

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mobile app for protecting cyclists and pedestrians in road traffic

Luís Alvela Duarte Mendes



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Ana Cristina Costa Aguiar

July 14, 2021

Mobile app for protecting cyclists and pedestrians in road traffic

Luís Alvela Duarte Mendes

Mestrado Integrado em Engenharia Informática e Computação

July 14, 2021

Abstract

In today's interconnected world, vehicles have autonomous systems that allow them to communicate between themselves and improve road safety. However, although there is much research in vehicle-to-vehicle communication, this vision usually neglects Vulnerable Road Users (VRU). The point of this work is to develop a limited mobile application prototype that will serve as a human-computer interface for pedestrian-to-vehicle and even cyclist-to-vehicle communication systems, allowing for a safer road environment.

The application, known as the Unprotected Road User Shield (URU-S), was developed using software engineering techniques, like requirements gathering, design and prototyping, implementation, validation, verification (unit testing), and documentation.

Previous work in developing an effective pre-crash collision detection system with smart devices has struggled with power consumption, channel congestion, security, and many other issues. URU-S differentiates itself from past efforts to conceive VRU safety systems by directly involving VRU in the interaction design process and studying which methods are best to warn them. Focus groups and usability tests were conducted to achieve user-centered design, making an effort to develop a high-fidelity prototype of a road safety system that these users would want to install in the future. Additionally, measurements were done to best gauge the application's performance, determining how long it took to detect movement, indoor and outdoor localization, and the latency associated with each warning that it could receive.

The usability tests found that the application was straightforward for most of the participants. In contrast, the additional measurements suggest that more work must be done regarding the communication infrastructure. Overall, the result was a robust, well-documented, validated, and functional prototype that includes the established use cases.

Keywords: Vulnerable Road User, VRU, mobile, safety, human-computer interaction, pedestrian-to-vehicle, vehicle-to-vehicle, cyclist-to-vehicle, pedestrian, cyclist, vehicle, protection, collision avoidance, smartphones, user-centered design, HCI.

Resumo

No mundo interconectado de hoje, os veículos possuem sistemas autónomos que lhes permitem comunicar entre si para evitar colisões. No entanto, apesar de haver muita pesquisa no domínio da comunicação veículo-para-veículo, esta visão geralmente negligencia os Utilizadores Vulneráveis da Estrada (VRU). O objetivo deste trabalho é desenvolver um protótipo de aplicação móvel limitado que pretende servir como interface humano-computador para sistemas de comunicação peão-para-veículo e até ciclista-para-veículo, permitindo um ambiente de estrada mais seguro.

A aplicação, conhecida como *Unprotected Road User Shield (URU-S)*, foi desenvolvida através de técnicas de engenharia de software como levantamento de requisitos, desenho e prototipagem, implementação, validação, verificação (testes unitários) e documentação.

Trabalho prévio em desenvolver sistemas eficazes de aviso pré-colisão com dispositivos inteligentes deparou-se com desafios tais como consumo energético, congestionamento de canais, segurança e muitos outros. *URU-S* distingue-se de esforços anteriores de desenvolvimento de sistemas de segurança VRU ao envolver VRU diretamente no processo de desenho de interação e estudando quais os melhores métodos de os avisar. Grupos de foco e testes de usabilidade foram feitos para atingir um desenho centrado no utilizador, fazendo-se um esforço para desenvolver um protótipo de alta-fidelidade para um sistema de segurança rodoviária que estes utilizadores poderão querer instalar no futuro. Adicionalmente, medições foram feitas para averiguar o desempenho da aplicação, determinando quanto tempo é que demorava a detetar movimento, localização *indoor* e *outdoor*, e a latência associada a cada aviso que recebia.

Os testes de usabilidade confirmam que a aplicação é de uso fácil para a maioria dos participantes. Em contraste, as medições adicionais sugerem que mais trabalho devia de ser feito em termos da infraestrutura de comunicações. Em geral, o resultado foi um protótipo funcional, robusto, validado e bem documentado que inclui os casos de uso estabelecidos.

Palavras-chave: Utilizadores Vulneráveis da Estrada, VRU, móvel, segurança, interação computador-humano, peão-para-veículo, veículo-para-veículo, ciclista-para-veículo, peão, ciclista, veículo, proteção, evasão de colisões, *smartphones*, desenho centrado no utilizador, HCI.

Acknowledgements

I want to thank:

- My family, friends, and colleagues for their love and support.
- IT-Porto, the Telecommunications Institute (UIDB/50008/2020) for supplying me with the tools and the environment necessary for the development of my work.
- My supervisor, Professor Ana Aguiar, for all of the support and orientation given.
- Professor Falko Dressler and Professor Klaus David for helping with the requirements phase and giving me some tips on how to improve the project further.
- Professor Teresa Galvão for helping with some tips in Human-Computer Interaction.
- Professor Lars Wolf and Professor Miguel Pimenta Monteiro, for the comments given during the final examination.
- Every attendee and organizer of the IEEE VNC 2020 conference, for also helping me gather some requirements and understand the technologies of the automotive world better.
- Every attendee of my focus group and usability test.
- Doctor Isidro Ribeiro Pereira, for lending me some materials that were essential for the dissertation's progress.
- The developers of Moqups.com and Remove.bg, for providing a platform for developing my WireFrames and a handy tool for removing backgrounds (respectively).
- Pngtree, for the stock png of the "shield" used in the URU-S logo.
- Raven17, the artist of "bird" png used in the URU-S logo.
- The developers of NumberEight (the context awareness mechanism), specifically Chris Watts, for helping me out with some doubts that I had about the API.

Luís Alvela Duarte Mendes

*“The computing scientist’s main challenge is
not to get confused by the complexities of his own making.”*

Edsger W. Dijkstra

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Objectives	2
1.4	Document Structure	3
2	State of the Art	5
2.1	Background on VRU Protection	5
2.2	Background on HCI Methodologies	6
2.3	Architectures and Relevant Technologies	8
2.3.1	Direct Connection	8
2.3.2	Cellular Communication	13
2.3.3	Central Server (MEC)	13
2.4	Related Work	16
2.4.1	Vehicle-centered Solutions	16
2.4.2	Requirements and Architectures	17
2.4.3	Positioning and Context-Awareness	19
2.4.4	Warning Interface	19
2.4.5	Mobile/Multi-Access Edge Computing	21
2.4.6	Psychological Studies on Human Stimuli	22
2.5	Discussion Relative to Previous Work	23
3	Smartphone Based VRU Protection	25
3.1	Application Use Case and Assumed System Model	25
3.2	Framing the Problem	26
3.3	Proposed Solution and Approach	28
3.3.1	User Centric Design Methodology	28
3.3.2	Software Engineering Methods	28
3.3.3	Tools	30
3.4	Use Cases	31
4	URU-S Design and Implementation	33
4.1	URU-S Design	33
4.1.1	Preliminary Design (WireFrame)	33
4.1.2	Focus Group	38
4.1.3	Resulting WireFrame	38
4.1.4	Additional Features	43
4.2	Architecture	43

4.3	Context-Awareness and Glimpses	45
4.4	Geofencing Foreground Service	47
4.5	Geofencing Data Acquisition	48
4.6	Geofence Accuracy	49
4.7	Alert System	51
4.8	Requirement Satisfaction	53
4.9	Final Screens, Flow and Use Cases	57
4.9.1	Home Screen	57
4.9.2	Options Menu	58
4.9.3	Alert Settings	59
4.9.4	Statistics	60
4.9.5	Real-Time Alert Notification	61
4.9.6	Context-Awareness Notification	62
4.9.7	Context-Awareness Override	63
4.9.8	Alert Suspension	63
4.9.9	Geofencing and Danger-Zone Entry	64
5	Evaluation of the Proposed Solution	65
5.1	Unit-Test-Based Verification	65
5.1.1	Results	71
5.2	Usability-Test-Based Validation	74
5.2.1	Setup	75
5.2.2	Result	76
5.3	Additional Measurements	76
5.3.1	Context-Awareness	77
5.3.2	Latency	81
6	Conclusions and Future Work	91
6.1	Conclusion	91
6.2	Further Work	92
A	Focus Group Report	95
A.1	Executive Summary	95
A.2	Methodology & Participant Profile	96
A.2.1	Instrument Development	96
A.2.2	Site Selection	96
A.2.3	Participant Selection	96
A.2.4	Procedure for Current and Future Focus Groups	96
A.2.5	Participant Profile	96
A.3	Demographics	97
A.4	Discussion Results	98
A.4.1	Question #1 - What would people like to see in a road safety app such as this?	98
A.4.2	Question #2 - What would people like in terms of notifications? Sound, Touch, Visual?	98
A.4.3	Question #3 - If you would like to have sounds, which sounds?	98
A.4.4	Question #4 - If visual, in what way?	99
A.4.5	Question #5 - Do you like the ability to configure/suspend the notification settings?	99

A.4.6	Question #6 - Do you like the ability to suspend for a given amount of minutes?	99
A.4.7	Question #8 - Would they like some form of widget integration with a PSP (police) map of the most dangerous locations in terms of accidents? . . .	100
A.4.8	Question #9 - Would you like to be capable of instructing the app's notifications to shut down in certain areas that you know for a fact are prone to false positives?	100
A.4.9	Question #10 - Are you interested in the login/register option for saving your settings, or would you like some other form, like the upload/download or even some other form (please specify)?	100
A.4.10	Question #11 - Do you think the documentation is accessible where it is now, via the options menu?	100
A.4.11	Question #12 - What do you think was the most vital thing discussed today?	100
A.5	Conclusion	101
A.6	Recommendations	101
A.7	Appendices	102
A.7.1	Privacy Policy	102
A.7.2	Presentation Used	102
A.7.3	Question Guide	102
A.7.4	Sketch of an interface misunderstanding	102
A.7.5	Polls	103
A.7.6	Interest in Danger Reports or Crowd-Sourcing Elements	108
B	Usability Testing Report	109
B.1	Introduction	109
B.2	Executive Summary	109
B.3	Methodology	111
B.3.1	Participants	112
B.4	Results	113
B.4.1	Task Completion Success Rate	114
B.4.2	Task Ratings	115
B.4.3	Time on Task	117
B.4.4	Errors	118
B.4.5	Outdoor Phase	118
B.4.6	Summary of Data	118
B.4.7	Likes, Dislikes, Participant Suggestions	119
B.4.8	Participant Suggestions for Improvement	119
B.5	Recommendations	120
B.6	Conclusion	121
C	Privacy Policy	123
C.1	URU-S	123
C.2	Goals of the Treatment	124
C.3	Collected Personal Data	124
C.4	Conservation Dates	124
C.5	Receivers of Personal Data	125
C.6	Data Owner Rights	125
C.7	Security Measures	125
C.8	Use of Equipment and Questionnaires	125

C.9 Remove Consent	126
C.10 Contacts	126
C.11 Responsible for Data Protection	126
C.12 Informed Consent Term	126

List of Figures

2.1	Scheme Illustrating the advantages of "Wizard-of-Oz" experiments [13]	7
2.2	General concept of a VRU safety system. [14]	8
2.3	Left column: Three scenarios in which the ego vehicle's sensor limitations may compromise safety. Right column: How collective perception messages may improve the safety of the ego vehicle for each scenario. [15]	9
2.4	Communication allowed between vehicles and people in WAVE: person to vehicle (above); vehicle to person (below) [5]	10
2.5	V2P system design in smartphone [7]	11
2.6	Extended architecture of the VCE with the newly added Collision Warning System (CWS) and signals (highlighted in black). Signals are provided to the cyclist via vibration motors (haptic), speakers (audio), or a virtual display in the Unity 3D visualization (visual). Collisions detected by the CWS are forwarded to the signals via the EVI. [20]	12
2.7	Cooperative perception system [4]	13
2.8	A conceptual view of the proposed MEC-based architecture. POAs of different mobile networks collect BSMs from both vulnerable users (such as the pedestrian in blue) and vehicles (such as the car in red) in steps 1a and 1b. POAs then convey the BSMs to a collision-detection server (steps 2a and 2b), which processes them and establishes whether there is a collision risk. If there is, alerts are conveyed to the appropriate POAs (steps 3a and 3b) and, hence, to the interested users (steps 4a and 4b). [27]	14
2.9	MEC-based collision detection. [14]	15
2.10	Block Diagram of the proposed architecture. [10]	16
3.1	Simplified Sketch of the Presumed Architecture	25
4.1	Optional Sign In, Sign Up, and Home	34
4.2	Options	35
4.3	Settings	36
4.4	Collision	37
4.5	Home, Reports and Options	39
4.6	Options, FAQ, Alert Settings	40
4.7	Alert Settings Complete View	41
4.8	Danger warning	42
4.9	Implemented Architecture	44
4.10	Danger Zones JSON	48
4.11	Location Request High Accuracy	50
4.12	Alert System State Transitions	51

4.13	Accessibility Delegate	53
4.14	Other states of URU-S	57
4.15	Home Screen Illustration	57
4.16	Options Menu Illustration	58
4.17	Alert Settings Illustration	59
4.18	Statistics Illustration	60
4.19	Real-Time Alert Notification Illustration	61
4.20	Context-Awareness Notification Illustration	62
4.21	Context-Awareness Override Illustration	63
4.22	Alert Suspension Illustration	63
4.23	Geofencing Notification Illustration	64
4.24	Danger-Zone Notification Illustration	64
4.25	Danger-Zone Positioning Illustration	64
5.1	Reports Activity Test First Scenario	66
5.2	Reports Activity Test Second Scenario	67
5.3	Banner Tests with Drawable Compare	68
5.4	Espresso and Ui Automator Working Together	69
5.5	Binding Services for Testing	69
5.6	Context-Awareness Service Test	70
5.7	Code Coverage Obtained with JaCoCo [66]	72
5.8	Receivers package coverage	72
5.9	Field Testing Area Illustration	75
5.10	Camera and Laptop Connected	76
5.11	Comparison Between the time to Swap in both Directions Indoor/Outdoor	78
5.12	Comparison Between the time to Swap in both Directions Moving/Stationary	79
5.13	Packet Travel Diagram where $L = t_3 - t_0$. We consider t_i : application layer timestamp placed in points of the infrastructure; t_0 : timestamp obtained before the data and token payload is delivered to the server (observable and comparable); t_1 : timestamp marking the arrival of the payload at the server (not observable); t_2 : timestamp for message acceptance by the server, after going through message queues, the internal broker and before sending the data out to the device (observable but not comparable due to an unknown time sync source); t_3 : timestamp for message acceptance by the smartphone, engaging the warning for the VRU (observable and comparable).	82
5.14	WiFi Boxplots for Good and Bad Coverage	87
5.15	5G Boxplots for Good and Bad Coverage	88
5.16	Outdoor testing setup	89
5.17	Comparing WiFi with 5G values	89
A.1	Interface Misunderstanding Scheme	103
A.2	Notification Types Poll	104
A.3	Notification Types Results	104
A.4	Sounds Poll	105
A.5	Sounds Results	105
A.6	On-and-Off-Switch Poll	106
A.7	On-and-off-Switch Results	106
A.8	Instructed-Notification-Shutdown Poll	107
A.9	Instructed-Notification-Shutdown Results	107

A.10 Danger Reports Poll 108

A.11 Danger Reports Results 108

List of Tables

3.1	Technical Requirements	26
3.2	Constraints	27
3.3	Business Rules	27
3.4	Use Cases	31
4.1	Active Mechanisms by State	47
5.1	WiFi <i>Ping</i> $\Delta_{0,1}$	85
5.2	WiFi <i>Ping</i> $\Delta_{2,3}$	85
5.3	5G <i>Ping</i> $\Delta_{2,3}$	85
B.1	Participant Demographics	112
B.2	Completion Rate per Task for each Participant	114
B.3	Difficulty on Task for each Participant	115
B.4	Time on Task for each Participant	117
B.5	Task Summary	118

Abbreviations

URU-S	Unprotected Road User Shield
VRU	Vulnerable Road User
FEUP	Faculdade de Engenharia da Universidade do Porto
C-ITS	Cooperative Intelligent Transport Systems
API	Application Programming Interface
ADAS	Advanced Driver Assistance Systems
SDK	Software Development Kit
HCI	Human Computer Interaction
IT	Instituto de Telecomunicações
LTE	Long Term Evolution
LOS	Line Of Sight
DSRC	Dedicated Short-Range Communications
GPS	Global Positioning System
WAVE	Wireless Access in Vehicular Environment
IEEE	Institute of Electrical and Electronics Engineers
BLE	Bluetooth Low Energy
SAE	Society of Automotive Engineers
VCE	Virtual Cycling Environment
TVA	Theory of Visual Attention
V2V	Vehicle to Vehicle
V2I	Vehicle to Infrastructure
V2C	Vehicle to Cyclist
V2P	Vehicle to Pedestrian
V2X	Vehicle to Everything

Chapter 1

Introduction

According to the World Health Organization in their latest (2018) Global Road Safety Report:

“The number of road traffic deaths on the world’s roads remains unacceptably high.” [1, chap. 1]

“Road traffic injuries are the leading killer of children and young adults.” [1, chap. 1]

“More than half of global road traffic deaths are amongst pedestrians, cyclists and motorcyclists who are still too often neglected in road traffic system design in many countries.” [1, chap. 1]

This dissertation is a direct response to that very last quote. Despite there being extensive research in vehicle-to-vehicle communication, there is very little concerning the protection of Vulnerable Road Users (VRU) such as pedestrians, cyclists, motorcyclists, people on mobility chairs, scooters, and any form of locomotion that lacks a proper protective structure. There are even articles that "identify gaps that exist in these fields that could be improved/extended/enhanced or newly developed" [2].

Thus, a novel solution is proposed to protect those most vulnerable: URU-S, a mobile (Android) application that intends to protect these users from potential collisions, emitting non-intrusive warnings in their devices that notify them when they are in imminent danger of colliding with a motorized vehicle, cyclist or even another pedestrian.

1.1 Context

The road is a hazardous and volatile environment where many accidents can happen. This is particularly true for intersections with limited visibility. Vehicular communication technology already exists today, and this vision could be extended to VRU. Several solutions have been devised over the past years, some of them including beacons or other specific hardware that is attached to the

VRU ("e.g., a C-ITS transmitter attached to a backpack or the safety vest of a worker at a road works site;" [3, sec. 4]). However, this technology would be quite cumbersome, and it cannot be presumed that all VRU would wear the same technology. Some solutions exist on vehicles themselves and are more centered around them, some of which are mentioned later in section 2.4.1.

It is important to note that this work is a complementary solution that can act in conjunction with others, such as the aforementioned VRU-portable devices and danger-perception technology (either on vehicles or road infrastructures themselves [4]).

1.2 Motivation

The primary motivation is to contribute to a safer road environment, where pedestrians, cyclists, and drivers alike may partake in a C-ITS that works towards their innate protection.

Studies estimate that "by 2020, 6.1 billion smartphone users are expected globally" ("70% of the world's population") [2]. Since most people have their smartphones on them at all times, particularly when they are out on the road, the phone itself could have an application working towards protecting them. The modern emphasis on smart devices truly shows the potential URU-S may have.

1.3 Objectives

The objective is to design a VRU protection smartphone application and develop a robust, validated, and well-documented prototype that will include a list of established use cases. The prototype will attempt to verify the following hypothesis:

"A Vulnerable Road User can be notified of danger effectively, through a smartphone application, if the appropriate software engineering and human-computer interaction methods are used to develop it."

This application would expand the field of pedestrian-to-vehicle and even cyclist-to-vehicle communication, thus integrating the VRU in the interconnected vehicle world. The risks of dangerous collisions would diminish by emitting warnings on the user's smart device to notify them in due time (considering reaction speeds) to prevent a dangerous crash. However, it is to note that actual collision detection is outside of this project's scope and is not implemented either; this factor will be simulated in the testing phases of the prototype with an external third-party tool. Instead, URU-S will make an effort to directly involve users in the design process as that seems to have been disregarded by past work.

The work developed will also have to recognize previous issues and trade-offs existing in this field. False positives are unavoidable, and the user should have the possibility to adjust their notification settings to allow them to be as non-intrusive as possible. It is important to note that false positives and intrusion are only some of the significant challenges that plague these applications;

others explored more in-depth in Chapter 2 are equally important. However, the primary focus of URU-S will be on the aspect of notifying the user effectively (and as non-intrusively as possible).

To clarify what is understood as "non-intrusively," all warnings are intrusive to a certain degree; however, this study attempts to develop an alert system that can be customized and catered to the user's preferences. The warning will also never cover the entire screen and block all activity on the phone. There are several ways to grab a user's attention, and URU-S will try to do it in the least annoying (but still effective) way possible, balancing effectiveness with intrusiveness.

Additionally, the dissertation will culminate in a prototype that is more focused on HCI components. This work will assume the existence of a collision-detection server and the transmission of positioning information to said server to handle the collision calculations.

1.4 Document Structure

The following document will be structured as such: in Chapter 2, the State of the Art will be presented, including some background, related work, a study of existing technologies, and approaches to the same or related problems. In Chapter 3, the proposed solution will be described, and the problem will be formulated in greater detail. In Chapter 4, the implementation of the solution will be discussed. In Chapter 5, the evaluation of the solution will be explicated, including some additional measurements conducted with the present infrastructure. Lastly, Chapter 6 presents the conclusions, which involve the result of this dissertation and further work that could be done in the field.

Chapter 2

State of the Art

There have been several efforts to develop applications that integrate VRU in a more interconnected C-ITS. These have struggled with intrusive notifications, battery consumption, range, latency, scalability, precision, vehicle-centrism, and others. This section aims to explore previous work and its upsides and downsides, offering a critical analysis of each article gathered so far.

2.1 Background on VRU Protection

Most articles involving Vulnerable Road Users, particularly in sensing or collision avoidance systems, tend to be from the car's perspective. Vehicle-centrism is particularly evident in specific standards stipulated by the Society of Automotive Engineers, an example of which is SAE J2735, which implies that mainly vehicles are the ones that assume responsibility in these kinds of situations [5, chap. 1]. An example would be the work conducted by Shoma Hisaka and Shunsuke Kamijo, which uses onboard wireless sensing mechanisms to determine where a pedestrian is, focusing entirely on vehicular technology [6]. Usually, the VRU is only involved passively in these systems, whereas this project appeals to their active participation.

Those that do regard the pedestrian more actively, such as the work done by Wu et al. [7], are commonly plagued by a poor user experience. This experience includes warnings covering up the entire screen, interrupting all activity on the phone, and blaring loud noises accompanied by bright visuals. Developing a non-intrusive and convenient application that integrates the VRU into a unified road safety system would be ideal.

Lastly, the 5G Automotive Association (5GAA) [8] does show a certain level of concern for solutions involving VRU, primarily as part of V2X (vehicle-to-everything) systems. In one of their reports, the association categorizes connected vehicle applications in four main groups of use cases: safety, convenience, advanced driving assistance, and vulnerable road user (VRU). The VRU use case group is explicitly described as "Detects and Warns drivers of VRUs in the vicinity" [9, sec. 2.1, table 5]; from this description, it seems it does not actively warn the VRU.

However, it does involve them in collision avoidance by having them "make their presence/location known through their mobile devices (e.g., smartphone, tablets)," this is listed in their "Use case example #3: Vulnerable Road User Discovery". Napolitano et al. take this one step further, building upon the advanced driving assistance and VRU use cases to propose a VRU protection system based on MEC. Their system allows road users to exchange potential crash information, warning them in case there are collision-prone entities nearby [10]. The system, as mentioned earlier, consists of a google-maps overlay application that displays information with pairs of latitude and longitude, requiring the user to enter a server IP to send positional information. The interface is quite technical and does not seem to be a byproduct of user-centered design but rather a prototype build strictly to test the underlying technology.

2.2 Background on HCI Methodologies

Three standard Human-Computer Interaction methodologies were used in the development of URU-S. These methodologies are known as focus groups, usability tests, and "Wizard-Of-Oz" experiments.

A focus group is a qualitative research method that consists of inviting five to ten participants to partake in an open discussion. Historically, focus groups were used for marketing purposes to infer which qualities consumers valued most in a product; this helps marketers focus on those aspects and gives valuable feedback to the development team. This methodology only became popular in the field of social science around the 1980s [11], being later adopted by the HCI field. These groups are a form of "focused interviewing" where the facilitator will ask a series of questions to stimulate discussion with the participants, encouraging them to share their thought process as much as possible. It is important to have participants from a wide array of sociological backgrounds to enable various discussions. The facilitator must also consider that some participants may be more assertive than others and must be skilled in stimulating all of those present to speak in their turn. Focus groups can be used in HCI, particularly in our context, to gather important feedback about user-valued features in our application. Our group would focus primarily on which notifications users would react to best and what features they would like to see in URU-S.

Usability testing is an empirical method of evaluation that attempts to gather feedback from users on improving the usability of an interface. These kinds of tests achieve their goal by having their participants (usually five to ten) interact with a prototype of the interface, following a structured set of tasks [12]. There are informal and formal ways of conducting a usability test; we will be analyzing those and their applications in chapter 5. For URU-S, we will be conducting informal usability tests coupled with a second phase that will feature a "Wizard-Of-Oz" experiment. We address this combination of informal usability tests plus "Wizard-Of-Oz" as "biphasic usability testing."

The "Wizard-Of-Oz" technique is a prototyping approach that allows a researcher to emulate the end behavior of an interface feature before the back-end system related to that feature is working entirely. The researcher that operates the system behind the scenes is known as the "wizard,"

and, as technology develops, their influence over the system diminishes because it becomes more autonomous [13]. One may consult an illustration of how the "wizard" bridges the technological gap in figure 2.1. We may take advantage of this technique to compensate for the fact that URU-S does not include automatic danger-detection technology, simulating warnings in our user's devices remotely instead.

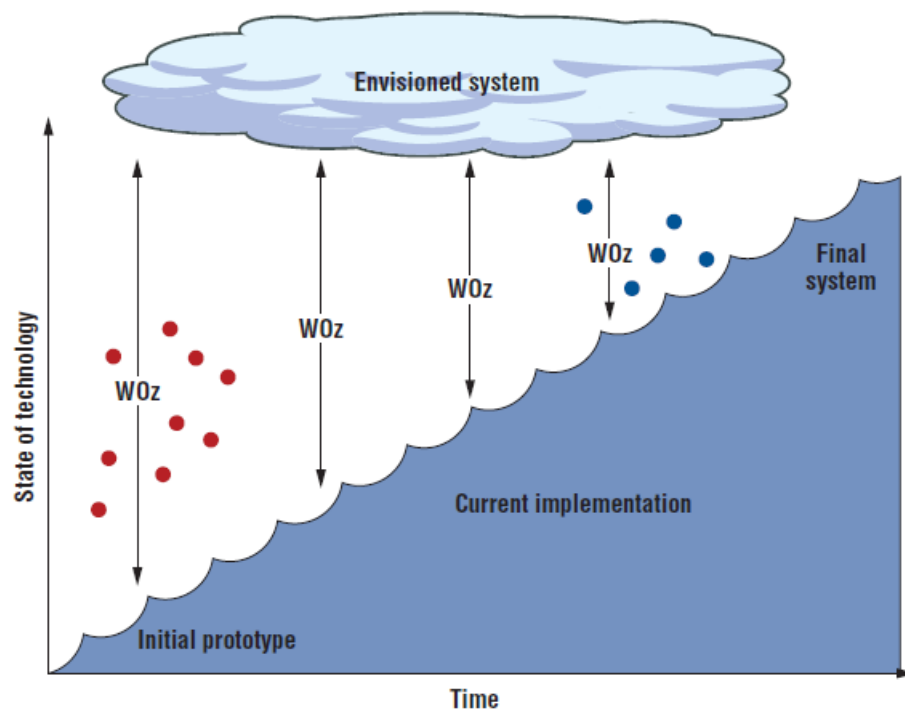


Figure 2.1: Scheme Illustrating the advantages of "Wizard-of-Oz" experiments [13]

2.3 Architectures and Relevant Technologies

Various architectures can be used to design a VRU protection system. For simplicity's sake, we will be grouping them into three main categories: direct connection between actors (represented in figure 2.2 as (1)), cellular communication (represented as (2) in figure 2.2), and MEC (multi-access edge computing) with a central server (represented as (3) in figure 2.2).

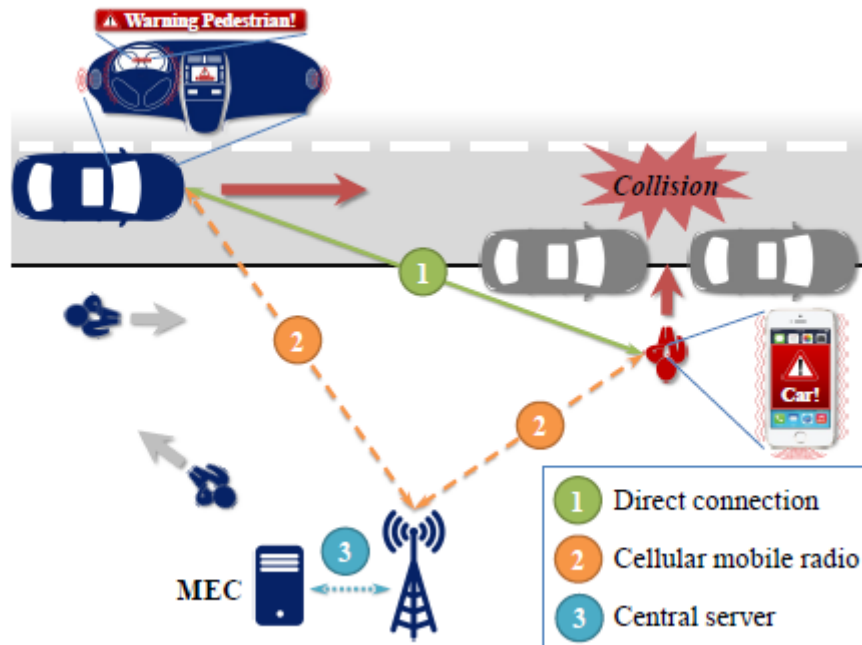


Figure 2.2: General concept of a VRU safety system. [14]

2.3.1 Direct Connection

Direct connection architectures involve communicating directly between system actors via sensors equipped in each participating entity. These communications can be one of the following: V2V (vehicle-to-vehicle), V2P (vehicle-to-pedestrian), V2C (vehicle-to-cyclist), V2I (vehicle-to-infrastructure), and even V2X (vehicle-to-everything). The communication is often bidirectional; for example, a V2P system would conversely have a P2V (pedestrian-to-vehicle) component. A system may include multiple types of communication at once or even all of them (V2X).

2.3.1.1 V2V

These works are predicated on the assumption that the vehicle is entirely responsible and is regarded as the leading entity in the system, communicating exclusively with other vehicles in its general vicinity. Many of these systems involve using sensors attached to the car and communicating with nearby automobiles to prevent crashes [15][16][6].

Many sensors utilizing vision, such as LIDAR and radar, have issues detecting other vehicles or pedestrians in scenarios with no line-of-sight; this problem could be remedied using V2I communication; however, that can be expensive. Some articles have proposed novel onboard sensing mechanisms [6], whereas others have used wireless communication to exchange Cooperative Perception Messages (CPM) [15]. An illustration of this may be found in figure 2.3.

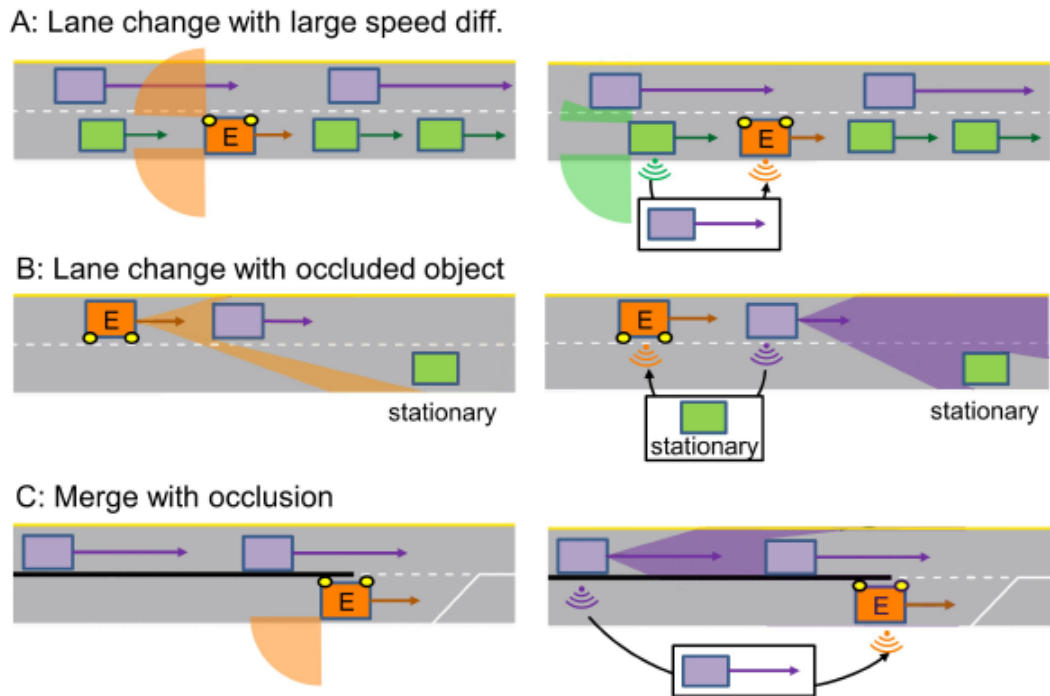


Figure 2.3: Left column: Three scenarios in which the ego vehicle's sensor limitations may compromise safety. Right column: How collective perception messages may improve the safety of the ego vehicle for each scenario. [15]

2.3.1.2 V2P

V2P is one of the main fields URU-S will focus on, the communication between vehicles and pedestrians. The articles surrounding V2P usually focus on swapping messages between the vehicle's onboard unit and the pedestrian's smartphone (or wearable) bidirectionally. These messages may come in the form of Basic Safety Messages (BSM) [7] or Personal Safety Messages (PSM) [5] (see also figure 2.4) that include positional information about these entities. Once the devices have harnessed messages, they may run their local collision detection algorithms and warn the driver, pedestrian, or both of imminent danger.

Some of these implementations integrate a technology widely used in this field known as DSRC (Dedicated Short-Range Communications). However, smartphones do not naturally come with hardware equipped to handle DSRC. Often it will have to be implemented through alternative methods like those used by Wu et al. [7]. Another example of DSRC, but now in accordance with the WAVE framework, is the work done by Kim et al. [5], also featuring direct communication, but now testing with an external DSRC module attached to the device. Since these solutions are local, they may use an architecture like the one represented in figure 2.5, which runs several modules locally. These modules could include context-awareness components (explored in further detail in subsection 2.4.3), the collision detection algorithm runs at the phone's level, and a DSRC implementation (if they are not using an external module).

Other ways of achieving this include the use of sensors attached to the car and the pedestrian, exchanging movement vectors between the two [17], the use of Wifi-Direct in a peer-to-peer network [18], and even the use of Bluetooth Low Energy (BLE) [18]. It is noteworthy that V2P communication does not necessarily need to be direct (device to device), as there are cellular and MEC alternatives. Those architectures will be discussed in the following sections.

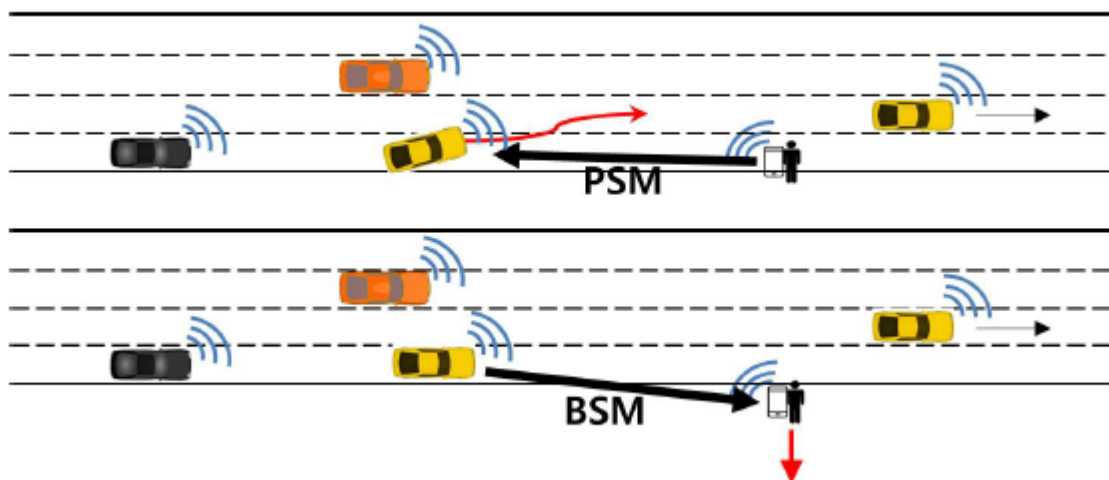


Figure 2.4: Communication allowed between vehicles and people in WAVE: person to vehicle (above); vehicle to person (below) [5]

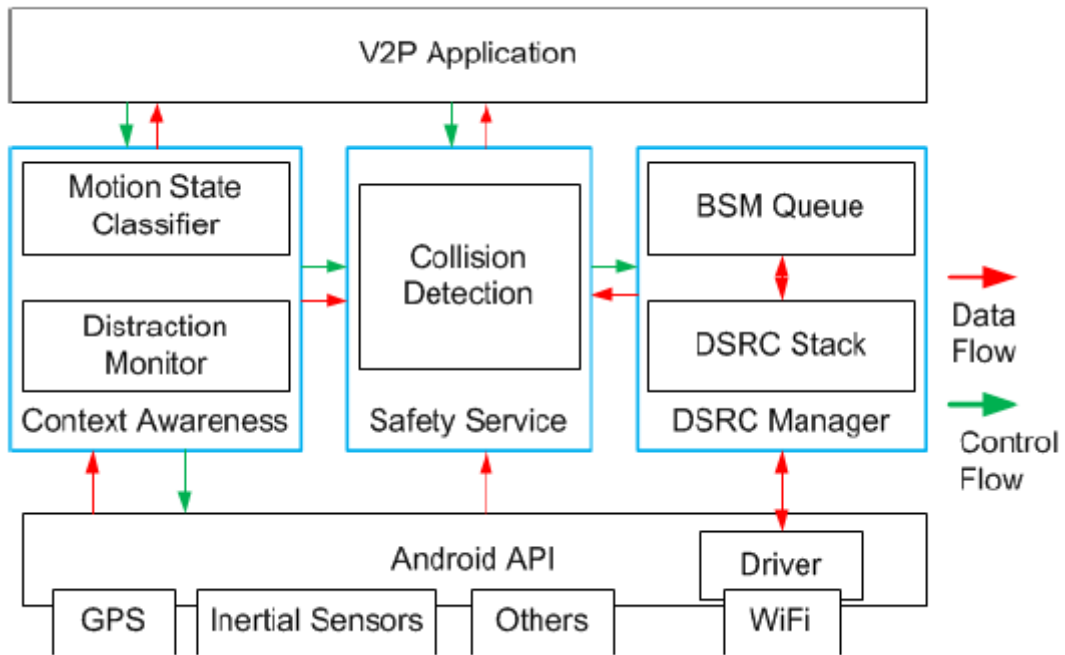


Figure 2.5: V2P system design in smartphone [7]

2.3.1.3 V2C

Most works in terms of cyclists tend to focus on accurately modeling a cyclists' behavior and integrating them in widely used systems that focus primarily on vehicles. Examples of these technologies would be the Vehicular Ad-hoc Network (VANET) and Advanced Driver Assistance Systems (ADAS). It is challenging to model their behavior without potentially placing them under dangerous situations; thus, a novel solution was proposed: simulate their behavior in a Virtual Cycling Environment (VCE) and assess VANET solutions' impact on their safety [19]. These studies even tried out different stimuli to figure out which would be best to warn cyclists [20]. The extended architecture of this system can be found in figure 2.6.

To further cement V2C communication, studies have aimed to extend established ITS standards to meet cyclists' special safety needs; these present extensions to the "ETSI ITS-G5 CAM and DENM message formats." [21] CAMs are Cooperative Awareness Messages, and DENMs are Decentralized Environmental Notification Messages. As they are defined in ITS-G5, these messages have mainly been used by motorized vehicles, lacking the necessary details to include cyclists.

