITS), Vehicular Communications, Human-Machine Interface, Mobile Application, Expo, React Native

**Abstract**

The exponential growth of the population and, consequently, of the amount of road users and vehicles has significantly impacted traffic congestion and road safety. Road accidents are responsible for the death of around 1.2 million people worldwide every year, but many could have been avoided. Intelligent Transportation Systems attempt to mitigate these problems by integrating a vast array of sensor data collection and processing techniques that provide real-time information to road users and traffic managers. Vehicular communications are a key component of these systems, as they allow vehicles to communicate with each other and with the road infrastructure to obtain contextually relevant information. This dissertation presents the development of a multi-platform mobile application designed to interface with either vehicles or a central information broker, facilitating monitoring and interaction within the ITS environment. The application integrates sensor data collection and sharing and can be used to create and visualize events that warn users of potential hazards, such as accidents, traffic jams, road works and adverse weather conditions in real-time, which can help prevent accidents and reduce congestion. Usability evaluations identified potential effectiveness challenges among older populations and unfamiliar technical terms for regular users, while performance evaluations revealed issues with real-time data processing and scalability, leading to device throttling. Insights from these evaluations informed improvements to the application and provided direction for future work, contributing to ongoing efforts to enhance road safety and alleviate congestion through the advancement of intelligent transportation systems and vehicular communication technologies.

# Contents

# List of Figures

# List of Tables

# Glossário

| | |
|---|---|
| **IoT** | Internet of Things |
| **EV** | Electric Vehicle |
| **ETSI** | European Telecommunications Standards Institute |
| **OSI** | Open Systems Interconnection |
| **OS** | Operating System |
| **VRU** | Vulnerable Road User |
| **TCC** | Traffic Control Center |
| **OBU** | On-Board Unit |
| **OBD-II** | On-Board-Diagnostics II |
| **RSU** | Roadside Unit |
| **JIT** | Just In Time |
| **C-ITS** | Cooperative Intelligent Transport System (ITS) |
| **C-V2X** | Cellular Vehicle-To-Everything (V2X) |
| **ITS** | Intelligent Transport System |
| **ITS-S** | ITS-Station |
| **CAM** | Cooperative Awareness Message |
| **DENM** | Decentralized Environmental Notification Message |
| **CPM** | Collective Perception Message |
| **VAM** | Vulnerable Road User (VRU) Awareness Message |
| **HD-Map** | High-Definition Map |
| **IVIM** | Infrastructure to Vehicle Information Message |
| **SAEM** | Services Announcement Essential Message |
| **SA** | Services Announcement Essential Message |
| **SAE** | Society of Automotive Engineers |
| **SPVSM** | SmartPhone's Vehicular Sensor Message |
| **SNVSM** | SoNar's Vehicular Sensor Message |
| **OVSM** | OBD-II's Vehicular Sensor Message |
| **VSM** | Vehicular Sensor Message |
| **TPM** | Tolling Payment Message |
| **EVCSNM** | Electric Vehicle Charging Spot Notification Message |
| **EVRSRM** | Electric Vehicle (EV) Recharging Spot Reservation Message |
| **V2X** | Vehicle-To-Everything |
| **V2I** | Vehicle-To-Infrastructure |
| **V2V** | Vehicle-To-Vehicle |
| **V2N** | Vehicle-To-Network |
| **V2B** | Vehicle-To-Business |
| **V2P** | Vehicle-To-Pedestrian |
| **V2H** | Vehicle-To-Home |
| **V2G** | Vehicle-To-Grid |
| **V2C** | Vehicle-To-Cloud |
| **I2C** | Infrastructure-To-Cloud |
| **P2C** | Pedestrian-To-Cloud |
| **JS** | JavaScript |
| **JSON** | JavaScript Object Notation |
| **USB** | Universal Serial Bus |
| **WS** | WebSockets |
| **HTTP** | Hypertext Transfer Protocol |
| **CoAP** | Constrained Application Protocol |
| **AMQP** | Advanced Message Queuing Protocol |
| **TS** | TypeScript |
| **RN** | React Native |
| **EAS** | Expo Application Services |
| **UI** | User Interface |
| | Pull Request |
| **SDK** | Software Development Kit |
| **API** | Application Program Interface |
| **MQTT** | Message Queuing Telemetry Transport |
| **URL** | Uniform Resource Locator |
| **ITS-G5** | ITS-G5 |
| **LTE** | Long Term Evolution |
| **MIB** | Management Information Base |
| **CAN** | Controller Area Network |
| **TPI** | Tolling Payment Information |
| **RPM** | Revolutions Per Minute |
| **SVG** | Scalable Vector Graphics |
| **PDF** | Portable Document Format |
| **RAM** | Random-Access Memory |
| **NPM** | Node Package Manager |
| **IQR** | Interquartile Range |

| | |
|---|---|
| **ADB** | Android Debug Bridge |
| **FPS** | Frames Per Second |
| **CPU** | Central Processing Unit |
| **WHO** | World Health Organization |
| **GNSS** | Global Navigation Satellite Systems |
| **GPS** | Global Positioning System |
| **CLI** | Command-Line Interface |
| **DSRC** | Dedicated Short-Range Communications |
| **CONCORDA** | Connected Corridor for Driving |
| **InterCor** | Interoperable Corridors |
| **C-MoBILE** | Accelerating C-ITS Mobility Innovation and depLoyment in Europe Automation |
| **HMI** | Human-Machine Interface |
| **OER** | Octet Encoding Rules |
| **ABS** | Anti-lock Braking System |

CHAPTER 1

# Introduction

*This chapter introduces the context and motivation of this dissertation by analyzing the topic from a social and technological standpoint. Then, it presents the objectives of the work and the dissertation structure.*

## 1.1 Context

Vehicle usage is consistently growing both in the European Union [1] and worldwide [2], following the same trend as the world population size [3]. Both are expected to keep rising, even in low projections, thus directly contributing to an increase in road congestion and number of accidents. Even though vehicle manufacturers have started to include (mostly proprietary) safety features, the World Health Organization (WHO) estimates 1.3 million deaths per year due to road traffic crashes with more than half of these number being accounted by Vulnerable Road User [4], such as pedestrians, cyclists and other two-wheelers. The consequences of road accidents cost most countries 3% of their gross domestic product. It is important to note that 93% of fatalities occur in low- and middle-income countries, even though their vehicles only account for 60% of the number of vehicles worldwide. These are worrying statistics that can be curbed in ITS environments.

ITSs encompass many technologies and applications whose goal is to improve the safety and efficiency of road users. An ITS has a wide range of sensors and data processing techniques to monitor and better manage traffic. Cooperative ITSs (C-ITSs) are a subset of ITSs that refers specifically to vehicular communications. Vehicles communicate among them, with the roadside infrastructure and other ITS applications. C-ITSs aim to have vehicles and infrastructure exchanging real-time information in order to cooperate in coordinated maneuvers situations, collaborate through the transmission of sensor information, and communicate, e.g. hazardous road and weather conditions to drivers. These alerts can assist collision prevention mechanisms, for instance in tight road turns where an heavy vehicle might occupy part of the opposite way lane. The exchanged messages can also disseminate the presence of emergency vehicles in march, general textual information or speed limits. Vehicular communications

can also be used for cooperative platooning applications, which involves a group of vehicles traveling closely autonomously or semi-autonomously.

## 1.2 Motivation

Standardized C-ITS environments keep growing which also implies a necessity for ways to disseminate the information to road users and enable their active participation. Road users are either drivers of vehicles containing On-Board Units (OBUs) or VRUs, having these two groups distinct goals and use cases.

Drivers can connect to the OBU of their vehicle in order to monitor information specific to it, including vehicle sensors, and from surrounding ITS-Stations (ITS-Ss). Drivers can also request and use V2X services. These respond to common drivers use cases, such as tolling and EV charging.

VRUs are the road users in most danger. They represent over half of road traffic fatalities and are also the least connected to the existing ITS environment. Having VRUs directly advertising their presence on the road can be very useful to prevent dangerous and/or fatal situations. The concept, importance and ways of increasing the VRU perception rate are explored in [5] where a comparison is made, using a virtual environment, between the perception rate accomplished using: no V2X communications; Collective Perception Messages (CPMs) shared by vehicles and road infrastructure; VRUs directly sharing their position with VRU Awareness Messages (VAMs); and finally with both awareness messages. It was concluded that V2X communications considerably increase the perception rate (figure 1.1).

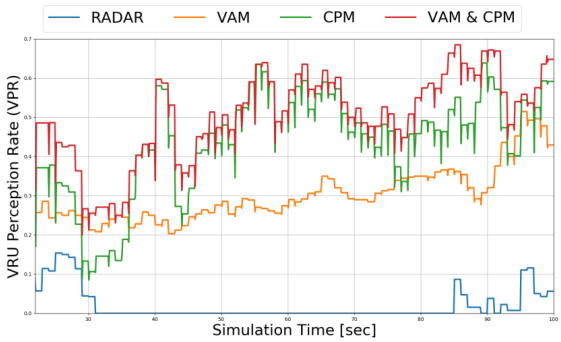


**Figure 1.1:** VRU perception rate.

In [6], a study is performed on the perception rate of V2X-enabled vehicles when communicating among themselves and how the impact of the delay associated with the perception process might make it impossible for the vehicle to execute an emergency brake. Hence, a big emphasis is made on very low latency communication technologies and protocols. The most effective way of increasing this perception rate is by having the VRU to directly announcing its location. A mobile application is a great way of achieving this goal, since people usually carry their smartphones.

Traffic management operators can also monitor and manage the traffic system by creating vehicular communications warnings that are disseminated to alert road users regarding various

types of traffic and weather events. These events can be created and visualized using any C-ITS application that adopts the standard and can help users better react to dangerous situations on the road.

These features can, thanks to the underlying standardized technologies, systems and services, all be integrated into a single application with the potential to greatly increase user retention. It also promotes competition between implementations which will inevitably lead to functional and useful mobile applications, thus improving road safety and efficiency.

## 1.3 Objectives

The primary goal of this dissertation is to develop a mobile phone application that enables road users to directly interact with the C-ITS environment. This application should:

- be able to connect to both a cloud server, using Wi-Fi, Long Term Evolution (LTE)/5G, and to the vehicle's OBU, via WiFi, in order to send and transmit V2X messages;
- be able to display, in a concise way, the information regarding road events, Roadside Units (RSUs), vehicles containing OBUs, their perceived objects, and other connected VRUs using a C-ITS application.
- advertise the location of VRUs;
- integrate advanced V2X services for drivers, such as EV charging and tolling;
- display and emit sensor messages.

## 1.4 Structure of the Dissertation

The dissertation is organized into the following chapters:

- **Chapter 2 - Fundamental Concepts and State-of-the-Art**: introduces technologies, techniques and algorithms used in this work and an analysis of the current mobile applications related to the one proposed in this dissertation;
- **Chapter 3 - Architecture**: defines the architecture of the C-ITS framework, the application user modes, the mobile application and its V2X services, sensors and interfaces;
- **Chapter 4 - Implementation**: describes the implementation details of all components, screens and application logic utilities;
- **Chapter 5 - Evaluation**: is divided into two sections: usability and performance evaluations. Both an heuristic and an empiric methods were used to evaluate usability. Performance was measured using Android Debug Bridge (ADB) and a network traffic analysis;
- **Chapter 6 - Conclusion**: gives an overview of the contributions and accomplished goals of this work. It also proposes possible ways to improve usability, performance and the utility of the application.

CHAPTER 2

# Fundamental Concepts and State-of-the-Art

This chapter describes the fundamental concepts of Cooperative ITS, which are relevant to the work developed in this dissertation. The first section presents the C-ITS environment and how the cooperation of ITSs can improve road safety and efficiency. The next section describes concepts related to vehicular communications and V2X services. The following section explains Message Queuing Telemetry Transport (MQTT) and how the topic structure can be used, with a quadtree, to filter the information based on the user location and the chosen quadtree zoom. The fourth section is dedicated to mobile application development and an exploration of technologies available, with a big emphasis on React Native (RN) and Expo. The last section covers the state-of-the-art, in which multiple application with similar features to the one developed in this dissertation are listed and compared.

## 2.1 Cooperative Intelligent Transportation Systems

C-ITS is a wide, and continuously evolving, combination of information and communication technologies. These are mainly focused on providing intelligence to vehicles, using the roadside infrastructure and other road users to collect and share sensor data, as well as other agents' intentions. C-ITS are focused on Vehicle-To-Everything communications that provide real-time, very low latency, ITS data and services to road users through Vehicle-To-Vehicle (V2V), Vehicle-To-Infrastructure (V2I), Vehicle-To-Network (V2N) and Vehicle-To-Pedestrian (V2P) communications.

ITS-Ss collect and share sensor information with each other using V2X messages. These messages are then used to detect the presence of road users (either vehicles or VRU), traffic incidents or other type of events on the road. In the future, V2X messages can also be employed to negotiate maneuvers, mainly involving autonomous vehicles. Cooperative awareness and collective perception are basic concepts in a C-ITS environment, through which the ITS-Ss collaborate to detect other road users, including non-connected legacy vehicles.

These concepts are of extreme importance to ITS-Ss and there are a lot of studies analyzing the requirements needed for an effective awareness of vehicles and VRUs [7]. These studies tend to use analytical models and simulations, with data collected from real test sites, to validate the awareness and evaluate it utilizing performance metrics, [8]. For instance, in [9] three metrics are used:

- neighborhood awareness ratio;
- ratio of neighbors above range, e.g. vehicles on a bridge that goes over another road;
- packet delivery rate.

The authors conclude that higher transmission power and lower message rate yields less unwanted messages/errors. A balance between awareness and interference must be managed, by having the specific context of the involved stations in mind. These works emphasize the benefits of environments where the different types of C-ITS messages, described in the next section, are used in conjunction to improve cooperative awareness.

An example of a research project that attempts to achieve these low latency communications, with enough time to avoid dangerous situations, is 5G-SAFE-PLUS [10]. This project integrates different communications technologies (5G, LTE, ITS-G5 and satellite) in the road infrastructure and vehicles to enable very low latency data exchanges among road users and with cloud-based services. It supports widespread deployments and advanced road and weather safety services. Other works focus on Cellular V2X (C-V2X) technology [11] in order to reliably communicate with very low latency, resulting in a breakthrough solution that can greatly improve traffic safety. There are, however, many places in which the cellular and Global Navigation Satellite Systems (GNSS) coverage can be very limited, such as tunnels, that often have many blind spots. The work in [12] points out that this could have a direct impact on the operation of the tunnel itself, affecting the safety and fluidity of traffic, showing C-ITS must be investigated under more demanding conditions.

## 2.2 Vehicular Communications

The cooperation in C-ITS environments is made possible thanks to vehicular communications. The information is exchanged using standardized messages, like Cooperative Awareness Messages (CAMs), CPMs and Decentralized Environmental Notification Messages (DENMs), as it will later described in subsection 2.2.2. These communications among vehicles and with other ITS-S can be either achieved via localized short-range communications or over long-range cellular networks. Typically, in the short-range case, the ITS-Ss communicate with surrounding stations using ITS-G5 (ITS-G5) or C-V2X.

### 2.2.1 ETSI ITS-Station

In a C-ITS environment, all elements that collect and/or share information, such as roadside infrastructure, connected vehicles with OBUs and C-ITS central platforms, are denominated ITS-Ss. In Europe, an ITS-S is defined by European Telecommunications Standards Institute (ETSI) via a reference architecture based on the Open Systems Interconnection (OSI) model. This model is expanded to also describe the access technologies and protocols used for ITS

services. The architecture is comprised of 4 horizontal and 2 vertical layers [13] [14] [15], as depicted in figure 2.1.

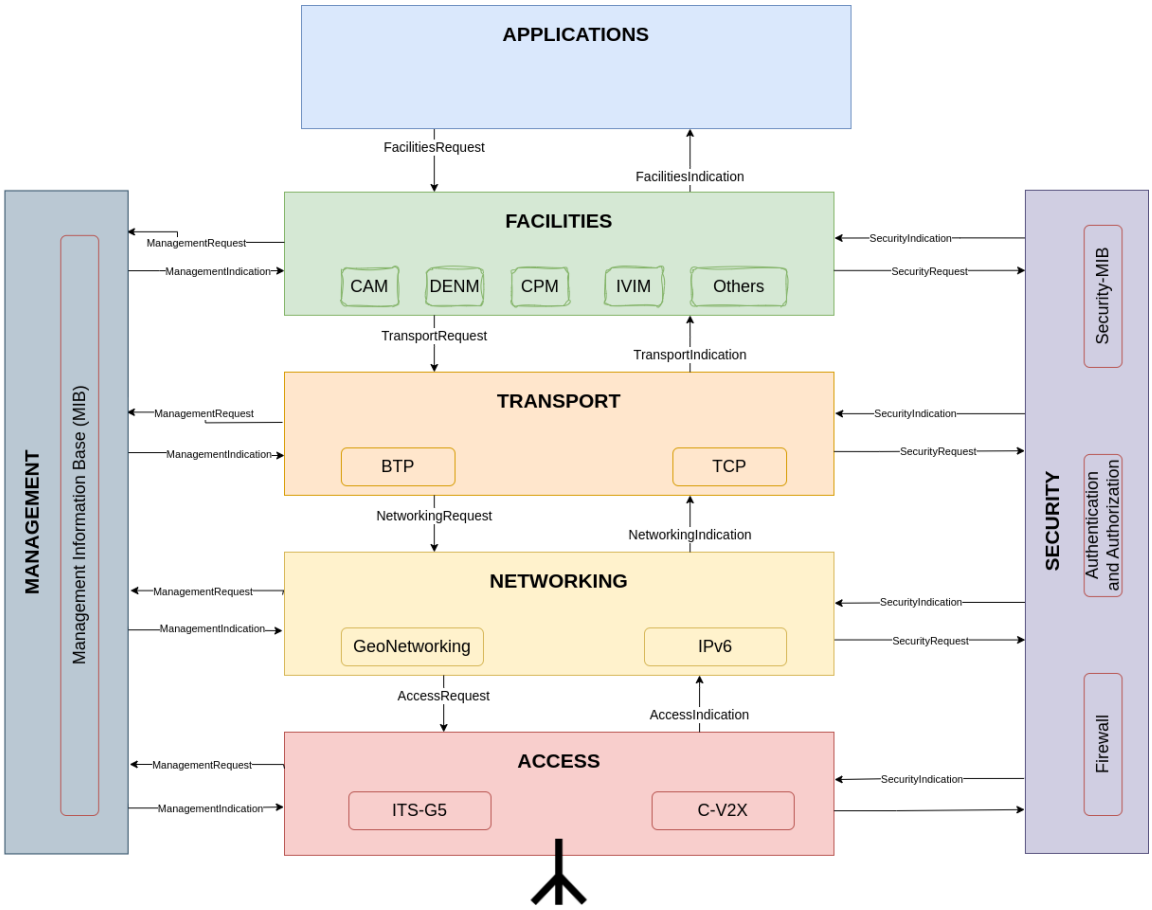


**Figure 2.1:** Reference architecture of an ETSI ITS-S.

This architecture relates to the OSI model as follows:

- The **Access Layer** represents the first and second layers of the OSI model, including both the physical and data link (medium access control and logical link control) features of the communications stack. It specifies the protocols, interfaces and types of media responsible for both internal and external communications. The external communications can be non-ITS specific, like LTE or ITS specific, as for instance ITS-G5;
- The **Networking** and **Transport Layers** represent the third and fourth layer, respectively. The networking layer describes the networking protocols, the addressing, data routing and geographical-based data dissemination. The transport layer specifies protocols for end-to-end encrypted, reliable, data exchanges and also other features such as congestion avoidance;
- The **Facilities Layer** corresponds to layers 5, 6 and 7. It describes access to common functionalities and ITS services with shared data/memory, being also responsible for creating, processing and handling C-ITS messages such as CAMs, DENMs, CPMs, Electric Vehicle Charging Spot Notification Messages (EVCSNMs) and others;

- The **Applications Layer** is responsible for advertisingITS services and it may function as an Human-Machine Interface (HMI) to access the data and functionalities from the Facilities Layer.
- The **Management Layer** is responsible for the configuration of the ITS-S, cross-layer functionalities and access to the Management Information Base (MIB), which contains important data sets and variables. These may include status and performance of traffic lights, sensors and other devices;
- The **Security Layer** contains the definition of many security and privacy functionalities that are commonly used throughout the horizontal layers in security protocols, being also responsible for handling the credentials used in those protocols. Some examples of the functionalities provided are:
  - Key management;
  - Firewalls;
  - Encryption and decryption;
  - Authentication.

### 2.2.2 ETSI C-ITS Message Types

There are several messages used in C-ITS communications defined by ETSI. Each has its own purpose:

- **Cooperative Awareness Message (CAM)** - Messages generated by ITS-Ss that are used to disseminate its location, dynamics, and type of the station [16]. In the case of vehicles, it contains information like:
  - Length;
  - Location;
  - Speed;
  - Longitudinal acceleration;
  - Heading;
  - Yaw rate;
  - Curvature;
  - Traces of past locations (optional);
  - State of exterior lights (optional);
  - Lane position (optional);
  - Steering wheel angle (optional);
  - Lateral and vertical acceleration (optional).

  The roadside infrastructure (RSUs) starts emitting CAMs when other ITS stations are in range and, if there are Dedicated Short-Range Communications (DSRC) tolling plazas nearby, it advertises the protected communication zones.
- **Collective Perception Message (CPM)** - Produced by any ITS-S that detects, with enough confidence, surrounding objects using its own sensors [17]. It shares the location of the ITS-S and a list of perceived objects detected. For each perceived object

it includes the confidence, the x and y distances to the originating station and also the x and y speed values. Optionally the station can also include a classification of objects as either a vehicle, human or animal.

- **Decentralized Environmental Notification Message (DENM)** - Messages that convey information about road events such as traffic congestion, abnormal weather conditions and others [18]. DENMs can be sent by a central manager or road user, using a C-ITS application, but may also be automatically generated by vehicles based on the information provided by their own sensors.
- **Infrastructure to Vehicle Information Message (Infrastructure to Vehicle Information Message (IVIM))** - A message shared by the infrastructure to vehicles[19]. It may contain road signage, speed limit and/or textual information, a geographic location and, optionally, a heading and a validity interval.
- **Services Announcement Essential Message (Services Announcement Essential Message (SAEM))** - Messages sent by the Services Announcement Essential Message (SA) service of an ITS-Ss [20]. It contains information about the ITS services a given station provides.
- **VRU Awareness Message (VAM)** - This type of messages are emitted by devices used by VRUs, such as pedestrians and cyclists, to directly share the presence and location of VRUs [21].

### 2.2.3 V2X EV Charging

The market share of EVs and hybrid plug-in vehicles has been growing exponentially [22], and these vehicles are expected to ultimately dominate the road vehicle market. Consequently, the demand for EV charging services has also increased. ETSI proposes, in [23] and [24], a standardized way to disseminate EV charging station locations and type as well as the communications procedure to create, update and cancel a charging reservation.

RSUs forward to passing vehicles the location and types of available charging stations, using EVCSNM. These can be displayed in a OBU-connected mobile application or directly in the vehicle's dashboard, enabling the user to select a station on the map and choose among the available types of charging.

Using a V2X application, connected to the EV, the user is able to schedule the charging of the vehicle by choosing the type of charging, desired energy amount and the start and end dates. This information is then sent to the charging station via an EV Recharging Spot Reservation Message (EVRSRM). The station will confirm the viability of the reservation details and then replies to the request. If the reply is successful, the payment information is sent in a new request, and the new reply will determine the reservation success or the error cause. An update allows the user to postpone the start date and/or prepone the end date. EVRSRMs can also be used to cancel existing reservations.

### 2.2.4 V2X Tolling

Tolling is often a part of drivers' daily activities. ETSI has proposed, in a pre-standardization study [25], a V2X solution to support the most common tolling use cases, a

proof of concept and possible evolution using C-ITS. It specifies how an ITS-S would need to be adapted and the performance considerations required for a reliable system.

Tolling can be advertised as an ITS service inside a SAEM. This type of message contains a **SAMapplicationData** field that can be used to encode information about the service, in this case the location and shape of the tolling zones.

Based on the pre-standardization study, Emanuel Vieira *et al.* [26] designed a new tolling system with custom tolling messages, named Tolling Payment Messages (TPMs). These messages are exchanged between the vehicle and an RSU and contain information such as the status of the payment operation and, when successful, the price paid. When the operation fails, the user is notified about what went wrong.

## 2.3 Cellular-based V2X Communications

In cellular-based V2X communications, relying on cloud and/or edge infrastructure, there is need for a mediator to enable the exchange of information among ITS-Ss. This mediator is responsible for managing a variable set of (one or more) communication protocols, including REST and/or Messaging Protocols such as:

- **Message Queuing Telemetry Transport (MQTT)** based on the publish-subscribe model [27];
- **Constrained Application Protocol (CoAP)** establishing client-server interactions [28];
- **Advanced Message Queuing Protocol (AMQP)** providing a message-oriented protocol [29];
- **WebSockets (WebSockets (WS))** based on the Data Streaming paradigm [30];
- Other options like **Hypertext Transfer Protocol (HTTP)**, etc.

There is still no standard protocol or set of protocols defined for utilization in this type of scenarios. However, functioning examples were implemented in the scope of various research projects like Connected Corridor for Driving (CONCORDA) [31], Interoperable Corridors (InterCor) [32] and Accelerating C-ITS Mobility Innovation and depLoyment in Europe Automation (C-MoBILE) [33]. These use a reference architecture with messages published via MQTT, on topics specifying the geographic location of the ITS-S by using a quadtree tiling system.

### 2.3.1 MQTT Publish-Subscribe Model

MQTT is a lightweight and efficient messaging protocol that is used in several Internet of Things (IoT) applications. The protocol is used to communicate with MQTT brokers. Most implementations of MQTT brokers, like Mosquitto [34], HiveMQ [35] and RabbitMQ [36] support connections using both the MQTT protocol and WS. WSs provide an easy way for web clients to connect to the broker. A message sent to the broker is published in a topic and is then relayed to all consumers subscribed to that topic.

A topic is a string comprised of a *keyword* followed by zero or more *keywords*. The topics used by the reference architectures of previously mentioned research project usually follow the format *stationID/messageType/quadtree*, e.g.:

- 1/DENM/1.
- 2/DENM/2/2;

A client can subscribe to both stations messages by using a single-level wildcard (+). A multi-level wildcard (#) is used to subscribe to all messages that match the topic after it:

- +/DENM/2/#

This topic structure allows for the consumer to subscribe to messages:

- from all stations, using a single-level wildcard, or any single desired station;
- all types of messages or a specific type, useful to subscribe only to topics with messages of interest;
- everywhere or inside a given subspace of the map, called a tile, using a quadtree.

### 2.3.2 Quadtree Tiling System

A quadtree is a tree data structure that recursively subdivides a given plane into four quadrants. The amount of recursive subdivisions/depth of the tree is chosen according to the desired level of detail. This tree structure is then encoded into a string (quadkey) using the chosen quadrants, in order, separated by a slash character (/).

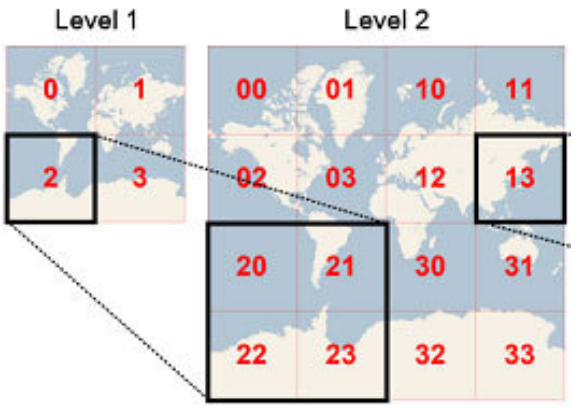


**Figure 2.2:** Quadtree tiling example[37].

When a consumer subscribes to the topic in the prior example, it won't receive any messages sent inside quadrant 1, since it's only subscribed to messages inside the subspace *2/#*. The depth of the quadtree is also called zoom level. In this example the zoom level is 1 since there is only one subdivision of the space.

