Politecnico di Torino

Politecnico di Torino

Ingegneria Informatica (Computer Engineering)

A.y. 2025/2026

Graduation Session March 2026

Bridging the Accessibility Gap: Leveraging Egocentric Vision for Indoor Navigation

Supervisors:

Federico Manuri

Andrea Sanna

Candidate:

Michele Perseo

Abstract

Navigating unfamiliar indoor environments remains a significant challenge for Blind and Low Vision (BLV) individuals, particularly when navigation relies on interpreting spatial signage. Traditional assistive methods, such as smartphone-based computer vision or human assistance, often prove cumbersome or limit physical autonomy. This thesis presents a hands-free navigation system developed for the indoor spaces of Politecnico di Torino, utilizing Meta's Project Aria smart glasses. A preliminary study about preferences and needs of BLV individuals was done, switching the focus on what they actually need and would want from assistive technologies, especially based on smart-glasses. Based on these insights, a custom Android application was developed to interface with Project Aria's glasses processing the glasses' real-time RGB data. The mobile application employs a YOLO (You Only Look Once) model for object detection alongside Text Recognition to identify and interpret room names, directional signs, and safety exits. To ensure complete accessibility, the system utilizes a Voice User Interface (VUI) consisting of Speech-to-Text (STT) and Text-to-Speech (TTS) technologies, allowing for an entirely hands-free experience. Experimental results demonstrate that the integrated system successfully identifies corridor signs and provides real-time vocal guidance to the user. This study, focusing on Low Vision (LV) individuals, contributes to the field of assistive technology by offering a non-intrusive, wearable solution that enhances the independent mobility of LV students within complex institutional environments equipped with signage. This work wants to set a benchmark and starting point for possible ways to enhance a BLV person's life, providing help in completing any of their daily-life tasks.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Navigating an unknown environment may seem like a trivial task for anyone, but according to the World Health Organization there are about 2.2 billion people suffering from near or distance vision impairment [1]. Among those, in 2020 it was estimated that about 43 million people are completely blind and about 300 million suffer from medium or severe vision impairment (MSVI) [2], heavily affecting their daily routines. These numbers as shown in Figure 1.1 are, unfortunately, expected to grow in the next years, with a possible 700 million people that will suffer from blindness or MSVI by 2050 [3]. This urges the investment in studies and assistive technologies supporting this growing number of people.

While recent technological developments have enabled the emergence of wearable solutions as the latest and most advanced vision enhancement tools for the blind and low vision community in their daily lives, this has not always been the case. Historically, these individuals relied on traditional mobility aids, including widely-used, conventional tools, such as white canes and guide dogs. However, these well-established solutions are not exempt from limitations, such as the potentially elevated cost and training time in the case of guide dogs. It was only in the early 2000s that the rise of GPS infrastructure and the ubiquity of smartphone devices created the possibility for the generation of a new kind of navigation aids. One example is the BlindSquare application, developed in Finland and first released in May 2012. It enabled GPS-based audio navigation in outdoor settings, in a similar way to Microsoft's Seeing AI application, released for the first time in July 2017 for iOS. The latter narrates the world with the help of AI, scanning text, products, currency, and scene, supporting different languages and assisting with a great number of tasks. Nevertheless, the accuracy of GPS-based solution significantly degrades indoors, due to unavailable or unreliable satellite signals. To compensate for the aforementioned shortcomings of such technologies, infrastructure-dependent approaches were explored, encompassing systems based on Bluetooth Low Energy (BLE) beacons, RFID tags, and Wi-Fi fingerprinting. Although such solutions

result to be effective in controlled deployments, they require costly installation and maintenance, making them impractical at scale. Such operational challenges as those posed by traditional aids, GPS-based applications, and infrastructure-dependent systems, motivated the exploration of vision-based, wearable approaches. The field is therefore pivoting toward egocentric vision and smart glasses as an emerging and potentially more robust direction.

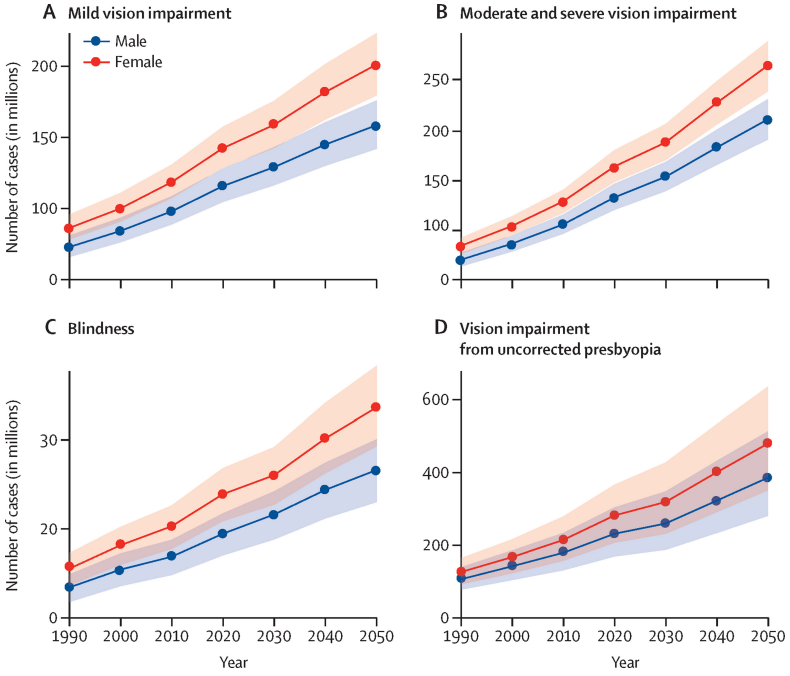

Figure 1.1: Forecast of number of people affected by mild vision impairment (A), moderate and severe vision impairment (B), blindness (C), and vision impairment from uncorrected presbyopia (D), all ages by sex, 1990–2050 [3]

1.1 Egocentric vision and smart glasses

The evolution and growing development of artificial intelligence technologies, combined with growing popularity of wearable devices, made egocentric vision one of the most interesting and studied challenges in academic research. This field exploits the acquisition of camera and sensor data from a wearable device, in order to recreate the user's point of view [4]. The most common among these wearable technologies, is often referred as smart glass. Numerous "smart glasses" solutions have already been studied and implemented in the light of accessibility and assisting visually impaired individuals. Studying these solutions is vital to understanding

what is feasible and what not, as well as to gaining insight on what may be commercially viable and sought-after by customers. Analyzing the existing solutions in various aspects, comprising their selected business plans, software architectures, and hardware designs, and identifying the differences between options, considerably helps to pinpoint the perfect combination of alternatives.

A few of the most notable existing solutions are considered in more detail in the paragraphs to follow. It is worth noting the differences between the various "smart-glasses" approaches that were used, and the fact that not all of them are focused on enhancing visually impaired individuals' lives.

1.1.1 Google Glass

One of the first companies to launch a smart glass in the modern sense was Google, in 2014, with their novel Google Glass [5]. It featured a semi-transparent display placed on the right eye that could be commanded mainly through a touch-pad, but at the same time supported hands-free functionalities by means of vocal commands. It was equipped with a 5-megapixel camera and WiFi internet access, and remarkably used bone conduction to propagate the output to the user's ear directly through vibrations. Although it was not a success in terms of sales and popularity, especially for privacy concerns and usability problems [6], the Google Glass marked the beginning of an era.

1.1.2 Ray-Ban Meta

Since Google Glass, there have been multiple attempts to further the innovation in the field of smart glasses, as explained by Lee et al. [7], all of which, however, faced some type of disapproval by the users. Only now with the rise of the Ray-Ban Meta glasses [8], we are experiencing the first true participation in our every-day lives of such devices. With their well-known and distinguishable design, they managed to enter our lives much easier than their predecessors. Focusing on accessibility, there is still a long way to go. Smart glasses like the Meta Ray-Ban glasses are not designed to help visually impaired people, or individuals with any type of disability. They are a product designed to be sold to the masses, that fits well with a style or outfit. At the same time, however, they offer a functionality like "Be my eyes" [9], where the wearer can connect via video-call to remote workers that will describe the user's surroundings, answer their questions and generally help them navigate the environment. This service can thus be considered as a first example of assistive technology for blind and low vision individuals using smart glasses, even though the assistance comes from another human rather than the glasses themselves.

1.1.3 Assistive Smart Glasses

Concerning strictly assistive devices, designed and produced with the only purpose to help the visually impaired, there are a few options available. For example, the Envision Glasses [10] took the Google Glass Enterprise Edition 2 [11] hardware, and built on top of it an ad hoc software application to exploit this device to help detect objects, read text, recognize faces and more.

Another example is OrCam MyEye [12], with similar features but a different base concept: it is a standalone attachable device that is mounted on top of the user's personal glasses. The Envision Glasses and OrCam MyEye have common technological characteristics among them, like the usage of a vocal user interface, and they are also comparable in market features, like their significantly elevated cost. Their prices, up to two thousand dollars for the first device and up to a 150 dollar monthly subscription for the second, make them hardly accessible to ordinary customers and users.

On the other hand, some devices like NuEyesPro4 [13] and eSight Go [14] (shown in Figure 1.2), use the definition of "smart glasses" quite differently, implementing OLED displays that replace traditional prescription lenses or see-through lenses like in the Google Glass. While they introduce a new approach, they do not differ much from the previously discussed devices in terms of affordability. Their adaptability is also limited, as they are primarily focused on specific tasks, like magnifying text at very large scales and projecting it onto the OLED displays.

(a) eSight Go smart glasses **(b)** NuEyesPro4 smart glasses

Figure 1.2: Smart glasses examples: eSight Go and NuEyesPro4.

In conclusion, all available options are moderately specialized on specific tasks and more crucially, they are economically inaccessible. However, they still provide precious information about the possibilities that can be explored and implemented. They all give central importance to the use of cameras and vocal interfaces, essentially making these modalities landmarks for future works.

1.2 Goal

The main purpose of this thesis is to explore the capabilities of Meta's Project Aria glasses in helping visually impaired individuals, finding and using augmented reality solutions to increase the quality of life for people with sight loss or sight deficiencies. As shown, the available technologies do not focus on guidance assistance, thus the chosen field of this study was indoor navigation, particularly helping Politecnico di Torino's low vision students navigate the university's spaces in full autonomy. Since it can take several minutes to move between different rooms and areas of Politecnico's campus, this is a crucial part of the everyday life of a student. Being able to do it autonomously and as quickly as possible can lead to a much more positive experience, leading to higher satisfaction and equity within the student community. On a larger scale, this thesis also aims at identifying ways to generalize these kinds of aids, in order to make them as accessible as possible for real-life and every-day use.

