

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

An Open-Source Smartwatch-based System to Enhance Cyclist Safety

Mafalda Inês Matos Dinis



Master in Electrical and Computer Engineering

Supervisor: Daniel Gouveia Costa

September 16, 2025

Resumo

O crescimento contínuo das populações urbanas a nível mundial tem impulsionado um aumento das taxas de urbanização. Consequentemente, desafios relacionados com o transporte, sustentabilidade ambiental e segurança pública têm-se tornado progressivamente mais proeminentes. Entre os diversos meios de transporte disponíveis, as bicicletas destacam-se pela sua sustentabilidade ambiental, acessibilidade financeira e promoção de estilos de vida mais saudáveis. Contudo, a adoção generalizada do ciclismo como um dos modos importantes de mobilidade urbana permanece limitada, principalmente devido a preocupações relacionadas com a segurança dos ciclistas na estrada.

Para responder a estas preocupações, esta dissertação apresenta um sistema inovador *open-source* para um *smartwatch*, desenvolvido com o objetivo de melhorar a segurança dos ciclistas em ambientes urbanos. Este utiliza um Sistema de Informação Geográfica (GIS), bem como técnicas de análise espacial de risco, para avaliar o risco associado às interseções urbanas. A abordagem desenvolvida atribui um valor de fator de risco a cada interseção com base no número de vias que nela convergem. Além disso, ao monitorizar continuamente a localização e velocidade do ciclista em tempo real, o *smartwatch* avalia e comunica dinamicamente os níveis de risco ao utilizador.

O sistema *wearable* proposto utiliza uma interface intuitiva que combina feedback visual e alertas hápticos. A mudança de cor de fundo do *smartwatch* alerta os ciclistas sobre o nível de risco da área que se aproximam, enquanto a vibração é acionada apenas caso a velocidade do utilizador ultrapasse um limite de segurança predefinido. Esta comunicação proativa do risco visa aumentar a consciência do utilizador, encorajando os ciclistas a adaptarem a sua velocidade e comportamento face a perigos iminentes.

Ao contrário de soluções previamente propostas que dependem de alterações complexas de infraestrutura ou de tecnologias menos adaptativas, esta abordagem baseada num *smartwatch* distingue-se pela sua natureza *open-source*, flexível e altamente configurável, assegurando ampla acessibilidade e baixo custo. Em última análise, esta investigação pretende fornecer uma contribuição tecnológica para o domínio da mobilidade ativa e da segurança ciclística, promovendo uma adoção mais ampla de práticas de transporte sustentável. Ao mitigar as barreiras relacionadas com a segurança dos ciclistas, o sistema *smartwatch* desenvolvido pretende contribuir para um ambiente urbano mais seguro, saudável e sustentável.

Abstract

The continuous growth of urban populations worldwide has prompted an increase in urbanization rates. As a result, challenges related to transportation, environmental sustainability, and public safety have progressively become more prominent. Among the various modes of transportation available, bicycles are highly regarded for their environmental friendliness, affordability, and promotion of healthier lifestyles. Nevertheless, the widespread adoption of cycling as a key mode of urban mobility remains hindered, primarily due to concerns regarding the safety of cyclists on the road.

To address these concerns, this dissertation presents an innovative open-source smartwatch-based system, developed with the goal of improving cyclist safety within urban environments. The system leverages a Geographic Information System (GIS), as well as spatial risk analysis techniques, to assess the inherent risk of urban intersections. The developed approach assigns a risk factor value to each intersection, based on the number of roads that converge to it. Furthermore, by continuously monitoring the cyclist's location and velocity in real-time, the smartwatch dynamically evaluates and communicates the risk levels to the user.

The proposed wearable system employs an intuitive user interface combining both visual feedback and haptic alerts. The smartwatch's changing background color alerts cyclists about the risk level of the area they are approaching, while the vibration is only employed if the user's velocity exceeds a predefined safety speed. This proactive risk communication aims to enhance user awareness, encouraging cyclists to adapt their speed and behavior in response to imminent hazards.

Unlike previously proposed solutions that either rely on complex infrastructure modifications or less adaptive technologies, this smartwatch-based approach stands out due to its open-source, flexible, and highly configurable nature, ensuring broad accessibility and affordability. Ultimately, this research aims to provide a technological contribution to the domain of soft mobility and cycling safety, promoting a broader adoption of sustainable transportation practices. By mitigating barriers associated with cyclist safety, the smartwatch system developed hopes to contribute towards a safer, healthier, and more sustainable urban transportation environment.

United Nations Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) are a set of 17 goals which aim to address pressing global challenges. Set in 2015 as part of the 2030 Agenda for Sustainable Development, these goals serve as an urgent call to action to address critical economic, social, and environmental issues worldwide. [3]



Figure 1: Sustainable Development Goals (SDGs) [3]

This dissertation focuses on an open-source approach to develop a cyclist safety system using wearable technology. By developing an affordable and versatile design, this project aligns with and contributes to the advancement of multiple SDGs.

Table 1 summarizes how this research aligns with the Sustainable Development Goals (SDGs). The "SDG" column identifies each goal addressed by the project. The "Target" column specifies the particular UN target within that goal which the work contributes to. The "Contribution" column explains the concrete ways in which the proposed smartwatch-based cyclist safety system supports

that target. Finally, the "Performance Indicators" column lists measurable criteria that can be used to evaluate the project's impact with respect to each SDG.

Table 1: Sustainable Development Goals Addressed by the Research

SDG	Target	Contribution	Performance Indicators
3 – Good Health and Well-Being	3.6 - Halve the number of global deaths and injuries from road traffic accidents	The smartwatch system provides risk alerts for cyclists, helping reduce accidents and improving the perception of safety. This encourages active mobility and healthier lifestyles.	- Reported decrease in cycling incidents - Post-deployment surveys on perceived cycling safety
11 – Sustainable Cities and Communities	11.2 - Provide access to safe, affordable, accessible and sustainable transport systems for all, improving road safety	The system fosters safe and sustainable urban mobility by encouraging cycling and supporting soft-mobility.	- Growth in cycling adoption in areas where the system is deployed - Reduction in perceived safety barriers reported by users
12 – Responsible Consumption and Production	12.6 - Encourage companies and organizations to adopt sustainable practices and to integrate sustainability information	The project's open-source nature and reliance on low-cost hardware promote responsible, resource-efficient technological development.	- Number of projects reusing the hardware/software - Citations of the solution in academic or technical publications
13 – Climate Action	13.2 – Integrate climate change measures into national policies, strategies and planning	By encouraging cycling instead of use of other transports reliant on fossil fuels, the system helps reduce greenhouse gas emissions and supports climate-mitigation strategies.	- Estimated CO_2 equivalent emissions avoided through increased cycling adoption - Increase in cyclists in areas where the system is deployed

Agradecimentos

Quero começar por agradecer aos meus pais por todo o apoio que me deram. Agradeço à minha mãe por toda a confiança que sempre teve em mim, e por me ter inculcido desde pequena que tudo está ao meu alcance se eu não me limitar e investir no que me faz feliz. Ao meu pai agradeço todo o esforço para me proporcionar o melhor futuro e a melhor educação possível, bem como a compreensão que me demonstrou perante os meus vários insucessos ao longo do percurso.

Agradeço também à minha irmã, que começou por ser a minha primeira professora de inglês numa sala de aula preenchida com peluches e uma menina que mal escrever sabia. Obrigada por me acompanhares desde sempre e por constantemente me demonstrares o melhor exemplo a seguir.

Quero agradecer ainda ao meu namorado, por toda a paciência durante uma das fases mais difíceis da minha vida, e pela ajuda incondicional para lidar com os problemas do dia-a-dia.

Agradeço ao resto da minha família e amigos por todo o apoio e motivação que me deram ao longo do percurso, por sempre acreditarem em mim e por todos os bons momentos passados juntos.

Finalmente, agradeço ao meu orientador por todo o apoio e paciência e por ter sempre acreditado no sucesso deste projeto.

Mafalda Dinis

“A mind needs books as a sword needs a whetstone, if it is to keep its edge.”

George R. R. Martin

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Goals	2
1.4	Dissertation Structure	3
2	Background and Literature Review	4
2.1	Smart Cities	4
2.2	Urban Mobility	6
2.3	Soft Mobility	7
2.4	Embedded Systems for Smart Cycling	7
2.5	State of the Art and Current Challenges	10
3	Methodology	16
3.1	System Architecture	16
3.1.1	Smartwatch	17
3.1.2	Smartphone	18
3.2	Proposed Approach	19
3.3	Preprocessing: City Map Generator	20
3.3.1	Python Packages	21
3.3.2	Implementation	21
3.4	Smartwatch Application	29
3.4.1	Development Tools	29
3.4.2	Implementation	30
3.5	Mobile Application	37
3.5.1	Development Tools	38
3.5.2	Implementation	38
3.6	System Algorithm Structure	41
4	Results and Discussion	44
4.1	Results	44
4.1.1	Graphical Results	44
4.1.2	Performance Metrics	47
4.2	Discussion	57
4.2.1	Developed System	58
4.2.2	Alternative Approaches Explored	58

5 Conclusion and Future Work	63
5.1 Objective Satisfaction	63
5.2 Future Work	64
References	67

List of Figures

1	Sustainable Development Goals (SDGs) [3]	iii
3.1	LILYGO T-Watch S3 [1]	17
3.2	Simplified System Architecture	19
3.3	Generated Map of Porto, Portugal - Every Node is Plotted	22
3.4	Maps of Porto, Portugal, Displaying the Effects of Consolidating Intersections and Respective Consolidation Ratio	24
(a)	Without Consolidating Intersections (consolidation_ratio = 100%)	24
(b)	Intersections within a 5 m Radius are Consolidated (consolidation_ratio = 61.88%)	24
(c)	Intersections within a 10 m Radius are Consolidated (consolidation_ratio = 46.48%)	24
(d)	Intersections within a 15 m Radius are Consolidated (consolidation_ratio = 34.89%)	24
(e)	Intersections within a 20 m Radius are Consolidated (consolidation_ratio = 26.08%)	24
(f)	Intersections within a 25 m Radius are Consolidated (consolidation_ratio = 18.56%)	24
3.5	Normalization Results	25
(a)	Histogram of Normalized Street Count	25
(b)	Cumulative Histogram of Normalized Street Count	25
(c)	Normalized Color Map	25
3.6	Uniformization Results using 3 Quantiles	26
(a)	Histogram after Quantile Transformation (3 Quantiles)	26
(b)	Cumulative Histogram after Quantile Transformation (3 Quantiles)	26
(c)	Color Map after Quantile Transformation with 3 Quantiles	26
3.7	Uniformization Results using 5 Quantiles	27
(a)	Histogram after Quantile Transformation (5 Quantiles)	27
(b)	Cumulative Histogram after Quantile Transformation (5 Quantiles)	27
(c)	Color Map after Quantile Transformation with 5 Quantiles	27
3.8	City Map Generator Flowchart	28
3.9	Proximity Analysis Flowchart	32
3.10	Contrast between Linear and Stepped Transformations	35
(a)	Linear Transformation	35
(b)	Stepped Transformation (5 steps)	35
3.11	Core Execution Loop Flowchart	36
3.12	Mobile Application Flowchart	40
3.13	System Flowchart	42

4.1	Smartwatch UI	45
(a)	Client Connected and Displayed Search Progress	45
(b)	Information Displayed on Screen	45
(c)	New Search Progress Displayed before Screen Information is Updated	45
4.2	Mobile App UI	46
(a)	UI Before Client Connection	46
(b)	Requesting Permission to Turn On Bluetooth	46
(c)	Scanning for Devices	46
(d)	UI After Client Connection	46
4.3	MAX_BATCH_SIZE Experiment	50
4.4	MIN_YIELD_MS Experiment	51
4.5	MAX_PROCESS_MS Experiment	52
4.6	RESULT_BUFFER_SIZE Experiment	53
4.7	DEBOUNCE_MS Experiment	55
4.8	LINE_LEN Experiment	56
4.9	Map of Intersections in Porto, Portugal with Grid Clustering Approach	59
4.10	Map of Porto, Portugal with Voronoi Cells Approach	60

List of Tables

1	Sustainable Development Goals Addressed by the Research	iv
2.1	Comparison of Related Works in Cyclist Safety	14
4.1	Fixed Values Assigned to Each Parameter	49
4.3	Recommended Values Assigned to Each Parameter After Evaluation	57

Abbreviations and Symbols

AI	Artificial Intelligence
API	Application Programming Interface
APK	Android Package Kit
BLE	Bluetooth Low Energy
CLI	Command Line Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
FAT	File Allocation Table
FreeRTOS	Free Real-Time Operating System
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
IDF	IoT Development Framework
iOS	iPhone Operating System
IoT	Internet of Things
ISR	Interrupt Service Routine
LCD	Liquid-Crystal Display
LED	Light Emitting Diode
LoRa	Long Range
MCU	MicroController Unit
ML	Machine Learning
OSM	OpenStreetMap
PSRAM	Pseudo-Static Random Access Memory
RAM	Random Access Memory
RTOS	Real-Time Operating System
SDGs	Sustainable Development Goals
SDK	Software Development Kit
SoC	System-on-Chip
SoftAP	Software Access Point
TLS	Transport Layer Security
UI	User Interface
UN	United Nations
USD	United States Dollar

Chapter 1

Introduction

The rapid increase of urban populations across the globe has accelerated urbanization, bringing forth numerous challenges that contribute to a decrease in quality of life and have environmental repercussions. Among these challenges, mobility emerges as a critical concern, as growing demands on transportation infrastructure strain the efficiency, safety, and sustainability of urban mobility. Thus, ensuring equitable and secure access to mobility solutions is fundamental in order to address the broader impacts of urbanization.

Among the various transportation alternatives, cycling stands out for its minimal environmental footprint, cost-effectiveness, and health benefits. In recent years, cities have fostered the adoption of cycling through the expansion of dedicated infrastructure, such as the expansion of bike lanes and the development of bicycle-sharing systems. However, the growing presence of bicycles within complex urban environments also brings new safety challenges that are not always mitigated by infrastructure alone.

Despite the benefits of cycling, concerns over cyclist safety continue to hinder its broader adoption as a key mode of urban transport. Therefore, this dissertation explores these challenges and proposes a technological solution designed to improve cyclist safety through the development of an innovative smartwatch-based system.

1.1 Context

The accelerated pace of global urbanization has significantly reshaped urban mobility patterns, influencing the way people commute and interact within urban environments [26]. This evolution has placed considerable strain on existing transportation infrastructure, exacerbating issues such as traffic congestion, air pollution, and rising greenhouse gas emissions [15, 18, 33]. In light of these escalating concerns, urban planners and policymakers have increasingly advocated for sustainable mobility strategies, aimed at mitigating environmental impacts while enhancing the overall quality of urban life.

One prominent approach within sustainable urban development is soft mobility, which encompasses individual modes of transportation that are typically used for short distances. These personal forms of mobility promote flexibility, reduce environmental impact, and alleviate pressure on traditional transport infrastructure. Examples of such modes include walking, cycling, roller-skating, and skateboarding [25]. Among these alternatives, cycling is particularly noteworthy due to its numerous advantages, since it offers an affordable, environmentally sustainable approach, with significant health benefits.

Nonetheless, despite these well-documented benefits, the widespread adoption of cycling remains constrained, primarily as a result of inadequate infrastructure and the heightened safety risks associated with close interaction with motorized traffic [13]. This prevailing occurrence underscores the importance of developing and implementing innovative technological solutions capable of enhancing cyclist safety, in order to foster a broader adoption of cycling as a key mode of urban transportation [22].

1.2 Motivation

The motivation underlying this dissertation arises from the pressing need to improve cyclist safety within urban contexts. Urban cyclists are routinely exposed to heightened risks associated with vehicular traffic, particularly at intersections, which constitute critical points of conflict and are frequently the sites of serious collisions. This persistent vulnerability not only discourages the wider adoption of cycling, but also contributes to substantial public health and economic burdens, resulting from traffic-related injuries and fatalities.

Conventional strategies aimed at improving cyclist safety often focus on infrastructural interventions, which, although effective, involve considerable financial costs and extended implementation periods. Meanwhile, existing technological solutions often suffer from limitations in terms of accessibility, scalability, or contextual adaptability, rendering them unsuitable for widespread deployment.

These constraints highlight a clear need for the development of a cost-efficient, practical, and easily deployable technological system capable of dynamically assessing and communicating safety notices to urban cyclists.

1.3 Goals

The primary goal of this dissertation is to develop an open-source smartwatch-based system aimed at improving cyclist safety in urban environments. The proposed system seeks to deliver a low-cost wearable technological solution capable of warning cyclists of danger, with particular emphasis on the risks encountered at urban intersections. Thus, this dissertation aims to develop an intuitive, user-friendly interface on a smartwatch, combining visual and haptic alerts to effectively communicate the assessed risks to cyclists. Some additional objectives are outlined as follows:

- Develop a reproducible and configurable system architecture that enables deployment across diverse urban environments, ensuring adaptability to different city infrastructures.
- Address known limitations in urban mobility networks by leveraging indirect geospatial risk indicators to assess and mitigate cyclist safety concerns.
- Promote the adoption of cyclist safety technologies by demonstrating an open-source and cost-effective wearable solution.

Together, these goals establish a foundation for a system that not only enhances individual cyclist safety, but also contributes to broader efforts in promoting sustainable urban mobility. The proposed solution therefore aspires to bridge the gap between emerging smart city paradigms and the immediate practical needs of vulnerable road users.

1.4 Dissertation Structure

In order to present the proposed technological solution in a structured and comprehensive manner, the document is divided into 5 chapters.

Chapter 2, the Background and Literature Review, offers a critical examination of the current body of research concerning urban mobility, soft mobility, and cyclist safety. It evaluates existing technological interventions and identifies prevailing gaps and opportunities for innovation within the field.

Chapter 3, Methodology, delineates the research approach, outlining the specific methods, tools, and techniques employed throughout the study. This section aims to ensure methodological transparency, enabling future reproducibility of the research.

Chapter 4, Results and Discussion, presents the outcomes derived from testing. The chapter, additionally, mentions other explored approaches that were excluded from the final project.

Chapter 5, Conclusion and Future Work, synthesizes the principal findings of the study, outlines the dissertation's contributions to the field, and discusses identified limitations. It concludes with recommendations for future research and potential directions for further development of technological solutions to improve cyclist safety and promote sustainable urban transport.

Chapter 2

Background and Literature Review

This chapter presents a comprehensive literature review, providing the necessary theoretical background and contextual foundation for this research. It begins in section 2.1 by introducing the concept of smart cities, exploring its definitions, applications, and relevance to urban environments. Section 2.2 follows with an overview of urban mobility, outlining both its benefits and its adverse effects on public health, infrastructure, and the environment. Subsequently, section 2.3 explores the notion of soft mobility, with a particular emphasis on cycling, highlighting its advantages as well as the safety challenges that hinder its widespread adoption. Section 2.4 then delves into the technical landscape of embedded systems for smart cycling, addressing development constraints, hardware and software considerations, and design trade-offs that shape modern cyclist safety solutions. Finally, Section 2.5 reviews the current state of the art in cyclist safety technologies, comparing recent approaches and identifying key limitations related to accessibility and contextual adaptability.

2.1 Smart Cities

Over the past several decades, the growth of the world's population has resulted in increased urbanization. This expansion at an unprecedented rate, has reshaped the spatial, social and environmental structure of cities worldwide. The rapid evolution of cityscapes is often associated with greater convenience and higher productivity in everyday life, yet it also amplifies pressures on other factors, such as air quality and land use. The complex nature of urban environments, along with the finite availability of resources, pose structural challenges that directly influence public health, ecosystem stability, and long-term economic resilience.

To address these pressures, scholars and policymakers have promoted data-driven planning practices that aim to optimize services and streamline decision-making. These efforts have resulted in the development of several technological interventions, with the shared objective of improving urban quality of life while minimizing environmental impact. Within this context, the term smart city has emerged as a way to describe the intentional application of Information and Communication Technologies (ICT) to mitigate complex metropolitan problems [28].

Despite this term being employed in literature numerous times in recent years, there is no universally agreed definition of a smart city. Gracias et al. [20] conducted a literature review, analyzing multiple definitions of the concept and proposed the following one: "Smart cities use digital technologies, communication technologies, and data analytics, to create an efficient and effective service environment that improves urban quality of life and promotes sustainability." Additionally, the paper discusses some smart city applications, listing 8 different domains where emerging technologies have been deployed to address urban challenges: "healthcare, governance, environment, transportation, energy, safety and security, infrastructure, and education".

Each smart city application domain employs similar technological concepts to impact urban life in distinct ways. By making use of widespread technologies, namely the Internet of Things (IoT), Artificial Intelligence (AI) and big data, real-world data can be collected and processed at a wide scale and with minimal delay, improving the global efficiency of city systems.

IoT sensor networks typically act as the perceptual layer of the smart city, streaming environmental conditions, user behavior, and infrastructure data to cloud or edge platforms, where advanced analytics and AI models generate actionable insights [7]. These edge-to-cloud data pipelines enable real-time monitoring, whereas Machine Learning (ML) models extract patterns that would be infeasible to identify through manual analysis alone, accelerating feedback loops between policy and outcome.

However, the rapid diffusion of sensors introduces privacy, security, and ethical challenges [19]. Large-scale data aggregation raises concerns over surveillance and unauthorized profiling, while heterogeneous device ecosystems increase the cyberattack vulnerability of city infrastructures. Contemporary research therefore advocates for privacy-by-design frameworks, end-to-end encryption, and ML approaches that retain sensitive data at the edge, in order to mitigate these risks without reducing analytical potential.

Equity considerations are also prominent in contemporary debates. Some studies reveal that smart city investments may inadvertently widen socio-economic disparities when unequal access to connectivity or digital literacy prevents certain groups from benefitting fully [10]. Researchers therefore highlight inclusive design principles, subsidized connectivity programs, and community outreach as essential countermeasures.

Smart city development must therefore remain citizen-centered, emphasizing transparency and public participation. Participatory governance mechanisms, ranging from digital forums to real-time feedback applications, are increasingly regarded as critical for aligning technological interventions with community priorities and ensuring that innovation addresses genuine social needs [30].

These emerging technological solutions can provide assistance across all the aforementioned smart city application domains, and each domain should be addressed to achieve an efficient smart city structure. Nevertheless, improvements in the transportation domain are pivotal to improve all others, since mobility networks intersect every other urban subsystem by mediating access to essential services such as healthcare, employment and education.

2.2 Urban Mobility

The concept of urban mobility is often employed to describe the usage of different modes of transportation, ranging from private to public transports, in order to reach a desired destination within a city environment. The implementation and maintenance of these transportation systems requires rigorous governmental planning and policymaking, a process that is often complex, resource-intensive, and time-consuming due to the need to balance technical feasibility, budget constraints, stakeholder interests, and long-term sustainability goals.

With the current trend of urbanization, city spatial footprints continue to expand as its population grows. Consequently, transportation has a key role in enabling city residents to access clustered resources and facilities in their environment [33]. Due to this fact, urban mobility significantly contributes to an improvement in quality of life for city dwellers. Moreover, it promotes an inclusive environment for urban residents by enabling ease of access to education, healthcare and employment facilities.