## 2.4 Mobile Application Development

Currently 99.35% of smartphones are either android (72.37%) or iOS (26.98%) based, so these are both very important platforms to target, as stated in [38]. In order to achieve the best performance, the application should be coded separately for each platform: one code base with Swift and/or Objective-C for iOS phones and another with Kotlin and Java for Android.

There are many alternatives that allow for a single code base solution to work on both platforms. The most attractive and reliable are RNs from Facebook and Flutter and Dart from Google. Flutter requires developers to learn a specific new programming language Dart,

the User Interface (UI) is created using a combination of pre-designed and custom widgets and it is possible to write and integrate platform-specific native code. RN allows for most of the development to be done in TypeScript (TS)/JavaScript (JS) using classes or components, being also able to incorporate native code and libraries.

Expo is a very popular, open-source, Software Development Kit (SDK) developed on top of RN for making universal native apps using only TS/JS, thus eliminating native code completely from the code base. Expo distributes an application, called Expo Go, via the App Stores that can be used during development to connect to the computer's server, bundle and run the application. It includes features such as:

- Fast refresh that detects when the source code is modified, re-bundling the application and reloading it;
- Performance monitor to detect potential excessive re-renders and memory leaks;
- Element inspector similar to the one available in the dev-tools of most browsers;
- Easy way to tunnel the server in order to have access from anywhere.

The Expo SDK also features:

- Multiple Application Program Interfaces (APIs) that provide lots of native implementations of common device and system functionalities and components. Each API has an extensive documentation, with TS examples;
- Free, but limited, server builds and integrated deployment in both platforms;
- Many ways to create and sign the builds locally:
  - Using the Expo Application Services (EAS) Command-Line Interface (CLI) with the option –**local**;
  - With the Expo CLI and the **prebuild** command, which creates the Android and iOS RN folders, followed by running the native build command, gradlew and gradle on Android and xcode on iOS.
- A way to eject the application toRN, in case there's a need to use functionalities not available in the SDK;
- Config-plugins that allow the integration of native code libraries that aren't part of the SDK, without the need of ejecting the Expo application. These require however, a custom version of Expo Go for the development phase, that must be built including the new dependencies.

A V2X mobile application is able to access a lot of data about the vehicle and its surroundings and can also be used to interact with V2X services. The prospects and impact of the emerging technologies in mobile application development and network standards are further analyzed in [39]. This work emphasizes the slowness in the general adoption of these technologies in the automotive industry when compared to the mobile phone industry. The authors foresee that the faster adoption cycles of the mobile phone industry might provide faster adoption paths in the automotive industry.

## 2.5 State-of-the-Art

The most used mobile applications for mapping and navigation, like **Google Maps**, **Waze** and **Traffic Spotter** have began to include ways to communicate road and weather events.

These platforms, however, aren't integrated into a C-ITS environment and because of that they lack most of the features, services and connectivity that a C-ITS application can provide. These applications don't directly share information with each other, which severely limits the real-time dissemination of information to all road users.

There aren't a lot of C-ITS applications available in the Android and iOS stores. These are mostly V2X applications focused on drivers and use their own custom C-ITS environments, which also limits the dissemination of information. Unifying these environments is an essential aspect in order to ensure interoperability and it wouldn't be too hard to achieve, since the exchanged V2X messages are standardized.

**Kapsch** is a company focused on developing a modern digital traffic infrastructure, including RSUs and OBUs, and a suite of C-ITS mobile apps, for both Android and iOS [40]. This suite contains both V2X and non-V2X applications. Kapsch V2X Assist [41], the V2X application, is able to display vehicle sensor data and provide graphical and voice alerts based on road and weather warnings sent by other users via DENMs. The application has support for speed limit IVIMs, being also able to present the time interval until traffic signals change state. Kapsch has also published a white paper in cooperation with Ford and the Central Texas Region Mobility Authority, [42], where they did a demonstration and tests incorporating V2X tolling although, currently, it's not yet available on their application.

**V2X-A3** [43], available for Android and iOS, is an application that allows a user to connect to its own vehicle using a custom On-Board-Diagnostics II (OBD-II) reader, named V2X-2 smart device. The application includes Global Positioning System (GPS) tracking, vehicle diagnostics, speed limit recommendation and journey history reports with distance, time, driving route, fuel consumption, fuel cost and speed analysis. The application is designed for personal use and doesn't include information about other road users at all.

**C-ITS** [44], by Inzinious, is an Android-only ITS mobile application that features road and weather events, information about traffic in intersections and the time needed for traffic lights to change state. It also mentions fee collection in what looks like a tolling use case. There is no explicit mention to V2X communications, so these use cases are probably implemented using other technological solutions provided by the company. The application is only accessible using old Android versions as there are no updates since 2018.

**Uproad** [45] from Kapsch Group, and **Slora** [46] by Globalvia, are mobile applications, available on both Android and iOS devices, used to pay and manage tolling services. These applications are not related to V2X environments. The initial setup is very simple, requiring only the license plate of the vehicle and a valid payment method. By using the application, users can skip cash lanes, get toll price alerts, and plan a route with trip cost estimates.

**ev.energy** [47] is a mobile application, available on both Android and iOS, that connects to electric vehicles and allows one to have detailed statistics and plan the EV charging. The application also allows bidirectional Vehicle-To-Grid (V2G), Vehicle-To-Home (V2H) and Vehicle-To-Business (V2B) charging. This allows the vehicle owner to use the vehicle's battery, for example, as a power source of a house. It tries to help the user planning when the vehicle must be charged for the least cost, since in a lot of cases the electricity cost is lower during

certain periods of the day. **miio** [48] is an Android and iOS based application created to help EV drivers find charging stations and reserve them, operating in Portugal, Spain and France. The application also displays many statistics about the reservation, including a live feed of the current battery charge, reservation and charging costs comparison and it also allows users to rate each charging station.

A comparison of features and services supported by these different mobile phone applications is provided in table 2.1.

| Mobile Application | Road and Weather Events | Tolling | EV charging | OBD-II | V2X |
|---|---|---|---|---|---|
| Google Maps | Yes | No | No | No | No |
| Traffic Spotter | Yes | No | No | No | No |
| Kapsch | Yes | No | No | No | Yes |
| V2X-A3 | No | No | No | Yes | No |
| Uproad | No | Yes | No | No | No |
| Slora | No | Yes | No | No | No |
| C-ITS | Yes | Yes | No | Yes | No |
| ev.energy | No | No | Yes | No | No |
| miio | No | No | Yes | No | No |

**Table 2.1:** Comparison of services provided by the mobility-related mobile applications.

The following works are not commercial C-ITS applications, but consist of solutions developed in the scope of academic or research projects. An example of a V2X application can be found in [49], which presents some interesting features like speed limit recommendation as well as the use of the phone's camera, for road conditions recognition, using image processing, and other sensors data, including the luminosity sensor, to help determine meteorologic conditions and even, in case of a crash, record the situation. This application works only on Android and the supported use cases are mostly focused on driver assistance. It also ended up not being integrated with a vehicle's OBU to accomplish its objectives, being only tested using Google's Firebase platform.

Another example is provided by A-to-Be V2X mobile application, developed in the context of 5G-MOBIX project [50], which is solely available for Android and can only communicate with vehicles' OBUs using a local MQTT connection. This link is used to receive ITS messages, being also able to send DENM alerts specified by the users. The most notable issue, aside from not working on 15 to 25 percent of mobile phones (iOS), is that the user interface is not very user friendly. When the user opens the application, it tries to connect to a MQTT broker using a default host and port, which would only function when connected to an OBU's WiFi hotspot. This application is really only usable by people who already know what the application does and how it operates, which makes the entry bar for new users very high. As a conclusion, the application usability needs to be evaluated and improved.

To access the settings one must open the side bar using a button with similar colors to the background, light gray and light blue. The scrollable tab has a header and a footer which occupy around 2/3 of the view height and the settings button appears third on the list. It

features a simulation option to record and then repeat what happened during a recording session. In this same tab it has an event history and High-Definition Maps (HD-Maps) history, for some reason separated, and finally a logs section. An example of the configuration tab screen is presented in figure 2.3.

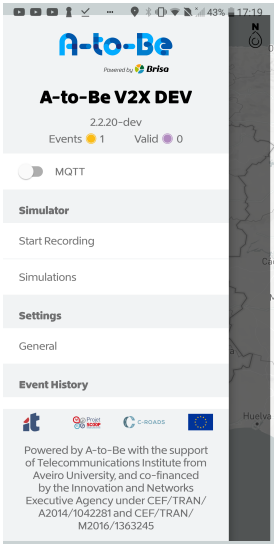


**Figure 2.3:** A-to-Be V2X App's configuration tab.

Furthermore there's no real emphasis on VRUs, since the application must be connected to an OBU MQTT broker to function. It is only useful for drivers who have a vehicle with an OBU which corresponds to a very low percentage of potential road users.

These works heavily influenced both the UI and the architectural choices made in the development of this work. As a conclusion, the application needs a minimalist UI and to support different user modes targeting the distinct types of road users: drivers, VRU and traffic management operators. These modes will be described in the next chapter, being created to make the application useful to as many end users as possible.

CHAPTER 3

# System Architecture

This chapter starts by describing the C-ITS architecture of this dissertation and then presents the mobile application user modes, architecture and supported C-ITS services. The final section explores the smartphone sensors and interfaces.

## 3.1 C-ITS Architecture

The developed C-ITS architecture focus on the role played by vehicular communications. There are multiple ways for a vehicle to interact with the surrounding environment, being it the roadside infrastructure, other vehicles or VRUs. The OBU integrated in the connected vehicles is responsible for these V2X communications, which are illustrated in figure 3.1.

The infrastructure can communicate with the cloud (Infrastructure-To-Cloud (I2C)) and nearby vehicles (V2I), while the vehicles can communicate with the cloud (Vehicle-To-Cloud (V2C)), nearby vehicles (V2V) and nearby roadside infrastructure (V2I). The mobile application may be utilized by VRUs, which use the V2X application to communicate with the cloud via Pedestrian-To-Cloud (P2C). The app can also be used by drivers, by first connecting the smartphone to the WiFi network hosted by the OBU of the vehicle and then to its local MQTT broker, in order to receive and send information.

The OBU in a vehicle is optionally connected to an OBD-II reader, using either Universal Serial Bus (USB), Bluetooth or WiFi. The reader uses the Controller Area Network (CAN) protocol to periodically collect sensor data. These interactions are showed in figure 3.2.

The C-ITS environment, in-vehicle smartphone setup, data collection and analysis are also described, in detail, in [51]. The article explains how the vehicle and phone sensors data is obtained and explores the many prospects of how the data may be reliably used in tasks such as predicting or detecting road congestion, road hazards and weather events.

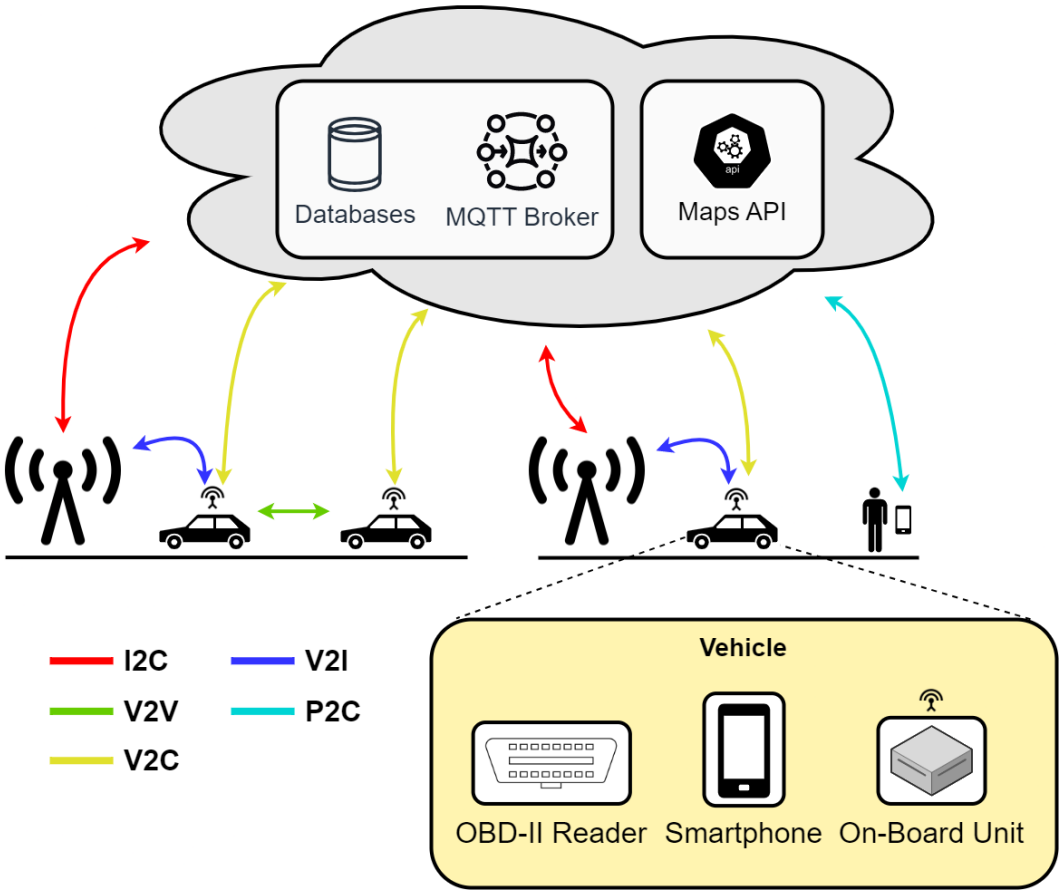


**Figure 3.1:** C-ITS architecture of the V2X communications framework.

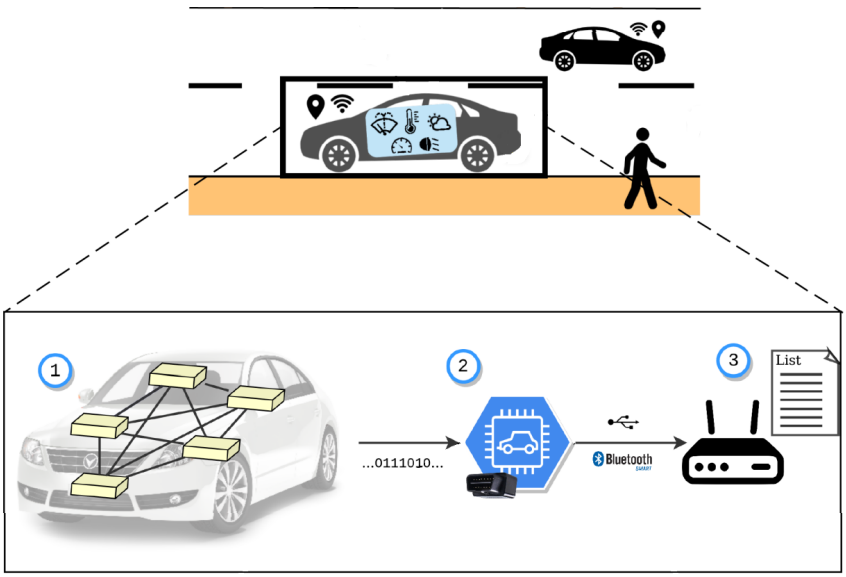


**Figure 3.2:** CAN data collection [51].

## 3.2 User Modes

The application features 3 user modes that cater to different types of users and use cases:

- VRU mode: the application connects to the cloud MQTT broker and directly shares the VRU location, using standard VAMs, for an improved cooperative awareness. The application uses, by default, a quadtree zoom of 10 and subscribes only to messages sent within the tile defined by the user location and quadtree zoom;
- Traffic Control Center (TCC) mode: as in the VRU mode, the application connects to the cloud MQTT broker but using the quadtree zoom of 0 to receive all exchanged messages. This way, a traffic control operator is able to monitor the whole road environment covered by RSUs. In this mode, the application uses the *stationID* 1 which corresponds to the *stationID* of a central ITS-S. The most common use case of this mode is to create/report DENM alerts that are then forwarded by the connected RSU to nearby ITS-Ss;
- OBU mode: the application directly connects to the local MQTT broker of the vehicle's OBUs. It employs a quadtree zoom of 0, in order to receive all messages, since this broker will only exchange messages produced by the vehicle, forwarded from nearby ITS-S stations or sent by the V2X application (e.g. DENMs). This mode provides access to additional C-ITS services like tolling and EV charging.

The flow of messages sent and received by the mobile application, as well as the associated interfaces and communication technologies, are represented in the following figures. Figure 3.3 refers to the VRU and TCC modes, while figure 3.4 depicts the OBU mode.

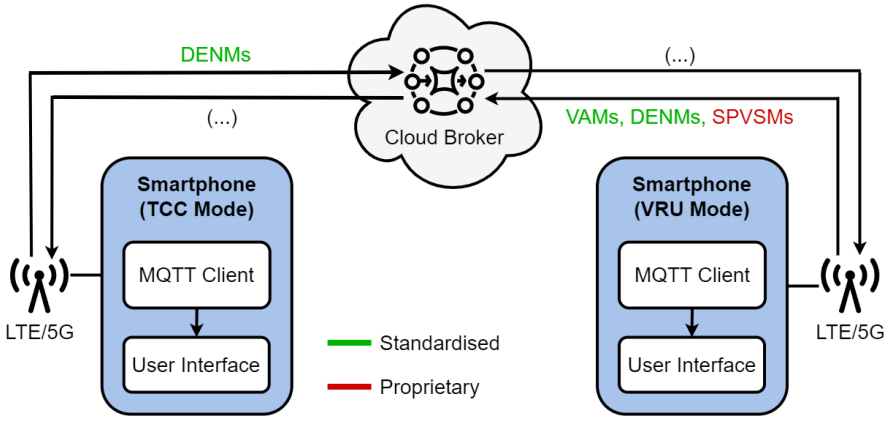


**Figure 3.3:** Message exchanges in VRU and TCC modes.

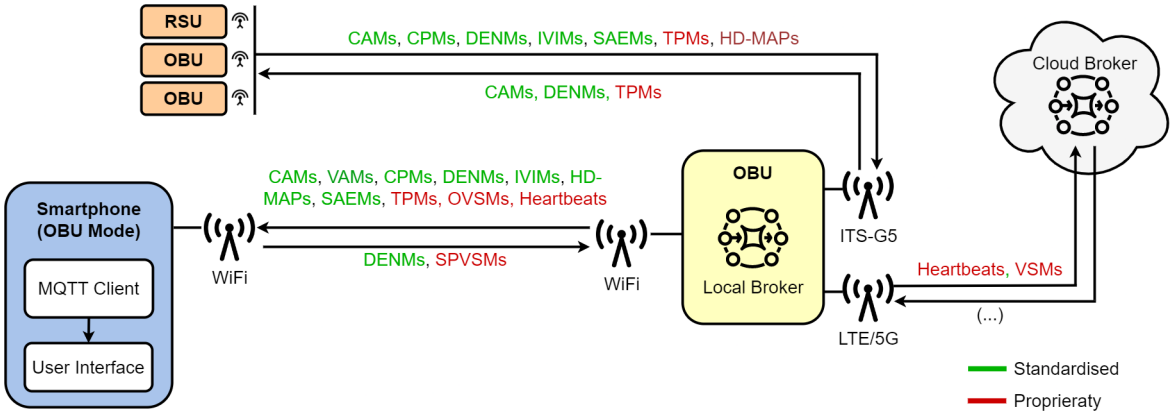


**Figure 3.4:** Message exchanges in OBU mode.

## 3.3 Mobile App Architecture

The mobile application was developed in RN using the Expo SDK and TS, a statically typed superset of JS. From an architecture standpoint, the most relevant libraries used, as depicted in figure 3.5, are:

- react-navigation: provides a way to define and manage different screens, transitions between them, implements the bottom navigator component and supports many navigation patterns [52];
- react-native-paho-mqtt: provides an MQTT client that communicates using WS, being an ES6-ready, promise-based RN compatible version of the Eclipse Paho client [53];
- react-native-maps: provides components such as the MapView (with the visualized road environment), Markers, Polyline used to draw lines and Polygons as well as methods to interact with these components [54];
- expo-sensors: part of the Expo SDK that provides a platform-agnostic API to measure the values from mobile phone sensors [55].

The use of Expo limits the development of many types of applications, since the developer is confined to the use of JS-only libraries and Expo native APIs. Examples of these limitations are the access to all native sensors, Bluetooth, WebRTC, Portable Document Format (PDF) support and others. Support for the last three mentioned features in Expo applications is only possible through the use of config plugins and custom development clients, which considerably increases the development complexity. If no config plugin exists for the desired functionality, the developer must eject the project to a bare RN project.

Expo greatly simplifies the build and deployment process for both Android and iOS platforms with EAS [56], including builds with config plugins. EAS provides a free tier account with 30 monthly low-priority queue builds and the developer can also compile the build on their computer using the same command used to upload the source code to EAS but with the *–local* flag. Expo was chosen because all the architectural requirements of the application described in this dissertation are satisfied by its SDK.

The application features three screens:

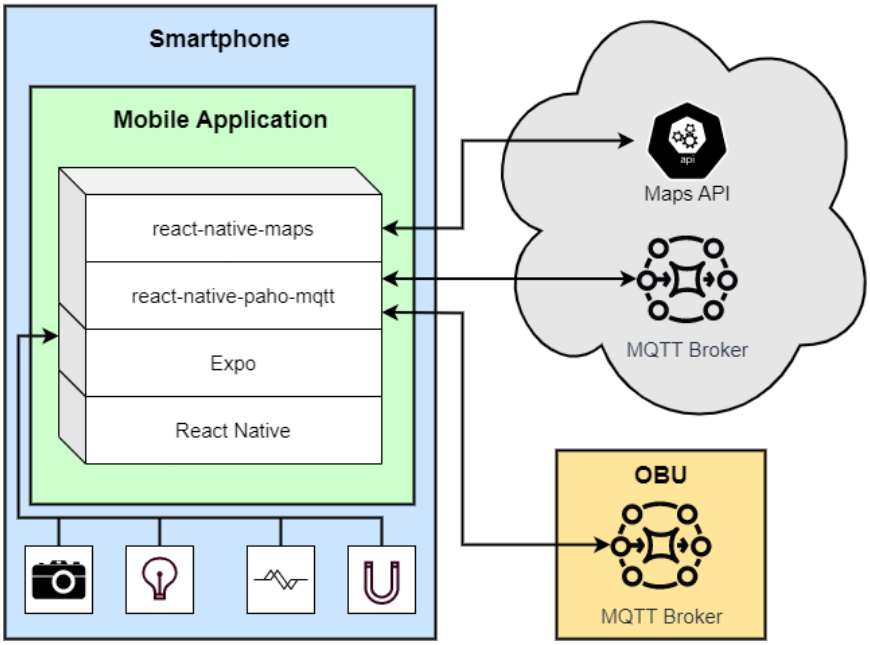


**Figure 3.5:** Mobile application architecture and external connections.

- Sensors Screen: screen used to visualize the sensor readings of the phone and connected vehicle, when using the OBU user mode. The screen features 3 tabs to switch between the sensors of the phone and of the road vehicle or boat;
- Map Screen: the main screen where the user can visualize other road users, interact with the map, create DENM alerts and browse the history of events and other messages. In OBU mode it also displays tolling information and provides a way to manage EV charging reservations;
- Settings Screen: the screen to configure the app language, broker connection Uniform Resource Locator (URL) and credentials, *stationID*, quadtree zoom, message transmission period (VAMs and SmartPhone's Vehicular Sensor Messages (SPVSMs)), the delay for automatic clearing of the notifications, and also a way to reset individually each setting of the application or all of them at the same time.

Due to the very high rate of messages incoming into the application, it utilizes a stateless architecture enabling the establishment of a refresh period instead of continually re-rendering the components upon receiving each new message.

## 3.4 Mobile Application Services

The application supports two additional V2X services: tolling and EV charging. These services are advertised by surrounding ITS-S and are further explained in the following subsections.

### 3.4.1 V2X Tolling

The tolling service is advertised using SAEMs, although the SA standard [20] doesn't specify the application specific announcement data for each service. The data announced is

encoded using the ASN1.C Octet Encoding Rules (OER) encoding that is embedded in the **applicationDataSam** field of SAEMs. It contains the type of tolling, the geometry of the tolling zones, expressed as lists of coordinates, and their corresponding azimuth angle [57].

The fact that the service is implemented using certified OBUs guarantees that the service uses correct vehicle characteristics for the transactions, eliminating the need for vehicle classification on the infrastructure side. The type of vehicle, weight or number of axles are often used to determine the fee to be collected.

There are two types of tolling systems supported using TPMs:

- **Open system**: when a vehicle enters a single tolling zone and the payment is done immediately. The flow of messages is represented in figure 3.6:

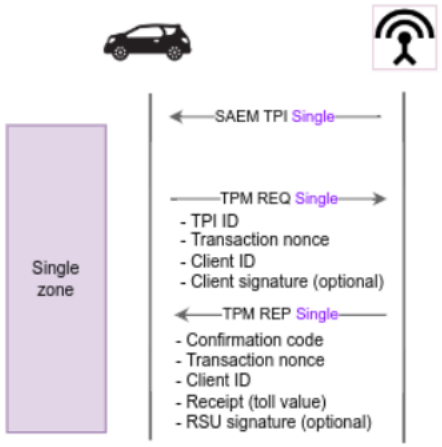


**Figure 3.6:** Open system tolling.

- **Closed system**: a vehicle firstly drives through an entry zone, in which a proof is generated to register the user's passage. When the vehicle passes the exit tolling zone, the entry proof is sent in a request and the payment is then processed. The message flow is depicted in figure 3.7:

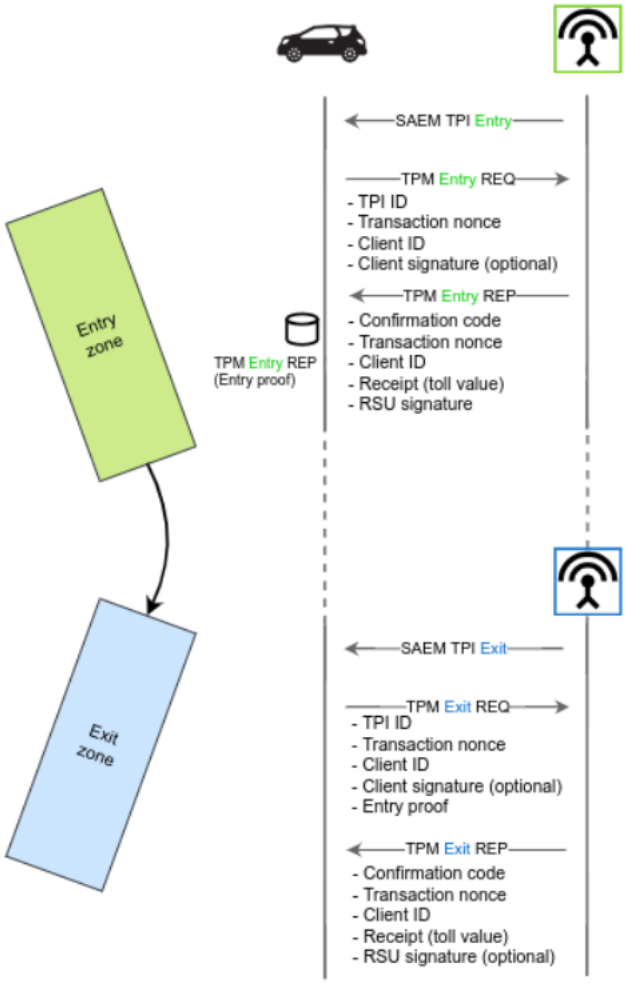


**Figure 3.7:** Closed system tolling.

While the vehicle crosses a tolling zone a series of TPMs are exchanged between the vehicle and the road infrastructure. These are shown to the user using the notification component, whose implementation will be described in subsection 4.2.8.