Additionally, studies have been performed to "characterize experimentally the wireless link performance and develop a model to estimate the received signal strength (RSSI) between WiFi devices installed on bicycles and cars equipped with built-in WiFi APs" [22]. Evaluating these communications' performance is vital to integrate V2C systems in real urban scenarios successfully.

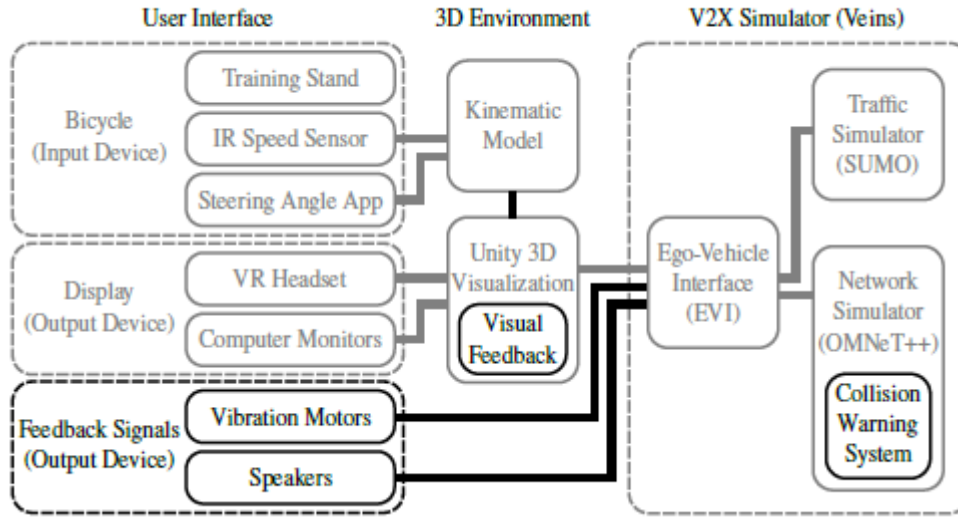


Figure 2.6: Extended architecture of the VCE with the newly added Collision Warning System (CWS) and signals (highlighted in black). Signals are provided to the cyclist via vibration motors (haptic), speakers (audio), or a virtual display in the Unity 3D visualization (visual). Collisions detected by the CWS are forwarded to the signals via the EVI. [20]

2.3.1.4 V2I

Vehicle-to-infrastructure communication is mainly used alongside vehicle-to-vehicle communication to bridge the gaps left by the latter. Vehicle sensors often have blind spots and issues with line-of-sight [6]. These issues could be alleviated by having vehicles share information between themselves and nearby infrastructures. This information could be GPS-related, such as latitude, longitude, speed, and heading, or vehicle-related, such as brake events, throttle position, and turn signal status [23].

Besides helping with sensory blind spots, traffic infrastructures, particularly roadside units (RSU), could also be used to compute collision probability in intersections by making vehicles broadcast their positioning and movement information between themselves or with RSU [24]. The RSU will use the received data to predict the vehicle's path and, thus, potential collisions.

2.3.1.5 V2X

Lastly, it is worth mentioning that vehicles may combine all of the previously mentioned types of communications into larger vehicle-to-everything systems. V2X is an effort to overcome the issues with local sensors [4], having the vehicle leverage technology around it to maximize system performance. A study, in particular, recognizes the limitations of technologies such as LIDAR and radar and attempts to use CAMs to form a cooperative perception system. This system harnesses data from onboard sensors to effectively detect objects and forward that information through V2X networks so that it may reach other vehicles and prevent accidents (see figure 2.7).

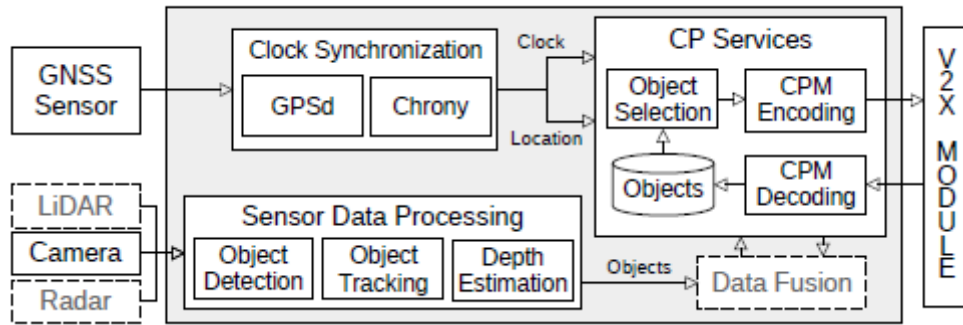


Figure 2.7: Cooperative perception system [4]

2.3.2 Cellular Communication

Over the years, a significant emphasis has been put on Cellular Vehicle-to-Everything (C-V2X) technologies, particularly in the V2P domain. C-V2X allows different devices to interconnect and device automation on the vehicle [25], which is particularly useful for designing road safety systems that can deal with several different types of communications.

LTE-based V2P services, such as V2PSense [25], may leverage the low-latency and pervasive coverage of 5G/LTE to timely warn pedestrians of imminent danger. These services can either be network-based (not scalable) or client-based (have troubles dealing with time and device diversity). They can implement LTE high-priority bearers, which are used by VoLTE (Voice over LTE) signaling to communicate, and they are centered around three main entities: a V2P application server, which could be deployed in the core LTE network or on edge; the vehicle's ADAS system; and the pedestrian's smartphone. The way these systems function is as follows: first, the pedestrian's smartphone and ADAS attach to the LTE network through the LTE base station (eNB), then, in the network, LTE gateways forward data packets between the eNB and the Internet or the server [25]. Once a dangerous vehicle approaches an intersection, all pedestrians around the area will be notified of the potential dangers.

Another possibility for cellular collision avoidance is using "context filters" that identify vulnerable road users in potentially dangerous situations based on their immediate context. It is possible to have this fine-grained to the point of recognizing a user stepping over the curb and onto the road as shown by Jahn et al. [26].

2.3.3 Central Server (MEC)

Central server architectures have been adopted recently to improve collision avoidance systems' effectiveness, addressing local devices' computational power and energy limitations. A generic MEC-based architecture forwards Basic Safety Messages (BSMs) containing relevant positioning information to Points of Access (POAs). POAs send this information to a collision detection server that will notify both the driver and the pedestrian if there is danger, communicating in the opposite direction. An illustration may be seen in figure 2.8.

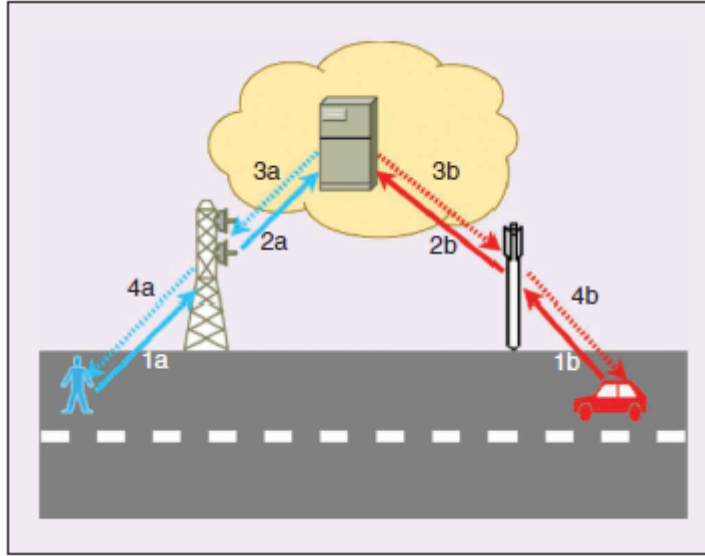


Figure 2.8: A conceptual view of the proposed MEC-based architecture. POAs of different mobile networks collect BSMs from both vulnerable users (such as the pedestrian in blue) and vehicles (such as the car in red) in steps 1a and 1b. POAs then convey the BSMs to a collision-detection server (steps 2a and 2b), which processes them and establishes whether there is a collision risk. If there is, alerts are conveyed to the appropriate POAs (steps 3a and 3b) and, hence, to the interested users (steps 4a and 4b). [27]

The way these solutions decide to handle BSMs internally may vary. For example, they may use an "obsolescence" system, where obsolescent BSMs are not factored into the algorithm. To determine if a BSM is obsolete, one may compare the timestamp associated with that BSM and check it against the current time, discarding it if it is over 0.8 seconds old. The information included in the current BSM is sent to a data structure, abstracted in the article as a "table," that contains positions and speeds of users that recently sent one of these messages. The positioning information from the tables is iterated upon and later fed to a collision detection algorithm. This algorithm is reasonably straightforward when not accounting for acceleration, using Euclidean distance and movement equations to determine a potential crash. With negligible acceleration, one only has to solve a polynomial of the second degree. The algorithm spikes in its complexity and requires a fourth-degree equation when acceleration must be considered. The article presents only a simplified version of the algorithm used and not the full scale, but it reassures the reader that the complete algorithm is more complex, versatile, and works on curved and straight roads. Finally, once the algorithm has calculated the probability of a crash using the information from the BSM, the appropriate alerts will be sent out if necessary [27].

Another way to implement a MEC-based architecture is to use CAMs and DENMs instead of BSMs. CAMs may be exchanged between vehicles and pedestrians via cellular communication with an LTE base station (eNodeB). In these systems, the User Equipment (UE), meaning vehicles or smartphones, will frequently report their existence to nearby eNodeB through CAMs. Base stations will forward these messages to other UE in the same context. UE that receives CAMs

will perform a collision detection algorithm (CDA) locally, triggering the right actions if necessary [14] (this is the so-called "Baseline approach"). However, some improvements can be made. For example, at a data level, smartphones may decide whether or not to migrate their context information calculation to the remote MEC server, offloading some of their raw sensor data instead of managing it locally. Nevertheless, now at the service level, another improvement is to offload the collision detection algorithm calculations to the MEC server, waiting for a DENM response if potential collisions are detected. An illustration of these approaches may be found in figure 2.9.

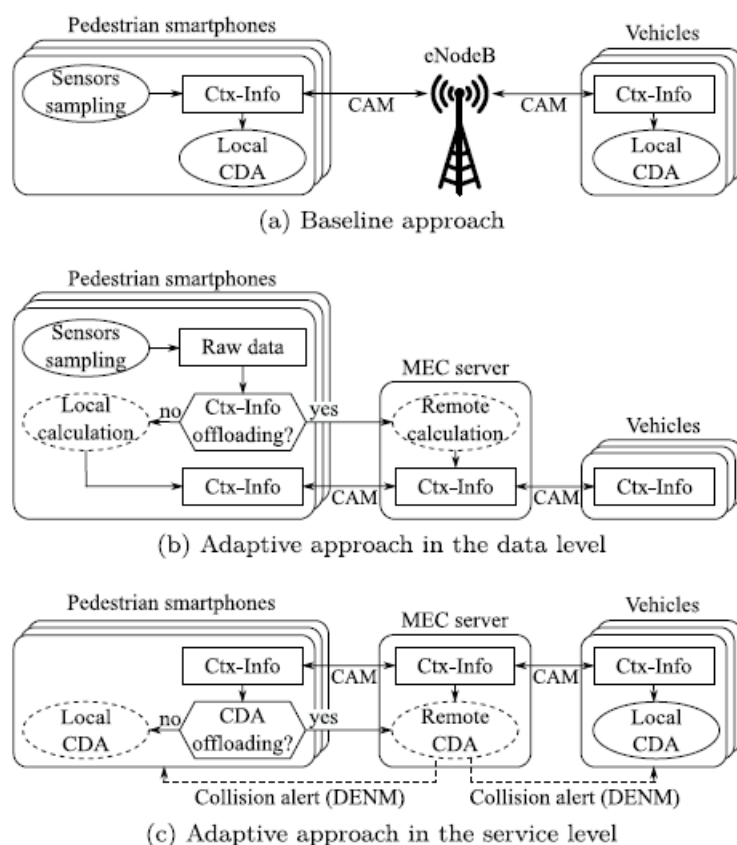


Figure 2.9: MEC-based collision detection. [14]

Lastly, another approach to the problem, using CAMs, consists of periodically sending CAMs to a CAM server, which then parses those messages through the appropriate module ("Listening Cam module") and stores the sender's information. The distance between VRUs and nearby entities is then computed using a formula of ellipsoidal Earth projection approved by the FCC (Federal Communications Commission); this is all done in Napolitano et al.'s "Geo-Computation Engine." Finally, the "Logic Sending Alerts" module will alert the involved road entities if the distance between them is less than a fixed threshold. Additionally, a "Client for Demo Viewer" module was created to send the needed information to a "Control Console" so that the entire process may be visualized in a browser [10]. A block diagram of the architecture may be found in figure 2.10.

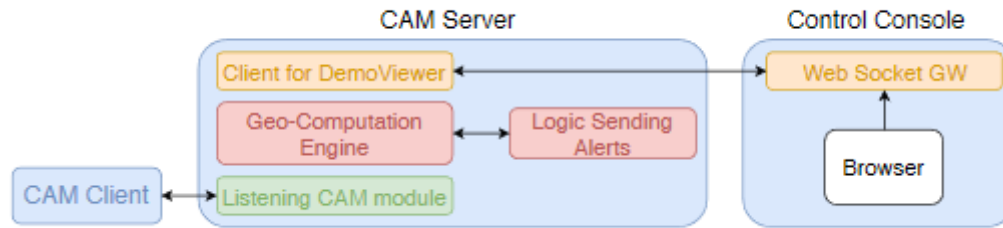


Figure 2.10: Block Diagram of the proposed architecture. [10]

2.4 Related Work

The related work can be grouped into six subsections: those that deal with positioning and context-awareness of the VRU, that propose more of a user interface to warn them in some way, that present more of a literature review on the field or architectures/requirements, that are more related with the automotive side of things, those that use edge computing, and even some psychological studies on human reaction to different alert stimuli (particularly more on the cyclist side of things).

2.4.1 Vehicle-centered Solutions

Autonomous vehicle technology is becoming more and more prevalent these days. This section will be brief compared to others since this work tries to detach itself from vehicle-centric ideals and actively involve the VRU.

Vehicle-centered work usually makes use of V2V communication. It may involve, for example, ways to compute the trajectory of a vehicle, which could be integrated into autonomous (or semi-autonomous) platooning. Despite not being directly related to VRU protection, these systems show the potential of autonomous vehicle technology. In these cases, the leading vehicle of the platoon is driven manually, whereas all of the others follow suit in an organized fashion, maintaining a safe distance between themselves. It can be applied to trucks, like what was accomplished by Lee et al. [16].

As mentioned previously in section 2.3.1.1, a car's onboard sensing mechanisms can be limited, particularly when it comes to situations with no line of sight. However, work has been done to use novel wireless sensing mechanisms that do not struggle with line-of-sight problems. The collision avoidance is done mainly through the car's sensory mechanisms. It does not involve pedestrians or cyclists as active entities [6].

Despite the inherent vehicle-centrism of this field of communications, there are attempts to include the VRU in simulators. Often, vehicular systems are tested using simulators, such as the VEINS simulator system framework. Before, there were no proper methods to include a VRU in these simulators. However, recently there have been some innovations in the code of existing tools to integrate "pedestrian" entities [28]. Even though this does not exactly classify as involving a

VRU directly in a protection system, it can be valuable to evaluate said systems through realistic simulated scenarios.

To conclude, it is worth reiterating the concept of perceived safety. As mentioned in 2.3.1.1, there are systems that, through the use of Cooperative Perception Messages (CPM), may determine the safety condition the vehicle is in, anticipating possible crashes by communicating with surrounding vehicles. The use of CPM is also proven (via simulation) to enhance connected automated vehicles' ability to perform safe lane changes [15].

2.4.2 Requirements and Architectures

The articles referenced here primarily supply an overarching view of the automotive/VRU world, enumerating past challenges and, in some cases, offering novel suggestions for dealing with them.

One of the goals of this project is to study past solutions and highlight some open issues in the VRU-protection-via-smartphone field. To that end, extensive literary studies about "outdoor localization and next step/movement prediction" via the use of smartphones were consulted. An example of this is what was accomplished by Vourgidis et al. [2]. Their work identifies gaps that could be expanded upon and briefly discusses some of the challenges related to the field mentioned above. Like other articles, it explores the possibility of using the sensors on the phone to transmit the pedestrian's position and heading to surrounding vehicles, prioritizing the accelerometer, gyroscope, and magnetometer. It includes a summary of vehicle-to-pedestrian systems that discriminates the sensors used, methods (e.g., collision risk evaluation, pedestrian behavior, dead reckoning, fusion), and the system type (e.g., collision avoidance, driver detection system). Several outdoor VRU positioning methods are described: GPS, Assisted GPS, Differential GPS, Multi-Satellite Systems, and Sensor Data Fusion. Finally, it includes some research regarding predicting the future position of a pedestrian in the next few seconds and proposes a framework for these kinds of systems.

It is of immense relevance to consider the architectural implications of integrating a VRU in C-ITS; predominantly, it would be ideal to have an architectural description that would elicit the interactions between actors at different levels of abstraction. Such an architectural description was already accomplished by Scholliers et al. in their paper [3]. Their work elaborates upon an architecture for the integration of the VRU in C-ITS, including requirements for the devices. This architecture describes the roles of and the interaction between actors (e.g., pedestrian, cyclist, vehicle driver) and components (e.g., VRU Transponder, Vehicle Connected System, Central ITS Station). It can be seen in three ways: from the physical standpoint ("high-level description of physical ITS sub-systems"), the functional standpoint (description of functional elements within those sub-systems), and from a communications standpoint (the "type of communication and networks used between functional elements"). Additionally, it lists and describes specific issues and their effects in these systems, such as road topology, location of the VRU, traffic infrastructure, visibility, compliance with traffic rules, context-awareness, range, latency, scalability, and standardization of messages exchanged. For some of the problems mentioned, it suggests solutions: in terms of position accuracy, it suggests using Global Navigation Satellite Systems (although

smartphone receivers for those systems have lower accuracy than those of cars). For context-awareness, it suggests sensor fusion and that the app should only function when the pedestrian is moving, saving power. For range, it places a range requirement of about 100 to 160 meters (although it acknowledges it can be hard to achieve with available hardware). For latency, a requirement of 300 milliseconds end-to-end latency time is placed; this can be attempted with 5G. Finally, it recommends Decentralised Congestion Control; however, it recognizes that scalability requirements end up being the most challenging and that LTE networks may be overloaded. It is worth pointing out that this work also presents a "roadmap" for these applications. This "roadmap" mentions technical performance, standardization, and privacy requirements, as well as the importance of a "non-distracting" user interface and an "efficient warning" strategy that contribute to a pleasant user experience (our main focus). It goes on to place applications in this domain into three phases: phase 1 includes basic applications; phase 2 has more of an advanced warning and sensing mechanism; phase 3 boasts fully cooperative sensing with "high-quality risk management" and "a low amount of false or missed alarms." URU-S will be more of a phase 1 application, as it is a reasonably limited prototype.

For the enhancement of pedestrian safety systems, the Wireless Seat Belt [17] was developed. Their article mentions requirements for ideal VRU protection systems in terms of position accuracy. It moves on to "experimentally show what can be done with current smartphones" (in terms of the aforementioned requirements). By relying on the "exchange of movement vectors between vehicle and pedestrian," they tried to evaluate if current smartphones could achieve an ideal accuracy level. The article found that, in normal conditions, the goal could not be achieved. For that, they offered two solutions: the possibility to detect the curb, which has been proven feasible in work done by Jahn et al. [26]; and using a technology known as Real-Time Kinematic positioning. Ideally, combining both of their approaches would lead to a more acceptable level of positioning accuracy.

Another vital requirement to consider is presented in work done by Morold et al. [29], regarding how time delays for both detection and communication of VRU contexts may affect the ability to detect collisions. Notably, this author also considers the option to detect a curb, presented and discussed in yet another project [26]. This article found that, for a positioning inaccuracy of 0.5 meters and 1.0 meters, the delay must not exceed 100 milliseconds and 300 milliseconds, respectively. For higher inaccuracies over 2 meters, the curb detection module can still increase the collision detection probability, even considering delays up to 806.25 milliseconds.

Lastly, it is worth mentioning work done in terms of transmission control. Even though this is technically not a literature review or an explicit "architecture"/requirement, it answers a typical problem that these kinds of systems have; channel congestion. More specifically, transmission control offers essential solutions to this problem. These solutions consist of: "defining accident models," "estimating the degree of risk a pedestrian is in by exploiting their context information," only transmitting on a channel when they are, in fact, in danger, and "differentiating pedestrian transmissions with different transmission intervals and channel access priorities" depending on their level of risk. This way, high-risk pedestrians have priority over the channel, and their mes-

sages are not lost. Experimental results have proven positive, improving packet delivery in high congestion scenarios (copious users transmitting on the same channel) [30].

2.4.3 Positioning and Context-Awareness

This field deals with using the device's sensors to predict pedestrian trajectories accurately. Usually, this field's work fuses input from the magnetometer, accelerometer, and gyroscope with or without GPS to infer valuable positional information about the VRU's whereabouts. This information can be fed into a dead-reckoning algorithm (determining the following position based on the previously known position) to determine where the pedestrian may be going.

Recognizing activities is a crucial aspect of any context-awareness mechanism. Based on the work done by Bocksch et al. [31], we see that it is possible to recognize activities such as standing, walking, running, lying, cycling, throwing, and entering or leaving a car. In this paper, the smartphone inertial sensors' fusion allows the technology to identify each activity using a linear classifier and decision tree approach. The algorithm managed to classify activities correctly with an accuracy of 91%, boasting a low complexity algorithm that uses low-cost sensors and is infrastructure independent. Such an algorithm could be helpful in the localization of the VRU in pedestrian (or cyclist) protection systems.

Furthermore, collision detection algorithms rely on knowing where the pedestrian is heading and where they might end up. The work done by Kotte et al. [32] may help shed some light on this issue, proposing enhanced pedestrian behavior models to predict more accurately where the pedestrian may end up. The article describes how they fused information from the inertial sensors on the phone with personal information of the pedestrian and even with their history, feeding it into a cost-function-based pedestrian trajectory prediction algorithm that estimates "the probability of a pedestrian's presence" in the next seconds.

Finally, as a testament to how specific context-awareness can get, it is worth mentioning the work done by Morold et al. [26], where we have an approach that attempts to recognize pedestrians stepping onto the road (crossing the curb). It identifies the problems existing in this field, such as inconsistent sensor data, over-representation of periodic activities (walking, cycling), and evaluation of the recognition. It uses a "context filter" to identify VRU in potentially dangerous situations. The results show that detecting the subtle movement of "stepping onto the road" is possible.

2.4.4 Warning Interface

These are examples of systems that consider the VRU as an active entity in collision avoidance, involving them in some way or another and proposing an interface to warn them. It is this field specifically that this dissertation aims to develop more.

An example of a complete system is described in work by Wu et al. [7], which gives 360 degrees, extended range, no line of sight required, where the driver and pedestrian alike are warned of a potential crash. It uses DSRC (Dedicated Short-Range Communications) to function. It deals

with issues, such as false positives, channel congestion, security, and localization, recommending novel ways to treat them.

DSRC is a technology not commonly found in smartphones. The approach mentioned in the previous paragraph had to implement a DSRC stack within the smartphone's WiFi chipset, leveraging the phone's GPS and inertial system. It contains three main modules: one for context-awareness, which allows it to tackle power-saving and channel congestion by only using power to occupy the channels when the VRU is effectively moving; a DSRC Manager for handling incoming or outgoing Basic Safety Messages; and a Safety Service that implements the collision detection algorithm used.

The article mentioned above even includes a Distraction Monitor in the system that "detects whether the pedestrian is engaged in potentially distracting activities" (e.g., texting, calling, listening to music) to adjust the safety algorithm threshold. However, despite all of the technological ingenuity, the user experience tied to the pedestrian's warning system is relatively poor. The interface is far too technical. The collision warning blocks the entire screen and blares out loud noises, all while showing bright yellow colors on the screen.

Another perspective would be the work done by Kim et al. [5], where the author implements a novel communication system through Personal Safety Messages sent by the user's smart device. The system implemented here attempts to protect the most distracted road user type (the ones continually staring at the phone) by scanning for nearby Basic Safety Messages sent by vehicles. The work's entirety is based on the "IEEE Wireless Access in Vehicular Environment (WAVE) standards." It represents danger via a "blob" on the corners of the screen. This "blob" is meant to track the vehicle's side as it passes by the user.

The "blob" warning, however, has some issues with it: firstly, it adds some computational overhead to the phone because knowing where to draw the "blob" relative to the screen, and the passing car involves calculations that factor in the yaw, pitch, roll, and others. Secondly, it depends on smartphone sensors to get the data needed for these calculations. These sensors can be imprecise, causing the "blob" to appear jittery on occasion. Finally, the "blob" is not very descriptive of what is happening. The "blob" can take three colors, indicating if the vehicle is close by or far away; the user must memorize what each colored "blob" means. Presumably, the color's meaning is documented somewhere in the application, like in a manual, although that is never explicitly stated in the article. As per Jakob Nielsen's sixth heuristic, "recognition rather than recall," user interfaces should be immediately recognizable and not appeal to recall of a manual description [33].

Another interesting project was developed by Titov et al. [18], which uses Bluetooth Low Energy and WiFi Direct in their implementation. Several "accident-pandering factors" were derived. Five "typical" scenarios that cause accidents were created based on environmental, sociological, and other factors. Each technology's performance was evaluated in each scenario. Bluetooth proved to cover shorter distances with high latency and only three messages each second, having low accuracy but good battery economy. WiFi substantially increased road users' visibility, boasting a considerable connection distance and rapid peer detection and connection establish-

ment. However, it had high battery consumption compared to the alternative. It also required user interaction to accept each first connection. The prototype mentioned above consisted of a simple Google-Maps-like overlay with an add-on that would draw a "conflict area" where a collision may occur. The article discusses the technical aspects and technologies used and does not divulge much about the interface.

More on the cyclist side of things, there were attempts to integrate haptic (tactile) signals with V2X-based Safety Systems. The integration of said signals was done using an extended Virtual Cycling Environment that can use tactile vibrations to inform cyclists of imminent danger in complex traffic situations [20]. Visual and audio signals were also used to detect possible safety-critical situations. The researchers found that "the use of vibrations turns out to be suitable for giving directional cues" [20]. Most participants found that the warning vibrations were helpful and improved their reaction time compared to not being warned. Visual cues were relatively inefficient and distracting, whereas audio signals were not very significant.

Last but not least, we have V2PSense, a cellular-based warning service that "trades off positioning precision for energy savings while achieving low latency" [25] values. For this, high-priority LTE bearers are used in conjunction with intermittent GPS (keeping the GPS always-on would increase precision but drastically drain the battery). Like other articles, it leverages the phone's sensors to infer pedestrian movement and combines it with an analysis of cellular signal strength changes to determine where the VRU may be going next.

The entirety of V2PSense operates on the notion that one "does not need to know precisely where each pedestrian is" [25] to estimate a collision. The disregard for a precise position is why it uses intermittent GPS and mobile sensing data to determine a Pedestrian Arrival Area (a general area where the VRU may end up). The device's signal strength variations are then used to curtail this area successively until it can accurately anticipate where the VRU will be. Should this area overlap with a dangerous spot, the pedestrian would be notified on their smartphone. It is to note that it can achieve "92.6% precision ratio with 20.8% energy saving" [25]. Despite this, even this last work does not precisely mention how it notifies the user or what interface is available (if any). It mentions that it runs as a "background service," which would not be ideal. In Android, foreground services should be used instead due to operating system specifications. According to the official developer documentation, "on devices running Android 9 (API level 28) or higher, apps running in the background have the following restrictions: sensors that use the continuous reporting mode, such as accelerometers and gyroscopes, don't receive events. Sensors that use the on-change or one-shot reporting modes don't receive events." [34].

Nevertheless, as we can see from the previous examples, this field lacks a more dedicated warning system that puts the VRU first in Human-Computer Interaction.

2.4.5 Mobile/Multi-Access Edge Computing

Edge Computing is a notable contender for a place within the automotive industry, particularly in road safety systems. The Cloud has potentially infinite computational power, reducing overhead

on existing devices and boasting low latency values (presuming the access points are nearby). These solutions are particularly convenient for implementing collision avoidance mechanisms.

First and foremost, in work done by Malinverno et al. [27], we are presented with a collision-avoidance system that works with Basic Safety Messages (BSM) that relay positioning, direction, acceleration, and even speed of any road user to a cloud service. BSM can be obtained from smartphone sensors, and vehicles can send them through their onboard units. These messages are factored into a collision avoidance algorithm; more specifically, they are sent to the network infrastructure point of access and combined within it to determine whether or not to send alert messages that will notify both participants if a potential crash is detected. The solution achieves low latency while conserving power on the phone and car, being very flexible and capable of integrating several different technologies. This work demonstrates a much higher detection rate than previous solutions that used more traditional architectures not involving MEC, detecting collisions on time (taking into account reaction speeds). However, it is not perfect, as it does report a slightly higher number of false positives.

There is also a study where the authors explore possibilities for exchanging "relevant safety mechanisms" in VRU-to-vehicle communication, using LTE networks and "Multi-access Edge Computing (MEC)" to run collision avoidance algorithms and even process some context information [14]. Offloading some computation spares the phone's energy supply and can also improve overall latency. The study showed successful results in offloading some computation to MEC servers, drastically reducing energy consumption. However, it is essential to note that this article achieves context-awareness through a Machine Learning algorithm that can also be offloaded to the MEC server. Several strategies were used depending on what is being offloaded or done locally on the smartphone. These may vary between local context-awareness, offloaded context-awareness, local Collision Detection, and offloaded Collision Detection. Out of these combinations, offloaded context-awareness calculations and offloaded Collision Detection proved to be quite advantageous in draining less battery power than any other alternative. Despite that, the pure offloading solution had poor latency performance, which is undesirable for a safety-critical system such as this one. In this case, to guarantee the timeliness of the warning messages, it is best to opt for a more local scheme, as it is more critical to timely warn the VRU than it is to save energy on the phone. However, part of the computation can still be offloaded to spend less energy than the purely local variant.

2.4.6 Psychological Studies on Human Stimuli

Some studies have been conducted on the most effective way to warn a cyclist of incoming danger, specifically how that would affect their behavior. Most of these derived from the establishment of a VCE platform that offered a means to model cycling behavior and improved bicyclists' safety at intersections [19].

It is essential to evaluate the psychological feasibility of using the VCE for human experiments. A study was conducted on cyclists to determine a method for measuring their attention [35]. This was a fundamental study for more advanced VRU safety systems to determine the best form

to warn these users effectively. It is essential to understand how they react to these warning systems and which signals are easily understood. The paper focuses primarily on visual attention. The entirety of the test was done in a VCE, and the results showed that high road traffic density required an excessive amount of visual attention compared to low traffic density. These results were validated by trials on the mental workload of road users and proved the VCE's potential to be used in "human-in-the-loop" setups.

Lastly, it is worth mentioning that there are studies on human visual attention as a whole, specifically on the theory of visual attention (TVA). In work such as the one conducted by Krueger et al. [36], TVA's advantages are analyzed to determine its potential for exploitation in cases where the degree of control over stimulus varies, including experiments with mobile devices. According to the previously mentioned paper, TVA parameters can be measured with sufficient precision. It was also found that TVA can be used to model real-world tasks and infer temporal-order judgment (determining which of two different stimuli appear first).

2.5 Discussion Relative to Previous Work

After researching the field of VRU protection, we can conclude that only a few papers characterize VRU protection systems that involve these users as active entities, focusing primarily on the vehicle and sometimes not mentioning how they intend to warn pedestrians and cyclists. Work involving these users does not seem to be a byproduct of user-centered design, never mentioning how to develop a proper interface for users to interact with and be warned accordingly. Considering we are proposing a mobile solution that VRU will want to have installed in their devices, it is nonsensical to expect them to install a highly technical product that offers no or minimal context for the application itself. Some of these works, as seen previously, even require that the pedestrian must actively be staring at their phones at all times to function correctly, and others require that they enter IP addresses, displaying raw longitude and latitude coordinates.

Moreover, the communications and networking technologies used to implement these solutions all have their associated trade-offs. For example, Bluetooth solutions have good battery economy but low range and high latency, whereas WiFi-direct has poor battery economy but compensates for the latency and range. MEC-based solutions can offload computation from devices, preserving their energy, but that may come at the cost of higher latency values or even a more considerable amount of false detection. Cellular solutions can provide low latency values on good coverage areas. However, they could have availability problems; different operators may experience high end-to-end communication delay, and, according to ETSI TR 102962 [37], LTE networks have difficulties scaling. More specifically, LTE may be overloaded when under traffic from 20 vehicles per cell at a rate of 10 CAMS per second. Direct connection between the smartphone and other nearby dangerous entities requires additional hardware to be attached to the device, consuming more battery and raising device discovery and trustful association problems. Additionally, direct connection usually implies that collision detection is computed locally, which fixes the issue with latency but introduces another problem: spending more energy and processing power on the

smartphone itself, drawing that away from other applications. Local computation can also have a more limited view of the scene and thus be more prone to errors. There is a great degree of uncertainty associated with which technologies will work or not in the future. In fact, most of these prototypes never left the experimentation phase. These works are usually field-tested in laboratory settings or simulators like Eclipse SUMO (Simulation of Urban MObility). Judging from the research done when writing this document, none of these prototypes have been implemented in a real road scenario.

URU-S differentiates itself from previous efforts to conceive VRU safety systems by directly involving VRU in the interaction design process and studying which methods are best to warn them. Focus groups and usability tests were conducted to achieve user-centrism, making an effort to develop a high-fidelity prototype of a road safety system that these users would want to install in the future. These were instrumental in obtaining live feedback from participants. That feedback allowed us to shape the interface and alert system into its present state, focusing on human-computer interaction. We chose the HCI aspect as it seems to have been neglected in previous assignments, and the more technical ones are somewhat unsettled.

Chapter 3

Smartphone Based VRU Protection

The following section will frame the problem of trying to protect a VRU through a smartphone application. We include a description of what URU-S is trying to do on a more conceptual level, detailing our presumptions, constraints, rules, requirements, and tools. The enumerated assets will then serve to formulate our solution and approach.

3.1 Application Use Case and Assumed System Model

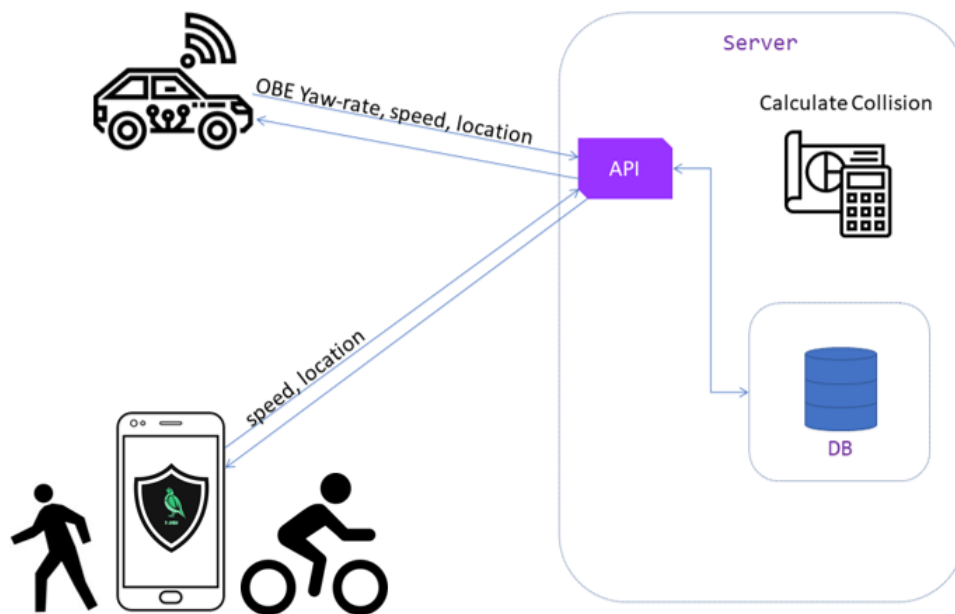


Figure 3.1: Simplified Sketch of the Presumed Architecture

This work focuses on the solution set within the smartphone itself, which would be integrated into a more extensive system. There are many assumptions associated with the monolithic perspective of a road safety application such as this one. In this case, we presume a Central Server

(MEC) architecture, like the ones referred to in section 2.3.3. Notably, as illustrated in figure 3.1, the phone would have to communicate with a server (preferably located nearby to keep latency to a minimum) that will calculate the probability of a collision. URU-S would ideally forward the user's speed and location to this server entity when VRU movement is detected. Forwarding of positioning information should only occur when the user is moving, and for that to happen, URU-S must be aware of the user's current activity. The application would then send the information by interacting with an appropriate API of some sort. Nearby cars would also send their relevant info to the same API. Internally, the server would run a computationally complex algorithm (MEC solutions such as those mentioned in the State of the Art could alleviate this burden). Should it determine that a collision will likely occur, it will warn both the VRU (by forwarding a warning to their smartphone) and the driver (sending a warning to the car's onboard unit).

3.2 Framing the Problem

At a conceptual level, the problem URU-S attempts to solve is effectively notifying the VRU of incoming danger (particularly collisions with vehicles, cyclists, or even pedestrians). Specifically, a non-intrusive notification must be fired off in the user's device in some way (preferably through the use of haptic technology).

Table 3.1: Technical Requirements

Identifier	Name	Description
TR01	<i>Accessibility</i>	The app should be accessible by all, regardless of their situation.
TR02	<i>Usability</i>	The app should offer a simple and easy to use interface.
TR03	<i>Security</i>	The app must not leak sensitive user information.
TR04	<i>Ethics</i>	The system must respect ethical software development methods.
TR05	<i>Robustness</i>	The system must be verified and checked against possible deadlocks.
TR06	<i>Context-Awareness</i>	The system should know (and only function) in scenarios where the VRU is moving.
TR07	<i>Battery Saving</i>	The app should preserve battery.
TR08	<i>Presence</i>	The app must use certain foreground services (shown in the status bar with an icon).
TR09	<i>Storage</i>	The app must store certain data.
TR10	<i>Portability</i>	The app should function on all Android devices.
TR11	<i>Scalability</i>	The system should work even with a large number of users interacting with it.
TR12	<i>Availability</i>	The system should be available 99% of the time 24 hours a day 365 days a year.
TR13	<i>Latency</i>	The system must have a maximum global end-to-end latency of 300 ms.
TR14	<i>Accuracy</i>	The system must try to have very little false positives.
TR15	<i>Range</i>	The collision should be detected 100 to 160 meters before it happens.

Based on previous literature and common sense (within the realm of application development), a series of technical requirements have been compiled. These may be consulted in table 3.1. The main challenges addressed in this work are usability (related to TR02 on table 3.1), context-awareness (TR06), and battery-saving (TR07). The usability aspect is vital. To guarantee it, users must be the center of the interaction design process, and notifications must be tailored to their needs and reactions. Context-awareness and battery optimization follow suit; these are closely related to one another, as the system should only drain resources "on the go" (movement detected)

to lessen the phone’s energy consumption. Additionally, the app must steer clear of any complex computations that may instill unnecessary overhead on the device.

Table 3.2: Constraints

Identifier	Name	Description
CO1	<i>Time</i>	We are limited by the time we have for development.
CO2	<i>Available Tech</i>	We are limited by tech to test or use and are constrained by its specifications.
CO3	<i>Sensors</i>	We are limited by current inertial sensing and global positioning technology.
CO4	<i>Channels</i>	We are limited by the communication channels we are allowed to use.
CO5	<i>Available Participants</i>	We are limited by the number and input from participants that can be gathered.
CO6	<i>Android OS permissions</i>	We are limited in terms of push-notification permissions given by the OS.
CO7	<i>Android API level 27</i>	We are limited by the API levels we can use, we can only use 27 or above.

There are also a series of constraints (table 3.2) that limit what can be achieved with our work. It is imperative to highlight available tech (CO2) and available participants (CO5). It was challenging to organize events that required active participants with the current health situation, dampening the statistical validity of the observations due to the reduced sample size. Although the tech available was sufficient to test the prototype, it is worth noting that different smart devices will have different hardware specifications and limitations that may hamper the application’s functionality in different ways. Different results could be obtained using phones with more precise sensors, more lax battery settings that allow URU-S to fetch sensor data at higher rates, and different software in terms of operating system API level or distribution. Two phones with the same hardware will behave in divergent ways if they run different systems with non-identical permissions.