In spite of all these benefits, the currently implemented urban mobility systems also impose their own set of drawbacks. Urban areas are subject to a myriad of transport-related issues, which range from the increase in air and noise pollution, the reduction of road safety, the intensification of urban congestion, as well as the promotion of oil dependencies [26].

These ramifications of urban mobility can result in serious long-term environmental and health implications. For instance, the increase in greenhouse gas emissions and the reliance in fossil fuels are significant contributors to climate change, while traffic accidents and pollution exposure also lead to an increase in the population's mortality and morbidity rates.

As discussed in section 2.1, smart cities have employed a variety of technological applications in order to mitigate several urbanization repercussions, including in the transportation domain. These smart applications aid mobility by improving the safety and efficiency of urban transportation systems.

Numerous solutions have been proposed in literature, tackling the problem with a multitude of approaches and defining different priorities. Some proposed strategies include, for instance, the integration of IoT devices and sensors to implement a system capable of monitoring and managing the traffic flow, thus addressing some congestion and safety concerns. Another example is the employment of AI and ML algorithms to aid road navigation which can be used in autonomous vehicle applications.

While some urban mobility concerns can be mitigated with the implementation of technological innovations, like the aforementioned systems, a wide range of research works claim that in order to attain a sustainable future in city living, it is necessary to reduce the use of conventional means of transportation, particularly car usage, while also reinforcing environmental concerns in society [18].

2.3 Soft Mobility

Due to the growing need for environmentally conscious transportation alternatives, the concept of soft mobility has gained importance. The term refers to individual and lightweight modes of transport, typically suited for short-distance or last-mile travel. These forms of mobility offer adaptable and low-impact solutions to urban transit challenges, easing congestion and complementing public transport systems. Common examples include walking, cycling, roller-skating, and skateboarding, among others, all of which contribute to a more sustainable and accessible urban mobility landscape.

Besides the clear environmental benefits, opting for these alternative transportation methods contributes to an overall improvement of the urban environment, since it helps minimize the urban mobility repercussions mentioned in section 2.2, such as the levels of air and noise pollution, traffic congestion and road safety issues [25].

Additionally, these alternatives promote a healthier lifestyle, due to the fact that daily exercise is linked to numerous health benefits and is heavily endorsed by medical professionals. Furthermore, the widespread adoption of these alternative transports contributes to a decrease in socioeconomic disparities, since the entry-level price for a soft mobility vehicle is significantly lower than the cost of purchasing and maintaining a car.

Among the variety of soft mobility vehicles available, bicycles have been the most commonly adopted worldwide. Despite a portion of the world's population opting for bicycle use has a greener commuting option, it is indisputable that, while helping their surrounding environment, this increased exposure can compromise the individual safety of cyclists. In fact, in the occurrence of a crash, cyclists are much more likely to suffer severe injuries when compared to car drivers, especially in the event of a crash between a bicycle and an automobile [13].

Unlike conventional transportation methods, soft mobility alternatives offer a myriad of advantages, nonetheless, a significant portion of the population continues to favor traditional means of transport. This reluctance can be attributed to a multitude of factors, ranging from cultural norms and deeply ingrained human habits, to the lack of governmental investment in adequate and safe infrastructures capable of accommodating to these alternative vehicles, such as the integration of bike lanes in urban areas to separate bicycles and similar vehicles from the incoming car traffic, while also keeping them away from the remaining pedestrians.

2.4 Embedded Systems for Smart Cycling

Research highlights that cyclists are considered vulnerable road users [29]. The increased speed and weight of e-bikes, for example, have been associated with increased traffic accident risks [40]. Furthermore, traffic environments are inherently unpredictable due to the sudden actions of drivers, pedestrians, and cyclists, creating a formidable challenge for safety [38].

The hazardous environment to which cyclists are subjected, as mentioned in section 2.3, has a significant impact when determining a predisposition for cycling. The endangerment of human

life associated with cycling in modern cities has become a deterrent toward its wider adoption [11]. Individuals who do not feel safe are less likely to adopt cycling as a daily mode of transport, which in turn leads to negative outcomes in urban mobility [12].

In order to remedy this reluctance to cycle, embedded systems can be used to improve cyclist safety by integrating computation, sensing, and control into bicycles and cycling accessories. These systems enable features such as real-time ride tracking, navigation assistance, performance monitoring, safety alerts, and electric power management for e-bikes. Many wearable devices utilize embedded sensors like accelerometers and gyroscopes, along with embedded software, to perform local processing and analysis [24].

Unlike general purpose gadgets, cycling embedded devices are purpose-built and must operate reliably under mobility constraints such as limited power, variable environments, and the need for user-friendly, safe interaction. Developing these systems involves a multifaceted approach that balances hardware and software considerations, energy efficiency, user interface design, and real-time responsiveness. Development, therefore, requires careful attention to several constraints and concerns unique to the context.

Environmental durability is critical, since devices must withstand vibrations, shocks, and weather exposure (rain, dust, temperature fluctuations) typical of cycling. Moreover, sensor performance can be limited by environmental factors such as weather conditions and temperature levels [29].

Power efficiency is another constant concern. Since most bike systems run on batteries, hardware and software must be designed for low power consumption to prolong runtime. This can involve using low-power components, efficient power regulators, and smart power management (sleep modes, duty cycling sensors when not in use, etc.).

Additionally, device size and weight are also important. Cyclists prefer lightweight gear, so embedded modules should be as compact and light as possible, often requiring high component integration and minimal peripherals.

Safety and reliability are paramount in any transportation-related embedded system. In cycling, this means that the system should not fail in a way that endangers the rider. Thus, developers must prioritize stability and thorough testing. Watchdog timers and fault-tolerant designs might be employed to ensure the device doesn't crash or freeze during operation.

Security is another consideration, especially as bikes become more connected, and an embedded software should guard against unauthorized access or tampering. Network access is a key enabler of cyberattacks, and the deep stack of conventional software (operating systems, libraries) can contain vulnerabilities [32]. Hence, data security is a significant challenge, as devices collecting location or biometric data raise privacy concerns about unauthorized access and use [39].

Furthermore, the user interface and ergonomics influence the design. A cycling device's embedded system often includes some form of user interaction (buttons, LEDs, or a display). The interface must be intuitive and safe to use while riding and this can dictate certain development choices. The system's feedback (beeps, vibrations, lights) should be clear but not distracting. All

these factors (environment, power, safety, UI) need to be balanced during development, making embedded system design a multidisciplinary challenge.

Smart cycling applications can employ a wide range of hardware platforms, chosen according to the required computation capability and energy constraints. On the low end, simple functions can be handled by small microcontrollers. These are typically 8-bit or 32-bit Microcontroller Units (MCU) with modest clock speeds and memory, but extremely low power consumption [36]. Such MCUs can operate for long periods on a small power budget, making them ideal for many IoT applications [31]. These devices might run on a coin-cell battery for months or years by periodically waking to collect and transmit data via low-energy protocols, such as Bluetooth Low Energy (BLE).

However, on the higher end, more complex smart cycling functions require more powerful hardware. These tasks benefit from a faster processor, more memory, and sometimes a dedicated graphics or signal processing unit. For instance, modern bike computers can run on System-on-Chip (SoC) platforms, similar to smartphones in architecture. This kind of platform provides ample computation headroom but draws more power, so it typically relies on a rechargeable battery with a capacity to last a long ride.

Open hardware approaches are also very common in cycling prototype development and research. Developers often start with off-the-shelf boards like an Arduino or a Raspberry Pi, since these platforms come with built-in hardware interfaces and community support, speeding up development. Such open hardware not only lowers development cost but also allows for flexibility, since the same prototype code can often run on upgraded hardware or be adapted to other single-board computers. Nevertheless, these may not be as energy-optimized as custom designs, so eventually a dedicated hardware design might be made for a product version, to minimize power usage and cost.

On the other hand, the software running on cycling embedded systems ranges from simple loop code to full operating systems, depending on complexity. Embedded firmware for very simple devices often runs without an operating system, using a bare-metal approach. Developers write a main program that initializes the hardware and then continuously checks sensors and updates outputs, or they use Interrupt Service Routines (ISR) to handle events.

For somewhat more complex devices that still have tight real-time requirements, such as an e-bike motor controller that must adjust power output within milliseconds, a Real-Time Operating System (RTOS) is often employed. This allows the developer to split functionality into concurrent tasks (threads), with priority levels and timing guarantees. The RTOS scheduler ensures critical tasks run at precise intervals, which would be difficult to manage reliably in a simple loop if many events are happening.

Another common architectural decision is whether the intelligence resides on the main device or is offloaded to an external one. The modern smartphone, with its powerful processor and internet connection, can record the data, display complex maps, or even call for help in an emergency. This approach can reduce the cost and complexity of a cycling device, essentially using the phone as part of the embedded system. The downside, however, is reliance on the rider's phone battery

and availability. Some cyclists might prefer independent devices so as not to drain their phone or to avoid issues like phone screens that are hard to read in sunlight. As a result, high-end dedicated bike computers and wearables still have a market, offering purpose-built software that runs efficiently on dedicated hardware.

One of the key challenges in designing smart cycling systems is balancing performance against energy consumption and component limitations. Every added sensor or function can enhance the cyclist's experience or safety, but it also introduces power and design costs.

Furthermore, hardware modularity or integration must be considered. Modular approaches let each embedded system be simpler and ultra-efficient at its one job, but the user ends up with multiple devices, each with its own battery and cost. An all-in-one system can share resources (one processor for all tasks), potentially leading to a more efficient overall system, but requires a powerful central processor and more complex software. Designers must therefore evaluate which approach yields the best performance for the intended users.

Ultimately, the development of embedded systems for smart cycling is shaped by the balance between cost and utility. High-performance systems with advanced features may appeal to enthusiasts and professional users but are less accessible to casual riders due to their cost and complexity. Conversely, low-cost solutions that provide essential functions with adequate reliability can reach a broader audience and promote widespread use. The low cost of MCUs has been a contributing factor for the expansion of smart cycling applications [31], demonstrating that technology must be economical to be widely deployable.

Therefore, open-source hardware and software play a crucial role in reducing development costs and enabling scalable solutions, making smart cycling technologies more inclusive and sustainable. As the field continues to evolve, the challenge lies in designing systems that deliver meaningful functionality, operate efficiently within tight energy budgets, and remain affordable enough to achieve broad adoption among cyclists.

2.5 State of the Art and Current Challenges

Several proposals have been discussed in the literature, analyzing different ways to improve the overall safety of cyclists based on different criteria. These range from urban planning research [9], used to design and build city infrastructure that accommodates bike use, all the way to smart city applications, like the use of wearable devices that alert the driver about potential dangers.

City infrastructural changes can be a crucial contributor toward citizen's road safety, however, these implementations are heavily dependent on extensive city planning and they are subject to significant costs. On that account, a variety of cycling safety implementations have focused on the current urban limitations to improve the population's well-being and quality of life. One of the prime ways to do this is to take advantage of technology to overcome some real-world restrictions. Previous works in literature showcase the variety of approaches taken to address the vulnerabilities of cyclists in current urban environments.

Steltenpohl et al. [16] introduced the Vibrobelt, a tactile navigation device designed to reduce visual distraction for cyclists. This belt, worn around the waist, delivers directional cues through vibrotactile feedback to indicate the next waypoint and proximity to it. The system allows cyclists to maintain their focus on the road while navigating unfamiliar routes. The evaluation shows that participants using the Vibrobelt are able to acquire spatial knowledge effectively and have fewer errors recognizing environmental cues compared to those using traditional visual navigation systems, although challenges remain in the ability to interpret endpoint cues consistently.

Dancu et al. [5] proposed a gesture bike system, which explores two safety technologies. The first experiment compares the projection of navigation cues onto the road and the use of a bike-mounted head-up display, to assess which one provides better visibility and reduced distraction. The second experiment contrasts two turn signaling approaches. One integrates gesture recognition and projects the output onto the road. The other employs a commercial off-the-shelf signal turning system triggered by button pressing, and the output is displayed through the blinking of arrow-shaped LEDs in the back of the bike along with an audio signal. The study shows that head-up displays are safer and more usable for navigation, while the gesture-controlled projection system is more intuitive and better understood by surrounding traffic than other signaling methods.

Sawitzky et al. [35] addressed the specific hazard of when parked car doors unexpectedly open into the cyclist's path. The system uses a connected infrastructure to notify cyclists of potential risks through awareness messages delivered via a smart helmet interface. The messages are presented in visual, auditory, and voice formats. User studies in a simulated cycling environment demonstrate that multimodal alerts significantly increase the lateral distance from parked cars and improve perceived safety.

Savino et al. [14] developed VeloCity, a mobile application that enables cyclists to report real-time traffic conditions. The application supports three interaction modalities (touch, in-app voice recognition, and using the device's built-in voice assistant) with the goal of identifying the least distracting method for cyclists. The results show that the use of the built-in voice assistant is the preferred and safest option, as it allows for hands-free and eyes-free interaction, minimizing cognitive and visual load while riding.

Peng et al. [17] presented a comprehensive safety service system combining wearable sensing, mobile application support, and backend infrastructure. The system monitors physiological signals (such as heart rate and oxygen saturation), team coordination (sending group separation alerts), and environmental parameters (such as air quality and ultraviolet indexes), while integrating a deep learning model to detect crashes automatically. The developed crash model achieves high accuracy and is designed to update itself based on real-world user feedback to reduce false positives, illustrating the potential of adaptive systems for enhanced cyclist protection.

Andres et al. [21] explored human-computer integration through the use of exertion-aware electric bicycles. This study designs three experimental e-bikes. The first bike increases engine support as the user leans forward, in an attempt to counteract cycling exertion. The second e-bike takes advantage of traffic light pattern data, as well as the user's speed, in order to either increase engine support or warn the cyclist they should slow down, communicating through bone-

conducting headphones, so that this speed regulation enables the cyclist to only catch green traffic lights. Finally, the last bike is designed to integrate the use of an electroencephalography cap that monitors neurological activity to evaluate the driver's peripheral awareness, increasing engine support or slowing down as peripheral awareness increases or decreases, respectively. These experiments emphasize how embedded intelligence can adapt to the context of the rider, enhancing both performance and safety while navigating complex urban scenarios.

Nguyen et al. [34] proposed a smartphone-based model for human activity recognition to classify whether a cyclist is standing, sitting, walking, cycling or falling. This model claims to outperform other traditional classifiers by leveraging time-series sensor data captured via a smartphone. Such models are crucial for detecting abnormal cycling states and triggering alerts or interventions in real-time.

Matviienko et al. [6] developed QuantiBike, a system that quantifies perceived cyclist safety by analyzing head movement patterns. The work combines indoor experiments, employing a virtual reality headset, with outdoor tests using Apple AirPods and a bicycle-mounted iPhone to track the frequency and duration of head movements relative to body position. The results show that increased head movement correlates with reduced perceived safety, particularly at uncontrolled intersections, providing a new modality to understand the emotional and cognitive states of cyclists.

Ferreira et al. [22] designed a low-cost embedded system for urban cyclist safety based on geospatial risk assessment. This solution, built on a Raspberry Pi Zero board, analyzes the proximity to emergency services and assigns a bikeability index to different city routes. This index varies depending on how inconvenient it would be for a cyclist to suffer an accident on that route. Real-time alerts are issued when riders enter high-risk zones, supporting informed route selection and real-time navigation decisions. This study demonstrates the viability of open-source, embedded solutions for urban safety.

Rinaldi et al. [27] conducted a preliminary analysis of sensor fusion data to recognize cyclist gestures. The system uses inertial and electromyography data to detect cyclist's hand signals, used for indicating, for instance, turning or stopping. These gestures are interpreted by a wrist-worn device and relayed to a smart backpack with a visual display, translating cyclist intentions into clearly visible signals for surrounding traffic. This research highlights the value of multimodal sensor integration in reducing miscommunication between cyclists and other road users.

The approach proposed in this dissertation complements the state of the art by introducing a smartwatch-based risk awareness system for cyclists. The system leverages city intersection mapping and assigns a risk score to each node based on how many roads converge to it. The smartwatch then provides color-coded visual cues to inform cyclists about the risk of nearby intersections. In addition, haptic feedback is activated if the user's velocity exceeds a predefined safety speed, alerting the user to slow down and encouraging safe riding behavior. This wearable solution intends to enhance cyclist geospatial awareness by employing an open-source development model, providing a broadly accessible technological solution to improve cyclist safety.

Table 2.1 presents a comparative overview of these cyclist safety studies, together with the proposed approach. Each work is evaluated according to several key criteria:

- **Interaction** — Specifies the primary modality through which the system communicates with or receives input from the cyclist.
- **Input** — Qualitatively rates the complexity of the user input required by the system from low to very high.
- **Platform** — Identifies the platform device used in the implementation.
- **Context Aware** — Indicates whether the system adapts dynamically to external conditions or user context.
- **Prevention** — Refers to the system’s ability to proactively prevent accidents or risky situations before they occur.
- **Detection** — Assesses whether the system can detect incidents or abnormal events after they occur.
- **Risk Model** — Notes the presence of an explicit mechanism to assess environmental or behavioral risk.
- **Open-Source** — Evaluates the reproducibility and affordability of the system, based on its use of open-source software and/or hardware.
- **Scalable** — Reflects on whether the system can be deployed in new settings with minimal dependency on fixed infrastructure or location-specific calibration.

The first three criteria are listed with descriptive text, whereas the remaining six are labeled using intuitive symbols (✓, ~ or ✗) to indicate whether each criterion is present and fully implemented (✓), partially implemented (~) or absent from the study (✗).

The comparative analysis presented in table 2.1 reinforces a key observation raised in chapter 1. Existing technological solutions, while innovative, often fall short in one or more critical areas that hinder their viability for broad, real-world deployment. Many systems focus on isolated functionalities or specific technological innovations but lack the design considerations necessary for accessible, scalable, and context-aware safety support.

For instance, several studies showcase complex interaction mechanisms such as gesture control [5, 27] or multimodal sensing involving physiological or neurological data [21, 17]. Although these demonstrate technical feasibility and open new avenues for cyclist awareness systems, they are constrained by high input complexity and reliance on specialized hardware. Such factors not only increase development and deployment costs but also limit adoption to highly motivated users or controlled environments.

Similarly, context awareness and risk modeling are inconsistently implemented across the surveyed systems. Only a subset of these studies include mechanisms for proactive safety enhancement, either through real-time alerts [35, 22] or adaptive support based on environmental understanding [6]. Even among those that do address contextual information, many rely on fixed infrastructure or external data sources, undermining portability and scalability.

Table 2.1: Comparison of Related Works in Cyclist Safety

Study	Interaction	Input	Platform	Context Aware	Prevention	Detection	Risk Model	Open-Source	Scalable
Vibrobelt: Tactile Navigation Support for Cyclists (2013) [16]	Tactile	Low	Wearable (Belt)	✗	~	✗	✗	~	✓
Gesture Bike: Examining Projection Surfaces and Turn Signal Systems for Urban Cycling (2015) [5]	Visual & Gesture	High	Bike-Mounted Device	✗	✓	✗	✗	✗	✗
“Attention! A Door Could Open.”—Introducing Awareness Messages for Cyclists to Safely Evade Potential Hazards (2021) [35]	Visual & Audio	Medium	Wearable (Helmet)	✓	✓	✗	✓	✗	✗
VeloCity: Using Voice Assistants for Cyclists to Provide Traffic Reports (2021) [14]	Audio	High	Smartphone	✗	✗	✗	✗	✗	✓
Development of a Cycling Safety Services System and Its Deep Learning Bicycle Crash Model (2022) [17]	Multimodal	High	Wearable & Mobile App	✓	~	✓	✓	✗	~
Integrated Exertion—Understanding the Design of Human–Computer Integration in an Exertion Context (2022) [21]	Multimodal	Very High	Wearable & Bike-Mounted Device	✓	✓	✗	✓	✗	✗
Attention Mechanism-Based Bidirectional Long Short-Term Memory for Cycling Activity Recognition Using Smartphones (2023) [34]	None	Low	Smartphone	✗	✗	✓	~	✗	✓
QuantiBike: Quantifying Perceived Cyclists’ Safety via Head Movements in Virtual Reality and Outdoors (2023) [6]	None	Low	Wearable & Smartphone	✓	✗	✗	✓	✗	✗
Enhancing Cycling Safety in Smart Cities: A Data-Driven Embedded Risk Alert System (2024) [22]	Visual	Low	Bike-Mounted Device	✓	✓	✗	✓	✓	~
Preliminary Analysis of Sensor Fusion Dataset for Cyclists’ Gesture Recognition (2024) [27]	Visual & Gesture	Medium	Wearable (Wristband & Backpack)	✗	✓	✗	✗	✗	✗
Proposed Approach (2025)	Visual & Tactile	Low	Wearable (Watch)	✓	✓	✗	✓	✓	✓

Additionally, the scarcity of open-source and scalable solutions within the analyzed works is particularly striking. Most of these studies rely on closed platforms, commercial devices, or prototype-specific setups, which restrict reproducibility and hinder adaptation to new geographic or cultural contexts. Adaptability is critical in the context of urban cycling, where infrastructure, rider behavior, and risk factors vary significantly from one city to another. Without scalable architectures and adaptable algorithms, these systems cannot meet the diverse needs of global urban populations.

In contrast, the approach proposed in this dissertation explicitly addresses these limitations. By adopting a lightweight smartwatch platform and leveraging geospatial risk modeling derived from public map data, the system provides proactive safety cues without reliance on external infrastructure or high-cost components. Its low input complexity, minimal hardware footprint, and modular software design make it both accessible to casual riders and easy to deploy in new settings. Moreover, the use of open-source tools and transparent methodologies aligns with principles of reproducibility and inclusive innovation, enabling broader adoption and community-driven improvement.

By comparing these studies, it is possible to observe that there is a scarcity of cyclist safety systems that combine contextual intelligence, proactive prevention, and practical deployability in a cost-effective and open framework. The proposed system therefore attempts to fill this gap, offering a feasible approach to develop a scalable smart cycling solution that prioritizes user safety without sacrificing accessibility. As such, it contributes meaningfully to the ongoing discourse on urban mobility innovation and provides a foundation for more inclusive and sustainable smart city initiatives.

Chapter 3

Methodology

The developed system is structured around three key components: a mobile web application, a smartwatch application, and a Python-based map generator. Each component is designed to fulfill a distinct role while communicating through standardized data formats and protocols. The mobile application serves as a sensor interface, acquiring GPS data from the smartphone and transmitting it to the smartwatch via Bluetooth Low Energy (BLE). The smartwatch acts as the main computational unit, processing the received location data in order to display the overall risk and alert the cyclist if needed.

The overall algorithmic structure allows for modular development and efficient execution on constrained embedded hardware. By offloading only sensor acquisition to the mobile phone and reserving all computation for the smartwatch, the system maintains independence from the mobile platform and supports future migration to fully self-contained wearable implementations.