### 3.4.2 V2X EV Charging

The EV charging service is advertised by the RSUs using EVCSNMs, as defined by ETSI in [23]. The messages contain the location of the charging stations and details about the availability and characteristics of the types of charging supported. The standard has support for Alternating Current, Direct Current, ChaDeMo and inductive charging and many types of sockets and Direct Current charging cables. It also supports battery swapping, alternatively known as quickdrop.

In the mobile application it is then possible to create a reservation, using the message flow of EVRSRMs, defined by ETSI in [24]. The application starts by sending a pre-reservation message with the amount of energy and schedule desired for the charging session. This message is generated in the mobile app, being sent by the vehicle's OBU to an RSU, which then forwards it to the platform responsible for managing the EV charging stations (Central ITS-S). The platform then replies with either a successful pre-reservation or with an error

and the reason, e.g. unavailability of the charging spot due to two users making requests with overlapping schedules for the same plug. After a successful pre-reservation reply, the application may create the reservation request using the *PreReservation-ID*, the identifier attributed after a successful pre-reservation, and the type of charging or type of batteries, arrival and departure dates, the energy amount and the payment information. The reservation response can also be successful or return an error, e.g. wrong payment information, insufficient power at the station or an invalid station identifier.

The reservation can also be canceled or updated with a change of schedule, however, it is only possible to postpone the arrival date and/or prepone the departure date. For this reason if the session must take place later on, the reservation must be canceled and a new one created. The described flow of messages between the different elements of the EV charging reservation process is presented in figure 3.8.

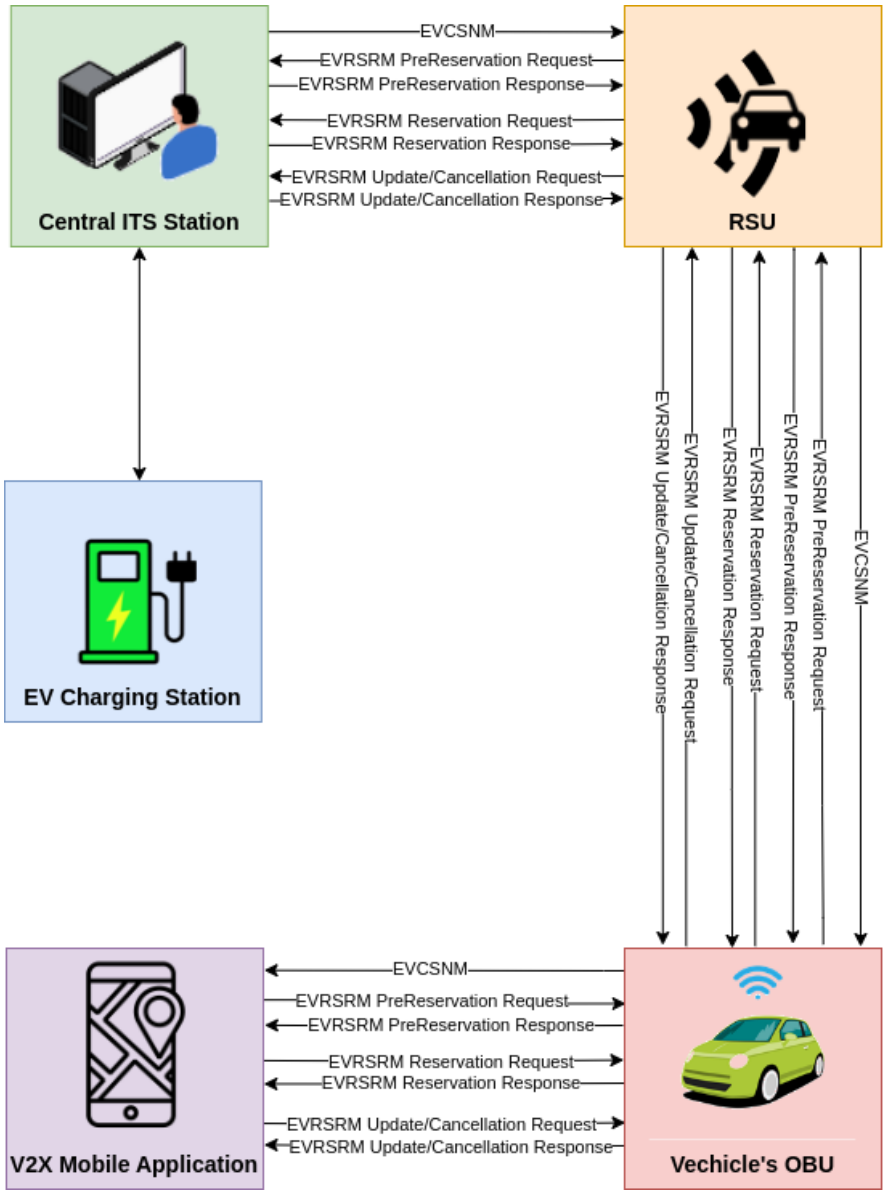


**Figure 3.8:** Message flow of the V2X EV charging reservation process.

## 3.5 Smartphone's Sensors and Interfaces

The mobile application, when in OBU mode, collects data from the phone sensors and sends it to the vehicle's OBU using a custom message type named SPVSM. Similarly, the vehicle's sensors reader also publishes its messages in the OBU's MQTT broker. If the vehicle is a car, the information is encoded in an OBD-II's Vehicular Sensor Message (OVSM), while boats use SoNar's Vehicular Sensor Messages (SNVSMs). The phone's and vehicle's sensors messages are combined, by the OBU, into a single message, called Vehicular Sensor Message (VSM), that is then forwarded to the cloud MQTT broker.

The sensors available in the phone are the ones implemented by Expo-Sensors:

- Accelerometer;
- Barometer:
  - Pressure;
  - Relative altitude (iOS only).
- Gyroscope;
- Light Sensor;
- Magnetometer;
- Pedometer.

The Light Sensor was added as a pull request in the scope of this dissertation, see [58], and it was integrated into Expo-Sensors in Expo SDK 47. There is also a discussion [59] on why it wouldn't be feasible, currently, to support the light sensor for iOS devices in Expo.

Road vehicles provide a lot of information, made available by the OBD-II, and include:

- Speed;
- Revolutions Per Minute (RPM);
- Load;
- Pressure;
- Temperature;
- Speed pedal status;
- Brake pedal status;
- Steering wheel angle;
- Lights status;
- Anti-lock Braking System (ABS);
- Wipers status;
- Individual wheel speed.

Boat's sensors differ considerably from road vehicles. The information shared in SNVSMs is listed bellow:

- Reference position;
- Speed;
- Heading;
- Water depth;
- Water temperature.

CHAPTER 4

# Implementation

This chapter outlines the implementation details of this work. It describes the libraries used and how the mobile application relies on them to provide the C-ITS services previously defined. In some occasions, platform-specific implementations, usually implemented using wrapper components, were required either for performance or appearance consistency.

## 4.1 Screens

There are three screens in the bottom tab navigator (figure 4.1), a commonly used navigation pattern, which were developed by employing the *react-native-navigation* library. These three will be explored in the following subsections.


**Figure 4.1:** Bottom tab navigator.

### 4.1.1 Map

The map is the default screen of the mobile application, it is used for navigation and visualization and creation of C-ITS message. The components for both the MapView, Marker, Polylines and Polygons are part of the *react-native-maps* library and are the basis for all visualization in this screen. The map provider used is Google Maps to increase the consistency in appearance across the Android and iOS, which has as default the Apple Maps provider, applications. Some C-ITS messages are only displayed using the notification component described in 4.2.8, while others are drawn directly on the map like Markers with images, Polygons or Polylines.

The VRU and TCC user modes have a refresh rate of 400 ms, while the OBU mode's refresh rate is 100 ms, since that is the maximum rate of CAM generation by the vehicle [16]. The higher refresh rate is important when the vehicle is traveling at high speeds, in order to

keep the movement of the vehicle on the map smooth. The use of interpolation and animated markers proved too computationally intensive for lower end Android devices and so it was discarded.

For both VRU and TCC modes, the location is obtained using expo-location. In OBU mode, the location is obtained from the received CAMs and the heading is calculated using the current and last position. For this purpose, it is necessary to setup the mobile application using the same *stationID* as the one published by the vehicle's OBU in the CAMs.

The application also uses a *react-native-maps* camera type for the user region on the Map component, which allows not only to control the coordinate position that the map is displaying, but also the heading, pitch and yaw.

While the user doesn't connect, there is a three state button, using *rn-tri-toggle-switch*, which will be further described in subsection 4.2.12. The toggle is located on the top left corner and allows for the selection of the user mode. There is also a big blue button, see subsection 4.2.1. When connected and the map isn't centered on the user location, there is a button on the bottom left to center the map on the user location. On the top left corner there is a button that shows the icon of the selected user mode and allows the user to access a dashboard with basic location information, presented in subsection 4.2.13. Below the dashboard button is the button for the History component, a modal, that provides a list of the past events to the user. It is possible to click on each of these past event messages in order to go to their location, allowing the user to visualize the events again, described in subsection 4.16. The EV charging reservation button, specified in subsection 4.2.5, is placed below the history button after a successful reservation and can be used to update or cancel it. Figure 4.2 presents the map screen for the three user modes, as well as all the buttons previously described.

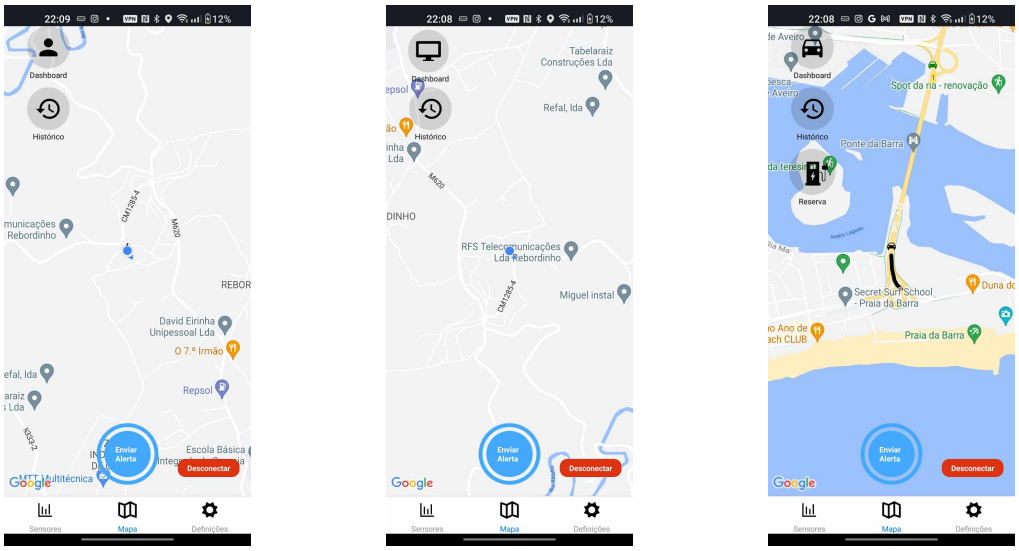


**Figure 4.2:** Map screen in all user modes: VRU, TCC and OBU.

Different user modes have distinct behaviors when reacting to the several events that are received through the message types described in section 2.2.2. The VRU mode makes use of the quadtree zoom and so it only receives and shows events within its location and zoom, while

the TCC mode can see all events regardless of the distance to it. The TCC always connects with the *stationID* set to 1, acting as a central ITS-S. All events that produce markers in the map disappear if the user zooms out too much, in order to help with performance. In OBU mode, the application calculates the heading of the vehicle and uses it to only show DENMs and IVIMs in which the events are within 45 degrees of the user heading. When no heading is present in this optional field of the message, the event is always displayed. DENMs and IVIMs also include a field with its validity duration that, after expiring, makes them disappear. For both VRU and OBU modes, DENM markers are only showed if the event is within the relevance distance. The notification, see subsection 4.2.8, for DENMs, IVIMs, TPMs and EVRSRMs are only showed once, while the markers are always displayed as long as they meet the criteria for each specific user mode, in terms of relevance distance, validity duration and heading. For instance, in OBU mode, if the user turns around and heads back to the location of the event, such as a DENM or an IVIM that includes the heading of the event, the notification won't reappear even if the marker of the event disappears and reappears again due to the user changing its heading.

Some of the most common events that the application displays are DENMs and IVIMs. These can appear when connected with any user mode. Other V2X events that are specific to the OBU mode concern tolling zone visualization and tolling message notifications with relevant information about the payment. The use case diagrams for these visualizations are shown in figures 4.3, 4.4 and 4.5.

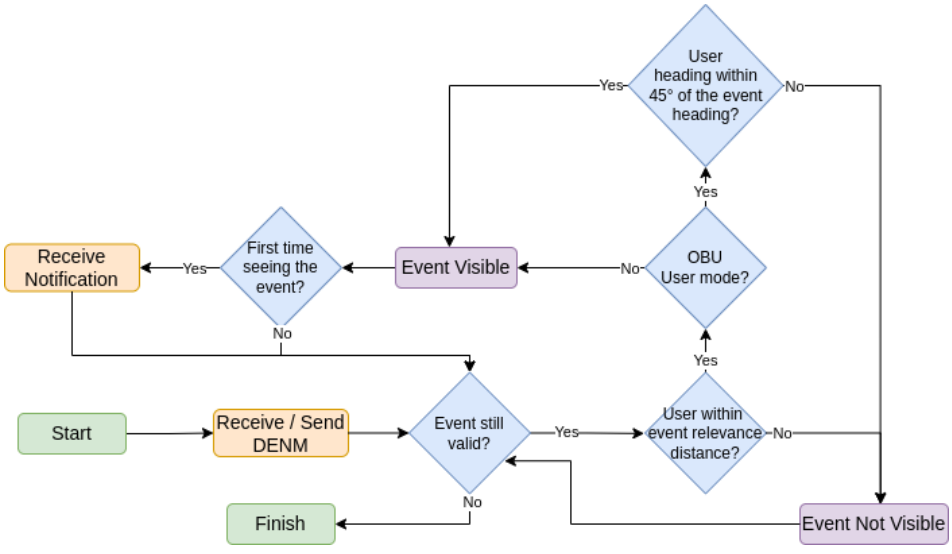


**Figure 4.3:** DENM visualization use case.

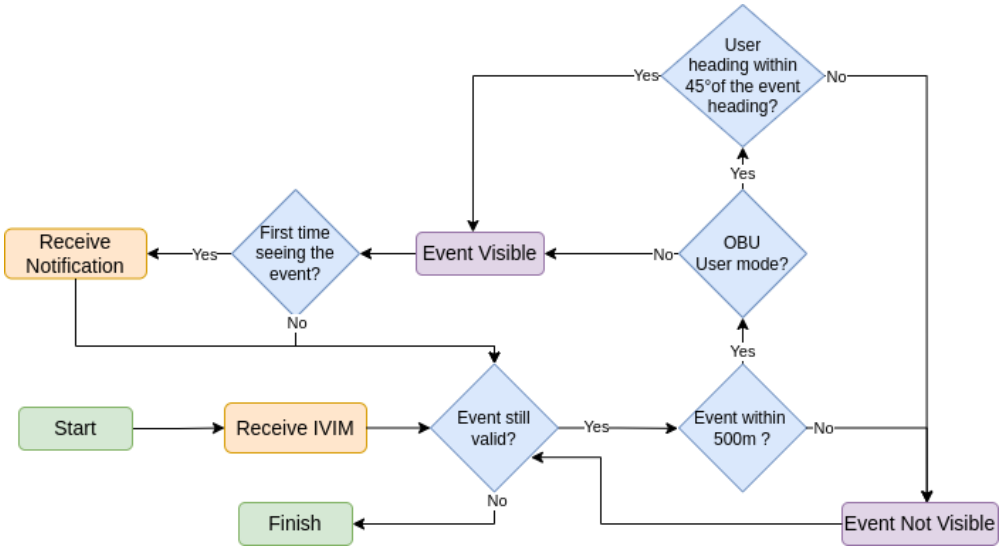


**Figure 4.4:** IVIM visualization use case.

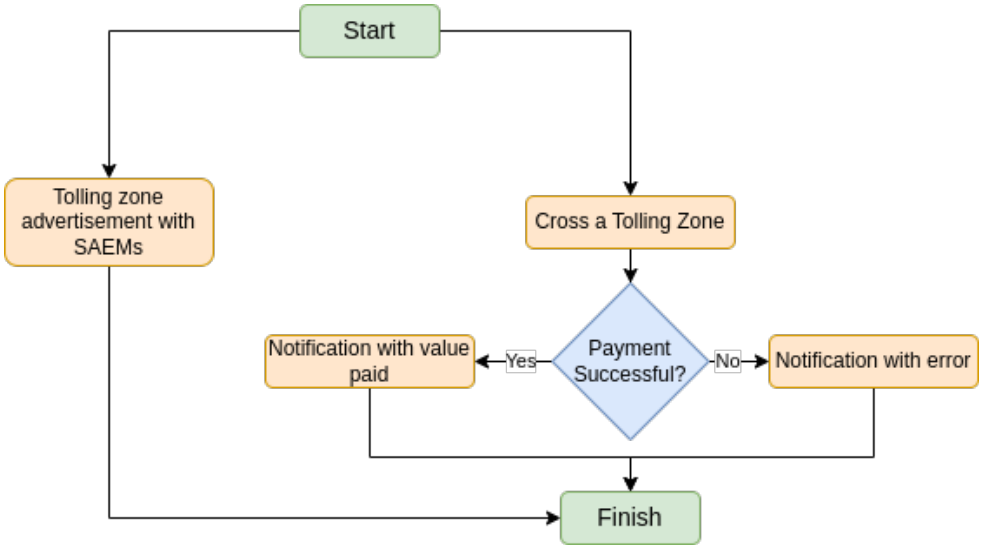


**Figure 4.5:** SAEM and TPM visualization use cases.

### 4.1.2 Sensors

The sensors screen is a component that allows the visualization of sensor information from three different data sources. The main/default view is the one for the mobile phone sensors, being always available. The other two are only accessible in OBU mode and require the presence of those messages to be available to the user. There is one view for the road vehicle sensors, using OVSMs and the other for boats, based on SNVSMs. The sensors available in each of these views are all described in section 3.5. Since the application already has a navigation tab on the bottom of the screen, the selection of each sensor view is done on the top using a 3-button component. The button with dark gray background is the one currently being displayed, while those with white background are the ones currently available, i.e. with sensor information to display to the user.

On the left screenshot of figure 4.6, the phone sensors are being displayed. The screenshot

example on the right shows the road vehicle sensor information. Since the mobile application is connected to an OBU and the OBU is producing OVSMs, the car sensors icon has white background on the left screenshot, while the boat icon is deactivated.

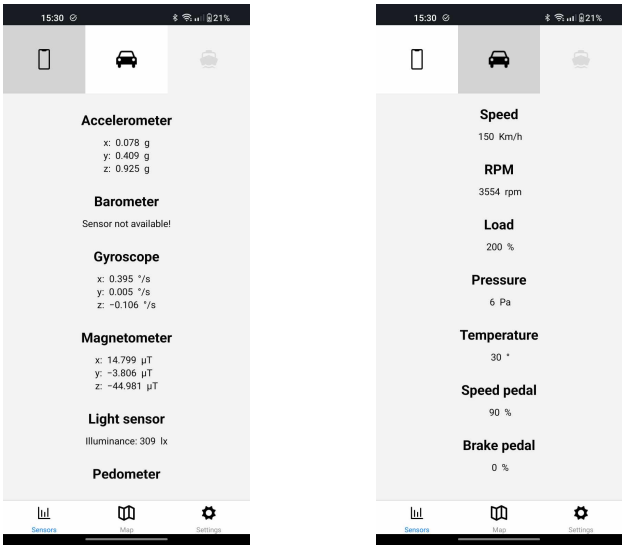


**Figure 4.6:** Sensors screen.

### 4.1.3 Settings

This screen is used to configure the behavior of the mobile application with the following parameters:

- Central MQTT broker **host**, **port**, **username** and **password**;
- OBU MQTT broker **host**, **port**, **username** and **password**;
- **Custom *stationID***:
  - used to identify the ITS-S. By default, the application uses a pseudo-random value that is the result of an hash function with the following input parameter:
    - *androidId*, from **expo-application**, on Android. A hexadecimal string that corresponds to a unique combination of app-signing key, user, and device;
    - *getIosIdForVendorAsync()*, also from **expo-application**, on iOS. A unique combination of device and app vendor.
  - in OBU mode, it can be used to identify the driver's vehicle. The user location on the navigation map is then obtained from CAMs received with the specified *stationID*;
  - in TCC mode this value is ignored, the *stationID* is always 1.
- **Quadtree zoom**, selected using the component described in subsection 4.2.3 and used only in VRU mode;
- **Message disappear delay**, used to create a callback with the specified timeout, in ms, value for making the notifications disappear;
- **VAMs sending interval**, in ms, messages sent only in VRU mode;
- **SPVSMs sending interval**, in ms, sent in VRU and OBU mode;

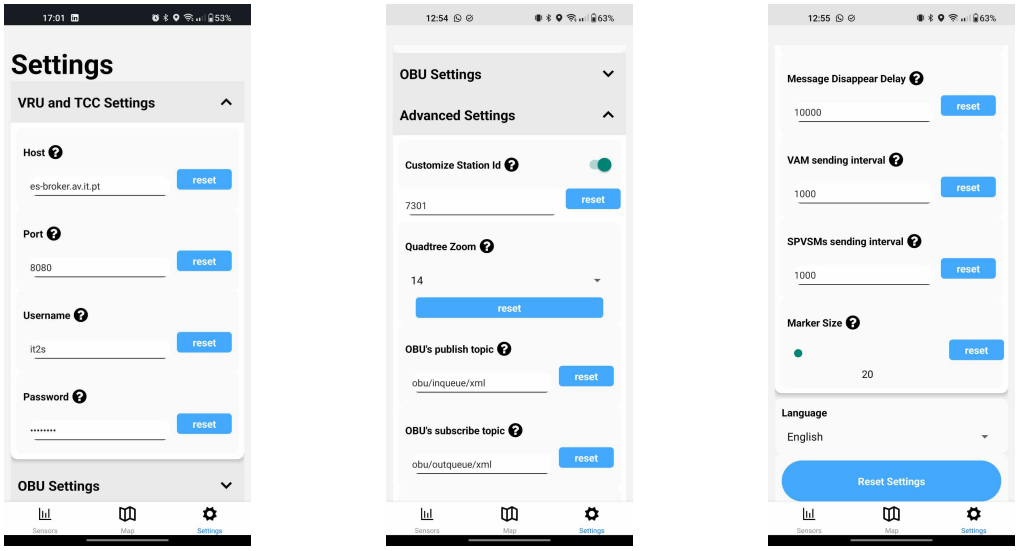


**Figure 4.7:** Settings screen.

- **Language**, either Portuguese or English, being chosen through the component described in subsection 4.2.3.

The multiple settings are divided into accordions, a component which shows a title and arrow that can be clicked to expand or collapse a view with its corresponding settings. The accordions are for VRU and TCC modes, opened by default, OBU mode and advanced settings, both closed by default. This is done because there's a large amount of settings , thus increasing usability and user recognition. All settings have, next to its label, a question mark icon that places a small text component on top of it. It shortly describes the corresponding setting to help the user understand what the settings does. All settings also have an individual reset button to restore them to the default value and at the end of the screen there is a button to reset all settings. The application saves the settings, immediately after any change, using *AsyncStorage*, an API from expo, and loads them again when the the application is relaunched.

## 4.2 Components

This section describes the different components that the application has to display information and to provide user interaction. The only component addressed in this section that wasn't developed in the scope of this dissertation is the User Mode Toggle, available using *rn-tri-toggle-switch*, and to which two contributions were done, see 4.2.12.

### 4.2.1 Connect Button

The connect button is the blue button on the bottom of the map screen. When the application is not connected to any MQTT broker, the button has the format of a large rectangle with semi-circle ends and with the text "Connect". When the button is pressed and a connection is successfully established, it animates into a circle with the text "Create Alert", that can then be used to create and send DENMs. The animation is accomplished using a

RN animated view and by manipulating, over a small time period, the width of the button view. Since the border radius is 50%, the circle can be achieved with a square view. Figure 4.8 shows the button while disconnected from the broker, on the left, and when connected, on the right.

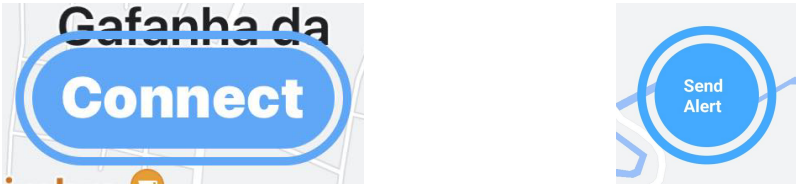


**Figure 4.8:** Connect button: disconnected (on the left); connected (on the right).

### 4.2.2 DENM Popup

The popup, commonly called a modal, appears when the application is connected to an MQTT broker and the connect button (displaying "Send Alert") is pressed. This component is used to create and send DENMs. The user is able to select the type, validity duration and relevance distance of the event.

When the current event type icon is clicked, a menu with all possible cause codes is displayed, each of them represented by a name and an image (figure 4.9). Then, after an event is selected, a sub-menu appears with the sub-cause codes corresponding to the cause code selected. All the code and sub-cause codes defined in the DENM standard [18] are supported and are listed in the DENM utils subsection 4.3.5.

A custom implementation, using a wrapper component, of a community picker is used to select the validity duration and relevance distance of the event, as described in subsection 4.2.3. To select the relevance distance, two pickers are used: one to select the time units and another to select the value, from 50 meters up to 10 kilometers.

After pressing the button with the text "Send" (figure 4.9), the event is transmitted to the broker, received by the application and only then showed on the map at the user location.


**Figure 4.9:** DENM parameters and event type selection popups.

Figure 4.10 depicts the flow diagram that represents the process of creating a DENM.

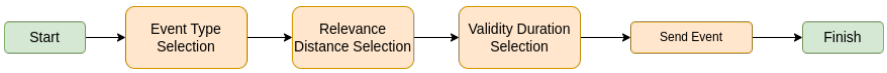


**Figure 4.10:** Flow diagram of DENM creation use case.

A similar component was created to update and cancel a DENM. It allows the user to change the type of event, validity duration and relevance distance of an existing event by sending a new DENM that updates the one selected. There is also a button to cancel the event with a cancellation DENM.

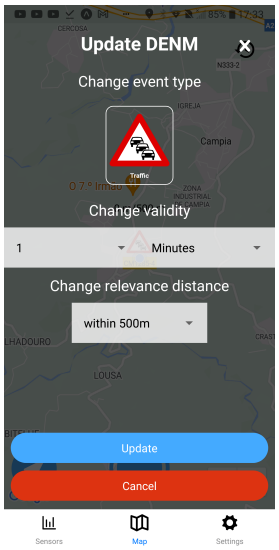


**Figure 4.11:** DENM update/cancellation popup.

The update/cancellation popup is only accessible when the selected DENM was created by the user, which is determined by comparing the originating *stationID* of the DENM and the one from the mobile application. DENMs created by other users can be negated by sending a negation DENM, thus removing the event from the map. The option to update/cancel or negate a DENM is given when the user clicks the marker of the event. Figure 4.12 shows the two custom callouts components that are displayed on the map.