1.3 Approach

In order to achieve the stated objective, this work leverages the capability of the Project Aria platform to stream RGB camera frames in real time. The proposed system focuses on detecting and interpreting relevant visual cues present in the indoor environment of Politecnico di Torino, such as directional signs, room identifiers, and staircases.

Instead of reconstructing a global map of the environment or relying on pre-existing spatial information, the system adopts a semantic perception approach, where meaningful elements of the scene are directly identified and interpreted. Object detection and optical character recognition (OCR) models are used to recognize signage and extract textual information that can help the user navigate the campus environment.

The extracted information is then processed by the navigation logic of the system, which generates appropriate instructions that are delivered to the user through audio feedback. In this way, visually impaired students navigating the university corridors can receive guidance toward their desired destination without needing to independently detect or read environmental signage

1.4 Thesis structure

This work is structured into six chapters, each showcasing a critical step of the research and development process:

- **Chapter two** explores the state of the art, analyzing possible indoor navigation assistive technology while focusing also the needs of VI individuals.
- **Chapter three** details the methods and technological framework of the research. It focuses on the hardware and software limitations imposed by Meta Project Aria proposing a custom system to overcome those constraints.
- **Chapter four** focuses on the technical implementation of the proposed architecture. This chapter dives into the algorithmic choices and implementations.
- **Chapter five** presents how the system's performance experimental analysis. It measures latency times and accuracy obtained within a real-world campus environment.
- **Chapter six** summarizes the whole realization and evaluation process, making a critical analysis of the accomplishments obtained. It acknowledges the implementation's limitations and proposes possible future developments.

Chapter 2

State Of The Art

2.1 Indoor Navigation

Guiding a person in an outdoor environment, a phone, an internet connection and a GNSS (Global Navigation Satellite System) like GPS (Global Positioning System), have often proved to be enough to provide the fastest way to the desired destination. However, this does not apply to indoor spaces, where the GPS signal may not be able to provide accurate localization and positioning, making it almost impossible to guide the user. These spaces are defined as GPS-denied environments, and navigating them presents a long-time studied challenge. Two types of approaches can be found in the literature, depending on the architecture of the system: infrastructure-based systems, also referred to as radio-based [15], and infrastructure-independent or infrastructure-free systems [16].

2.1.1 Infrastructure-based systems

In this kind of systems, sensors and beacons are embedded in the environment. Technologies such as Bluetooth Low Energy beacons (BLE), WI-FI fingerprinting, and Radio Frequency Identification (RFID) provide precise positioning exploiting wireless communication with the user's device (e.g., smartphone). While these infrastructures offer high accuracy and performance, on the other hand, they present many challenges and disadvantages, such as being hardly scalable and having a high maintenance cost [17]. Deploying these systems implies an important financial investment, hardware upkeep, and constant battery replacement. This certainly limits their scalability and long-term sustainability, especially in large public or dynamically changing environments. The significant increment in installation and maintenance cost has been highlighted by several surveys on indoor positioning systems, as a consequence of the dense deployment of anchors or beacons to achieve the required meter-level accuracy [18, 19]. Furthermore, signal instability due

to multipath propagation, interference, and environmental changes can degrade positioning performance over time, requiring periodic recalibration [18]. For assistive navigation applications, particularly for blind and low-vision users, these limitations become even more critical, as the system must be continuously reliable and widely available rather than confined to instrumented buildings. Consequently, although infrastructure-based systems can achieve high positioning accuracy under controlled conditions, their dependency on pre-installed hardware and institutional maintenance presents substantial barriers to large-scale adoption.

2.1.2 Infrastructure-independent systems

Infrastructure-free (or infrastructure-independent) systems do not rely on pre-installed hardware within the environment. Instead, they exploit sensors embedded in personal devices such as smartphones, including inertial measurement units (IMUs), cameras, magnetometers, and barometers. Techniques such as Pedestrian Dead Reckoning (PDR), visual Simultaneous Localization and Mapping (SLAM), magnetic fingerprinting, and sensor fusion enable localization without requiring dedicated external infrastructure. These approaches eliminate deployment costs and significantly improve scalability, as they can operate in previously unseen or uninstrumented environments [18, 19]. Also called infrastructure-free, infrastructure-independent systems do not rely on pre-installed hardware within the environment. Instead, they exploit sensors embedded in personal devices, which may include inertial measurement units (IMUs), cameras, magnetometers, and barometers. Techniques such as Pedestrian Dead Reckoning (PDR), visual Simultaneous Localization And Mapping (SLAM), magnetic fingerprinting, and sensor fusion enable localization without requiring dedicated external infrastructure. These approaches eliminate deployment costs and significantly improve scalability, as they can operate in previously unseen or uninstrumented environments [18, 19].

However, infrastructure-free systems introduce their own technical challenges. Inertial-based methods suffer from cumulative drift errors over time, which can significantly degrade positional accuracy without periodic correction [19] . Vision-based SLAM approaches are sensitive to lighting variations, dynamic obstacles, and texture-poor environments, and impose considerable computational demands on mobile devices [20]. Additionally, magnetic field-based techniques may be affected by structural changes or electromagnetic disturbances within buildings. For assistive navigation applications targeting blind and low-vision users, reliability and safety are paramount; therefore, accumulated drift or temporary localization failures may pose usability risks [16].

Therefore, while infrastructure-free systems offer superior scalability, independence, and ease of deployment compared to infrastructure-based approaches, achieving robust, long-term, and safety-critical performance remains an active research

challenge. Recent work explores hybrid solutions that combine onboard sensing with opportunistic environmental cues to mitigate these limitations [16, 19].

2.1.3 Egocentric Vision

To address the limitations of both pure infrastructure-based and traditional infrastructure-free approaches, research has recently explored leveraging egocentric vision and semantic perception for indoor navigation. In these systems, wearable cameras and inertial sensors are used to build real-time visual-inertial maps of the environment while simultaneously detecting and recognizing salient features such as signage, labels, and other visual landmarks. Because these features are naturally present in most indoor spaces, using them as reference cues can be viewed as an opportunistic hybrid approach that retains infrastructure independence but enhances localization and guidance performance with environmental semantics rather than with deployed hardware. This combination of visual SLAM and semantic understanding has already been shown to improve positioning accuracy and contextual awareness in navigation systems designed for visually impaired users by generating not only spatial maps but also meaningful environmental information that can be communicated to the user [21, 22]. Specifically, wearable systems that implement RGB cameras, have demonstrated the ability to recognize objects and scene context in real time, which can be used to provide richer navigational feedback than geometry alone [23, 24]. Furthermore, AR research focused on low-vision users has investigated how augmented cues can make landmarks more perceptible and useful for navigation, improving orientation and decision making in unfamiliar indoor environments [21]. As a result, egocentric systems that exploit both motion tracking and visual semantics present a compelling pathway for indoor guidance that combines scalability with human-interpretable environmental cues, which is a crucial requirement when developing navigation support for blind and low-vision users.

2.2 Understanding the needs of LV Individuals

The fundamental starting point in the design of assistive smart glasses is understanding the genuine needs and expectations of blind and low vision individuals. To achieve a truly meaningful and useful invention, the design of every step must prioritize the lived experience of the intended users.

User-centered design for accessibility technologies is especially essential for Visually Impaired (VI) users, due to the vast diversity of needs among this demographic. [25] (Understanding Personalized Accessibility through Teachable AI: Designing and Evaluating Find My Things for People who are Blind or Low Vision)
Therefore, one of the very first phases of the research before choosing the main direction of the development for this thesis, was reviewing and comparing various studies regarding the needs of BLV individuals. Conclusions drawn from this examination pointed to the notable discrepancy between their answers and the current focus of academic research.

Another critical aspect found is that only in 18% of the studies, a visually impaired was involved in the study. After interviewing a sample of blind and low-vision individuals it was also found that many tasks deemed as important by researchers, were instead deemed unimportant by blind and low-vision individuals. Who was interviewed, also reported that most researchers focused on smartphone applications instead of head-mounted devices, which are much more preferred. This is what Gamage et al. [26] state in their work, where they reviewed 646 papers about assistive technologies for blind and low-vision people: "We found only a weak positive correlation between BLV participants' perceived importance of tasks and researchers' focus." This study sets a very important benchmark and guideline for this and what all future works relative to this topic should follow. For example, Fig.2.1 shows the preferences indicated by the interviewed who were asked to identify the five most useful features (the presented tasks are presented in order from most to least preferred).