3.1 System Architecture

The system developed in this research is designed to deliver cyclist safety feedback through a distributed and modular architecture that leverages affordable, open-source technologies. Central to this architecture is a wearable device responsible for processing spatial risk data and providing multimodal alerts, complemented by a smartphone that supplies critical geolocation input.

It is important to highlight that the decision to rely on a smartphone for geolocation input stems from both practical and economic considerations. As most cyclists already carry a smartphone during their rides, leveraging its built-in GPS capabilities allows the system to avoid redundant hardware components and maintain a low-cost profile. This approach not only preserves the accessibility and open-source principles of the proposed solution but also simplifies integration into cyclists' existing routines without requiring dedicated infrastructure or modifications to the bicycle itself.

This section details the rationale behind the selection of both the smartwatch and smartphone components, highlighting their complementary roles within the system. It presents the hardware characteristics that make the smartwatch suitable for wearable risk assessment, and justifies the use

of a smartphone as a geolocation source. Together, these elements form a modular architecture that relies on off-the-shelf components and efficient communication protocols to balance performance, user comfort, and accessibility, while remaining adaptable to diverse urban environments.

3.1.1 Smartwatch

The smartwatch device selected for the execution of this project is the LILYGO T-Watch S3. The motivation for this hardware choice is due to its exceptional balance of affordability, modularity, and suitability for an open-source development context.



Figure 3.1: LILYGO T-Watch S3 [1]

Priced at approximately 40 USD, the device offers an accessible entry point for users and researchers, while providing full transparency through its publicly released firmware, schematics, and development libraries. These characteristics align well with the open-source ethos of this research project and ensure a sustainable, modifiable foundation for iterative system improvements.

A significant advantage of the T-Watch S3 lies in its adoption of the widely used ESP32 microcontroller, common across numerous development boards, while presenting it in a wearable form factor. This design ensures that the device remains both non-intrusive and practical for cyclists, enabling real-time safety feedback during riding while minimizing distraction. The open-source development environment, combined with accessible hardware, makes the T-Watch S3 not only an effective platform for this research but also a replicable and customizable solution that aligns with the goals of transparent and sustainable innovation.

Internally, the T-Watch S3 utilizes the aforementioned ESP32-S3 dual-core MCU, enhanced by 8 MB of PSRAM and 16 MB of flash storage. This hardware configuration supports the real-time execution of spatial risk-assessment algorithms and signal processing routines. Additionally, the device incorporates a 1.54 inch capacitive touch LCD, a BMA423 accelerometer, real-time clock, microphone, speaker amplifiers, and a DRV2605 haptic motor, all of which enable the multimodal

interactions (visual and haptic alerts), central to the cyclist safety application. Wi-Fi, Bluetooth 5.0, and LoRa (SX1262) connectivity further enhance its adaptability for diverse data-exchange and field-testing scenarios.

Despite its comprehensive feature set, the absence of integrated GPS hardware in the T-Watch S3 imposes a significant challenge. This limitation is of particular concern, since accurate geolocation forms the backbone of the system's operational logic, as real-time coordinates are required for both intersection risk evaluation and alert timing.

To address this restriction, the proposed implementation integrates smartphone sensors as an external GPS source. The mobile device is paired to the smartwatch via Bluetooth Low Energy (BLE) and it relays geolocation data in real time to the watch, thereby bridging the gap created by the watch's hardware constraints.

This solution inevitably increases system complexity, both in terms of software architecture (BLE communication between potentially heterogeneous devices) and workflow (application installation and pairing). However, the use of a smartphone device to compensate for the hardware constraints, preserves the open-source and cost-effective nature of this research, since smartphones are already owned by the vast majority of users, and its inclusion in the system will incur no additional expenses.

3.1.2 Smartphone

The smartphone functions as a versatile auxiliary element in the proposed system, temporarily filling the hardware gap left by the smartwatch's lack of integrated geolocation capabilities. Its inclusion is grounded in both technical pragmatism and user accessibility, since modern smartphones are ubiquitously available, and equipped with high-accuracy GPS receivers and Bluetooth Low Energy (BLE) support across nearly all mid-range and flagship devices. These embedded capabilities allow the system to maintain real-time location tracking and wireless communication without requiring dedicated hardware additions.

Importantly, the smartphone acts purely as a sensor and communication interface rather than a computational core, ensuring minimal resource demands. The application developed for this research is platform-agnostic and can be deployed to both Android and iOS environments, taking advantage of cross-platform development frameworks that abstract device-specific functionality. This design ensures that the system remains accessible regardless of the user's operating system or device manufacturer, and it aligns with the broader goals of affordability, inclusivity, and wide-scale replicability.

From a development standpoint, the smartphone contributes only transient data to the system, namely, time-stamped geolocation updates and connection status metadata. These are relayed over BLE using widely supported protocols, allowing for stable communication even on devices with modest hardware specifications. The modularity and platform independence of the mobile component reinforce the flexibility of the system as a whole, enabling rapid adaptation and deployment across different urban contexts and user groups without introducing significant barriers to entry.

3.2 Proposed Approach

In light of the hardware constraints imposed by the selected smartwatch, the proposed system architecture adopts a modular and distributed approach to ensure robust performance and maintain design objectives related to affordability and user accessibility. The system architecture can be divided in three parts, as depicted in Figure 3.2, labeled as preprocessing, smartwatch and smartphone for simplicity. Each module is structured to fulfill a specific role in the acquisition, processing, and communication of cyclist safety data. The source code developed for the project is available in a GitHub repository [23].

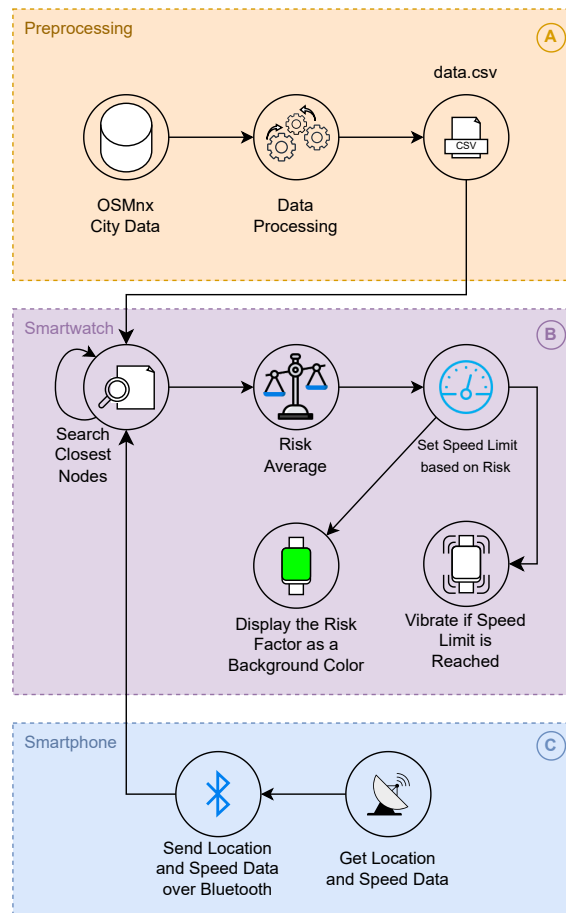


Figure 3.2: Simplified System Architecture

The system begins with a preprocessing stage (Stage A), executed prior to deployment. This stage utilizes urban street network data acquired through the OSMnx library, which extracts and formats geospatial information from OpenStreetMap (OSM).

This data is then processed to generate a structured dataset, stored as a CSV file, containing the coordinates of the city intersections, as well as the risk factor assigned to each node. The resulting

CSV file forms a lightweight yet information-rich spatial database that can be stored locally on the smartwatch's flash memory.

Once operational, the system shifts to its runtime configuration, composed of the smartwatch and the paired smartphone. The smartphone (Stage C) is responsible for acquiring real-time location and speed data using the onboard sensors in the device's GPS module. This data is then transmitted to the smartwatch via Bluetooth Low Energy (BLE), providing the positional input necessary for real-time risk assessment.

Upon receiving the location data, the smartwatch (Stage B) initiates a spatial query to identify the closest intersection nodes within the preloaded CSV dataset. It then computes a weighted average risk score based on the proximity and associated risk factors of nearby nodes.

This weighted value is used to dynamically set a recommended maximum speed for the cyclist. To clarify, if the user's velocity exceeds the speed limit, the system triggers a haptic alert using the device's vibration motor. By taking advantage of the weighted risk average, different speed limits can be set according to how dangerous the surrounding area is.

Concurrently, the weighted risk level is shown visually through a color-coded background on the watch's display, providing the cyclist with a non-intrusive and continuous indication of the environmental risk.

The modular nature of this architecture ensures adaptability to varying urban contexts, as the preprocessing stage allows the system to be reconfigured for different cities or neighborhoods. Moreover, by leveraging the processing capabilities of the ESP32-S3 microcontroller and minimizing reliance on external cloud infrastructure, the system remains lightweight, responsive, and fully operable in offline conditions.

The proposed approach, therefore, distributes responsibilities strategically between mobile and wearable components, making efficient use of available hardware while preserving the open-source and cost-effective principles central to this research. The integration of BLE communication, local risk evaluation, and multimodal feedback mechanisms forms a cohesive architecture aimed at enhancing cyclist safety through context-aware alerts.

3.3 Preprocessing: City Map Generator

A core component of the cyclist safety system is the ability to generate city-specific geospatial datasets that quantify intersection risk based on structural network properties. The city map generator (Stage A) is designed to fulfill this role by retrieving, processing, and exporting simplified representations of urban street networks. It leverages open-source tools to automate the acquisition of OpenStreetMap (OSM) data, consolidate redundant intersection nodes, and compute a normalized risk factor that captures intersection complexity. This processed data is then exported in formats suitable for both analytical visualization and embedded deployment. This section therefore describes the software dependencies, data processing logic, risk estimation methodology, and overall algorithm structure underpinning this component.

3.3.1 Python Packages

The development of a city map generator component relied on a range of Python packages that together facilitated the acquisition, processing, analysis, and visualization of geospatial data. Each package was selected for its robustness, open-source availability, and suitability for integration within an automated data pipeline.

The combination of geographic data extraction, data transformation, interactive visualization, and risk quantification required a cohesive and extensible set of tools. Hence a short description is provided, detailing the primary Python packages employed in the implementation of the preprocessing workflow, which ultimately generates the CSV file used by the cyclist safety system.

- **branca** — A library for generating HTML and JavaScript elements via Jinja2 templates. Originally extracted from folium to host common, non-map-specific features, it supports programmatic creation of interactive components such as pop-ups, legends, overlays, and color maps.
- **folium** — A Python interface to Leaflet.js that enables the creation of interactive web maps. It leverages branca as its underlying rendering engine and facilitates the embedding of geospatial layers, tile maps, markers, and dynamic visual styles on maps, supporting a user-friendly integration between geospatial data and web interaction.
- **matplotlib** — A widely adopted 2D plotting library providing both a procedural pyplot interface and an object-oriented API. It supports the generation of static visualizations, including line charts, histograms, and scatter plots, which are suitable for scripts, interactive notebooks, and graphical interfaces.
- **osmnx** — A specialized toolkit for downloading, modeling, analyzing, and visualizing street networks and other geospatial features from OpenStreetMap (OSM). Built on NetworkX and GeoPandas, the package automates the creation of graph representations of urban infrastructure, corrects network topology, computes routes and metrics, and enables export of geospatial data.
- **pandas** — A library offering labeled tabular data structures and comprehensive data-manipulation tools. It underpins data ingestion, cleaning, transformation, aggregation, and export processes, in coordination with numerical libraries such as NumPy.
- **scikit-learn** — An open-source library implementing a unified interface for machine learning algorithms, covering classification, regression, clustering, dimensionality reduction, and model evaluation. It offers preprocessing utilities and estimator APIs suitable for medium-scale analytical workflows and integrates seamlessly with NumPy and pandas.

3.3.2 Implementation

The technical implementation of the preprocessing pipeline transforms raw geospatial data into a structured dataset that supports intersection-level risk estimation for cyclist safety. The process is

designed to extract and simplify street network information from OpenStreetMap, normalize and refine intersection attributes, and produce a compact output suitable for real-time use in embedded systems. Key operations include network retrieval, intersection consolidation, risk factor computation, and structured data export. These stages are organized into a modular and configurable architecture that enables efficient processing and adaptability across different urban contexts.

3.3.2.1 Map Data Acquisition and Processing

The first step in generating the spatial dataset involves retrieving the street network of a target location using the OSMnx Python library. As mentioned in subsection 3.3.1, this package provides access to OpenStreetMap (OSM) data and enables the automatic construction of a topologically correct graph representation of urban infrastructure.

However, mapping all nodes present in the retrieved street network often results in a cluttered and oversampled representation. In dense urban areas, multiple nodes may exist in close proximity, particularly where divided roads or roundabouts are represented with multiple centerlines and connectors. Additionally, nodes at the ends of dead-end streets are included in the raw network, even though they do not represent true intersections. These characteristics introduce redundancy and hinder the interpretability and scalability of the resulting spatial dataset.

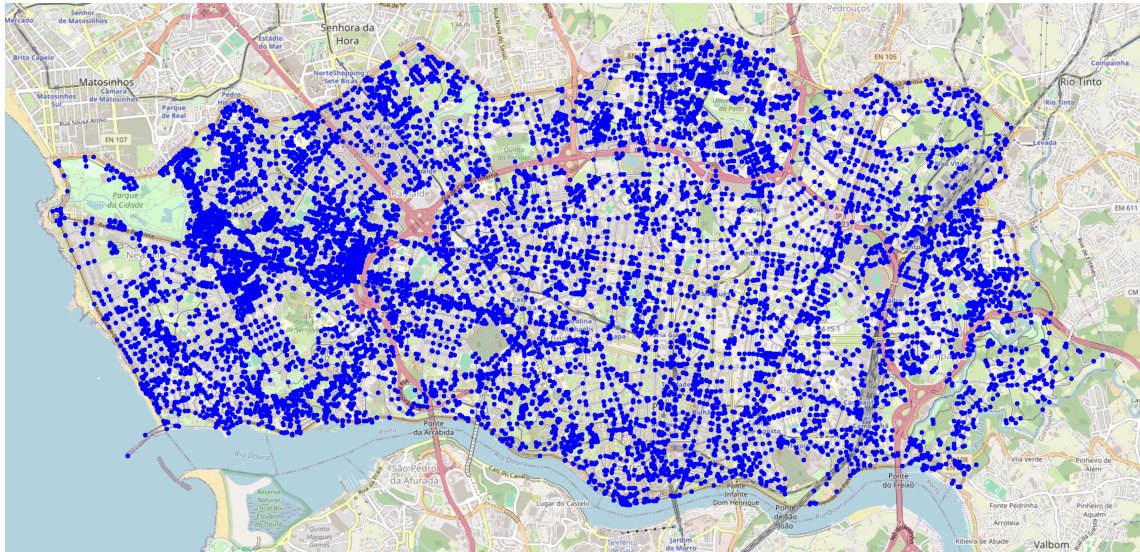


Figure 3.3: Generated Map of Porto, Portugal - Every Node is Plotted

To address these challenges, the preprocessing pipeline incorporates a node consolidation step using a function provided by the OSMnx documentation. Not only does this function discard dead-end nodes, but it also applies a topological algorithm to merge clusters of topologically close nodes, returning a cleaner representation of true street intersections [8].

The function identifies connected clusters of nodes and replaces them with single representative centroids. The radius within which nodes are considered part of the same cluster can be

adjusted and, in fact, this value should be tuned according to the typical street width or design standards in the area of interest.

To clearly visualize the impact of the consolidation procedure, Figure 3.4 presents multiple output graphs generated with varying consolidation radius values. For each case, the number of remaining nodes is recorded and compared to the total number of nodes in the original graph. This allows for the calculation of a consolidation ratio, defined by Equation 3.1, which quantifies the proportion of retained nodes after simplification.

$$\text{consolidation_ratio} = \frac{\text{consolidated_node_count}}{\text{original_node_count}} \quad (3.1)$$

A lower consolidation ratio implies that the map will generate a smaller file size, but it may also entail a loss of spatial precision. The appropriate consolidation radius value must therefore be defined, in order to balance visual clarity, file size reduction, and geometric fidelity.

To promote usability and adaptability, the codebase exposes the location and consolidation radius parameters as configurable options in the CLI. This design allows users to quickly adjust the target area and the granularity of intersection consolidation, facilitating the generation of tailored spatial datasets for different urban environments. By integrating flexible parameterization with robust preprocessing tools, the system ensures both technical accuracy and operational convenience in the preparation of cyclist safety data.

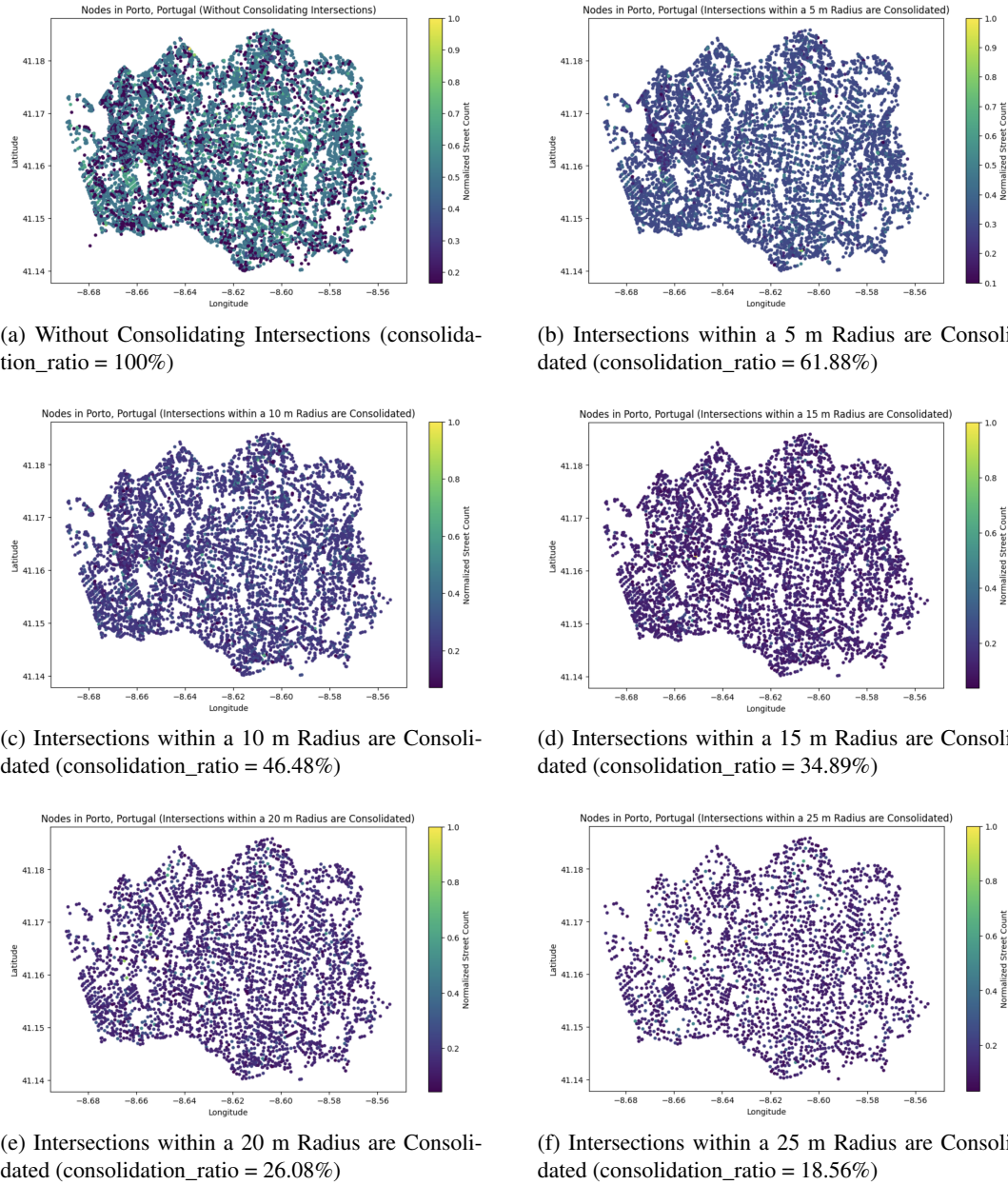


Figure 3.4: Maps of Porto, Portugal, Displaying the Effects of Consolidating Intersections and Respective Consolidation Ratio

3.3.2.2 Intersection Risk Factor Estimation

In order to categorize how dangerous an intersection is, a risk value must be estimated. In this study, this estimation is based on the assumption that the complexity and potential danger of an intersection will increase with the number of streets that converge upon it. Intersections with a high number of connecting streets tend to involve more possible vehicle and cyclist trajectories, making them more difficult to navigate and more likely to be associated with conflict points. Thus, in this research the number of streets entering each intersection is employed as the primary proxy for risk estimation.

The OSMnx library facilitates this process by directly providing a street count attribute for each node in the processed graph, however, it is still necessary to ensure these values are comparable and interpretable. Hence, these street count values are rescaled using a max absolute normalization strategy, converting all values to the interval $[0,1]$. This results in a continuous risk factor where the values closer to 0 represent the intersections with a lower risk, and the values closer to 1 correspond to the more complex intersections in the dataset.

To assess the statistical properties of the normalized data, a frequency histogram, cumulative histogram, and corresponding color map are generated, as shown in Figure 3.5. These visualizations reveal that the distribution of normalized street count is heavily right-skewed, with the vast majority of intersections concentrated in the lower end of the scale. This limits the discriminative power of the risk factor, as most nodes receive low or indistinguishable values.

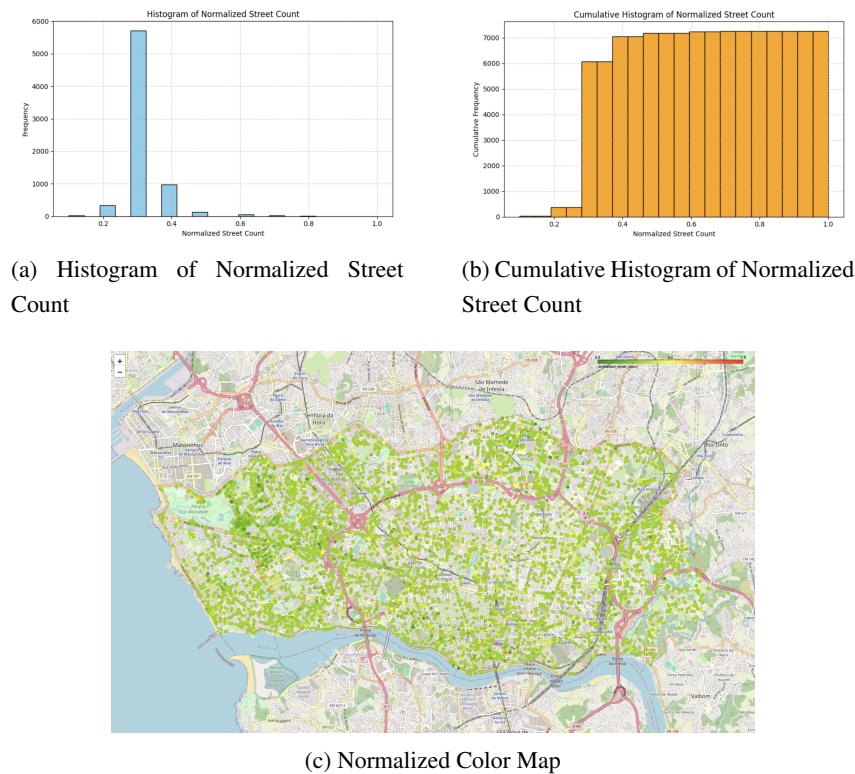


Figure 3.5: Normalization Results

To improve the uniformity of the risk distribution and better differentiate intersections across the full scale, a quantile transformation is applied to the normalized values. A quantile transformer is a statistical preprocessing technique that transforms the input variable into a uniform or normal distribution by mapping the quantiles of the original distribution to a target distribution. In this case, the transformation targets a uniform distribution and divides the data into discrete intervals (quantiles) with equal representation. This process is commonly used to improve interpretability and ensure consistent visual gradation when dealing with skewed data.