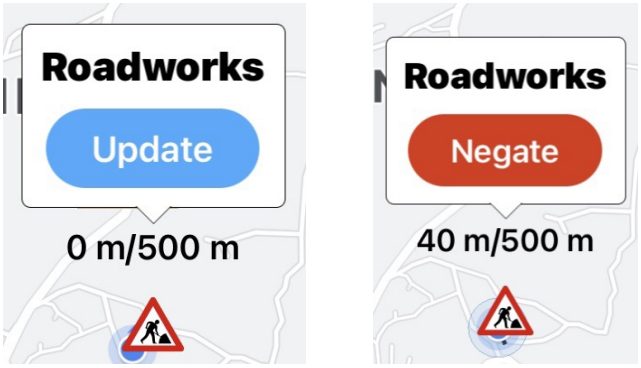


**Figure 4.12:** DENM update, cancellation and negation callouts.

Figure 4.13 shows the flow diagram associated with the update, cancellation and negation of DENMs.

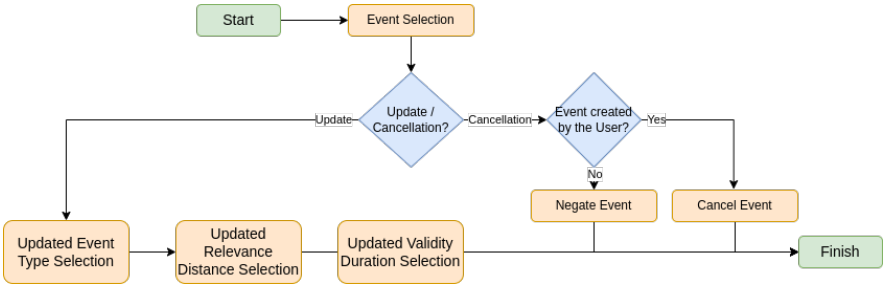


**Figure 4.13:** Flow diagram of DENM update, cancellation and negation use cases.

### 4.2.3 Custom Picker

The custom picker is a wrapper component of the picker provided by the library *@react-native-community/picker*. The Android implementation acts like a dropdown, with the options presented inside of a modal, placed on the center of the screen. The color scheme is impossible to adapt to dark mode because the background color only applies to the collapsed component, which makes the white text invisible in the modal. Figure 4.14 includes both the picker when collapsed and the modal.


**Figure 4.14:** Custom picker on Android.

The iOS implementation consists of a scrollable barrel shaped component. It does introduce a usability issue, by allowing the implementation of scrollable components inside another scrollable component. This issue can be mitigated by ensuring that no two scrollable children are consecutive. This can be achieved by leaving space for the user to navigate the scrollable parent.


**Figure 4.15:** Custom picker on iOS.

### 4.2.4 History Modal

The history modal allows the user to see the history of DENMs, IVIMs, SAEMs, TPMs and HD-Maps. Each type of messages is stored inside a separate accordion to be easier to navigate. All messages have an individual delete button and, at the bottom of each accordion view, there is an option to delete all messages of that type. It is also possible to view the event again, by pressing it: the map positions itself in the location of event and displays it again. Figure 4.16 shows the history component with the DENM accordion opened with an active DENM, characterized by the blue background. If there are no messages of a given type, the accordion of that type won't be present.

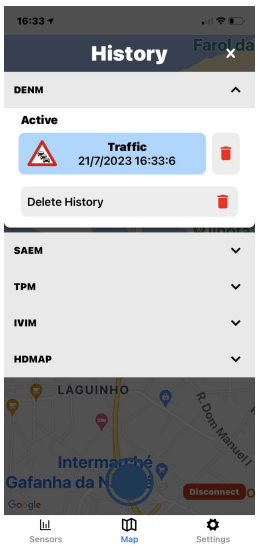


**Figure 4.16:** History component.

### 4.2.5 EV Reservation Components

This subsection describes the EV charging components used to advertise charging stations and to create and manage reservations.

- **Marker Callout**: a component that appears when the user taps the EV charging station marker icon on the map. The library used (*react-native-maps*) imposes a limitation for the inclusion of multiple buttons inside callouts, as discussed in this issue [60]. Therefore, even though the component has a button, it merely serves as a visual cue for the type of action users can expect. Therefore, pressing the button or anywhere inside the callout will produce the same result. Figure 4.17 shows the callout.

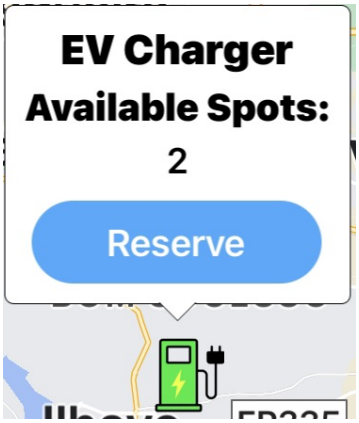


**Figure 4.17:** EV reservation marker and callout.

- EV charger choice: this component is a modal that presents the multiple choices regarding the types of charging and types of receptacle offered by the chosen station. Figure 4.18 is an example of a station that has 2 charging spots and only one charging type available.

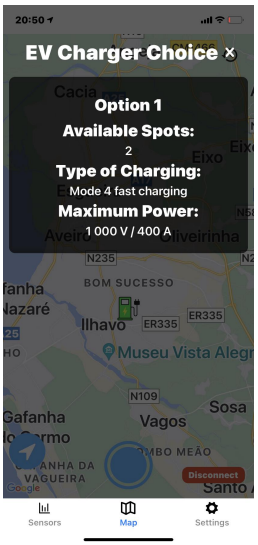


**Figure 4.18:** EV charger choice component.

- EV reservation component: a modal that allows the user to schedule the charging session by providing the start and end dates/times, the desired energy amount and the type of power source. The reservation process is separated into two different phases, as mentioned in section 3.4.2. The type of power source and desired amount of energy are only sent in the reservation message, so the application saves this information and performs the message exchanges in a transparent way to the end user while using notifications to inform the status of the responses. Figure 4.19 shows the modal with the components used to input the necessary information. The EV charging reservation creation use case diagram is depicted in figure 4.20.
- EV reservation management component: it is a modal similar to the previous one, created to update and cancel reservations. It is accessible, after a successful reservation,

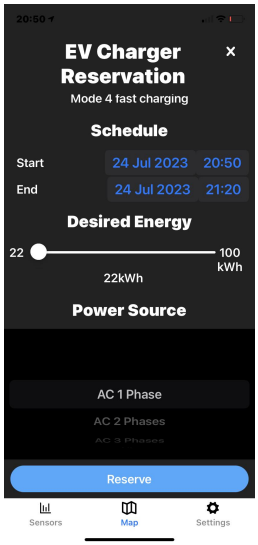


**Figure 4.19:** EV reservation component.

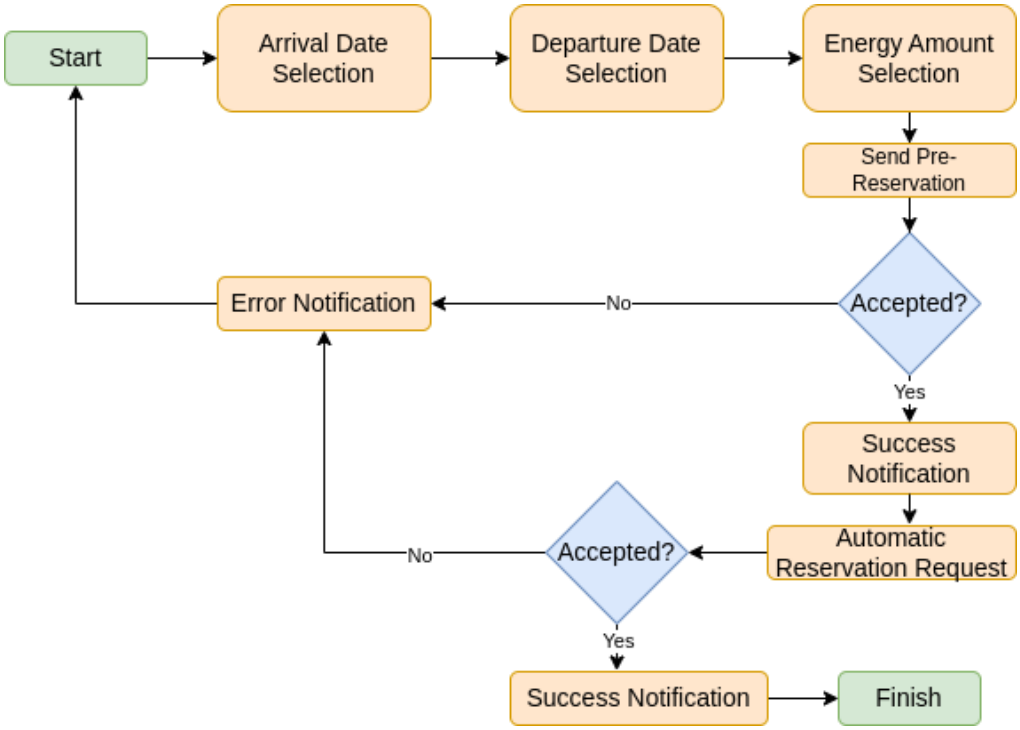


**Figure 4.20:** Flow diagram of EVRSRM creation use case.

on the map screen below the history button. The update functionality only allows the user to postpone the arrival time or to anticipate the departure time. Figure 4.21 shows the modal appearance, while figure 4.22 depicts the use case diagram for the update and cancellation of EV charging reservations.

The reservation process has many ways in which it can fail, i.e. if another user is able to reserve the EV charger before the transmitted request, or if the payment information is invalid. The application uses its notification system to queue alerts that communicate to the user the status of the operations executed.


**Figure 4.21:** EV reservation update, cancellation and negation component.

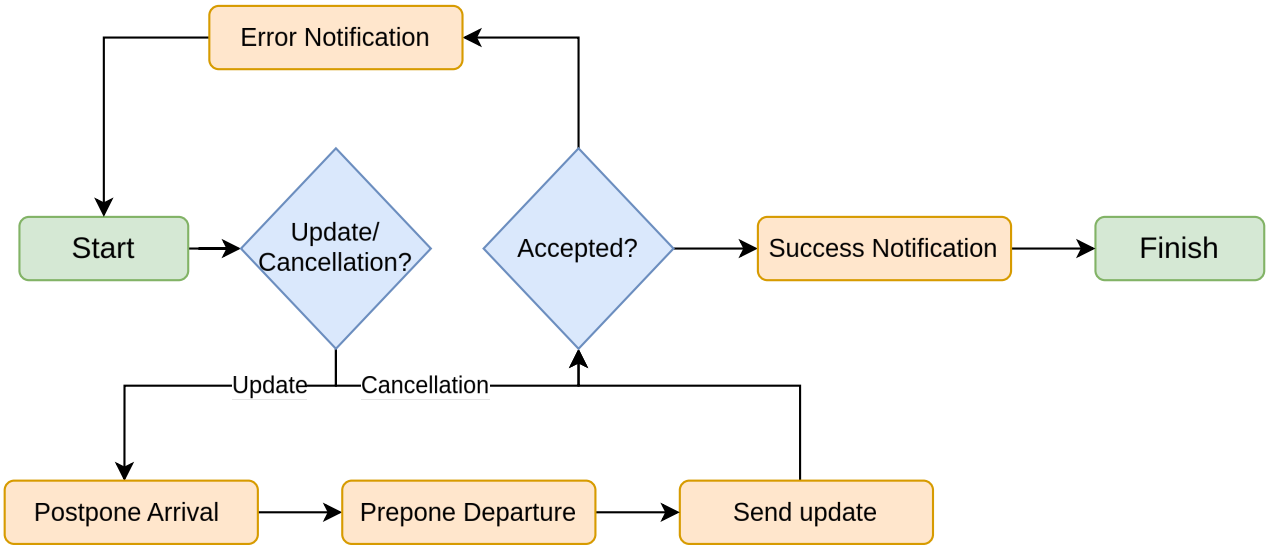


**Figure 4.22:** Flow diagram of EVRSRM update, cancellation and negation use cases.

### 4.2.6 Custom Date Picker

The implemented custom date picker is basically a wrapper of the component provided by the library *@react-native-community/datetimepicker*. The Android implementation allows the same picker to be used, first for the date selection, and then, without closing the modal, for the time. Figures 4.23, 4.24 show the Android implementation.

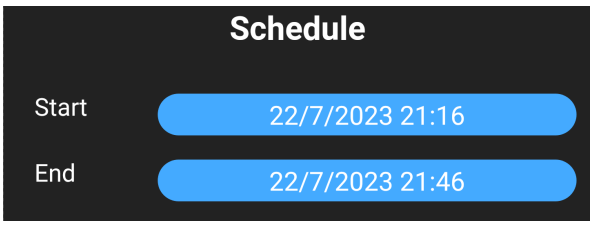


**Figure 4.23:** Custom date/time picker on Android.

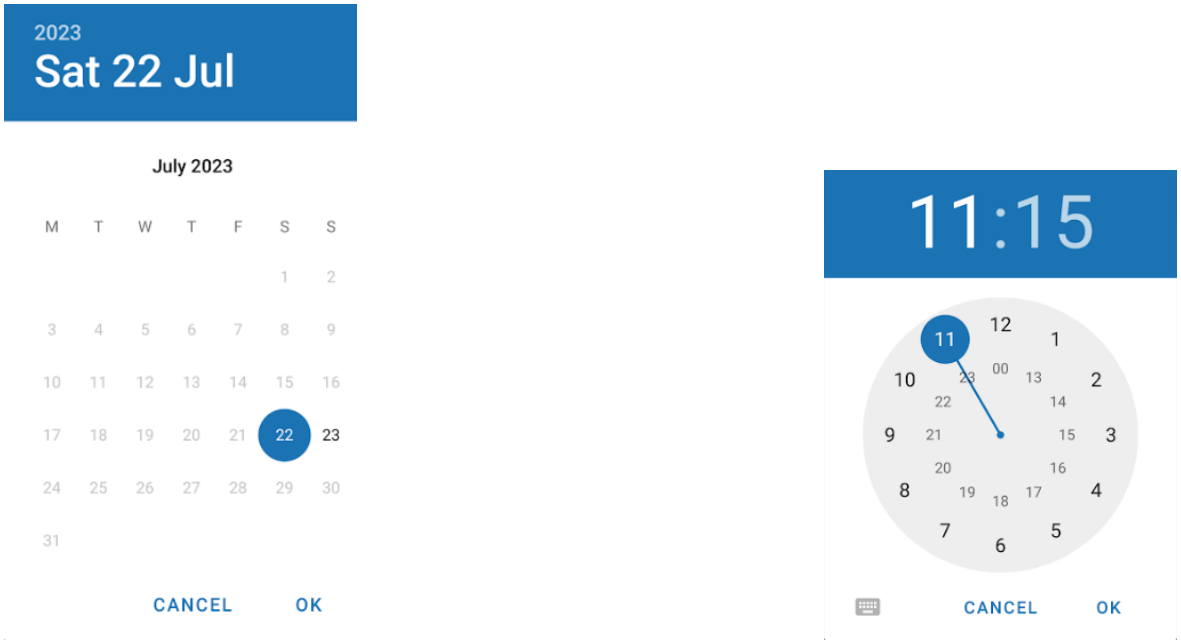


**Figure 4.24:** Date and time selection on Android.

The native iOS picker implementation can be used to select either a date or a time and so it needs two pickers. The iOS calendar is similar to one in Android but the time selection is considerably different. The Android time selection is done through clicking/dragging the selection in a analog like clock, while the iOS one only allows the scrolling of the hour and minute values.

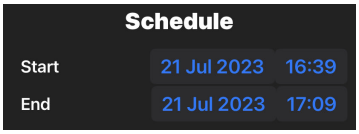


**Figure 4.25:** Custom date/time picker on iOS.


**Figure 4.26:** Date and time selection on iOS.

Due to these distinct behaviors, it is practically impossible to keep the user interface consistent across both platforms. Nevertheless, this custom picker serves as a single wrapper component that is able to perform both date and time selection, either on Android or iOS.

### 4.2.7 SVG Speed Limit

This is a simple component that receives the desired speed limit value as a number and returns a Scalable Vector Graphics (SVG) with a red circle in the background, a smaller white circle on top and the value as black text placed on the center. This component is used in the presentation of some HD-Maps and IVIM notifications and in the history component. Figure 4.27, shows a speed limit IVIM that has this component present, not only on the map icon, but also on the message notification box.

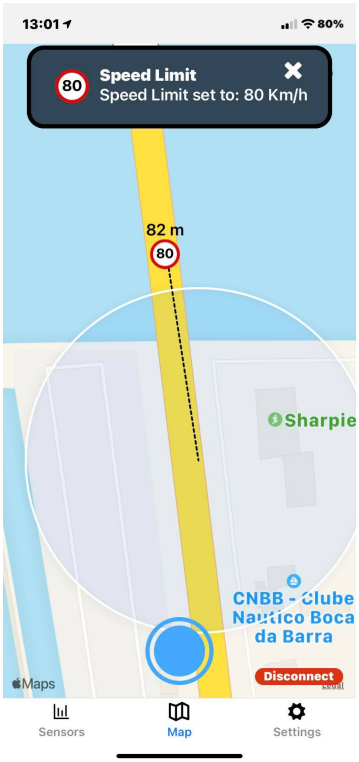


**Figure 4.27:** IVIM speed limit SVG component.

### 4.2.8 Message Notification

Although Expo offers an API to send push notifications using the Operating System (OS), this option was deemed too invasive and not very customizable. A different component was used instead, essentially consisting of a small height modal placed at the top of the screen, that displays a title, subtitle and, optionally, an icon, that can either be a IVIM speed limit (subsection 4.2.7) or a DENM event.

The application features a queue of notifications that moves after the *messageDelay* period, a parameter that can be configured on the Settings screen (subsection 4.1.3). After that period, the notification disappears and if there is another one in the queue, that one will replace it. The user can also dismiss the notifications by pressing the 'x' button, on the top right corner, thus moving the remaining notifications in the queue. Figure 4.28 shows the notifications presented when a user receives an IVIM with speed limit information and a DENM with a slippery road event.


**Figure 4.28:** Message notification examples.

### 4.2.9 Tolling Zones

This component was developed to show the tolling zones on the map, as well as the lane directions. The tolling zones information is disseminated using SAEMs, where the *applicationDataSAM* field includes a custom Tolling Payment Information (TPI) structure, which has the zones described as arrays of 2D geographic coordinates and, for each of the zones, a corresponding heading angle.

The last point in the coordinates array of each zone must be connected to the first point, in order to close the geometry. The shape is displayed using a *react-native-maps* polygon and an arrow with the lane direction, inside the polygon. The line coordinates are calculated by creating an imaginary point at the center of the polygon and then, using the heading, stretching a line until it intersects with any two edges. The arrow is built with a polyline and a marker with the image of a triangle rotated according to the zone heading.

The color of the zones are an indicator of the type of tolling. Purple is used for the open system, with a single tolling zone, and green for the closed system, that has entry and exit zones, both explained in subsection 3.4.1. Figure 4.29 shows, on the left, an open system and, on the right, a closed system where the exit tolling zone is a highway exit.

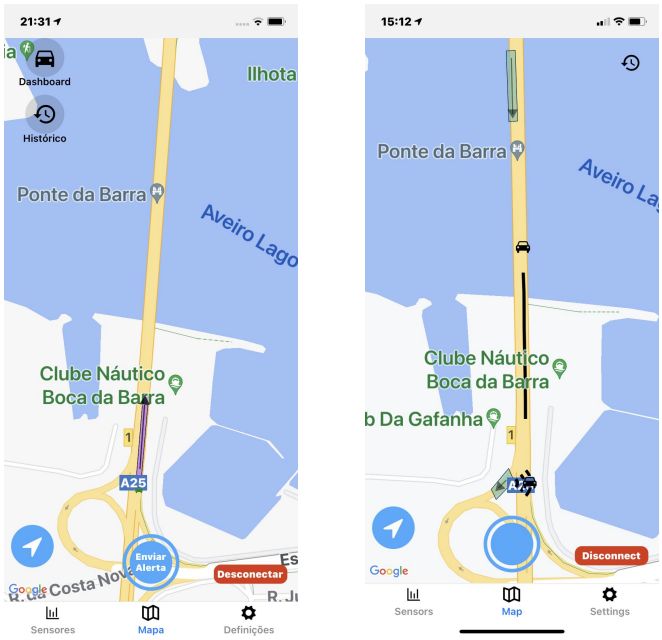


**Figure 4.29:** Open and closed system tolling service advertised using SAEMs.

When the application is connected, in OBU mode, and the vehicle enters a tolling zone, it sends a TPM request and then receives a response determining the success or failure of the

payment. Visual information regarding this message exchange is presented to the user via the notification component, described in subsection 4.2.8.

### 4.2.10 Custom Marker

The *react-native-maps* marker has three different ways of providing the image that is used as the marker icon. An in-depth discussion was conducted on issue [61], being concluded that the best solution, performance-wise, is to use the image prop on Android and to render a child Image component on iOS. Furthermore, the custom callout component, used in subsection 4.2.5, is passed as a prop to the custom marker and is rendered as a child of the marker. The component will render a default callout with the provided title and subtitle if the custom callout prop is undefined.

Animated markers were also implemented, but later removed because, even though they would allow a lower refresh rate, the performance hit of animating all markers proved too intensive for lower-end smartphones. Figure 4.30 shows the marker and default callout, with just a title and subtitle, of a perceived object from a CPM.

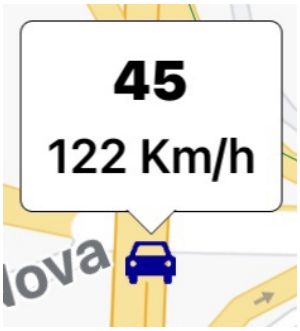


**Figure 4.30:** CPM's perceived object marker and callout.

### 4.2.11 DENM and IVIM Marker Wrapper Components

These components wrap a custom marker, described in the prior subsection 4.2.10, that shows the distance to the event and a dashed line connecting the user to the marker of the event. Figure 4.31 displays, on the left, a slippery road DENM alert and, on the right, a speed limit IVIM, both with a notification on the top of the screen, which only appears the first time the user visualizes the event.

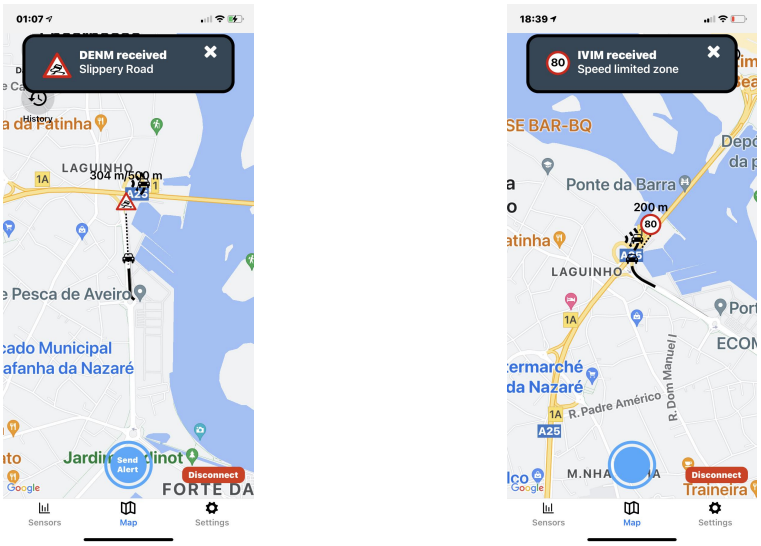


**Figure 4.31:** Wrapper component with distance information and dashed line to the event location.

### 4.2.12 User Mode Toggle

This component is part of an open source library available on Node Package Manager (NPM) that was improved with two pull requests: one to avoid updating the code to use the *useNativeDriver* prop on animations [62], required by RN 0.62; and the other to fix the toggle behavior, that used to be able to leave the component when the user dragged it [63]. Both were successfully merged but the author, however, did not update the NPM package. The cleanest solution is to apply a patch, created with *patch-package*, in order to update the library immediately after installing it, with a *postinstall* script.

The component is located in the upper left corner of the screen and, before connecting, it allows user mode selection, as described in subsection 3.2. After successfully connecting, the toggle is replaced by the dashboard button, which will be detailed in subsection 4.2.13. Figure 4.32 shows the toggle used to select one of the available user modes: VRU, TCC and OBU.

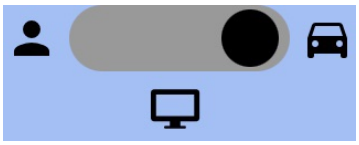

**Figure 4.32:** User mode selection toggle.

### 4.2.13 Dashboard

When the application is connected, the user mode selection toggle disappears and is replaced by a button with the icon of the mode being used and the "Dashboard" text below it. When the button is clicked, a modal is presented with the current geographic coordinates, the speed in km/h, and the heading, measured in degrees from North. Figure 4.33 shows the dashboard button, surrounded by a red square, and the dashboard modal.

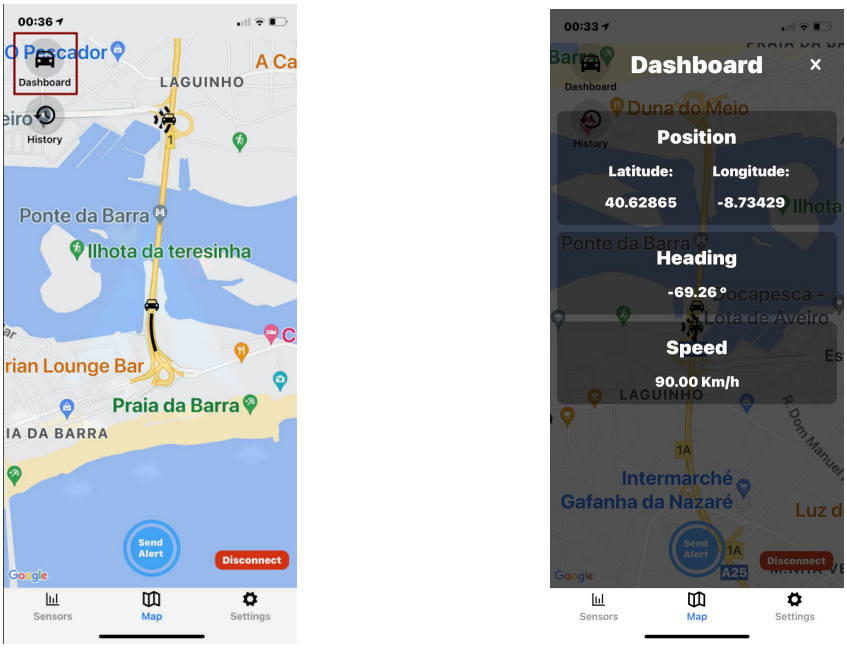


**Figure 4.33:** Dashboard icon and modal.

## 4.3 Utilities

This section is dedicated to the multiple utility functions used by the application to display and process information. It covers the following utilities: math functions related to geographic coordinates (map utilities); static types; async storage; translation objects; DENM utilities; message creation function; message callbacks; and icon functions that provide all supported *stationTypes* used in ETSI standard messages and all CPM perceived object classifications.

### 4.3.1 Map Utilities

The map utilities are a set of functions that help to create the structures needed to display the information present in received C-ITS messages:

- *camTracesToCoordinates()* and *denmTracesToCoordinates()*, for CAMs and DENMs respectively, are functions that translate the traces array, an array of objects that contain the **deltaPosition** change from the last location point, starting with the position of the vehicle itself. They create an array of coordinate points that are then used by the component rendering functions, called *drawCamTraces()* and *drawDenmTraces()*, that create multiple polylines to connect the traces;
- *getPerceivedObjectCoordinates()* is a function called when a CPM is received. It calculates the coordinates of every perceived object based on the **x** and **y** distance, measured in meters to the ITS-S emitting the CPM;
- *queueNotification()* is used to queue a new notification. It makes internal use of *clearNotification()*, which is the function passed to the timeout callback that performs the shift of the notification queue. It either displays the notification immediately, if no other notification is in the queue, or adds it to the end of the queue;

- *getLocation()* is the function that is called when the application first initializes. It determines the position of the mobile phone using the Expo-location API. The Expo-location function *getCurrentPositionAsync()*, just like other native geo-location methods, accepts the desired accuracy as a parameter. The function first attempts to get the highest available accuracy, and if that fails, it tries, sequentially, to obtain the location using a lower accuracy.
  The *getCurrentPositionAsync()* function is then called every time the application re-renders (400 ms), when in VRU or TCC mode, using the highest available accuracy.