Table 3.3: Business Rules

Identifier	Name	Description
BR01	<i>Suspend Notifications</i>	A user may suspend his notifications for a given period of time.
BR02	<i>Adjust Notification Settings</i>	A user may change the behavior of notifications to suit his needs.
BR03	<i>Ignore Notifications</i>	A user may ignore the notification in case of false positive.
BR04	<i>Battery Optimization</i>	A user may turn off URU in battery-optimization mode.
BR05	<i>Transparency</i>	A user may know everything about how we are accessing their information.
BR06	<i>Documentation</i>	A user has the right to consult functionality documentation.
BR07	<i>User-Specified Permissions</i>	A user has the right to alter URU-S notification permissions via the OS.
BR08	<i>Warranty</i>	A user has no warranty that URU-S will work as intended if tampered with.

Taking the two last paragraphs into account, it is worth mentioning a set of business rules (table 3.3) tied to this project. The most relevant business rules are user-specified permissions (BR07) and warranty (BR08). It is presumed that the user will not jeopardize the application’s functionality by altering the phone’s permissions, blocking our notification channels, or denying us access to their activity (e.g., standing still, running), location, or storage for alerts and custom sounds. Should the user limit us in this sense, we cannot guarantee TR01 (table 3.1), as that requires activity tracking or any use case that requires the user’s position, saving, or uploading anything. Should the user tamper with the permissions, there is no warranty that the system will behave as intended. URU-S may need to be switched exclusively to manual mode, functioning

with limited features. To clarify, "adjust notification settings" (BR02) refers to using the URU-S interface. BR02 is a rule regarding using URU-S to change the sound, vibration, and others; it should not be confused with BR07. BR07 refers to the user specifying permissions via the Android interface and altering the back-end logic of the application.

3.3 Proposed Solution and Approach

3.3.1 User Centric Design Methodology

The proposed solution is designing a smartphone-based VRU warning application that focuses on HCI aspects, which for the most part have been neglected, and developing a prototype of said application in native Android. The thesis's primary focus will be on the HCI dimension, providing an easy-to-use and functional interface that can successfully involve the user (through methods mentioned in section 2.2). It is imperative to have a user-centered design, involving them early on in the process through events such as focus groups and a series of usability tests.

The focus group will be mainly for requirements gathering purposes; several discussions will be stimulated between the participants and the researcher, based on a series of questions. These questions mainly consist of asking the users what kind of notification they would react to best, what they would like to see in an application such as this, and how the preliminary design may be improved.

Later on, biphasic usability testing will be conducted. In the first phase, our testing participants will be placed in a quiet and controlled room. The participants will sit down with the testing facilitator and get acquainted with the interface. A series of tasks will be presented. The participants are encouraged to think out loud and express their opinions. It is important to note what participants are trying to do and their understanding of the interface. The second phase will include field testing, and a simulated dangerous situation will arise. Notifications will be fired off remotely in the testing devices (in a "Wizard-Of-Ozz" style testing) whenever a dangerous situation presents itself. In the second phase, we will test the participant's reactions to different situations. The situations include false positives (the user is warned, but there is no danger), false negatives (the user is not warned, but there is danger), and true positives (the user is warned justly). In terms of metrics, primarily, the study will concern itself with the effectiveness of the notifications, the user's difficulty with utilizing the interface (on a scale of 1 to 5), the percentage of tasks completed, and the amount of time taken to complete said tasks.

3.3.2 Software Engineering Methods

Throughout this work, an Agile Scrum [38] approach was used, dividing the work into a series of sprints. A couple of product backlog items were selected to be worked on for each sprint. The product backlog itself was constructed throughout the project's development. The backlog contents and priorities were agreed upon with the supervisor (playing the "product owner," more

or less). Regular meetings were held with the supervisor to keep them updated on the development of the project. At the end of each sprint, a testable and stable version of the product was delivered.

First and foremost, an effective prototype was designed through low-fidelity methods, primarily wireframes. These wireframes were instrumental for the focus group, held shortly after their development. The focus group gave much crucial feedback to the prototype, inspiring another set of wireframes. The final set, and a list of use cases to include in the final prototype, were validated by the supervisor. Eventually, the interface's high-fidelity prototype was developed in Android, taking into account the established use cases.

This high-fidelity prototype was evaluated through a series of usability tests with a reasonable number of participants. Of course, the actual collision detection was simulated and triggered remotely by a third party. Due to time constraints and technical complexity, the application was technologically agnostic in relaying positioning information to a collision detection module or implementing such a module (these technologies presumably exist to be leveraged by the application in the future). Whenever deemed necessary, specialists were consulted to improve the prototype further.

Per TR05 on Table 3.1, the application must also be extensively verified. Model Checking would be the preferred formal method, preventing it from going into undesirable states ("Dead-Locks") that can occur. An extension to the Java Model Checker, JPF, has already been developed, called JPF-Android [39]. It is open-source and extensively documented; however, it has not been maintained in the last three to four years. JPF-Android is also quite limited and cannot verify the user interface, which is quite concerning as this project is preoccupied primarily with providing a user-centered interface prototype. Errors were obtained while trying to install the model checker, and upon trying to contact the maintainer, no replies were received. Over the years, there have been endeavors in applying formal verification methods to mobile applications [40][41][42]; however, these projects are either not open source or have been discontinued for whatever reason.

Another approach was taken to satisfy the robustness requirement since available formal tools were either outdated or unavailable. Many of these tools, an example being the TRIANGLE framework [42], leveraged existing testing frameworks that were a part of Android, automating them to generate several different test cases without burdening the developer with the task to create them manually. Examples of these native Android testing tools include UI Automator [43], Espresso [44], and instrumented unit testing [45]. Since obtaining access to the formal automated tools was impossible, a series of unit tests using the aforementioned Android framework tools were developed.

To further complement the robustness of the system, a stress testing tool was used. This tool is the UI/Application Exerciser Monkey [46], and it made it possible to inject series of pseudo-random events into an emulator (including clicking, swiping, and others) to test how URU-S would behave under stress.

3.3.3 Tools

Different tools were used to achieve different means to accelerate and streamline the development of the prototype. All of these tools are open-source and licensed adequately. The tools are:

- **NumberEight SDK:** The NumberEight SDK [47] is what allows the application to be context-aware, giving it contextual glimpses over time. The SDK delivers context information in the form of glimpses by using signal processing and sensor fusion algorithms. It is tied to a foreground service that subscribes to different types of glimpses depending on the situation.
- **Google Services:** Google Services supply us with the Maps SDK [48], allowing us to make use of Google Maps for our heat-map of dangerous areas and the Geofencing API [49].
- **Firebase Cloud Messaging:** Firebase Cloud Messaging [50] is what allows us to trigger warnings remotely in the user's testing smartphone. We achieve this by having a Firebase Messaging service running in the application and sending that service a data payload. The service is triggered by an external testing tool that runs locally on the testing facilitator's laptop.
- **Contrast Ratio Web Tool:** This tool calculates a ratio between a foreground and background color and checks if it is under the proper accessibility guidelines. It is worth mentioning that it follows Web Content Accessibility Guidelines (WCAG) 2.1 [51] and can check if elements of the URU-S interface contrast appropriately.
- **Testing Tool:** The testing tool is a simple React.js [52] localhost web application that we developed ourselves. This tool has a set of buttons related to each testing smartphone. Upon clicking one of those buttons, an HTTP request will be sent to a specific endpoint that uniquely identifies one of the testing smartphones. At a back-end level, that endpoint will have the Firebase token associated with the testing device. Triggering the endpoint will cause it to send a data payload to the Firebase Cloud Messaging server, eventually routing it to the testing device.
- **Lingver:** URU-S allows the user to change the language between Portuguese and English. As such, all its string resources are translated in both of the previously mentioned languages. To keep track and swap the application's locale, which determines which string resource to fetch, an open-source library, Lingver [53], is used.
- **Markwon:** To format URU-S documentation pages included in the app, like the application manual, an open-source library, Markwon [54], was used that allows programmers to embed markdown code into Android TextViews.
- **Donuts:** The circular graphs present in the statistics section of URU-S were drawn using an open-source library, Donut [55]. These display the percentages of true positives, false positives, and suspended alerts.

3.4 Use Cases

The following are all of the use cases derived from focus group participants and agreed upon with this project's supervisor. They have all been implemented and are accounted for in the final version of the application.

Table 3.4: Use Cases

Identifier	Priority	Name	Description
US01	High	<i>Protection Status</i>	As a VRU, I want to consult the app's status so that I may know that I am being protected.
US02	High	<i>Notification Volume</i>	As a VRU, I want to change the volume of the URU-S notifications so that I may adjust it to my liking.
US03	High	<i>Notification Suspension</i>	As a VRU, I want to suspend the URU-S notifications so that I may have control over the alert system.
US04	High	<i>Receive Notifications</i>	As a VRU, I want to receive URU-S notifications so that I may know that I am in imminent danger and avoid it.
US05	High	<i>Notification Vibration</i>	As a VRU, I want to adjust the intensity of the vibrations so that I may be more comfortable with them.
US06	Medium	<i>Language</i>	As a VRU, I want to change the language so that I may change it to one I am more comfortable with.
US07	Medium	<i>Options menu</i>	As a VRU, I want to access the options menu so that I may change configurations that are relevant to my interests.
US08	Medium	<i>Select Sound</i>	As a VRU, I want to select different sounds for the notification so that I may choose a sound that I prefer.
US09	Medium	<i>Customize Sound</i>	As a VRU, I want to upload my sound so that I may give the app my personal touch.
US10	Medium	<i>Access Statistics</i>	As a VRU, I want to access the statistics feature so that I may have an added component to the real-time alerts.
US11	Medium	<i>Access Heat-map</i>	As a VRU, I want to have a heat-map of the most dangerous spots so that I may know which places to avoid.
US12	Medium	<i>Receive Danger-Zone Notifications</i>	As a VRU, I want to receive a warning emitted whenever I enter a dangerous zone so that I may be more aware.
US13	Medium	<i>Disable Danger-Zone Notifications</i>	As a VRU, I want to disable the warnings emitted whenever I enter a dangerous zone so that I may control them.
US14	Low	<i>Neutralize Notification</i>	As a VRU, I want to signal the app that I have avoided danger, to count that as a true positive.
US15	Low	<i>Sharing</i>	As a VRU, I want to share the app with others so that I may tell others about it.
US16	Low	<i>Documents</i>	As a VRU, I want to consult the documentation so that I may know more about the app and who made it.
US17	Low	<i>Ignore Notifications</i>	As a VRU, I want to ignore URU-S notifications so that I may dismiss them in case of a false positive.

Chapter 4

URU-S Design and Implementation

This chapter will detail the implemented solution and its design. We include our user-centered interaction design process, the architecture, an overview of the essential functionalities, a more in-depth description of how each requirement was satisfied, the final screens of the interface, the flow between said screens, and the use cases they cover.

4.1 URU-S Design

User-centered design is a vital component that is reinforced throughout the entirety of URU-S. Previously neglected in past research, this aspect must be considered if we ever hope to convince VRU to install this application on their smartphones.

VRU protection systems are predicated on the assumption that the VRU will have an application like URU-S installed and running in their phones, using it in their daily lives. Users will only install and use a product that appeals to them in some way, meeting their expectations and providing a decent user experience.

To guarantee our users an optimal experience with URU-S, we must bring them into the design process of the application, especially in its early stages, to have them be the main driving force of the development. We accomplish their early involvement through focus groups. We then validate our product by having VRU test its usability in an environment that we can solicit feedback from to iterate even further on our design; these are our usability tests.

4.1.1 Preliminary Design (WireFrame)

The following is a wireframe conceived by considering the previous work done in the field (this work is mentioned in chapter 2). It served as an essential asset to explain the prototype to others during the focus group and stimulated discussion. This wireframe is a rough draft of the URU-S interface that can be iterated upon with participant feedback, altering and shaping it into a model that will later guide the implementation process.

In figure 4.1, we may see an optional sign-in and sign-up feature; this is to preserve the user's settings so that they may load these into other devices. The Home screen is also presented; the only things to note here are the menu and some text fields. The upper text field shows the current status (if danger was detected or not), and the bottom field is static text.

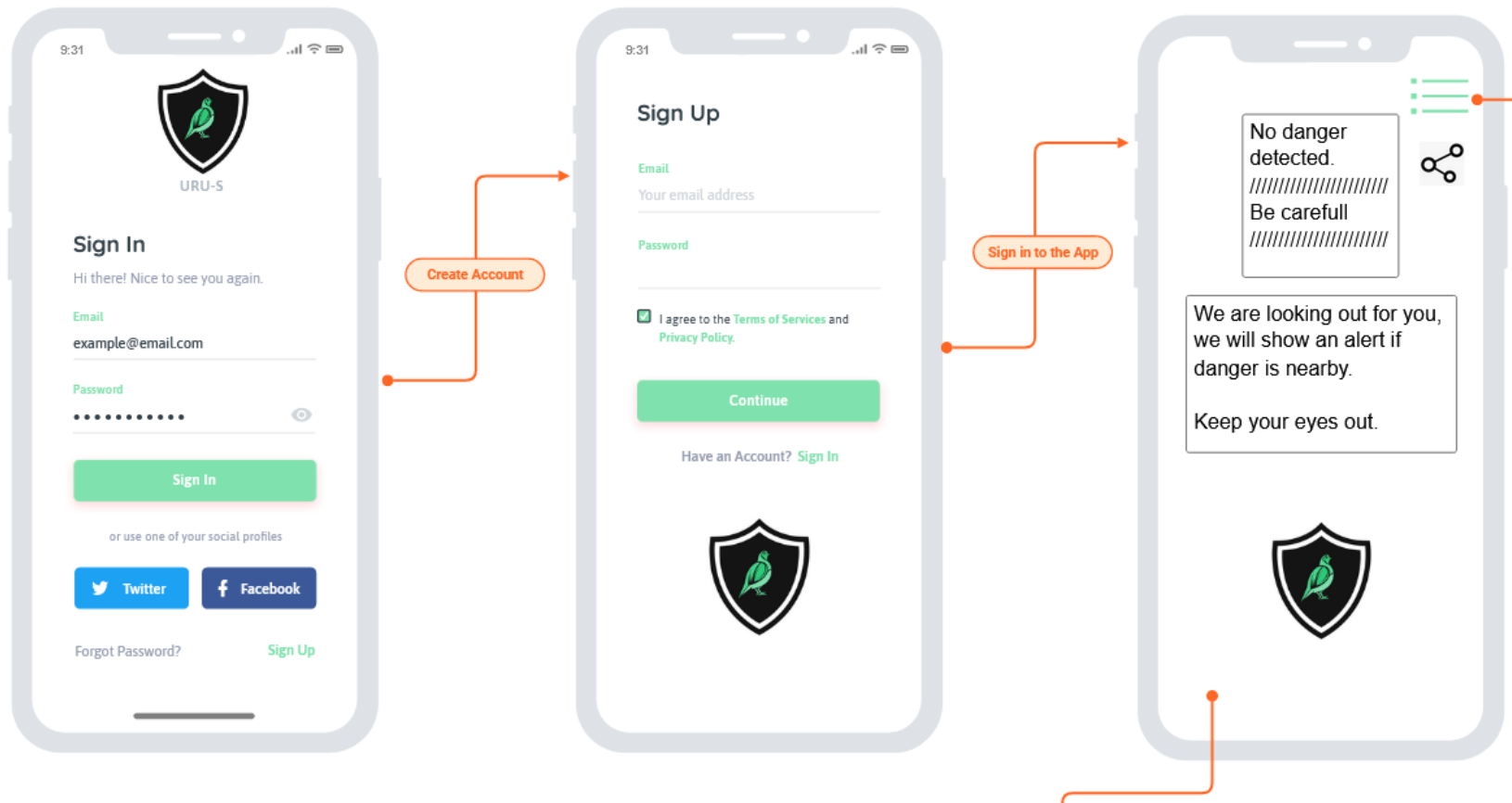


Figure 4.1: Optional Sign In, Sign Up, and Home

If the user clicks on the menu, they are directed to figure 4.2, where they may consult some additional documentation (static textual pages), swap language, or access the internal settings.



Figure 4.2: Options

Upon clicking on the "URU Settings" button, they are directed to figure 4.3, where they may change the notification settings or suspend them entirely. If they wish to suspend them, they are met with the popup that we may see on the right of the image; this popup will ask them if they are sure and specify the time to suspend the application.

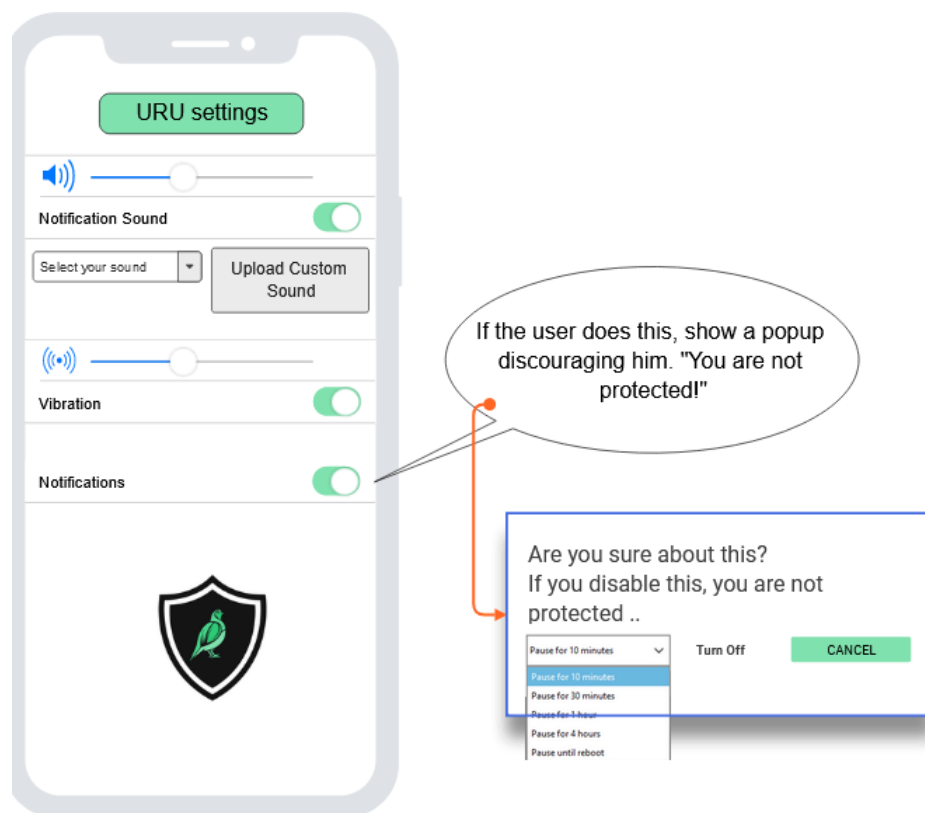


Figure 4.3: Settings

Lastly, whenever a collision happens, a push notification will be generated, as shown in figure 4.4. The user may ignore (in case of a false positive) or use it to better position themselves.

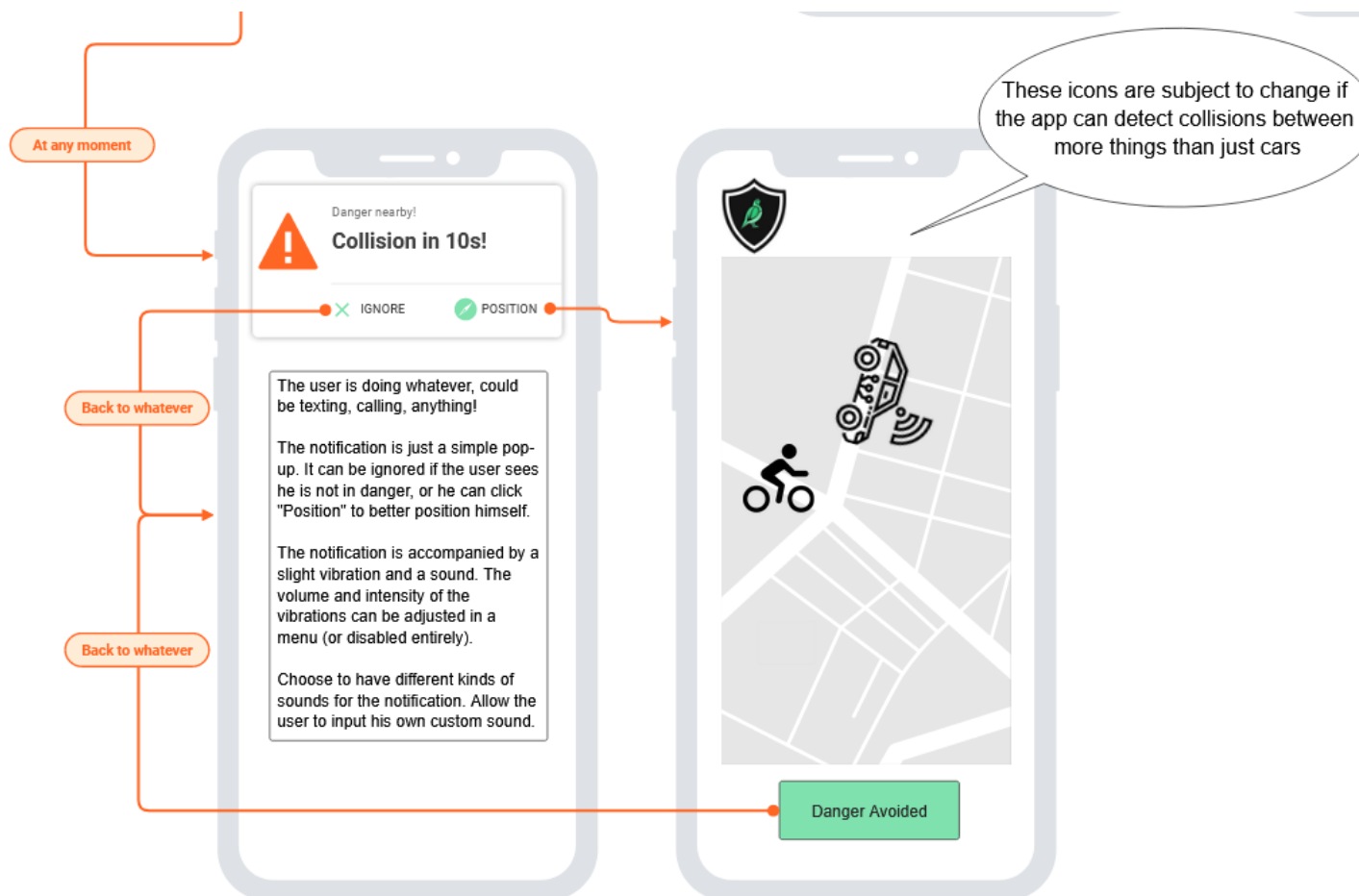


Figure 4.4: Collision

4.1.2 Focus Group

The focus group was instrumental in obtaining feedback from our participants. We found that participants mainly gravitated to a mix between a haptic (tactile) and acoustic (sound) notification. All participants showed interest in some form of feature that would warn them of potentially dangerous areas beforehand. The whole discussion between real-time and non-real-time notifications sparked great interest. Nevertheless, the wireframe successfully illustrated the general behavior of the interface and warranted some suggestions from those present.

The group lasted for about an hour, and the researcher managed to ask all of the previously planned questions. Despite the small sample size, the group raised interesting concerns and possible features. The concerns include not hearing or feeling the warning when the phone is tucked away and the need to have a warning that is not strictly visual as users may not be staring at the device constantly. Additional features include portability, a heat-map of the most dangerous areas, a "danger-level" that indicates false and genuine alerts, and safe path prediction. Some recommendations were taken into account when iterating on our preliminary design, and others were discarded as they fell outside of this project's scope.

The group is described in greater detail in appendix [A](#). The appendix includes a summary of questions asked, discussions had, the demographics, the poll results, and conclusions.

4.1.3 Resulting WireFrame

The researcher altered the preliminary wireframe based on participant feedback. The wireframe was then presented to the project's supervisor in conjunction with the corresponding use cases. This design was agreed upon and later served to guide the developer during the implementation process.

To jump-start the development process in an agile manner, we did not waste additional time creating more wireframes. The wireframe from the focus group is not to be interpreted as the final appearance of the system either. During the development, we had to add certain features that were not predicted at its conception. These features will be made evident in section [4.9](#).

Regardless, the wireframe's point is to serve as a low-fidelity prototype that can orient the development and not as an end-system representation. Overall, it was a convenient asset to have, and, most importantly, it resulted from a user-centered effort.

As we can see in figure 4.5, we no longer have any sign-in or sign-up feature, as participants were not interested in that. The home screen itself is primarily the same, except for the real-time alerts switch at the center for quick access. We also have the reports screen that shows the user their danger level and a danger heat-map, allowing them to choose if they want to have a popup notification whenever they enter a dangerous zone. Lastly, the user may navigate to the options menu. What changes here is the terminology on our "alert settings" that seemed more explicit than the "URU settings" label we had previously.

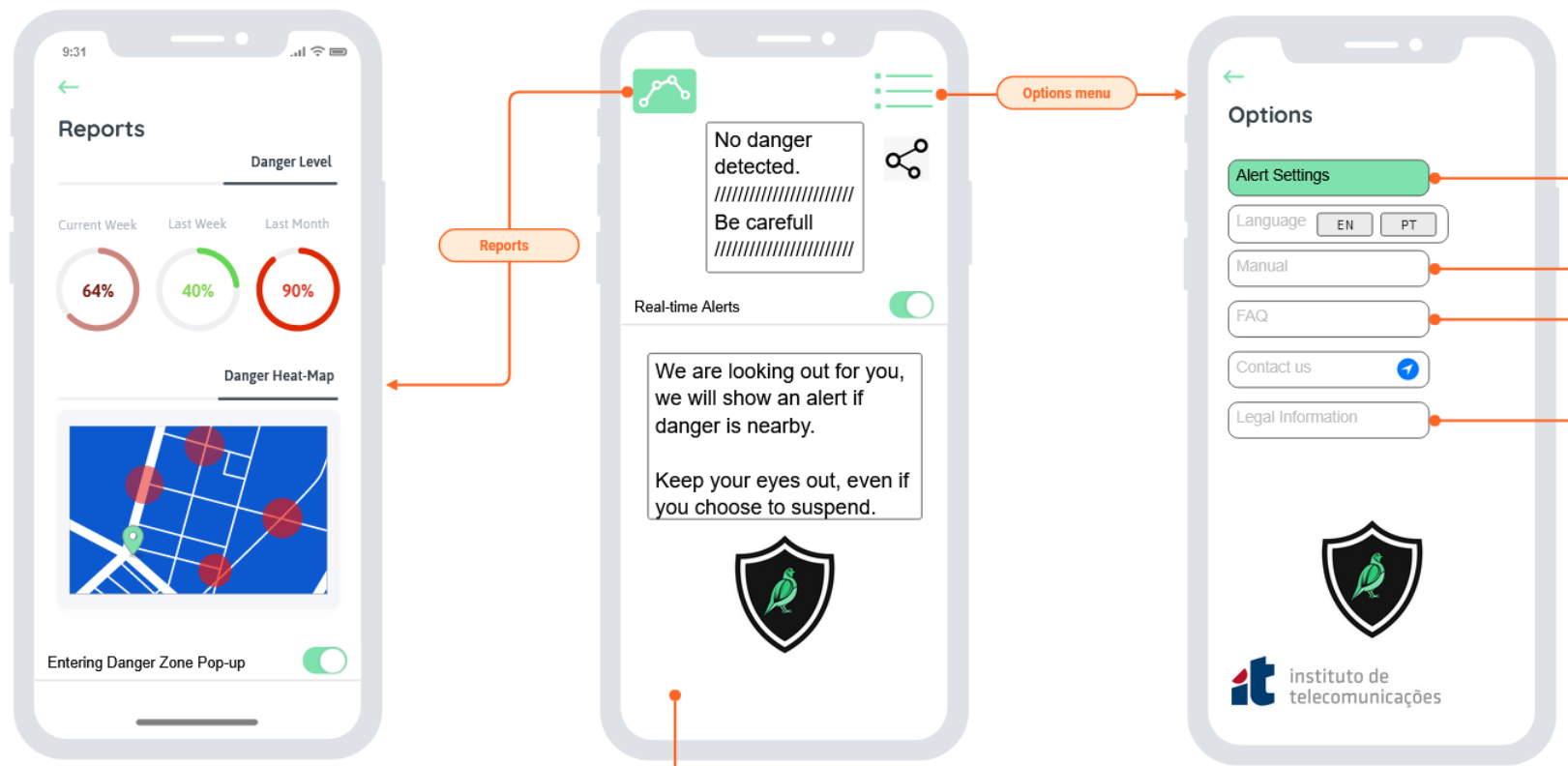


Figure 4.5: Home, Reports and Options

Continuing throughout the design, we can see from figure 4.6 that the options menu gives us access to some textual pages (manual, FAQ, legal info) and, more importantly, the alert settings. The alert settings screen is largely the same, except there is now a division between suspending the popup that appears with a warning and the real-time alerts themselves, as that caused some confusion during the focus group. We maintain the confirmation pop-up when suspending alerts, as shown in 4.7.

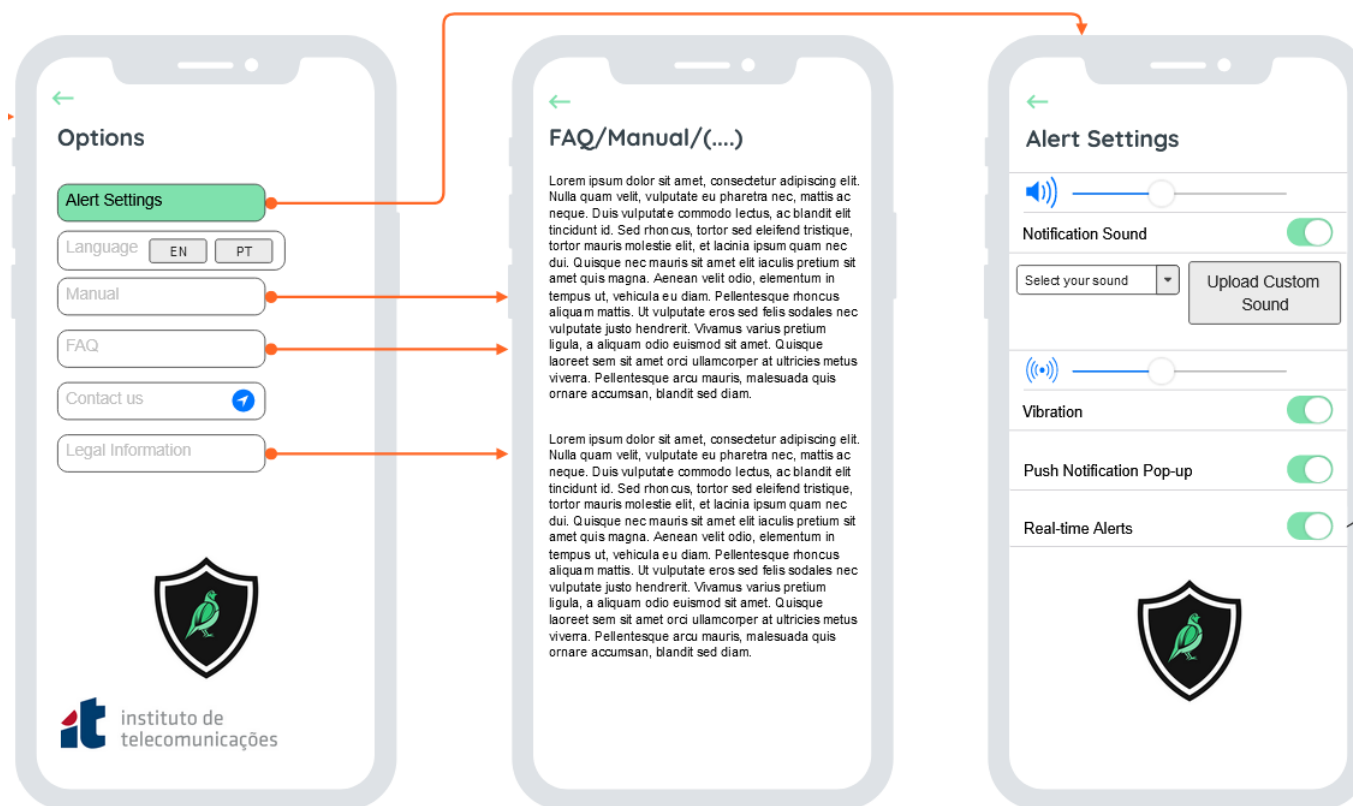


Figure 4.6: Options, FAQ, Alert Settings

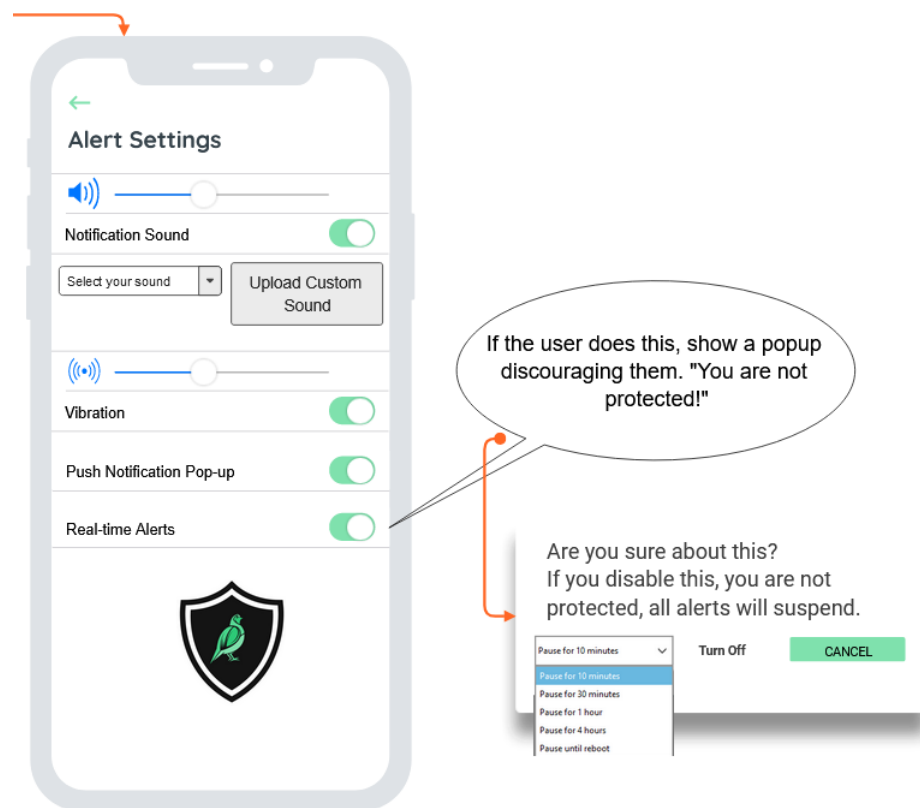


Figure 4.7: Alert Settings Complete View

Finally, figure 4.8 shows our real-time alert popup warning. It is still a push notification like before but does not open the "map" feature as participants found that useless. Instead, the notification now has two buttons that allow users to classify the dangerous event as something that helped the user avoid danger or a false alarm to ignore.

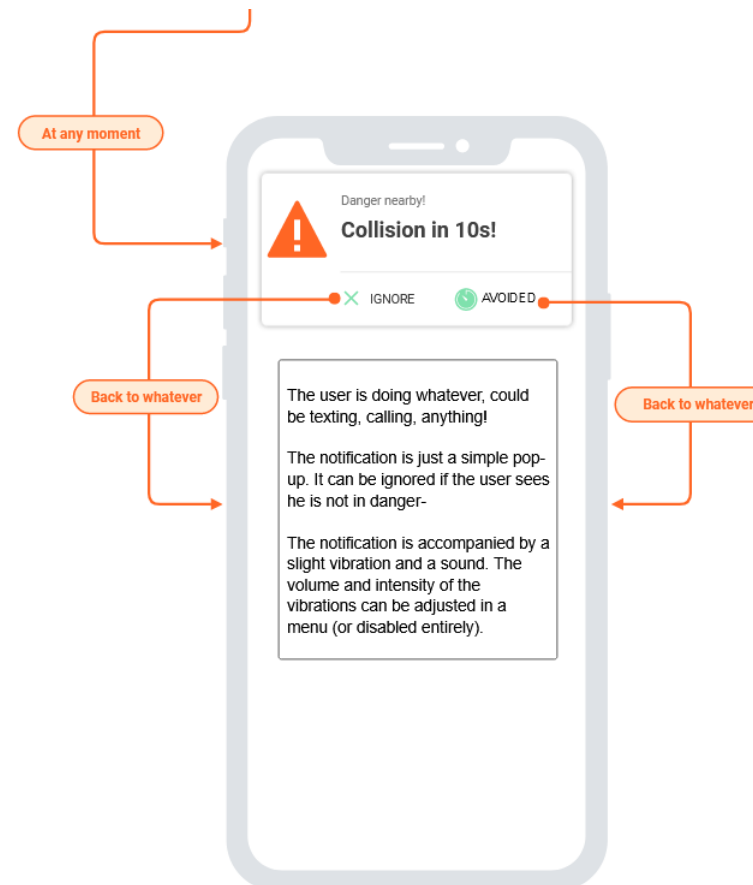


Figure 4.8: Danger warning

4.1.4 Additional Features

Certain features were idealized based on the focus group. These features include some technology that would be aware of whether the user is indoors or outdoors. One of our participants mentioned they would like to have the application suspend alerts automatically when indoors. We also need a system that can keep track of whether users have entered a dangerous zone or not to give them a forewarning.

The user's activities can be recorded using context-awareness. This mechanism can allow us to know if the user is indoors or outdoors, walking or standing still, carrying their phones in their pockets, or with headphones plugged in. We can use this information to alleviate specific concerns our users had, such as raising the alert volume when they have their phones in their pockets and lowering it when they have headphones plugged in. We can also suspend the application when the user is standing still or indoors. Moreover, we can delineate dangerous areas using geofencing, a Google service. This service can help us monitor when the user enters a dangerous place and warn them of such.

The following sections will detail how the developer managed to implement these services and how they fit into the application's overall design.

4.2 Architecture

The architecture, as represented by figure 4.9, is comprised of three main entities: the application running in the testing devices (URU-S), the external testing tool that triggers alerts remotely, and the Firebase Cloud Messaging server. It is worth noting that this is the overarching testing architecture that we used to evaluate URU-S. We thought it would make sense to include the entire architecture and not just the application module to describe how all of these elements fit together and contextualize our system. The explanations given here will also be relevant for understanding chapter 5.