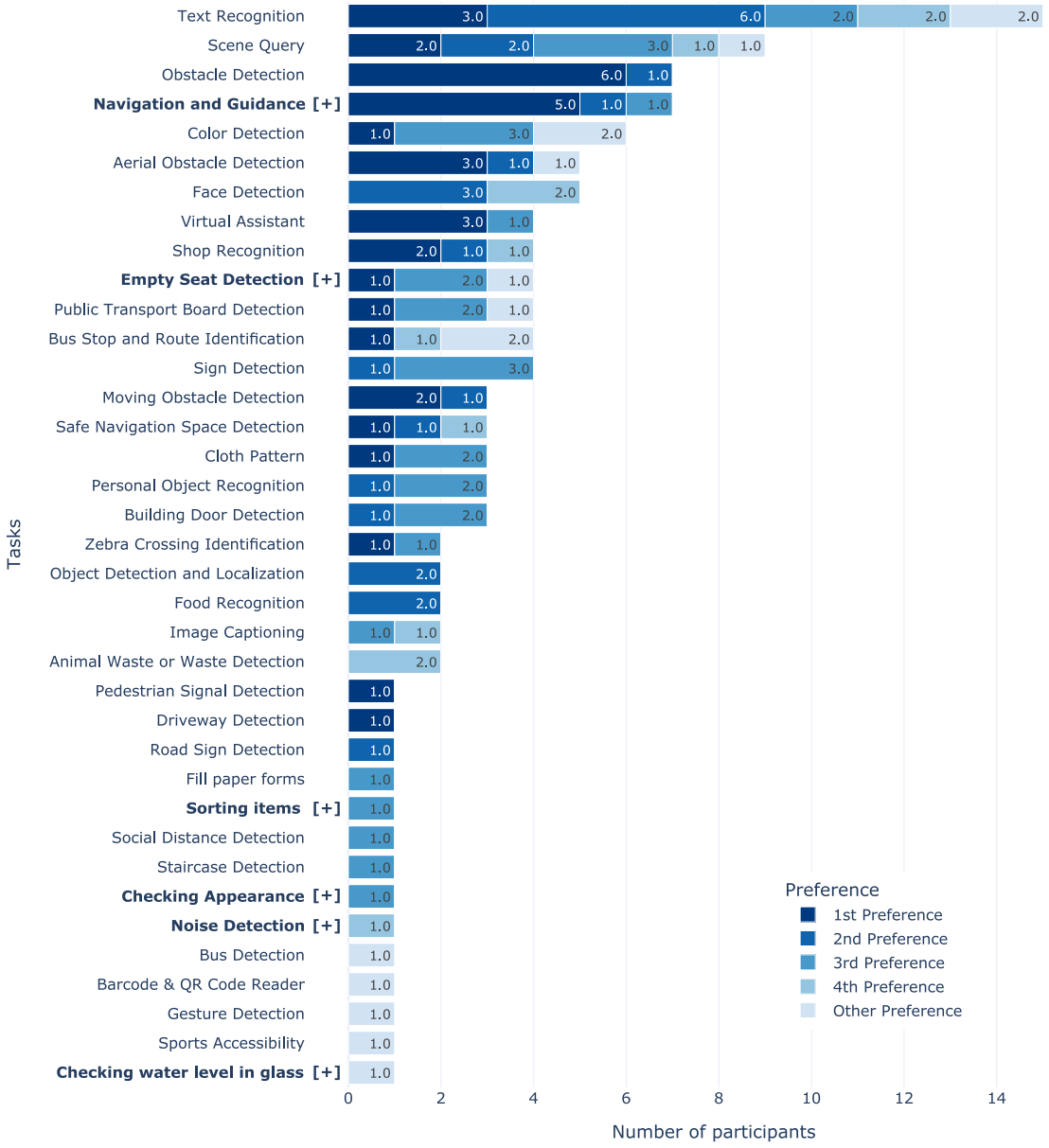

Figure 2.1: Participants' Five Most Useful Tasks Ranked by Order of Preference; "What do Blind and Low –Vision People Really Want from Assistive Smart Devices? Comparison of the Literature with a Focus Study" B. Gamage et al. [26]

What can be found is that the most requested tasks are "Text Recognition", "Scene Query", "Obstacle Detection" and "Navigation and Guidance". This indicates that not only participants want to be able to get text read to them or to be warned about obstacles, but also they would like to interact with the assistive

device through Scene Query.

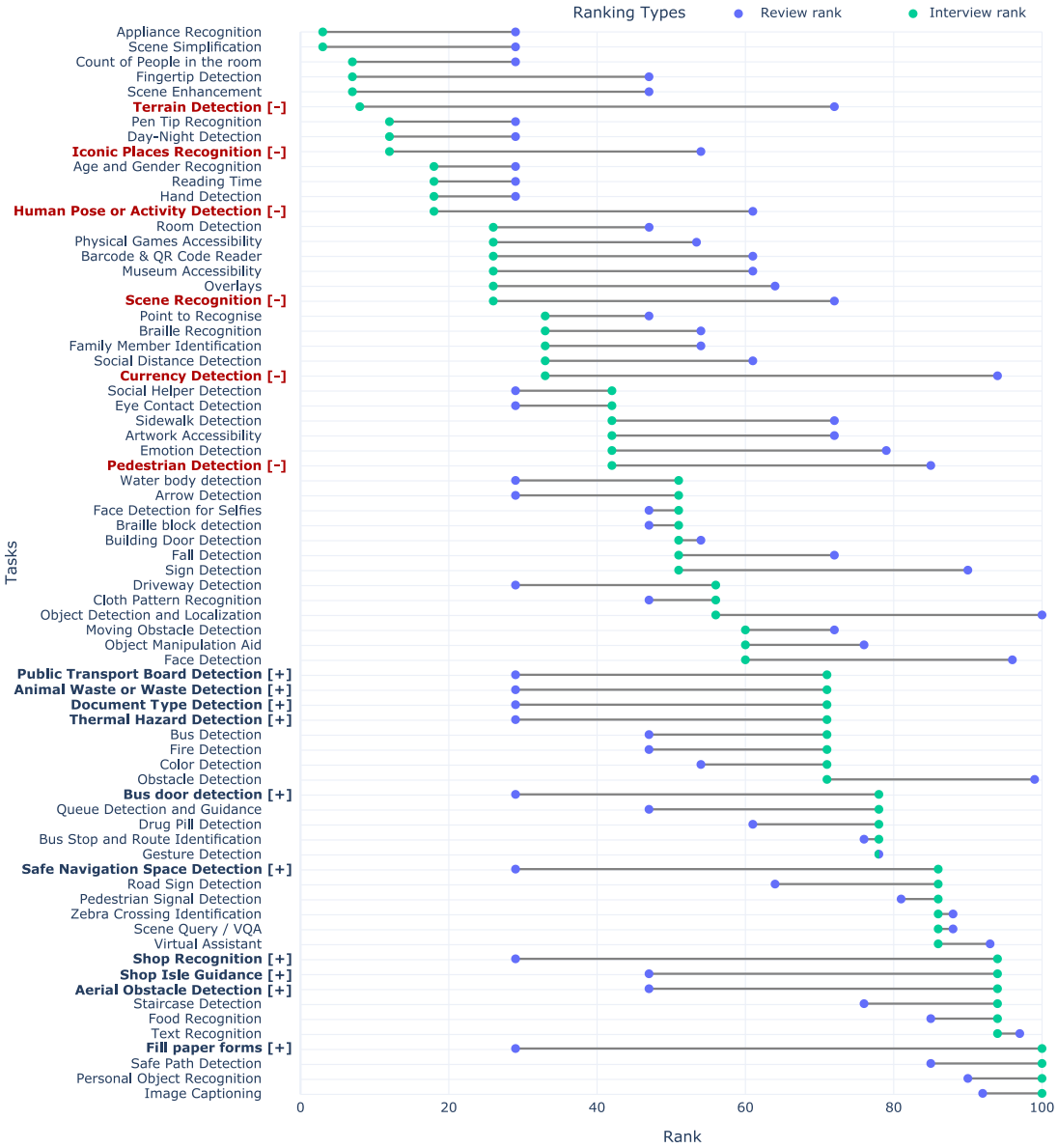

Figure 2.2: Ranking of various tasks based on the number of papers in the review [Review rank] and interview preference [Interview rank] [26].

This also emerges when comparing the number of papers in the review implementing a specific feature [Review rank] with the interview preference [Interview rank]. These findings align closely with those of Ruffieux et al. [27], who surveyed 50 visually impaired individuals across five distinct pathological groups -

AMD, Retinitis Pigmentosa, Glaucoma, Optical Nerve Lesion, and Others - using a combination of standardized and custom questionnaires. Their results reinforce the importance of prioritizing user needs, and further demonstrate that those needs vary considerably depending on the specific visual pathology. Across all groups, reading assistance and face recognition emerged as the two most universally requested functionalities, with 86% and 88% of participants respectively requesting them. This directly mirrors the top priorities identified by Gamage et al., where text recognition ranked first and scene understanding was among the most desired features. Notably, Ruffieux et al. also found that fewer than a third of participants (32%) were even aware that smart glasses technology existed as an assistive option - a striking finding that reinforces the concern raised by Gamage et al. about the disconnect between the research community and the actual VI user base. Moreover, Ruffieux et al. identified social interaction as a critically underserved domain: 94% of participants reported difficulties with face identification, and the vast majority struggled with visual attention awareness in both one-on-one and group conversations - a need that existing systems rarely address. This resonates with the "Scene Query" category highlighted by Gamage et al. [26], suggesting that VI individuals do not simply want passive information delivery, but rather an interactive, context-aware device capable of responding to the dynamic nature of real-world social environments. Taken together, both studies converge on the same core message: assistive smart glasses must be designed from the ground up around the expressed needs of their users, accounting for the diversity of pathologies, the primacy of reading and face recognition, and the largely unmet demand for socially-aware assistive functionalities.

2.3 Research studies on assistive technologies with smart glasses

Assessing and analyzing academic research focused on assistive technologies for the BLV community is a fundamental step. The goal is to identify benchmarks and understand the evolution of existing methodologies within the field. A review of the literature reveals how diverse approaches have been developed to tackle distinct navigational and environmental challenges, ranging from immediate obstacle avoidance to complex spatial mapping and personalized semantic understanding. Several studies focus on mapless, reactive navigation systems. For instance, Aladrén et al [28] proposed a mapless approach where an RGB-D camera detects obstacles using depth-sensing range expansion, prioritizing immediate physical safety over global environmental modeling. Taking this mapless philosophy further using modern everyday hardware, PathFinder [29] acts as a real-time wayfinding assistant that runs entirely on a smartphone. By processing monocular depth images, it

utilizes a depth-first search algorithm to identify the longest obstacle-free path without requiring prior knowledge of the environment.

In contrast, other research tackles pure turn-by-turn navigation by relying on pre-existing maps or active mapping algorithms. Systems like NavCog [30] exploit BLE beacons installed in the environment to achieve precise indoor localization and mapping. Similarly, the ISANA system proposed by Li et al. [22] utilizes a dedicated RGB-D tablet paired with a haptic SmartCane to provide a holistic indoor wayfinding solution. ISANA aligns visual odometry with pre-extracted architectural semantic maps to generate cognitive waypoint paths and perform dynamic obstacle avoidance. Bridging the gap between mapless reactivity and pre-mapped infrastructure, Chen et al [21], implemented a wearable navigation device based on real-time Semantic Visual SLAM. Rather than relying on static maps, this system actively builds a semantic map while navigating, combining localization with deep learning-based segmentation to interpret the scene at an object level.

Finally, a paradigm shift from rigid navigation to user-driven customization is evident in software-centric systems like ProgramAlly [31] Rather than providing a fixed spatial navigation tool, this approach utilizes Vision Language Models (VLMs) and Large Language Models (LLMs) to offer a Do-It-Yourself (DIY) end-user programming system. It empowers BLV users to design their own tailored visual assistance tools (e.g., filtering for a specific bus number), highlighting a broader research transition from strictly geometric obstacle detection to highly personalized, semantic scene understanding.