### 4.3.2 Static Types

The application is fully statically typed using TypeScript. Static types greatly improve error handling and development efficiency. There are thirteen message types definitions, each in its own typescript module:

- CAM;
- CPM;
- DENM;
- IVIM;
- VAM;
- HD-Map;
- RSU Heartbeat;
- EVCSNM;
- EVRSRM;
- OVSM;
- SNVSM;
- SAEM;
- TPM.

There is also a module that describes the types of global structures used by the application. It contains the definitions for the settings context, the EV reservation objects and the information saved from each received message, e.g. for DENMs the *denm_markers_t* contains a timestamp, a flag of whether the event was already seen by the user, a flag of whether the *validityDuration* has expired and the event cause and sub-cause codes. Saving the entire content of the messages would radically increase the amount of Random-Access Memory (RAM) usage, therefore this simplified format was followed.

### 4.3.3 Async Storage

Expo provides an asynchronous, unencrypted, persistent, key-value storage API that is used to store and load both the settings and message history, when reopening the application. The application features a TypeScript safe module that exports the following asynchronous functions:

- Settings storage:
  - *saveSettings()*: called whenever the user changes any setting in the settings screen;

  - *getSettings()*: used to apply the settings context when the application first initializes. If this method returns undefined, the context will be set with the default settings;
  - *clearSettings()*: a function used to reset the stored settings and the settings context to the default values.
- Message storage:
  - *getMessages()*: passed to *useEffect()*, a React Hook, to obtain the full message history. It runs only once after the map screen is first rendered;
  - *setMessages()*: used internally to add/remove a message from the history. Each pair of user mode and message type is associated with a different message history;
  - *addMessage()*: receives as arguments the user mode, message type, and the message itself. It's a void function that is called on every message callback, exemplified in subsection 4.3.1. This function also calls *getMessages()* to obtain the current history, then it appends the message and finally invokes *setMessages()*;
  - *clearMessages()*: this is a function that is used to delete all messages of a given user mode and message type pair. It is not the same as calling the *setMessage()* using an empty object, since it actually deletes the key-value pair from memory;
  - *clearAllMessages()*: it is called when the bottom button of the history pop-up is clicked. It deletes all messages stored in the current user mode.

### 4.3.4 Locales

The application supports both Portuguese and English languages, being possible to switch between them in the settings screen, as detailed in subsection 4.1.3. Adding support for new languages is as simple as adding new translations on every object with the new language as a key. Any translation object, i.e. an object with several translations, must have all languages supported in its keys and the translations or other translation objects as values. Objects related to notifications usually have translation objects for the title and subtitle. The translations follow a tree-like decomposition of the many textual elements. Listing 1 provides an example of a translation object for the EVRSRM pre-reservation request notification:

```
{
  title: {
    pt: "Pré Reserva de Carregador",
    en: "EV Charger Pre Reservation",
  },
  subtitle: {
    pt: "Pedido de pré reserva efetuado!",
    en: "Pre Reservation request done!",
  }
}
```

**Code 1:** Notification translation example.

### 4.3.5 DENM Utils

The DENM standard [18] specifies, in section 7.1.4, the 24 supported cause codes, each with a varying amount of sub-cause codes. The application supports all cause and sub-cause codes and has at least one icon for each cause code category, as shown in table 4.1.

This module exports an object that maps the event cause and sub-cause codes to a translation object with the description and the corresponding icon name. This object is used in the DENM creation and update/cancellation pop-ups, as described in subsection 4.2.2, as well as on the map when a DENM is received.

| Cause code | Description | Icon |
|---|---|---|
| 1 | Traffic condition |  |
| 2 | Accident |  |
| 3 | Roadworks |  |
| 6 | Adverse weather condition - adhesion |  |
| 9 | Hazardous location - surface condition |  |
| 10 | Hazardous location - obstacle on the road |  |
| 11 | Hazardous location - animal on the road |  |
| 12 | Human presence on the road |  |
| 14 | Wrong way driving |  |
| 15 | Rescue and recovery work in progress |  |
| 17 | Adverse weather condition - extreme weather condition |  |
| 18 | Adverse weather condition - visibility |  |
| 19 | Adverse weather condition - precipitation |  |
| 26 | Slow vehicle |  |
| 27 | Dangerous end of queue |  |
| 91 | Vehicle breakdown |  |
| 92 | Post crash |  |
| 93 | Human problem |  |
| 94 | Stationary vehicle |  |
| 95 | Emergency vehicle approaching |  |
| 96 | Hazardous location - dangerous Curve |  |
| 97 | Collision risk |  |
| 98 | Signal violation |  |
| 99 | Dangerous situation |  |

**Table 4.1:** DENM alerts supported by the mobile app.

### 4.3.6 Message creation

This module is used to create DENMs, EVRSRMs, SPVSMs and VAMs.

- *buildDenm()*: function called by the DENM pop-up, described in subsection 4.2.2, that returns the DENM in JavaScript Object Notation (JSON) format, with the current position, **stationID**, **stationType** and selected **validityDuration**, **relevanceDistance** and cause and sub-cause codes;
- *createPreReservationEVRSRM()*: used by the EV reservation pop-up, described in subsection 4.2.5, to creates a pre-reservation EVRSRM with the type of charging and schedule;
- *createReservationEVRSRM()*: called after a successful pre-reservation to create the EVRSRM using the current time, energy amount and payment information;
- *createSPVSM()*: is used periodically, in OBU mode, to create a SPVSM with the current location and sensor readings;
- *createVAM()*: used periodically, in VRU mode, to build the VAM with the current location.

### 4.3.7 Message callbacks

When the MQTT client is created on the mobile application, a callback named *onMessage()* is passed as an argument. This callback enables the automatic reception of all messages on the topics that the application subscribes to, according to the description provided in subsection 3.2. There is a *onMessage()* function for every message type, for instance *onDenmReceived()* for DENMs. These functions are declared inside the map screen, which gives the function access to the variables inside its scope, and are declared using *useCallback()*, which is a React hook to memorize a function and that only redefines it when a dependency, passed as an argument to the hook, changes. The callbacks first determine the received message type by using the information of the topic on which the message was received, and then attempt to parse it. If the parsing is successful, it calls the specific function that processes and stores the information necessary for the visualization of the event in a React ref variable declared inside the map screen. Some of these functions, the ones pertaining to the message types presented in the history modal (subsection 4.2.4), also save the information in the permanent storage of the device with the *addMessage()* function described in subsection 4.3.3.

The different *onMessage()* functions correspond to all the types of message supported by the application and are the following:

- *onCamReceived()*: stores the **stationID**, **stationType**, coordinates, path history and speed of the vehicle. It creates a callback, called three seconds later, that checks if there were new CAMs received with the same **stationID** and, if not, deletes the message. If the CAM's **stationID** matches the user ID configured in the application, the coordinates are used to set the user location;
- *onCpmReceived()*: calculates the coordinates of the perceived objects and saves them along with the classifications given;
- *onDenmReceived()*: stores new DENMs, a flag to know if it was seen by the user and creates the callback, with the *validityDuration*, that later deletes the message when

the event expires. It is also responsible for updating, canceling and negating existent DENMs.

- *onEvcsnmReceived()*: stores the message;
- *onEvrsrmReceived()*: notifies the user about the received message contents and, when a pre-reservation response with status success is received, creates and sends a reservation request;
- *onHDMapsReceived()*: stores the message;
- *onHearbeatReceived()*: stores the message;
- *onIvimReceived()*: stores the message and creates a timeout with the duration of the (optional) **validTo** field that makes the marker disappear. It also stores the message in permanent storage;
- *onSaemReceived()*: calculates and stores the points of the polygon and arrow coordinates of each tolling zone;
- *onTpmReceived()*: creates notifications for TPM requests and replies, including a callback for each request to later check if a reply is received, in order to inform the user if the transaction succeed or failed. It also stores the messages;
- *onVamReceived()*: stores the message.

### 4.3.8 Icons

The application uses a lot of icons, most are used to display DENMs, detailed in subsection 4.3.5, but there are also others employed to show different vehicle and VRU classification types. Vehicles are classified either by themselves, based on the **stationType** field of CAMs, or by an RSU, in the perceived objects list of CPMs. VRUs can also classify themselves, using the **stationType** in VAMs, or be classified as such in CPMs.

Table 4.2 displays the icons employed to depict CAM and VAM **stationType**s. The icons appear in black when the user is the message transmitter and in green when sent by others.

Tables 4.3, 4.4, 4.5 and 4.6 show the icons used for all possible classifications of perceived objects in CPMs. The objects are first classified into one of four classes:

- **Humans**: classification of many types of VRUs:
- **Animals**: no subclass classification exits in this class and the only possible classification is unknown;
- **Vehicles**: classification of all types of road vehicles;
- **Others**: classification of RSUs and unkown objects;

| Station Type | Description | Own Icon | Others Icons |
|---|---|---|---|
| 0 | Unknown |  |  |
| 1 | Pedestrian |  |  |
| 2 | Cyclist |  |  |
| 3 | Moped |  |  |
| 4 | Motorcycle |  |  |
| 5 | Passenger Car |  |  |
| 6 | Bus |  |  |
| 7 | Light Truck |  |  |
| 8 | Heavy Truck |  |  |
| 9 | Trailer |  |  |
| 10 | Special Vehicles |  |  |
| 11 | Tram |  |  |
| 15 | RSU |  |  |

**Table 4.2:** CAM and VAM icons for each station type.

| Station Type | Description | Icon |
|---|---|---|
| 0 | Unknown |  |
| 1 | Pedestrian |  |
| 2 | Person In Wheelchair |  |
| 3 | Cyclist |  |
| 4 | Person With Stroller |  |
| 5 | Person On Skates |  |
| 6 | Person Group |  |

**Table 4.3:** CPM human class icons.

| Station Type | Description | Icon |
|---|---|---|
| 0 | Unknown |  |

**Table 4.4:** CPM animal class icons.

| Station Type | Description | Icon |
|---|---|---|
| 0 | Unknown |  |
| 1 | Moped |  |
| 2 | Motorcycle |  |
| 3 | Passenger Car Car |  |
| 4 | Bus |  |
| 5 | Light Truck |  |
| 6 | Heavy Truck |  |
| 7 | Trailer |  |
| 8 | Special Vehicles |  |
| 9 | Tram |  |
| 10 | Emergency Vehicle |  |
| 11 | Agricultural |  |

**Table 4.5:** CPM vehicle class icons.

| Station Type | Description | Icon |
|---|---|---|
| 0 | Unknown |  |
| 1 | RSU |  |

**Table 4.6:** CPM other class icons.

CHAPTER 5

# Evaluation

This chapter describes the usability and performance tests of the developed mobile application, the corresponding results, and the continuous integration of visual and functional features.

## 5.1 Usability

One of the most important aspects for user retention and satisfaction is how easy and intuitive it is to perform the many actions required by the application use cases. In this sense, two techniques were employed to evaluate usability. Firstly, a heuristic-based evaluation was performed by people directly involved in the development of the application, using **Nielsen's Heuristics**, in which the most concerning usability issues were detected and corrected. After that, an evaluation based on the **Thinking Aloud** technique, an empirical method, was carried out with a wide variety of first-time users.

### 5.1.1 Nielsen's heuristics

Nielsen's heuristics consist of ten broad rules of thumb that help identify and categorize usability problems.

| | |
|---|---|
| 1 | Visibility of System Status |
| 2 | Match Between the System and the Real World |
| 3 | User Control and Freedom |
| 4 | Consistency and Standards |
| 5 | Error Prevention |
| 6 | Recognition Rather than Recall |
| 7 | Flexibility and Efficiency of Use |
| 8 | Aesthetic and Minimalist Design |
| 9 | Help Users Recognize, Diagnose, and Recover from Errors |
| 10 | Help and Documentation |

**Table 5.1:** The 10 Nielsen heuristics.

After becoming familiar with the heuristics evaluators, people directly involved in the development of the application, are tasked with finding and classifying usability issues using the heuristic that relates most to it and a usability severity, a scale with a range of zero to four, as described in [64]. Each evaluator must first do the task alone and then evaluate the list of all issues found by other evaluators, a process explained in [65]. Figure 5.1 shows the proportion of usability problems found as a function of the number of evaluators.

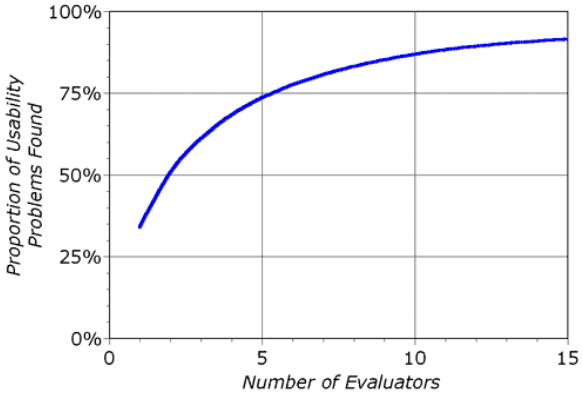


**Figure 5.1:** Proportion of usability issues found in relation to the number of evaluators [66].

The number of evaluators recommended by the Nielsen Norman Group is between three and five [66]. The author also concludes that there is not much to gain by further increasing this number. An elaborate model for the estimation of costs of the preparation, execution and analysis of results was created and tested, on real projects, and then compared with the proportion of usability issues found in function of the number of evaluators 5.1. This comparison results in the graph of the ratio of costs over benefits in relation to the number of evaluators where 4 is the optimal number and where benefits are 62 times greater than costs.

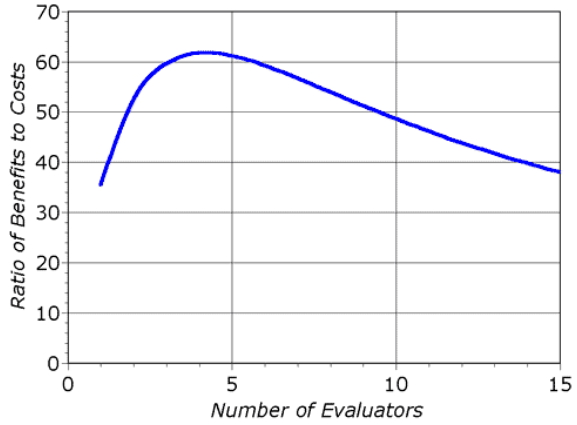


**Figure 5.2:** Cost/benefit ratio as a function of the number of evaluators [66].

In this dissertation, four evaluators, the ones most connected to the development of the application, completed the test. The evaluation was conducted by presenting each of the evaluators with a document (appendix B). The document starts with an header that explains the following instructions:

- The format to be used when filling the document;
- A link to an article [64] that names the heuristics and describes them;
- The usability evaluation scale, also explained in the same article.

The body of the document also includes a title for every heuristic and space in which the evaluator must place issues related to it and rate them. The documents of all evaluators were then merged and the result was presented to have the evaluators rate the issues found by others. The final document (appendix B) includes all the problems found and the rating given by each evaluator.

Figure 5.3 was created using the methodology explained in [66]. Each row represents one of the four evaluators, while each column represents one of the problems found. Rows are ordered by the success of the evaluator, which corresponds to the amount of identified problems, and columns are ordered by the easiness of finding that specific problem, i.e. the number of evaluators that managed to find the given issue. A square in row $i$ and column $j$ is black if the evaluator $i$ found the problem $j$ and white otherwise. Inside each square is the severity value given by the evaluator to the specific problem. The evaluators found twenty problems and at least one using each heuristic. The graphic was created using a *Python 3* script that parses the evaluations and automatically creates it [67].

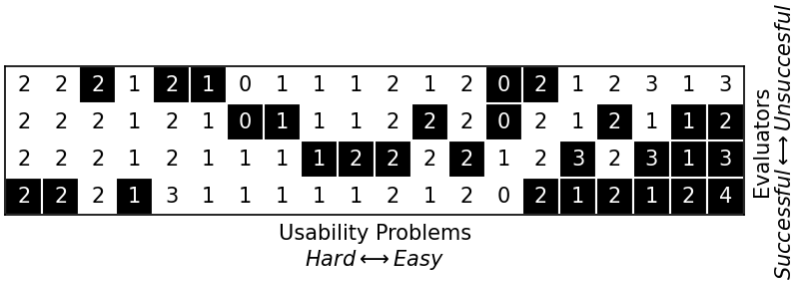


**Figure 5.3:** Evaluation results of usability tests using Nielsen's heuristics.

The highest rated problem had an evaluation of **4**, which corresponds to a major usability problem with a high priority to be fixed. It was the only one with this severity and was the result of including test code in a production build. The issue was found using the 5$^{\text{th}}$ heuristic - **error prevention**.

Two other problems were given an evaluation of **3** and were found using the 5$^{\text{th}}$ and 8$^{\text{th}}$ heuristics:

- **#5: Recognition rather than recall**
  When connected, the blue button which is now a circle, wasn't labeled to indicate that it serves to create DENM alerts. A label with the text "Send Alert" was added to help guide the users;
- **#8: Aesthetic and minimalist design**
  Navigating the settings to find the ones related to a specific user mode is confusing. The solution was to explicitly group these settings and placing them inside accordions, a type of component that looks like a dropdown menu when closed but allows, when opened, to show the hidden settings.

A total of twelve problems were classified with a severity of **2** and five were rated with **1**. Three issues of severity **2** and two rated **1** were either indirectly resolved or not fixed at all due to:

- Additional proprietary messages are required to implement the automatic detection of the vehicle's *stationID* connected to the user's smartphone;
- Automatically connecting the application would make it even harder to recognize the user modes;
- No better solution for the RSU icon was found;
- The library, used to connect to the MQTT broker, only provides a generic error;
- Having a link to a user guide was deemed unnecessarily unfriendly to users and a better way to help the user understand what each setting does was implemented: question mark icons next to each setting that reveal a help message.

All other issues were resolved before the evaluation described in the next subsection 5.1.2 was performed.

### 5.1.2 Thinking aloud

Thinking aloud is an empirical and observational method of conducting an usability evaluation. It's flexible and inexpensive but carries the risk of influenced user behavior. The purpose of this test was to evaluate how easy completing the most essential tasks is for new users.

It was performed by ten Portuguese users with ages between 10 and 79 years old. The only personally identifiable information collected, with the proper permission, was the first name and year of birth. This information is relevant since older people have been introduced to smartphones and mobile applications in a later stage of life and, as confirmed by the questionnaire results, are the ones that tend to show the most difficulty and time spent on each task. This method makes use of two questionnaires, one for the observer (appendix D) and another for the participants (appendix C):

- **Observer**: someone deeply familiar with the application that takes notes, for each question, of the time employed to complete the task, eventual user comments and whether the user:
  - Failed;
  - Answered incorrectly;
  - Was lost;
  - Asked for Help.
- **User**: composed mostly by difficulty ratings, from 1 to 10 (1 being very easy and 10 very difficult), and some short text responses that ensure the user memorized the information presented in prior tasks. All participants employed Android-based devices for the test.

The following tasks were presented to the users:

- **1. Open the settings and set the application language to your preference. Indicate your preference:**
  Most of the users had a preference for Portuguese while one chose English, which confirms the usefulness of having support for multiple languages.

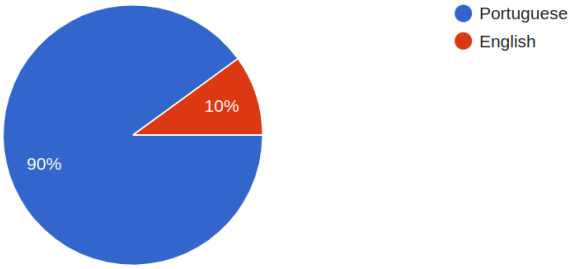


**Figure 5.4:** User language preference.

  - **1.1** Which settings do you have the most difficulty understanding what the setting means? Please answer at most 5.
    Most users (*6/10*) answered that they didn't understand most settings even after noticing and reading the help texts. Most acronyms used are not directly translatable to Portuguese and that the help text should be more comprehensive.
- **2. Return to the map screen and connect the application in TCC mode. How easy was it to perform the action?** (Average: 5.3/10, Median: 5.5/10)
  Seven users commented that the position and size of the user mode selector, described in subsection 4.2.12, are not enough for them to perceive it as the way to change the user mode. Furthermore, after explaining the user modes, the users proposed adding the text of the acronym below the icon, since two of them chose the vehicle icon when the goal was to select the TCC mode, represented by the icon of a monitor.

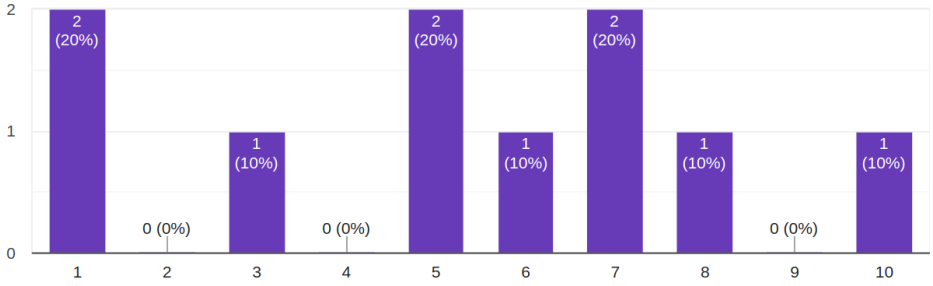


**Figure 5.5:** Connect in TCC mode.

- **3. Locate an RSU within the Aveiro district and choose any vehicle perceived by it. What is the speed of the vehicle? How difficult was the task?** (Average: 4.3/10, Median: 3.5/10)
  The average rating was 4.3 but most users thought that the icon used for the RSU also contained a vehicle. The icon is the result of the combination of the SVG used for the vehicle with another of radar waves cut to only show the first quadrant. In subsection 5.1.1 the evaluators also pointed out the same issue, and it is something that must be improved upon.
- **4. Create a slippery road alert with a relevance distance of 500 meters and a validity duration of 2 minutes. Classify the difficulty.** (Average: 3.3/10, Median: 2.5/10)

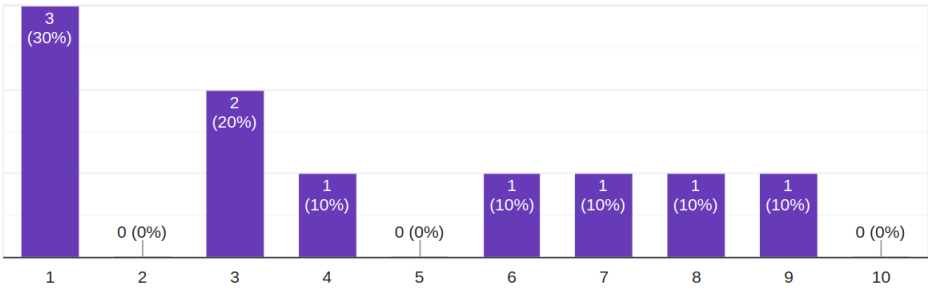


**Figure 5.6:** Find an RSU and the speed of a vehicle perceived by it.

Most of the users completed the task easily, but one reported that the **Connect Button** (subsection 4.2.1), which displays the *Send Alert* text, did not catch their attention because the text size was too small. Another user, due to lack of attention, did not change the type of DENM.

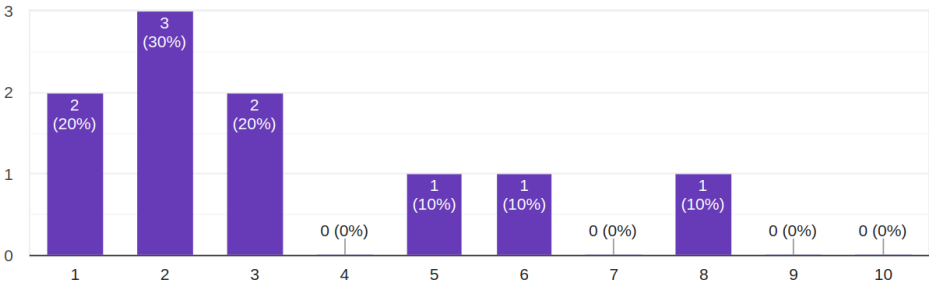


**Figure 5.7:** Create a DENM event.

  - **4.1 Update the event relevance distance to 200 meters. Classify the difficulty.** (Average: 3/10, Median: 2.5/10)
    The main issue found in this task was that, due to the third task, the map was placed in the Aveiro region. All tests were carried out in Viseu, a nearby city, so the user did not see the DENM event icon on the map and had no way to know that it was placed at the user's location. Most of the users noticed the button to center the location on their own and took the initiative. One of them opened the history component and clicked on the event which placed the map in the event position. The map location should be automatically centered on the place where the event is created, and, eventually, there should be a way to drag the event to the desired location before sending.

- **5. Open the history of the events received by the application and delete the event created in the prior two questions. Classify the difficulty.** (Average: 2.2/10, Median: 1/10)
  This task was generally easy, with only one user failing because he clicked the "delete all events" button. Note that there was only one event in the history component. Another user could not read the text "history", placed below the history icon, and did not associate the icon with the history. This user asked for help. Perhaps the choice of controlling the entire UI font and icons size should be part of a welcome guide to the

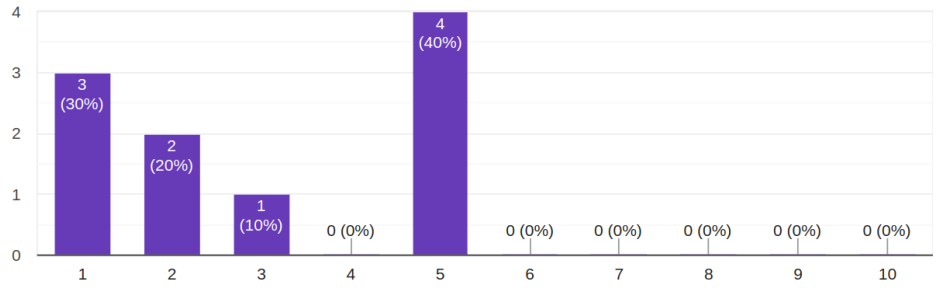


**Figure 5.8:** Update the DENM event.

application. Most of the users stated that selecting accessibility settings before using the application would have a great impact.

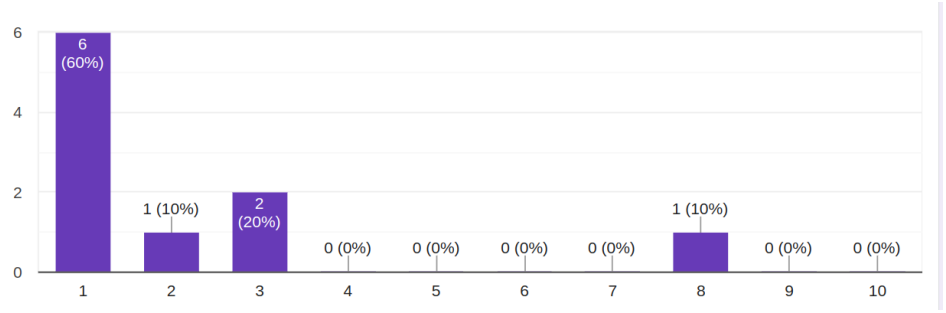


**Figure 5.9:** Delete the DENM event in the history component.

- **6. Navigate to the sensors screen and observe the sensors of the mobile phone. Classify the difficulty.** (Average: 2.1/10, Median: 1/10)
  The task was solved by 9 of the 10 participants and a 2.1 average difficulty rating was given. One user, who had completed the first task with ease, did not notice the button on the bottom tab of the sensors screen and asked for help. This user stated that scaling the icons and font size in the UI, before starting to use the application, would eliminate the problem.