Figures 3.6 and 3.7 illustrate the results of applying the quantile transformation with three and five quantiles, respectively. In both cases, the transformation yields a more balanced distribution, as seen in the reshaped histograms and color maps. Compared to the initial normalization, the transformed datasets exhibit clearer segmentation across the map and more meaningful visual distinctions between regions of differing intersection complexity.

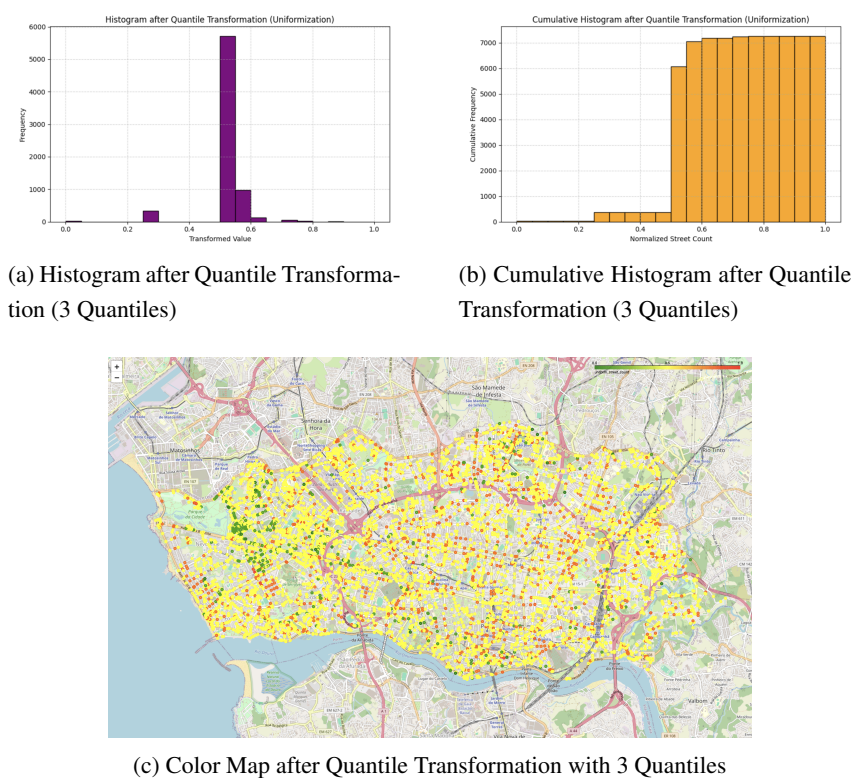


Figure 3.6: Uniformization Results using 3 Quantiles

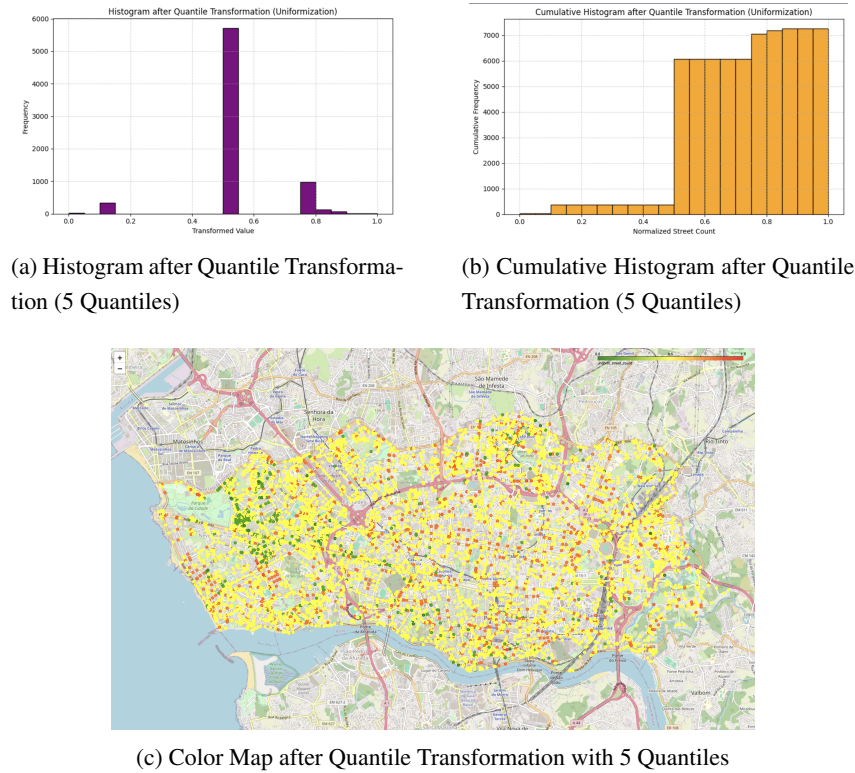


Figure 3.7: Uniformization Results using 5 Quantiles

After comparative analysis of both transformations, the version using 5 quantiles is selected for use in the final dataset. This choice is supported by the improved visual balance achieved in the resulting color map, which better captures variation in intersection complexity across the urban environment. The quantile-transformed street count values are thus adopted as the intersection risk factor exported in the CSV file.

3.3.2.3 Algorithm Structure

The city map generator is implemented as a modular pipeline that sequentially processes geospatial data to produce a simplified and risk-aware dataset suitable for embedded use in the cyclist safety system. Figure 3.8 illustrates the logical structure of the algorithm, from input parsing to data export and visualization.

The process begins in the Command Line Interface (CLI), where the user provides two key arguments: the name of the location to be processed and the consolidation radius. These parameters govern the retrieval of the street network and the degree of node simplification applied.

In the subsequent node data extraction stage, the OSMnx library is used to download the full street network graph corresponding to the specified place. If the consolidation radius is greater than zero, the algorithm applies the node merging process using the consolidate intersections function. Otherwise, node merging is skipped. After this step, the node set is converted into

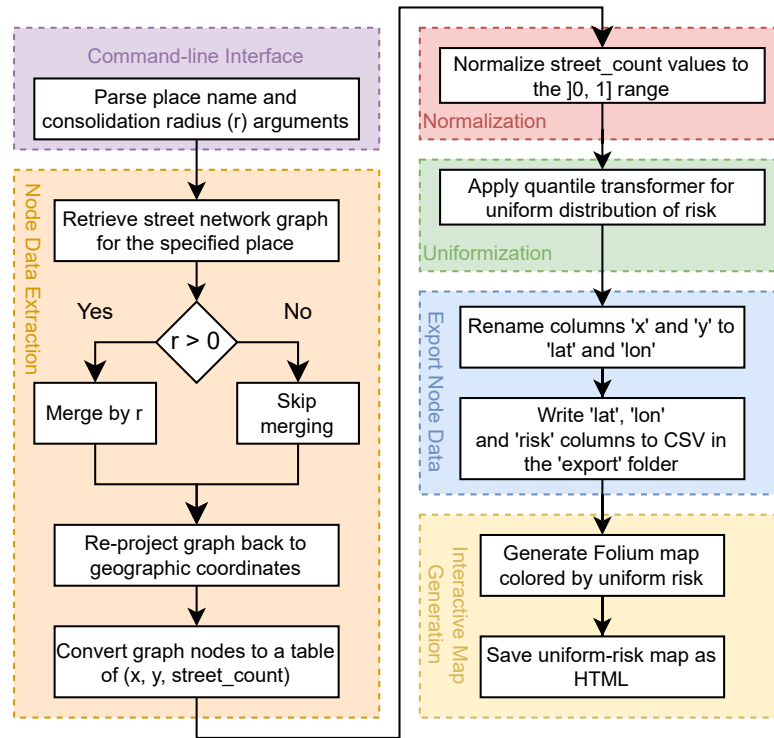


Figure 3.8: City Map Generator Flowchart

a structured table containing the geographic coordinates (latitude and longitude) and the street count attribute, for each node.

The resulting table is then passed through a normalization stage, where the street count values are scaled to the $[0,1]$ interval. However, due to the observed skewness in the distribution of normalized values, a uniformization step is applied next. This involves the use of a quantile transformer that redistributes the risk values such that the final data exhibit a more uniform distribution. This transformation enhances interpretability by improving the visual and numerical contrast between nodes of differing risk levels.

Following uniformization, the export phase prepares the dataset for downstream use. The coordinate columns are renamed to lat and lon, and a reduced table containing only the lat, lon, and transformed risk values is written to a CSV file in the designated export directory. This output is lightweight and optimized for efficient parsing on resource-constrained devices such as the smartwatch.

As an optional final step, the generator also produces an interactive HTML map using the Folium library. This map visualizes the node data using a color gradient corresponding to the uniformized risk factor and provides a graphical validation tool for verifying the effectiveness of the transformation process. The map is saved as an HTML file and serves as both a development aid and a potential standalone visualization asset.

This structured, modular architecture enables a high degree of adaptability, allowing users to

generate city-specific spatial datasets with minimal configuration. It also ensures that the resulting risk factor data are consistent in format and semantics, supporting seamless integration with the downstream components of the cyclist safety system.

3.4 Smartwatch Application

The smartwatch component (Stage B) serves as the embedded runtime environment responsible for executing the cyclist safety logic in real-time. Functioning as a self-contained co-processor, the smartwatch continuously monitors geolocation data, assesses proximity to hazardous intersections, computes risk estimates, and delivers feedback through visual and haptic modalities.

This section details the technical composition of the application, including the development tools used, the design and behavior of the proximity analysis algorithm, and the structure of the core execution loop. Emphasis is placed on the system's responsiveness, resource efficiency, and compatibility with the constraints of wearable hardware.

3.4.1 Development Tools

The development of the smartwatch application relied on a combination of manufacturer-provided resources and open-source frameworks that facilitate embedded system programming for the ESP32 platform. These tools were selected based on their compatibility with the hardware, their support for wireless communication protocols, and their integration with modern development environments.

Central to the implementation process is the open-source library maintained by the device manufacturer [37]. This library provides a high-level abstraction over the T-Watch S3's core hardware components, including drivers and initialization routines for the capacitive touch display, accelerometer (BMA423), vibration motor (DRV2605), real-time clock, and power management units. By encapsulating low-level device-specific functionality, the library accelerates the application development process and ensures compatibility with the hardware's intended usage patterns.

This library is built on top of the Arduino-ESP32 framework [4], an open-source project maintained by Espressif Systems that extends the Arduino ecosystem to support ESP32 microcontrollers. The framework includes board support packages, hardware abstraction layers, and communication stack implementations for Wi-Fi, Bluetooth Classic, and Bluetooth Low Energy (BLE).

Of particular relevance to this project are the Bluetooth functionalities provided by the framework, which facilitate advertising, service definition, and data exchange between the smartwatch and the companion smartphone application. The use of this framework ensures a stable and well-documented environment for integrating BLE communication into the system's runtime behavior.

In order to manage the build process and dependency resolution, the PlatformIO ecosystem is employed. PlatformIO is an open-source, cross-platform toolchain designed for embedded development that supports a wide range of boards, frameworks, and environments. It offers integrated

project management, automated library resolution, and seamless compilation workflows within both command-line and Integrated Development Environments (IDE) such as Visual Studio Code. The use of PlatformIO enables reproducibility, modular code organization, and efficient firmware deployment to the T-Watch S3 device.

Together, these tools form a comprehensive development environment that streamlines the process of implementing, testing, and deploying the smartwatch firmware. Their open-source nature aligns with the principles of transparency and extensibility that underpin this research, while their technical features ensure compatibility with the target hardware and communication requirements.

3.4.2 Implementation

The proposed implementation for the smartwatch device integrates geospatial preprocessing, proximity-based risk computation, and multimodal feedback into a structured firmware pipeline. Each component of the application, from data acquisition to runtime analysis and haptic alert delivery, is implemented with attention to the computational limitations of embedded hardware. This subsection details the design and execution logic of the key modules, supported by representative flowcharts, algorithmic strategies, and relevant mathematical formulations.

3.4.2.1 Proximity Analysis

A critical task within the smartwatch application involves identifying which intersection nodes are geographically closest to the user's current location. This proximity-based selection is fundamental in order to enable risk estimation and feedback, as the computed risk value depends on the spatial characteristics of nearby intersections. The implemented algorithm achieves this by iterating through a structured dataset of geolocated intersections (preprocessed and stored locally on the smartwatch) and calculating the geodesic distance between each node and the current user position.

To compute this distance with sufficient accuracy and efficiency, the system employs the Haversine formula, a well established method for calculating great-circle distances between two points on the surface of a sphere. This method is particularly appropriate for global geospatial computations due to its simplicity and numerical stability across a wide range of coordinate values. The employed haversine formula is represented in Equations 3.2 to 3.6.

$$\Delta\phi = (lat_2 - lat_1) \frac{\pi}{180} \quad (3.2)$$

$$\Delta\lambda = (lon_2 - lon_1) \frac{\pi}{180} \quad (3.3)$$

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\left(\text{lat}_1 \frac{\pi}{180}\right) \cos\left(\text{lat}_2 \frac{\pi}{180}\right) \sin^2\left(\frac{\Delta\lambda}{2}\right) \quad (3.4)$$

$$c = 2 \operatorname{atan2}(\sqrt{a}, \sqrt{1-a}) \quad (3.5)$$

$$\text{distance} = R \cdot c, \quad R = 6371 \text{ km} \quad (3.6)$$

The algorithm reads the spatial data file in batches to accommodate the limited memory of the smartwatch. For each intersection entry in the dataset, the coordinates are parsed, and the haversine distance to the user's current position is calculated. The system maintains a temporary buffer of the closest entries identified during the scan. Once the entire file has been processed, the buffer is sorted to yield the final list of nearest nodes. This list is then used for subsequent risk factor interpolation and alert generation.

To ensure responsiveness and compatibility with real-time constraints, the node searching routine is designed to operate incrementally. The file is read in timed batches, and control is periodically yielded to other system components to avoid interrupting critical tasks such as Bluetooth communication or user interface updates. Upon completion of the search, the results are flagged as ready for use by the risk evaluation module.

This approach enables scalable and accurate proximity analysis on a constrained embedded device, while preserving compatibility with the data format and spatial model established during the preprocessing phase. By integrating a well-established geospatial formula with an optimized batch-processing mechanism, the system efficiently identifies the most relevant intersection nodes for risk assessment at runtime.

3.4.2.2 Proximity Analysis Algorithm Structure

The proximity analysis subsystem implemented in the smartwatch firmware is responsible for determining the closest georeferenced entries from a stored dataset relative to the current location. To ensure robustness and efficiency in a resource-constrained environment, this component follows a staged execution model with state monitoring, validation steps, and interruptible batch processing. Figure 3.9 provides an easily interpretable representation of the algorithm's behaviour.

The flow begins with a conditional check on the execution state. If the system is in an idle, complete, or error state, no further action is required, and the procedure terminates. Otherwise, the process proceeds to verify the operational context, specifically determining whether it is in an initial processing phase. If this condition is met, the dataset of georeferenced location records is accessed from internal storage.

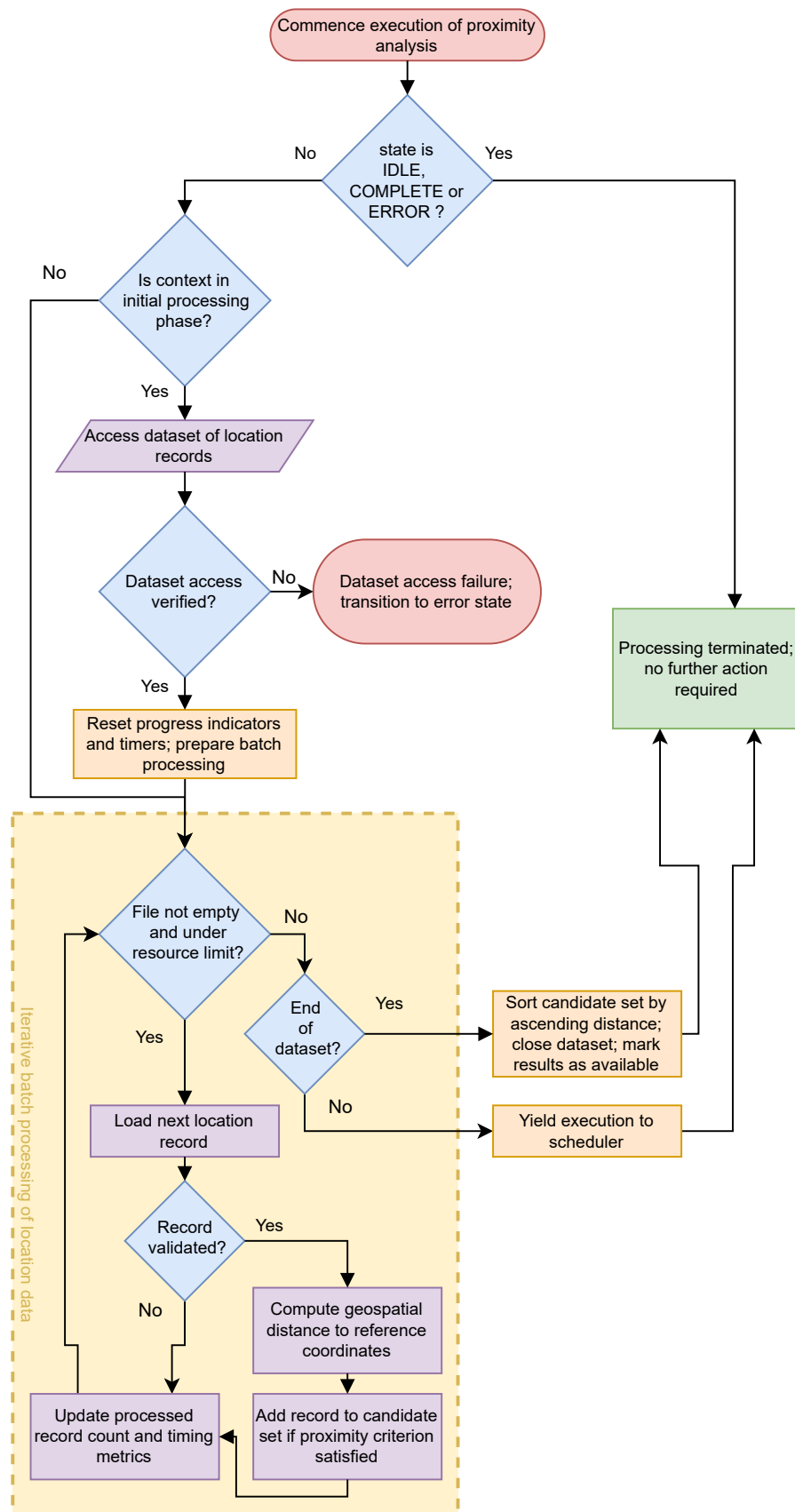


Figure 3.9: Proximity Analysis Flowchart

Upon accessing the dataset, a validation step is executed to ensure that the file is accessible and not corrupted. If the verification fails, the system transitions to an error state. If successful, progress indicators and timing variables are reset in preparation for iterative processing.

The core of the proximity analysis is structured as a batch loop, which iteratively reads and processes entries from the dataset while monitoring system resource usage. During each iteration, the algorithm checks whether the dataset is non-empty and that resource constraints (e.g. memory, CPU time) are not violated. If these conditions are satisfied, the next location record is read from the file.

Each retrieved record is subjected to a validation procedure. Invalid or malformed records trigger an update to internal counters and timing diagnostics, after which the loop proceeds to the next entry. For valid records, the geospatial distance to a reference location (typically the current GPS position) is computed. If the resulting distance satisfies the predefined proximity criterion, the record is added to a candidate set for further consideration.

The iteration continues until the dataset is exhausted or processing must pause due to resource limitations. Once the end of the dataset is reached, the accumulated candidate set is sorted in ascending order by distance. The dataset file is closed, and the results are flagged as available for downstream modules. If the processing loop is interrupted before reaching the end, control is yielded back to the task scheduler to allow for system responsiveness.

This structured approach enables reliable geospatial proximity analysis while accommodating the operational constraints of smartwatch hardware. The state-driven control flow, coupled with validation and yield mechanisms, ensures both correctness and responsiveness in dynamic runtime conditions.

3.4.2.3 Core Execution Loop

The smartwatch application's central processing logic is designed to evaluate cyclist safety conditions by analyzing spatial risk factors and issuing timely alerts. This module integrates geolocation reception, asynchronous proximity analysis, visual and haptic feedback, and Bluetooth Low Energy (BLE) communication in a unified structure. Additionally, to enable real-time performance and system responsiveness, the implementation leverages the dual-core nature of the ESP32-S3 microcontroller and is orchestrated through the FreeRTOS task scheduler.

Incoming geolocation data are received via BLE from a paired mobile device and parsed into usable coordinates. To avoid unnecessary computation, the system first verifies that the location has meaningfully changed since the last update. If confirmed, a non-blocking proximity search is triggered to identify the closest intersections from a preloaded local dataset stored in the internal flash using the FAT filesystem. This operation is delegated to a dedicated task running on one of the cores, ensuring that it does not interfere with other critical operations such as screen updates or BLE communication.

The proximity analysis computes a geospatially weighted risk factor based on the distances and corresponding risk scores of nearby nodes. The contribution of each node is determined by an inverse distance weighting mechanism, as shown in Equation 3.7. Each node's risk is scaled

by this distance weight and accumulated (Equation 3.8), while the sum of all inverse distance weights is also calculated (Equation 3.9) and finally the system computes a normalized average weighted risk (Equation 3.10). The resulting value ranges between 0 and 1 and encapsulates both the intensity and proximity of surrounding risk sources.

$$invDist_i = \begin{cases} \frac{1}{d_i}, & d_i > 0.01 \\ 100, & otherwise \end{cases} \quad (3.7)$$

$$\sum weightedRisk = \sum_i (risk_i \times invDist_i) \quad (3.8)$$

$$\sum invDist = \sum_i invDist_i \quad (3.9)$$

$$avgWeightedRisk = \frac{\sum weightedRisk}{\sum invDist} \quad (3.10)$$

Additionally, this average weighted risk value is used to dynamically derive a recommended speed limit for the cyclist. In order to calculate this dynamic velocity limit, two transformation strategies have been evaluated. The first approach applies a linear transformation that maps the entire risk range onto a corresponding speed range by employing Equation 3.11. This method continuously adjusts the speed recommendation in proportion to the risk factor.

$$v_{limit}(x) = Max_Speed - x (Max_Speed - Min_Speed) \quad (3.11)$$

Alternatively, the second approach proposes a stepped transformation, where the risk range is divided into n discrete bins, as show in Equations 3.12 and 3.13.

$$bin(x) = \lfloor x \times n \rfloor \quad (3.12)$$

$$v_{limit}(x) = Min_Speed + (Max_Speed - Min_Speed) \frac{bin(x) + 1}{n} \quad (3.13)$$

Figure 3.10 illustrates the behavior of both strategies. The transformation chosen to employ is the stepped model, due to its ability to suppress sensitivity to marginal changes in the input

and produce more stable outputs, aligning better with the expected use case and enhancing user interpretability.