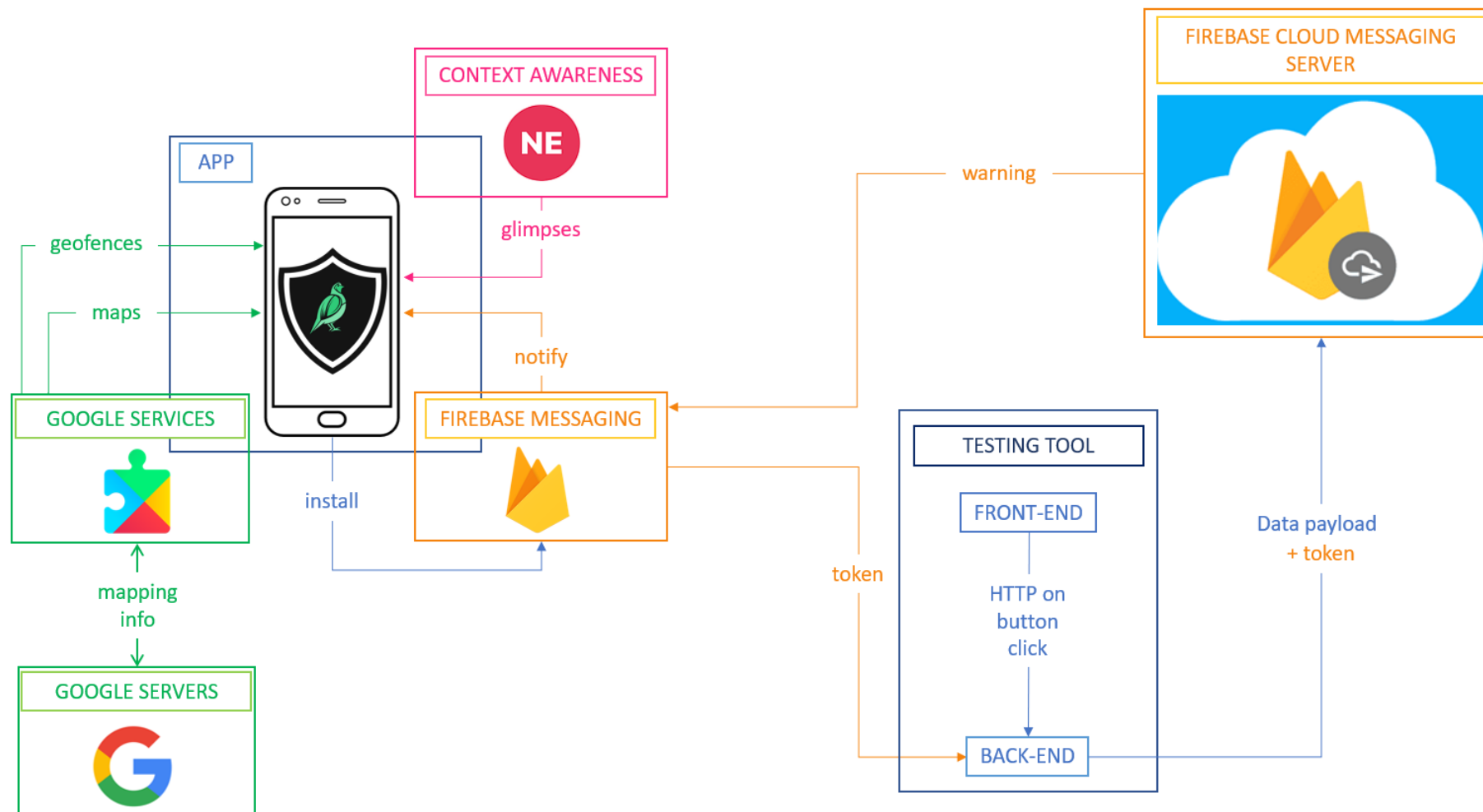


Figure 4.9: Implemented Architecture

URU-S is comprised of three main modules that provide it with crucial functionalities. These include context-awareness, Google services, and Firebase messaging. Context-awareness provides the application with contextual glimpses and works based on the NumberEight SDK. The glimpses can be concerning the user's current environment (if they are indoors or outdoors), their movement (e.g., walking, running, standing), having their phone in their pockets, and having headphones plugged into their device. Context-awareness runs as a foreground service in the application, showing as an icon in the upper left corner of the screen; this service will subscribe to NumberEight glimpses depending on which state the application is in; the way subscriptions are handled will be covered in the following section. The Google services module allows the application to make use of the geofencing API and Google maps. Geofencing is used to outline accident-prone areas, creating a 120-meter radius around a given set of latitude and longitude coordinates. Geofencing will notify the application when the user enters a geofence; similarly to context-awareness, it is also tied to a foreground service; this will be explored in greater detail later. Google maps are used to provide a visual representation of the accident-prone areas. They use the maps SDK, particularly the heat-map variant, to draw a map with heat bubbles around the most dangerous locations. Finally, the Firebase messaging module is used to receive simulated warnings.

Upon installing the application on a given testing device, a Firebase token is emitted and printed out on the console; this token is later extracted and associated with an endpoint in the testing tool. Once the testing tool has been triggered, the resulting warning is delivered to a background service that extends the "FirebaseMessagingService" class. Once the background service has received the warning, it will mobilize an additional foreground service to play sounds and vibrations and call upon a utility class to draw a push notification in the user's device.

The external testing tool is a JavaScript application that runs on localhost. In terms of its front-end, it consists of a series of buttons associated with each testing device. These buttons send HTTP requests to given endpoints when clicked, one for each device. Each activated endpoint will send a data payload with a hard-coded token belonging to the corresponding device at a back-end level. This payload will be routed through the Firebase Cloud Messaging server and be delivered to the application.

The Firebase Cloud Messaging server allows routing messages between the external testing tool and the application running on the testing devices. If the application is uninstalled, the token for the given testing device will change, and the server will not find it anymore. In case of a reinstall, the new token must be extracted manually, and the corresponding endpoint for that testing device must be altered to match the new token.

4.3 Context-Awareness and Glimpses

As mentioned previously, context-awareness relies on glimpses provided by the NumberEight SDK. These glimpses are obtained by subscribing to specific NumberEight "topics." Topics include indoor and outdoor localization, movement detection, positioning of the device, and head-

phones. By calling specific methods defined in the SDK, one may subscribe to a given topic to obtain glimpses regarding said topic. These glimpses come in the form of organized array lists of "value pairs." In NumberEight's terms, a value pair is an object comprised of a NumberEight "type" and a probabilistic confidence value. NumberEight types could be activities, device positions, localization, connections, and more (in our case, only those mentioned earlier are relevant). The array list is ordered by the confidence associated with each value pair. The most probable value pair is obtained using a method from the SDK and accessed to determine the state of a given topic. For example, to detect if the user is moving or standing still, one must first subscribe to the movement detection topic. Upon subscribing, the SDK will asynchronously report glimpses to a specified handler; once that glimpse is received, one must obtain the value pair with the highest confidence. With that value pair, one may obtain the type associated with that pair. Once we have the type, we can then access the state of said type. For example, for the "NEActivity" type, we may obtain the following states: "stationary", "walking", "running", "cycling", "in-vehicle", and "unknown". If it is likelier that the state is "stationary," that means the user is standing still. If the likelier state is anything else besides "in-vehicle," that means the user is likely moving around (we assume that the "unknown" state means a foreign type of movement).

The entity responsible for managing all of what was previously described is the "ContextAwarenessForegroundService." As the name suggests, it is a foreground service that, on start, will launch a scheduled thread executor at a fixed rate of 30 seconds that will alter shared preferences of the application and start and stop subscriptions to specific topics depending on the state of URU-S. This service is tied to context awareness and will only be active when context awareness is enabled in the application. It is a foreground service because background services do not have all of the necessary permissions to handle the NumberEight subscriptions and prevent Android from killing the service prematurely.

Context-awareness is enabled in the application by default, which means the corresponding foreground service is also running. The service subscribes to the indoor and outdoor localization topic, determining if the user is indoors or outdoors. Should the user be indoors, it disables geofencing, the alert system (covered in greater detail in another section), and all other subscriptions. Should the user be outdoors, it enables geofencing, the alert system, and starts the movement detection and the headphone detection. In terms of movement detection, if the user moves, the previously mentioned services keep working or are re-enabled if they recently restarted their motion. Should the user be standing still, then the headphone detection, device positioning detection, and geofencing are stopped until the user moves once more. In terms of headphone detection, the user will either have headphones plugged in or not. If the user has headphones on, the notification volume will be adjusted, making it slightly lower than usual since the sound source is closer to the ears. If no headphones are detected, we stop looking for them and start looking for the device's position. Regarding the device position detection, we are concerned with whether the device is covered by something, either a bag or a pocket. Should the device be in a pocket, the alert's sound will be made louder, so it is still audible even with the device tucked away elsewhere. Should the device be resting in the user's hand or any other detectable position, the default alert volume

is used. This behavior of shutting mechanisms off and on can best be summarized by table 4.1, which shows the mechanisms that are enabled (Y) or disabled (N) for each state. In that table, we use ciphers to divide the different outdoor states. Cipher "ST" represents a stationary user, "MW" is for a moving user without headphones plugged in, and "MH" for a moving user with headphones on.

	Geofencing	Movement	Device position	Localization	Headphones
INDOOR	N	N	N	Y	N
OUTDOOR (ST)	N	Y	N	Y	N
OUTDOOR (MW)	Y	Y	Y	Y	Y
OUTDOOR (MH)	Y	Y	N	Y	Y

Table 4.1: Active Mechanisms by State

It is to note that subscriptions may become idle, hence why it is crucial to have a scheduled thread resubscribe them so the application does not go too long without receiving glimpses. Additionally, these glimpses are susceptible to false positives, as they rely on imprecise smartphone sensor data. Furthermore, so that URU-S is not overloaded with glimpses constantly, there is a filtering system. NumberEight allows us to filter glimpses in terms of sensitivity and changes in confidence levels. The localization glimpses have meaningful filtering, only glimpsing after being inside or outside after approximately 30 seconds and when the most probable value pair changes. The movement detection has mild filtering, glimpsing faster than the localization services but still not quite in real-time, and only when changes occur to the value pairs. The remaining glimpses have real-time sensitivity and only occur with changes in value pairs.

4.4 Geofencing Foreground Service

Geofencing allows URU-S to delineate circular perimeters around a predefined set of coordinates (latitude and longitude) that mark accident-prone zones. We will explain how we obtained these zones in the next section.

Regardless, when started, the geofencing foreground service will parse our raw data into geofence objects and set up each geofence, associating a pending intent to them. This pending intent will be triggered when the user enters a geofence. The entry event is relayed to a broadcast receiver in the application that catches geofencing events. Should the event be an entry event, a push notification will be generated, including a message with the identifiers for each triggered geofence. Multiple geofences can be triggered if the user wanders into an area that results in the intersection of the two (or more) circular perimeters. If multiple geofences are triggered simultaneously, only one message will be sent with an enumeration of those geofences, avoiding spam.

Lastly, it is worth mentioning that the service is tied to context-awareness. These two mechanisms being associated means that the application will only register geofences when the user is

outdoors and moving. In all other states, the service will be stopped. Stopping the geofencing service will immediately destroy all registered geofences, whereas starting it once more will re-create them.

4.5 Geofencing Data Acquisition

The danger zones are obtained from a JSON file, representing an array of objects where each object has an "id," "lat," and "lng" attribute. The "id" attribute is a textual identifier for the area, recognizable by users (e.g., "Amial," "Bolhão"), whereas "lat" and "long" represent the latitude and longitude of the circle's center point. This JSON file is used as a raw, hard-coded resource of the application, and it is transcribed in figure 4.10.

```
[
  {"id": "Areinho", "lat": 41.144514, "lng": -8.578401 },
  {"id": "Varandas", "lat": 41.143515, "lng": -8.585638 },
  {"id": "Amial", "lat": 41.178638, "lng": -8.614011 },
  {"id": "VCI", "lat": 41.174949, "lng": -8.613545 },
  {"id": "Arca d'Água", "lat": 41.172541, "lng": -8.612708 },
  {"id": "Covelo", "lat": 41.165406, "lng": -8.606347 },
  {"id": "Faria Guimarães", "lat": 41.162178, "lng": -8.606905 },
  {"id": "Marquês", "lat": 41.160610, "lng": -8.605202 },
  {"id": "Hotel Grande Rio", "lat": 41.158405, "lng": -8.606583 },
  {"id": "Bom Jardim", "lat": 41.155453, "lng": -8.607661 },
  {"id": "Jornal de Notícias", "lat": 41.153561, "lng": -8.608691 },
  {"id": "Europcar", "lat": 41.152981, "lng": -8.605160 },
  {"id": "Bolhão", "lat": 41.150180, "lng": -8.605837 },
  {"id": "Sá da Bandeira", "lat": 41.148791, "lng": -8.607826 },
  {"id": "Santa Catarina", "lat": 41.146959, "lng": -8.606821 },
  {"id": "Combatentes", "lat": 41.164722, "lng": -8.598939 },
  {"id": "Lima", "lat": 41.162290, "lng": -8.599897 },
  {"id": "Alegria", "lat": 41.159417, "lng": -8.601097 },
  {"id": "Escola Normal", "lat": 41.156698, "lng": -8.602166 },
  {"id": "Antero de Quental", "lat": 41.162584, "lng": -8.610029 },
  {"id": "Piscina Constituição", "lat": 41.163018, "lng": -8.613504 },
  {"id": "Escola Constituição", "lat": 41.163420, "lng": -8.616580 },
  {"id": "Serpa Pinto", "lat": 41.163922, "lng": -8.620672 },
  {"id": "Unilabs", "lat": 41.164294, "lng": -8.623747 },
  {"id": "Areosa Circunvalação", "lat": 41.180435, "lng": -8.581933 },
  {"id": "Rodrigues Semide", "lat": 41.177806, "lng": -8.582641 },
  {"id": "Sta. Justa", "lat": 41.174376, "lng": -8.584596 },
  {"id": "Br. Costa Cabral", "lat": 41.171494, "lng": -8.586023 },
  {"id": "Contumil", "lat": 41.169042, "lng": -8.587271 },
  {"id": "Junta de Massarelos", "lat": 41.152568, "lng": -8.630137 },
  {"id": "Gólgota", "lat": 41.152454, "lng": -8.634474 },
  {"id": "Planetário", "lat": 41.153312, "lng": -8.639376 },
  {"id": "Gomes Costa", "lat": 41.162335, "lng": -8.655794 },
  {"id": "Foco", "lat": 41.160954, "lng": -8.647560 },
  {"id": "Bessa", "lat": 41.160069, "lng": -8.641733 },
  {"id": "Eupoupo", "lat": 41.159237, "lng": -8.636993 },
  {"id": "Agramonte", "lat": 41.158609, "lng": -8.633235 },
  {"id": "Praça Mouzinho de Albuquerque", "lat": 41.157914, "lng": -8.629128 },
  {"id": "Hospital Militar", "lat": 41.159541, "lng": -8.625488 },
  {"id": "Ferrari Porto", "lat": 41.157080, "lng": -8.623860 },
  {"id": "RAMP", "lat": 41.179027, "lng": -8.595491 }
]
```

Figure 4.10: Danger Zones JSON

It seems that, at least in Portugal, accident data comes mainly from the police, specifically, *Polícia de Segurança Pública* (PSP). This data is then supplied monthly to other entities, particularly *Direção Municipal de Mobilidade e Transportes* (DMMT) in .xls format [56]. This data is usually under strict privacy policies and, as such, was not made available for the development of the URU-S prototype. Ideally, we would have access to such a data-set or something similar, like the one used in the study conducted by Vilaça et al. [57]. The study, as mentioned previously, works with a comprehensive data-set of road accidents involving VRU to assess the factors that may affect the severity of their injuries. The study uses several machine learning classifiers to determine the factors that may influence these injuries and make recommendations for improvements in those areas based on their findings. However, the so-called "CRASH VRU RECORDS" that stand at the basis of these deductions are not publicly available either.

As we have established, accident data is not easy to come by and involves adamant bureaucracy. Even if we could obtain it, there is nothing to suggest that it would be in a format that could easily be parsed into a JSON that URU-S could interpret. Thus, to collect our data, we consulted civil engineering dissertations [56][58] and their findings to extrapolate our points. In the previously mentioned dissertations, the authors usually outline entire avenues or streets, never specifying a single set of coordinates. Since most accidents happen at intersections, we explored those areas with Google Maps, extracting the latitude and longitude of intersections with reduced visibility and many connecting roads. The previously mentioned process is under the presumption that if the given avenue, street, or area is particularly prone to accidents, then those intersections must be the danger hot-spots. This presumption may not be the ideal method to draw the geofences and outline these areas, but given the resources available, it was the best that could be accomplished.

4.6 Geofence Accuracy

Our geofencing foreground service requires the user's location to function appropriately. As such, it will periodically request location updates to the device while it is active. It does this by using a `FusedLocationProviderClient` [59] and calling its "requestLocationUpdates" method, using a certain location request. This location request is expressed in figure 4.11

As we can see, it is a high accuracy request, which means the location provider will do its best to obtain the most accurate position possible, fusing different sensors if need be. The high accuracy request drains more battery than a less accurate request; however, since geofencing is tied to context-awareness, it will only be active while the user is outdoors and in motion. This project is also under the presumption that most users do not spend the entirety of their day outdoors and in motion but rather a small portion of it. Despite this, we do not have the required infrastructure to measure geofencing's energy drain accurately; that could remain a factor to be looked at in the future.

Geofences also have certain limitations. Accurate location may not be available if the geofence has a radius of fewer than 100 meters, as most devices end up relying on network location for

```
/*  
 * Location update time interval, considering high accuracy  
 */  
long PRECISION_UPDATE_INTERVAL = 10000;  
  
/*  
 * Location update time fastest interval, considering high accuracy  
 */  
long PRECISION_UPDATE_FASTEST_INTERVAL = 2000;  
  
max_precision = LocationRequest.create()  
    .setInterval(PRECISION_UPDATE_INTERVAL)  
    .setFastestInterval(PRECISION_UPDATE_FASTEST_INTERVAL)  
    .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)  
    .setMaxWaitTime(PRECISION_UPDATE_INTERVAL*2);
```

Figure 4.11: Location Request High Accuracy

geofencing to work correctly. The official documentation [49] suggests using a radius of about 100 to 150 meters to mitigate this issue. In this work, a radius of 120 meters is used to allow for greater accuracy. Additionally, if Wi-Fi is turned off, location accuracy may diminish. Furthermore, having no reliable network connectivity inside of a geofence may cause alerts to be missed. Alerts can also have a given latency that can go up to 2 minutes. There have been studies conducted to develop more accurate location queries that mitigate the limitations of geofences, using less power and achieving higher precision [60]. The study mentioned earlier does not seem to have an open-source solution. It is from 5 years ago; it is also worth mentioning that the geofences used had a radius of 25 meters, which is one-fourth of the recommended value. The small radii may explain why Weaver et al. had such inconsistent findings with geofences. In their article, they experience varying precision, latency, and energy drains. Their solution seems to have energy use similar to geofencing but allegedly has better accuracy for smaller radii.

4.7 Alert System

The URU-S alert system is a crucial mechanism that controls whether or not the user receives danger notifications. This service can be integrated with context-awareness or be used manually. The interactions with context-awareness can be modeled by a state machine like shown in figure 4.12.

In the state machine, acceptance states represent states where the alerts are active and being received. Non-acceptance states do not have alerts enabled. The meaning behind the labels of each state and transition is as follows:

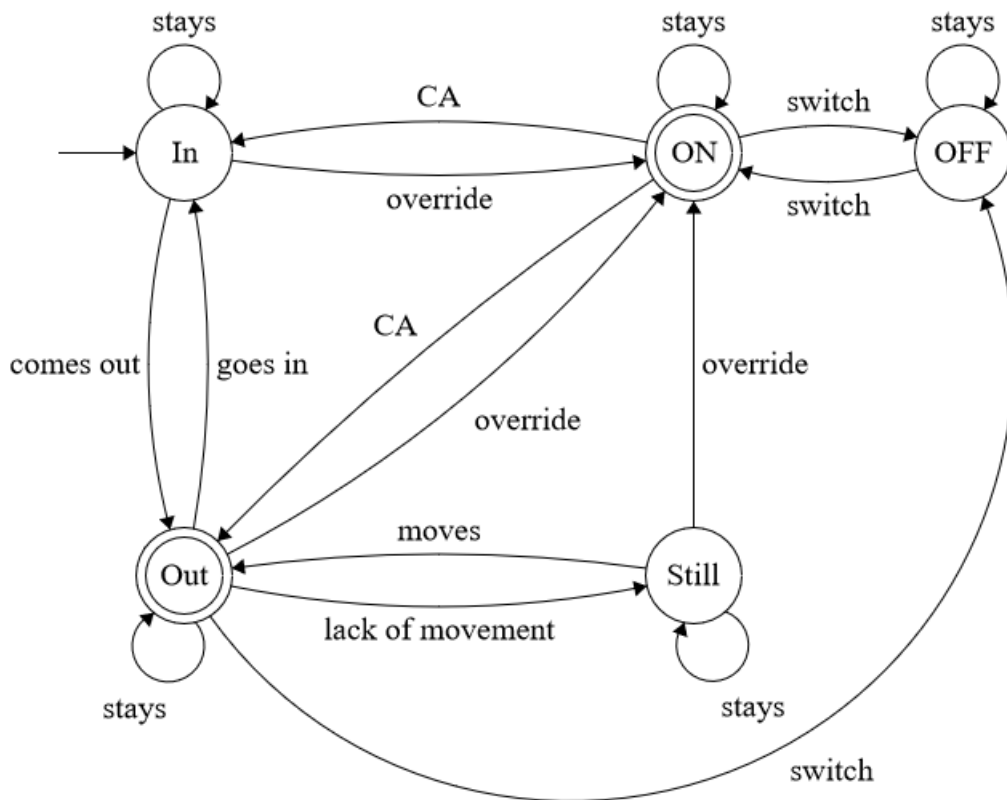


Figure 4.12: Alert System State Transitions

- **In:** represents the user is indoors, which is detected by the respective NumberEight SDK glimpse.
- **Out:** represents the user being outdoors, which is detected by the respective NumberEight SDK glimpse. The application is receptive to alerts, but they may be disabled if the user's context changes.
- **Still:** represents the user is still, which is detected by the respective NumberEight SDK glimpse.
- **ON:** represents the application being in manual mode and receptive to alerts.

- **OFF**: represents the application being in manual mode and non-receptive to alerts. It may become receptive again after a given time or stay that way until the user manually re-enables (this is specified via the interface).
- **stays**: represents a transition used to signify that the user may remain in this state for an undetermined period until they willingly decide to alter their situation or have set a timer in the application to do so.
- **comes out**: represents the user stepping out of their indoor place and moving around outdoors. The transition does not happen instantly as the state machine suggests; it may take approximately 30 seconds or more to occur in reality.
- **goes in**: represents the user stepping into their indoor place, coming from the outside. The transition does not happen instantly and may take up to 30 seconds.
- **lack of movement**: represents the user standing still for a given period. The transition does not occur instantly and may take up to 30 seconds.
- **moves**: represents the user moving. The transition does not occur instantly and may take up to 30 seconds.
- **CA**: represents a change in state that is triggered by context-awareness. States may change upon receiving the right glimpses that indicate that they should. If context-awareness is disabled, these transitions do not happen, and to re-enable it, the application must first be turned **ON**.
- **override**: represents an override to the context-awareness mechanism, denying a context-awareness state change to return the application to its manual **ON** state, disabling context-awareness until manually re-enabled.
- **switch**: represents an interaction with the real-time alerts switch. The switch can turn the application on and off by flipping it manually or setting a timer using the interface. The switch prevails over context-awareness.

It is worth noting that to streamline the interactions between context-awareness and the alert system, any attempt to override context-awareness will disable it until the user manually re-enables it in the settings. When manual, the application will be receptive or non-receptive to alerts at the user's control. It will not benefit from geofencing or alert suspension during unnecessary periods (e.g., indoors, standing still).

Furthermore, since implementing an algorithm that creates safety messages that relay the user's trajectory and speed is not within this project's scope, the alert system being suspended means that we disable geofencing, sound adjustments to the notifications, and the notifications themselves. In a future implementation, with collision and trajectory detection modules, those kinematic computations could be stopped in our non-acceptance states as well. By stopping the

added modules in states where they are not strictly necessary, we could optimize the phone's battery supply.

Note that "stopping the notifications themselves" does not mean stopping the Firebase Messaging service. Due to the way it is implemented and made available in the Firebase API, one cannot override the binding method for the service, and starting and stopping it may have unpredictable behavior. For the reasons explained previously, we refrained from stopping the messaging service entirely and instead made it so, when the application is not in a receptive state, it will not generate a notification. The service still receives a data payload, but it is overlooked. Fortunately, the Firebase messaging service is battery-optimized, and having it running causes negligible drain [50].

4.8 Requirement Satisfaction

It is worth mentioning that the prototype includes technical requirements ranging from TR01 to TR09, as they are stated in table 3.1, inclusively. Each requirement was considered during the development of URU-S in the ways detailed in the following paragraphs.

Starting at TR01 (Accessibility), generic accessibility good practices were applied to develop URU-S. These good practices include making every clickable view at least 48dp in length and width (some are even larger than that), giving all relevant views a content description (available in Portuguese and English), not specifying "control type" or "control state" in content descriptions (like "button" or "switch"), giving decorative views a null content description, grouping together containers using the "android:focusable" attribute, having a logical top-down and low-density view hierarchy for each activity, having a reasonable contrast ratio, making use of accessibility delegates, and testing out the application manually with TalkBack, SwitchAccess, and the Accessibility Scanner, as recommended in the Android developers official documentation [61]. The Accessibility Scanner was instrumental, as running it gave us plenty of suggestions on improving the interface. These suggestions essentially consisted of contrast recommendations and expanding specific views to meet the minimum criteria of 48dp by 48dp. Lastly, we would like to note that there were problematic interactions between the switch views in URU-S and SwitchAccess, as activating a switch with the tool did not count as "pressing" the view. To remedy this, we used accessibility delegates and associated them with relevant switches. These delegates will respond to accessibility events and set the switch to be "pressed," as shown in figure 4.13 below (this corresponds to the popups switch).

```
pop_up_switch.setAccessibilityDelegate(new View.AccessibilityDelegate(){
    @Override
    public void onInitializeAccessibilityEvent(View host, AccessibilityEvent event) {
        pop_up_switch.setPressed(true);
        super.onInitializeAccessibilityEvent(host, event);
    }
});
```

Figure 4.13: Accessibility Delegate

Moving on to requirement TR02 (Usability), general design principles, most of which overlap with those mentioned in the previous paragraph, were followed to make the interface usable. Additionally, URU-S obeys the ten basic heuristics for interface design, specified by Jakob Nielsen [33]. To fulfill "visibility of the system status," URU-S displays its status on the homepage and through foreground services that are visible in the notification tray. When it comes to "match between system and the real world," the design uses familiar words, phrases, and concepts, such as "Real-time alerts," "Danger Level," "Danger Map," and others. To guarantee "user control and freedom," we have attached confirmation dialog boxes to destructive actions that involve deleting something with the application. It is possible to undo turning components off or on using the respective switches. Furthermore, "consistency and standards" are guaranteed, such as using the garbage-can-icon to delete something, using a familiar icon for the refresh functionality, and the "hamburger button" for the menu. Other factors to consider are "error prevention" and "error recovery." There is a slight possibility of errors occurring while using the interface; should they happen, they are presented to the user. An example is when the user tries to play a custom sound for their alert notification that no longer exists in the device. The sound is removed from the list of available sounds, and the user is told that sound does not exist in their device anymore, replacing it with the default; moreover, drastic actions, such as disabling the alert system and overriding context-awareness prompt the user with confirmation dialog-boxes. Even though the iconography used in URU-S is quite familiar, to further enhance "recognition rather than recall," minimizing the user's memory load, labels were placed next to icons that may not have a very explicit meaning. The system is also somewhat flexible, reinforcing "flexibility and efficiency of use." Examples include having the real-time alerts switch available right on the main page, so users do not have to go all the way to the alert settings to use it. The application also uses negative space to provide an "aesthetic and minimalist design," containing relevant information units and never saturating the screen with views. Finally, "help and documentation" are available in the options menu, containing a detailed manual, FAQ, contact, and legal information. We must also point out that these heuristics are usually regarded as simplistic guidelines that outline the most grievous issues with interfaces. To meet this requirement, we also conducted focus groups to understand better what users wanted to see in URU-S, using WireFrames to explore different possibilities. Also, we conducted a usability testing phase, the results of which are covered in chapter 5.

Next up, we have TR03 (Security); this requirement was achieved primarily due to the prototype's limitations and testing. User-sensitive data is never, in fact, "leaked" as the application is never installed in the user's devices. URU-S is in closed internal testing phases, which means participants only ever test the application using specific testing devices at the testing facility. Since the application is never installed in a personal phone and does not require any form of authentication to be used, privacy can be guaranteed by omitting systems that may compromise one's information in the current conditions. However, an argument could be made that, since we utilize Google services, which involves a call to a Google provider, the user's position could be comprised as it is received and processed by Google in an end-version of the system. We presume that if a user has an Android phone, which uses Google technology by default, they already know this. For

the phone to be completely secure and devoid of Google's presence, they would have to resort to a completely different operating system, like Secure OS [62]. Nevertheless, this project's scope involves a regular Android system, and free reign was given to use whichever development tools necessary. In the future, other tools could be explored to remove the Google services dependency from URU-S entirely. Regardless, this will not change the fact that a regular Android system would still have Google's presence in other applications unless those are entirely disabled, which could limit the device severely.

When addressing TR04 (Ethics), we would like to clarify that these ethics are not the ones tied to strict codes of conduct, mainly since those can be very controversial and challenging to put into practice as they cannot orient engineers on particular ethical conundrums that occur every day [63, sec.4.1]; moreover, those codes are usually applied to the practice of engineering at a higher level, which is beyond the scope of URU-S as an academic project. We define TR04 as treating everyone with dignity and producing software exclusively to serve our end-users. It is also crucial to be transparent with our users by allowing them to consult our detailed privacy policy, which complies with *Regulamento Geral sobre a Proteção de Dados* (RGPD). Additionally, we never take or base code off of somewhere without attribution to that source. We do not have any form of malware or spyware adjacent to the application's source code, and we use open-source licensed tools and libraries whenever possible.

Another requirement would be TR05 (Robustness), which is guaranteed by extensively covering the application using instrumented testing tools. Despite not being the most formal (for reasons explained in chapter 3), the methodologies used are still effective at verifying the system, as they cover all significant branches of the code through relevant assertions. We will be expanding on how our verification was conducted and the results obtained in chapter 5.

Furthermore, a standard requirement would be TR06 (Context-Awareness), which is provided mainly through our integration with the NumberEight SDK. Using the context-awareness foreground service, as explained in section 4.3, we can instill state changes in the application through, for example, movement detection.

A crucial requirement would be TR07 (Battery Saving), which is accentuated by our battery-optimized services. The only instance of the code where we do not prioritize battery power is when we specify high-precision location requests. However, those requests are tied to context-awareness and only function for a reduced period of a user's day (when they are moving outdoors). Whenever the user stops, goes indoors, or disables context-awareness, the location requests will not be put into practice, and, as such, we do not view this as a violation of this requirement. We are under the presumption that our users will not walk outdoors nonstop for prolonged periods, as that is not consistent with the average urban lifestyle. Admittedly, suppose the user is in motion for hours on end without stopping once. In that case, our reliance on high-precision location requests will start to drain a more significant amount of battery. Drain under continuous outdoor movement is an inherent limitation that we cannot work around without lowering the precision of the locations that we obtain or increasing the interval for obtaining said locations. Lowering the precision or increasing the interval for requests may compromise the inner workings of the geofencing service

and make it unusable, as it is less likely to notify users helpfully. Less granular location requests would also increase the inaccuracy in positioning our users relative to dangerous zones, hurting the application's credibility. In the end, if we are to presume we cannot alleviate battery drain by suspending requests (i.e., the user is constantly moving outdoors and thus needs permanent protection), we are met with an inescapable trade-off between precision and energy. After consulting with the project supervisor, the orientation received indicated a necessity to prioritize accuracy over energy due to the above presumption.

Next, we would discuss TR08 (Presence), which is evident throughout the entirety of URU-S. Whenever we want to accomplish a long-running task with an application that does not require user input, such as geofencing and context-awareness, foreground services are used. There are two reasons these services are of the foreground kind instead of the background kind. The first reason is that they use sensors and location permissions that are restricted in the background. The second reason is that Android may kill these services prematurely if they are declared as simple background services; in our implementation, these services return "START_STICKY," which, according to the documentation, is the appropriate mode for services that are "explicitly started and stopped to run for arbitrary periods of time" [64].

Finally, we have TR09 (Storage), which is the most straightforward requirement to fulfill. We use an "SQLiteOpenHelper" class to construct a simple internal database within the device. This database will store alerts received as well as custom sounds that the user uploads. The reasoning behind this database will be made more explicit when we look at the application's practical functionality in the next section.

4.9 Final Screens, Flow and Use Cases

The various screens of the application are a result of user-centered design. URU-S has gone through a prototyping phase that consisted of WireFrames. The first WireFrame was presented at a Focus Group and iterated upon given the feedback from the present participants. That WireFrame guided the development of the prototype in Android code. The prototype was then subjected to a usability testing phase that caused the developer to alter the interface. The final results are shown in this section.

4.9.1 Home Screen

This portion of URU-S aims to satisfy the following use cases: US01 (Protection Status), US03 (Notification Suspension), US04 (Receive Notifications), US07 (Options menu), and US10 (Access Statistics), US15 (Sharing).

As we can see from figure 4.15, it is possible to consult the status of the application by looking at the banner at the top of the page, which could be in different states. The "real-time" alerts switch allows a user to regulate whether they are receiving notifications or not. One may navigate to the "Options" menu or the "Statistics" section, accessing the application's settings and documentation or the danger statistics, respectively. Lastly, one may also share the app by clicking on the "Share" image-button.



Figure 4.14: Other states of URU-S

The states the application can be in are "INDOOR," "STILL," "NORMAL," and "OFF." An illustration of the other states besides "NORMAL" can be seen in figure 4.14. The first two are only detected with context awareness enabled, whereas the last two can be enabled or disabled manually. The application will always show the current state that

it is in through this screen, having different colors and descriptions.

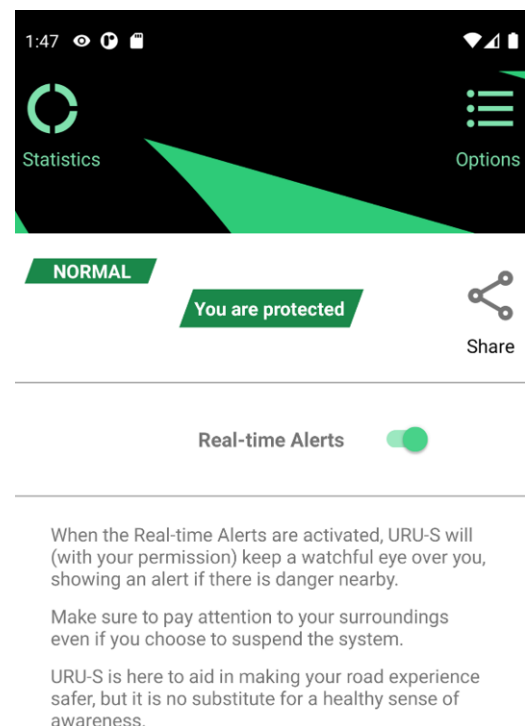


Figure 4.15: Home Screen Illustration

4.9.2 Options Menu

The options menu aims to satisfy the following use cases: US06 (Options menu), US07 (Language), and US16 (Documents).

As we can see from figure 4.16, this menu allows the user to go into their alert settings, swap the language between English (EN) and Portuguese (PT), access the application manual frequently asked questions, contact, and legal information. The user can also navigate to the previous screen by using the arrow in the upper left corner.

In terms of languages, only English and Portuguese are available. Support for other languages may be added in the future. Additionally, it is worth mentioning that translating the interface will also translate every notification the application receives, and even the channel names and descriptions. Our "frequently asked questions" were extrapolated from remarks made by participants in the focus group and usability tests. Legal information consists of the privacy policy participants had to consent to before each event required their participation. The manual explains the interface and all of its elements in great detail. Ideally, users will not need the manual,

but it is at their disposal if need be.

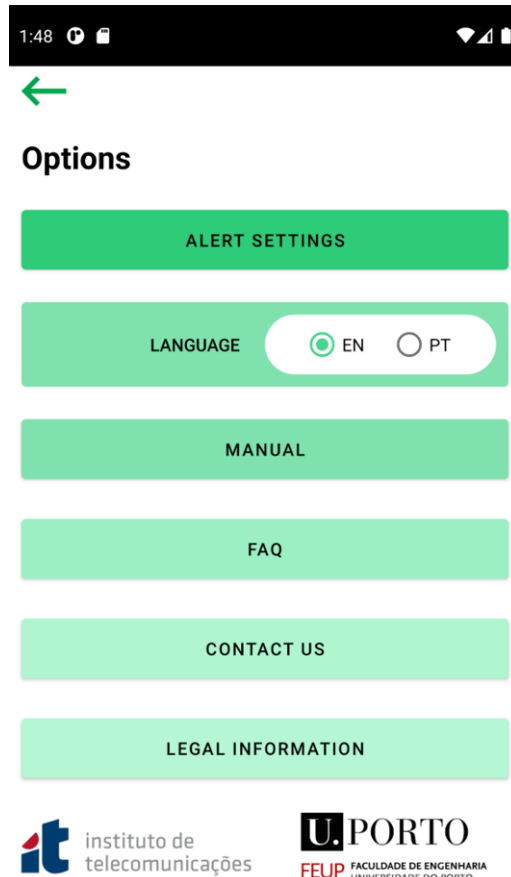


Figure 4.16: Options Menu Illustration

4.9.3 Alert Settings

This section aims to satisfy the following use cases: US02 (Notification Volume), US03 (Notification Suspension), US04 (Receive Notifications), US05 (Notification Vibration), US08 (Select Sound), US09 (Customize Sound).

Given the research presented in the State of the Art section 2.4.6, the most effective method of warning the VRU would be tactile. Haptic feedback can be achieved by emitting vibrations on the phone [65]. Acoustic signals are also possible to accomplish by leveraging the phone’s speakers. This interface was developed to allow the user to configure these warnings.

As we can see from figure 4.17, this screen allows the user to configure their real-time alert settings. Users may adjust the volume of their notification, select a sound by choosing from a list of preset sounds, or even upload their custom sound. Users may also adjust the vibration until it is to their liking. Upon changing a sound or vibration setting, a preview of the new sound or vibration will play. It is also possible to remove or re-enable the notification popup that accompanies the sounds and vibrations.

Besides configuring acoustics, haptics, and visuals of one’s alert, the user may also use this screen to suspend or manually enable real-time alerts, much like in the home screen. Moreover, the user may suspend or re-enable the alert regulation mode. In doing so,

the application will kill or engage the context-awareness foreground service, respectively.

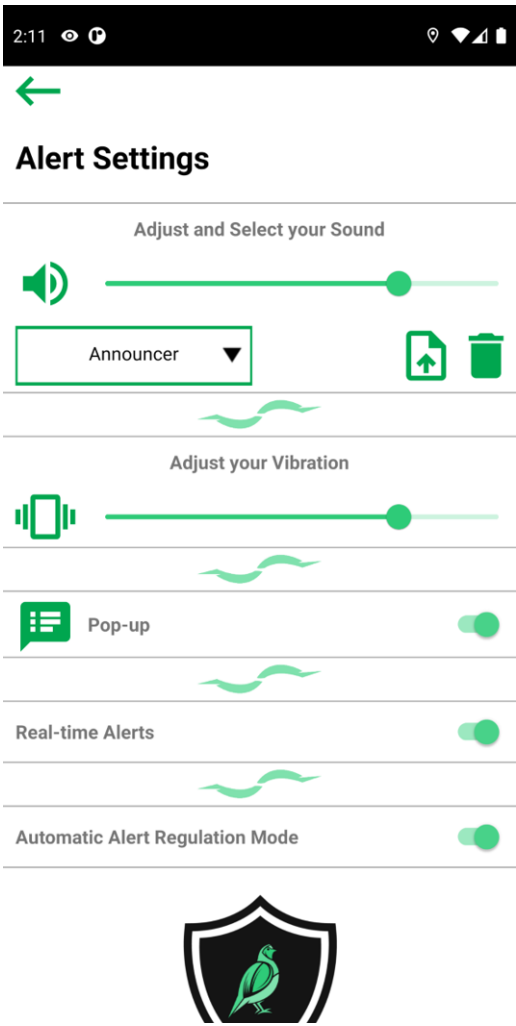


Figure 4.17: Alert Settings Illustration

4.9.4 Statistics

This section aims to satisfy the following use cases: US10 (Access Statistics), US11 (Access Heat-map), US12 (Receive Danger-Zone Notifications), US13 (Disable Danger-Zone Notifications).

As we can see from figure 4.18, this screen allows the user to consult their "Danger Level" statistics, mainly how many alerts they have received for a given date range (which could be today, this week, this month, this year, or of all time). Users can also see which percentage of their alerts were false alarms or genuine alerts. The application distinguishes between a false alarm or an alert that successfully prevented danger via user input. Suppose a user is struggling to figure out what these terms mean. In that case, they may click on the question mark in the center to access a textual activity that explains the meaning of the graphs and how they are obtained. Essentially, the "false alarm" is when a warning does not help the user avoid danger, and the "genuine" alarm helped avoid a collision. The user labels events using the notification that we will be looking at in the next section.

It is also possible to consult a "Danger Map" that displays a heat-map of the most dangerous areas. This heat-map can be adjusted to the user's liking, similar to any Google Map. Users may zoom in and out of the map, use the "my location" feature to situate themselves better relative to the dangerous zones, or even use the icon with the green arrows pointing outward to make the map full-

screen, as represented in the figure. Additionally, users can set whether or not they want a popup to appear, warning them once they have entered a dangerous zone.