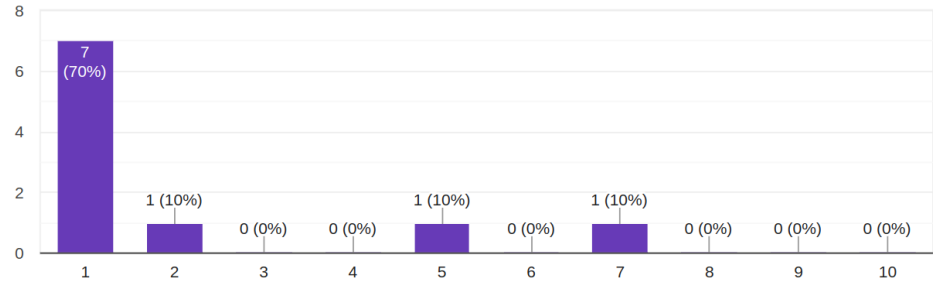


**Figure 5.10:** Visualize the mobile phone sensors.

**7. Users were instructed to disconnect the application before proceeding to this task. Configure the connection to the vehicle in the settings using the following information:**

- **Host**: 192.168.94.35

- **Port**: 8081
- **Username**: testUser
- **Password**: testme
- **Custom Station ID**: 7301

**Go back to the map and connect the application in OBU (vehicle) mode. Classify the difficulty of the settings configuration.** (Average: 4.4/10, Median: 4.5/10)

This task was considerably harder than most, with a 4.4 average. The worst issue was when the user tried to configure the settings while still connected, which does not work and has no alert for the user. A lot of users did not understand the button that hides and shows the text input to customize the *stationID* of the application and then, after returning to the map, considered the task completed without connecting. Most of the users who successfully connected did not associate the OBU mode and settings with a car and chose the wrong user mode. In retrospective, this task was too long and should have been split into smaller ones. Since some users did not know English, the form had to be verbally translated to them which made the task even more difficult.

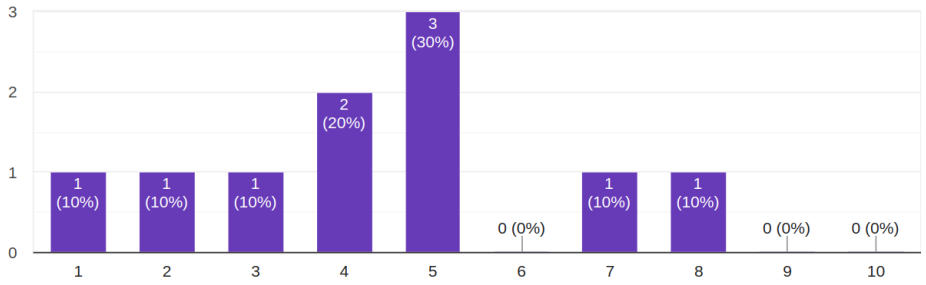


**Figure 5.11:** Configure the OBU settings and connect to it.

- **8. Create an EV reservation by selecting one of the available charging station icons on the map. Set the time of arrival to 30 minutes from now and the charging session duration to an hour. Select the highest amount of energy required. Classify the difficulty.** (Average: 3.8/10, Median: 3/10)

  The most recurring critic given was related to the Android implementation of the date picker, described in subsection 4.2.6. The picker is the native picker provided by the OS and is hard to use. The alternative would be to choose another library that implements a date-time picker. A user asked for help when selecting the type of charger in the charger selection pop-up, as shown in figure 4.18, not realizing that a choice had to be made.

  - **8.1 After a successful reservation cancel it. Classify the difficulty.** (Average: 2.1/10, Median: 1/10)

    During four out of ten evaluations, the reservation process stopped working and so one user classified this task with a difficulty rating of 10, due to the icon to manage the reservation not being present. The same was true for the other three

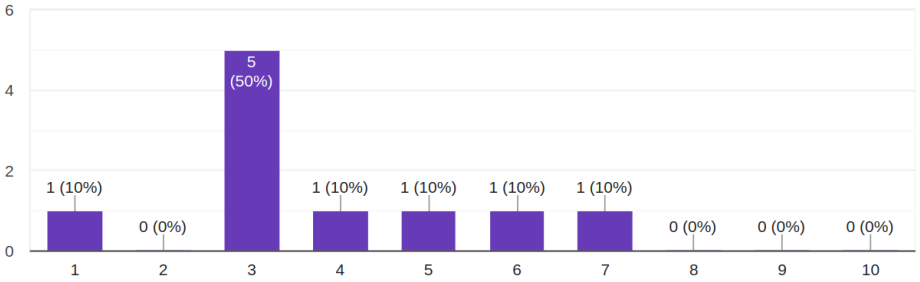


**Figure 5.12:** Create an EV reservation.

users that classified it with 5. Users in the correct conditions to answer the test classified it with difficulty 1 and said that the icon blinking was a really good way to capture their attention.

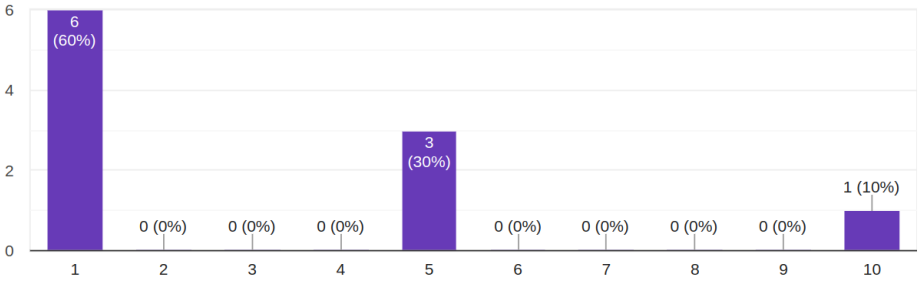


**Figure 5.13:** Cancel the EV Reservation.

- **9. Navigate back to the sensors screen and observe the vehicle sensors. How difficult was the task?** (Average: 2.3/10, Median: 1/10)
  One user, the one who classified this task with a difficulty rating of 9, pointed out that having the tab on the bottom, to switch between screens, and the tab on the top to switch between the different sensors is not logical. Otherwise most users considered it a very easy task.

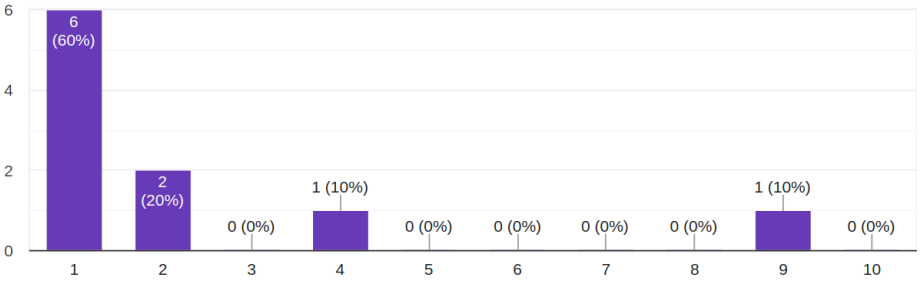


**Figure 5.14:** Visualize the vehicle sensors.

- **10. How would you rate the app?** (Average: 3.8/5, Median: 4/5)
  The last task was to give an overall rating to the application, between 1 and 5, while taking into account that these are first time users that do not regularly use or have never used applications with the same characteristics and goals as the one being evaluated.

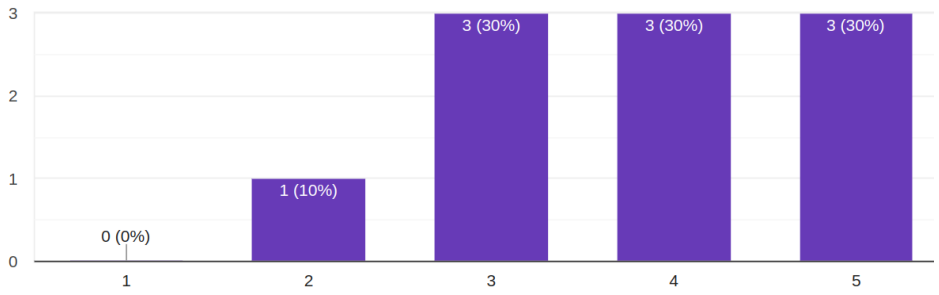


**Figure 5.15:** Rate the application.

– **10.1 How would you improve the application?**
  This task was an open question with the space of a paragraph for the evaluator to provide the response. Some users did not write any answer, but most contributed to the problem assessment. Most of the changes are related to accessibility and the need for help regarding the introduction of acronyms and the role of the different user modes.

The observer noted the time each user took to complete each task and the results were plotted in a box plot where each box represents the Interquartile Range (IQR), with the median time indicated by a red line inside the box. The lines outside the box represent the minimum and maximum values within *first quartile minus 1.5 * IQR* and *third quartile plus 1.5 * IQR*, respectively. Outliers are denoted by circles outside these values ranges. The results are shown in figure 5.16.

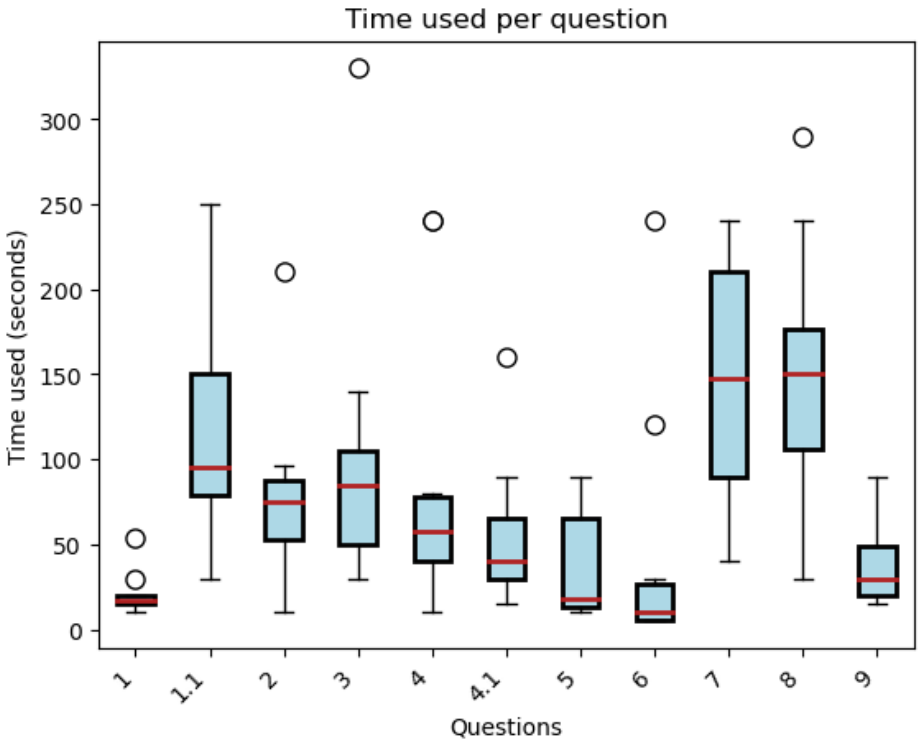


**Figure 5.16:** Box plot of the time to complete each task.

Since the evaluators' dates of birth varied from 1944 to 2013, the older people were outliers in most tasks considered hard, as expected. The time used by each participant to complete the usability test by age is represented in figure 5.17.

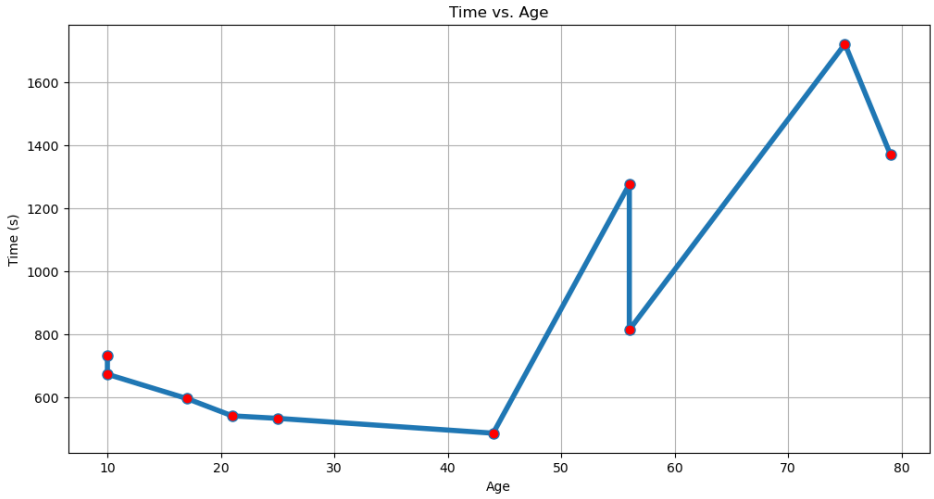


**Figure 5.17:** Graph of the time to complete the usability test by age.

The big discrepancy between the two users inside the 50-60 years old interval is due to the fact that the younger user has some visual deficiencies and is not so well versed in using their smartphone and applications. Young adults had the fastest times performing the tasks.

## 5.2 Performance

Android Debug Bridge (ADB) was used to measure Central Processing Unit (CPU) and RAM usage of a production build. Finally, a network analysis was also performed to show the download and upload traffic of the application.

These tests were carried out on a Oneplus 8 PRO with an 8-core Snapdragon 865 CPU and 12 GB of RAM. The application was first tested in OBU mode using a variable number of vehicles generating CAMs (1, 5, 25 and 125) published in a local broker running on the OBU. Each vehicle was transmitting ten CAMs per second at a fixed rate. To simulate a configurable number of vehicles transmitting CAMs, two *Python 3* scripts were created:

- **getCAMTrip()** records the path of any vehicle until it reaches the same initial location (circular routes) and outputs it to a file, in order to collect a list of realistic geographical coordinates;
- **createCAMs()** uses the file created by the prior function to send CAMs with a period of 100 ms. In addition, it enables the generation of a configurable number of simulated vehicles for CAMs transmission, which are created with an offset relatively to the originally recorded messages.

The second test was performed using the VRU and TCC user modes. This test relied on two other *Python* scripts:

- **recordCCAM()** records all messages present in the cloud MQTT broker, as well as the associated topics and timestamps, for one-hour interval, which corresponds to the maximum duration of any test done;
- **emulateCCAM()** uses the file created by the prior function to send the exact same message load to a local broker where the tests were conducted.

The results graphs are further optimized for visualization by using a moving average of 300 frames. This adjustment facilitates the interpretation of functions with lots of data points in a graph. In a 5-minute recording comprising 18000 frames (60 fps), the technique replaces the value of each point for the average of the last 300 points, i.e. the last 5 seconds, greatly reducing the variance and facilitating the visualization of multiple tests in a single graph. In the first 299 frames, there are not enough preceding values to calculate the average using 300 data points, thus the average is calculated with all prior frames, meaning that the first points represent an average value with less number of frames.

### 5.2.1 ADB Performance Monitor

Processor and memory performance were measured using the ADB tool, on a phone with no other applications opened and a production build, thus providing a reliable way to obtain performance metrics, in this case CPU and RAM usage. The CPU usage is represented as a percentage of the work that a single processor can do which, for the 8-core device used, means that the maximum theoretic usage is 800%. Two *Python 3* scripts were used:

- *recordADBCpuAndRam* which for a specified amount of time, five minutes in the tests done, collects the maximum amount of samples of CPU and RAM usage. An average of 3.4 real samples per second was obtained when using the *top* command in the ADB shell. In order to allow the reuse of some data processing scripts, the collected samples were interpolated to have 60 values per second.
- *cpuAndRamGraphs* creates the plots for both the CPU and the RAM usage over time. It combines the information of all tests, with the varying amount of vehicles generating CAMs, into two graphs (figures 5.18 and 5.19).

For the 125 CAMs scenario, the trend of the RAM usage over time looks similar to a memory leak given the excessive re-rendering of so many objects. This is due to the fact that the JS thread creates the new markers and JS objects for the map refresh, not having enough time to properly perform the garbage collection task before rendering the new objects. This is an inevitable result of having such a high refresh rate and receiving too many messages. A simple solution could be to lower the refresh rate when there are too many markers to be rendered. It is important to note that Hermes, the JS engine used, has an aggressive strategy that makes it so that, if the JS thread is being used too much, the garbage collector will be called less often. This is true until a certain threshold of RAM usage, at which point the garbage collector is forced to clean up unused memory and the JS thread is extremely throttled because of this. For this reason, the initial Frames Per Second (FPS) rate of the application is high, alongside responsiveness, and then decreases over time until it stabilizes. This is visible in figures 5.20 and 5.21 that correspond to a one-hour test with 125 vehicles generating CAMs.

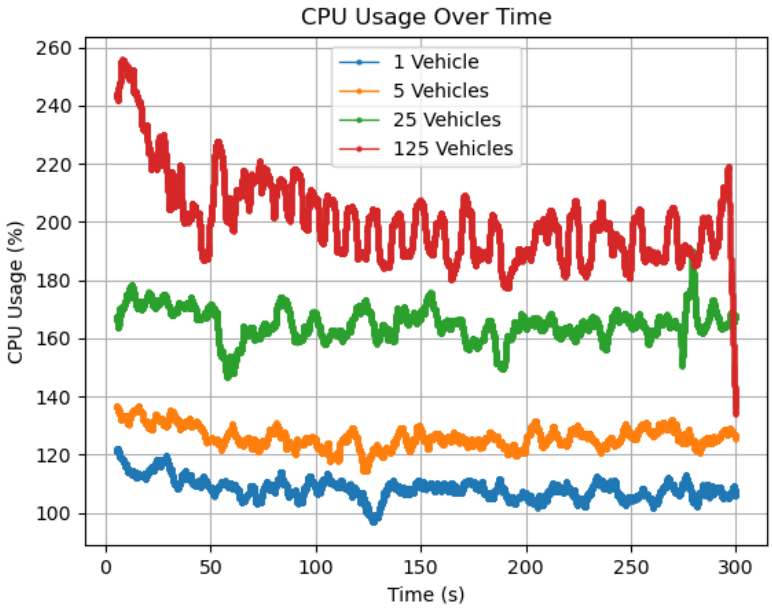


**Figure 5.18:** CPU usage over time in OBU mode (Hermes JS engine).

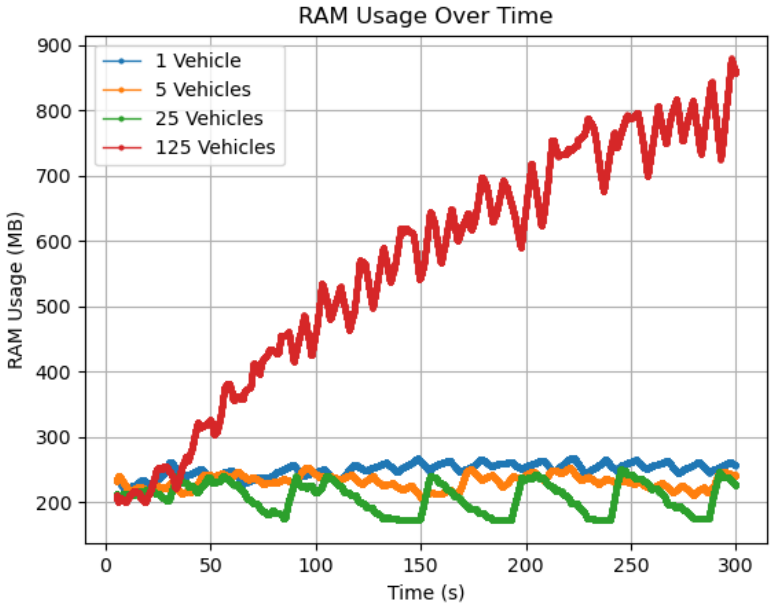


**Figure 5.19:** RAM usage over time in OBU mode (Hermes JS engine).

During these tests there was no user interaction with the screen so that both the UI thread, responsible for receiving touch events and triggering the appropriate responses, and JS thread are able to be completely used by the test. The application becomes unresponsive and after some time, at around 600 s, the whole application stabilizes with lower CPU and RAM usage, but unusable.

Throughout the performance evaluations many optimizations were implemented and compared. The most notable ones were: animated markers with a lower re-render frequency;

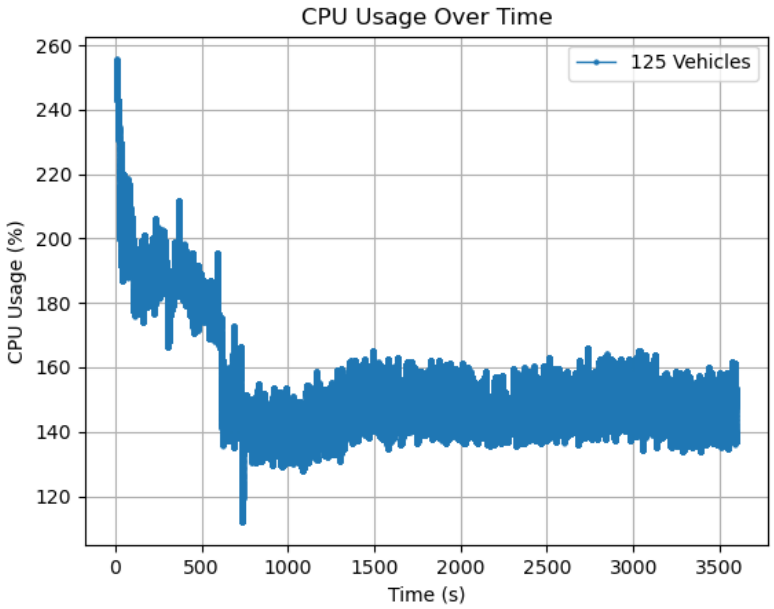


**Figure 5.20:** CPU usage over time in OBU mode for one-hour test with 125 vehicles generating CAMs (Hermes JS engine).

not rendering markers outside the users view; memoizing components and value; and, the last one tested and most impactful one, was to use a different JS engine, an option not commonly taken by the Expo community. The engines available on Expo SDK 48 are Hermes, JSC and V8. Switching to JSC is as easy as adding a key-value pair to a configuration file, while setting up V8 involves installing *react-native-v8* and the Just In Time (JIT) compiler, NPM packages, and the corresponding Expo config plugin. The one-hour test was repeated with both JSC (figures 5.22 and 5.23) and V8 engines (figures 5.24 and 5.25).

JSC performed better than Hermes, at least in terms of RAM usage, but V8 is without a doubt the JS engine with the best results. Just like Hermes, JSC presented a behavior similar to a memory leak and ended up throttling the application. Since V8 performed well with 125 vehicles generating CAMs, by keeping the CPU and RAM usage at stable values, another one-hour test was performed, this time with 625 vehicles generating CAMs, and the results are shown in figures 5.26 and 5.27. With these many markers, the V8 engine also started to present a behavior similar to a memory leak. In the last 5-minute interval of the test, the connection to the MQTT broker was lost.

The second test, which involves a wide variety of messages from the central broker and evaluates the VRU and TCC modes, showed that the VRU mode is more CPU intensive than TCC mode due to the periodic sending of VAMs, visible in figure 5.28, and that in the VRU mode with quadtree 14 the amount of RAM used is considerably lower due to receiving and storing way less messages, see figure 5.29.