2.4 Summary and Expectations

Despite significant progress in indoor navigation focused on assisitng the BLV comunity, some problems still remain present. Infrastructure-based approaches have shown to be highly accurate but suffer from scalability, costs and maintainance constraints. On the other hand, infrastructure-free systems imrpove accessibility and scalability, but face challenges related to drift and robustness. Egocentric vision and semantic scene understanding seems to offer a valid compromise, but needs to be studied and validated further. Having also understood what BLV individuals expect from assistive technology, and having seen what already was studied and developed academically, there are all the elements to develop a new technology that incrorporates all these aspects. The review of BLV individuals exposed a big misalignment between research focus and what actually BLV individuals' priorities are. This thesis addresses these gaps by using Meta's **Project Ari**a to develop a navigation-centric solution tailored for the indoor navigation of a university campus. By focusing on the high-priority needs of "Navigation and Guidance" and

"Text Recognition," this work seeks to move beyond simple obstacle avoidance. The following chapters will detail how the raw sensor data from Project Aria is processed to create a seamless, autonomous navigation experience for students at Politecnico di Torino.

Chapter 3

Materials and Methods

This chapter picks up on what was found and discussed in the previous one. It will be discussed which methods and technologies can be employed to achieve the goal of developing an assistive tool for indoor navigation. From what was found in chapter 2

3.1 Meta Project Aria (Gen 1)

The primary hardware platform for this research is the **Project Aria (Gen 1)** research device. Developed by Meta Reality Labs [32], it serves as a high-fidelity egocentric sensor suite designed to capture multi-modal data that approximates the input of future AR glasses. All the data below are specified in Meta Project Aria official documentation [33].

3.1.1 Sensor Configuration

The Gen 1 device is a passive sensing platform optimized for high-quality data collection. The device is equipped with (Fig. 3.1):

- **Mono Scene (SLAM) Cameras**: Two global-shutter cameras (640x480 resolution) with a 150° FOV, recording at a maximum of 30 FPS. These are positioned to provide stereo overlap, facilitating Visual SLAM and 3D environment reconstruction.
- **RGB Camera**: A single rolling-shutter camera with a 110° FOV, operating at a maximum of 30 FPS, used for semantic scene understanding.
- **Eye Tracking**: Two inward-facing infrared cameras (320x240) track the wearer's gaze at a maximum of 90 FPS.

- **Non-visual Sensors**:
  - Two 1kHz Inertial Measurement Units (IMUs) provide the high-frequency motion data
  - One Barometer operating at 50Hz
  - One Magnatometer operating at 10Hz
- **Spatial Audio**: A 7-microphone array captures a 360° sound field.

The hardware supporting these sensors is composed of:

- A Qualcomm SD835 System on Chip, with a system memory of 4GB RAM
- A Flash memory (UFS) storage of 128GB
- Android 7.1 as the Operating system
- Software-configurable user button and switch

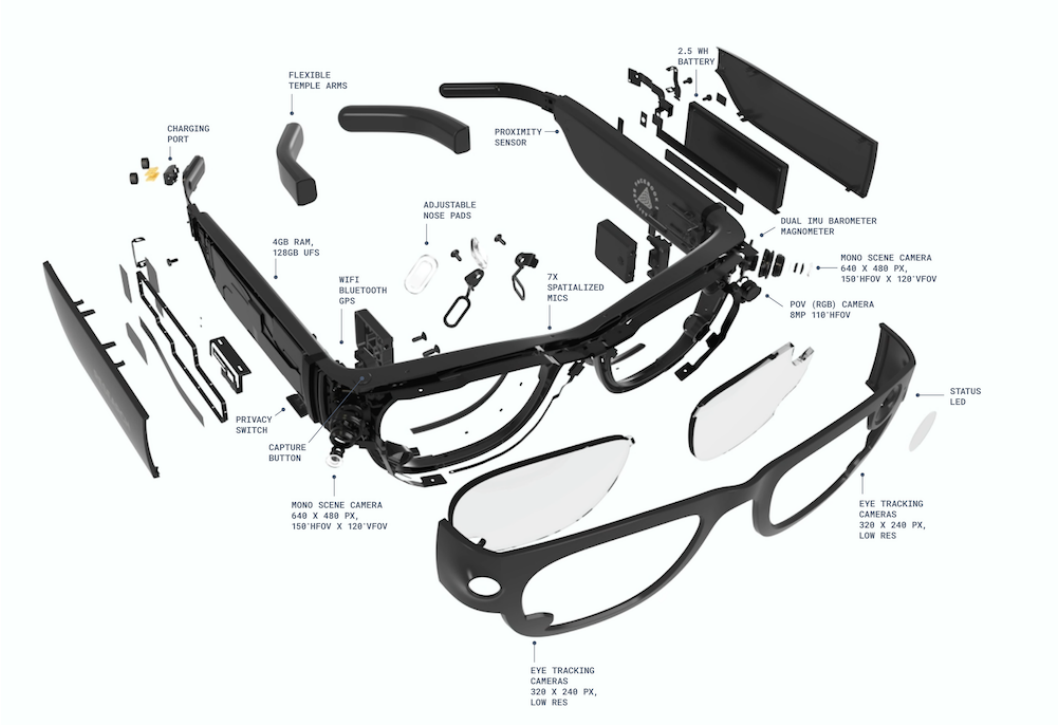

Figure 3.1: Project Aria Glasses Sensor Diagram

Moreover, the device is equipped with removable lenses that can be substituted with prescription lenses accommodating correction strengths between -4.5D and

+3.5D. This feature makes the smart glasses much more accessible to many low vision individuals, allowing them to integrate their lenses into the Project Aria system, instead of replacing them. However, there is still room for improvement regarding lens compatibility, as many LV users require corrective lenses beyond the currently supported Diopter range.
The device's battery life depends on the operating profiles, detailed in Table 3.1 and Table 3.2. During full-sensor constant recording, it is expected to be around 1.5 hours. Although it may increase to 30 hours when the glasses are used in "Profile 0", where the sensors' frame rates are at a minimum, the high-intensity real-time processing required for indoor navigation makes this relatively short battery lifespan a drawback for prolonged usage.
The possible recording profiles are listed below:

Prof.	**Microphones**		**ET Cameras**			**RGB Cameras**			
	Ch.	**kHz**	**Res.**	**FPS**	**Fmt.**	**Res.**	**FPS**	**AE**	**Fmt.**
0	7	48	320x240	10	JPEG	2880x2880	1	ON	JPEG
2	-	-	-	-	-	1408x1408	20	ON	JPEG
4	7	48	-	-	-	1408x1408	10	ON	JPEG
5	-	-	640x480	20	JPEG	1408x1408	20	ON	JPEG
7	7	48	-	-	-	1408x1408	10	ON	RAW
8	7	48	320x240	30	JPEG	1408x1408	5	ON	JPEG
9	7	48	320x240	10	JPEG	1408x1408	20	ON	JPEG
10	7	48	320x240	10	JPEG	1408x1408	10	ON	JPEG
12	-	-	320x240	10	JPEG	1408x1408	10	ON	JPEG
14	-	-	320x240	10	JPEG	1408x1408	1	ON	JPEG
15	7	48	320x240	10	JPEG	1408x1408	30	ON	JPEG
16	2	48	640x480	90	JPEG	1408x1408	10	ON	JPEG
18	7	48	320x240	10	JPEG	1408x1408	10	ON	JPEG
19	-	-	-	-	-	1408x1408	10	ON	JPEG
20	2	48	-	-	-	-	-	-	-
21	7	48	320x240	30	JPEG	1408x1408	15	ON	JPEG
22	7	48	320x240	10	JPEG	1408x1408	30	ON	JPEG
23	7	48	320x240	10	JPEG	1408x1408	30	ON	JPEG
24	-	-	-	-	-	2880x2880	10	ON	JPEG
25	-	-	-	-	-	1408x1408	10	ON	JPEG
26	2	48	-	-	-	2880x2880	1	ON	JPEG
27	7	48	320x240	10	JPEG	1408x1408	10	ON	JPEG
28	7	48	320x240	60	JPEG	1408x1408	30	ON	JPEG
29	2	48	-	-	-	2880x2880	1/0.1	ON	JPEG with decimation

Table 3.1: Project Aria reording profiles: Microphone, RGB camera and ET cameras configurations.

Prof.	SLAM (Mono)				GPS	IMU/Mag/Baro (Hz)				Scans (s)	
	Res.	FPS	AE	Fmt.	(Hz)	IMU1	IMU2	Mag.	Baro.	Wi-Fi	BT
0	640x480	10	ON	JPEG	1	1000	800	10	50	10	10
2	640x480	20	ON	JPEG	1	1000	800	10	50	10	10
4	-	-	-	-	1	1000	800	10	50	-	-
5	-	-	-	-	-	1000	800	-	-	-	-
7	-	-	-	-	1	1000	800	10	50	-	-
8	640x480	15	ON	JPEG	-	1000	800	10	50	-	-
9	640x480	10	ON	JPEG	-	1000	800	10	50	-	-
10	640x480	10	ON	JPEG	1	1000	800	10	50	10	10
12	640x480	10	ON	JPEG	-	1000	800	10	50	-	-
14	640x480	30	ON	JPEG	1	1000	800	10	50	-	-
15	640x480	30	ON	JPEG	-	1000	800	10	50	-	-
16	-	-	-	-	-	1000	800	10	50	-	-
18	640x480	10	ON	JPEG	-	1000	800	10	50	-	-
19	640x480	10	ON	JPEG	1	1000	800	10	50	10	10
20	-	-	-	-	-	1000	800	-	-	-	-
21	640x480	15	ON	JPEG	-	1000	800	10	50	-	-
22	640x480	10	ON	JPEG	-	1000	800	10	50	-	-
23	640x480	10	ON	JPEG	-	1000	800	10	50	-	-
24	640x480	10	ON	JPEG	-	1000	800	10	50	-	-
25	640x480	20	ON	JPEG	1	1000	800	-	-	10	10
26	-	-	-	-	1	-	-	-	-	-	-
27	640x480	10	ON	JPEG	-	1000	800	10	50	10	-
28	640x480	30	ON	JPEG	-	1000	800	10	50	-	-
29	-	-	-	-	1	-	-	-	-	-	-

Table 3.2: Project Aria recording profiles: SLAM Tracking and Environmental Sensor Specifications.

3.1.2 Aria Research Kit

The integration of the glasses into the research pipeline is managed via the Project Aria Research Kit, which includes a Software Development Kit (SDK), a Command Line Interface (CLI) and other tools to manage, record and stream data.