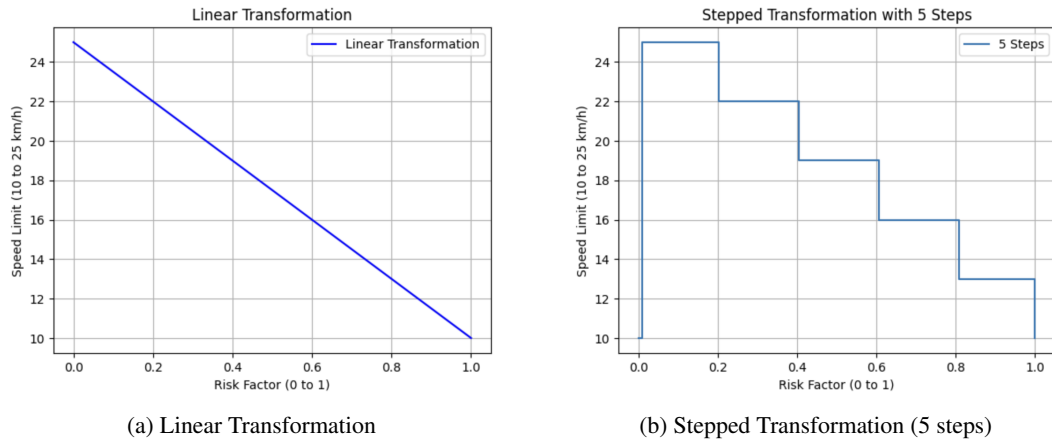


Figure 3.10: Contrast between Linear and Stepped Transformations

If the cyclist's current speed exceeds the calculated limit and a high-risk intersection is detected within a predefined alert radius, a haptic feedback alert is issued via the watch's vibration motor. This response is triggered by a secondary FreeRTOS task dedicated exclusively to managing tactile notifications, thereby avoiding blocking operations in the main loop.

Ensuring a reliable scheduling architecture is the primary challenge in the development of this system. The design has to support concurrent data acquisition, BLE communication, continuous spatial analysis, and visual/haptic feedback without overloading memory or exceeding processing limits.

The proposed solution involves distributing responsibilities across two cores: one primarily for BLE and UI handling, and the other for continuous geospatial batch processing. This decoupling, combined with tight memory control and asynchronous task execution, enables the system to maintain responsiveness while continuously analyzing proximity risks.

Through this architecture, the module effectively enables the smartwatch to serve as an autonomous safety co-processor, capable of processing high-frequency location updates, computing actionable risk metrics, and delivering context-aware guidance to enhance cyclist safety.

3.4.2.4 Core Execution Loop Algorithm Structure

The algorithm governing the smartwatch application is organized around a modular and task-parallel execution model, with clearly delineated responsibilities across asynchronous components. The structure is designed to ensure continuous responsiveness to location updates while maintaining a non-blocking user interface and feedback mechanism. The flowchart in Figure 3.11 provides an overview of this logic.

Execution begins with the initialization procedure, during which the system sets up essential hardware and software resources. This includes configuring the display, mounting the internal

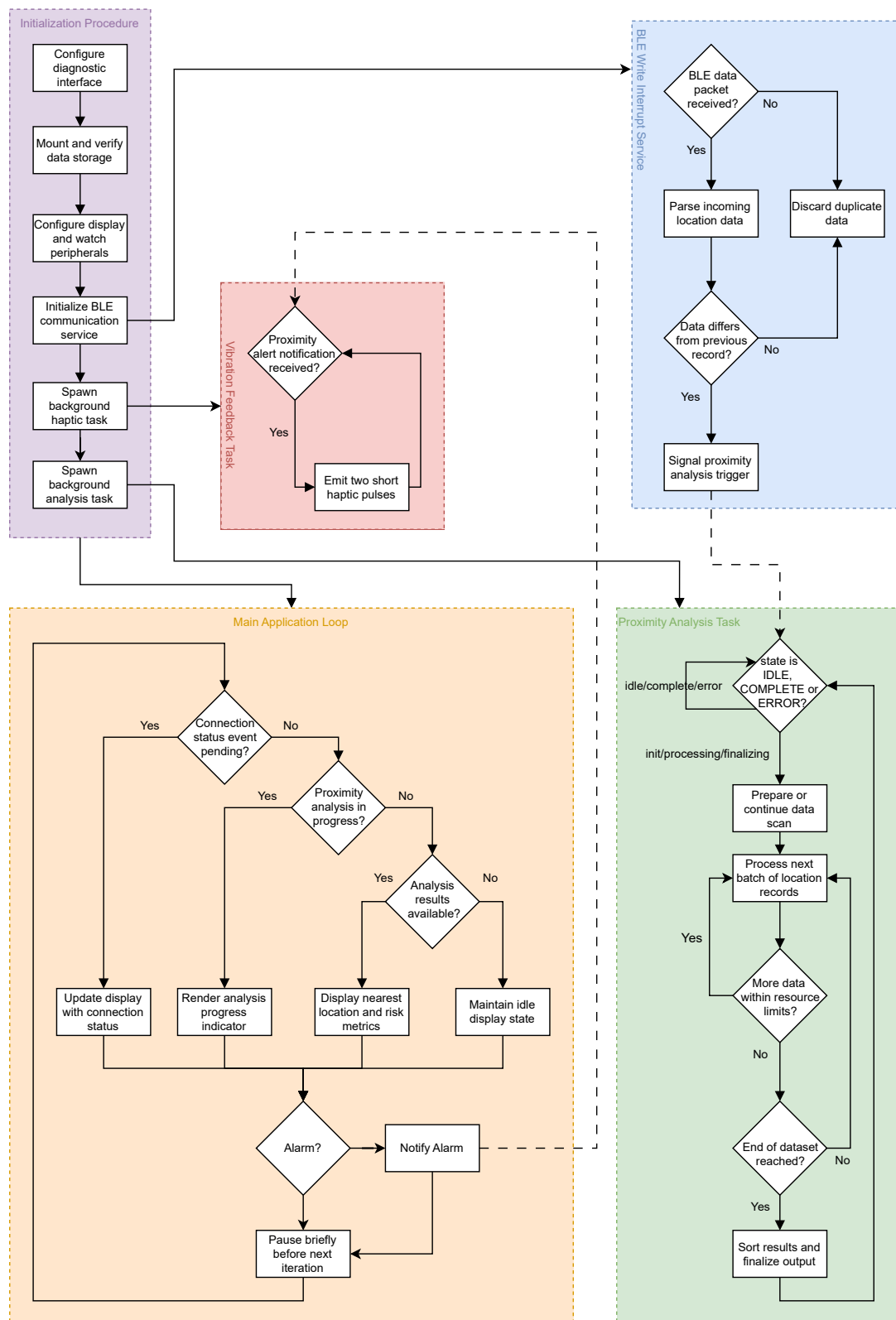


Figure 3.11: Core Execution Loop Flowchart

data storage, initiating the BLE communication service, and spawning two dedicated FreeRTOS tasks: one for handling proximity analysis and the other for controlling vibration feedback.

Location updates are handled in the BLE write handling stage, which monitors for incoming BLE packets. Upon reception, the data are parsed and compared to the last known position. Redundant updates are discarded, and only meaningful changes in location trigger the downstream proximity analysis task. This design avoids excessive computation and prioritizes efficiency.

The proximity analysis task operates in its own real-time thread, progressing through a state machine. It cycles through initialization, processing, and finalization stages, depending on its current state. During the processing phase, the algorithm reads the geospatial dataset in batches, checking whether the current processing load remains within memory and execution constraints. If resource limits are reached, the task yields and waits for resource availability. Once all data elements are processed, the task sorts the results and produces a list of nearby intersections with their associated risk values.

The main application loop operates concurrently, polling for events and updating the user interface accordingly. If a new BLE connection is detected, the loop updates the screen to reflect connection status. When analysis is ongoing, a progress indicator is rendered. Upon availability of new proximity data, the application displays the nearest intersection, associated risk metrics, and current location information. In the absence of new data, the screen remains in an idle state.

The main loop also determines whether a safety alert condition has been met, specifically, whether the cyclist's speed exceeds a risk-dependent limit when approaching an intersection. If triggered, a signal is issued to activate the vibration feedback task. This task is isolated from other operations and waits for notification to emit a predefined haptic response. This separation ensures that feedback generation does not block BLE communication or interrupt the main user interface.

This architecture maximizes concurrency while preserving deterministic behavior. Each subsystem is encapsulated within its own task context and communication between tasks is event-driven, allowing for scalable and predictable operation within the real-time constraints imposed by the embedded environment.

3.5 Mobile Application

The mobile application (Stage C) serves as an auxiliary component within the proposed system architecture, facilitating sensor data acquisition and wireless transmission in support of the smartwatch's core functionalities. Developed using a hybrid web-to-native approach, the application leverages mobile device sensors to collect geolocation data, which is then relayed to the smartwatch via Bluetooth Low Energy (BLE).

This approach addresses hardware limitations of the smartwatch, which lacks integrated GPS capabilities, by enabling the mobile phone to temporarily fulfill the role of a sensor gateway. The design emphasizes modularity, cross-platform compatibility, and real-time responsiveness, while ensuring that the smartwatch remains the primary processing unit. This section details the

development tools employed, the data acquisition pipeline, and the internal algorithmic structure of the mobile application.

3.5.1 Development Tools

The mobile application in this system is developed using a combination of web-based and native mobile development tools, specifically Capacitor and Android Studio. These tools are selected for their complementary capabilities in enabling cross-platform deployment, streamlined device integration, and efficient debugging workflows.

Capacitor, maintained by the Ionic team, serves as a cross-platform runtime environment that bridges modern web applications with native mobile functionalities. It enables developers to write application logic using standard web technologies, such as HTML, CSS and TypeScript, while maintaining access to native device features through a plugin system. Capacitor abstracts away platform-specific implementation details and provides a consistent JavaScript API for accessing hardware capabilities, including geolocation services, file system operations, and Bluetooth communication. This approach ensures that the application remains maintainable, extensible, and compatible with both Android and iOS platforms.

Android Studio functions as the primary Integrated Development Environment (IDE) for managing native platform builds, emulating device behavior, and packaging the final application for deployment. It provides advanced tooling for debugging, profiling, and inspecting application behavior on physical or virtual Android devices.

Within the context of Capacitor, Android Studio is used to configure the Android-specific project settings, manage permissions (e.g. location access and Bluetooth privileges), and compile the web-based application into a native APK. The synergy between Capacitor and Android Studio enables a streamlined workflow where web development practices are seamlessly integrated into native mobile deployment pipelines.

Together, these tools support a hybrid application architecture that combines the flexibility of web development with the performance and hardware access of native applications. Their integration is critical to achieving a responsive and feature-rich mobile interface capable of communicating efficiently with the smartwatch component.

3.5.2 Implementation

The operational behavior of the mobile application plays a supportive role in the system architecture by acquiring sensor data and relaying it to the smartwatch via Bluetooth Low Energy (BLE). The smartwatch, in turn, assumes responsibility for all analytical computations and feedback mechanisms. This subsection describes the sensor data acquisition process and the underlying algorithmic structure of the mobile application.

3.5.2.1 Sensor Data Acquisition

The mobile application performs real-time geolocation data acquisition and transmits this information to the smartwatch via Bluetooth Low Energy (BLE). This design choice arises from the hardware constraints of the selected smartwatch model, which does not include an onboard GPS module. Consequently, the mobile device is employed solely as a sensor proxy, enabling the smartwatch to function as the central computational unit.

Geolocation tracking is implemented using the Capacitor Geolocation API, which provides access to high-accuracy positioning information available on the host device. Upon initialization, the application requests location permissions and establishes a continuous geolocation watch. Each time a new position is retrieved, relevant parameters are extracted and structured into a Comma-Separated Values (CSV) format. This standardized data representation ensures compact transmission and efficient parsing on the receiving device.

To transmit this data to the smartwatch, the application utilizes the Capacitor Community Bluetooth Low Energy (BLE) plugin and the data string is encoded into a byte buffer and sent to the smartwatch. A dedicated write operation ensures reliable delivery without interrupting the app's user interface or geolocation tracking functionality. Only when a BLE connection is established does the application attempt to send updates, preserving resources and avoiding redundant operations.

To assist in the development and system testing, the mobile application integrates a live map visualization. This interactive interface allows real-time monitoring of the user's current location and visual validation of surrounding intersection risk levels. The map renders preprocessed geospatial data in the form of circular markers colored according to risk values. These markers are loaded from the same CSV dataset used by the smartwatch and plotted alongside the user's current position. A toggle control allows developers to enable or disable the visibility of this layer, facilitating targeted testing scenarios and debugging of the location-to-risk pipeline.

This process reinforces a clear separation of roles: the mobile application provides sensor input only as a fallback mechanism, while the smartwatch remains solely responsible for all computational tasks, including risk analysis and feedback generation. The overall system is designed with the intention that future hardware iterations, equipped with integrated sensors, could eliminate the need for a separate mobile component altogether.

3.5.2.2 Mobile Application Algorithm Structure

The mobile application follows a modular, event-driven architecture designed to manage initialization routines, sensor data acquisition, user interface interactions, and Bluetooth Low Energy (BLE) communication. The system, represented in Figure 3.12, is divided into distinct functional flows, each corresponding to key operational states.

During the initialization phase, the system verifies the presence of the required HTML structure. If the primary application container is not found, an error is raised. On Android platforms, the

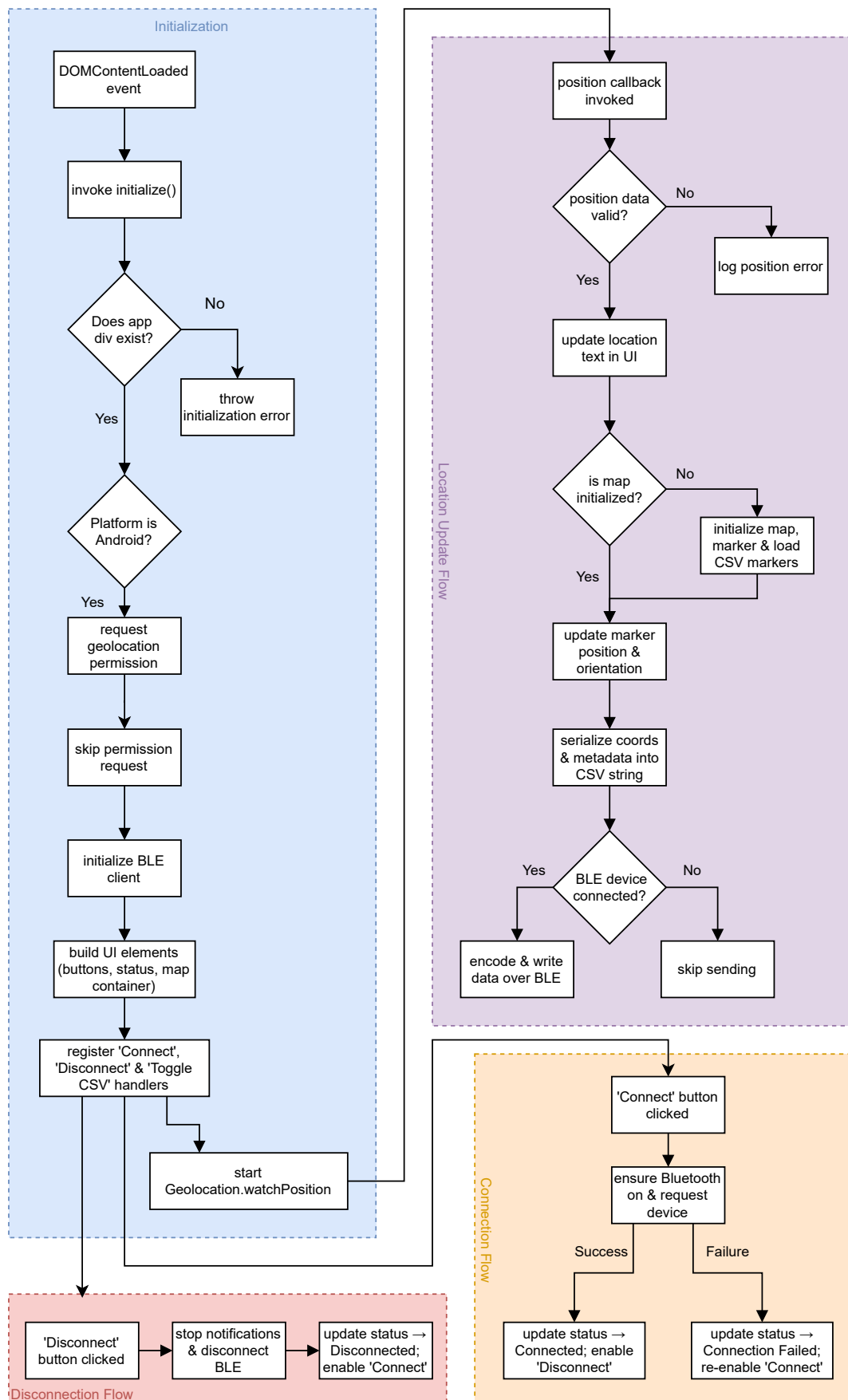


Figure 3.12: Mobile Application Flowchart

application explicitly requests geolocation permissions to ensure access to device-level positioning services. Following this, the BLE client is initialized, and the user interface is populated with essential elements, including control buttons and status indicators. Event handlers for user actions, such as connecting, disconnecting, and toggling CSV data layers are registered, and continuous geolocation tracking is initiated.

Location data updates are handled asynchronously through a position callback. Upon receiving a valid geolocation update, the application refreshes the location information displayed in the interface. If the map interface has not yet been initialized, the application creates a map, places a location marker, and loads nearby risk markers from a CSV dataset to aid in system debugging and validation. If the map is already active, the marker's position and orientation are updated in real time.

Each location update is serialized into a CSV-formatted string containing coordinates and additional metadata, such as speed. If a BLE device is currently connected, the CSV string is encoded and transmitted to the smartwatch over the specified BLE service and characteristic. If no BLE device is connected, the transmission step is skipped, and the application continues to await further updates.

Connection management is handled through user interaction with the "Connect" button. Upon activation, the application attempts to enable the Bluetooth radio and scan for available devices. If a device is successfully selected and connected, the user interface is updated accordingly, and the "Disconnect" button is enabled. In the event of a connection failure, an error message is displayed, and the "Connect" button is re-enabled for retry attempts.

The disconnection process is similarly user-driven. When the "Disconnect" button is clicked, the application halts BLE notifications and cleanly disconnects from the paired device. The status is updated to reflect the disconnection, and the interface returns to its default state.

This structured algorithm enables robust and responsive behavior within the mobile application, ensuring that data acquisition and transmission operate continuously and reliably, while isolating UI and BLE logic into manageable, event-driven components.

3.6 System Algorithm Structure

The overall system follows a modular, event-driven architecture that integrates a web-based mobile application, a Bluetooth-enabled smartwatch, and a preprocessed geospatial dataset to perform risk-aware situational analysis for cyclists. Figure 3.13 presents a consolidated flowchart outlining the interactions between the mobile front-end, the embedded smartwatch firmware, and the supporting map generator utility.

The WebApp module, implemented using web technologies and deployed as a hybrid mobile application, handles user interaction and sensor data acquisition. Upon startup, the application loads the User Interface (UI) from the index.html file and initializes the logic defined in main.ts. User interaction begins with pressing the "Connect" button, which triggers a BLE pairing procedure between the phone and the smartwatch. In parallel, the application continuously monitors

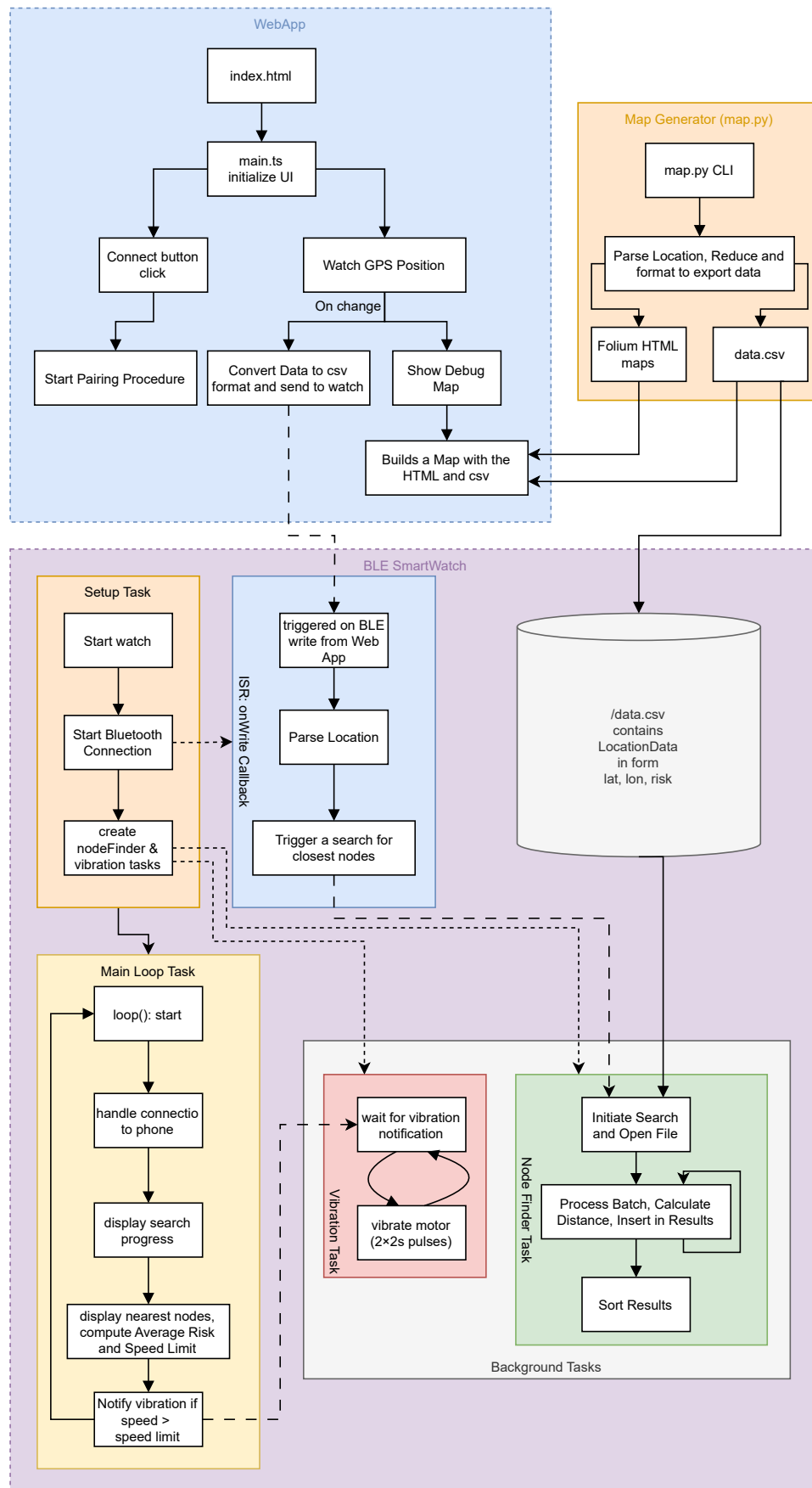


Figure 3.13: System Flowchart

the GPS position using native device sensors. Each new position update is converted into a CSV-formatted line and transmitted to the smartwatch over the BLE protocol. An optional debugging feature is also included, which renders a live map of the user's location along with annotated intersection risk markers.