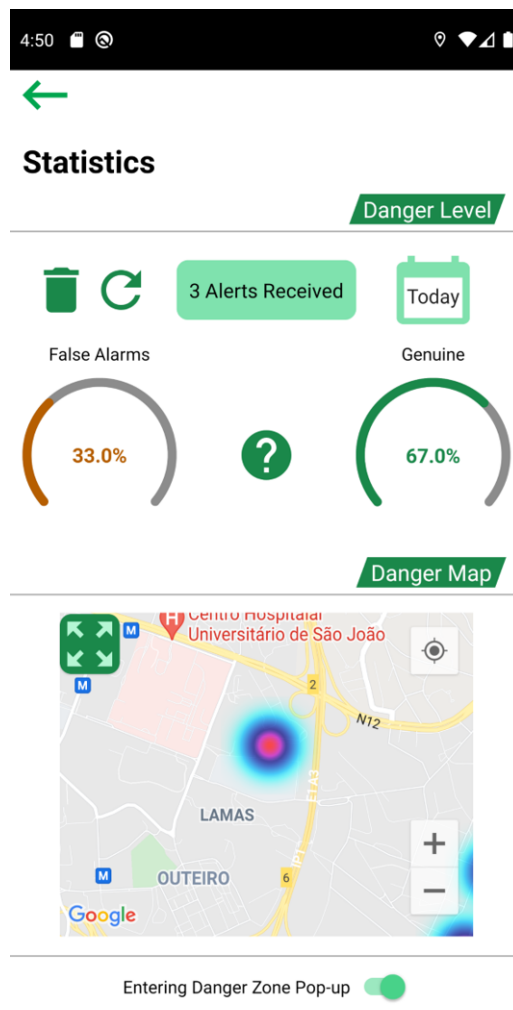


Figure 4.18: Statistics Illustration

4.9.5 Real-Time Alert Notification

The real-time alert notification is the one that appears when the user is in imminent danger, and it serves to satisfy the following use cases: US14 (Neutralize Notification), US17 (Ignore Notifications).

As we can see from figure 4.19, this is a simple push notification with two action buttons attached to it. These action buttons feed into the application's statistics directly, allowing the user to classify each detected danger event as either a false alarm or something that genuinely helped them avoid danger. The purpose of the notification is to serve as a visual indicator and integrate with the "Statistics" section, but not to serve as the determining visual queue that would get the user's attention. We presume that the sound and vibration URU-S emits would be more immediate and effective ways of notifying our users. The visual element, as represented by the figure mentioned above, is just an add-on to the overall user experience.

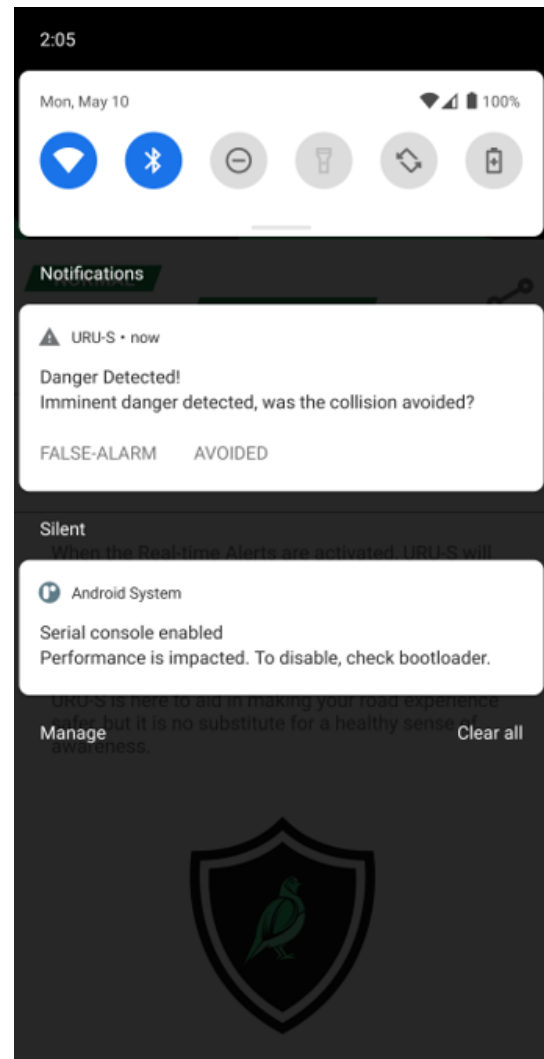


Figure 4.19: Real-Time Alert Notification Illustration

4.9.6 Context-Awareness Notification

The context-awareness notification is the one that appears when the "automatic alert regulation mode" option is checked in the alert settings screen.

As we can see from figure 4.20, the notification tells the user that the device is regulating alerts automatically but does not blare out the term "context-awareness" right away, as that could confuse people. Instead, if the user interacts with the notification by clicking on it, they will be brought to an explanation screen that tells them what context-awareness is and what it does for them.

The explanation screen should elucidate the user by reassuring them that context-awareness monitors their whereabouts, activity, device position, and headphone state. The user is informed that localization and activity will suspend alerts whenever they are not needed, and device position and headphone state will regulate the notification's volume accordingly.

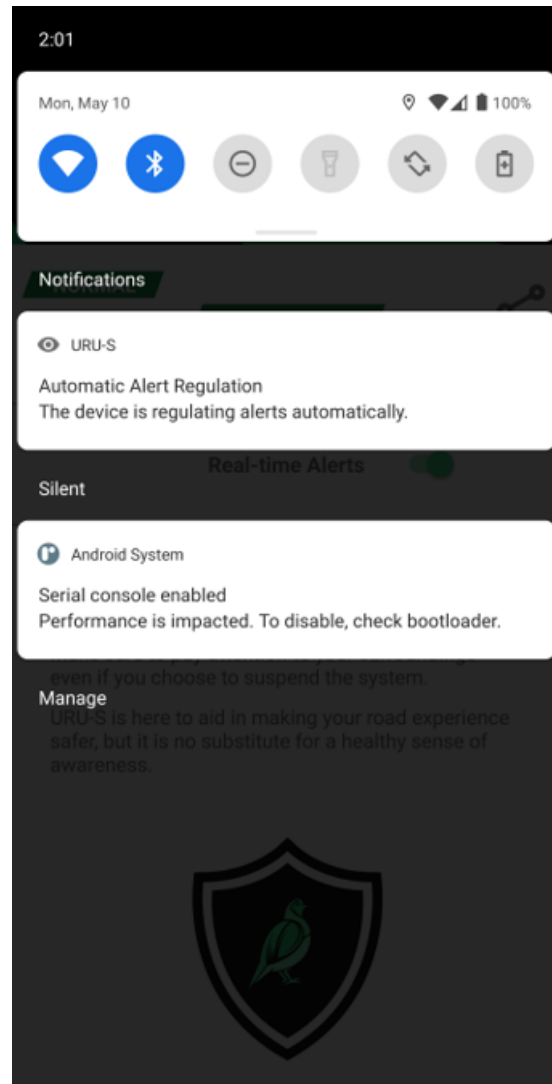


Figure 4.20: Context-Awareness Notification Illustration

4.9.7 Context-Awareness Override

At any point, users may choose to override context-awareness. This override is done by manually engaging the alert system through one of the real-time alerts switches. Users can do this either in the alert settings or home screens. If they choose to override context-awareness, they will be prompted with a confirmation dialog, as seen in figure 4.21. Upon clicking "YES," URU-S will kill the context-awareness service entirely. In manual mode, the application is receptive to alerts unless the user manually disables them by flipping the alert switch to "OFF."

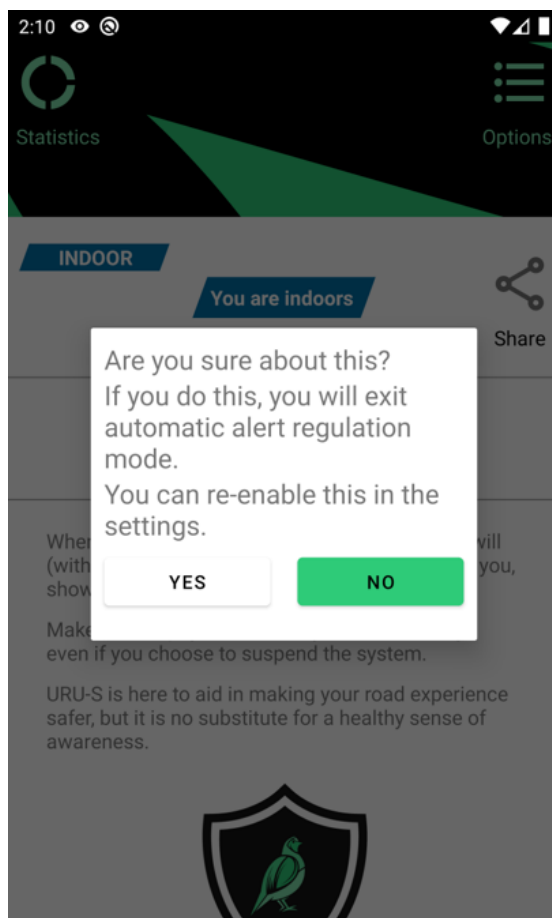


Figure 4.21: Context-Awareness Override Illustration

4.9.8 Alert Suspension

If the application is in its "NORMAL" state, whether because it has detected the user is walking outdoors through context-awareness or manually set to receive alerts, the user may suspend the alert system; we may see an illustration of this in figure 4.22. Alert suspension is a fail-safe mechanism built into the application in case the user either wants to preserve battery on the device, notifications are spamming them because they are in a highly congested area, or for any other motive. The user may suspend the application for 10 minutes, 30 minutes, 1 hour, 4 hours, or until they decide to reactivate manually.

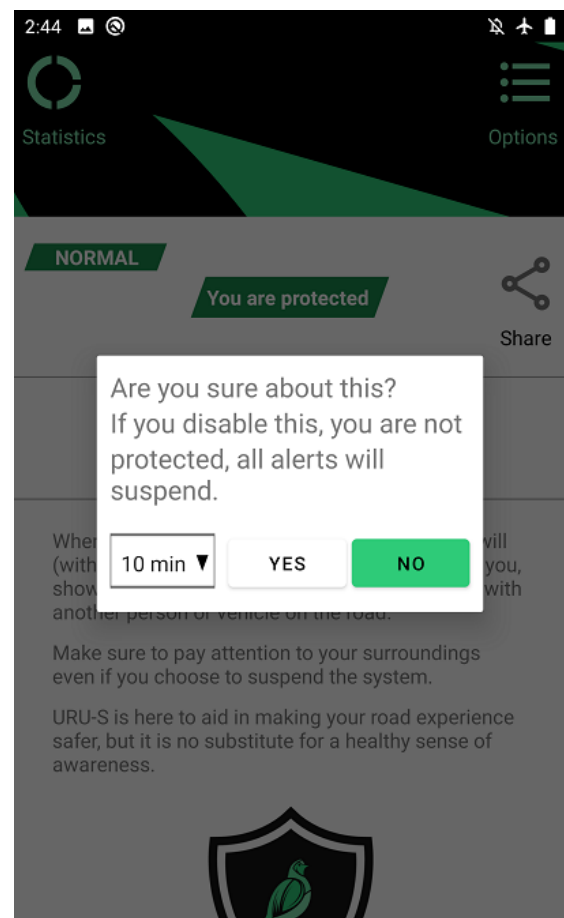


Figure 4.22: Alert Suspension Illustration

4.9.9 Geofencing and Danger-Zone Entry

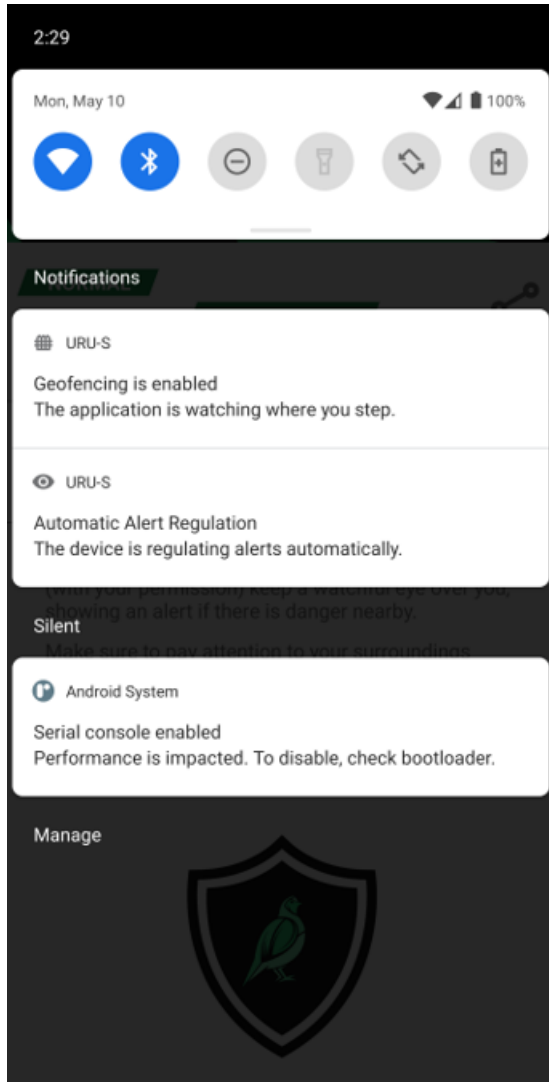


Figure 4.23: Geofencing Notification Illustration

Geofencing is essential to guarantee US12 (Receive Danger-Zone Notifications). It will only be active while the user is outdoors and moving, which is intrinsically tied to context-awareness. When active, the notification shown in figure 4.23 will appear on the user's device. The notification informs the user that URU-S is watching where they step, and, at any moment, they may click on it to know more about the service. Should they click on

the geofencing notification, they are brought to the explanation activity that tells them what geofencing is and what it does for them.

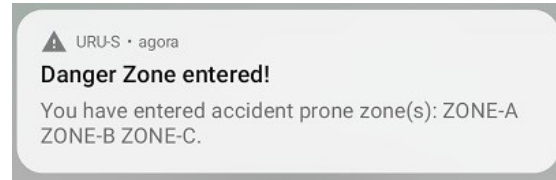


Figure 4.24: Danger-Zone Notification Illustration

Furthermore, if the user enters a dangerous area, and they have the Danger-Zone Entry Popup enabled, they will receive a notification telling them they have entered an accident-prone zone, which is shown in figure 4.24. Should they wish to position themselves better in regards to the danger they may potentially be facing, they can click on that notification to be brought to the map activity, as shown in figure 4.25.

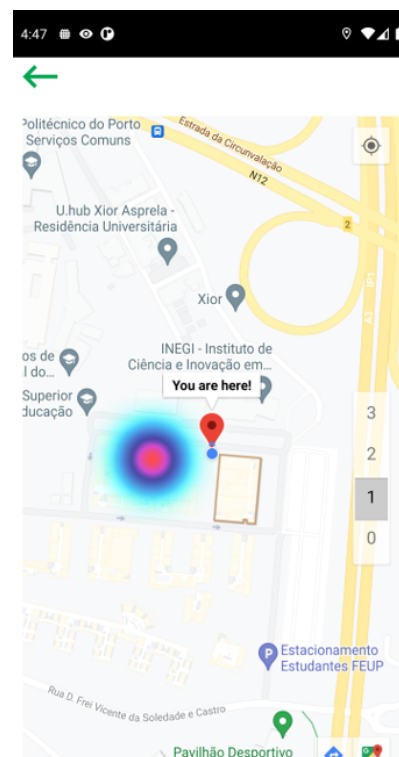


Figure 4.25: Danger-Zone Positioning Illustration

Chapter 5

Evaluation of the Proposed Solution

This chapter details an evaluation of the URU-S prototype from three different standpoints. First, from a programming verification standpoint, we will explain the methodology taken to write our unit tests using proper Android framework tools. Second, we will be looking at how our usability tests were conducted and our reasoning behind that. Third, we will show our additional measurements regarding context-awareness and latency.

5.1 Unit-Test-Based Verification

As mentioned in section [3.3.2](#), due to the technical difficulties experienced when trying to perform model checking, the verification was done primarily through instrumented unit tests. Each test uses AndroidJUnit4, and they make up a testing suite of 86 unit tests in total. To calculate code coverage, the results of which are shown in the next section, a Gradle command is used, "gradlew createDebugCoverageReport." The coverage report is calculated using JaCoCo [\[66\]](#), which considers the percentage of lines of code covered by the tests and the number of execution branches within said code that are being covered, organizing everything by package.

We made use of "ActivityScenarioRule" and "ServiceTestRule" to write our tests. Both of these components are a part of the "androidx.test" package. They were combined with previously mentioned tools such as Espresso and UiAutomator to check the behavior of activities and services, respectively.

To test an activity, we obtain an "ActivityScenario" object for the given activity and call the "onActivity" method of that class to test different scenarios. Each scenario consists of setting up a factor that we want to test and then asserting if the correct elements are on screen or the correct event took place. For example, to test the method that sets up the "donut" graphs in the Statistics section of the app, we have the code demonstrated in excerpt [5.1](#) and [5.2](#).

```

package org.feup.luis.uru_s;
import ...

@RunWith(AndroidJUnit4.class)
public class ReportsActivityTest extends TestCase {

    @Rule
    public ActivityScenarioRule<ReportsActivity> rule =
        new ActivityScenarioRule<>(ReportsActivity.class);

    @Test
    public void setUpDonuts() {
        Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();
        Database database = new Database(appContext);
        ActivityScenario<ReportsActivity> scenario = rule.getScenario();

        scenario.onActivity(activity -> {
            database.clearAlertData();

            activity.setUpDashboard();

            TextView total_alerts = activity.findViewById(R.id.total_alert_count);
            TextView true_positive_text = activity.findViewById(R.id.true_positive_alerts_text);
            TextView false_positive_text = activity.findViewById(R.id.false_positive_alerts_text);

            DonutProgressView donut_true_positive = activity.findViewById(R.id.donut_true_positive);
            DonutProgressView donut_false_positive = activity.findViewById(R.id.donut_false_positive);

            assertEquals(total_alerts.getText().toString(), activity.getString(R.string.no_alerts));
            assertEquals(true_positive_text.getText().toString(), actual: "0.0%");
            assertEquals(false_positive_text.getText().toString(), actual: "0.0%");
            assertTrue(donut_true_positive.getData().isEmpty());
            assertTrue(donut_false_positive.getData().isEmpty());
        });
    }
}

```

Figure 5.1: Reports Activity Test First Scenario

As we can see, two scenarios are being tested. The first scenario is when the alert database is empty, so when we call the "setUpDashboard()" method, we should expect to see all graphs empty and all labels saying nothing was received. The second scenario consists of inserting notifications of type "NOTIF_AVOIDED" (true positive), two of the type "NOTIF_IGNORED" (false positive), and one of type "NOTIF_SUSPENDED" (suspended alert). We expect there to be six alerts received; half of those should be true positives, a third should be false positives, and a sixth should be suspended. Additionally, we can also verify if the graphs have the correct labels and the correct colors. We expect green to be associated with the true positives, orange to false positives, and blue to suspended alerts. Should all of these factors be true, the test will pass. Should anything go wrong, then the test will fail.

```

scenario.onActivity(activity -> {
    database.clearAlertData();
    database.insertIntoAlert(Constants.NOTIF_AVOIDED);
    database.insertIntoAlert(Constants.NOTIF_AVOIDED);
    database.insertIntoAlert(Constants.NOTIF_AVOIDED);
    database.insertIntoAlert(Constants.NOTIF_IGNORED);
    database.insertIntoAlert(Constants.NOTIF_IGNORED);

    activity.setUpDashboard();

    TextView total_alerts = activity.findViewById(R.id.total_alert_count);
    TextView true_positive_text = activity.findViewById(R.id.true_positive_alerts_text);
    TextView false_positive_text = activity.findViewById(R.id.false_positive_alerts_text);

    DonutProgressView donut_true_positive = activity.findViewById(R.id.donut_true_positive);
    DonutProgressView donut_false_positive = activity.findViewById(R.id.donut_false_positive);

    assertEquals(total_alerts.getText().toString(), actual: "5 " + "Alerts Received");
    assertEquals(true_positive_text.getText().toString(), actual: "60.0%");
    assertEquals(false_positive_text.getText().toString(), actual: "40.0%");
    assertFalse(donut_true_positive.getData().isEmpty());
    assertFalse(donut_false_positive.getData().isEmpty());

    String true_positive = "True Positive Alerts";
    String false_positive = "False Positive Alerts";

    assertEquals(donut_true_positive.getData().get(0).component1(), true_positive);
    assertEquals(donut_true_positive.getData().get(0).component3(), actual: 0.6f);
    assertEquals(donut_true_positive.getData().get(0).component2(),
        ContextCompat.getColor(appContext, R.color.urv_green1_darker));

    assertEquals(donut_false_positive.getData().get(0).component1(), false_positive);
    assertEquals(donut_false_positive.getData().get(0).component3(), actual: 0.4f);
    assertEquals(donut_false_positive.getData().get(0).component2(),
        ContextCompat.getColor(appContext, R.color.urv_orange_darker));
});
}

```

Figure 5.2: Reports Activity Test Second Scenario

We try to test all relevant behaviors within all of our activities, and we can use other techniques to verify different kinds of behavior. For example, in our main activity, we have a status banner that changes depending on the application's state. We can instantiate some activity scenarios, as done previously, to test out if the banner is always consistent with the correct state; this is shown in code excerpt 5.3. As shown in the excerpt, to check if the correct graphic is on screen, we use a "DrawableCompare" auxiliary class that compares the number of pixels and bytes in two images to ascertain if they are identical.

```

@Test
public void checkCurrentAppStatus() {
    Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();
    ActivityScenario<MainActivity> scenario = rule.getScenario();

    scenario.onActivity(activity -> {
        MyPreferenceManager.editStringSharedPreferences(
            "APP-STATUS",
            appContext,
            "NORMAL"
        );

        activity.checkCurrentAppStatus();

        TextView statusBannerField = activity.findViewById(R.id.status_banner);
        TextView statusAsideField = activity.findViewById(R.id.status_text);
        SwitchMaterial suspension_switch = activity.findViewById(R.id.kill_switch);

        assertTrue(suspension_switch.isChecked());
        assertEquals(statusBannerField.getText().toString(), "NORMAL");
        assertEquals(statusAsideField.getText().toString(), "You are protected");

        assertTrue(
            DrawableCompare.drawableEqualPixels(
                statusBannerField.getBackground(),
                ContextCompat.getDrawable(appContext, R.drawable.app_working_rectangle)
            )
        );

        assertTrue(
            DrawableCompare.drawableEqualBytes(
                statusBannerField.getBackground(),
                ContextCompat.getDrawable(appContext, R.drawable.app_working_rectangle)
            )
        );
    });
}

```

Figure 5.3: Banner Tests with Drawable Compare

Furthermore, we can combine Espresso and UiAutomator in several different scenarios to simulate user interactions and directly manipulate the device, respectively. Espresso allows us to ascertain if certain elements are within a view or to interact with said elements. For example, we can use Espresso to slide our sliders in the alert settings activity. UiAutomator allows us to do similar things. However, it has the extra ability to open the notification tray of the device, which we use to test out our push notifications, and it can click anywhere on the screen. In contrast, Espresso is limited to the identifiable components of a view. A code excerpt that shows them both in action is [5.4](#).

```

@Test
public void customSoundPicker(){
    ActivityScenario<AlertSettingsActivity> scenario = rule.getScenario();

    scenario.onActivity(activity -> {

        onView(withId(R.id.custom_sound)).perform(ViewActions.click());

        UiDevice mDevice = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());

        /*
         These should be the coordinates of a file that is in the downloads folder
         In our emulator, the file sitting at those coordinates is "air.mp3"
        */
        mDevice.click(x: 1104, y: 814);
    });
}

```

Figure 5.4: Espresso and Ui Automator Working Together

The excerpt is trying to test the method that uploads a custom sound into the device. It belongs to the "AlertSettingsActivity," so that activity is instantiated with an adequate "ActivityScenario," however, we do not need to set anything up in the said scenario. We then use Espresso's "onView" method to localize the custom sound "image button" using its respective Android resource identifier. After localizing, we perform a "click" "ViewAction." Clicking on that button opens up the device's file system. At that point, Espresso is no longer valid as it is limited to "AlertSettings-Activity" components, and the Android file system activity exists outside of it. To compensate for Espresso's limitations, we use the UiDevice object (available from UiAutomator) and click a specific point on the screen where the respective card for a custom file sound is located. Of course, this test presumes that the emulator it runs on has the file downloaded and placed in that specific screen position. Whenever a test requires some form of simulated user interaction that requires assets to be in the emulator, code annotations will detail those assumptions.

```

/**
 * This service is not meant to be bound normally,
 * but for instrumentation testing purposes it will.
 * This is not used in the actual code, only for testing.
 */
public class LocalBinder extends Binder{
    ContextAwarenessForegroundService getService() {
        return ContextAwarenessForegroundService.this;
    }
}

```

Figure 5.5: Binding Services for Testing

To test our services, we must use a custom binder to bind the service being tested to a service "rule." Besides the binding process, the procedures and technologies used are the same as activities. Code excerpt 5.5 shows the example of a custom binder, in this case, associated with the "ContextAwarenessForegroundService" that is used explicitly for testing purposes. Additionally, a

code sample for the test relative to the method that starts and stops the different context-awareness detection is supplied in excerpt 5.6 to exemplify the binding process at an instrumented test level. We test every single detection and possible case; however, for simplicity's sake, the excerpt is abbreviated and only shows us binding the service and ascertaining that the movement detection has started by consulting a shared preference.

```
package org.feup.luis.urv_s.services;
import ...

@RunWith(AndroidJUnit4.class)
public class ContextAwarenessForegroundServiceTest extends TestCase {

    private static final long TIMEOUT = 14;
    @Rule
    public final ServiceTestRule serviceRule = new ServiceTestRule();

    @Test
    public void startAndStopDetection() throws TimeoutException {
        Context context = ApplicationProvider.getApplicationContext();
        Intent serviceIntent = new Intent(ApplicationProvider.getApplicationContext(),
            ContextAwarenessForegroundService.class);
        IBinder binder = serviceRule.bindService(serviceIntent);
        ContextAwarenessForegroundService service =
            ((ContextAwarenessForegroundService.LocalBinder) binder).getService();

        // Movement tests
        service.startAndStopDetection(Constants.MOVE_DETECTION, start: true);
        String should_have_move = MyPreferenceManager.getStringFromPreferences(
            "SHOULD-HAVE-MOVE",
            context,
            Constants.SHOULD_HAVE_MOVE_DETECTION_POSITIVE
        );
        assertEquals(should_have_move, Constants.SHOULD_HAVE_MOVE_DETECTION_POSITIVE);
    }
}
```

Figure 5.6: Context-Awareness Service Test

We test other components besides activities and services, but the techniques used are essentially the same. We use built-in methods from the Android testing framework whenever possible. If we need to simulate user actions in the current view or across multiple views, we may use Espresso or UiAutomator, respectively.

5.1.1 Results

The code coverage shown in figure 5.7 demonstrates that our instrumented tests have covered a great majority of the code (approximately 90%). We can also see that most execution branches have been covered as well. All application packages have next to 100% instruction coverage, except for the "receivers" and "dialogs" packages.

As we can see from figure 5.8, the "receivers" package is not fully covered because of the `GeofenceBroadcastReceiver`. We cannot fully cover the geofencing methods as, to do so, we would have to simulate a geofence entry event. Although we have the methods to attempt such a thing, the official documentation [67] is deficient in explaining how to generate a valid geofence entry transition intent through hard-code. Due to this, we cannot test the code execution branch where the application enters a geofence. Regardless, the geofencing mechanism has been verified via field testing and confirmed to be working as intended when a geofence entry is detected.

Moreover, we can see that the "dialogs" package has insufficient coverage. Testing a dialog involves having them show up on the screen. Even though this is possible as individual tests, using them in the testing suite of the application will cause it to hang indefinitely. This deadlock happens because tools like Espresso can experience unpredictable behavior when dealing with dialog objects, even when the dialog itself has been dismissed by calling the correct method. Despite this, the code within each dialog consists of a simple "YES" or "NO" case, depending on whether the user clicked the former or the latter; confirming a dialog will set a given shared preference to another value. At most, a dialog will start a timer, which is what happens when the user suspends the application's alerts with the real-time alerts switch, as seen in figure 4.22 of the previous chapter. The code behind these elements is uncomplicated and similar to other instructions that are extensively covered. The dialog code is also a very minute portion of the overall code-base, and they have been tested empirically.

In conclusion, every relevant behavior of the interface was appropriately verified, which is further cemented by the logs of our UI Exerciser Monkey [46]. The application was submitted to stress tests of up to one hundred thousand pseudo-random events. Barring a single experiment that yielded a crash from URU-S that was swiftly patched, the application managed to resist all stress tests without crashing once. An excerpt of one of the various logs from the "Monkey" tool can be consulted in excerpt 5.1. It is also worth mentioning that participants in the usability testing phase did not find any bugs with the application.

debugAndroidTest





















Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 org.feup.luis.uru_s.utilities.dialogs		17%		0%	45	49	95	117	14	18	2	4
 org.feup.luis.uru_s		93%		66%	117	404	74	1,206	14	216	0	29
 org.feup.luis.uru_s.services		92%		68%	41	129	23	353	5	55	1	7
 org.feup.luis.uru_s.receivers		70%		40%	7	18	17	60	0	8	0	3
 org.feup.luis.uru_s.utilities		92%		61%	11	41	11	131	6	32	0	8
 org.feup.luis.uru_s.database		98%		72%	34	100	4	128	2	23	0	1
 org.feup.luis.uru_s.utilities.interfaces		100%		n/a	0	1	0	8	0	1	0	1
Total	885 of 8,188	89%	267 of 695	61%	255	742	224	2,003	41	353	3	53

Figure 5.7: Code Coverage Obtained with JaCoCo [66]

org.feup.luis.uru_s.receivers








Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 GeofenceBroadcastReceiver		64%		40%	7	14	17	48	0	4	0	1
 AvoidedNotificationReceiver		100%		n/a	0	2	0	6	0	2	0	1
 IgnoreNotificationReceiver		100%		n/a	0	2	0	6	0	2	0	1
Total	68 of 229	70%	12 of 20	40%	7	18	17	60	0	8	0	3

Figure 5.8: Receivers package coverage


```

1 05-25 18:15:14.999 17411 17411 D AndroidRuntime: Calling main entry com.android.commands.monkey.Monkey
2 05-25 18:15:15.000 17411 17411 W Monkey : args: [-p, org.feup.luis.uru_s, --throttle, 500, --ignore-crashes, --ignore-timeouts,
  --ignore-security-exceptions, --monitor-native-crashes, -v, 100000]
3 05-25 18:15:15.000 17411 17411 W Monkey : arg: "-p"
4 05-25 18:15:15.000 17411 17411 W Monkey : arg: "org.feup.luis.uru_s"
5 05-25 18:15:15.000 17411 17411 W Monkey : arg: "--throttle"
6 05-25 18:15:15.000 17411 17411 W Monkey : arg: "500"
7 05-25 18:15:15.000 17411 17411 W Monkey : arg: "--ignore-crashes"
8 05-25 18:15:15.000 17411 17411 W Monkey : arg: "--ignore-timeouts"
9 05-25 18:15:15.000 17411 17411 W Monkey : arg: "--ignore-security-exceptions"
10 05-25 18:15:15.000 17411 17411 W Monkey : arg: "--monitor-native-crashes"
11 05-25 18:15:15.000 17411 17411 W Monkey : arg: "-v"
12 05-25 18:15:15.000 17411 17411 W Monkey : arg: "100000"
13 05-25 18:15:15.001 17411 17411 W Monkey : data="org.feup.luis.uru_s"
14 (...)
15 05-25 18:15:15.057 17411 17411 I Monkey : // Event percentages:
16 05-25 18:15:15.057 17411 17411 I Monkey : // 0: 15.0%
17 05-25 18:15:15.057 17411 17411 I Monkey : // 1: 10.0%
18 05-25 18:15:15.057 17411 17411 I Monkey : // 2: 2.0%
19 (...)
20 05-25 21:29:20.663 17411 17411 I Monkey : :Sending Touch (ACTION_DOWN): 0:(67.0,189.0)
21 05-25 21:29:20.668 17411 17411 I Monkey : :Sending Touch (ACTION_UP): 0:(127.03428,253.60945)
22 05-25 21:29:21.173 17411 17411 I Monkey : :Sending Touch (ACTION_DOWN): 0:(913.0,1289.0)
23 05-25 21:29:21.176 17411 17411 I Monkey : :Sending Touch (ACTION_UP): 0:(897.45966,1286.4905)
24 05-25 21:29:22.184 17411 17411 I Monkey : :Sending Trackball (ACTION_MOVE): 0:(-2.0,-4.0)
25 05-25 21:29:22.186 17411 17411 I Monkey : Events injected: 100000
26 05-25 21:29:22.188 17411 17411 I Monkey : :Sending rotation degree=0, persist=false
27 05-25 21:29:22.198 17411 17411 I Monkey : :Dropped: keys=0 pointers=37 trackballs=0 flips=245 rotations=0
28 05-25 21:29:22.198 17411 17411 I Monkey : ## Network stats: elapsed time=11647137ms (0ms mobile, 0ms wifi, 11647137ms not
  connected)
29 05-25 21:29:22.198 17411 17411 I Monkey : // Monkey finished

```

Listing 5.1: Monkey log example

5.2 Usability-Test-Based Validation

When it comes to usability testing, there are essentially two routes to take: quantitative and qualitative. The quantitative route is adequate for proving something. However, it must be done very rigorously and with a large representative sample size that could be extrapolated to a population to draw significant conclusions. Experts in laboratories usually take this route, observing participants through a one-way mirror and communicating tasks to them in a "Voice of God" sort of way. These experts measure factors like the number of tasks completed and how long it takes to complete said tasks; a rigorous protocol must be followed for each participant. The qualitative route enables people to get insights on improving what they are building, is simpler to do, and far more informal. The facilitator sits in a room with a participant, gives them a series of tasks to complete, and asks them to "think out loud," querying them about why they feel a certain way, what difficulties they may have, and how that could be improved [68].

Due to the current COVID-19 pandemic and the conditions available at the Faculty, it would be very challenging to gather hundreds of participants and observe each of them through a one-way mirror as if we had an actual usability lab and a recruitment agency at our beck and call. Thus, the testing done regarding the URU-S prototype is closer to the qualitative end of the spectrum. To test our prototype, we invited participants to room I322 and sat with them, giving them a series of tasks and asking them to externalize their thought process. The facilitator also tried to do some measurements of how long participants took to complete a task, whether or not they had completed it, and how difficult it was for them on a scale of 1 to 5. The usability test also included an outdoor phase where the participants would interact with the application's status change. A simulated "danger-zone" would be approached, and then participants would have to face a dangerous element in a limited visibility intersection. For more details, one may consult the usability testing report in appendix B.

It is worth mentioning that we had eight participants for the testing phase. The sample size is small and not enough to reach statistical relevance. However, to quote Steve Krug in his book "Rocket Surgery Made Easy" [68], "the point of this kind of testing is not to prove anything; the point is to identify major problems and make the thing better by fixing them. It just works because most of the kinds of problems that need to be fixed are so obvious that there is no need for 'proof'". That statement is further reinforced by the fact that URU-S is a straightforward application with only four main screens ("Statistics," "Home," "Options," and "Alert Settings"), a couple of notifications, and two services that act to protect our users. The prototype is not particularly complex to deal with, and even less tech-savvy participants managed to find their way around it.

5.2.1 Setup

In appendix B we go more in-depth into the results obtained in the usability testing phase and offer a conclusion, but we omit our setup. We can use this section to describe that. As mentioned in the report, the test consisted of two phases. Phase 1 was held in room I322, where the participant and the testing facilitator sat next to each other and worked their way through a set of tasks; the prototype was presented in testing phones available at the institute, namely the "Xiaomi Mi MIX 3 5G" and the "SAMSUNG-SM-G977B". Phase 2 was held outdoors and culminated at the "RAMP" field testing area. An illustration of that area may be found in figure 5.9. The participant would wait at the "X" marked on the left of the image, while the facilitator would hide behind the rightmost wall. The facilitator would then hook up their laptop to the camera like what is shown in figure 5.10. After setting that up, the facilitator would prompt the participant to descend the ramp, following the direction of the green arrow highlighted in 5.9. Once the participant had breached the camera's field of view, the facilitator would engage a warning on the testing device and simultaneously launch a remote-controlled car in the direction of the red arrow. The remote-controlled car would act as a simulated danger, and the entire situation would mimic a limited visibility intersection. The camera's feed was also recorded, using the windows built-in camera application so that the facilitator could keep a registry of the participant's video reactions. Ideally, each participant should stop before hitting the remote-controlled car, as the warning is given well in advance. Once the participant has avoided the car, they are expected to register to avoid danger using URU-S's notification.

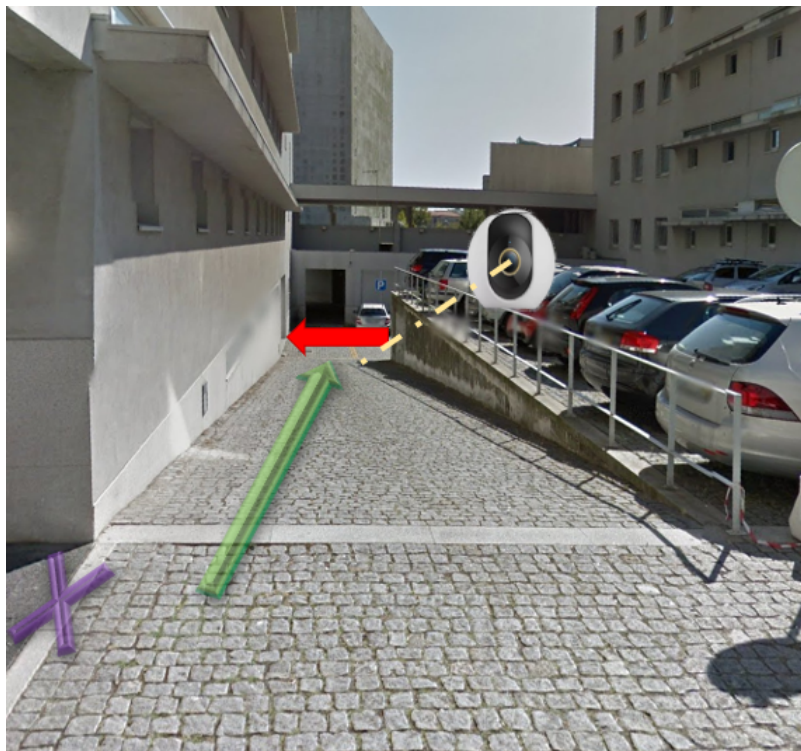


Figure 5.9: Field Testing Area Illustration



(a) Laptop placed behind the wall



(b) Camera overseeing the ramp

Figure 5.10: Camera and Laptop Connected

5.2.2 Result

All participants managed to react to the warning and avoid the remote-controlled car thrown at them, stopping before hitting it. As described in appendix B, they also had minimal difficulties with the interface. In that appendix, we can also see how long users took to complete tasks (this was measured by starting a timer on a smartphone for each task), their difficulty, and whether they managed to complete them without help. Overall, participants managed to understand the general concept of the application, contributing to its development with a plethora of minor tweaks that could improve its general usability; these recommendations are also described in the appendix. It is imperative to continue to work with users to keep URU-S centered on their needs and reactions.

5.3 Additional Measurements

Latency and context-awareness measurements have been done with the current setup to evaluate the performance of URU-S quantitatively. The latency requirement is outside of this project's scope; however, we measured it as a relevant dimension for an application such as URU-S. We conducted some measurements using WiFi, 5G, and the Firebase server as an intermediate. Latency was measured by subtraction of endpoint application timestamps placed in strategic points of our testing infrastructure; it is relevant to measure latency under these different technologies to evaluate the network access comparatively.

5.3.1 Context-Awareness

URU-S contains a context-awareness mechanism. It is relevant to understand the time and distance traveled until a state swap happens because those represent "dead-zones" for the alert system. When a user travels outside, coming from a building, for the period that the application takes to recognize that they are outdoors, alerts are still disabled. A similar thought process is applied when the user is outdoors, stops, goes into the "STATIONARY" state, and then begins to move again. Until the application recognizes that the user is moving, alerts are still disabled. The following questions can frame our experiments:

1. How long does the application take to switch to a "NORMAL" state when exiting a building?
2. How much do we walk out of the building until the application switches to a "NORMAL" state?
3. How long does the application take to switch to an "INDOOR" state when entering a building?
4. How much do we walk into the building until the application switches to an "INDOOR" state?
5. How long does the application take to switch to a "STATIONARY" state when we stop moving?
6. How long does the application take to switch to a "NORMAL" state when we started moving and were previously standing still?
7. How much do we walk until the application switches to a "NORMAL" state, knowing we were previously standing still?