- **Aria Client SDK & CLI**: This interface allows for the configuration of recording profiles and the streaming of raw sensor data to a local workstation via Wi-Fi or USB.
- **VRS Data Format**: All captured data is stored in the VRS (Video Recording System) format, an open-source container optimized for multi-modal sensor synchronization.
- **TICSync**: This sub-component of the SDK is employed to ensure millisecond-level synchronization between the glasses and external sensors (if any) or to align data from multiple wearers in a shared space.
- **Machine Perception Services (MPS)**: Spatial AI machine perception capabilities to analyze egocentric data offered by Meta itself tailored to Project

Aria's data, enhancing accuracy compared to open-source algorithms. These services are offered as post-processing of VRS field via a cloud service.

- **SLAM services**: This set of MPS services offers analysis on SLAM plus IMU data, like 6DoF trajectory, Semi-dense point cloud, Online sensor calibration and Multi-SLAM where multiple recordings can be analysed if they are sharing the same co-ordinate frame.
- **Eye Gaze services**: Service allowing to track the eye gaze from the ET cameras.
- **Hand Tracking**: Service allowing hand tracking outputs, it needs SLAM cameras recordings.

3.1.3 Project Aria software and hardware limitations

As shown, Meta Project Aria Gen 1 offers many valuable and powerful features, but it does not certainly lack certain limitations due to both software and hardware design choices.

Software limitations

The SDK is currently available only as a Python library (Python versions 3. 9 to 3.12) and is officially supported only on a restricted set of environments:

- x64 Linux distributions of Fedora 36 (or newer) and Ubuntu 22.04 (or newer);
- Mac Intel or Mac ARM-based (M1) platforms with MacOS 11 (Big Sur) or newer.

This narrow support significantly reduces the number of compatible devices, completely excluding most portable and edge-computing platforms. For instance, devices like Raspberry PI and NVIDIA Jetson run on Linux ARM64 architectures, for which no official pre-compiled binaries were made available yet. More importantly, this limitation severely impacts mobile deployment, such as on Android smartphones. While Python environments can theoretically run on Android, the Aria SDK Python package relies heavily on pre-compiled C/C++ extensions. Because these native dependencies are not compiled for Android's specific architecture, deploying the SDK directly on a smartphone is impossible without manually cross-compiling the underlying C codebase using the Android NDK [34], a highly complex, and often unfeasible workaround.

Hardware limitations

One of the main hardware limitations comes from the mandatory need for a router that supports WI-FI 6. If it does not satisfy this requirement, live-streaming data over WI-FI will not work. In addition to this specification, the firewall should be disabled. This makes it impossible at the moment, as stated in the documentation, to stream data when connected to corporate, university, or public networks. Thus, few specific types of routers are recommended to correctly achieve live-streaming of the data.

3.2 System Requirements and Constraints

The system design should take into account the following requirements:

- **On-device computing**: the user should be able to move around comfortably, without additions to their normal load. Thus, running the system on an edge-computing platform would significantly enhance usability. Since dependence on heavy computational devices and complex cabling systems should be avoided, the optimal solution would be having a mobile device connected wirelessly to the Project Aria glasses, managing all the data coming from the device locally, avoiding also cloud services dependencies.
- **Real-time processing**: in order to guarantee useful instruction to the user, the system should be able to give a feedback in real-time. Delays longer than two seconds between an object detection and the system announcing it, would result in being cumbersome.
- **accessibility and hands-free use**: the system should be accessible through hands-free technology. BLV users indicate that this option as the preferred way, since it would not interfere with the usage of other assistive devices like white canes and guide dogs. Above all, voice inputs are the most requested to achieve this goal [26].

3.2.1 Functional Requirements

The functional requirements of the systems are defined as follows:

FR-1 The system shall retrieve real-time sensor data streamed from the Project Aria glasses over a WiFi connection.

FR-2 The system shall allow the user to input a desired destination using voice commands.

FR-3 The system shall provide navigation instructions to the user through audio feedback.

FR-4 The system shall perform real-time object detection on the incoming video stream.

FR-5 The system shall extract textual information from detected objects using optical character recognition.

FR-6 The system shall process and interpret the user's spoken destination command.

FR-7 The system shall compute a navigation path toward the selected destination.

FR-8 The system shall start the navigation process upon detection of a predefined voice activation keyword.

FR-9 The system shall terminate the navigation process upon detection of a predefined stop keyword.

FR-10 The system shall establish and maintain a wireless connection with the Project Aria glasses.

FR-11 The system shall initiate the streaming process of the Project Aria device via wireless communication.

FR-12 The system shall terminate the streaming process of the Project Aria device via wireless communication.

FR-13 The system shall execute machine learning inference locally on the mobile device.

FR-14 The system shall process video frames at a minimum rate sufficient for real-time navigation.

FR-15 The system shall convert navigation instructions into synthesized speech.

FR-16 The system shall handle temporary loss of wireless connectivity without system crash.

FR-17 The system shall synchronize incoming sensor data to ensure temporal consistency during processing.

FR-18 The system shall notify the user when reaching the desired destination.

ID	Description
FR-1	Retrieve real-time sensor data from Project Aria via WiFi
FR-2	Accept vocal destination input
FR-3	Provide audio navigation instructions
FR-4	Perform real-time object detection
FR-5	Extract text via OCR
FR-6	Interpret spoken destination
FR-7	Compute navigation path
FR-8	Start navigation via voice keyword
FR-9	Stop navigation via voice keyword
FR-10	Establish wireless connection with glasses
FR-11	Start streaming process remotely
FR-12	Stop streaming process remotely
FR-13	Run ML inference locally on mobile device
FR-14	Maintain real-time frame processing
FR-15	Convert navigation output to synthesized speech
FR-16	Handle connectivity loss
FR-17	Ensure synchronization of streaming data
FR-18	Notify user upon arrival at destination

Table 3.3: Functional Requirements summary

3.2.2 Non-Functional Requirements

The non-functional requirements establish performance, quality, reliability and usability constraints that the system should satisfy. These requirements ensure high quality real-time navigation assistance.

Performance Requirements

NFR-1 The end-to-end system latency, measured from video frame acquisition to audio feedback delivery, shall not exceed 300 ms under normal network conditions.

NFR-2 The system shall process video frames at a minimum rate of 10 frames per second.

NFR3-3 The machine learning inference time per frame shall not exceed 200 ms on the target mobile device.

NFR-4 Voice command recognition and interpretation shall be completed within one second.

Accuracy Requirements

NFR-5 The object detection component shall achieve a minimum mean Average Precision (mAP@0.5) of 70% on the evaluation dataset.

NFR-6 The text recognition component shall achieve a minimum character recognition accuracy of 70% under standard lighting conditions.

NFR-7 Voice command recognition shall achieve at least 80% accuracy in controlled indoor environments.

Reliability Requirements

NFR-8 The system shall automatically attempt reconnection in case of temporary WiFi disconnection.

NFR-9 Failure of a single processing module shall not cause total system crash.

NFR-10 The system shall tolerate packet loss up to 30% without critical navigation failure.

Usability Requirements

NFR-11 The system shall operate without requiring manual interaction during navigation.

NFR-12 The system shall sustain continuous operation for at least 30 minutes on a fully charged mobile device.

NFR-13 No video or audio data shall be permanently stored unless explicitly authorized.

NFR-14 The system shall notify the user when detection confidence falls below a predefined threshold.

Table 3.4: Summary of Non-Functional Requirements

ID	Category	Description
NFR-1	Performance	End-to-end latency shall not exceed 300 ms under normal network conditions.
NFR-2	Performance	The system shall process at least 10 frames per second.
NFR-3	Performance	ML inference time per frame shall not exceed 200 ms on the target device.
NFR-4	Performance	Voice command recognition shall complete within one second.
NFR-5	Accuracy	Object detection shall achieve mAP@0.5 $\geqslant$ 70%.
NFR-6	Accuracy	OCR shall achieve $\geqslant$ 70% character recognition accuracy under standard lighting.
NFR-7	Accuracy	Voice recognition shall achieve $\geqslant$ 80% accuracy in controlled indoor environments.
NFR-8	Reliability	The system shall automatically attempt reconnection after WiFi disconnection.
NFR-9	Reliability	Failure of a single processing module shall not cause total system crash.
NFR-10	Reliability	The system shall tolerate packet loss up to 30% without critical navigation failure.
NFR-11	Usability	The system shall operate without manual interaction during navigation.
NFR-12	Usability	The system shall sustain continuous operation for at least 30 minutes.
NFR-13	Privacy	No video or audio data shall be permanently stored without authorization.
NFR-14	Usability	The system shall notify the user when detection confidence falls below a threshold.

3.3 Design constraints

After meticulously analysing the requirements specified above, taking into consideration all limitations and capabilities of Project Aria, various possible implementations were studied in order to find the best hardware-software composition to achieve the desired objectives.

3.3.1 Platform compatibility and device choice

One of the first implementation-related issues to be addressed, was the compatibility with a computational device. In order to connect to the glasses, other than the Companion app [35], the only other way is to pair the device to a compatible platform as shown in 3.1.3. After the pairing process, which still requires the Companion app, it is possible to control the glasses from the device using either the CLI or the SDK. In this case a HP Victus [36] running Ubuntu 24.04 was used.

This configuration sets the starting point for developing the entire system. However, the goal of achieving on-device computing through an edge-computing device is not met. Consequently, a solution is required to remove the dependency on a host computer (e.g., a desktop or laptop). As previously stated though, the compatibility with the majority of mobile and edge-computing devices is strongly limited.

Many attempts of running the Project Aria SDK on an ARM64 based architecture were done, since it is clearly stated from the developers from Meta themselves, there is no available version yet for these platforms. Both tools like BeeWare [37] and Termux [38] were used in the attempt of having a running and working version of the Prject Aria SDK on Android. The first is an open-source tool used to exploit Python as programming language to create native mobile and desktop Android applications. Python libraries are supported through pre-compiled wheels. When these libraries rely on compiled C/C++ extensions, these must be specifically built for the target architecture. Thus, for the most popular versions this operation was done in order to make them available for anyone for Android development. Project Aria SDK unfortunately does not belong to this set yet. The same goes for Termux, which is capable of emulating an x86 architecture and run bash commands, but fails when installing the SDK due to missing native shared libraries. At the end the only way to get real time processing is to coonnect through a compatible device listed in the documentation, excluding all possible mobile and edge-computing devices to directly collect the streamed data. The found workaround for this particular scenario enforced by the limitations imposed by Project Aria design, is to exploit a compatible device as a server which downloads data from the glasses and forwards them to a mobile device connected on the same network.