On the smartwatch, system execution is initiated by a dedicated setup task. This task configures the hardware, establishes the Bluetooth connection, and spawns two background tasks: one for managing haptic feedback and another for proximity-based spatial analysis. Upon receiving a BLE write from the mobile device, an Interrupt Service Routine (ISR) is triggered, parsing the received GPS data and initiating a background search for the closest nodes in the intersection risk dataset.

The geospatial dataset, stored locally on the smartwatch as `/data.csv`, is generated using a Python script (`map.py`) that extracts and preprocesses OpenStreetMap (OSM) intersection data for a target urban area. The script exports this data in CSV format, pairing each geographic coordinate with a normalized risk score. The same dataset is reused in both the mobile map interface and the smartwatch, ensuring consistency across modules.

Once GPS coordinates are received and parsed, the node search task is activated. This task opens the dataset, processes it in batches, calculates distances between the user's location and nearby nodes, and sorts the results based on proximity. The main application loop then displays the nearest nodes, computes the average associated risk, and retrieves the corresponding recommended speed limit. If the current speed exceeds the calculated limit, the smartwatch triggers the vibration task, which activates the onboard haptic motor using a sequence of two short pulses to alert the cyclist.

Throughout the process, the mobile application operates only as a sensing and transmission aid, while the smartwatch performs all computations related to data parsing, proximity analysis, and feedback generation. This division of responsibilities supports a scalable system design that can be migrated to fully embedded platforms in future hardware revisions. The modular architecture and task separation also enhance system responsiveness and reliability, allowing concurrent operations without overloading any individual component.

This structured division ensures that the overall system adheres to the core design principles established in this study: modularity, responsiveness, and hardware efficiency. By distributing responsibilities across dedicated components the system maintains clarity of roles and minimizes interdependencies. This architecture not only facilitates debugging and iterative development but also demonstrates the feasibility of executing real-time spatial risk assessments on constrained embedded devices.

Chapter 4

Results and Discussion

This chapter integrates empirical findings with a critical appraisal of the design choices underpinning the cyclist safety prototype. Section 4.1 presents a qualitative analysis along with quantitative benchmarks that verify functional correctness, user-interface clarity, and real-time computational suitability. Section 4.2 then interprets these outcomes, relating performance trends to architectural constraints and contrasting the implemented solution with alternative approaches explored during development.

By combining objective measurements with reflective analysis, the chapter establishes how the system meets its stated requirements, identifies residual limitations, and outlines technical directions that could enhance robustness, scalability, and standalone operability.

4.1 Results

This section consolidates the empirical evidence that verifies both the functional and performance objectives of the proposed cyclist safety system. Qualitative output, captured through representative images of the smartwatch and mobile interfaces, demonstrates that spatial metrics, risk values, and connection status are conveyed through clear visual hierarchies, color-coded cues, and haptic signaling suited to in-motion contexts. Complementing this user-facing perspective, a quantitative study probes the timing and memory characteristics of the proximity-analysis firmware under systematically varied parameters, enabling data-driven optimization of computational throughput, task utilization, and heap preservation. Together, these graphical and numerical findings confirm that the prototype satisfies its design requirements while also delineating avenues for subsequent refinement.

4.1.1 Graphical Results

The graphical results presented in this subsection summarise the operational output of the developed prototype and illustrate how risk information is communicated to end-users. Representative pictures of both the smartwatch and mobile interfaces demonstrate the system’s ability to convey

connection status, spatial metrics, and context-aware risk cues through concise visual arrangements and color-coded feedback.

Together, these resulting interfaces confirm that the user-facing elements align with the functional requirements outlined in chapter 3 and provide a coherent, quickly interpretable solution, suitable for in-motion cyclist scenarios.

4.1.1.1 Smartwatch UI

The resulting smartwatch interface presents the user with continuous feedback by displaying risk-related information in a clear and structured format to support on-the-go decision-making. The interface is designed to ensure maximum visibility under varying lighting conditions and prioritizes a minimal visual hierarchy for ease of interpretation during motion.

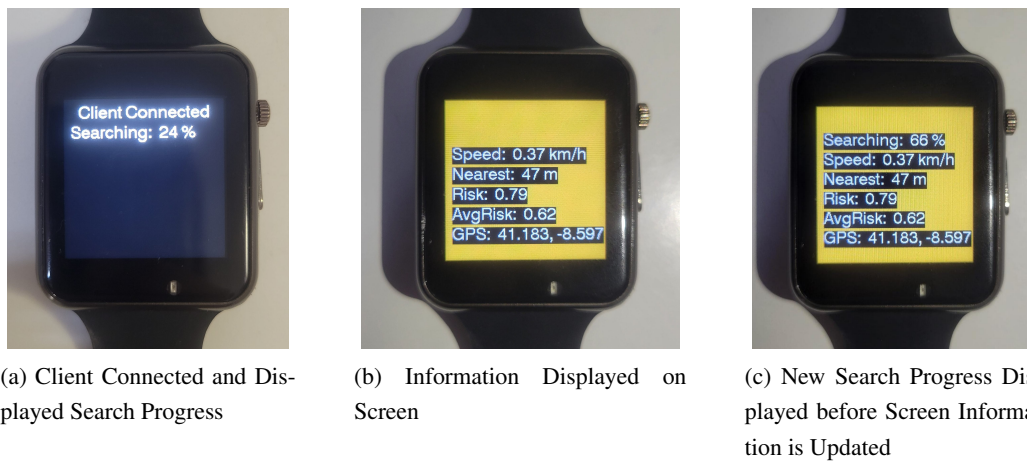


Figure 4.1: Smartwatch UI

As illustrated in Figure 4.1, the UI transitions through multiple states based on system execution. Upon establishing a connection with the mobile application, the interface displays a search progress indicator, confirming that the location data has been received and that the system is actively querying the onboard dataset. This screen is updated dynamically as the analysis proceeds.

Once a complete analysis cycle concludes and the nearest intersection candidates are identified, the interface displays the current speed and current GPS coordinates, along with the computed metrics, including distance to the nearest intersection, nearest intersection risk value and average environmental risk.

Additionally, the display background color is dynamically updated to reflect the average environmental risk, using a continuous gradient that transitions from green (low risk), to yellow (moderate risk), and red (high risk). This color-coded feedback mechanism provides an at-a-glance indication of environmental safety levels.

The resulting structured output supports user awareness by surfacing both spatial proximity and relative safety in a single glanceable interface. A vibration feedback mechanism complements the visual UI by issuing alerts when predefined safety thresholds are exceeded.

4.1.1.2 Mobile App UI

The mobile application interface is designed to facilitate Bluetooth Low Energy (BLE) pairing, present location telemetry, and assist in the development stage through the validation of spatial risk data. It adopts a minimalistic layout to maximize usability while ensuring the availability of key system information in real time.

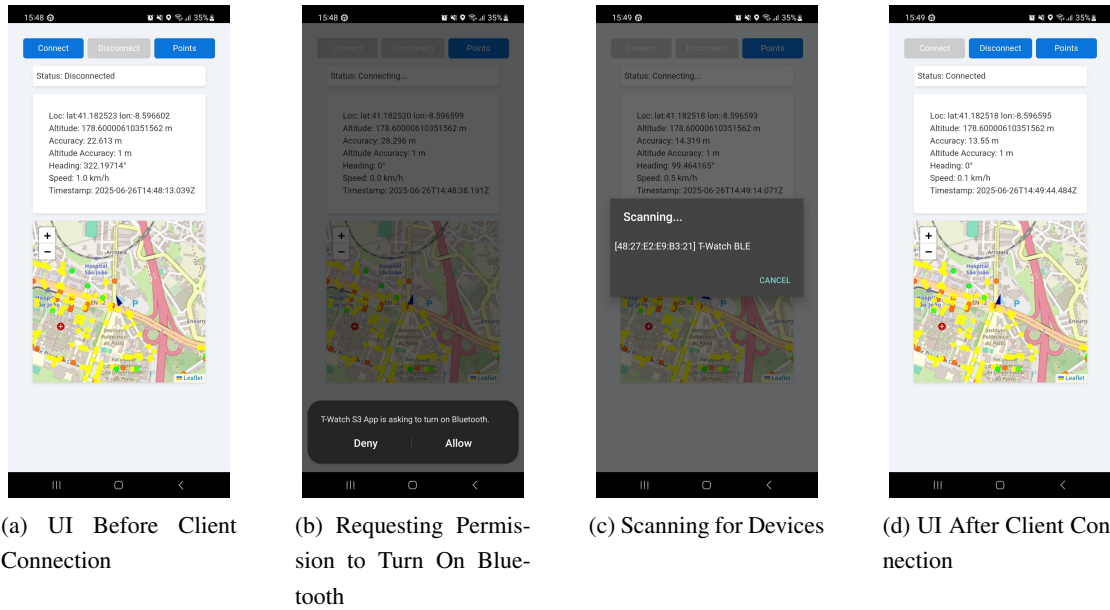


Figure 4.2: Mobile App UI

Figure 4.2 illustrates the mobile app's primary interaction states. Prior to establishing a connection with the smartwatch, the interface remains in a passive state, displaying only raw sensor information and a map with overlaid risk indicators.

Upon user initiation of the connection sequence, the application requests permission to enable Bluetooth communication and begins scanning for nearby BLE devices. If a compatible device is discovered and successfully paired, the interface updates to reflect the connected state.

Throughout operation, the top of the interface provides control buttons for initiating and terminating BLE sessions, as well as toggling the visibility of map overlays. In addition, a diagnostic pane presents live geolocation metadata, including latitude, longitude, accuracy estimates, heading, speed, and timestamp.

The embedded map view renders the user's position and risk-colored intersection markers sourced from the preprocessed dataset. This visual layer serves as a tool for validating the correctness of transmitted coordinates and the contextual alignment of environmental risk metrics.

This interface configuration ensures that the application remains responsive and transparent during system operation. By clearly exposing sensor data, connection status, and risk context, the mobile UI supports both end-user awareness and developer-level validation. Although the app

functions as an auxiliary component due to hardware constraints, its interface remains integral to system testing and deployment workflows.

4.1.2 Performance Metrics

This subsection establishes a quantitative foundation for assessing the real-time suitability of the proximity analysis firmware. It delineates the experimental framework used to capture processor utilization, execution latency, and heap consumption under systematically varied algorithm parameters. By isolating the influence of each configurable constant, while maintaining all secondary conditions, this analysis reveals the sensitivity of task timing and memory behavior to design choices that govern batch processing, task yielding, execution bounds, buffer sizing, debounce intervals, and parsing limits. The resulting empirical data is analyzed in order to update the recommended configuration. Additionally, a critical analysis of methodological limitations provides an evidence-based rationale for future optimization and validation strategies.

4.1.2.1 Developed Evaluation Strategy

With the proposed system developed and deployed, it is imperative to evaluate system performance, since this constitutes an essential phase of any embedded-system development. In order to validate whether the implementation satisfies real-time requirements and to be able to accurately tune system parameters, quantitative evidence on timing, processor loading, and memory utilization must be acquired.

Systematic measurement establishes an empirical baseline, providing developers with a better understanding of system behavior and enabling future optimizations. Hence, performance evaluation constitutes an indispensable prerequisite for iterative improvement and for demonstrating that design objectives are met under representative operating conditions.

A robust assessment of embedded real-time software typically relies on the runtime statistics provided by the operating system scheduler. However, the PlatformIO Arduino-ESP32 framework adopted for this project omits some FreeRTOS task utilities, which are available in Espressif's native IDF and are responsible for the retrieval of standard runtime statistics. To overcome this framework limitation, manual integration of the necessary utilities was attempted, but due to the scarcity of literature on the subject, this approach was not successful.

The evaluation strategy implemented for this system therefore relies on a custom alternative evaluation approach, designed to enable performance analysis of the developed software, with a particular focus on the proximity analysis task executed on the smartwatch. This task was selected for performance evaluation since it represents the most computationally intensive component of the system.

While other elements are responsible for user interface rendering, geolocation tracking, or system orchestration, the proximity analysis operation processes geographic datasets to identify

nearby intersections and assess risk values in real time. The intensity and frequency of this computation render it an ideal candidate for performance optimization, as its behavior can significantly affect system responsiveness and energy efficiency.

To analyze this component, a series of experiments were conducted by systematically varying the values of key configuration parameters that influence the proximity algorithm's behavior. For each parameter, a range of values was tested individually while all other settings remained fixed, enabling controlled analysis of their isolated effects. The fixed values set for each parameter are presented in Table 4.1.

- **MAX_BATCH_SIZE** — Defines the maximum number of data entries processed in a single task iteration. This parameter governs how much of the dataset is analyzed in each computational cycle, directly affecting throughput and processing intensity. A higher batch size can increase efficiency by reducing overhead associated with frequent task switching, but may also lead to elevated memory usage and potential latency if not properly bounded.
- **MIN_YIELD_MS** — Specifies the minimum time, in milliseconds, that the system must pause between processing batches. This parameter serves to throttle execution and allow other concurrent system operations to proceed, contributing to cooperative multitasking in the absence of preemptive scheduling. Appropriately configured, it helps maintain system responsiveness and reduces the likelihood of watchdog resets or UI interruptions.
- **MAX_PROCESS_MS** — Establishes an upper bound on the total duration of any individual processing task. Once this time threshold is reached, the system forcibly yields control regardless of whether the current batch has been fully processed. This mechanism ensures temporal isolation of the proximity analysis task, preventing monopolization of system resources and promoting predictable real-time behavior.
- **RESULT_BUFFER_SIZE** — Determines the number of nearest candidate nodes retained in memory after processing. This buffer stores the top intersection candidates ranked by spatial proximity. Limiting the buffer size constrains memory usage and enforces a fixed computational footprint, while ensuring that the most relevant results remain accessible for subsequent analysis or UI rendering.
- **DEBOUNCE_MS** — Sets a minimum time interval between successive proximity searches. This parameter prevents redundant executions of the algorithm when new data inputs arrive in rapid succession, reducing unnecessary computational load. It acts as a filtering mechanism to suppress transient fluctuations in input data and to pace the update frequency of risk feedback to the user.
- **LINE_LEN** — Indicates the maximum expected length of a single line in the CSV dataset. This constant is used to allocate sufficient memory when reading from storage, ensuring robust parsing of geospatial records. Accurate configuration of this parameter avoids buffer overflows and supports safe handling of varying data sizes across datasets.

Table 4.1: Fixed Values Assigned to Each Parameter

Parameter	Value
MAX_BATCH_SIZE	10
MIN_YIELD_MS	5
MAX_PROCESS_MS	50
RESULT_BUFFER_SIZE	5
DEBOUNCE_MS	2000
LINE_LEN	128

The performance-testing workflow follows an automated loop that iterates over combinations of parameter values and records the resulting execution metrics. On the device side, within the firmware, high-resolution timers embedded in the ESP32 microcontroller delimit the duration of each task instance. Every 5 seconds, a dedicated monitoring routine aggregates these measurements and prints a formatted output line through the serial interface.

On the host computer, a Python script manages the full testing loop. This script modifies the configuration header file that stores the parameter definitions. Once the value of a selected parameter is changed, the script compiles and flashes the new firmware to the smartwatch using the PlatformIO toolchain.

After deployment, the script connects to the serial interface and monitors the output. Each time a statistics line is detected, the script appends the captured metrics, together with a time-stamp and the current parameter annotation, to a uniquely named CSV file that identifies the active configuration.

A separate analysis script is then used to process the collected CSV data. This script generates the performance graphs, allowing for a comparative evaluation of how each parameter setting influences system behavior. This methodology enables systematic testing, consistent data collection, and clear visualization of performance trends.

The evaluation metrics of the proximity analysis algorithm focus on three primary performance indicators: CPU utilization, running time, and memory footprint. CPU utilization is defined as the ratio between the time actively spent executing the proximity analysis task and the total duration of a complete execution cycle, offering insight into how intensively this task occupies the processor. Running time corresponds to the total number of milliseconds consumed per analysis cycle, providing a direct measure of computational latency. Lastly, the memory footprint is characterized by two distinct metrics: the free heap and the minimum free heap recorded during execution.

The free heap indicates the typical amount of memory available throughout the analysis cycle, offering a general sense of the algorithm's ongoing memory consumption. In contrast, the minimum free heap captures the lowest observed level of available memory, highlighting the peak

demand imposed by the algorithm at its most memory-intensive moments. Together, these indicators provide a comprehensive view of the algorithm's memory usage patterns and its potential to affect long-term system stability.

Additionally, the graphs plotted for the CPU utilization in Figures 4.3 to 4.8 also discern between two metrics named "Proximity Analysis" and "Idle" which represent, respectively, the time spent in the proximity analysis task and the time spent in the remaining tasks.

These metrics intend to collectively provide information about the efficiency and sustainability of the algorithm when deployed in resource-constrained embedded environments, providing a basis for identifying configurations that optimize system efficiency without compromising accuracy or responsiveness.

4.1.2.2 MAX_BATCH_SIZE Experiment

The experiment in Figure 4.3 explores the influence of the MAX_BATCH_SIZE parameter (the number of CSV lines processed before the task yields) on computational load and memory demand. Since this variable governs how much work the proximity routine performs in a single invocation, it directly shapes both execution latency and scheduler fairness.

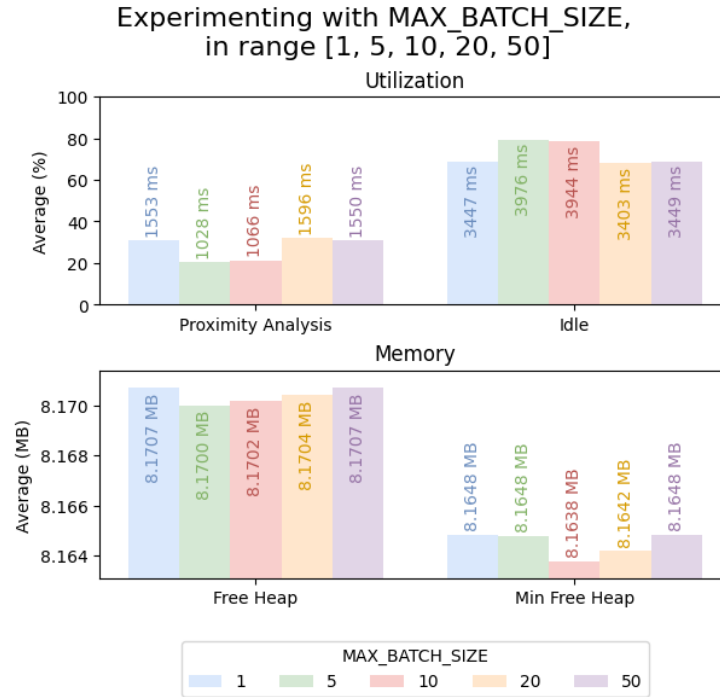


Figure 4.3: MAX_BATCH_SIZE Experiment

Results indicate that a batch size of 5 lines produces the most favorable timing profile. At this setting the proximity task completes one analysis cycle in approximately 1028 ms, the shortest duration observed across the sweep, while idle time rises modestly. This combination is indicative of a more efficient task yield and reduced contention for CPU resources. However, increasing

the batch size beyond this point does not yield further benefits, with processing times slightly increasing and idleness remaining relatively stable.

Heap statistics remain uniformly high, so overall memory availability is not compromised by batch size alone. Nevertheless, the minimum free heap metric exhibits subtle sensitivity. A batch of 10 lines temporarily reduces available memory, whereas batches of 5 and 50 lines sustain a marginally larger buffer, suggesting that intermediate sizes may cause brief spikes in dynamic allocation pressure.

In summary, the analysis identifies a `MAX_BATCH_SIZE` of 5 as the configuration that best reconciles execution speed with memory stability, thereby supporting responsive real-time behavior without imposing unnecessary strain on limited embedded resources.

4.1.2.3 MIN_YIELD_MS Experiment

The experiment depicted in Figure 4.4 investigates the impact of the `MIN_YIELD_MS` parameter, which determines the minimum delay introduced between successive batches of the proximity analysis routine. This parameter directly affects how often the task voluntarily yields CPU time, playing a key role in balancing computational throughput with cooperative multitasking on the embedded system.

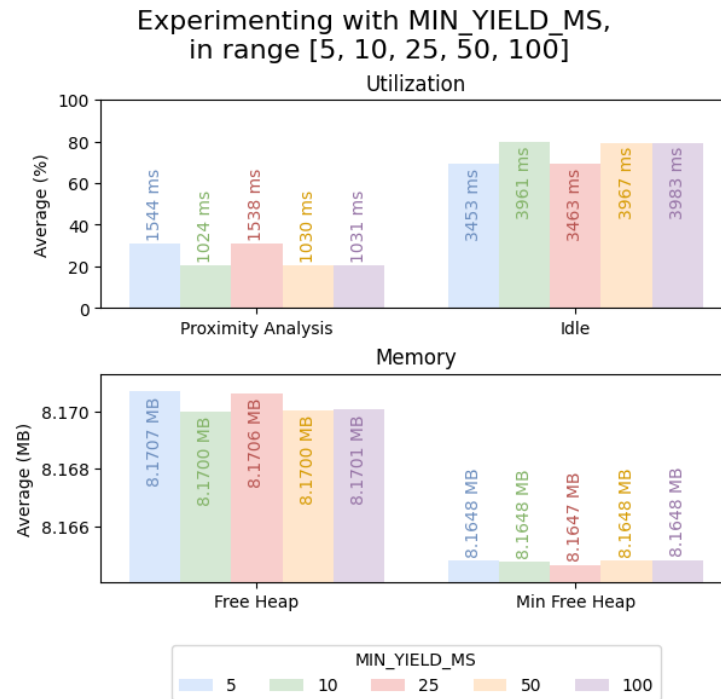


Figure 4.4: `MIN_YIELD_MS` Experiment

The results show that setting the yield interval to 10 ms produces the shortest average proximity analysis time, with a cycle completing in approximately 1024 ms. This configuration also maintains a relatively high proportion of idle time, indicating efficient CPU sharing. In contrast,

shorter intervals, such as 5 ms, appear to introduce unnecessary delays, increasing the overall task duration. Longer intervals, such as 25 ms or above, do not meaningfully improve processing speed and result in only marginal differences in idle time.