To get a sense of how damaging these "dead-zones" would be for the prototype, we measured the time taken, and distance traveled until a state swap occurred. The way this was done was using a screen recorder on a device that had URU-S installed and using another application called SenseMyCity [69] to measure the distance traveled during each testing session. SenseMyCity also gave us measurements of time taken during the session, but we would always match that with the screen recording to see how long it took for the swap to occur. The tester would always initiate the screen recorder and SenseMyCity simultaneously before beginning a test.

Tests consisted of walking in and out of a building to test the indoor and outdoor detection and moving and stopping on an outdoor space to test movement detection. In total, 75 measurements were made, 36 of which are indoor and outdoor, and the remaining 39 are movement detection.

5.3.1.1 Indoor/Outdoor

The tester had to walk inside and outside of a building to get these measurements. The tester would always factor out the time taken to walk to the door to measure how long it took to swap into the proper state; the countdown would only start once they had stepped foot inside or outside the building.

For all 36 measurements in both directions (indoor to outdoor and outdoor to indoor), we noticed that the application changed states more rapidly when going into the building instead of leaving it. This pattern motivated the tester to split the data by direction. This split means we would analyze the parts of the overall sample relative to moving in the indoor to outdoor direction separate from the outdoor to indoor direction. Thus, the main data-set was split into two groups of 18 measurements for each direction.

The "indoor to outdoor" group had a mean value of 34,33 seconds with a sample deviation of 10,48 seconds, coupled with a mean value of 56,11 meters with a deviation of 22,53 meters for distance traveled. The data gathered means we can answer question number one with "approximately 34,3 seconds" and question number two with "roughly 56,1 meters".

On the other hand, the "outdoor to indoor" group had a mean value of 19,28 seconds with a deviation of 13,06 seconds, coupled with a mean value of 20,11 meters with a deviation of 19,11 meters. Our data means we can answer question number three with "approximately 19,3 seconds" and question number four with "roughly 20,1 meters".

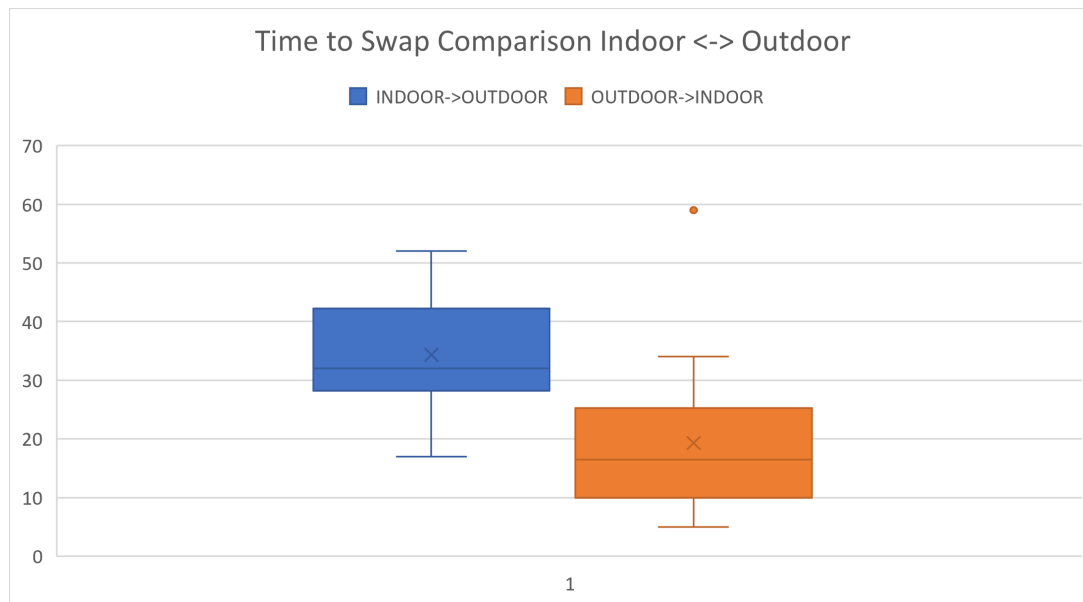


Figure 5.11: Comparison Between the time to Swap in both Directions Indoor/Outdoor

Moreover, in terms of outliers, our data is far more consistent here than for the latency measurements. If we plot the time taken to swap in either direction, we can see from figure 5.11 that we only detect a single outlier in the outdoor to indoor direction. This outlier may be explained by the fact that the tester lost network connectivity during one session, as the Faculty had an outage.

When there is no network connectivity, it seems that URU-S has trouble detecting that the user is indoors.

5.3.1.2 Moving/Stationary

The tester started and stopped their motion on an open space to measure this detection. There were 39 measurements taken, 20 of which were the tester stopping after moving, and the remaining 19 consisted of moving out of a stationary state. It is important to note that time is tracked differently for these tests. Time starts ticking for the "moving to stop" scenario after the tester goes into a full-stop; time stops ticking once the application recognizes they are no longer moving. For the "stopped to moving" scenario, time begins to tick as soon as the tester starts their movement and finishes ticking once the application has realized they are no longer standing still. In the scenario where we stop and wait for the application to realize our immobility, the distance traveled is 0 meters. In the other scenario, distance traveled relates to a "dead-zone" where the tester is still labeled as being "stationary" by the application when they are already moving.

Similar to the previous test, we have two groups of divided samples. It is worth noting that our values relative to distance traveled only consider the 19 values of the "stationary to moving" sample, as the inverse direction has 0 travel distance.

When analyzing the samples separately, a mean value of 11,80 seconds with a deviation of 7,40 seconds was obtained for the "moving to stationary" scenario. These measurements mean we can answer question number five with "approximately 11,8 seconds".

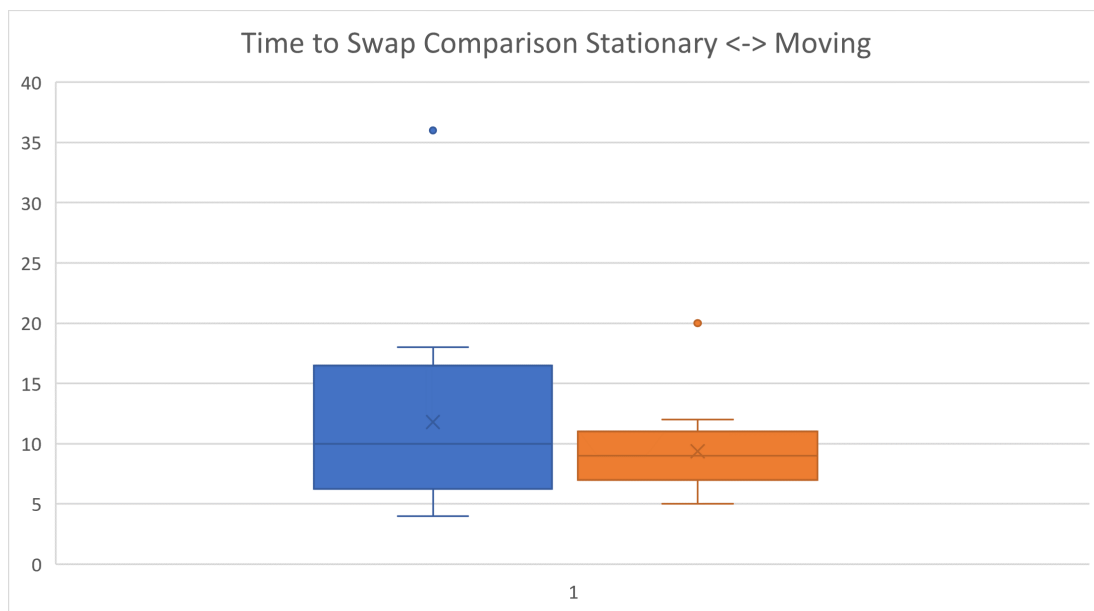


Figure 5.12: Comparison Between the time to Swap in both Directions Moving/Stationary

The "stationary to moving" scenario had a mean value of 9,37 seconds with a deviation of 3,27 seconds. For distance traveled, we had a mean value of 5,63 meters with a deviation of 11,05 meters. As we can see from figure 5.12, the time swap difference between each scenario is

far less relevant for this experiment than the last. However, we can still see that the application swaps faster in this case. Either way, we can answer question number six with "approximately 9,4 seconds" and question number seven with "roughly 5,6 meters".

5.3.1.3 Conclusion

We can conclude that URU-S takes longer to recognize that users have left a building than their entry upon it, presuming when they enter the building, it can automatically connect to an indoor WiFi connection. Additionally, it can detect that the user is moving or standing still even faster than localizing them. These differences in detection sensitivity are consistent with the implementation detailed in section 4.3. In the mentioned section, we specify that our context glimpses are filtered to prevent the application from being overloaded. The indoor to outdoor glimpses have more filtering than the moving to stationary ones, thus justifying these differences in detection.

Overall, these "dead-zones" are not too damming for the application, as, on average, URU-S will detect changes in a reasonable time span. This time span coupled with average human walking speed means users will not travel too far while in a "dead-zone" either. Moreover, the application allows users to override context-awareness and manually enable alerts if they need immediate protection, mitigating the problems associated with not having instant and ultra-reliable state detection.

5.3.2 Latency

URU-S attempts to prototype a VRU protection system. These systems are safety-critical, and receiving warnings hundreds of milliseconds too late can be the difference between avoiding an accident or getting hit by another road user. It is important to get a sense of the latency associated with these warnings, the components that make up that latency, and the different access technologies that can be integrated into the system to minimize latency. By studying each component of the latency, we can discover the main source of slowdown and make recommendations for future systems regarding which aspects should be improved. By experimenting with different access technologies in different coverage scenarios, we may compare network access and get a sense of how they would behave in a system such as this one. Additionally, our target delay here is 300 milliseconds; as mentioned by Scholliers et al., "road safety and pre-crash applications require an estimated 300 ms end-to-end latency time as stated in ETSI TS 101539-1" [3]. The following questions can frame our experiments:

1. What is the main component that affects our latency values, considering our current infrastructure?
2. What latency values can we expect when connected to a poor coverage WiFi network?
3. What latency values can we expect when connected to a good coverage WiFi network?
4. What latency values can we expect when connected to a poor coverage 5G network?
5. What latency values can we expect when connected to a good coverage 5G network?
6. When comparing WiFi to 5G at their best, which one guarantees lower latency?
7. Can we meet a target delay of 300 milliseconds?

First and foremost, we define "latency" as the time required for a warning message to leave the designated JavaScript testing tool running in a laptop and reach the destination smartphone's instance of the Firebase Messaging service. To calculate this, we use the formula listed below.

$$L = t_3 - t_0$$

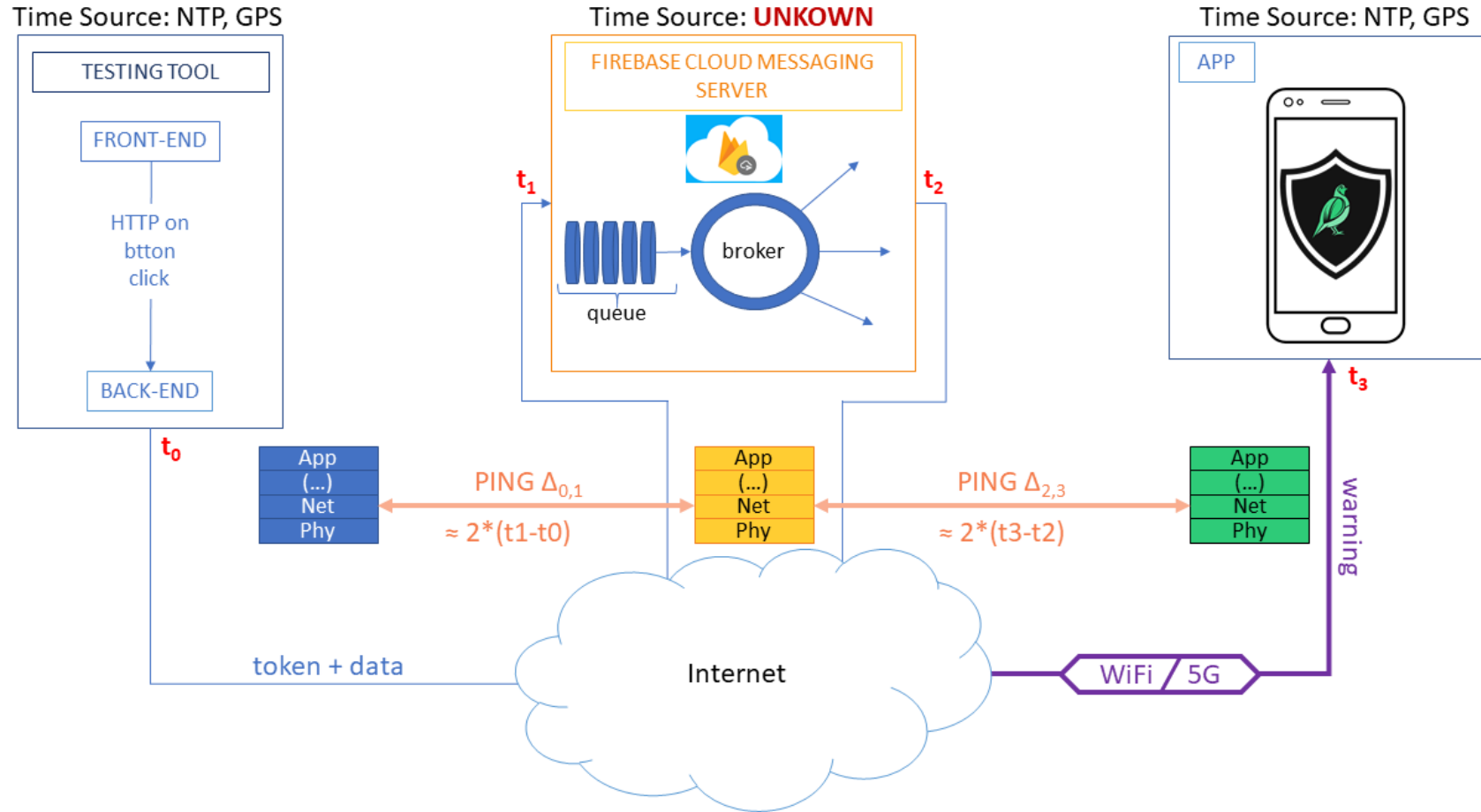


Figure 5.13: Packet Travel Diagram where $L = t_3 - t_0$. We consider t_i : application layer timestamp placed in points of the infrastructure; t_0 : timestamp obtained before the data and token payload is delivered to the server (observable and comparable); t_1 : timestamp marking the arrival of the payload at the server (not observable); t_2 : timestamp for message acceptance by the server, after going through message queues, the internal broker and before sending the data out to the device (observable but not comparable due to an unknown time sync source); t_3 : timestamp for message acceptance by the smartphone, engaging the warning for the VRU (observable and comparable).

In the formula, L is a latency value expressed in milliseconds, t_3 is the timestamp for message delivery, and t_0 is the timestamp for message sending. Additionally to the endpoint timestamps, it is helpful to understand the components of the latency, and that is why it is essential to try to measure intermediate timestamps such as t_1 and t_2 that mark the arrival and departure of the data payload to the Firebase server, respectively. These timestamps are marked according to our infrastructure, as we can see in figure 5.13. In this diagram, t_i represents an application layer timestamp, being that t_1 is not observable and t_2 is observable but not comparable, as the time source for the Firebase Cloud Messaging Server cannot be controlled. Since we do not know the synchronization mechanism for the Firebase server, we cannot evaluate all components of the absolute delay.

We will rely on the larger interval from t_3 to t_0 . We evaluated WiFi and 5G access comparatively and *pinged* the Firebase server from the laptop and the smartphone. Suppose the overhead from the Firebase server's message queues is negligible. In that case, the *ping* from the laptop to Firebase should be representative of twice the interval $t_1 - t_0$ and vice-versa for the interval $t_3 - t_2$. Should the *ping* values be relatively low, then this indicates that the latency is mainly affected by the overhead introduced by the Firebase server despite its physical distance. On the other hand, if the *ping* values are high, it could mean that the server is positioned too far to be useful, highlighting the importance of having a more local alternative (situated in Lisbon, for example).

To guarantee that $L > 0$, the system clocks of the laptop and smartphone must be appropriately synchronized. Once both clocks are synchronized, we print out t_0 before making the HTTP request to the Firebase Cloud Messaging Server and then print out t_3 when the message lands on the smartphone, triggering the "onMessageReceived()" method of the Firebase Messaging service. We then take the logs obtained from the testing tool and the smartphone logs and subtract each message timestamp pair. Two methods can be used to achieve clock synchronization: NTP and GPS. Both of these methods were attempted. With NTP (Network Time Protocol), we tried to synchronize the devices by forcing the laptop to sync its system clock with "2.android.pool.ntp.org", using the Linux command "ntpdate 2.android.pool.ntp.org". According to the official Android documentation on the subject [70], the previously mentioned server is what Android devices use by default to sync their clocks, and most commercial Android devices sync their clocks with network-provided time. Synchronizing with GPS time was also tested; however, this required a rooted device. Some Android versions allow the user to change the time synchronization to GPS time instead of network-provided time without root permissions, but those were not available for testing. Using a rooted device and Smart Time Sync [71], we managed to synchronize the smartphone with GPS time. To synchronize the laptop, we used a GNSS 7 CLICK circuit board connected to a GPS receiver. The circuit-board was plugged into the laptop, and the NMEATime2 - PC GPS Time Synchronization [72] program was used to synchronize the computer's clock with GPS time.

It is worth mentioning that, even with both devices synchronized, the time interval for our current infrastructure could be affected by a myriad of different factors. The Firebase Cloud Messaging Server could be experiencing much traffic, causing a significant slowdown on its behalf;

this issue is not only a traffic peak but an inherent delay that we cannot quantify. Additionally, it is essential to remember that the message queues for the previously mentioned server are not optimized for low-latency safety-critical road-safety scenarios, as that is not the primary use case. Moreover, the Firebase Cloud Messaging Server would sometimes return "connection refused" errors during our experiments, perhaps because we were sending too many messages at a time. Thirdly, there could be paging delays associated with the experimental 5G station. Only the very last portion of the interval, colored purple in figure 5.13, relies on the phone's connectivity.

To reduce the number of varying factors and control our variables better, we tried to measure the interval $t_3 - t_2$. We would obtain t_3 at the smartphone and, instead of tracing the origin point back to when the message is sent initially from the testing tool, we would subtract t_2 . The new variable t_2 would be the timestamp for when the message is accepted by the server and sent to the device. Theoretically, it is possible to do this by linking our Firebase Messaging instance with the BigQuery service [73]. However, when we attempted this, we quickly found that we would sometimes "lose" messages that had been accepted but did not have the corresponding delivery timestamp. BigQuery would also occasionally list values that suggested messages had been delivered before being accepted by the server, resulting in negative latency. The negative latency values lead us to believe that the timestamps obtained from the integration were not synchronized. Furthermore, BigQuery would report that we had different instances of the same Firebase Messaging service running in the same device simultaneously, which is nonsensical. All of these issues added to the fact that all data gathered using this method would come with a delay of about 48 hours, unlabelled. What we mean by "unlabelled" is that, supposedly, we are allowed to append a Firebase Analytics label to our messages, which we would use to label our experiments. BigQuery never loaded a single Firebase Analytics label. After filing bug reports, no significant answers were given. Due to these problems, we have decided not to take this route. Even though our current interval ($L = t_3 - t_0$) makes it challenging to reach definite conclusions, it has well synchronized and immediate values, so that we will opt for that instead due to a lack of a better option.

In the following section, we will be estimating the intervals $t_1 - t_0$ and $t_3 - t_2$ using the *ping* command to send packets to the Firebase server.

5.3.2.1 Ping values

Let us consider that $\Delta_{0,1} = t_1 - t_0$ and $\Delta_{2,3} = t_3 - t_2$. As we can see by figure 5.13, we can *ping* the Firebase server to approximate the $\Delta_{0,1}$ and $\Delta_{2,3}$ intervals. The approximation via *ping* will give us a notion of the network layer latency on our messages. As mentioned previously, we can also deduce if our latency values will be mostly influenced by the network connectivity on our devices or by the overhead introduced by the server. It is worth mentioning that each *ping* travels to and from the server, whereas we are interested in only one of the travel paths. For $\Delta_{0,1}$, we are interested in packets heading towards the server from our testing tool. For $\Delta_{2,3}$, we are interested in packets heading from the server to our smartphone. In any case, we presume that our intervals can be calculated by approximation of half the average *ping* (*AVG*), so: $\Delta = \frac{AVG}{2}$.

The following measurements for $\Delta_{0,1}$ and $\Delta_{2,3}$ are relative to the same WiFi network. Both devices (phone and laptop) were right next to the router, emulating a good coverage condition. We obtained the following:

Table 5.1: WiFi *Ping* $\Delta_{0,1}$

Packets sent	AVG (ms)	$\Delta_{0,1}$ (ms)
198	19,292	9,9645
180	19,896	9,948
299	18,694	9,347
82	18,079	9,0395

Table 5.2: WiFi *Ping* $\Delta_{2,3}$

Packets sent	AVG (ms)	$\Delta_{2,3}$ (ms)
289	72,935	36,4675
256	54,289	27,1445
34	51,912	25,956
155	62,452	31,226

All of our measurements were done through a *ping* with hostname "fcm.googleapis.com." We experimented with a varying number of packets. We found that the *ping* on the smartphone's side, $\Delta_{2,3}$, was significantly superior to the one obtained on the laptop's side, $\Delta_{0,1}$. Evidently, it seems we can presume that $\Delta_{0,1} \approx 10$ (ms) and $\Delta_{2,3} \approx 30$ (ms), giving us an overall network latency of $\Delta_{0,1} + \Delta_{2,3} \approx 40$ (ms). It is worth remembering that this latency is calculated without factoring in the overhead from the Firebase server.

The following measurements for $\Delta_{2,3}$ were obtained with a cellular connection on the smartphone, placing it in line of sight of the experimental 5G station and thus emulating a good coverage scenario. We obtained the following:

Table 5.3: 5G *Ping* $\Delta_{2,3}$

Packets sent	AVG (ms)	$\Delta_{2,3}$ (ms)
105	39,714	19,857
63	41,959	20,9795
141	41,007	20,5035

As we can see, in the 5G test, $\Delta_{2,3} \approx 20$ (ms). It is worth mentioning that the *ping* on the laptop's side remained relatively the same at about 20 milliseconds of average *ping*. Since the *ping* for the laptop did not change, we can keep $\Delta_{0,1} \approx 10$ (ms). Taking the above into consideration, the network latency is now $\Delta_{0,1} + \Delta_{2,3} \approx 30$ (ms). Once again, these calculations are without factoring in the overhead from Firebase.

We can expect a ten millisecond difference between a good connectivity scenario for WiFi and 5G in receiving packets from the Firebase server. We conclude that even with WiFi, we only have a network latency of about 40 milliseconds, which is not too significant and indicates that the

Firestore server's physical distance from the device does not critically impair the latency interval (L). However, it is worth mentioning that with an unstable internet connection, one can obtain high *ping*. For example, suppose the smartphone is placed within the Telecommunications Institute, outside of the 5G station's line of sight. In that case, we observe an average *ping* of about 186 milliseconds, meaning $\Delta_{2,3} \approx 93$ (ms). If we were to add ten milliseconds for the laptop, that would mean an overall network latency of 103 milliseconds. Moreover, *ping* may experience fluctuations for both the smartphone and the laptop, being more significant on the smartphone's side even when connected to the same network; as far as we can tell, this is due to a hardware characteristic. We decided to evaluate the connectivity in a good coverage scenario and disregard other factors for this study. These factors include outliers, outages, instability, high-traffic congestion, and scalability. We want to observe network latency in nominal conditions; we are also limited in our time and capacity to investigate all of those factors at once.

On that note, since our *ping* values are relatively low (taking into account the above assumptions), this means that the values of L that will be presented in the following sections will be majorly influenced by the time that a message takes to sort itself out within the Firestore server. We delimit this interval in figure 5.13 with time stamps t_2 and t_1 . The time a message takes to be accepted and forwarded to the correct phone in the Firestore server is given by $t_2 - t_1$. Once again, this interval cannot be calculated directly since t_1 is not observable, and it can only be estimated based on our *ping* values.

In conclusion, we can answer question number one and say that "the main component that affects our latency values, taking into account our current infrastructure" is the delay associated with the Firestore server's internal message handling, given by the interval $t_2 - t_1$.

5.3.2.2 WiFi

WiFi measurements were done in the tester's domicile using their Xiaomi Redmi Note 8 Pro with network-time sync enabled by default. The router is a dual-band WiFi certified 802.11ac device that operates in both 2,4 GHz and 5 GHz. The tester's laptop had been synchronized with the NTP server "2.android.pool.ntp.org". Two scenarios were tested: one with good WiFi coverage and another with worse coverage. In both of those scenarios, a series of warning messages were sent using a script.

The poor coverage scenario was done first. The poor coverage conditions were achieved by accessing the home WiFi network from a distant spot in the house. The network speed at the time of the test was 12,40 Mbps download and 1,23 Mbps upload. The smartphone and the laptop were both connected to the same network. With a sample size of 176 messages, we obtained a mean value of 471,35 milliseconds and a sample deviation of 231,80 milliseconds. Most of our values are within the 253 to 533 millisecond range. Our boxplot, shown in figure 5.14, also demonstrates that we had six outlier values that were above 1 second, which could be explained by network fluctuations in an unstable connectivity area such as this one. We can answer question number two and say that in a "poor coverage WiFi network," we expect a latency of about 471 milliseconds.

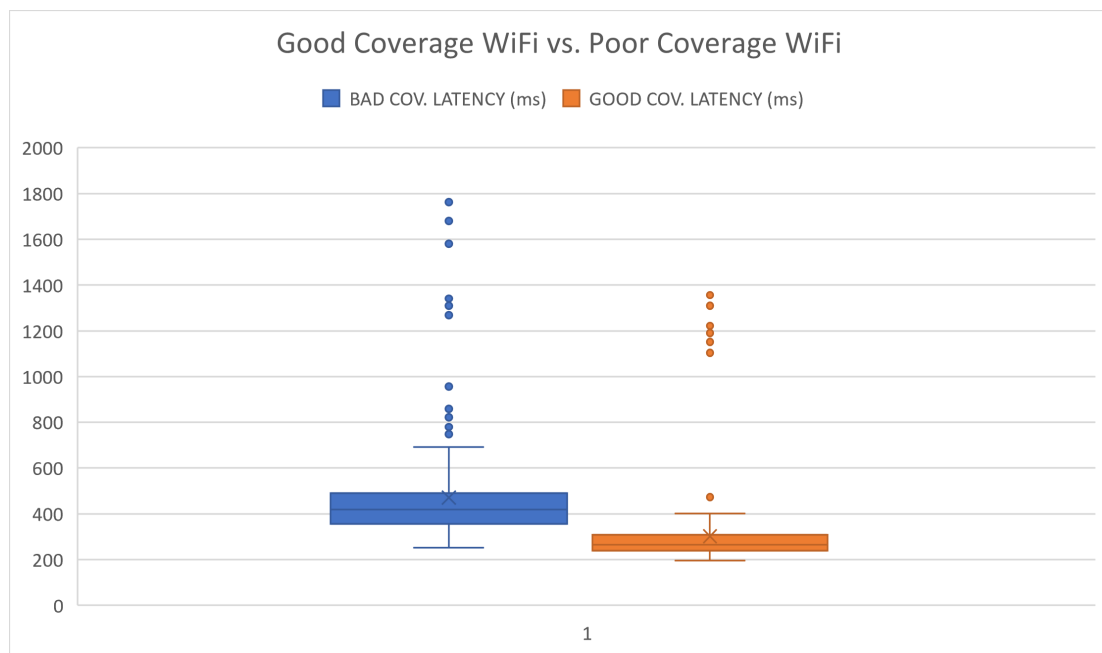


Figure 5.14: WiFi Boxplots for Good and Bad Coverage

The subsequent testing scenario was a good coverage scenario. The coverage conditions were achieved by accessing the home network while standing right next to the router. The network speed at the time of the test was 295,50 Mbps download and 102 Mbps upload. The smartphone and the laptop were, once again, connected to the same network. With a sample size of 193 messages, we obtained a mean value of 301,91 milliseconds with a sample deviation of 172,28 milliseconds. Most of our values are within the 197 to 397 millisecond range. Similar to our previous test, our boxplot, shown in figure 5.14, demonstrates six outlier values above 1 second. However, we now have fewer outliers in total, and they barely reach 1,40 seconds, whereas, in the previous experiment, two of our outliers were above 1,60 seconds. We can answer question number three and say that in a "good coverage WiFi network," we expect a latency of about 302 milliseconds.

5.3.2.3 5G

5G measurements were done at the Telecommunications Institute Shannon Laboratory (I322) and outdoors in the grass fields next to building B at FEUP. The Android device used for testing was a Xiaomi MI 10 5G with root privileges and the capability of accessing the experimental 4G/5G station at FEUP. To synchronize the smartphone and the laptop, GPS was used. Two scenarios were tested: one with good 5G coverage and another with worse coverage. In both of those scenarios, a series of warning messages were sent using a script.

The poor coverage scenario was done first. The coverage conditions were achieved by enabling mobile data on the testing device and placing it on one of the laboratory window-sills. Due to how the 5G station is angled, the laboratory should be a low coverage area, as it sits almost wholly

out of its field of view. The network speed at the time was 2.11 Mbps download and 13.4 Mbps upload. With a sample size of 103 messages, we obtained a mean value of 577,7 milliseconds and a sample deviation of 850,6 milliseconds. Most of our values are within the 305 to 945 millisecond range. The test also experienced much higher variability than the WiFi variant. As we can see from the boxplot in figure 5.15, we have three outliers above 1 second, which is less than the six obtained with WiFi. However, these outliers are far more outrageous, reaching up to 8 seconds. The high variability experienced could be a combination of network instability due to it being a lousy coverage area and paging delays on the station's behalf. We can answer question number four and say that in a "poor coverage 5G network," we expect a latency of about 578 milliseconds.

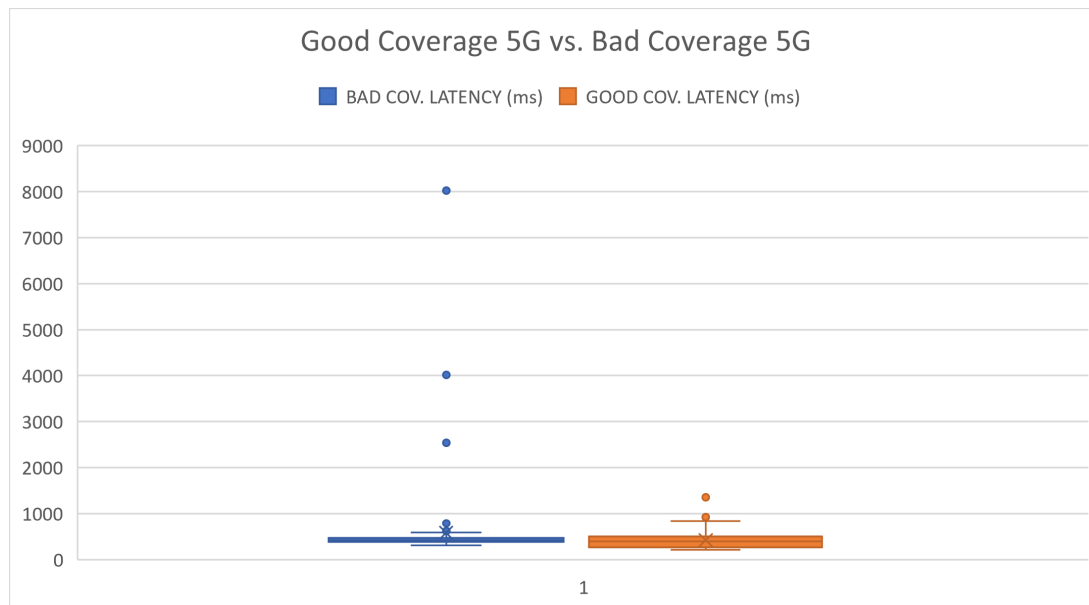


Figure 5.15: 5G Boxplots for Good and Bad Coverage

The following test was done in a good coverage scenario. The coverage conditions were achieved by bringing all of the tester's hardware to the grass fields outside building B. The tester set up their workbench to have both devices facing the experimental station directly, as shown in figure 5.16. The coverage obtained in this location was far superior to that of the laboratory since we had a direct line of sight. The network speed at the time was 201,2 Mbps download and 135,6 Mbps upload. With a sample size of 71 messages (21 of which were removed due to a Firebase error that caused a series of outlier values), we obtained a mean value of 421,8028 milliseconds and a sample deviation of 195,6643 milliseconds. Most of our values are within the 216 to 556 millisecond range. Our boxplot, shown in figure 5.15, surprisingly shows the least amount of outliers so far, only one of them being above 1 second and consistent with outliers obtained when testing with high-speed WiFi. We can answer question number five and say that in a "good coverage 5G network," we expect a latency of about 422 milliseconds.

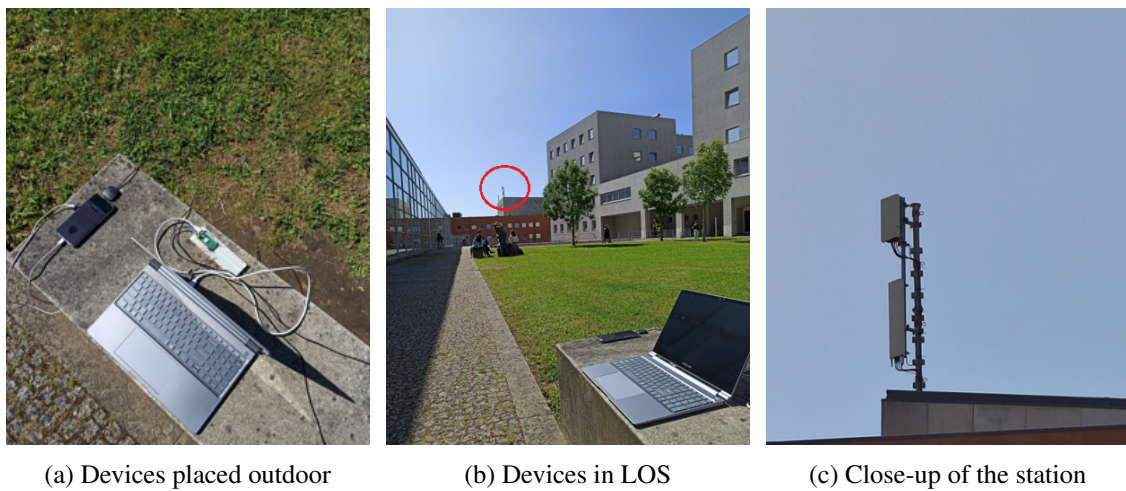


Figure 5.16: Outdoor testing setup

5.3.2.4 5G vs. WiFi

It is worth reiterating that our latency interval is influenced mainly by the overhead introduced by the Firebase server and not the network access, meaning we cannot accurately compare which of these technologies would fare best in this scenario. Judging exclusively from the *ping* values we showed previously, 5G seems to be the best candidate; however, as we can see from figure 5.17, we obtained worse values with it than we did in the WiFi scenario.

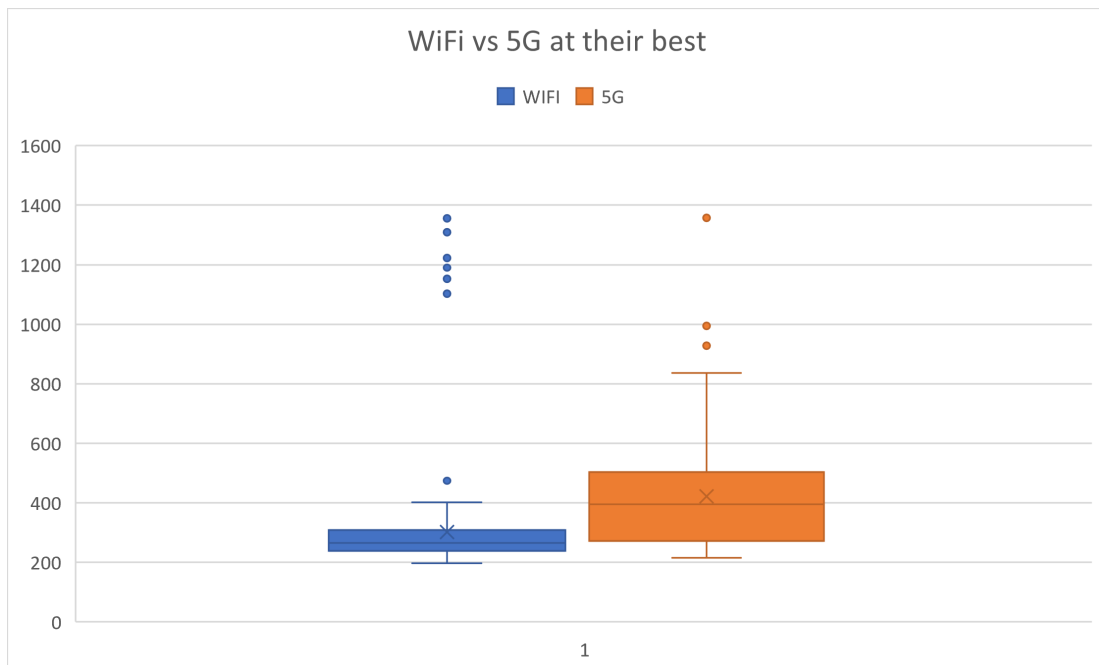


Figure 5.17: Comparing WiFi with 5G values

This discrepancy could be attributed to the fact that the Firebase server may have been experiencing much traffic when the 5G measurements were conducted. That assumption is further

accentuated by the fact that our values show a peak in latency after a given point in the experiment; this peak is consistent with console logs that report "connection refused" errors from Firebase.

In conclusion, we cannot give an accurate answer to question number six. We can only claim that we happened to have lower values when measuring with WiFi access, which is not to say that it is the superior technology.

5.3.2.5 Meeting the Target Delay

As we detailed in section [5.3.2.2](#), the best values we obtained had an average latency of about 302 milliseconds, meaning that, on average, they are above the target delay. We can answer question number seven and say that no, it is not probable that we can "meet a target delay of 300 milliseconds" with our current infrastructure.

5.3.2.6 Conclusion

We cannot conclude 5G's superiority over WiFi from the data collected. Additionally, our latency values can go above 300 milliseconds, even in a best-case scenario. We cannot meet the requirement echoed by Scholliers et al. and enunciated in the ETSI norm, indicating that our infrastructure is a sub-optimal representation of one of these systems.

It is worth highlighting that our latency is comprised of network access components and server overhead. The most significant cause of our exacerbated latency values is not related to the former but the latter. In the future, perhaps with highly optimized message queues, lower latency values could be guaranteed.

In conclusion, we cannot claim to have an accurate latency reading representative of a finalized VRU safety system for this study. Moreover, the Firebase Cloud Messaging Server's queues are not optimized for our use case since, as seen previously in section [5.3.2.1](#), the overhead introduced by the server is the leading cause of these high latency values. If possible, another implementation would have to mitigate that by having high-priority low-latency queues to guarantee the sub-300-millisecond target delay.

Chapter 6

Conclusions and Future Work

The research done in [2.4.4](#) shows that the field of warning VRU through applications running on their smartphones must be expanded upon, as it is usually plagued by interfaces that provoke an unpleasant user experience and emit inefficient warnings.

Our work attempted to develop a solution that can build upon those gaps. However, there will be room for improvement, as there is quite a lot of potential for further development in this field. This dissertation will not cover everything in its generously limited time.

6.1 Conclusion

The conclusion is that one may build a smartphone application that effectively warns a VRU through the appropriate Human-Computer Interaction and Software Engineering methods. URU-S is a verified, robust, and validated high-fidelity prototype of a road-safety system that includes the established use cases. It was built based on feedback obtained from our users, and it includes some end-system technology such as context-awareness and geofencing.