3.3.2 Wireless connection

One of the biggest bottlenecks is the wireless connection aspect. To achieve a stable and functioning streaming over a wireless connection, Project Asria glasses needs a specific configuration. In order to leverage at best the 5Hz badnwith for WIFI streaming, a rotuter compatible with WIFI6 standard is needed. This requires an ad-hoc netowrk configuration in the environment and setting of the system. The device connecting through the SDK to the glasses, must be connected to the same

network as the glasses do. Since the projrct is developed in a university, as stted by the docuetnation, the campus' network can't be used for security reasons. When the streaming is attempted on such a type of network, the streaming will start but no data will be received by the connected device. The possible solutions are limited: either the glasses's personal hotspot network is used, or a network satisfying the pre-requisites is employed. The first option would drastically limit the range of motion, as the glasses would quickly move out of range of the device, assuming that the host computer remains stationary. The same limitation applies when using the host computer's hotspot. Therefore, carrying and operating a personal computer while navigating is not a practical solution. Thus the choice converged on using a custommizable router, which supprots WIFI6 and with no firewall enabled. Thankfully the connection to the glasses and the streaming process does not require internet connection, meaning any router that satisfies the constraints can be used, regardless of a functioning internet connection.

3.4 Deployment Environment

3.4.1 Campus Environment

Analyzing the environment where the system will be deployed adds useful information in order to develop the best system possible. From here the idea to exploit this characteristic in order to gather semantic understanding of the user's surroundings. If the system would be able to distinguish the meaningful signs and extraxt their displayed information, this already could be of big help for an LV student trying to navigate Politecnico di Torino's corridors trying to read those signs (Figure 3.2). An object detection model would easily learn to distinguish the different types of signage, including room numbers, exit signs, direction signs and more.

Figure 3.2: Example of Politecnico di Torino signage

3.5 Proposed System Architecture

3.5.1 Architecture Overview

The following figure shows the proposed architecture of the system. The design is based on all the constraints mentioned above and is thought of to achieve all the functional and non-functional requirements. The overall architecture of the

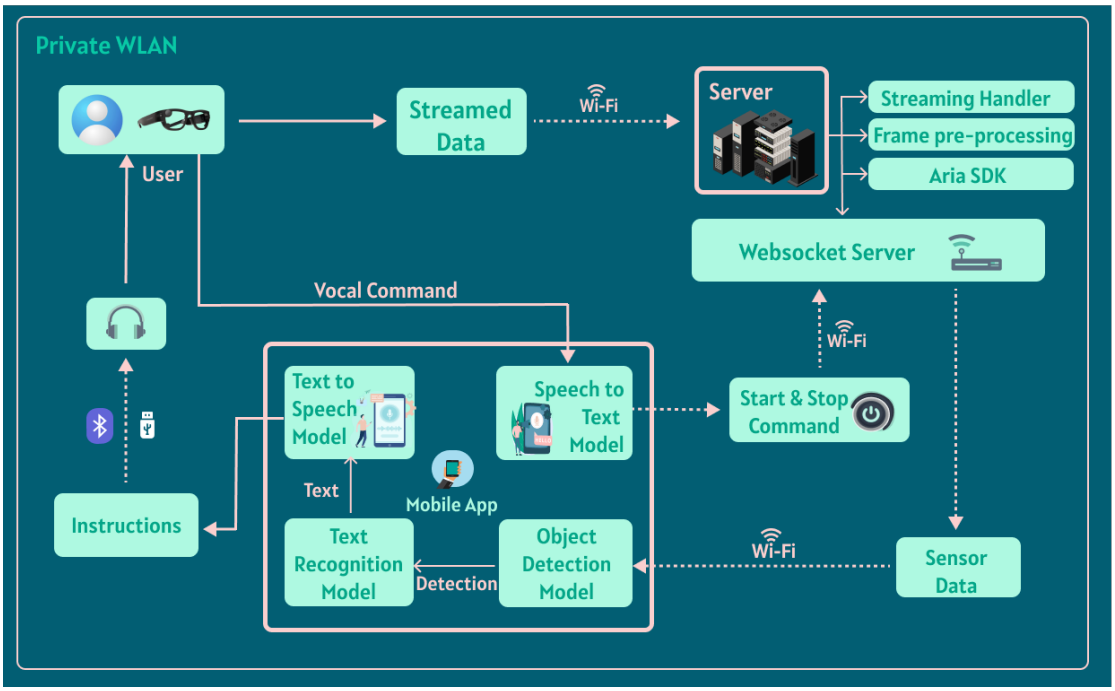

Figure 3.3: System Architecture

proposed system (Figure 3.3) is designed to overcome the hardware and software limitations imposed by the Project Aria platform while still enabling real-time assistive navigation. The architecture is composed of three main components: the Project Aria glasses, a server device responsible for handling the data stream using the Project Aria SDK, and a mobile device performing the main processing tasks.

The glasses capture RGB images of the environment and transmit them through a local WiFi network. The server receives the data stream and extracts the relevant sensor data using the SDK. These data are then forwarded to a mobile device through a WebSocket connection. The mobile device performs object detection, text recognition, and navigation logic, and provides audio feedback to the user through a speech interface.

This distributed architecture enables the system to leverage the streaming capabilities of Project Aria while maintaining the portability and usability required

for assistive applications.

3.5.2 Local Network

A TIM HUB+ [39] router is used. Its ability to support the WiFi-6 protocol makes it perfect for this use case. From the router management page, a WLAN without firewall is created. Then, Project Aria glasses are connected through the Companion app to the local network via WiFi, while the server is directly plugged to the router through an ethernet cable. The mobile device is also connected to the same WLAN through the WiFi connection. The router is not connected to the internet, so all the connected devices will not be able to access the internet.

3.5.3 Data acquisition

When navigation starts, the glasses enter in recording mode (indicated by a blue light turning on) and will start collecting data. All data is collected through the sensors included in the chosen recording (streaming) profile. In this case profile number 12 is used, since it allows maximum resolution from the RGB camera at a solid 10 FPS. The server receives all the streamed data using a streaming observer, which will handle each type of data accordingly.

3.5.4 Data forwarding

The server is responsible only for receiving and forwarding the sensor data generated by the glasses. Once RGB frames are extracted from the incoming stream, they are transmitted to the mobile device through a WebSocket connection established over the local network.

WebSockets were selected as the communication protocol because they allow persistent bidirectional communication with minimal overhead compared to traditional HTTP requests. This approach ensures low-latency transmission of image frames while maintaining a simple and lightweight communication architecture.

Each RGB frame received by the server is serialized and transmitted as binary data to the mobile device, which acts as the main computational node of the system.

3.5.5 Mobile Device Processing

The mobile device performs the core processing tasks of the system. Once the RGB frames are received from the server through the WebSocket connection, they are processed sequentially to extract semantic information from the environment. The processing pipeline consists of three main stages:

1. object detection;
2. text recognition;
3. navigation logic.

First, each frame is processed using a YOLO-based object detection model trained to recognize relevant types of indoor signage. These include directional signs, room numbers, exit indicators, and other navigation-related visual cues commonly found in university buildings.

When a sign is detected with sufficient confidence, the region of interest corresponding to the detected bounding box is extracted and passed to an Optical Character Recognition (OCR) module. The OCR component extracts textual information from the detected sign, enabling the system to identify relevant navigation instructions such as room identifiers or directional arrows.

The extracted semantic information is then interpreted by the navigation logic module, which determines whether the detected sign is relevant to the user's destination.

3.5.6 User Interface

Interaction between the user and the system is designed to be entirely hands-free, in order to accommodate visually impaired users who may already rely on assistive tools, such as white canes or guide dogs.

Voice commands are used to control the navigation process. Through speech input, the user can:

- start the navigation system;
- specify the desired destination;
- stop the navigation process at any time.

The interpreted commands are processed by the mobile application and used to control the navigation state.

When relevant information about the environment is detected, the system provides feedback through audio instructions generated using a text-to-speech engine. These instructions may include:

- identification of nearby signs;
- confirmation of detected destination;
- directional guidance based on detected signage.

The goal of this design is to provide clear and timely auditory cues while minimizing cognitive load for the user.

3.6 Design Choices and Architectural Trade-offs

The proposed architecture derives from both the hardware limitations showcased by Project Aria and the preferences of individuals from BLV about assistive navigation technologies. The main goal was to achieve a real-time processing performance, portability and usability for VI users.

3.6.1 Distributed System Architecture

The portability aspect was threatened by the Project Aria Client SDK platform compatibility. In order to maintain this crucial aspect, a distributed system was needed. The system, composed by a x8-based server and a mobile device, solves the compatibility problem, and keeps the portability requirement. On the other hand, a small latency is added due to the additional data transfer.

3.6.2 Edge Computing vs Cloud Computing

Executing the data processing pipeline locally provides several advantages, most notably the ability to operate independently from external network connectivity. Since the system relies on a local wireless network to stream sensor data from the Project Aria glasses, depending on remote cloud services would introduce additional latency and require stable internet connectivity. Given that the availability of networks capable of reliably supporting Project Aria's high-bandwidth Wi-Fi streaming is limited, maintaining a fully local processing pipeline represents a more robust solution.

Another possible approach would have been to rely on the Meta Project Aria Machine Perception Services (MPS), which provide advanced processing capabilities such as visual-inertial SLAM, trajectory estimation, and environmental reconstruction. However, MPS currently operates only on recorded datasets rather than live streams, with processing times that typically range from several minutes depending on the recording length. Consequently, these services cannot be used in real-time assistive applications where immediate feedback to the user is required.

If real-time access to the MPS pipeline had been available, it would have enabled the use of the dedicated SLAM cameras integrated into the Project Aria device. These cameras are designed to operate within Meta's visual-inertial SLAM framework, which exploits tightly synchronized camera and IMU measurements together with precise intrinsic and extrinsic calibration parameters. Within this optimized pipeline, the system is able to compensate for the strong geometric distortion introduced by the wide field-of-view lenses and to accurately fuse the information coming from the multiple sensors.

Outside of this framework, however, directly using the raw SLAM camera streams becomes significantly more challenging. The cameras produce highly distorted fisheye images that require precise calibration and rectification, and their spatial placement on the frame of the glasses results in a relatively limited overlapping field of view. Without access to the calibration pipeline and optimized algorithms provided by the MPS ecosystem, implementing a robust stereo or visual-inertial SLAM system from the raw streams would require substantial additional development effort.

For these reasons, the proposed system focuses instead on semantic perception using the forward-facing RGB camera, applying object detection and optical character recognition techniques that can operate effectively on individual frames while still supporting real-time navigation assistance.