Memory usage remains largely consistent across the tested configurations, since free heap values fluctuate within a narrow band, reflecting stable memory availability. Additionally, the minimum heap never diverges by more than a few kilobytes (kB), indicating that the yield interval has a negligible influence on transient allocation patterns.

In conclusion, the evaluation identifies a MIN_YIELD_MS value of 10 ms as the most effective choice, offering an optimal balance between rapid task completion and sufficient task yielding to preserve system responsiveness. This configuration ensures efficient use of processing time without compromising memory or interrupt latency.

4.1.2.4 MAX_PROCESS_MS Experiment

The experiment shown in Figure 4.5 explores the influence of the MAX_PROCESS_MS parameter, which defines the maximum allowed processing time per cycle before the proximity analysis task is forced to yield execution. This constraint is intended to safeguard system responsiveness by limiting uninterrupted computation, especially in multitasking environments such as those using FreeRTOS.

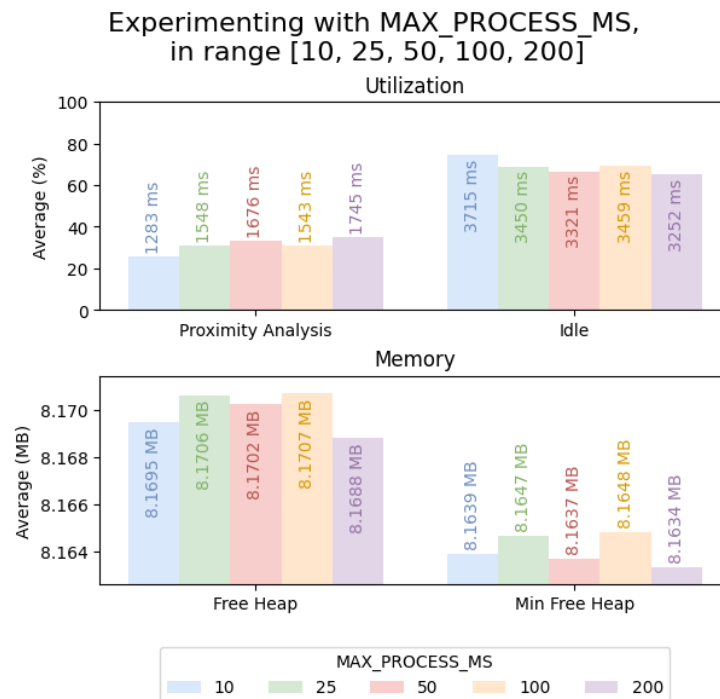


Figure 4.5: MAX_PROCESS_MS Experiment

Performance results indicate that a value of 10 ms yields the shortest average cycle duration at approximately 1283 ms, with progressively higher values resulting in increased task durations. In

particular, values beyond 50 ms exhibit diminishing returns or even performance regressions. Notably, while 100 ms maintains similar timing to lower values, setting the threshold to 200 ms leads to a longer average analysis time of 1745 ms, indicating inefficiencies introduced by prolonged uninterrupted execution. Idle time remains relatively stable across all configurations, suggesting that changes to the MAX_PROCESS_MS parameter have a more localized effect on the analysis task than on overall task scheduling.

The memory profile is similarly consistent in terms of free heap values, although lower values of the parameter, particularly 10 ms, lead to slightly reduced memory availability. The minimum free heap exhibits its lowest reading at 200 ms, reinforcing the trend that longer processing windows tend to increase transient memory consumption.

In summary, the analysis supports the use of a 10 ms or 25 ms limit for the MAX_PROCESS_MS parameter, as these settings minimize execution time while preserving memory headroom. These values achieve an effective trade-off between computation time and system responsiveness, contributing to a more stable multitasking environment.

4.1.2.5 RESULT_BUFFER_SIZE Experiment

Figure 4.6 presents the evaluation results for the RESULT_BUFFER_SIZE parameter, which controls the number of top-scoring nodes retained during each cycle of the proximity analysis. This parameter plays a central role in determining the granularity and scope of spatial risk evaluation, directly affecting the algorithm's memory and processing demands.

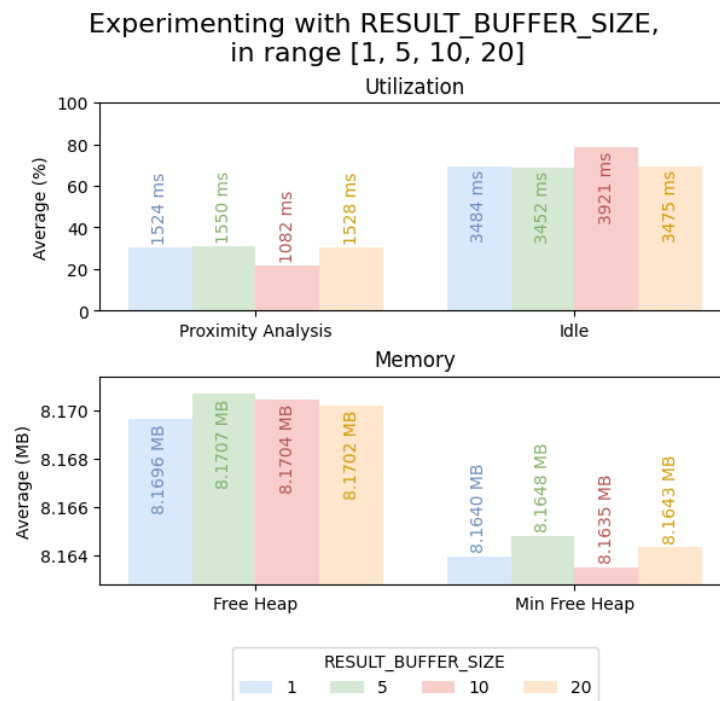


Figure 4.6: RESULT_BUFFER_SIZE Experiment

Execution time data shows that a buffer size of 10 produces the shortest average cycle duration at 1082 ms, making it the most efficient configuration in terms of computation speed. Lower or higher values result in moderate increases in execution time, suggesting that there exists an optimal balance between buffer size and the overhead associated with result sorting and retention. The idle time remains relatively stable across all configurations, indicating that variations in buffer size have limited impact on overall CPU scheduling behavior.

Memory analysis shows only minor variations in free heap, with buffer sizes of 5 and 10 maintaining consistently high values. The minimum free heap, however, is more sensitive to this parameter. A buffer size of 10 leads to the lowest memory availability, while values of 1 and 20 show comparatively better retention. These results indicate that larger buffer sizes increase the likelihood of temporary memory pressure during intensive processing, which may be undesirable in memory-constrained environments. Nonetheless, the discrepancy between minimum free heap values is not significant enough to deter the choice of any particular batch size.

In conclusion, a `RESULT_BUFFER_SIZE` of 10 achieves the best balance between execution efficiency and memory constraints. This configuration provides sufficient detail for effective spatial filtering while minimizing runtime duration and maintaining acceptable memory usage, making it a suitable choice for real-time proximity evaluation tasks.

4.1.2.6 DEBOUNCE_MS Experiment

Figure 4.7 presents the evaluation results for the `DEBOUNCE_MS` parameter, which sets the minimum time interval between consecutive proximity analysis triggers. This variable governs how frequently the system is allowed to run its main analysis cycle, acting as a form of rate-limiting that prevents redundant processing when user position does not significantly change.

The execution time analysis indicates that longer debounce intervals reduce the proximity analysis workload per time unit, resulting in lower average processing times. The configuration using 5000 ms yields the fastest cycle time at 1184 ms, followed closely by the 2000 ms setting. In contrast, the shortest debounce value of 250 ms increases task frequency, which leads to more frequent computations and consequently higher execution time. Idle time follows an inverse trend, decreasing gradually as debounce intervals shorten and the proximity analysis task utilization rises.

Memory analysis shows minimal variation in free heap across all tested values. However, the minimum free heap suggests a subtle but consistent decline as debounce intervals increase. This effect is likely due to the sporadic nature of memory usage during less frequent task execution, where brief peaks in resource allocation are more pronounced and less frequently averaged out.

In conclusion, increasing the `DEBOUNCE_MS` interval reduces processing time and improves system idleness without compromising memory efficiency. A value of 5000 ms achieves the most favorable performance, suggesting that longer debounce periods effectively balance responsiveness and system load, particularly when rapid location changes are uncommon.

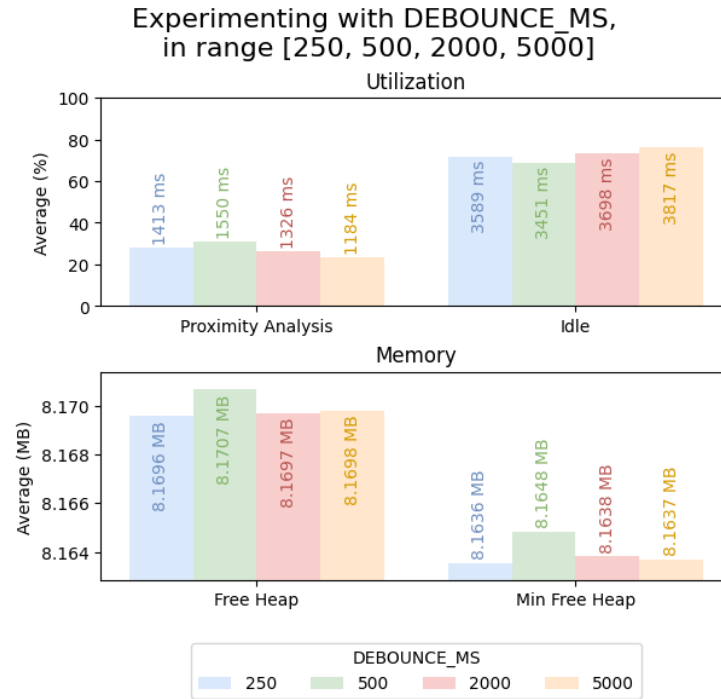


Figure 4.7: DEBOUNCE_MS Experiment

4.1.2.7 LINE_LEN Experiment

In the experiment shown in Figure 4.8, the `LINE_LEN` parameter specifies the maximum number of characters accepted for each CSV record read from the on-board dataset. Since every line is parsed in RAM before distance calculations proceed, this limit directly influences the transient memory allocated for buffering and therefore has the potential to affect both execution time and heap usage.

Across the tested range, from 64 to 512 bytes, average processing time remains tightly clustered around 1500 ms, and overall CPU utilization shows negligible variation. These results indicate that even the shortest buffer length affords sufficient space for the longest lines in the dataset; consequently, no measurable penalty arises from string reallocation or fragmentation when the limit is increased. Idle-time averages follow the same stable trend, confirming that the task scheduler experiences no scheduling pressure attributable to this parameter.

Free-heap measurements are essentially identical at every configuration, underscoring that buffer allocation is modest relative to total available RAM. Minimum-heap values exhibit a slight downward drift when the buffer length is raised to 512 characters, yet the absolute difference remains well above the threshold that could jeopardize stability. The minimal sensitivity of heap metrics confirms that the system's memory footprint is dominated by other static structures rather than by the temporary storage reserved for line parsing.

Since extending `LINE_LEN` beyond 128 characters offers no measurable performance benefit while marginally reducing the minimum free heap, a ceiling of 128 characters represents an ap-

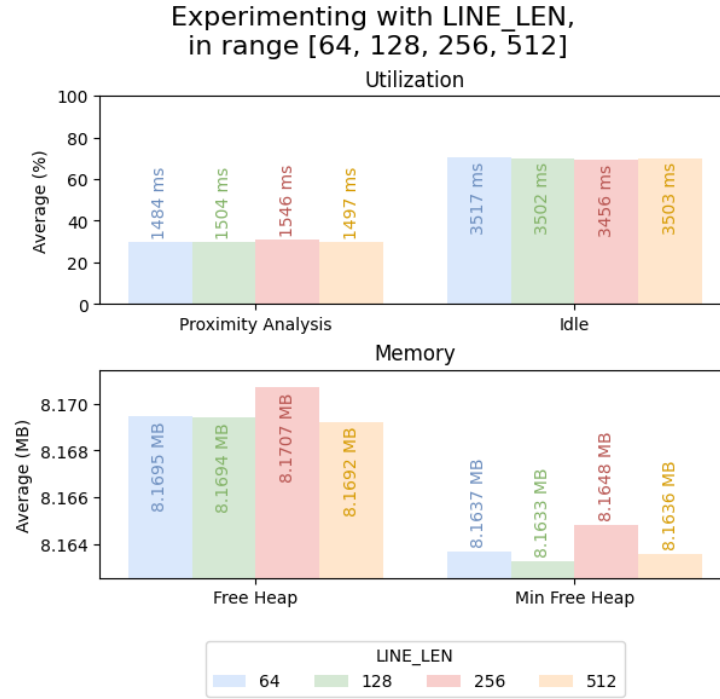


Figure 4.8: LINE_LEN Experiment

proprate compromise. This value guarantees headroom for anomalously long records yet avoids allocating superfluous buffer space, thereby preserving predictable memory margins under all operating conditions.

4.1.2.8 Recommended Configuration and Evaluation Strategy Assessment

Based on the experiments run during the evaluation process, the values reported in Table 4.3 show the configuration that will theoretically yield a more favorable trade-off between execution time, processor utilization, and heap preservation within the proximity-analysis routine.

Some modifications, such as reducing MAX_BATCH_SIZE and MAX_PROCESS_MS, promote shorter bursts of computation followed by more frequent task yields, which in turn lowers contention for CPU cycles and minimizes transient pressure on memory. Increasing RESULT_BUFFER_SIZE improves temporal consistency in the ranking of candidate nodes without imposing a discernible memory penalty, whereas enlarging DEBOUNCE_MS lengthens the minimum interval between search triggers and therefore mitigates redundant evaluations during rapid waypoint updates. The parameter LINE_LEN however, whose updated value coincides with their original setting, already offers a satisfactory balance between parsing reliability and storage overhead, therefore no measurable benefit emerges from further expansion.

Table 4.3: Recommended Values Assigned to Each Parameter After Evaluation

Parameter	Previous Value	Updated Value
MAX_BATCH_SIZE	10	5
MIN_YIELD_MS	5	10
MAX_PROCESS_MS	50	25
RESULT_BUFFER_SIZE	5	10
DEBOUNCE_MS	2000	5000
LINE_LEN	128	128

Despite the performance metrics provided by the executed experiments, the adopted evaluation protocol remains narrowly scoped. Metrics are sampled exclusively inside the proximity analysis task, and no direct observation is made of secondary activities such as the Bluetooth communication, display refresh, or power-management interrupts.

As mentioned in subsection 4.1.2.1, this specificity arises from a fundamental limitation of the PlatformIO Arduino-ESP32 framework, which omits the FreeRTOS run-time statistics normally available on native ESP-IDF builds. Under these constraints, microsecond-level timing and heap instrumentation embedded in the task itself provide the most practical means of performance assessment. Nevertheless, an inevitable consequence of this design choice is that the broader dynamics of the firmware are only inferred rather than explicitly measured.

An additional relevant consideration concerns parameter interdependence. Each sweep varies a single setting while all others remain at their nominal values, however, the interaction of multiple parameters may introduce emergent effects not captured by a one-factor analysis. A more comprehensive validation will therefore require additional experimentation that explores combined parameter sets, extends monitoring to the remaining firmware tasks, and, when feasible, incorporates system-level profiling facilities. Such follow-up studies would clarify any residual coupling effects and confirm the robustness of the configuration proposed in Table 4.3.

4.2 Discussion

This section situates the results obtained in section 4.1 within a broader engineering context. It synthesizes performance evidence and highlights architectural trade-offs revealed through alternative design experiments. By articulating both the strengths and the residual limitations encountered throughout project development, this section delineates a pathway for transitioning the prototype toward a more robust, scalable, and standalone cyclist safety platform.

4.2.1 Developed System

The empirical findings reported in section 4.1 demonstrate that the prototype fulfills its principal functional objectives while operating within the resource constraints of a low-cost wearable platform.

Visual inspections of the smartwatch and mobile interfaces confirm that connection status, spatial context, and risk assessments are communicated through concise textual cues, color-graded backgrounds, and haptic alerts, thereby satisfying the requirement for glanceable feedback suited to dynamic riding conditions. The color gradient displayed according to the average environmental risk, combined with a stepped vibration pattern when speed exceeds the locally derived limit, establishes a multimodal channel that promotes rapid user comprehension without imposing a high cognitive load.

Quantitative analysis of the proximity analysis firmware suggests that the algorithm achieves its real-time performance goal. The suggested parameter set should lower average cycle times, reduce CPU utilization peaks, and preserve ample heap headroom, thereby mitigating the risk of watchdog resets or memory fragmentation. These results corroborate the viability of executing geospatial proximity checks and risk weighting directly on the ESP32-S3 microcontroller, without the need to delegate computation to external servers or companion devices.

Despite these strengths, the assessment highlights several structural limitations. For instance, the absence of an integrated Global Navigation Satellite System (GNSS) in the smartwatch demands a continuous reliance on a paired external device, introducing an additional point of failure and constraining standalone usage. Additionally, the employed evaluation strategy samples performance metrics exclusively from the proximity analysis task, leaving potential interactions with BLE communication, display refresh, and power management routines unquantified.

In practical terms, the prototype constitutes a modular foundation for context-aware cycling safety but warrants further refinement. Furthermore, long-duration field trials under variable urban topologies and network densities are required to validate the risk mapping methodology beyond the controlled scenarios used for benchmark generation.

4.2.2 Alternative Approaches Explored

Exploratory experimentation preceded the final system design, encompassing multiple candidate solutions for spatial data preparation, device-to-device communication, and firmware tooling. Each alternative pursued a different objective while contending with the constraints imposed by the wearable hardware and the project timeline. By documenting these trials alongside their technical limitations and prospective advantages, the discussion contextualizes some design decisions implemented in the prototype and delineates avenues for future refinement.

4.2.2.1 Map Processing

Several alternative data processing strategies were investigated before adopting the city map generator routine described in Chapter 3. Two of these approaches are presented here as alternatives

to map generation that can be explored and refined further.

The first experiment begins with an unfiltered OpenStreetMap graph. Every dead end is then filtered out by removing all nodes with a degree ≤ 2 , thereby isolating genuine intersections. However, the resulting intersection map is still very cluttered, with some nodes in very close proximity. This dataset is not optimal for the proposed application, since it can contain redundant information and the overall file size can become too big for the hardware memory constraints.

Thus, to further compress the resulting network, the urban environment is partitioned into a fixed grid. Every intersection inside each cell is then mapped to a single cluster centroid computed as the degree-weighted mean of its member coordinates. Figure 4.9 provides a visual representation of the resulting map.



Figure 4.9: Map of Intersections in Porto, Portugal with Grid Clustering Approach

This coarse aggregation lowers the overall node count, but it introduces two critical drawbacks. Firstly, relocating nodes to cell centroids displaces many intersections, degrading the location accuracy of nearest-intersection queries. Secondly, the uniform grid can not accurately adapt to heterogeneous street densities, leading to over-aggregation in compact city quarters and under-aggregation in more disperse environments. These disadvantages contributed to the decision of abandoning this approach.

A second explored strategy replaces discrete intersection points with a continuous risk surface. Intersection coordinates serve as nuclei for a planar Voronoi tessellation, and each cell inherits the street-count-based risk of its generating node. During runtime the smartwatch could, in principle, determine risk by locating the cell enclosing the current GPS fix, thereby avoiding repeated searches for the nearest intersection. The resulting map is plotted in Figure 4.10.

Although conceptually attractive, this method may be harder to deploy in resource-constrained hardware applications. The Voronoi diagram of a dense urban network yields several thousand polygons with arbitrary vertex counts. Serializing that geometry, performing point-in-polygon

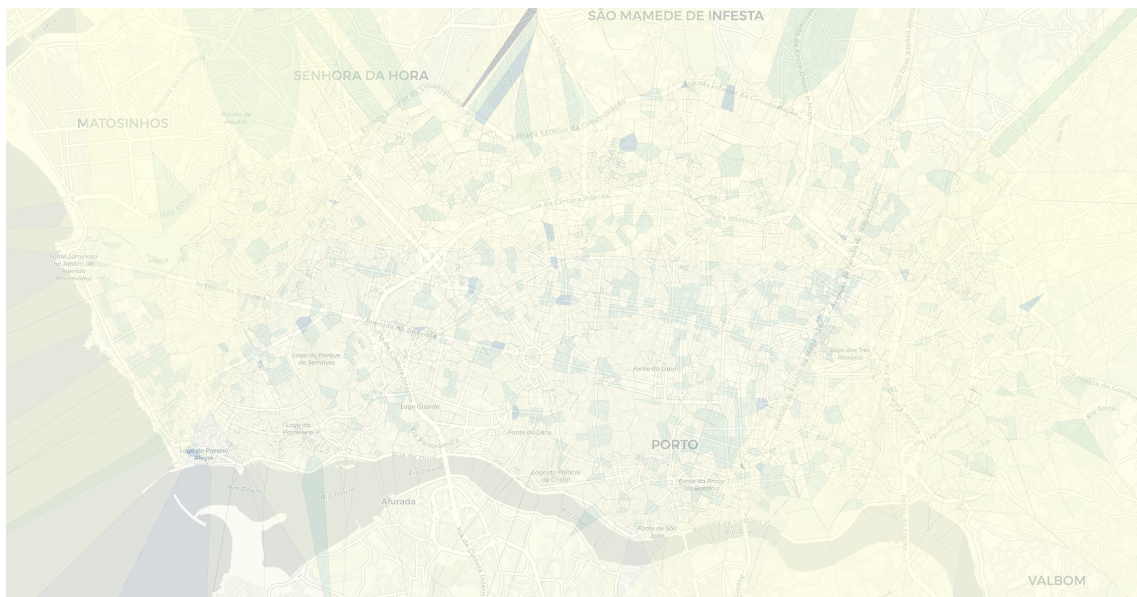


Figure 4.10: Map of Porto, Portugal with Voronoi Cells Approach

tests, and updating color gradients may exceed both storage capacity and real-time processing limits of the ESP32. Moreover, the abrupt boundaries between cells produce risk discontinuities that could conflict with the visual feedback strategy adopted for the user interface.

Both approaches underscore the difficulty of balancing both geometric fidelity and computability on wearable hardware. Grid clustering sacrifices localization precision, while the Voronoi formulation may exceed memory and computational limitations.

The intersection consolidation function incorporated into the preprocessing pipeline mitigates these limitations: it collapses topologically adjacent nodes only where redundancy is likely, preserves original coordinates elsewhere, and produces a compact tabular dataset readily parsed on-device. Nonetheless, these alternative map processing approaches may still prove useful in future developments, provided they are carefully adapted to the constraints and requirements of the target environment or specific use case.