Our sample sizes for studies that required active human participation were small due to the current health situation. However, this fact is not too damning as the straightforward nature of the prototype guarantees its validation by five to ten participants, and our values were within that range. Moreover, we managed to do some additional measurements, where our results are indicative and specific of the infrastructure that we had.

We would also like to prioritize the requirements that we have found to be most important for a VRU safety system using smart devices. First and foremost, we must focus on achieving user-centrism, collaborating with our users to meet their needs with a usable interface that provides a good user experience. Secondly, low latency is vital as pre-crash collision detection is safety-critical, and each millisecond counts. Third, accuracy must be high; our users will grow desensitized with our warnings if they receive too many false positives, thus hurting the application's credibility. Fourth, battery optimization must be considered, as draining too much battery on the

device may frustrate our users and cause them to uninstall. Although we realize battery conservation may be a conflicting goal with high accuracy and low latency, we want to clarify that it is preferable to warn the user rightfully and timely than to preserve energy. Fifth, we have scalability, another crucial element to the application. We may have several users trying to access the system at once. Sixth, we have availability, which works in tandem with the previous requirement to assure our users will get their warnings no matter when or how many of them may need to. Seventh, we have security; evidently, the system should not leak sensitive data. Our more tech-savvy users may become aware of that fact and urge others to uninstall. Lastly, we have portability. Runners and avid cyclists would like to have this sort of solution in a smartwatch if possible. We must also investigate its integration with smart cars and other devices running Android.

In conclusion, if we are to reiterate our initial hypothesis:

“A Vulnerable Road User can be notified of danger effectively, through a smartphone application, if the appropriate software engineering and human-computer interaction methods are used to develop it.”

We may claim its veracity as we did develop a smartphone application, through the enunciated methods, that culminated in a high-fidelity prototype of an effective danger notification system that places the Vulnerable Road User first.

6.2 Further Work

Overall, the "Mobile app for protecting cyclists and pedestrians in road traffic" shows great potential. However, further investigation must be done. More investment must be placed into a robust back-end communication network that could guarantee low latency pre-crash collision detection at a large scale since the technology we have available would struggle to accomplish that. One may also invest in local computation, but we must be mindful of the energetic costs. Either way, further work would have to look into the more technical aspects of the application, including electronic communications through the most appropriate means available at the time. In this work, a service that relays the VRU position to a collision detection server is presumed to exist. A future project could investigate the development of such a service.

Additionally, a server that appropriately calculates collisions, based on information supplied by the phone and any nearby vehicles, could be explored since that was not developed in this dissertation. The collision detection algorithm itself would also merit its research study.

Furthermore, added usability testing could improve the interface, gathering data from several different VRU and forming a new prototype based on that; this would also increase the statistical validity of our findings by testing with even more participants. Moreover, work could be done with cyclists, perhaps with potential integration with the VCE, as most of the participants in this study were not avid cyclists. Portability could be investigated as well, perhaps redesigning for tablets, smartwatches, or other devices. There is also further potential to expand the usability testing to drivers and the solution that would run in a smart car.

One more study that we did not have the time for in this project would be perceptual/motor-stimuli testing. URU-S invokes the senses of the user through visual queues, haptic and acoustic signals. Perceiving and acting upon a stimulus is no easy task. It may draw knowledge from neuroscience, psychology, kinesiology, and other medical fields [74][75]. The test could include bringing the participant next to a road and stimulating them, measuring their reaction speed with different warning elements in alternative scenarios, either using electromyography or any other available method. It is important to note that studying the perception of a stimulus is different from its motor reaction. Stimulus perception is fundamentally psychological and can be studied with temporal-order judgments, simultaneity judgments, or even duration judgments. However, all of those methods have their associated biases and inconsistencies [75], and one would have to construct a controlled environment to deal with them accordingly. The physical component may be more relevant to URU-S, and the participant's motor reaction could be studied using simple reaction times [74] and studying which combination of warning elements would elicit the best nervous response. Regardless, at this point, we would be studying the human biomechanical system's adjustment to our application, and not URU-S itself, transcending the realm of informatics, suggesting this may be best left off for other researchers.

In conclusion, even though this field is not as well developed as it should be, there is an overwhelming possibility of constructing a C-ITS system that values the VRU as an active participant. URU-S will only scratch the surface, and its efforts may be continued.

Appendix A

Focus Group Report

The purpose of this appendix is to successfully communicate the findings of the Focus Group held on the 18th of February 2021, going through the methodology, demographics, and discussion results.

Prepared for: The interaction design process associated with the Unprotected Road User - Shield (URU-S).

Prepared by: Luís Alvela Duarte Mendes

A.1 Executive Summary

The initial hypothesis was that the Focus Group would ascertain what people would like to see in a road safety app such as this one. Primarily, what kinds of notifications they would like (that warn them either when they are in immediate danger or perhaps afterward with some form of a report feature).

The group found that participants mainly gravitated to a mix between a haptic (tactile) and acoustic (sound) notification. Two participants voiced their concern that the vibration had to be particularly intense, as even when carrying their phones in their pockets, they sometimes would not feel vibrations coming from text messages or calls. All participants showed interest in some form of offline reporting feature that would warn them of potentially dangerous areas beforehand. The whole discussion between real-time notifications and non-real-time reports raised fascinating concerns.

Despite this, all participants had a solid technical background and, as such, could not separate themselves from their urge to know more about the technical, back-end aspects of the application. That factor caused some confusion since the Focus Group did not go in-depth about those issues and was more concerned with the overall user experience.

A.2 Methodology & Participant Profile

A.2.1 Instrument Development

To develop the discussion guide, the researcher mainly considered what people would like in terms of notifications (sound, visual, touch), added functionalities, the ability to configure and interact with the notification system, the placing of UI elements, if those accessible, and discussion of alternative interface ideas. The end questions would be for discussing anything else the participants may want to bring up. Finally, the last portion was related to what each participant found to be the most crucial aspect discussed today.

A series of sound samples were played to complement the discussion guide so that the researcher could get a better understanding of what types of sounds users would like best. Additionally, polls were created to gather a census for each of the questions posed (note that some issues were not polled as the participants had uniformly voiced their opinions on the subject).

Even though some issues were not polled, the group's small size assured a common opinion in most scenarios. The small size is perhaps the most debilitating factor of the group. However, all the participants were gathered, taking into account the available contacts at the time and the current pandemic situation.

A.2.2 Site Selection

Due to the current pandemic situation, the Focus Group was held online via Zoom.

A.2.3 Participant Selection

Since this took place as part of a dissertation project, the researcher had limited participant recruiting resources. For the most part, they were academic contacts gathered via e-mailing other academics. The participants were offered no incentive (money, items, or otherwise) to participate.

A.2.4 Procedure for Current and Future Focus Groups

Focus Groups will be held with a minimum of 5 participants at dates scheduled, taking into account each participant's availability; they will last for 45 minutes up to perhaps an hour if the participants are available. All sessions will be held via Zoom and recorded (with the participant's consent); later, they will be documented and made available to the project supervisor.

A.2.5 Participant Profile

Most participants were avid researchers in the tech industry that consider themselves quite tech-savvy. Their routines varied, some taking walks and biking on occasion.

A.3 Demographics

The demographic data of each participant can be summarized as follows:

Participant	Age	Education	Profession	Tech-savvy	Routine
A	25	MSC	Researcher	4	Usually takes private transport; does leisure walking and biking.
B	25	MSC	Data scientist	4	Mainly private transport; sometimes jogs in the middle of the city.
C	31	MSC/PHD	PHD Student/Researcher	4	Mainly takes the car; sometimes jogs.
D	33	PHD	Researcher	4	Public transport; sometimes walks and takes leisure bike rides.
E	30	MSC/PHD	PHD Student/Researcher	4	Public transport and walks a lot.

The "tech-savvy" field is a measurement of how proficient these users are with technology. A scale from 1 to 5 was used to quantify technological aptitude, with "1" meaning very little proficiency and "5" being an expert level. A level 1 tech-savvy participant would struggle with just using a regular smartphone. A level 2 participant would be more capable of interacting with basic applications like FaceBook or WhatsApp. A level 3 participant can use a wide variety of applications but does not know the internal workings of the Android or iOS frameworks. A level 4 participant has inside knowledge of the operating system, knowing how to use it to configure and alter the behavior of their applications; they can use most applications with ease. A level 5 participant is a master, knowing the ins and outs of technology and using any application.

A.4 Discussion Results

This section includes a detailed analysis of the topics discussed and stimulated through a series of questions.

A.4.1 Question #1 - What would people like to see in a road safety app such as this?

Summary: The very first concern raised here was related to the application's inherent Context-Awareness (this is covered in section 4.3). One of the participants was particularly curious if the application would mainly work on the go or otherwise. This concern sparked a conversation later on where a different participant mentioned that even when the user is standing still, they could still be in danger. Thus, they voiced their interest in manually enabling the alert system even if they were standing still. After that, the participant further voiced their concerns with the "map" visual cue present in the prototype, as they thought that such a cue would cause the user to become tunnel-visioned. This topic had already been reassured by Professor Falko Dressler's research [20]. Other participants pitched in, and the conversation seemed to point out that a real-time visual cue for the danger notification did not seem ideal. However, this could work well for an "afterward" consultation. A "heat-map" of the most dangerous locations also seemed appealing. All participants thought a mix between vibrations and sounds was best suited for the real-time notification. The "heat-map" of the most dangerous locations was validated as a non-real-time component to the application. A participant managed it would be interesting to integrate the application's notification system with a smartwatch (used while running), calling to the application's portability. Also, the best times or path recommendations to jog were mentioned as an alternative feature. Nevertheless, another concern was the application being spammed by a series of false positives; mainly, the application should be designed to prevent this. At this moment, some of the more research-heavy participants started to voice some entropy surrounding the fact that they wanted to dive deeper into some of the technical questions.

A.4.2 Question #2 - What would people like in terms of notifications? Sound, Touch, Visual?

Summary: The participants were polled on this matter. Such may be consulted in the "Appendices" section. Most of them agreed that a mix between vibrations and sound was best suited for the real-time notification. At this point, one participant mentioned that they often did not feel their phone's vibration. Integrating with a smartwatch was mentioned here, but everyone settled on a mix between vibrations and haptics.

A.4.3 Question #3 - If you would like to have sounds, which sounds?

Summary: The participants were polled on this matter, such may be consulted in the "Appendices" section, and most of them agreed that the best possible sound was an "announcer" voice

saying "warning" (a sample is provided in the "Appendices"). The participants voiced their concerns yet again in distinguishing this sound, or sound plus vibration pairing, from other notifications in their device (e.g., text messages, calls); the "announcer" stood out as a unique sound.

A.4.4 Question #4 - If visual, in what way?

Summary: This question was more or less aborted by the group moderator, as all participants made themselves clear that a visual cue was not the best here.

A.4.5 Question #5 - Do you like the ability to configure/suspend the notification settings?

Summary: All participants agreed that configuring and being capable of suspending notifications was a good idea, mainly since they had different preferences. It was at this point that a participant voiced their concerns for a possible misunderstanding regarding the interface. They were concerned that the button to suspend notifications was perhaps a bit misleading. What they meant by this was: the switch to disable "notifications" made it seem like they were shutting down the entire program, whereas it could also mean "disable the push-notifications only" (as in, the pop-up message), so, in other words, they did not quite understand if it was meant to shut down the visual alert message that pops up or the entire alerting system as a whole. This misunderstanding could be remedied by splitting that button into two others, one that says to disable the push notifications specifically and another more aptly titled "Disable all" to suspend the entire system.

A.4.6 Question #6 - Do you like the ability to suspend for a given amount of minutes?

Summary: All participants agreed on the relevance of suspending the application for a given amount of minutes or until manual re-enable. At this point, a participant mentioned that they would like it if the application could automatically shut down indoors. However, another participant raised the concern that while indoors would like to shut down the real-time notifications, they would still like to keep the non-real-time reporting features active. They stressed that it should also shut down the reports, but these should be independent.

A.4.6.1 Question #7 - Do you like the positioning of the notifications on-and-off switch? Would you prefer it on the main page, or should it be kept in the settings?

Summary: The participants were polled on this, and the results may be found in the "Appendices" section [A.7](#). One participant mentioned it would be more accessible to have the "disable all" button on the home screen. However, another participant mentioned they would like it in both places (where it is now and on the home screen, but that it would be more detailed in the former than the latter). The poll alleviated some confirmation bias, and most people thought that having it available in both places would be ideal.

A.4.7 Question #8 - Would they like some form of widget integration with a PSP (police) map of the most dangerous locations in terms of accidents?

Summary: Participants were, once more, polled on this. Everyone across the board agreed that this was an exciting integration that could be made, specifically since they were unaware that something like this existed and manifested interested in seeing it in the application.

A.4.8 Question #9 - Would you like to be capable of instructing the app's notifications to shut down in certain areas that you know for a fact are prone to false positives?

Summary: A poll was made here as well. Participants were very divided on this, as half the group thought it could be interesting, as they would pass by those same places every day and they know for a fact it will not work all that well there, but others thought it was of no value.

A.4.9 Question #10 - Are you interested in the login/register option for saving your settings, or would you like some other form, like the upload/download or even some other form (please specify)?

Summary: Most participants thought that they would be using the same device for quite a while and that they would not swap them frequently enough to warrant any of these functionalities. They thought that they could configure the settings on their own. Another participant voiced their concerns about being able to access the reports independently of logging in. Some other participants mentioned that it would not be necessary to have this feature unless the application had a learning mechanism. The settings seemed easy enough to reconfigure, so no one truly manifested a desire for a porting feature.

A.4.10 Question #11 - Do you think the documentation is accessible where it is now, via the options menu?

Summary: This was a resounding yes from all participants, and they viewed it as a non-issue. A participant voiced their concerns regarding the group's small sample size (after all, we started with 5 participants only, which was reduced to 4 more to the end of the session). That should be taken into consideration for the data analysis. They also did not understand that the notification poll allowed them to pick more than one option (sound, visual, tactic) and stressed that having vibrations combined with sound was their choice.

A.4.11 Question #12 - What do you think was the most vital thing discussed today?

Summary: One of the users confessed that this was their first focus group, and as such, they were a bit new to its dynamics. They felt that the design's technical implications should have been discussed more, suggesting it was somewhat difficult to think like an end-user in this scenario.

However, they stressed that the most critical factor was distinguishing between real-time and afterward notifications and which types of real-time notifications to emit (sound, visual, tactile). Another echoed the importance of how the notifications are displayed and other relevant features (such as the PSP map). Additionally, they suggested that the app be called something else, as they thought it could be confused with other ciphers that should not be associated with it. A different participant praised the idea of bringing a prototype to the discussion and that they wished we spent more time discussing the functionalities in a somewhat abstract form. They also reinforced their interest in a planning element to the application and that the UX concerns were perfectly valid. The final participant assured that real-time versus non-real-time features were the most relevant discussion and that they would like to see more of a planning/reporting element to the application.

A.5 Conclusion

Overall, the group seemed to have difficulty detaching themselves from the more technical aspects of the application, but this did not impair the discussion process. From the various topics discussed, such as the types of notifications participants enjoyed and the inclusion of non-real-time elements, one can conclude that participants preferred a mixture of haptic and acoustic signals to warn them in real-time. Additionally, they valued having a non-real-time aspect to the application through danger reports or maps of accident-prone areas.

As far as the user interface goes, barring the slight misunderstanding with the settings interface, pertaining to the "Notifications" switch being confusing, participants found that what they needed was accessible. Although some thought having the kill switch on the home page could be advantageous, this last change did not seem particularly exciting.

A.6 Recommendations

It is recommended to conduct another group (if possible) with people more distant from the tech side of things to hone in on the human-computer interaction elements.

Aside from that, the topic of having the application available on other devices (such as smart-watches) came up more than once, and, although the dissertation focuses exclusively on smart-phones, this is worth mentioning as an example of further work to be done as there is a substantial amount of fascination associated.

Another aspect to consider is that the interface should be altered, in the settings menu, to differentiate between the suspension of the visual pop-up or the entire alert system as a whole. Making the switch available on the home page may be looked into, although that did not garner much value from the group.

Lastly, looking into a non-real-time component is advised. This component should be independent of the real-time notifications and may consist of missed alerts, danger-level, or even a heat-map of the most dangerous places. The recommendation of "safe paths" was also mentioned;

however, this deviates from the application's scope. The functionality of "instructing" the application to suspend notifications in given areas was quite divisive, so, considering the added layer of false positives resulting from this, perhaps it would be best not to pursue it.

A.7 Appendices

The appendices include all assets used and data gathered in the focus group:

A.7.1 Privacy Policy

The privacy policy ensures that the participant's rights are respected and that their data will never be traced personally. All participants received a copy of this 40 minutes before the session began.

Even though the policy explicitly contains the right to film sessions, the moderator asked all participants for their consent to film the session previously, and they allowed it. The recording itself will not be shared in the URU-S repository and will remain in the researcher's computer unless mandated by a higher authority.

This document was not in its final form at the time the Focus Group was conducted. However, it was edited and re-purposed for the usability testing phase. The final version is in appendix C, whereas the ones used in the Focus Group are stored in GitHub ([portuguese version](#) and [english version](#)).

A.7.2 Presentation Used

The presentation helped support the moderator in showing the prototype and explaining some of the study's context, objectives, and motivation. It is currently stored in [GitHub](#).

A.7.3 Question Guide

The question guide helped the moderator stimulate discussion during the group. However, some questions were not asked explicitly (or verbatim) because the participants had already answered, or the conversation had lead to slight modifications that were better contextualized. It is also stored in [GitHub](#).

A.7.4 Sketch of an interface misunderstanding

One of the participants mentioned that the "Notifications" switch was not clear in what it meant. Figure A.1 represents the sketch that and was used so that the moderator could better comprehend their problem. Essentially, the participant was confused by the disabling switch, not sure if it was only disabling the pop-up of the alert or the alert as a whole (no sound, vibration, or pop-up). The moderator opened up Microsoft Paint and tried to reach an understanding with the participant.

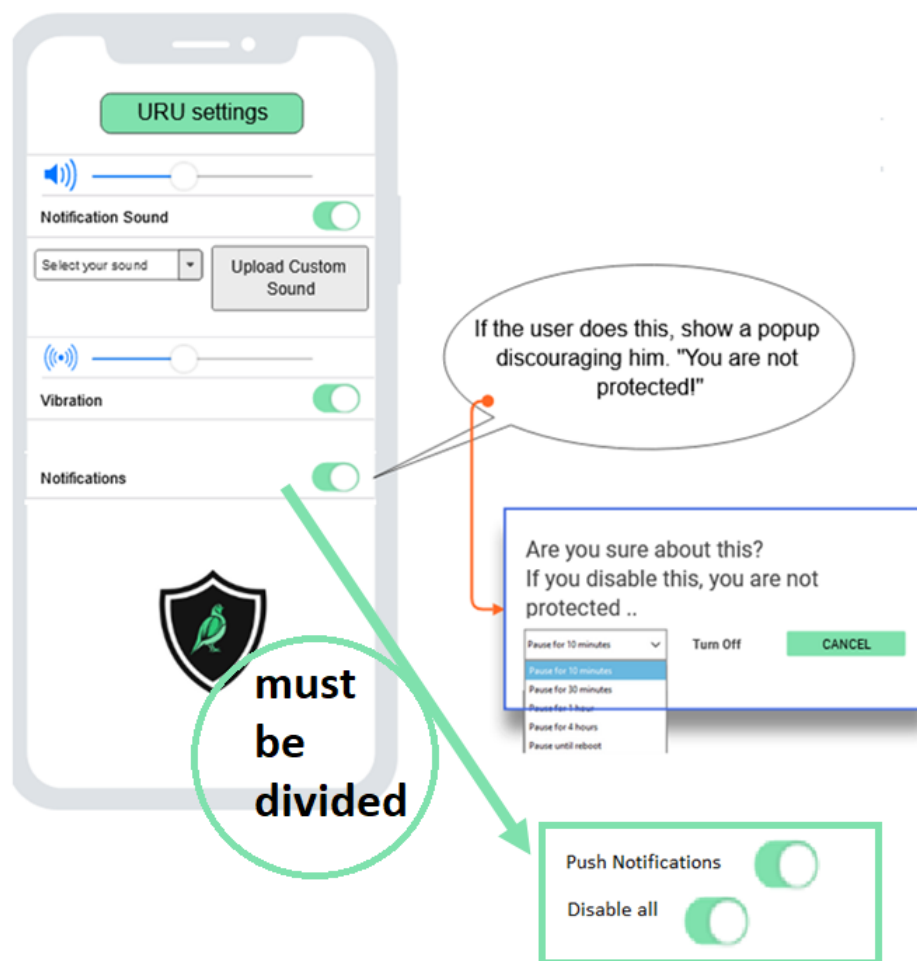


Figure A.1: Interface Misunderstanding Scheme

A.7.5 Polls

A series of Strawpolls were used to alleviate confirmation bias. Like in any focus group, a more assertive participant voiced their opinions more intensively than the others. For issues where the moderator was not entirely clear on the group's consensus, polls were used where participants could vote anonymously. These are the results of those polls.

A.7.5.1 Notification Types

The following is a visual representation of the poll itself and the results.

URU-S: Notification type
por Luís Mendes · há 5 dias · 👤

”
What type of notification do you prefer?

Escolha uma ou mais respostas:

- ☐ Visual
- ☐ Touch (Haptic)
- ☐ Sound (Acoustic)

Figure A.2: Notification Types Poll

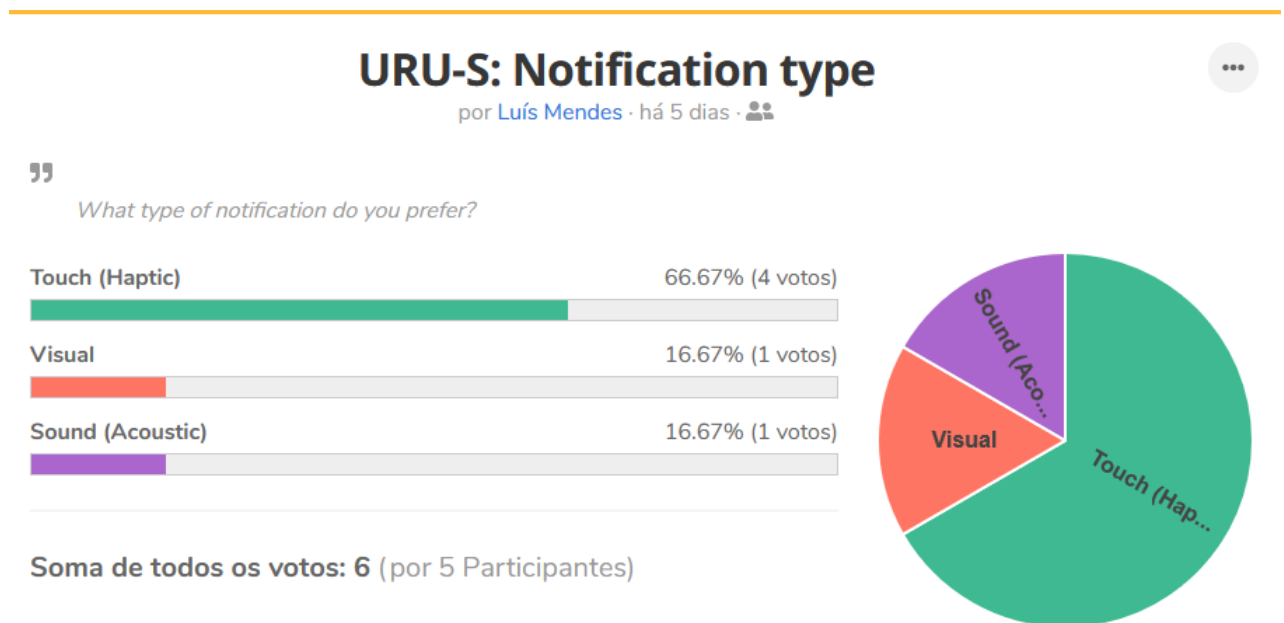


Figure A.3: Notification Types Results

It is to note here that some participants thought this was multiple choice, which is why "sound" is not higher. However, everyone voiced their opinion and made it clear that they would like to have a sound following the vibration.

A.7.5.2 Sounds

Since the participants clarified they were interested in having the notification accompanied by a sound, there was a poll where participants were given samples of sounds to listen to and pick the type they liked most.

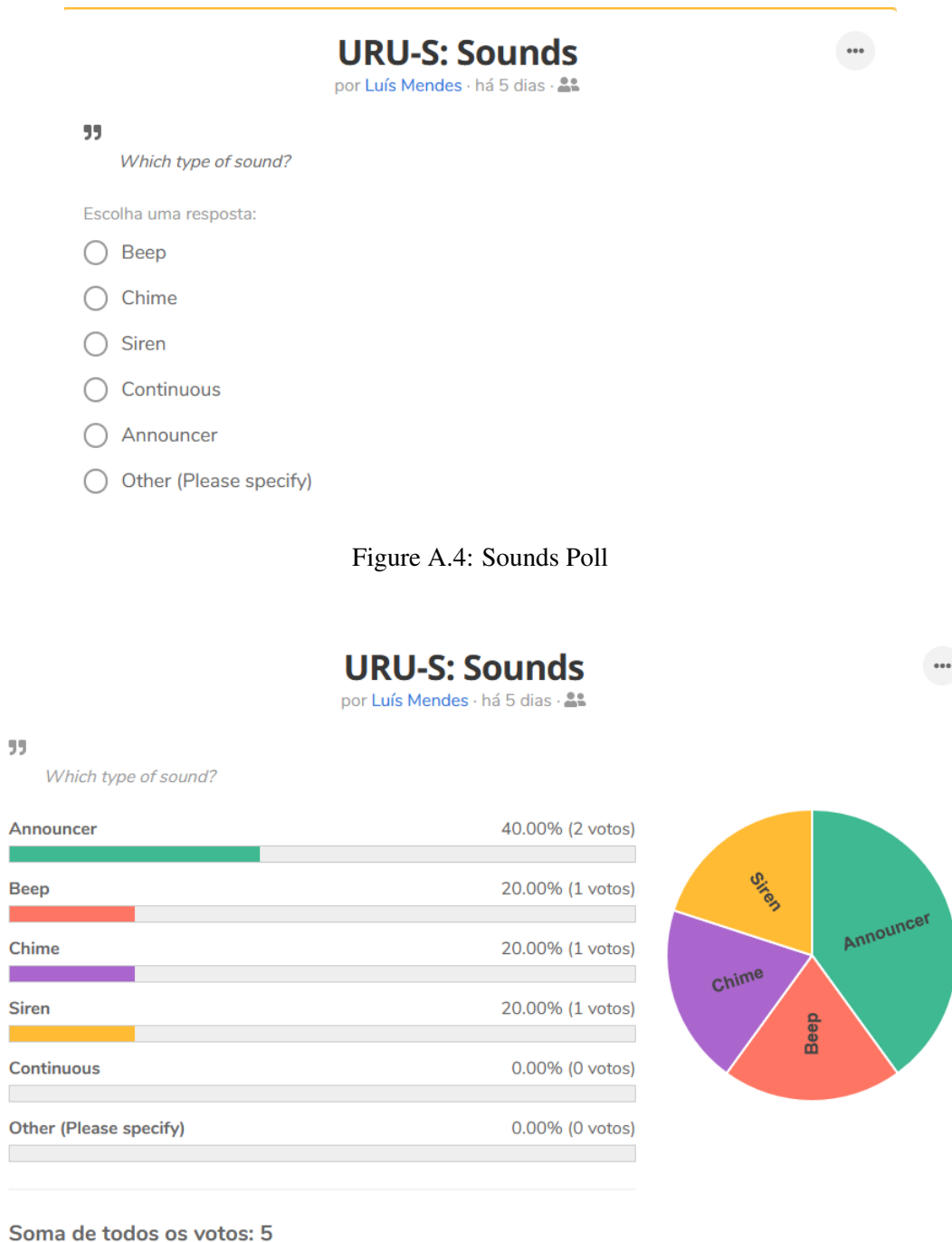


Figure A.5: Sounds Results

The samples used are the following are courtesy of [Floyd Bell](#), and are available in [GitHub](#).

A.7.5.3 Placing of the On-and-Off-Switch

Placing the on-off switch somewhere else

por [Luís Mendes](#) · há 2 dias · 

”

Do you like the current placing, or should it be somewhere else?

Escolha uma resposta:

- ☐ It should be in the main page.
- ☐ It is fine in the settings page.
- ☐ It should be in both places.
- ☐ Other (please specify)

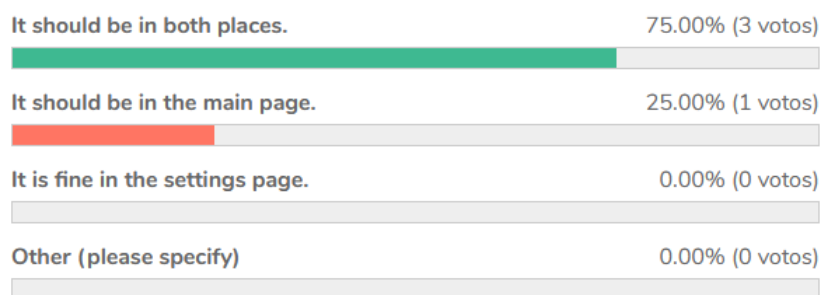
Figure A.6: On-and-Off-Switch Poll

Placing the on-off switch somewhere else

por [Luís Mendes](#) · há 2 dias · 

”

Do you like the current placing, or should it be somewhere else?



Soma de todos os votos: 4

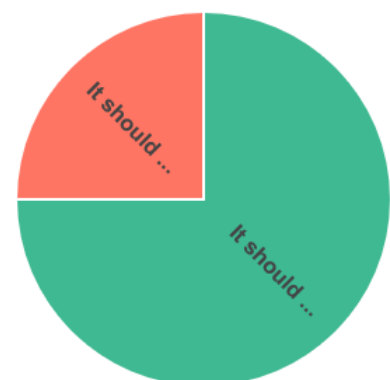


Figure A.7: On-and-off-Switch Results

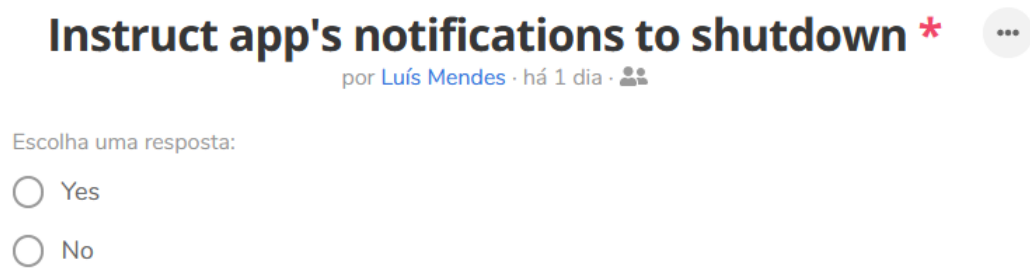
A.7.5.4 Ability to Instruct the Application to Shutdown in Certain Areas

Figure A.8: Instructed-Notification-Shutdown Poll

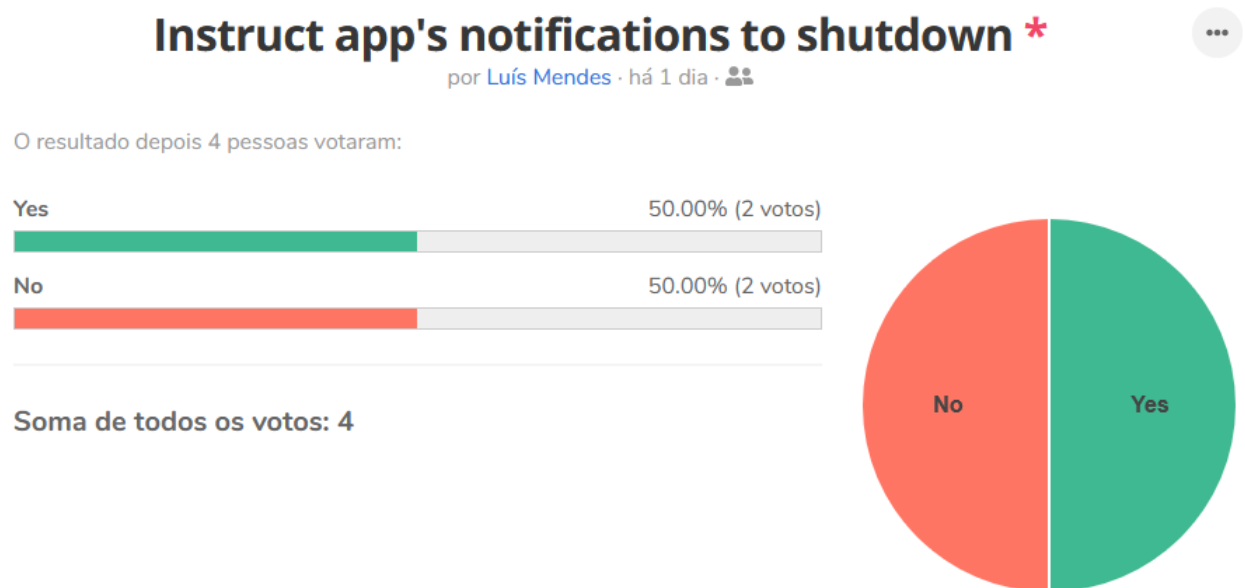


Figure A.9: Instructed-Notification-Shutdown Results

A.7.6 Interest in Danger Reports or Crowd-Sourcing Elements

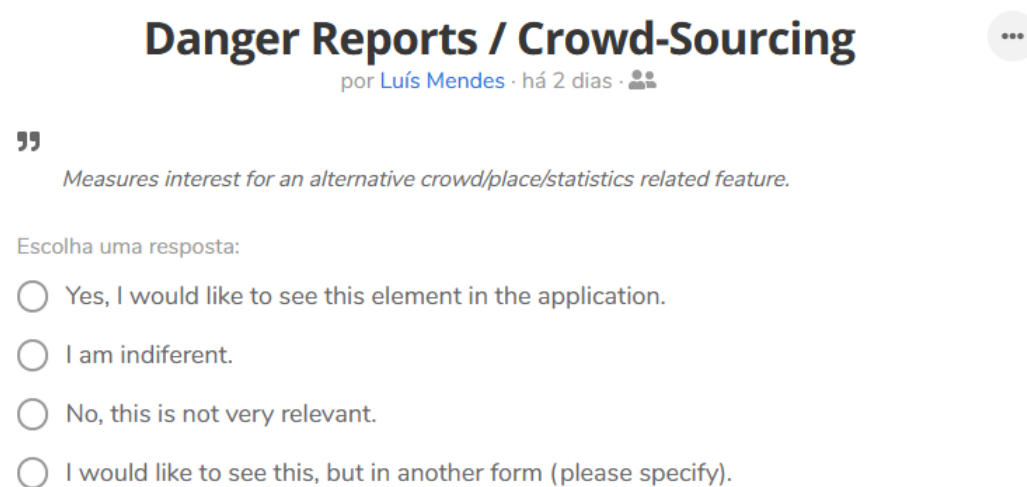


Figure A.10: Danger Reports Poll

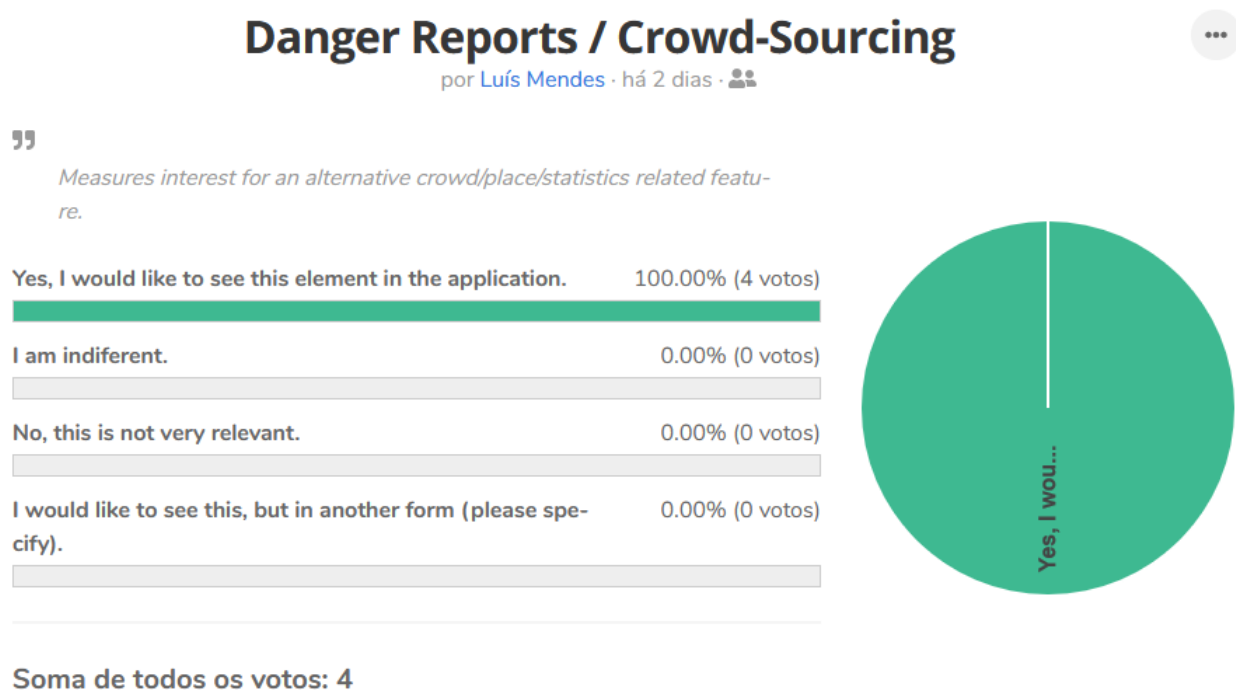


Figure A.11: Danger Reports Results

Appendix B

Usability Testing Report

The purpose of this appendix is to communicate the findings of the Usability Testing phase successfully.

B.1 Introduction

URU-S is the Unprotected Road User Shield. It serves as a high-fidelity prototype of a road safety system concerned with actively integrating vulnerable road users into the interconnected vehicle world. Despite this being outside of the project's scope, the end-system would be capable of predicting dangerous collisions before they occurred, warning the user appropriately.

This project is a part of the dissertation formally known as "Mobile app for protecting cyclists and pedestrians in road traffic." The student assigned to the dissertation conducted an onsite usability test using a live version of URU-S located on testing devices available at the Telecommunications Institute. The testing devices had the screen-recorder enabled, capturing input from the microphone and recording the participant's conversation with the testing facilitator. Additionally, the screen-recorders were capable of keeping track of where the participant was clicking. The test had two phases, indoor and outdoor; the facilitator was with the participants throughout both phases. Participants were also recorded with a webcam during the outdoor phase. The session captured the participant's difficulty completing tasks, the time required to complete said tasks, the task completion rates, questions, and feedback.

B.2 Executive Summary

As mentioned above, the test had two phases. The first phase was an onsite usability test at the Telecommunications Institute Shannon Laboratory, room number I322, at the Faculty of Engineering and University of Porto. The second phase was done around campus and consisted of walking around the Department of Informatics Engineering outskirts, eventually reaching a ramp situated in the parking lot between Department "I" and Department "L." The testing took place on May 6th,

May 18th, May 20th, and May 22nd, 2021. The test's purpose was to assess the interface's usability, the user's understanding of the underlying mechanisms, and the warning signal's effectiveness when the user is in imminent danger.

A total of eight participants were amassed for the usability test. Usually, to ensure the stability of the results, a total of eight to ten participants is ideal. We had the minimum amount, as it was unmanageable to arrange more participants due to the current COVID-19 Pandemic. Each individual session lasted approximately half an hour.

In general, participants found that URU-S was an aesthetically pleasing, straightforward, and practical application. However, 50% of them had difficulties understanding the context-awareness mechanism. Despite this, all of them understood the general concept and were capable of completing the tasks given by the testing facilitator. Furthermore, 88% of the participants used mobile applications frequently, and 38% were not very tech-savvy.