3.6.3 Perception Strategy: Object Detection

As discussed, an object detection model is used in order to extract semantic information to guide the user's navigation. The YOLO model is known for its real-time performance and for the possibility to easily execute it on mobile devices. Compared to heavier models, it can deliver a slightly lower accuracy, which is compensated with a better computational efficiency. On mobile devices, due to smaller computational capabilities, execution times levitate, giving much more importance to more efficient models.

Chapter 4

System Implementation

This chapter focuses on describing the detailed implementation of the system, starting from the data collection, moving to the websocket usage and ending with the kotlin-based application that handles object detection, text recognition and the user interface.

4.1 Server Implementation

One of the two main parts of the system is represented by the server hosted on a host platform compatible with Project Aria Client SDK. In this case, the server is entirely written with Python programming language. The choice is almost inevitable, since the SDK itself is a Python library.

4.1.1 Device Authentication and Connection

The server's main task is to connect to the glasses and retrieve the data it produces. When the system is started, the server checks if there is an already paired device, if negative, a pairing procedure is initiated (this needs the Project Aria device connected via USB to the host). Then, given the device's IP address, the server tries to reach the device at that address. The IP address is retrieved from a configuration file (config.ini) so that it can be changed easily at any moment. When connecting from Linux based platforms, the SDK automatically runs a command to update the machine's IP tables in order to enable incoming UDP traffic (on ports 7000-8000), required for Data Distribution Service (DDS) required by the glasses.

4.1.2 Streaming Handling

The streaming activity is handled by a data class defined as StreamingHandler. In this class, all the streaming configurations, like the streaming profile and the

streaming interface, are set and assigned to a streaming client, which is an attribute of the SDK's streaming manager class. Other than that, the life-cycle of the streaming is controlled and kept alive until the server receives a stopping command. In order to handle all the different types of data the glasses can collect, a streaming observer is hooked to the streaming client. In the observer class a different callback is instantiated for each transmitted data type.

4.1.3 Asyncronous Behaviour

The system utilizes the asyncio library [40] to orchestrate concurrent task execution. A custom AsyncEventBus is implemented to bridge the gap between the Project Aria SDK's dedicated background thread and the application's main asyncio event loop.

In this architecture, sketched in Figure 4.1, the StreamingObserver acts as a producer: when a new sensor frame is received on the SDK thread, it uses asyncio.run_coroutine_threadsafe to securely publish an Event object to a specific topic of the bus. The bus manages these events using asynchronous queues specific for each topic (asyncio.Queue), which buffer the data until it is consumed by background workers, such as the WebSocket. This decoupled design ensures thread-safe internal communication and prevents high-frequency sensor data from blocking the main application logic.

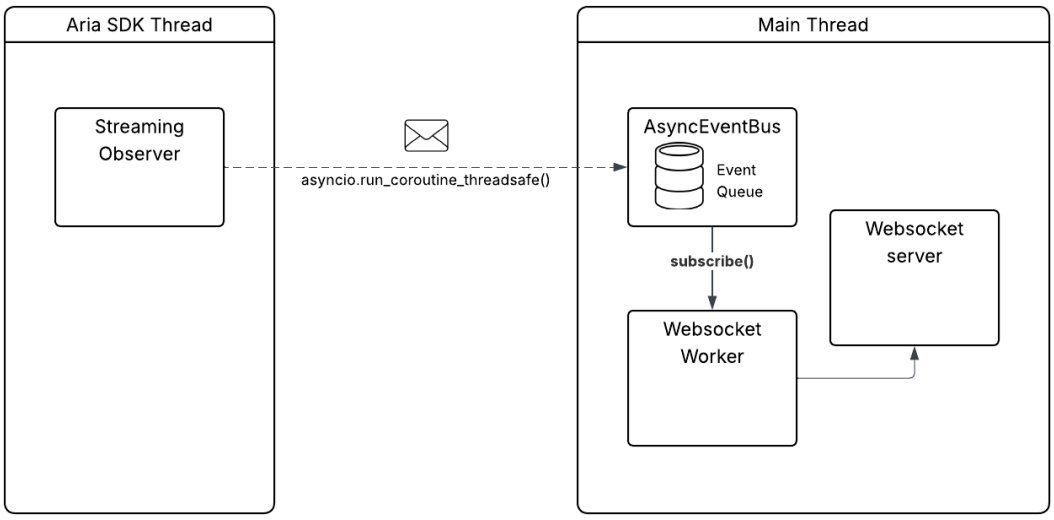

Figure 4.1: Server architecture

4.1.4 Websocket Implementation

In order to implement the wireless communication between the server and the mobile device, the Python Websockets library was employed [41]. As for the IP address before, the port on which the connection is hosted, is taken from the config.ini file. In this case port 8080 is used. The server starts by binding to all available network interface, making itself available on the address 0.0.0.0. In this way, the server is able to accept connections both from local and external clients on the specified port.The server's operational stability is further reinforced through several parameters configured in the websockets.serve call:

- **max_size**: Limits the maximum accepted message size to 2MB to prevent memory exhaustion from excessively large data payloads.
- **write_limit**: Establishes a high-water mark for the write buffer to manage outgoing data flow effectively.
- **ping_interval and ping_timeout**: Both set to 20 seconds to actively monitor connection "liveness." This ensures that stale or silent connections are detected and terminated promptly, maintaining the integrity of the communication channel.
- **close_timeout**: Provides a 10-second grace period for the WebSocket closing handshake to finalize, ensuring a graceful termination of resources before the connection is forcibly closed.

  In this particular implementation only one client at the time can be connected and stored in the connected_client variable. As soon as the client disconnects, the server handler catches the ConnectionClosed exceptioin and clears the reference to ensure system reliability.

```
async def start(self):
    """Start the WebSocket server."""
    server = await websockets.serve(
        self.client_handler,
        "0.0.0.0",
        self.port,
        max_size=2*1024*1024,
        write_limit=2**176,
        ping_interval=20,
        ping_timeout=20,
        close_timeout=10
    )
    await server.wait_closed()
```

Listing 4.1: Websocket server inititialization

4.1.5 WebSocket Worker

Once a connection with a client is successfully established, the server begins collecting frames and preparing them for transmission. The WebSocket worker is implemented as an asynchronous task that continuously subscribes to the `rgb_frame` topic of the `AsyncEventBus`. Whenever a new frame is published to this topic, the worker retrieves the frame and begins a pre-processing pipeline before forwarding it to the connected client.

In order to transmit frames efficiently while preserving sufficient visual information, several image processing steps are applied. First, the frame is converted from RGB to BGR format, which is required by the OpenCV library for proper image processing. Then, the image resolution is reduced by resizing the frame to a fixed width of 800 pixels while preserving the original aspect ratio. This downscaling significantly reduces the amount of data that needs to be transmitted.

After resizing, the frame is compressed using JPEG encoding through the OpenCV `imencode` function with a quality factor of 90. This compression step further reduces the payload size, allowing faster transmission over the WebSocket connection while maintaining acceptable image quality.

To maintain a stable streaming pipeline and prevent processing bottlenecks, the worker ensures that only one frame is processed and transmitted at a time. If a new frame arrives while the previous one is still being processed or sent, the incoming frame is skipped. This strategy avoids queue buildup and prevents increasing latency in the stream.

4.2 Android Application

The core logic of the system is handled via an Android application developed with the Kotlin programming language. Android Studio was used as the Development Environment (IDE).

4.2.1 Project Configuration

The project was developed using the following configuration:

- **Build Tool**: Gradle version 8.13.
- **Android Gradle Plugin (AGP)**: version 8.11.2.
- **Java Compatibility**: The project is configured to use Java 17 for both source and target compatibility.
- **Kotlin Version**: version 2.2.21.

- **Kotlin Compiler Target**: Configured for JVM 17.
- **Compile SDK**: API Level 36.
- **Target SDK**: API Level 36.
- **Minimum SDK**: API Level 33 (Android 13), ensuring the app runs on modern Android devices.
- **Supported Architectures (ABI Filters)**: The application supports armeabi-v7a, arm64-v8a, x86, and x86_64.

4.2.2 Object Detection Model

One of the core functionalities of the application is the object detection process. The model chosen was YOLO (You Only Look Once), developed by Ultralytics and available through the PyTorch library. More specifically, YOLOv11 was selected since, at the time of development, it represented the most advanced and best-performing version of the architecture, as shown in Figure 4.2.

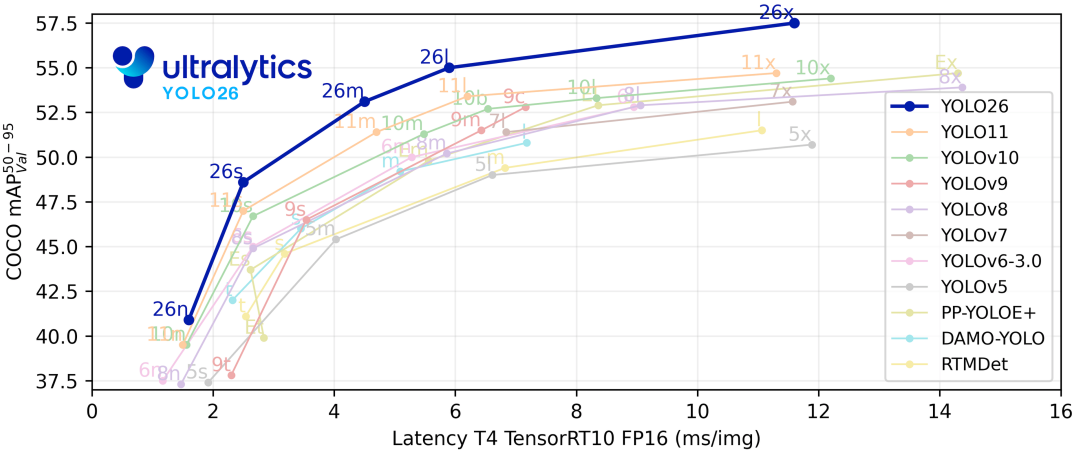

Figure 4.2: YOLO versions comparison

Since the goal was to deploy the model on a mobile device, the smallest available variant was used: YOLOv11n (nano). As reported in the official Ultralytics documentation [42]:

As reported in Table 4.1, the nano variant excels in execution speed, which is fundamental to guarantee real-time performance on device. Although it achieves a slightly lower mean average precision (mAP) compared to larger variants, it still delivers reliable detection results for the target use case. Depending on the

Table 4.1: YOLOv11 performance based on model size

Model	**size (pixels)**	**mAPval 50-95**	**Speed CPU ONNX (ms)**	**Speed T4 TensorRT10 (ms)**	params (M)	FLOPs (B)
YOLO11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9

hardware available, heavier variants could be employed; however, in order to ensure broader compatibility across a wide range of Android devices, the nano variant was preferred. The choice was further motivated by compatibility requirements. The model needed to run inside a native Android application, without access to the Python-based machine learning libraries typically used in desktop or server environments. Given that YOLO is the most widely adopted architecture for real-time object detection, it benefits from extensive community support and well-established tooling for cross-platform deployment, making it the most practical choice for bridging this compatibility gap.