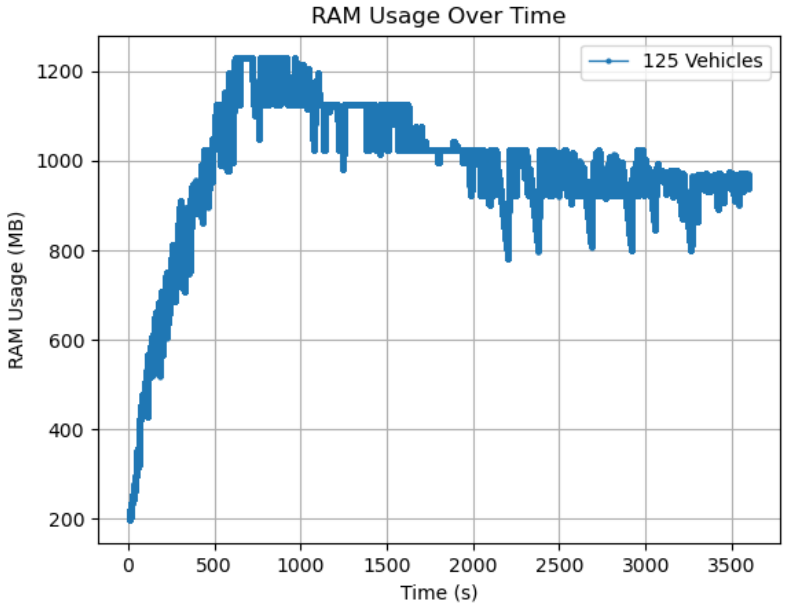


**Figure 5.21:** RAM usage over time in OBU mode for one-hour test with 125 vehicles generating CAMs (Hermes JS engine).

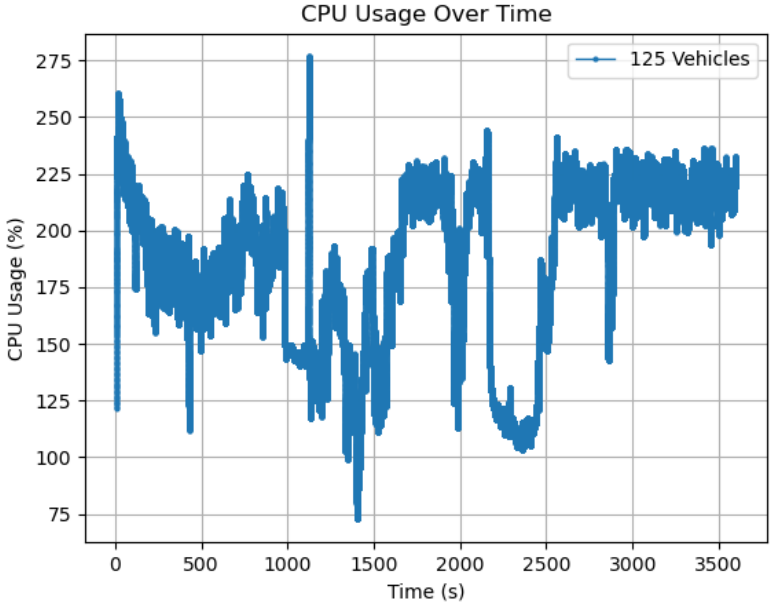


**Figure 5.22:** CPU usage over time in OBU mode for one-hour test with 125 vehicles generating CAMs (JSC JS engine).

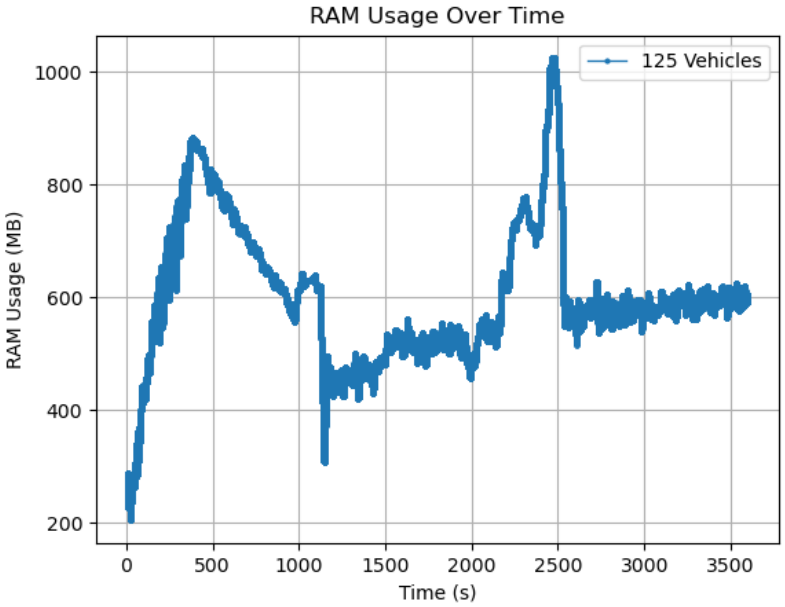


**Figure 5.23:** RAM usage over time in OBU mode for one-hour test with 125 vehicles generating CAMs (JSC JS engine).

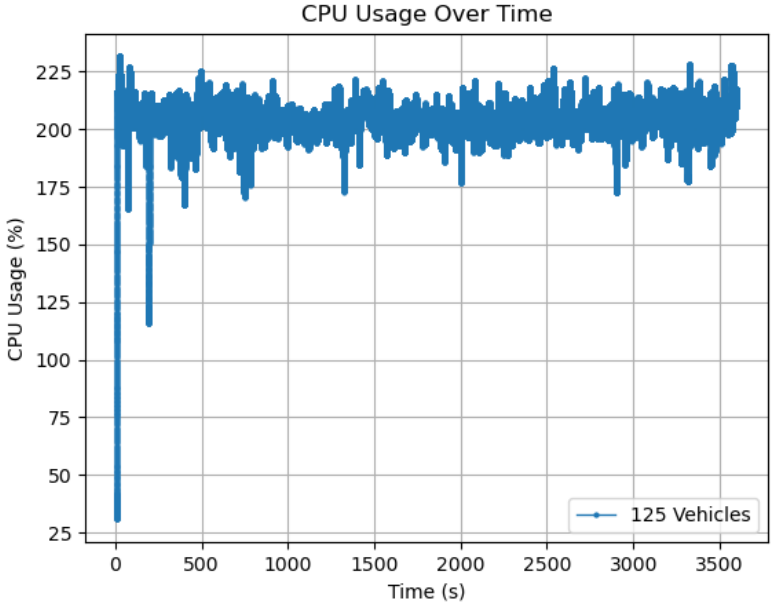


**Figure 5.24:** CPU usage over time in OBU mode for one-hour test with 125 vehicles generating CAMs (V8 JS engine).

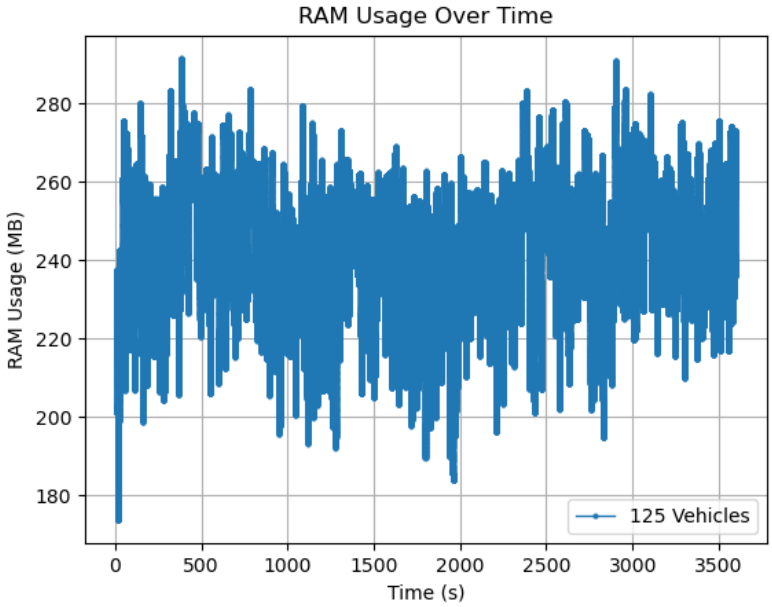


**Figure 5.25:** RAM usage over time in OBU mode for one-hour test with 125 vehicles generating CAMs (V8 JS engine).

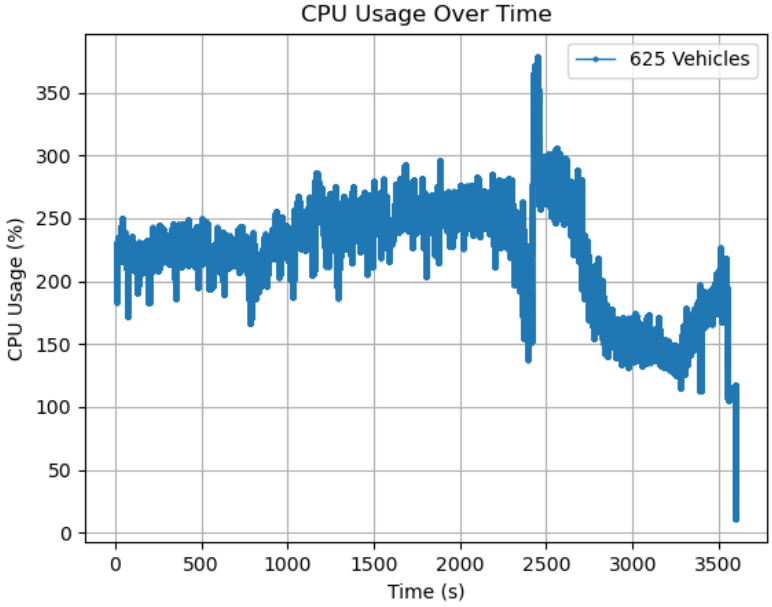


**Figure 5.26:** CPU usage over time in OBU mode for one-hour test with 625 vehicles generating CAMs (V8 JS engine).

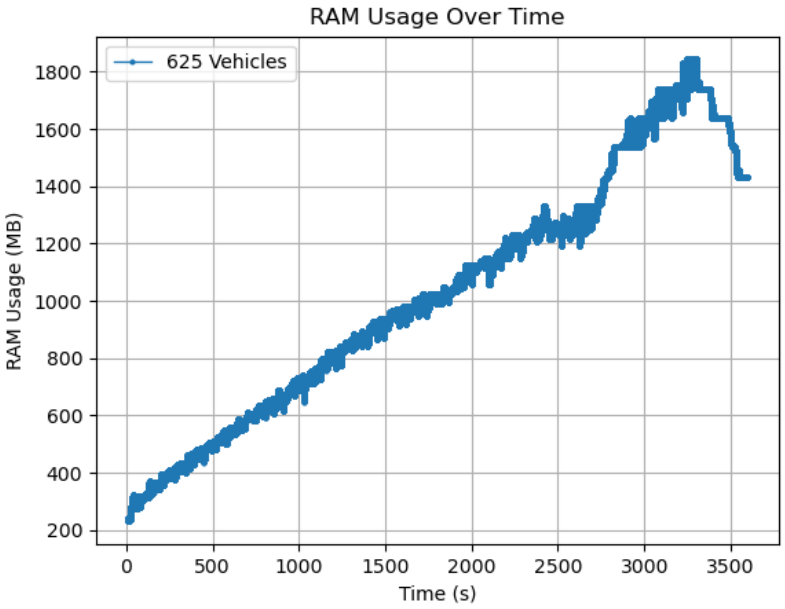


**Figure 5.27:** RAM usage over time in OBU mode for one-hour test with 625 vehicles generating CAMs (V8 JS engine).

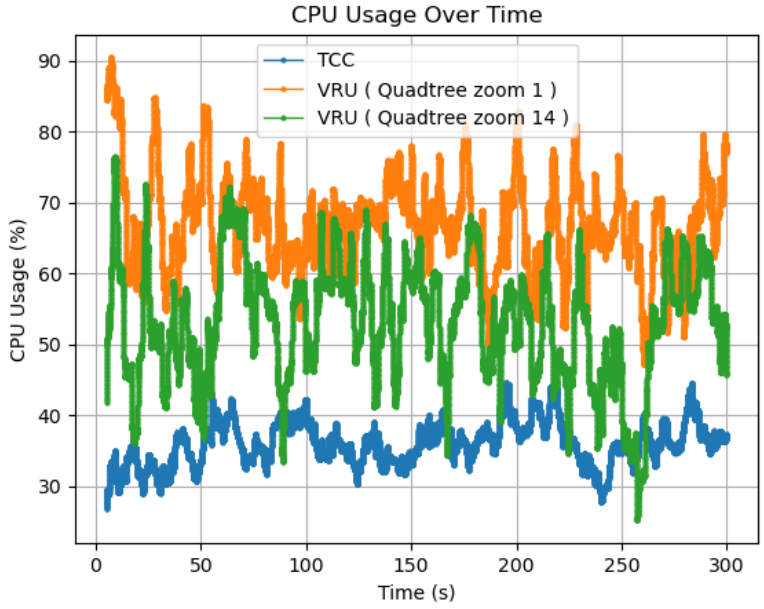


**Figure 5.28:** CPU usage over time in TCC and VRU modes (Hermes JS engine).

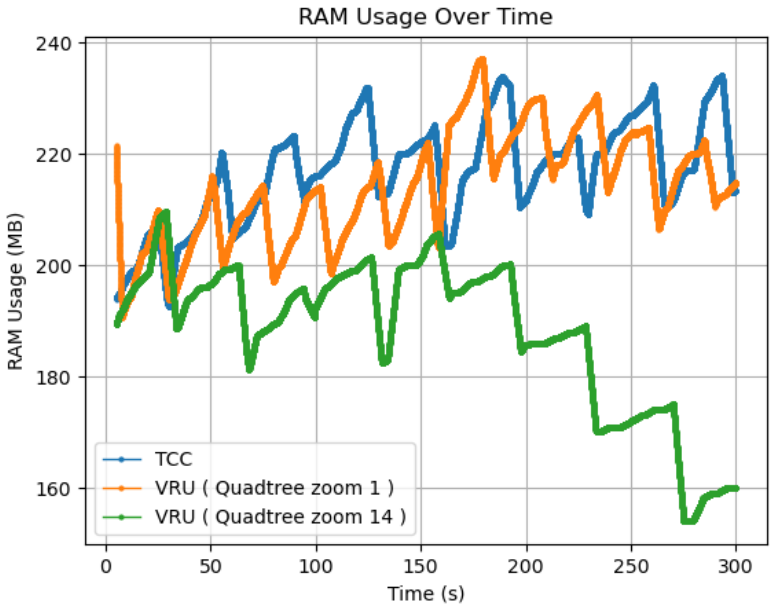


**Figure 5.29:** RAM usage over time in TCC and VRU modes (Hermes JS engine).

### 5.2.2 Network Analysis

Another important metric is related to the amount of mobile data consumed by the application, which has an impact on the generated network traffic and on the consumer bill. For this reason, the download and upload traffic was measured and plotted over time for each of the two tests, firstly with the variable amount of simulated CAMs to evaluate OBU mode and then with the replica of the central broker to test the VRU and TCC modes. For the first test, since all CAM MQTT packets contain the same payload length, given the way the protocol works, all packets will have the same size and the same is true for SPVSMs. CAMs are counted as download traffic, since they are received by the mobile application, and SPVSMs as upload because they are sent by the application. The messages were captured and the payload length of them was accumulated and saved, every second, using a *Python 3* script, to an object and afterwards to a file. These message flows are represented in figure 3.4. Figures 5.30 and 5.31 show the moving average of download and upload traffic, respectively, in MB for the first test.

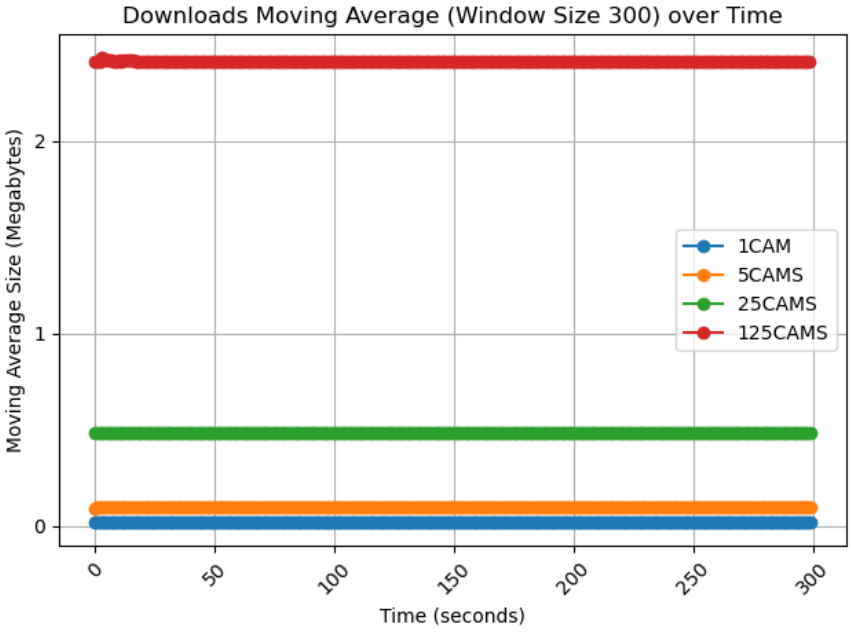


**Figure 5.30:** Download traffic (MB) in OBU mode over time.

SPVSMs are sent every second and the payload size of the message is not very large, according to the structure defined in the appendix A. As shown by the CPU and RAM usage, in subsection 5.2.1, the scenario with 125 vehicles generating CAMs, on both the JSC and hermes engines, throttles the application and, as expected, the number of messages sent significantly decreases as the application is unable to function properly.

Finally, to compare the TCC mode with the VRU one for both quadtree zoom levels 1 and 14, the second test was also performed. The results are shown in figures 5.32 and 5.33.

The VRU mode with quadtree zoom level 14 receives a very small number of messages, which has a positive impact on the mobile data consumption. The TCC mode does not upload

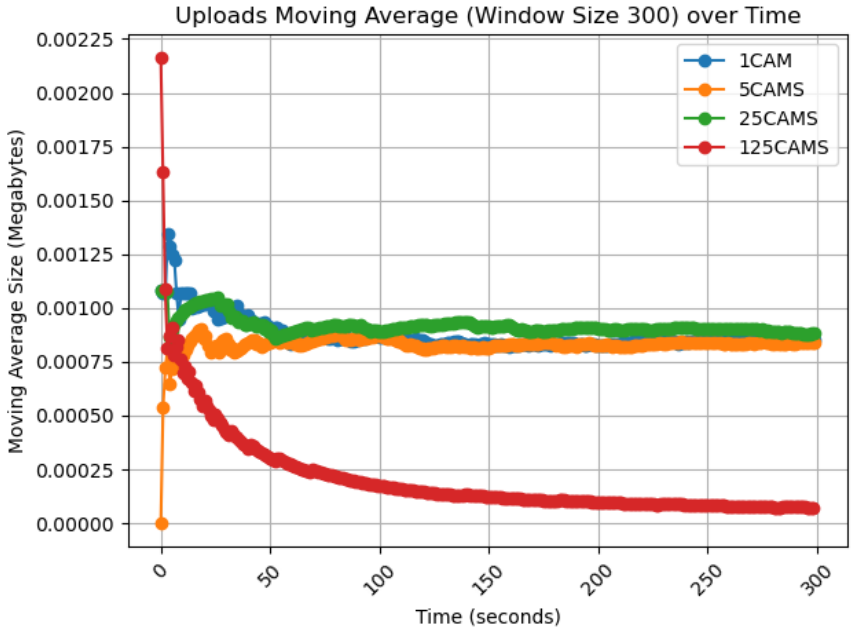


**Figure 5.31:** Upload traffic (MB) in OBU mode over time.

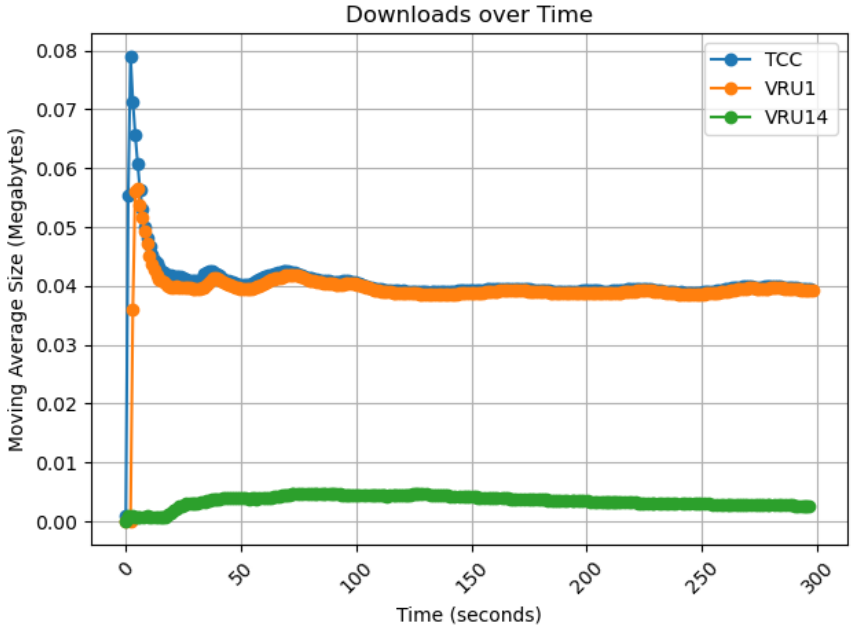


**Figure 5.32:** Download traffic (MB) in TCC and VRU modes over time.

any messages, while the VRU mode transmits a VAM per second.

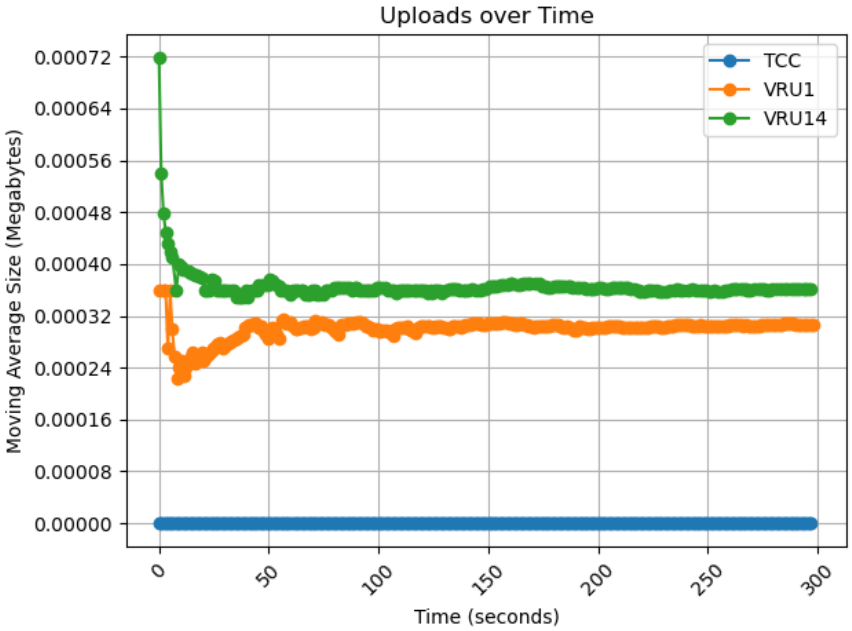


**Figure 5.33:** Upload traffic (MB) in TCC and VRU modes over time.

## 5.3 Sensor Results

Sensor data collection is very relevant in the current world with many machine learning techniques being able to detect dangerous situation and relevant information about the road conditions. A detailed description of the integration of this mobile application with an OBD-II is provided in [51], and the following results are essentially a subset of those presented in the article.

Adding the smartphone's sensors information to the data collection system has two main advantages: the mobile phone has access to sensors that the vehicle does not; and the sensors they do have in common generally have an higher precision on the mobile phone. Figure 5.34 shows a comparison between barometric pressure measurements in both the smartphone and the vehicle.

The scenario of detecting/inferring road holes or speed bumps can also be accomplished with the use of machine learning and the data of the the smartphone's gyroscope and/or accelerometer. In the following tests the vehicle went over a speed bump near the start and end of the trip. When comparing the X axis of the gyroscope with the Y axis of the accelerometer, the gyroscope seems to have a more pronounced change, when the vehicle goes over the bump, than the accelerometer. The test conditions and the sensor directions are shown in figure 5.35.

The accelerometer data, particularly on the Y axis, does not seem to have much change in the vertical direction (figure 5.36, thus it is not possible to identify the presence of the speed bump.

On the other hand, it is possible to identify some significant variation on the roll (X) axis of the gyroscope, as visible in 5.37, making the detection of the speed bump feasible by using

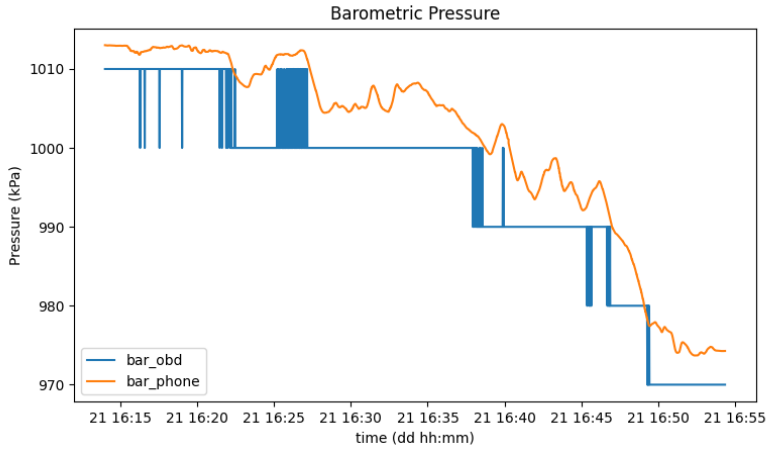


**Figure 5.34:** Barometric pressure measurements comparison between smartphone and vehicle sensors.

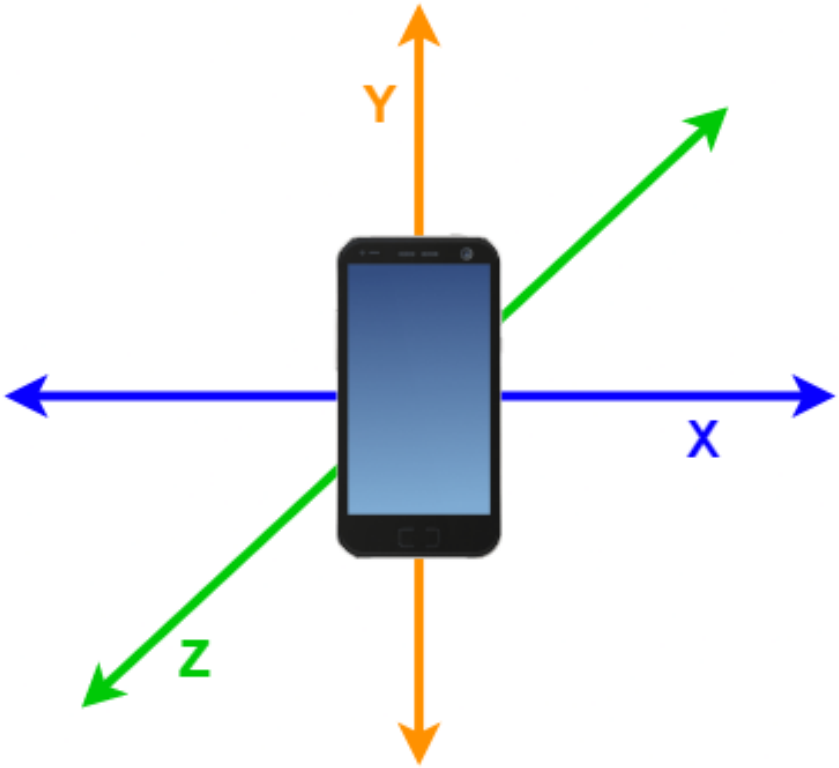


**Figure 5.35:** 3D coordinate system of the smartphone in the environment used for the test.

this sensor measurement.

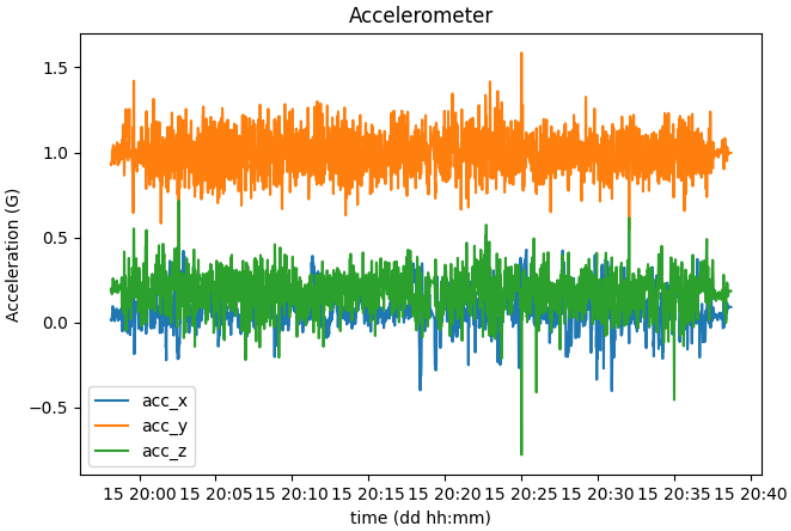


**Figure 5.36:** Smartphone's accelerometer data collected during a test trip.

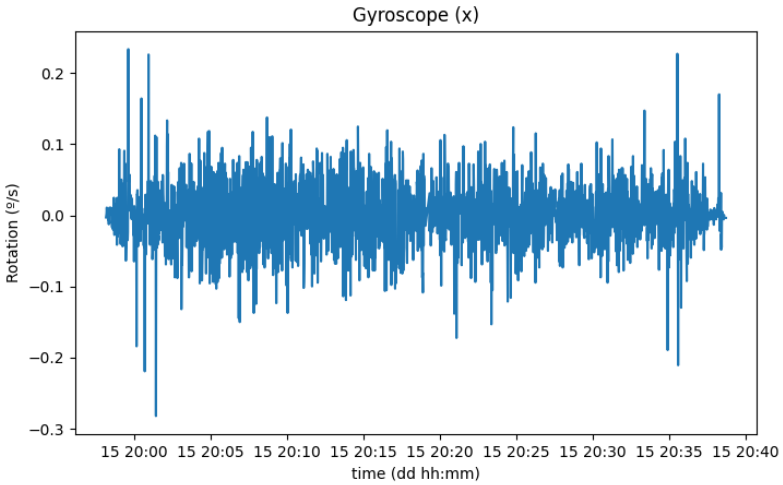


**Figure 5.37:** Smartphone's gyroscope data (X axis) collected during a test trip.

CHAPTER 6

# Conclusion

This chapter describes the main conclusions of this work, as well as some ideas to further improve it in the future.

## 6.1 Conclusions

The work described in this document pertains to an implementation of a C-ITS mobile application that supports vehicular communications (V2X) services and sensor data collection. There is a great emphasis on the implementation details that allow such a low-end computing device to reliably display and generate information, as well as interact with the existing ITS infrastructure.

A comprehensive review reveals that the majority of map applications available on the Android and iOS app stores lack integration with V2X communications and services, as well as essential features such as road and weather alerts. Moreover, these applications do not facilitate sharing user location and sensor data nor the interchange of said information in a standardized manner. Direct sharing of position by VRUs correlates significantly with enhanced VRU perception rates and overall road safety.

To enhance implementation, comprehensive evaluations were conducted focusing on both usability and performance. These evaluations proved invaluable in refining both the user experience and application performance. Usability tests unveiled that the ITS concept remains unfamiliar to many users. Additionally, it was found that most users could navigate the application effectively if provided with an introductory overview of the application's features and terminology prior to its use, thus proving to be highly beneficial. Performance evaluations assumed critical importance, given the anticipated expansion of C-ITS infrastructure, and delivered significant enhancements. The collection of sensor data is pivotal, serving as a foundational source of valuable information. This data plays a crucial role in empowering machine learning algorithms to accurately forecast road and weather conditions, enabling timely warnings to users.