4.2.2.2 Communication

An alternative communication architecture was considered, in which the smartwatch and the mobile application would exchange data through an ad-hoc HTTPS channel rather than Bluetooth Low Energy. An approach using HTTP was initially considered, but the secure context requirements imposed by recent mobile operating systems define that geolocation data can be accessed only from pages served over HTTPS. Consequently, any web-based telemetry pipeline must employ encrypted transport even when devices operate within a private network and without internet connectivity.

Two deployment variants were examined. In the first, the smartwatch hosts a Software Access

Point (SoftAP), serves encrypted pages, and accepts location updates from the phone. In the second, the phone assumes the server role, while the watch connects as a Wi-Fi client and periodically requests or receives position data.

Although both variants satisfy the design objective of remaining fully functional in the absence of external infrastructure, the smartwatch-centered configuration proved to be very challenging. Running the Wi-Fi stack, maintaining a SoftAP, and terminating an HTTPS web server concurrently imposes a substantial processing and memory burden on the embedded microcontroller. The necessity to provision, store, and periodically refresh Transport Layer Security (TLS) certificates further increases complexity and amplifies connection latency during the initial handshake.

When the phone operates as the server, certificate handling becomes more tractable, yet the watch must still maintain a persistent Wi-Fi association, which elevates power consumption and reduces runtime compared with Bluetooth-based alternatives.

The adopted arrangement therefore reverts to a split model in which the watch exposes a minimalist BLE characteristic that accepts raw geolocation bytes, while the mobile application implements the complete web stack (including secure JavaScript APIs, certificate management, and network permissions) within its more capable operating environment. This partitioning confines energy-intensive and security-sensitive operations to the smartphone, lowers the firmware footprint on the watch, and preserves a low-latency, connectionless transport suited to short, periodic data bursts.

4.2.2.3 Framework

The feasibility of migrating the firmware to Espressif's ESP-IDF with the Arduino core compiled as a secondary component was evaluated as an avenue for increased low-level control. ESP-IDF exposes a comprehensive set of microcontroller-specific drivers and FreeRTOS utilities, thereby permitting finer granularity in task scheduling, access to detailed run-time statistics, and native support for peripheral features that remain inaccessible or cumbersome in the stand-alone Arduino-ESP32 environment.

Practical integration, however, encountered extensive incompatibilities. The LILYGO T-Watch S3 support libraries for PlatformIO have been developed using the Arduino-ESP32 build system. Porting them to ESP-IDF would require refactoring a broad dependency chain encompassing graphics primitives, power-management abstractions, and third-party sensor drivers. At the time of experimentation, both frameworks were simultaneously undergoing major-version migrations that introduced breaking changes, amplifying maintenance overhead and impeding a stable compilation target.

Although these constraints precluded immediate adoption, the ESP-IDF tool-chain remains attractive for future iterations. Its richer diagnostic utilities are capable of addressing the system's current limitations on performance instrumentation. Additionally, its deterministic FreeRTOS configuration options can facilitate more precise tuning of task priorities and core affinities.

Once the surrounding ecosystem converges on updated releases, and provided that hardware vendors supply compatible board-support packages, the ESP-IDF configuration is likely to offer

a more reliable and extensible foundation than the Arduino-centric workflow employed in the present prototype.

Chapter 5

Conclusion and Future Work

This closing chapter consolidates the insights gained throughout the investigation, evaluating how effectively the prototype satisfies its initial objectives and identifying realistic trajectories for subsequent research and development.

First, the degree to which the system meets functional, performance, and openness requirements is examined, highlighting the principal achievements and residual constraints that emerge from the empirical evidence presented in the previous chapters.

The discussion then transitions to prospective enhancements, detailing hardware upgrades, framework migrations, data-model refinements, interface adaptations, and validation strategies capable of elevating both autonomy and analytical depth. Together, the two sections provide a holistic appraisal of the current solution while charting a coherent roadmap toward an improved cyclist safety platform.

5.1 Objective Satisfaction

The developed system successfully addresses the main objectives outlined at the beginning of the project, establishing a functional and extensible foundation for enhancing cyclist safety through wearable technologies.

The prototype leverages the LILYGO T-Watch S3, a commercially available and cost-effective smartwatch built around the ESP32-S3 platform. Its firmware is developed within the Arduino-ESP32 framework and supported by companion tools written in Python and TypeScript. All code and data processing routines are made available under permissive open-source licenses, ensuring accessibility and transparency.

This approach fulfills the project's commitment to openness and reproducibility, enabling independent deployment, adaptation, and improvement by the wider community. The design explicitly avoids reliance on proprietary cloud services or licensed data sources, further reinforcing its open-source and standalone character.

The user interface design for both the smartwatch and mobile application reflects a strong emphasis on in-motion usability. The smartwatch interface displays dynamically updated risk metrics

using a limited but well-structured visual hierarchy, complemented by color-coded backdrops that map to environmental risk levels. Haptic signaling is employed to alert users when critical thresholds are crossed, offering a secondary feedback channel that is perceptible without requiring visual focus.

The mobile application, while intended primarily as a development and testing tool, provides an additional layer of interaction, enabling live geolocation monitoring, Bluetooth connection management, and visual debugging of transmitted risk values. These multimodal interfaces collectively support the objective of delivering clear and actionable feedback under real-world cycling conditions.

From a computational perspective, the proximity analysis task constitutes the most demanding aspect of the firmware. The implemented algorithm is designed to identify the nearest intersection within a geospatial dataset and retrieve its associated risk metrics in real time.

Experimental evaluation confirms that the algorithm performs this task efficiently across a range of parameter configurations. Execution cycle durations consistently fall below critical latency thresholds, processor utilization remains within sustainable bounds, and available heap memory remains sufficiently preserved to avoid instability. These outcomes validate the feasibility of performing spatial filtering and risk evaluation entirely on-device, without the need for offloading computation to a paired smartphone or external server.

To ensure that the system can be adapted to diverse urban environments, a preprocessing pipeline is introduced that transforms raw OpenStreetMap data into a compact and hardware-compatible format. This pipeline consolidates intersections, quantises risk values, and outputs a tabular dataset optimised for parsing on resource-constrained devices.

Multiple alternative preprocessing strategies, such as grid-based clustering and Voronoi-based risk surfacing, were explored but ultimately set aside due to performance limitations or geometric fidelity concerns. The chosen consolidation method strikes a practical balance between compression and accuracy, enabling real-time spatial queries on wearable hardware while maintaining city-specific contextual relevance.

Although the system meets its stated objectives, certain limitations and trade-offs remain. The absence of an integrated GNSS module on the watch requires continuous tethering to a smartphone for geolocation data, constraining standalone operation. Moreover, the PlatformIO Arduino-ESP32 framework's lack of comprehensive runtime instrumentation tools impedes full system-level performance analysis, limiting the scope of evaluation to isolated tasks such as the proximity analysis routine. Nevertheless, these limitations are not intrinsic to the system's architecture and can be minimized with further refinements in future work.

5.2 Future Work

Subsequent iterations of the system can eliminate the dependency on an external device by selecting wearable hardware that integrates a Global Navigation Satellite System (GNSS) receiver.

LILYGO recently introduced the T-Watch S3 Plus [2], a form-factor successor to the device employed during prototyping that incorporates on-board GPS capabilities while preserving the same ESP32-S3 architecture.

Migrating the firmware to this model would require only minor adjustments to driver initialization and sensor-fusion routines. Thereafter, all geolocation, risk analysis, and feedback generation could be executed autonomously on the watch. Comparable results could be obtained with any commercially available smartwatch that exposes GNSS data over an open SDK, although additional refinements might be necessary to accommodate vendor-specific peripheral mappings and power-management schemes.

Broader functional gains can also arise from adopting Espressif's ESP-IDF build system. Direct access to FreeRTOS run-time statistics, deterministic scheduling parameters, and advanced peripheral drivers would enable finer control over task affinities, interrupt latencies, and energy budgets, thereby improving both performance observability and battery endurance. A native ESP-IDF implementation would further permit integration of modern cryptographic libraries should secure communication channels or over-the-air update mechanisms become desirable in production deployments.

On the data side, the present intersection-centric risk model offers a tractable baseline but admits multiple avenues for enrichment. Incorporating supplemental geospatial layers, such as documented crash hotspots, traffic volume estimates, dedicated cycling infrastructure, or surface condition reports, would yield a multidimensional risk index with higher validity.

A complementary crowdsourcing mechanism could allow cyclists to flag emerging hazards in real time, enabling dynamic updates to the on-device dataset and fostering a participatory safety network. Machine Learning (ML) techniques, trained on aggregated location traces and sensor telemetry, may also reveal latent predictors of hazardous encounters that are not captured by static street-network metrics alone.

Interface improvements constitute another tractable research path. Context-aware layout adaptation can reduce visual clutter by prioritising only those metrics that are immediately pertinent to the cyclist's current speed or route complexity. Gesture-based acknowledgement of alerts and configurable colour palettes would further personalise the experience while minimising cognitive load. Concurrently, expanded haptic vocabularies, for example, distinguishing between speed-limit exceedance and proximity to a high-risk junction, could enhance non-visual information bandwidth.

Finally, extended field studies are essential for validating long-term robustness under diverse environmental and behavioural conditions. Multi-week deployments across heterogeneous urban morphologies would test dataset scalability, communication reliability, and energy-consumption profiles. Insights from such trials would inform iterative refinements to the preprocessing pipeline, parameter auto-tuning heuristics, and power-saving strategies, guiding the platform toward a mature, production-ready solution for cyclist safety.

In summary, the developed system demonstrates that a low-cost, open-source smartwatch can deliver meaningful, context-aware cyclist safety features through a combination of local process-

ing, minimal interfaces, and precomputed geospatial risk models. Furthermore, the project offers a validated blueprint for future systems seeking to enhance urban mobility safety using embedded and wearable technologies.

References

- [1] T-Watch S3. URL: <https://lilygo.cc/products/t-watch-s3>.
- [2] T-Watch S3 Plus. URL: <https://lilygo.cc/products/t-watch-s3-plus>.
- [3] THE 17 GOALS | Sustainable Development. URL: <https://sdgs.un.org/goals>.
- [4] espressif/arduino-esp32, June 2025. original-date: 2016-10-06T06:04:20Z. URL: <https://github.com/espressif/arduino-esp32>.
- [5] Alexandru Dancu, Alexandru Dancu, Velko Vechev, Velko Vechev, Adviye Ayça Ünlüer, Adviye Ayça Ünlüer, Simon Nilson, Simon Nilson, Oscar Nygren, Oscar Nygren, Simon Eliasson, Simon Eliasson, Jean-Elie Barjonet, Jean-Elie Barjonet, Joe Marshall, Joe Marshall, Morten Fjeld, and Morten Fjeld. Gesture Bike: Examining Projection Surfaces and Turn Signal Systems for Urban Cycling. *International Conference on Intelligent Tutoring Systems*, pages 151–159, November 2015. MAG ID: 1917389667 S2ID: 954828887fb7abce546a5b1e4590c9b331826ac4. doi:10.1145/2817721.2817748.
- [6] Andrii Matviienko, Manuel Lehé, Florian Heller, Bastian Pfleging, and Max Mühlhäuser. QuantiBike: Quantifying Perceived Cyclists’ Safety via Head Movements in Virtual Reality and Outdoors. *Symposium on Spatial User Interaction*, October 2023. MAG ID: 4387612030 S2ID: b35b3b05ea7dae1d58b53b9eec14d3b9a5bfa6cd. doi:10.1145/3607822.3614532.
- [7] Vivek Bhardwaj, A. Anooja, Lovkesh Singh Vermani, Sunita, and Balwinder Kaur Dhaliwal. Smart cities and the IoT: an in-depth analysis of global research trends and future directions. *Discover Internet of Things*, 4(1):19, October 2024. doi:10.1007/s43926-024-00076-3.
- [8] Geoff Boeing. Topological Graph Simplification Solutions to the Street Intersection Miscount Problem. *Transactions in GIS*, 29(3):e70037, 2025. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.70037>. doi:10.1111/tgis.70037.
- [9] Bernardo Brito, Daniel G. Costa, and Ivanovitch Silva. Geospatial Risk Assessment of Cyclist Accidents in Urban Areas: A K-means Clustering Approach. In *2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON)*, pages 744–749, 2024. doi:10.1109/MELECON56669.2024.10608791.
- [10] Andrea Caragliu and Chiara F. Del Bo. Smart cities and the urban digital divide. *npj Urban Sustainability*, 3(1):43, July 2023. doi:10.1038/s42949-023-00117-w.
- [11] David Castells-Graells, Christopher Salahub, and Evangelos Pournaras. On cycling risk and discomfort: urban safety mapping and bike route recommendations. *Computing*, 102(5):1259–1274, May 2020. doi:10.1007/s00607-019-00771-y.

- [12] Miguel Costa, Manuel Marques, Carlos Lima Azevedo, Felix Wilhelm Siebert, and Filipe Moura. Which Cycling Environment Appears Safer? Learning Cycling Safety Perceptions From Pairwise Image Comparisons. *IEEE Transactions on Intelligent Transportation Systems*, 26(2):1689–1700, 2025. doi:10.1109/TITS.2024.3507639.
- [13] E. Rad, Fatemeh Kavandi, L. Kouchakinejad-Eramsadati, Kamran Asadi, and N. Khodadadi-Hassankiadeh. Self-reported cycling behavior and previous history of traffic accidents of cyclists. *BMC Public Health*, 2024. S2ID: e8996d5b70e739bbcf8956a7fff7f68c30da3e4d. doi:10.1186/s12889-024-18282-7.
- [14] Gian-Luca Savino, Gian-Luca Savino, Jessé Moraes Braga, Jessé Moraes Braga, Johannes Schöning, and Johannes Schöning. VeloCity: Using Voice Assistants for Cyclists to Provide Traffic Reports. *ACM Multimedia*, October 2021. MAG ID: 3206980414 S2ID: 15c3f864254ef6962f894a22be0c130c4e064151. doi:10.1145/3474085.3475509.
- [15] Hana Brůhová Foltýnová, Hana Brůhová Foltýnová, Eliška Vejchodská, Eliška Vejchodská, Eliška Vejchodská, Kristýna Rybová, Kristýna Rybová, Kristýna Rybová, Viktor Květoň, and Viktor Květoň. Sustainable urban mobility: One definition, different stakeholders' opinions. *Transportation Research Part D-transport and Environment*, 87:102465, October 2020. MAG ID: 3047866091 S2ID: 7fd00b96aefad0027397895d2ee9ca55e53b619. doi:10.1016/j.trd.2020.102465.
- [16] Haska Steltenpohl, Haska Steltenpohl, Anders Bouwer, and Anders Bouwer. Vibrobelt: tactile navigation support for cyclists. *International Conference on Intelligent User Interfaces*, pages 417–426, March 2013. MAG ID: 2074835449 S2ID: d850a947683cccea4ced1cbe826f691239d042cb. doi:10.1145/2449396.2449450.
- [17] I-Hsuan Peng, I-Hsuan Peng, Pei-Jun Lee, Pei-Chun Lee, Chen-Kang Tien, Chen-Kang Tien, Jyun-Sen Tong, and Jyun-Sen Tong. Development of a cycling safety services system and its deep learning bicycle crash model. *Journal of Communications and Networks*, pages 1–18, January 2022. MAG ID: 4226126693 S2ID: 7089c4170018ce71b2f14a5c50f6c818f1adf372. doi:10.23919/jcn.2022.000007.
- [18] Inese Mavļutova, Dzintra Atstāja, Jānis Grasis, Jekaterina Kuzmina, Inga Uvarova, and Dagnija Roga. Urban Transportation Concept and Sustainable Urban Mobility in Smart Cities: A Review. *Energies*, 16(8):3585–3585, April 2023. MAG ID: 4366773389 S2ID: 4675e7818ca5c9196717d5b2c8d2d4c4330603b4. doi:10.3390/en16083585.
- [19] Elvira Ismagilova, Laurie Hughes, Nripendra P. Rana, and Yogesh K. Dwivedi. Security, Privacy and Risks Within Smart Cities: Literature Review and Development of a Smart City Interaction Framework. *Information systems frontiers : a journal of research and innovation*, 24(2):393–414, 2022. Place: United States. doi:10.1007/s10796-020-10044-1.
- [20] Jose Sanchez Gracias, Gregory S. Parnell, Eric Specking, Edward A. Pohl, and Randy K. Buchanan. Smart Cities—A Structured Literature Review. *Smart Cities*, 6(4):1719–1743, July 2023. MAG ID: 4384067905 S2ID: 26e265750b363d2a1900bc2dfc3137ecce8754b2. doi:10.3390/smartcities6040080.
- [21] Josh Andrés, Josh Andres, Nathan Semertzidis, Nathan Semertzidis, Zhuying Li, Zhuying Li, Yan Wang, Yan Wang, Yan Wang, Florian Mueller, and Florian 'Floyd' Mueller. Integrated Exertion – Understanding the Design of Human–Computer Integration in an Exertion Context. *ACM Transactions on Computer-Human Interaction*, May 2022. MAG ID: 4280492076 S2ID: da6e81028401a1df383f31f278b721dde1ee41c7. doi:10.1145/3528352.

- [22] José Miguel Ferreira and Daniel G. Costa. Enhancing Cycling Safety in Smart Cities: A Data-Driven Embedded Risk Alert System. *Smart Cities*, 2024. S2ID: c85dda632680abc7a6a32326a60388c7107d204e. doi:10.3390/smartcities7040079.
- [23] mafaldadinis. mafaldadinis/Cycling-Safety, July 2025. original-date: 2025-07-16T17:52:42Z. URL: <https://github.com/mafaldadinis/Cycling-Safety>.
- [24] Mahdi Pedram, Seyed Ali Rokni, Marjan Nourollahi, Houman Hodayoun, and Hassan Ghasemzadeh. Resource-Efficient Wearable Computing for Real-Time Reconfigurable Machine Learning: A Cascading Binary Classification. In *2019 IEEE 16th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 1–4, 2019. doi:10.1109/BSN.2019.8771065.
- [25] Rosa Anna La Rocca and Rosa Anna La Rocca. Soft Mobility and Urban Transformation. *Tema. Journal of Land Use, Mobility and Environment*, 2, April 2010. MAG ID: 1570335838 S2ID: 8650d9c73f5c539acf03ec69f190fecdd012f663. doi:10.6092/1970-9870/125.
- [26] S Bosetti, S Bosetti, Carmelo Di Bartolo, C Di Bartolo, P Malgieri, P Malgieri, A Sitran, Alessio Sitran, A Sitran, Hana Brůhová Foltýnová, H Bruhova-Foltynova, Radomíra Jordová, R Jordova, P Kurfurst, P Kurfurst, D Smutkova, and D Smutkova. Policy Recommendations: For EU Sustainable Mobility Concepts based on CIVITAS Experience. January 2014. MAG ID: 578670729 S2ID: e9b293b076b7b2c25c382cb90ceceaceb8f2d1aa.
- [27] S. Rinaldi, S. D. Iacono, M. Pasetti, D. Astolfi, Dennis Brandão, A. Flammini, P. Ferrari, and E. Sisinni. Preliminary Analysis of Sensor Fusion Dataset for Cyclists’ Gesture Recognition. *International Workshop on Metrology for Automotive*, 2024. S2ID: b7cd3e8c33c17fe2832c5a80b36bfdcaac87226b. doi:10.1109/metroautomotive61329.2024.10615469.
- [28] Reem Al Sharif and Shaligram Pokharel. Smart City Dimensions and Associated Risks: Review of literature. *Sustainable Cities and Society*, 77:103542, 2022. doi:10.1016/j.scs.2021.103542.
- [29] Renato M. Silva, Gregório F. Azevedo, Matheus V. V. Berto, Jean R. Rocha, Eduardo C. Fidelis, Matheus V. Nogueira, Pedro H. Lisboa, and Tiago A. Almeida. Vulnerable road user detection and safety enhancement: A comprehensive survey. *Expert Systems with Applications*, 292:128529, 2025. doi:10.1016/j.eswa.2025.128529.
- [30] Anthony Simonofski, Estefanía Serral, Johannes De Smedt, and Monique Snoeck. Citizen Participation in Smart Cities: Evaluation Framework Proposal. July 2017. doi:10.1109/CBI.2017.21.
- [31] S. Soro. TinyML for Ubiquitous Edge AI. *ArXiv*, abs/2102.01255, 2021.
- [32] Ufuk Topcu, Nadya Bliss, Nancy S. Cooke, Missy Cummings, Ashley J. Llorens, Howard E. Shrobe, and Lenore D. Zuck. Assured Autonomy: Path Toward Living With Autonomous Systems We Can Trust. *ArXiv*, abs/2010.14443, 2020.
- [33] United Nations Human Settlements Programme (UN-Habitat). *The New Urban Agenda Illustrated*. 2020.

- [34] Van Sy Nguyen, Hyunseok Kim, and Dong-Hyuk Suh. Attention Mechanism-Based Bidirectional Long Short-Term Memory for Cycling Activity Recognition Using Smartphones. *IEEE Access*, 2023. S2ID: 28be537a9dea2a707cf2de552391234208d575a4. doi:10.1109/access.2023.3338137.
- [35] Tamara von Sawitzky, Tamara von Sawitzky, Thomas Grauschopf, Thomas Grauschopf, Andreas Riener, and Andreas Riener. “Attention! A Door Could Open.”—Introducing Awareness Messages for Cyclists to Safely Evade Potential Hazards. *Multimodal Technologies and Interaction*, 6(1):3–3, December 2021. MAG ID: 4206465794 S2ID: 55cf85025c7cef0f9b06c031fa9b5a1352a50aeb. doi:10.3390/mti6010003.
- [36] Yawen Wu, Zhepeng Wang, Zhengge Jia, Yiyu Shi, and Jingtong Hu. Intermittent Inference with Nonuniformly Compressed Multi-Exit Neural Network for Energy Harvesting Powered Devices. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. doi:10.1109/DAC18072.2020.9218526.
- [37] Xinyuan-LilyGO. GitHub - Xinyuan-LilyGO/TTGO_twatch_library at t-watch-s3. URL: https://github.com/Xinyuan-LilyGO/TTGO_TWatch_Library.
- [38] Huan Yan and Yong Li. A Survey of Generative AI for Intelligent Transportation Systems. *ArXiv*, abs/2312.08248, 2023.
- [39] Gokul Yenduri, M. Ramalingam, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, Rutvij H. Jhaveri, Ajay Bandi, Junxin Chen, Wei Wang, Adarsh Shirawalmath, Raghav Ravishankar, and Weizheng Wang. Spatial Computing: Concept, Applications, Challenges and Future Directions. *ArXiv*, abs/2402.07912, 2024.
- [40] Gangcheng Zhang, Yeshuo Shu, Keyi Liu, Yuxuan Wang, Donghang Li, and Liyan Xu. Vehicles, Pedestrians, and E-bikes: a Three-party Game at Right-turn-on-red Crossroads Revealing the Dual and Irrational Role of E-bikes that Risks Traffic Safety. *ArXiv*, abs/2411.02183, 2024.