The test identified some minor problems that the lead developer quickly patched, including:

- "Context-awareness" was a very technical term that was not quite clear to the participants in its meaning.
- The Geofencing service was not very clear to some participants in what it did.
- The heat-map in the Statistics section had an overwhelming zoom feature that increased the radius of the heat bubbles too much and confused the participants.
- The dropdown menus lacked affordance.
- The Statistics section was hard to scroll through due to the map taking up too much of the screen.
- The sound selector in the Alert Settings section should play a sample of the selected sound immediately and not only when the volume is adjusted.
- The "my location" button was missing in the Statistics section map.
- The Statistics section used to be called the "Reports" section, which confused users as they associated it with reporting a bug.
- "Suspended" alerts confused some participants, causing them to be removed from the app later on.
- The introduction paragraph in the Home screen should be more explicit in what kinds of dangers URU-S prevents.
- Deleting the records of received alerts should only delete those selected in the date range shown in the dropdown.

This document contains the participant feedback, task completion rates, difficulty ratings, time on task, errors, and recommendations for future improvements.

B.3 Methodology

The student assigned to the dissertation played the role of testing administrator, facilitator, lead developer and was responsible for gathering the participants. The facilitator tried to contact as many participants as possible, reaching out to fellow students, personnel at the Shannon laboratory, and others. The invitations were sent via e-mail, WhatsApp, and Slack, informing the participants of the test logistics and requesting their availability and participation. Participants responded with a date and time that was more convenient for them. Each individual session lasted approximately thirty minutes. During the session, the facilitator explained the test and mentioned that it would have two phases. Before the test began, participants were assured that their data would be protected at all costs and were asked to sign an informed consent form. With the paperwork out of the way, the facilitator proceeded to ask the participants a series of demographic questions. Those inquiries included age, education, profession, and level of tech-savvy. Participants were also inquired about their daily routines, whether they used mobile apps frequently, and their favorite apps. Once the facilitator was done with the questions, phase 1 of the test commenced. First, they showed the intro screen to the participant and asked them to do a narrative of what they saw without touching anything yet. Participants were encouraged to include details such as what kind of application they were looking at and what they could do. After the narrative, the participants were given a series of tasks for them to accomplish with URU-S.

After each task, the facilitator asked the participant to rate the difficulties they had in completing the said task on a scale of 1 to 5. The former means the task was straightforward, and the latter that it was complicated to accomplish. The facilitator also noted whether or not the participant managed to complete the task successfully and how long they took to complete it.

Throughout phase 1, participants were encouraged to "think out loud" and say what they were trying to do, how they could do it, and if they were having any difficulties accomplishing their goal. At any point, participants were free to request a break or suggest improvements for the prototype.

Once the participant had completed all tasks, the facilitator would ask to be accompanied on a short trip outside. The facilitator and the participant would then exit I322, descend the stairs, and exit the department; this marks the beginning of phase 2. As they walked outside, the facilitator informed the participant that the application should experience different behavior outdoors. Once the application had switched its state, participants were motivated to figure out what it was doing and why. Finally, upon reaching the ramp between departments, the facilitator would ask the participants to wait at the top of the ramp while setting up the recording software and "simulated-danger" equipment on the other side of the testing area. After a short time, the facilitator would prompt the participant to descend the ramp. As soon as they were on camera, a warning would be emitted in their devices, and a simulated danger would arise.

B.3.1 Participants

A total of eight different people participated in the usability tests held throughout May. Three participants were involved on May 6th, one participant on the 18th, two on the 20th, and another two on the 22nd.

Most of these participants were tech-savvy academics. However, three of them had some more troubles with technology, and one of them even claimed they did not use mobile applications frequently. The age range here was also broader than what was achieved in the previously held focus group. Participants now had ages ranging from their early twenties to early forties and came from different backgrounds, holding different perspectives.

Their profile can be summarized in the following table:

Table B.1: Participant Demographics

ID	Age	Education	Profession	Tech-savvy	Routine	App-User	Favourites
0	32	MSC	Software Developer	5	Frequently goes on walks, public and private transport	Y	AirBnB, Google Keep
1	22	BSC	Student	4	Frequently walks and uses public transport	Y	Facebook, Instagram, Youtube
2	25	BSC	Student	4	Mainly private transport	Y	NOSTV, FlashScore, MSN
3	25	Highschool	Student	5	Frequently walks and occasionally private transport	Y	WhatsApp, Youtube, Twitter
4	36	Highschool	Research Assistant	2	Mainly private transport	Y	Instagram, Facebook, Twitter
5	41	MSC	Researcher	3	Mainly private transport and walks on occasion	N	Weather
6	40	BSC	Manager	2	Mainly private transport	Y	Pinterest, Outlook, Facebook
7	25	MSC	Researcher	5	Mainly private transport	Y	Spotify, Linguist, Maps, WhatsApp

B.4 Results

The testing facilitator recorded the participant's interactions with the testing devices by activating a screen recorder on said devices at the beginning of each session. The screen recorder also captured the conversation the participants had with the facilitator and tracked their clicking. Additionally, the facilitator started a timer after reading out each task to the participant and stopped that timer when they had completed the task at hand. Furthermore, participants were asked how difficult they found each task on a scale of one to five, where one means extremely easy, and five is very difficult.

The tasks given to the participants during phase 1 are as follows:

- **Task 1:** Change the sound, adjust the volume, and change the vibrations of your danger alert until they are to your liking.
- **Task 2:** Change the application's language.
- **Task 3:** Try to alter the current behavior of the alert system.
- **Task 4:** Try and find out what your "automatic alert regulation mode" is doing, and what "context-awareness" is and what it does for you.
- **Task 5:** Try to enable context-awareness.
- **Task 6:** Try to fully disable the alert system, no context-awareness, no alerts, nothing.
- **Task 7:** Try to restore the alert system to the way it was at the beginning of the test.
- **Task 8:** Try to find out how many alerts you have been receiving.
- **Task 9:** Try to find out if you are near a dangerous area and if you can receive some form of warning when that happens.
- **Task 10:** Imagine you are walking down the street, and this happens *a warning is simulated on the device*. Imagine that after that happens, a car zooms past. What can be done upon seeing this new notification that showed up on the device?

B.4.1 Task Completion Success Rate

The task completion success rate is the number of participants that managed to complete the task without needing any form of hint from the facilitator divided by the total number of participants that attempted that task. All of the participants managed to complete every single task. However, some of them need hints prompted by the facilitator. In cases where that happened, we will label that task with an "N." For cases where the participant managed to realize everything independently; we labeled that with a "Y." Participants and tasks are identified by their respective numbers, with "T1" being task 1 and so forth.

Table B.2: Completion Rate per Task for each Participant

Participant	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3	Y	Y	N	N	Y	Y	Y	Y	Y	Y
4	Y	Y	Y	N	Y	Y	Y	N	Y	Y
5	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
6	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
7	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Completion Rate	100%	100%	87,5%	50%	100%	100%	100%	87,5%	100%	100%

B.4.2 Task Ratings

After completing each task, participants rated the difficulty of completing that task on a scale of 1 to 5. It is worth noting that this rating is subjective, and some participants were uncertain on an integer number for their difficulty. Despite this, no one found any particular task of difficulty level 5 (extremely difficult). The following table shows the average difficulty for each task. It is to note that participants 3-7 had an easier time, generally speaking, than participants 0-2. The reason for this is, participants 3-7 already experienced a version of the prototype with patches based on the feedback given by the first three participants.

Table B.3: Difficulty on Task for each Participant

Participant	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
0	3.5	1	4	3	1	2	1	3	2.5	3
1	2	1	1	3.5	1	2	1	2	3	2
2	2	1	1.5	2	1	1	1	3	1.5	1
3	1	1	2	2	1	1	1	3.5	1	1
4	1	1	1	1	1	1	1	3	1	1
5	1	1	1	3	1	1	1	2	1	1
6	1	1	1	2	1	1	1	4	1	1
7	1	1	2	3	1	3	1	1	1	1
Average Difficulty	1.286	1	1.714	2.286	1	1.5	1	2.571	1.333	1.375

Participants 0-3 had some difficulties configuring their alert settings due to the fact they were expecting sounds to play right away when selecting different sounds from the dropdown, which, at the time, was not very affordable. The sound selector and all other dropdowns were fixed, and participants 3-7 exposed to the new prototype had no difficulty. It is worth mentioning that Participant 0 experienced more difficulty because they experimented with the real-time alerts switch first, instead of going to the options menu. Participant 0 demonstrated anomalous behavior, as the real-time alerts switch is not meant to be used for this task, and all other users went to the options menu directly. Nevertheless, task 1 is reasonably intuitive.

All participants managed to change the language with ease. Altering the behavior of the alert system was also easier for participants 3-7, as they were no longer presented with the term "context-awareness" right away. That had been replaced with "automatic alert regulation mode." Furthermore, participants 3-7 had a quick and easy way to know what the "automatic mode" was and what it did for them by clicking on its notification. Participants 0-3, however, had the unpatched version of the prototype and thus struggled more with this task, mainly since the term "context-awareness" confused them. Despite this, even with the added efforts to explain "context-awareness" better, some users still had troubles figuring out where they could go to know what this was and how it fit into the app.

Figuring out how to turn systems on and off was simple across the board. However, participants had some difficulties finding out how many alerts they had been receiving and which percentage of them were false alarms, genuine, or suspended alerts. This difficulty was not so much due to navigation (except for participant 4) but due to them having trouble understanding what a "false alarm," "genuine," or "suspended" alert meant. Most participants found the danger heat-map very simple and accessible. It is worth noting that participants 0-3 had more difficulty with task 9 because the prototype lacked a "my location" button on the heat-map and the zoom feature caused the heat bubbles to expand too much. The issues causing difficulties with task 9 were patched; thus, participants 0-7 had no trouble with the task. Lastly, practically all participants found the danger notification intuitive, except participants 0 and 1. Participant 0 preferred an "in-app" method of responding to the push notification, and Participant 1 did not appreciate the terminology on one of the notification action buttons.

B.4.3 Time on Task

The time on task (in seconds) was measured by the testing facilitator, and it clearly shows that some tasks were more difficult than others. The most difficult tasks were tasks 8 and 4. It took the participants a little over a minute to figure out what "context-awareness" meant and decipher the percentage of alerts received, respectively. Task 9 was significantly easy for most participants but took longer because it relied on a map, and users usually navigated and explored the map for a couple of seconds.

Table B.4: Time on Task for each Participant

Participant	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
0	47	8	30	77	10	40	5	93	150	150
1	61	8	5	49	5	21	8	115	39	24
2	45	6	45	60	5	30	5	8	37	102
3	61	5	10	30	5	14	15	66	107	25
4	50	5	62	130	5	46	12	161	82	20
5	89	5	52	46	5	21	6	14	70	44
6	76	14	29	154	5	31	5	48	30	20
7	82	10	21	40	5	67	9	23	49	32
Average Time on Task	63,875	7,625	31,75	73,25	5,625	33,75	8,125	66	70,5	52,125

B.4.4 Errors

The participants did not commit any errors that hindered them from completing the tasks; it simply took them a while to understand certain aspects of the task. Participants did not stumble into wrong portions of the interface often or tried to do things that had nothing to do with the task at hand.

Regardless, there are two cases of anomalous behavior. Participant 0 tried to reach for the alerts switch when told to try different sounds and vibrations instead of going for the options menu. Participant 4 had trouble finding the Statistics section because they had forgotten that the main page also allowed them to go there.

B.4.5 Outdoor Phase

The outdoor phase does not have a series of structured tasks, mainly because the participant was on the move with the facilitator, causing difficulty performing measurements. However, the participants all quickly understood that the application detected they were outdoors and enabled geofencing. They could also connect the geofencing service to the simulated dangerous area later approached during this phase.

Judging from the recordings, the test at the ramp yielded promising results, as no participant ever bumped into the danger thrown at them (a remote-controlled car) in the limited visibility intersection that had been simulated. Upon receiving the warning, the participants heard the sound and felt the vibration, stopping and becoming more aware of the encompassing danger.

B.4.6 Summary of Data

Table B.5: Task Summary

Task	Completion	Difficulty	Time
T1	100%	1,285714	63,9
T2	100%	1	7,63
T3	87.5%	1,714286	31,8
T4	50%	2,285714	73,3
T5	100%	1	5,63
T6	100%	1,5	33,8
T7	100%	1	8,13
T8	87.5%	2,571429	66
T9	100%	1,333333	70,5
T10	100%	1,375	52,1

Table B.5 is the summary of all data. As we can see, the main problem areas are related to tasks 4 and 8. Participants had difficulty figuring out how context-awareness fit into the application and interpreting the "danger level" graphs that showed them how many alerts they received and what each percentage meant. Overall, participants did not struggle too much with URU-S and could grasp its general concepts and inner mechanisms. During the outdoor phase, they also understood the geofencing service and its integration with warning them of nearby dangerous areas.

B.4.7 Likes, Dislikes, Participant Suggestions

Participants would comment on the application throughout the tasks and state what they did and did not appreciate, offering suggestions.

B.4.7.1 Likes

Most participants enjoyed the application's aesthetic and were capable of comprehending its interface elements. There were no complaints regarding color contrasts or hard-to-read text. Once participants understood what context-awareness was and what it did for them, they greatly appreciated it and found it interesting. The same can be said about the geofencing service and the ability to know when they are near dangerous areas. Great appreciation was shown for the manual and all of the documentation surrounding the application to help if need be.

B.4.7.2 Dislikes

Two participants did not appreciate the reliance on push notifications, as they had personal gripes with that mechanism. Other than that, every participant found it somewhat complicated to understand the integration between context-awareness and the alert system. These doubts were cleared once they had read the help documentation, but they would rather not have to read the text to understand the functionality.

B.4.8 Participant Suggestions for Improvement

Participant suggestions may be summarized as follows:

- The introduction paragraph to the application should be more explicit about the types of dangers URU-S protects people from (i.e., collisions with vehicles, cyclists, and pedestrians).
- There could be a single switch to disable the entire application instead of two separate switches.
- One could be capable of adjusting vibration intensity and duration separately.
- The application could have an "in-app" notification answering system instead of using push-notification action buttons.
- The "suspended" alerts in the Statistics section are not useful and should be removed.
- The zoom-out feature of the heat-map should be different, as it makes the map unreadable.
- It should be more explicit what context-awareness and geofencing do.
- "Ignore" is not a very good term for the push-notification button associated with the danger alert.

- "Reports" is not a good name for the Statistics section.
- Dropdowns should look more like dropdowns, having a little down-facing arrow next to a box.
- Sound selectors should play a sample of the selected sound immediately.
- The heat-map should have the "my location" button on it.
- The heat-map should be a little smaller so that one may scroll better through the Statistics section.
- The date selector for the alerts range could be divided into further periods, representing "trips" taken by the user.
- Deleting alerts from a certain date range should be specific to the selected date range.

B.5 Recommendations

It is recommended to keep all participant suggested improvements in the application, except for the following:

- There is no need to have a single switch to turn off the entire system. This change was suggested by a single participant that did not want to shut down alerts and the automatic mode through two different switches, but no one else had any issues with flipping the two switches. Additionally, it may add increased complexity to the code and the interface, particularly since it already has enough switches.
- Adjusting vibration intensity and duration with two separate sliders is meaningless. The Android OS does not allow one to set a wide range of amplitude values for vibrations. To distinguish a vibration from another, we have to use a single slider that alters duration in tandem with intensity; otherwise, the difference would be imperceptible.
- Only a single participant suggested changing the date range on alerts received to narrow it down to "trips" taken by the user. This alteration would involve detecting a "trip" event and automatically logging it through advanced context-awareness. Due to the complexity associated with this suggestion, and the rarity with which it was brought up, it may not be a good idea to pursue this.

Furthermore, the decision to keep the push notification for the app with action buttons or to have an "in-app" mechanism to answer the notification remains a bit of a controversial issue. A single participant brought up their gripes with buttons attached to notifications. This suggestion motivated the development team to create an alternative "in-app" method. During task 10, participants 3-7 were presented with two alternatives for the notification, one "in-app" and the other not.

Only a single participant preferred the "in-app" variant. So, we recommend keeping the push notification with buttons for now. It is advisable to leave the code for the "in-app" version available for further testing in the future.

B.6 Conclusion

Our participants found that URU-S was a relatively intuitive application. Furthermore, they stressed its importance for runners, cyclists, and avid walkers. All participants recognized the importance of having a vulnerable road user protection application such as this one. We recommend that we implement the recommendations and continue to work with users to ensure that we develop a practical and user-centered application.

Appendix C

Privacy Policy

C.1 URU-S

The gathering of personal data is done under Regulation (EU) 2016/679 of the European Parliament and Council, April 27th, 2016. The regulation is relative to the protection of singular people in terms of personal data gathering and the free circulation of said data (RGPD), including the national legislation that complements it, mainly Law n. 58/2019, August 8th, that ensures the execution, in terms of the juridical Nacional order, of RGPD.

Storage of personal data will be done only during the duration of the masters' dissertation known as "Mobile app for protecting cyclists and pedestrians in road traffic," this is scientific research in the area of usability of a road safety system for vulnerable road users that aims to:

- Raise awareness to the protection of vulnerable road users.
- Prototype a usable interface to protect vulnerable road users by emitting warnings in their smartphones.
- Conduct a user-centered design process that considers all the needs and opinions of users, involving them through Focus Groups and Usability Testing to guarantee the development of an efficient and helpful interface.

The masters' thesis is hosted by the Telecommunications Institute – Porto (IT-Porto) and the Faculty of Engineering of the University of Porto (FEUP); these organizations are responsible for the conjoint treatment of personal data involved in the study. This thesis is formally known as "Mobile app for protecting cyclists and pedestrians in road traffic" and is led by FEUP.

This project will culminate in a prototype available in a private GitHub repository, not being open to the general public. The source code is only accessible through gaining access to the repository. The application itself will be available in phones meant explicitly for testing, lent by the Telecommunications Institute, shared with the voluntary participants. It will not be necessary for the participants to install the application on their phones.

The study will not gather personal data without your consent. For testing purposes, we will present a privacy policy that is clear and detailed, describing the objectives of the study and the treatment of your data throughout the said study; this project will always solicit your consent.

Your privacy is a significant concern to our team and will be faced with great seriousness.

C.2 Goals of the Treatment

This investigation work is framed within the context of a masters' dissertation in Informatics Engineering and Computation. It aims to raise awareness of the protection of vulnerable road users, involving them actively in a cooperative collision avoidance system through smartphones. Personal data will be exclusively used for investigation purposes related to the dissertation "Mobile app for protecting cyclists and pedestrians in road traffic." Data will not be reutilized for other means, shared with third parties, or marketing targets.

C.3 Collected Personal Data

The categories of personal data treated in this investigation, developed for the dissertation, are the following:

- Time-gated questionnaires:
 - Sociodemographic data (Age, Education-level, Profession).
 - Data pertaining to how tech-savvy someone is (on a scale of 1 to 5).
 - Data about your daily routine (if you frequently bike or go on walks, either individually or on your way to public transportation).
 - Electronic data (e-mail to notify you of other testing phases).
- Video data during the usability testing phase.

This data will be gathered in two moments: during the focus group and the usability testing phase. The latter will be done with devices lent by the Institute and do not require the participant's personal phone use.

Upon your first use of the application, there will be a demographic questionnaire (Age, Education-level, Profession) after giving your consent. Additionally, there will be questions about how tech-savvy you are (1 to 5) and about your routines and e-mail address in case you are interested in participating again.

At the end of each usability testing phase, there will be a questionnaire related to the difficulty the user had in reacting to the notification emitted in the smartphone that was lent to them.

C.4 Conservation Dates

Data will be stored until the end of the masters' dissertation, 2021-10-31.

Data can be deleted upon request of the participants and will be destroyed/anonymized after the mentioned date.

C.5 Receivers of Personal Data

Gathered data will only be shared after pseudonymization.

There will be no transference of data to countries outside of the European Union.

Members of the investigation team associated with URU-S will be the only ones treating personal data. The final result of the investigation, which will contain anonymized or aggregated data that make it impossible to identify each participant, will be disseminated through the scientific community.

C.6 Data Owner Rights

The law allows you the right to Information, Access, Correction, Deletion, Portability, and Limitation of the treatment, which are respected unless it is impossible or heavily damaging to the treatment objectives for investigative purposes. As the owner of the data, you also have the right to remove consent to data treatment at any time.

As the data owner, you have the right to present formal complaints to a European Authority of supervision; in Portugal, the competent Authority is the CNPD (www.cnpd.pt).

At any moment, you may contact the DPO of the UP, the entity that establishes a common contact point through the following e-mail address dpo@up.pt

C.7 Security Measures

The gathered data will not be shared with third parties, and there will be a series of technical and organizational measures in place to assure confidentiality, integrity, physical and logical security of the information, namely, pseudonymization and control of logical and physical access. Given the nature and minimal quantity of the gathered data, it is implausible that it would be possible to revert the pseudonymization process. The one responsible for the study will adopt good practices in the use of informatics systems.

C.8 Use of Equipment and Questionnaires

The testing phones will be lent only during the usability testing phases and then promptly return to IT-Porto. Since the user never has to install the application on their actual phones, they will only have to consent to the questionnaires that will be made available, mentioned in other sections.

C.9 Remove Consent

The data owner may remove their consent to the treatment of their data; if that happens, they may contact up201605769@fe.up.pt to reach a compromise.

C.10 Contacts

If you have any doubts about the investigation project or exercising your rights as owner of the data, please contact up201605769@fe.up.pt.

C.11 Responsible for Data Protection

Luís Mendes (up201605769@fe.up.pt)

C.12 Informed Consent Term

I declare that I read and understood the information regarding the “Mobile app for protecting cyclists and pedestrians in road traffic” and that I wish to participate, knowing that I may give up at any moment without any form of repercussion or need of a justification.

Knowing that this project is directed to people above 18 years of age, I confirm that I am over 18 years old.

I consent to the treatment of my sociodemographic data and video for research purposes in the present study.

Date: / / 2021

Bibliography

- [1] World Health Organization. *Global status report on road safety 2018*. Licence: CC BY-NC-SA 3.0 IGO. World Health Organization, 2018. URL: <https://www.who.int/publications/i/item/9789241565684>.
- [2] Ioannis Vourgidis et al. “Use Of Smartphones for Ensuring Vulnerable Road User Safety through Path Prediction and Early Warning: An In-Depth Review of Capabilities, Limitations and Their Applications in Cooperative Intelligent Transport Systems”. In: *Sensors* 20.4 (2020), p. 997. ISSN: 1424-8220. DOI: [10.3390/s20040997](https://doi.org/10.3390/s20040997).
- [3] Johan Scholliers, Marcel Van Sambeek, and Kees Moerman. “Integration of vulnerable road users in cooperative ITS systems”. In: *European Transport Research Review* 9.2 (2017). ISSN: 1867-0717. DOI: [10.1007/s12544-017-0230-3](https://doi.org/10.1007/s12544-017-0230-3). URL: <https://dx.doi.org/10.1007/s12544-017-0230-3>.
- [4] Goncalo Pereira, Pedro M. Dorey, and Ana Aguiar. “Poster: Cooperative Perception Platform for Intelligent Transportation Systems”. In: *2020 IEEE Vehicular Networking Conference (VNC)* (2020). DOI: [10.1109/vnc51378.2020.9318404](https://doi.org/10.1109/vnc51378.2020.9318404).
- [5] Taeho Kim et al. “Vulnerable Road User Protection through Intuitive Visual Cue on Smartphones”. In: *Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services* (2017), pp. 13–17. DOI: [10.1145/3131944.3131950](https://doi.org/10.1145/3131944.3131950).
- [6] Shoma Hisaka and Shunsuke Kamijo. “On-board wireless sensor for collision avoidance: Vehicle and pedestrian detection at intersection”. In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)* (2011), pp. 198–205. DOI: [10.1109/itsc.2011.6082853](https://doi.org/10.1109/itsc.2011.6082853).
- [7] Xinzhou Wu et al. “Cars Talk to Phones: A DSRC Based Vehicle-Pedestrian Safety System”. In: *2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall)* (2014). DOI: [10.1109/vtcfall.2014.6965898](https://doi.org/10.1109/vtcfall.2014.6965898).
- [8] 5GAA. *5GAA Website*. Accessed 25 March 2021. Sept. 2016. URL: <http://5gaa.org/>.
- [9] Dario Sabella et al. *Toward fully connected vehicles: Edge computing for advanced automotive communications*. 5GAA, 2017, pp. 4–16.

- [10] A. Napolitano et al. “Implementation of a MEC-based Vulnerable Road User Warning System”. In: *2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)* (2019). DOI: [10.23919/eeta.2019.8804497](https://doi.org/10.23919/eeta.2019.8804497).
- [11] Raymond M. Lee. “The Secret Life of Focus Groups: Robert Merton and the Diffusion of a Research Method”. In: *The American Sociologist* 41.2 (2010), pp. 115–141. ISSN: 0003-1232. DOI: [10.1007/s12108-010-9090-1](https://doi.org/10.1007/s12108-010-9090-1).
- [12] John D. Gould and Clayton Lewis. “Designing for usability: key principles and what designers think”. In: *Communications of the ACM* 28.3 (1985), pp. 300–311. ISSN: 0001-0782. DOI: [10.1145/3166.3170](https://doi.org/10.1145/3166.3170).
- [13] S. Dow et al. “Wizard of Oz Support throughout an Iterative Design Process”. In: *IEEE Pervasive Computing* 4.4 (2005), pp. 18–26. ISSN: 1536-1268. DOI: [10.1109/mprv.2005.93](https://doi.org/10.1109/mprv.2005.93).
- [14] Quang-Huy Nguyen et al. “Car-to-Pedestrian communication with MEC-support for adaptive safety of Vulnerable Road Users”. In: *Computer Communications* 150 (2020), pp. 83–93. ISSN: 0140-3664. DOI: [10.1016/j.comcom.2019.10.033](https://doi.org/10.1016/j.comcom.2019.10.033).
- [15] Sergei S. Avedisov et al. “Perceived Safety: A New Metric for Evaluating Safety Benefits of Collective Perception for Connected Road Users”. In: *2020 IEEE Vehicular Networking Conference (VNC)* (2020), pp. 131–134.
- [16] Yongki Lee et al. “A Novel Path Planning Algorithm for Truck Platooning Using V2V Communication”. In: *Sensors* 20.24 (2020), p. 7022. ISSN: 1424-8220. DOI: [10.3390/s20247022](https://doi.org/10.3390/s20247022).
- [17] Marek Bachmann et al. “The Wireless Seat Belt Requirements, Experiments, and Solutions for Pedestrian Safety”. In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)* (2018), pp. 361–366. DOI: [10.1109/percomw.2018.8480123](https://doi.org/10.1109/percomw.2018.8480123).
- [18] Titov Waldemar, Felix Boehm, and Thomas Schlegel. “Prototyping Approach of Networking Road Users for Cooperative Collision Avoidance using Smartphones”. In: *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)* (2019), pp. 374–379. DOI: [10.1109/iot sms48152.2019.8939273](https://doi.org/10.1109/iot sms48152.2019.8939273).
- [19] Julian Heinovski et al. “Modeling Cycling Behavior to Improve Bicyclists’ Safety at Intersections – A Networking Perspective”. In: *20th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2019)*. Washington, D.C.: IEEE, June 2019. ISBN: 978-1-7281-0270-2. DOI: [10.1109/WoWMoM.2019.8793008](https://doi.org/10.1109/WoWMoM.2019.8793008).
- [20] Marie-Christin H. Oczko et al. “Integrating Haptic Signals with V2X-based Safety Systems for Vulnerable Road Users”. In: *IEEE International Conference on Computing, Networking and Communications (ICNC 2020)*. Kailua, HI: IEEE, Feb. 2020, pp. 692–697. DOI: [10.1109/ICNC47757.2020.9049723](https://doi.org/10.1109/ICNC47757.2020.9049723).

- [21] Stefan Loewen et al. “Backwards compatible extension of CAMs/DENMs for improved bike safety on the road”. In: *2017 IEEE Vehicular Networking Conference (VNC)* (2017). DOI: [10.1109/vnc.2017.8275657](https://doi.org/10.1109/vnc.2017.8275657).
- [22] José Bastos Pintor. “Modeling and Performance Evaluation of Bicycle-to-X Communication Networks”. MA thesis. s/n, R. Dr. Roberto Frias, 4200-465 Porto: FEUP - Faculdade de Engenharia da Universidade do Porto, Feb. 2019.
- [23] Yifu Liu et al. “Vehicle position and context detection using V2V communication with application to pre-crash detection and warning”. In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (2016). DOI: [10.1109/ssci.2016.7850093](https://doi.org/10.1109/ssci.2016.7850093).
- [24] Mahmoud Shawki and M. Saeed Darweesh. “Collision Probability Computation for Road Intersections Based on Vehicle to Infrastructure Communication”. In: *2020 32nd International Conference on Microelectronics (ICM)* (2020). DOI: [10.1109/icm50269.2020.9331802](https://doi.org/10.1109/icm50269.2020.9331802).
- [25] Chi-Yu Li et al. “V2PSense: Enabling Cellular-Based V2P Collision Warning Service through Mobile Sensing”. In: *2018 IEEE International Conference on Communications (ICC)* (2018). DOI: [10.1109/icc.2018.8422981](https://doi.org/10.1109/icc.2018.8422981).
- [26] Andreas Jahn, Michel Morold, and Klaus David. “5G Based Collision Avoidance - Benefit from Unobtrusive Activities”. In: *2018 European Conference on Networks and Communications (EuCNC)* (2018), pp. 352–356. DOI: [10.1109/eucnc.2018.8442711](https://doi.org/10.1109/eucnc.2018.8442711).
- [27] Marco Malinverno et al. “Edge-Based Collision Avoidance for Vehicles and Vulnerable Users: An Architecture Based on MEC”. In: *IEEE Vehicular Technology Magazine* 15.1 (2020), pp. 27–35. ISSN: 1556-6072. DOI: [10.1109/mvt.2019.2953770](https://doi.org/10.1109/mvt.2019.2953770).
- [28] Alexis Yáñez and Sandra Céspedes. “Pedestrians also Have Something to Say: Integration of Connected VRU in Bidirectional Simulations”. In: *2020 IEEE Vehicular Networking Conference (VNC)* (2020), pp. 155–158.
- [29] Michel Morold et al. “Requirements on Delay of VRU Context Detection for Cooperative Collision Avoidance”. In: *92nd IEEE Vehicular Technology Conference (VTC 2020-Fall)*. Virtual Conference: IEEE, Nov. 2020.
- [30] Suhua Tang, Kiyoshi Saito, and Sadao Obana. “Transmission control for reliable pedestrian-to-vehicle communication by using context of pedestrians”. In: *2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES)* (2015), pp. 41–47. DOI: [10.1109/icves.2015.7396891](https://doi.org/10.1109/icves.2015.7396891).
- [31] Jasper Jahn Marcus Bocksch Jochen Seitz. “Pedestrian Activity Classification to Improve Human Tracking and Localization”. In: *2013 International Conference on Indoor Positioning and Indoor Navigation* (2013).

- [32] Jens Kotte et al. “Concept of an enhanced V2X pedestrian collision avoidance system with a cost function–based pedestrian model”. In: *Traffic Injury Prevention* 18.sup1 (2017). PMID: 28368684, S37–S43. DOI: [10.1080/15389588.2017.1310380](https://doi.org/10.1080/15389588.2017.1310380). eprint: <https://doi.org/10.1080/15389588.2017.1310380>. URL: <https://doi.org/10.1080/15389588.2017.1310380>.
- [33] Jakob Nielsen and Rolf Molich. “Heuristic Evaluation of User Interfaces”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’90. Seattle, Washington, USA: Association for Computing Machinery, 1990, pp. 249–256. ISBN: 0201509326. DOI: [10.1145/97243.97281](https://doi.org/10.1145/97243.97281). URL: <https://doi.org/10.1145/97243.97281>.
- [34] *Sensors Overview | Android Developers*. https://developer.android.com/guide/topics/sensors/sensors_overview. Accessed 16 May 2020.
- [35] Lukas Stratmann et al. “Psychological Feasibility of a Virtual Cycling Environment for Human-in-the-Loop Experiments”. In: *Jahrestagung der Gesellschaft für Informatik (INFORMATIK 2019), 1st Workshop on ICT based Collision Avoidance for VRUs (ICT4VRU 2019)*. Ed. by Claude Draude, Martin Lange, and Bernhard Sick. Vol. LNI P-295. Kassel, Germany: Gesellschaft für Informatik e.V. (GI), Sept. 2019, pp. 185–194. ISBN: 978-3-88579-689-3. DOI: [10.18420/inf2019_ws21](https://doi.org/10.18420/inf2019_ws21).
- [36] Alexander Krüger et al. “TVA in the wild: Applying the theory of visual attention to game-like and less controlled experiments”. In: *Open Psychology* (2020). to appear.
- [37] Sophia Antipolis Cedex. *Intelligent Transport Systems (ITS); Framework for Public Mobile Networks in Cooperative ITS (C-ITS)*. 1.1.1. ETSI, 2012, pp. 5–63.
- [38] Jeff Sutherland and Ken Schwaber. *The 2020 Scrum Guide*. <https://www.scrumguides.org/scrum-guide.html>. Accessed 23 December 2020. 2020.
- [39] Heila Van Der Merwe, Brink Van Der Merwe, and Willem Visse. “Verifying android applications using Java PathFinder”. In: *ACM SIGSOFT Software Engineering Notes* 37.6 (2012), p. 1. ISSN: 0163-5948. DOI: [10.1145/2382756.2382797](https://doi.org/10.1145/2382756.2382797).
- [40] Ana Rosario Espada et al. “Using Model Checking to Generate Test Cases for Android Applications”. In: *Electronic Proceedings in Theoretical Computer Science* 180 (2015), pp. 7–21. ISSN: 2075-2180. DOI: [10.4204/eptcs.180.1](https://doi.org/10.4204/eptcs.180.1).
- [41] Guangdong Bai et al. “Towards Model Checking Android Applications”. In: *IEEE Transactions on Software Engineering* 44.6 (2018), pp. 595–612. ISSN: 0098-5589. DOI: [10.1109/tse.2017.2697848](https://doi.org/10.1109/tse.2017.2697848).
- [42] Ana Rosario Espada et al. “A formal approach to automatically analyse extra-functional properties in mobile applications”. In: *Software Testing, Verification and Reliability* 29.4-5 (2019), e1699. ISSN: 0960-0833. DOI: [10.1002/stvr.1699](https://doi.org/10.1002/stvr.1699). URL: <https://dx.doi.org/10.1002/stvr.1699>.

- [43] *Android UI Automator* | *Android Developers*. <https://developer.android.com/training/testing/ui-automator>. Accessed 28 April 2021.
- [44] *Android Espresso* | *Android Developers*. <https://developer.android.com/training/testing/espresso>. Accessed 28 April 2021.
- [45] *Android Instrumented Unit Tests* | *Android Developers*. <https://developer.android.com/training/testing/unit-testing/instrumented-unit-tests>. Accessed 28 April 2021.
- [46] *UI or Application Exerciser Monkey* | *Android Developers*. <https://developer.android.com/studio/test/monkey>. Accessed 29 April 2021.
- [47] *NumberEight SDK*. <https://www.numbereight.ai/>. Accessed 29 April 2021.
- [48] *Maps SDK for Android* | *Android Developers*. <https://developers.google.com/maps/documentation/android-sdk/overview>. Accessed 29 April 2021.
- [49] *Geofencing API* | *Android Developers*. <https://developer.android.com/training/location/geofencing>. Accessed 29 April 2021.
- [50] *Firebase Cloud Messaging*. <https://firebase.google.com/docs/cloud-messaging>. Accessed 29 April 2021.
- [51] *Contrast Ratio*. <https://contrast-ratio.com/>. Accessed 02 May 2021.
- [52] *React - A JavaScript library for building user interfaces*. <https://reactjs.org/>. Accessed 29 April 2021.
- [53] *Lingver*. <https://github.com/YarikSOffice/lingver>. Accessed 29 April 2021.
- [54] *Markwon*. <https://github.com/noties/Markwon>. Accessed 29 April 2021.
- [55] *Android Donut Library*. <https://github.com/futuredapp/donut>. Accessed 29 April 2021.
- [56] David Marcelo Duarte Lourenço. “Análise Espacial da Sinistralidade Rodoviária na Cidade do Porto”. MA thesis. s/n, R. Dr. Roberto Frias, 4200-465 Porto: FEUP - Faculdade de Engenharia da Universidade do Porto, Nov. 2019.
- [57] Mariana Vilaça, Eloísa Macedo, and Margarida C. Coelho. “A Rare Event Modelling Approach to Assess Injury Severity Risk of Vulnerable Road Users”. In: *Safety* 5.2 (2019). ISSN: 2313-576X. DOI: [10.3390/safety5020029](https://doi.org/10.3390/safety5020029). URL: <https://www.mdpi.com/2313-576X/5/2/29>.
- [58] Tânia Sofia Almeida Marinho. “Avaliação do Impacto de Ocorrências na Mobilidade da Rede Viária do Porto”. MA thesis. s/n, R. Dr. Roberto Frias, 4200-465 Porto: FEUP - Faculdade de Engenharia da Universidade do Porto, Sept. 2020.
- [59] *Android FusedLocationProviderClient* | *Android Developers*. <https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient>. Accessed 01 May 2021.

- [60] Jesse Weaver, Qi Han, and Archan Misra. “Building the Case for Dynamic Location Query Processing”. In: *2016 17th IEEE International Conference on Mobile Data Management (MDM)* (2016), pp. 40–49. DOI: [10.1109/mdm.2016.20](https://doi.org/10.1109/mdm.2016.20).
- [61] *Test your app’s accessibility* | *Android Developers*. <https://developer.android.com/guide/topics/ui/accessibility/testing>. Accessed 02 May 2021.
- [62] *Secure OS*. <https://securegroup.com/secure-os/>. Accessed 02 May 2021.
- [63] Gogoll, Jan and Zuber, Niina and Kacianka, Severin and Greger, Timo and Pretschner, Alexander and Nida-Rümelin, Julian. “Ethics in the Software Development Process: from Codes of Conduct to Ethical Deliberation”. In: *Philosophy I&’ Technology* (2021). ISSN: 2210-5433. DOI: [10.1007/s13347-021-00451-w](https://doi.org/10.1007/s13347-021-00451-w).
- [64] *Service* | *Android Developers*. <https://developer.android.com/reference/android/app/Service>. Accessed 02 May 2021.
- [65] *Android Vibrator API* | *Android Developers*. <https://developer.android.com/reference/android/os/Vibrator>. Accessed 22 December 2020.
- [66] *JaCoCo Java Code Coverage Library*. <https://www.eclemma.org/jacoco/>. Accessed 28 April 2021.
- [67] *Geofence.GeofenceTransition* | *Google Play services* | *Google Developers*. <https://developers.google.com/android/reference/com/google/android/gms/location/Geofence.GeofenceTransition>. Accessed 29 May 2021.
- [68] Steve Krug. *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems*. 1st. USA: New Riders Publishing, 2009. ISBN: 0321657292.
- [69] *SenseMyCity - Google Play Apps*. https://play.google.com/store/apps/details?id=future.cities.sensemycity&hl=pt_PT&gl=US. Accessed 1 June 2021.
- [70] *Android Framework Code* | *config.xml*. <https://android.googlesource.com/platform/frameworks/base/+40caf8f4432acd2b9d9230b2b1371660521415c2/core/res/res/values/config.xml>. Accessed 30 May 2021.
- [71] *Smart Time Sync - Google Play Apps*. https://play.google.com/store/apps/details?id=com.pautinanet.smarttimesync&hl=pt_PT&gl=US. Accessed 30 May 2021.
- [72] *VisualGPS, LLC*. <https://www.visualgps.net>. Accessed 30 May 2021.
- [73] *Understanding message delivery* | *Firebase*. <https://firebase.google.com/docs/cloud-messaging/understand-delivery>. Accessed 31 May 2021.
- [74] Ryota Kanai et al. “Larger Stimuli Require Longer Processing Time for Perception”. In: *Perception* 46.5 (2017), pp. 605–623. ISSN: 0301-0066. DOI: [10.1177/0301006617695573](https://doi.org/10.1177/0301006617695573). URL: <https://dx.doi.org/10.1177/0301006617695573>.

- [75] Daniel Linares and Alex O. Holcombe. “Differences in Perceptual Latency Estimated from Judgments of Temporal Order, Simultaneity and Duration are Inconsistent”. In: *i-Perception* 5.6 (2014), pp. 559–571. ISSN: 2041-6695. DOI: [10.1068/i0675](https://doi.org/10.1068/i0675). URL: <https://dx.doi.org/10.1068/i0675>.