## 6.2 Future Work

Several aspects of this work warrant further improvement, with performance standing out as particularly crucial. While notable enhancements have been achieved, an application designed for this purpose must demonstrate robust capabilities in handling a very high volume of ITS stations. The most relevant aspects to be improved are:

- Modify the callbacks that delete CAMs info, 3 seconds after no new message from the same vehicle is received, from individual **setTimeout()** callbacks to a single **setInterval()** that periodically analyses all CAMs in memory;
- Dynamic throttling the refresh rate when the load is higher than a certain limit. Since the architecture is stateless, it is possible to manipulate the refresh period programmatically.
- Given the application's capability to represent all ITS-S types, users should also have the flexibility to configure their own station types from those supported in VAMs or in CAMs. IVIMs should, like DENMs, support update, cancellation and negation of the event.
- It should incorporate introductory screens, which users can optionally skip, to provide clear explanations of the most essential features and configurations available. The application should also possess the capability to generate heat maps depicting the depth of water bodies, utilizing sonar information emitted by boats.
- Considering the ongoing V2X standardization efforts by prominent institutes, such as ETSI and Society of Automotive Engineers (SAE), for ITS stacks and protocols, the application should include an option in the settings menu to toggle between these standards.

## 6.3 Publications

The mobile application developed in the scope of this dissertation is publicly available online at the Google Play Store.

Furthermore, this work has been presented at the 29th ITS World Congress:

- **Gil Teixeira**, David Rocha, Emanuel Vieira, João Almeida, Joaquim Ferreira and José Fonseca. "A Mobile Application for Road Sensing and V2X Services", *29th ITS World Congress*, 16-20 October, 2023, Suzhou, China

Finally, the on-board system architecture, including the developed application, has also been published in a journal publication:

- David Rocha, **Gil Teixeira**, Emanuel Vieira, João Almeida, and Joaquim Ferreira. 2023. "A Modular In-Vehicle C-ITS Architecture for Sensor Data Collection, Vehicular Communications and Cloud Connectivity", *Sensors* 23, no. 3: 1724. https://doi.org/10.3390/s23031724

APPENDIX A

# Sensor Messages Format

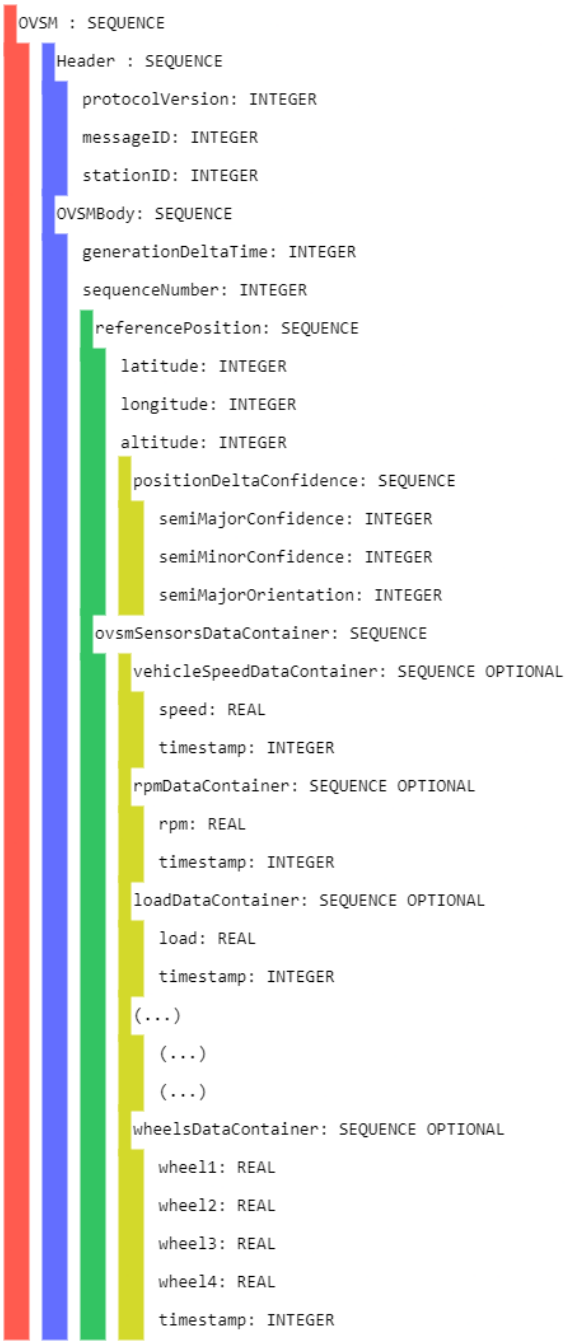


**Figure A.1:** OVSM ASN.1 message format.

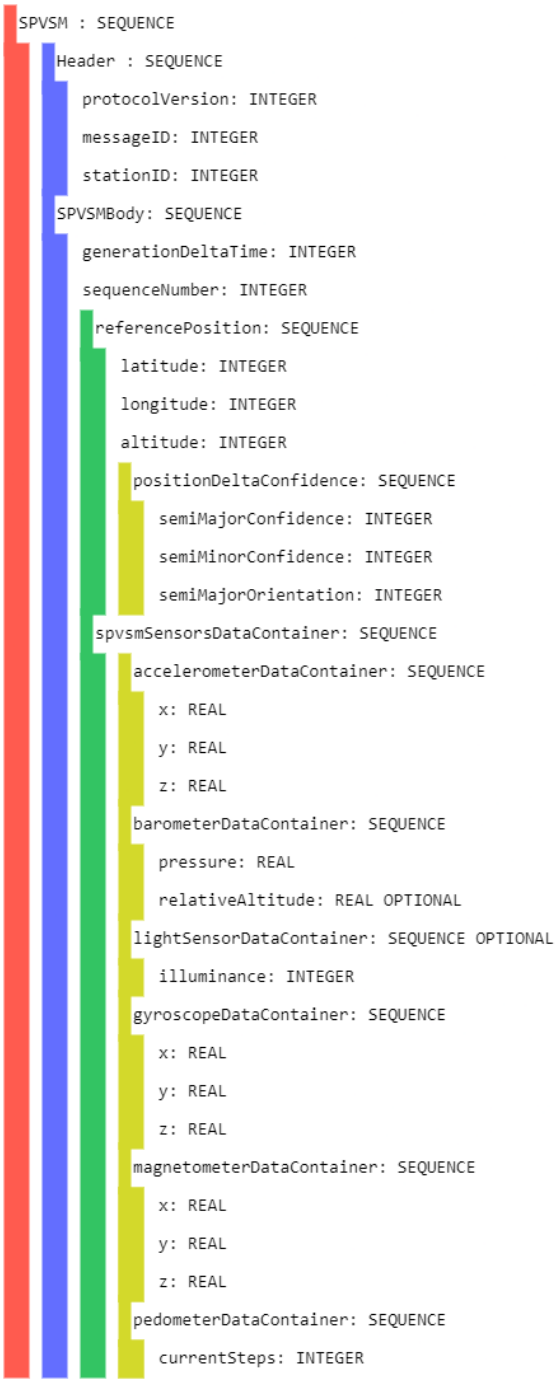


**Figure A.2:** SPVSM ASN.1 message format.

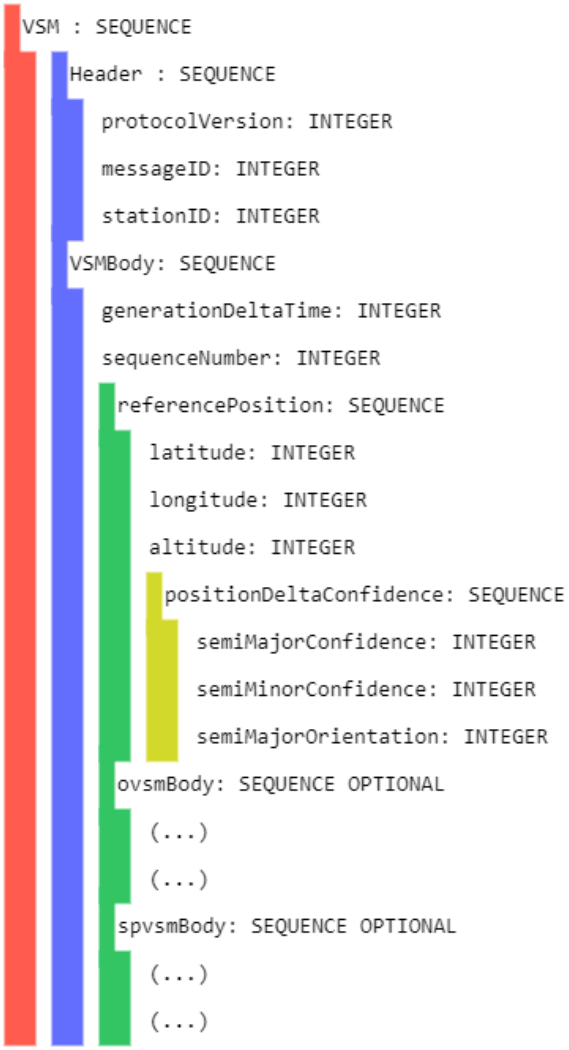


**Figure A.3:** VSM ASN.1 message format.

APPENDIX B

# Nielsen Heuristics Evaluators Guide

**Evaluators:** Gil, Emanuel, David, Mohannad

**Format used:** (Who found the problem) Problem -> '(' (comma separated evaluator ratings using the same order as at the top of this document)+ ')'.

The 10 heuristics are explained in detail, with practical examples, here. The following 0 to 4 rating scale is to be used to rate the severity of usability problems:

| | |
|---|---|
| 0 | I don't agree that this is a usability problem at all |
| 1 | Cosmetic problem only: need not be fixed unless extra time is available on project |
| 2 | Minor usability problem: fixing this should be given low priority |
| 3 | Major usability problem: important to fix, so should be given high priority |
| 4 | Usability catastrophe: imperative to fix this before product can be released |

**Table B.1:** Usability issue severity scale.

Problems found per heuristic:

**#1: Visibility of system status**

- (Emanuel, Mohannad, David) Current position/speed/heading in all modes could be viewed somewhere -> (2,1,1,1)
- (Mohannad) The mode that the app is working in (VRU, OBU, directly to the broker) disappears after the connection is established. I think it is useful to add an icon to indicate the activated mode -> (2,2,2,2)

**#2: Match between system and the real world**

- (Gil, Emanuel) In settings OBU, Station ID and Quadtree may not be part of the user language -> (0,1,0,0)

**#3: User control and freedom**

- (Mohannad) I would move the "Advanced Setting" into a separate view, which includes (besides the available advanced settings) options that allow the customization of the sending and receiving topics, with individual "Reset Setting" options -> (2,2,2,2)

**#4: Consistency and standards**

- (Gil) Denm sending occurs when the user clicks the sub-cause code event image unlike the Denm update which allows for the selection of the code with the image but has a separate button for sending -> (2,2,2,2)
- (Mohannad) Related to the previous point in this category, I think that putting the "Change Validity" and "Change Relevance Distance" at the top of the screen can help to make it more intuitive to set these values before clicking on a sub-cause image -> (2,2,2,2)

**#5: Error prevention**

- (Gil, Mohannad) Denm sending occurs immediately when the user selects the sub-cause code and the user might not see or use the relevanceDistance and validityDuration -> (2,2,2,2)
- (Mohannad) Related to the previous point in this category, I think that the swapping idea could help to avoid the issue -> (2,2,2,2)
- (Mohannad) When I first used the app to publish a DENM, I did not notice that the relevanceDistance and the validityDuration can be set. I think that changing the style or even using sliders or wheels to set these values can help to make it faster and more noticeable to set these values -> (1,1,1,1)
- (Mohannad, Emanuel, David) I have noticed that the app (sometimes) returns to the default settings after it gets completely closed -> (4,3,2,3)

**#6: Recognition rather than recall**

- (Gil) The round connect button is used for creating DENM but it's not labeled -> (3,2,2,2)
- (Gil) The user might not notice the history and EV reservation management icons (1,1,1,1)
- (Mohannad, David) I would explicitly group the options in the settings according to the mode that they apply to (VRU, OBU, directly to the broker) -> (1,3,1,1)

**#7: Flexibility and efficiency of use**

- (Emanuel, Mohannad) A more straightforward way to find and set the current stationID in OBU mode should be implemented. Maybe through a special type of message, e.g., heartbeat. (2,2,2,2)
- (Emanuel) Perhaps the app could connect right away to the broker upon startup? Needs refactoring also of the mode slider -> (1,1,0,0)

**#8: Aesthetic and minimalist design**

- (Emanuel) RSU icon should be changed -> (1,1,1,1)
- (David) Small screens get cluttered when many messages are being received, there could be a filter list to only show what we are looking for. maybe this could also help with the performance (less callback calls ?) -> (1,1,1,1).
- (David) The spacing on the text present in the sensors tab is not consistent -> (1,2,1,1)
- (David, Mohannad) The connection configurations should be further separated depending on the mode -> (1,3,1,3).

**#9: Help users recognize, diagnose, and recover from errors**

- (David) The MQTT connect error code as well as the meaning should be shown when connection to the broker fails, instead of the generic (“Please make sure that Host and Port are...”) -> (2,2,2,2)

**#10: Help and documentation**

- (Emanuel) There should be a link to the documentation in the app. A overlay step-by-step usage guide would be optimal -> (1,2,2,1)
- (David) There could be a clickable (?) icon explaining some configuration parameters, such as the meaning of the quadtree zoom, the units of the message sending intervals and the user modes it affects (is the username and password only used in TCC or also in OBU mode?) -> (2,2,2,2)

APPENDIX C

# Thinking Aloud Observer Form

# (Observer) IT2S-Mobile-App - Thinking aloud

leagueoflegendsfeeds@gmail.com Switch account

Not shared

Draft saved

* Indicates required question

Participant ID *

Your answer

| | Failed | Answer Error | Lost | Asked for Help |
|---|---|---|---|---|
| 1. open settings | ○ | ○ | ○ | ○ |
| 1.1. understanding settings | ○ | ○ | ○ | ○ |
| 2. connect | ○ | ○ | ○ | ○ |
| 3. RSU object speed | ○ | ○ | ○ | ○ |
| 3.1 Difficulty of the prior task | ○ | ○ | ○ | ○ |
| 4. create denm | ○ | ○ | ○ | ○ |
| 4.1. relevance distance and validation duration | ○ | ○ | ○ | ○ |

| | Failed | Answer Error | Lost | Asked for Help |
|---|---|---|---|---|
| 5. History | ○ | ○ | ○ | ○ |
| 6. Phone Sensors | ○ | ○ | ○ | ○ |
| 7. Connect in OBU mode | ○ | ○ | ○ | ○ |
| 8. EV reservation | ○ | ○ | ○ | ○ |
| 8.1. Highest requestable energy amount | ○ | ○ | ○ | ○ |
| 8.2. Cancel reservation | ○ | ○ | ○ | ○ |
| 9. Vehicle Sensors | ○ | ○ | ○ | ○ |
| 10. Rate the app | ○ | ○ | ○ | ○ |
| 10.1 How could it be bettered | ○ | ○ | ○ | ○ |

Time used *

Your answer

Comentários

Your answer

APPENDIX D

# Thinking Aloud User Form

# IT2S-Mobile-App - Thinking aloud

Usability test for the it2s-mobile-app, the application developed for my master dissertation.



* Indicates required question

Participant ID

Your answer

Install the application

The application is available on the Google Play store with the name CCAM. The store url is: https://play.google.com/store/apps/details?id=com.it2s.it2smobileapp&hl=en_SG&gl=US

1. Open the settings and set the application language to your preference. Indicate your preference: *

- ◯ Portuguese
- ◯ English

1.1. Which settings do you have the most difficulty understanding what the setting means? If there aren't many please answer at most 5. *

Your answer

2. Return to the map and connect the application in TCC (Traffic Control Center) mode. How easy was it to perform the action? *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

3. Locate a Road Side Unit (RSU) within the Aveiro district and choose any vehicle perceived by it. What is the speed of the vehicle? *

Your answer

3.1 How difficult was the prior task? *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

4. Create a slippery road alert with a relevance distance of 500 meters and a validity duration of 2 minutes. Classify the difficulty: *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

4.1. Update the event relevance distance to 200 meters. Classify the difficulty: *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

5. Open the history of the events received by the application and delete the event created in the prior two questions. Classify the difficulty: *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

6. Navigate to the sensors screen and observe the sensors of the mobile phone. Classify the difficulty: *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

Disconnect the application.

7. Configure the connection to the vehicle in the settings using the following information: *

Host: 192.168.94.35
Port: 8081
Username: testUser
Password: testme
Custom Station ID: 7301

Go back to the map and connect the application in OBU (vehicle) mode. Classify the difficulty of the settings configuration:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

8. Create an EV reservation by selecting one of the available charging station icons on the map. Set the time of arrival to 30 minutes from now and the charging session duration to an hour. Select the highest amount of energy required. Classify the difficulty: *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

8.1. What is the highest amount of energy that can be requested? *

Your answer

8.2. After a successful reservation cancel it. Classify the difficulty: *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

9. Navigate back to the sensors screen and observe the vehicle sensors. How difficult was the task? *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Very Easy | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Very Hard |

10. How would you rate the app? *

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ |

10.1 How would you improve the application? *

Your answer

Submit Clear form

Never submit passwords through Google Forms.



# References


[1] *EU vehicle fleet: size and distribution*, Accessed on September 2023. [Online]. Available: https://www.acea.auto/publication/report-vehicles-in-use-europe-2023/.

[2] *Motorization rate 2020 – worldwide*, Accessed on October 2023. [Online]. Available: https://www.oica.net/category/vehicles-in-use/.

[3] *World population*, Accessed on October 2023. [Online]. Available: https://en.wikipedia.org/wiki/World_population.

[4] *Road traffic injuries*, Accessed on October 2023. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[5] S. Lobo, A. Festag, and C. Facchi, "Enhancing the Safety of Vulnerable Road Users: Messaging Protocols for V2X Communication," in *2022 IEEE 96th Vehicular Technology Conference (VTC2022-Fall)*, 2022, pp. 1–7. DOI: 10.1109/VTC2022-Fall57202.2022.10012775.

[6] A. Willecke, K. Garlichs, F. Schulze, and L. C. Wolf, "Vulnerable Road Users Are Important As Well: Persons in the Collective Perception Service," in *2021 IEEE Vehicular Networking Conference (VNC)*, 2021, pp. 24–31. DOI: 10.1109/VNC52810.2021.9644669.

[7] I. Khan, G. Hoang, and J. Härri, "Rethinking cooperative awareness for future V2X safety-critical applications," in *2017 IEEE Vehicular Networking Conference (VNC)*, 2017, pp. 73–76. DOI: 10.1109/VNC.2017.8275645.

[8] M. A. Javed and E. Ben Hamida, "Measuring safety awareness in cooperative ITS applications, year=2016," in *2016 IEEE Wireless Communications and Networking Conference*, pp. 1–7. DOI: 10.1109/WCNC.2016.7564927.

[9] M. Boban and P. M. d'Orey, "Exploring the Practical Limits of Cooperative Awareness in Vehicular Communications," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3904–3916, 2016. DOI: 10.1109/TVT.2016.2544935.

[10] *5G Enabled Road Safety Services*. [Online]. Available: https://5gsafeplus.fmi.fi.

[11] B. Wang, C. Chen, and T. Zhang, "Commercial Vehicle Road Collaborative System Based on 5G-V2X and Satellite Navigation Technologies," in *China Satellite Navigation Conference (CSNC 2021) Proceedings*, C. Yang and J. Xie, Eds., Singapore: Springer Singapore, 2021, pp. 274–282, ISBN: 978-981-16-3138-2.

[12] J. Brož, T. Tichý, V. Angelakis, and Z. Bělinová, "Usage of V2X Applications in Road Tunnels," *Applied Sciences*, vol. 12, no. 9, 2022, ISSN: 2076-3417. DOI: 10.3390/app12094624. [Online]. Available: https://www.mdpi.com/2076-3417/12/9/4624.

[13] "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions," European Telecommunications Standards Institute, Standard, 2009.

[14] "Intelligent Transport Systems (ITS); Communications Architecture," European Telecommunications Standards Institute, Standard, 2010, p. 44.

[15] "Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 3: Network Architecture," European Telecommunications Standards Institute, Standard, 2014.

[16] "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service," European Telecommunications Standards Institute, Standard, 2014.

[17] "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Analysis of the Collective Perception Service (CPS); Release 2," European Telecommunications Standards Institute, Standard, 2019.

[18] "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service," European Telecommunications Standards Institute, Standard, 2014.

[19] "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Facilities layer protocols and communication requirements for infrastructure services," European Telecommunications Standards Institute, Standard, 2020.

[20] "Intelligent Transport Systems (ITS); Facilities layer function; Part 1: Services Announcement (SA) specification," European Telecommunications Standards Institute, Standard, 2019.

[21] "Intelligent Transport Systems (ITS); Vulnerable Road Users (VRU) awareness; Part 3: Specification of VRU awareness basic service; Release 2," European Telecommunications Standards Institute, Standard, 2020.

[22] *EV market share*. [Online]. Available: https://www.ev-volumes.com/.

[23] "Infrastructure to Vehicle Communication; Electric Vehicle Charging Spot Notification Specification," European Telecommunications Standards Institute, Standard, Jul. 2012.

[24] "Infrastructure to Vehicle Communications; Part 3: Communications system for the planning and reservation of EV energy supply using wireless networks," European Telecommunications Standards Institute, Standard, Oct. 2014.

[25] "Pre-Standardization Study on payment applications in Cooperative ITS using V2I communication," European Telecommunications Standards Institute, Study, 2019.

[26] E. Vieira, T. Dias, J. Almeida, A. V. Silva, J. Ferreira, and L. Moura, "V2X Tolling System for C-ITS Environments," in *Proceedings of the 9th International Conference on Vehicle Technology and Intelligent Transport Systems - VEHITS*, INSTICC, SciTePress, 2023, pp. 103–112, ISBN: 978-989-758-652-1. DOI: 10.5220/0011994000003479.

[27] "MQTT Version 3.1.1," OASIS, Standard, 2014.

[28] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, RFC 7252, Jun. 2014. DOI: 10.17487/RFC7252. [Online]. Available: https://www.rfc-editor.org/info/rfc7252.

[29] I. JTC1, *Advanced Message Queuing Protocol (AMQP) v1.0 specification*, https://www.iso.org/standard/64955.html, 2014.

[30] A. Melnikov and I. Fette, *The WebSocket Protocol*, RFC 6455, Dec. 2011. DOI: 10.17487/RFC6455. [Online]. Available: https://www.rfc-editor.org/info/rfc6455.

[31] *Connected Corridor for Driving Automation*, https://connectedautomateddriving.eu/project/concorda/.

[32] *InterCor Project, "Milestone 4 - Common set of upgraded specifications for hybrid communication*, https://intercor-project.eu/wp-content/uploads/sites/15/2019/01/InterCor_M4-Upgraded-Specifications-Hybrid_v2.1-final_INEA.pdf.

[33] *C-MoBILE (Accelerating C-ITS Mobility Innovation and depLoyment in Europe)*, https://connectedautomateddriving.eu/project/concorda/.

[34] *Eclipse Mosquitto*. [Online]. Available: https://mosquitto.org/.

[35] *HiveMQ*. [Online]. Available: https://www.hivemq.com/.

[36] *RabbitMQ*. [Online]. Available: https://www.rabbitmq.com/.

[37] Bing, Microsoft, *Bing Maps Tile System*, [Online; accessed May 9, 2023]. [Online]. Available: https://learn.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system.

[38] *Mobile Operating System Market Share Worldwide - December 2022*. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[39] J. H. Gao and L.-S. Peh, "Automotive V2X on phones: Enabling next-generation mobile ITS apps," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 858–863.

[40] *Kapsch Connected Vehicles*, Accessed on June 2023. [Online]. Available: https://www.kapsch.net/en/connected-vehicles.

[41] *Kapsch V2X Assist*, Accessed on June 2023. [Online]. Available: https://play.google.com/store/apps/details?id=net.kapsch.v2x.assist&hl=pt&gl=US.

[42] C. T. R. M. A. Kapsch Ford, "CV2X TOLLING WHITE PAPER Demonstrated September 15th, 2020 for the IBTTA 88th Annual Meeting," Tech. Rep., Accessed on June 2023. [Online]. Available: https://mcusercontent.com/9f7779d2f03a31e0a9ca66bcc/files/c44f026c-f1e2-0a21-0d20-d942dea1ce10/IBTTA_2020_CTRMA_Ford_KTC_CV2X_Tolling_Demo_White_Pape.pdf.

[43] *V2X-A3*, Accessed on June 2023. [Online]. Available: https://v2x.co.th/product-v2xa3/.

[44] *CITS C-ITS (by Inzinious, Inc.)* Accessed on June 2023. [Online]. Available: https://play.google.com/store/apps/details?id=com.inzinious.hud.cits.release&hl=en&gl=US.

[45] *UPROAD*, Accessed on June 2023. [Online]. Available: https://www.uproad.com/.

[46] *Slora*, Accessed on June 2023. [Online]. Available: https://openvia.io/en/nuestras-soluciones/movilidad-digital/slora/.

[47] *ev.energy*, Accessed on June 2023. [Online]. Available: https://ev.energy.

[48] *Miio*, Accessed on June 2023. [Online]. Available: https://www.miio.pt/.

[49] N. F. S. Aparicio, J. J. d. C. Ferreira, and P. J. d. C. Bartolomeu, *Serviços de monitorização e alerta meteorológicos para sistemas cooperativos de transporte inteligentes*, Feb. 2021. [Online]. Available: http://hdl.handle.net/10773/31507.

[50] T. Dias, A. Silva, J. Almeida, M. Jooriah, J. Ferreira, and L. Moura, "Portuguese 5G Connected Vehicle and Infrastructure in the 5G-MOBIX project," May 2022.

[51] D. Rocha, G. Teixeira, E. Vieira, J. Almeida, and J. Ferreira, "A Modular In-Vehicle C-ITS Architecture for Sensor Data Collection, Vehicular Communications and Cloud Connectivity," *Sensors*, vol. 23, no. 3, 2023, ISSN: 1424-8220. DOI: 10.3390/s23031724. [Online]. Available: https://www.mdpi.com/1424-8220/23/3/1724.

[52] *react-navigation*, Accessed on June 2023. [Online]. Available: https://reactnavigation.org/.

[53] *react-native-paho-mqtt*, Accessed on June 2023. [Online]. Available: https://github.com/robhogan/react-native-paho-mqtt.

[54] *react-native-maps*, Accessed on June 2023. [Online]. Available: https://www.ev.energy/.

[55] *Expo-Sensors*, Accessed on June 2023. [Online]. Available: https://docs.expo.dev/versions/latest/sdk/sensors/.

[56] *EAS (Expo)*, Accessed on July 2023. [Online]. Available: https://expo.dev/eas.

[57] *Azimuth*, Accessed on November 2023. [Online]. Available: https://pt.wikipedia.org/wiki/Azimute.

[58] G. Teixeira, *Pull Request to fix visual bugs and warnings for dated code*. [Online]. Available: https://github.com/expo/expo/pull/18225.

[59] ——, *Discussion on integrating the Light Sensor into the Expo SDK*. [Online]. Available: https://github.com/expo/expo/discussions/18101.

[60] *React-native-maps callout onPress issue*, Accessed on July 2023. [Online]. Available: https://github.com/react-native-maps/react-native-maps/issues/1541.

[61] *React-native-maps custom marker issue*, Accessed on July 2023. [Online]. Available: https://github.com/react-native-maps/react-native-maps/issues/3769.

[62] ——, *Pull Request to update the lib to RN 0.62.* [Online]. Available: https://github.com/summerfed/rn-tri-toggle-switch/pull/2.

[63] ——, *Pull Request to fix the limits to the position of the circle selector.* [Online]. Available: https://github.com/summerfed/rn-tri-toggle-switch/pull/5.

[64] *The ten Nielsen usability heuristics*, Accessed on July 2023. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/.

[65] *Guidelines on conducting a heuristic evaluation*, Accessed on July 2023. [Online]. Available: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/.

[66] *Theory on heuristic evaluations*, Accessed on July 2023. [Online]. Available: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/theory-heuristic-evaluations/.

[67] *Nielsen heuristic evaluation graph creator*, Accessed on July 2023. [Online]. Available: https://github.com/bearkillerPT/NielsenHeuristicsGraph.