Model Training

The selection of a training dataset is one of the most critical factors in obtaining strong performance from a machine learning model. Since the objects of interest in this use case are highly specific - consisting exclusively of the signage found inside Politecnico di Torino's campus buildings - a custom dataset was the only viable option, as no existing public object detection dataset contains this type of signage. After a thorough review of the navigation-relevant visual elements present throughout the campus, the following object classes were identified:

- **room**: a sign indicating the name of a specific classroom or study room on campus, characterised by an L-shaped form with a distinctive yellow-orange colour.
- **room_direction_left** and **room_direction_right**: signs mounted on corridor walls at intersections or turns, indicating the direction in which a group of classrooms or study rooms can be found.
- **exit_left**, **exit** and **exit_right**: signs indicating the presence of, or the direction toward, the nearest emergency exit.
- **stair_sign**: a sign indicating that a group of classrooms or study rooms is located on another floor, characterised by a staircase symbol.

- **staircase**: a class for detecting physical staircases leading to another floor.

The directional distinction between left and right variants was introduced to extract navigation direction information directly from the detection output, rather than relying on the OCR stage to interpret arrow symbols. This was a deliberate design choice: directional arrows used on campus signage are non-standard characters outside the ASCII set and would not be reliably recognised by conventional text recognition models. Data collection was performed directly using the Project Aria glasses, ensuring that the training images closely matched the perspective and optical characteristics of the deployment device. For each class, multiple video sequences were recorded from different angles and under varying lighting conditions. While Ultralytics recommends a minimum of 200 annotated frames per class for training, a minimum of 500 frames per class were collected and annotated to improve robustness, with an additional minimum of 100 frames per class reserved for the validation set.

TensorFlow Lite Conversion and Android Integration

The next step was converting the trained model into a format compatible with Android deployment. To achieve this, the model was exported to the TensorFlow Lite format [43], Google's lightweight inference engine optimised for mobile and embedded hardware. TensorFlow Lite enables hardware-accelerated inference via the Android Neural Networks API (NNAPI), the GPU delegate, and the Hexagon DSP delegate - all of which are unavailable to raw PyTorch models and are essential for achieving real-time throughput on device. The conversion can be performed directly through a command provided by the Ultralytics library:

```
yolo export model=yolo11n.pt format=tflite imgsz=640
```

This produces a self-contained `.tflite` binary file encoding the complete model graph and weights in TensorFlow format. The converted model file, along with a metadata file containing the class label definitions, is copied into the Kotlin project as an asset. To execute the model within the application, a dedicated `YoloDetector` class was developed to manage the interpreter's lifecycle, pre-processing, inference, and post-processing. The implementation builds upon the open-source Android YOLO integration by Esteban Uri [44, 45], which provides a robust foundation for all stages of the pipeline. Inference is hardware-accelerated through the device's GPU delegate where available, with automatic fallback to CPU execution on devices without dedicated GPU support. The detection pipeline executes the following steps for each incoming camera frame:

- the raw frame bytes are decoded into an Android `Bitmap` object;

- the bitmap is resized to the model's expected input resolution and pixel values are normalised to the range [0,1][0, 1] [0,1];
- inference is run on the prepared bitmap, producing a large set of candidate bounding boxes with associated class scores across all detection scales;
- a confidence threshold filter is applied to discard all candidates below the minimum score (0.25), retaining only high-confidence proposals;
- Non-Maximum Suppression (NMS) is applied to remove redundant overlapping detections, retaining only the most confident prediction for each distinct object instance;
- the surviving detections are rescaled from the model's normalised coordinate space back to the original frame's pixel dimensions.

At the end of each frame's processing cycle, the detection pipeline returns a set of results for each identified object:

- **Confidence**: a floating-point value in the range $[0, 1]$ representing the model's certainty that the detected region contains an object of the predicted class. It is the product of the objectness score and the class probability assigned by the detection head. Only detections exceeding the confidence threshold (set to 0.25 in this implementation) are retained after post-processing.
- **Label**: a string identifying the predicted object class among the nine categories defined during training (*room*, *direction_left*, *direction_right*, *exit*, *exit_left*, *exit_right*, *stair_sign*, *staircase*, *wc*). The label is resolved by mapping the class index returned by the model to the corresponding entry in the metadata file bundled with the `.tflite` asset.
- **Bounding box**: a set of four integer pixel coordinates (x_{min}, y_{min}, x_{max}, y_{max}) delimiting the rectangular region of the original frame in which the object was detected. The coordinates are expressed in the reference frame of the camera preview surface after the inverse letterbox rescaling step, and are used both to overlay the detection rectangle on the live preview and to crop the region of interest passed to the subsequent OCR stage.

4.2.3 Text Recognition Model

To extract the text displayed on the signs of interest, a text recognition model is used to obtain optical character recognition (OCR). Rather than running OCR on the full camera frame - which would be computationally wasteful and prone to interference from background text - the system restricts recognition to the region

of interest already localised by the object detector. The bounding box predicted by YOLOv11 is passed directly to the TextRecognitionProcessor class, which manages the complete OCR pipeline.

Pre-processing.

Before recognition is attempted, the bounding box coordinates are validated and clamped to the bitmap dimensions via `validateAndClampBoundingBox`, ensuring that detection boxes that partially extend beyond the frame boundaries do not cause an out-of-bounds crop. The validated region is then extracted from the full-resolution frame as an independent Bitmap. Because the object detector operates on a downscaled 800×800 input, some of the resulting crops can be very small in pixel terms. If either dimension of the crop falls below 100 pixels, the image is upscaled by a factor of 5× using bilinear interpolation before recognition, since ML Kit's text detector performs significantly better on larger glyphs. Finally, a binarization step is applied to the (possibly upscaled) crop: the image is converted to a black-and-white representation, which improves OCR robustness under variable lighting conditions by reducing the sensitivity to shadows, reflections, and low-contrast text.

Recognition

The pre-processed bitmap is wrapped in an `InputImage` and submitted to Google ML Kit's TextRecognizer, configured with the default Latin-script options. The call is made asynchronously using Kotlin coroutines (await()), so it does not block the camera processing thread. ML Kit's on-device recognition model returns a structured Text object containing a hierarchy of TextBlock, Line, and Element objects, each carrying both the recognised string and its bounding box within the crop.

Output and diagnostics

If the recogniser returns a non-blank result, the raw text string is extracted and returned to the caller for further use. If recognition yields no text, the function returns null and all intermediate bitmaps are explicitly recycled to avoid memory pressure. The `TextRecognizer` also returns the bounding box of the recognized text, as shown in Figure 4.3.

4.2.4 User Interface

To guarantee hands-free usage, the application is also completely usable through vocal commands. To achieve this, a speech-to-text (STT) model called Vosk and

Figure 4.3: Example of recognized text in a frame

the Google text-to-speech (TTS) engine were chosen.

Vosk

Vosk is a speach recognition toolkit developed by alphacephei. It supports more than 20 languages and more importantly it completely runs offline. Running without the need to reach the internet was a very important aspect since during the usage of the system no internet connection is available. In this way also the user's privacy is protected, since everything is processed locally. Instead, the Google speech-to-text model was discarded since it would run offline only with the language set as default on the used device. Thus, to guarantee a better compatibility with any type of device and the user's spoken language, Vosk was chosen instead using the English language model `vosk-model-small-en-us-0.15` package.

Vocabulary

To avoid noisy speech recognitions, a closed grammar was employed. Since the possible destinations do not include common-spoken words, but more often letters followed by a number, the model would struggle to understand correctly words like "R1". This is why the possible words understandable by the model are limited to the possible destinations available. The same approach is used when asking the user for confirmation commands, limiting the possible answers only to those relevant for the scenario. So if the user is asked to confirm the destination said earlier, only "yes" and "no" will be recognised by the model. This approach significantly improves accuracy and eliminates the need to repeat what has been said multiple times until it is understood correctly.

Wake word system

The application is also able to start without touching the "start" button. In fact the application will be "listening" for a wake word as soon as it is opened. The

task is run in background and enables the detection of only three possible words: start, stop and cancel. When the word "start" is detected, and the system has not started yet the navigation, the user will be asked to insert a destination and the navigation process will start. Instead, if the navigation is running and the user needs to stop the system for any reason, the "stop" word said at any moment will shut down the connection with the server, stop the Project Aria streaming process, and put the application back to the listening state waiting for the start command. The "cancel" word instead can be used to interrupt any in-progress interaction and go back to the listening state. This approach of having a wake word followed by active listening avoids the overhead of running the whole vocabulary continuously and avoids spurious activations from ambient noise.

Navigation Goal Capture

Once the wake word is detected, the system transitions to active listening mode and the TTS engine prompts the user with *"Where do you want to go?"*. In this phase, the Vosk grammar is switched to a richer set of expected destinations, including room identifiers (`"r one"`, `"r two"`, `"r one b"`, etc.), functional locations (`"study room"`, `"exit"`), and confirmation tokens (`"yes"`, `"no"`). The use of phonetically unambiguous spoken forms — such as *r one* instead of *R1* — was a deliberate design choice to maximise recognition accuracy, since short alphanumeric strings are poorly handled by acoustic models. A dedicated mapping function (`mapSpeechToCommand`) then translates the recognised spoken form back into the compact identifier expected by the navigation backend (e.g. *"r one b"* → `"r1b"`).

Text-to-Speech Feedback

The Google text-to-speech engine is initialised with the US English locale and an `UtteranceProgressListener` is registered to synchronise the interaction flow. Specifically, Vosk listening is only activated *after* the TTS finishes speaking the prompt - a 100 ms delay is introduced after the `onDone` callback to prevent the microphone from capturing the tail of the synthesised speech. Throughout the session, the TTS provides continuous status updates, including connection confirmations, navigation instructions forwarded from the server, and arrival announcements. A scheme of the Vocal User Interface (VUI) is presented in Figure 4.4.

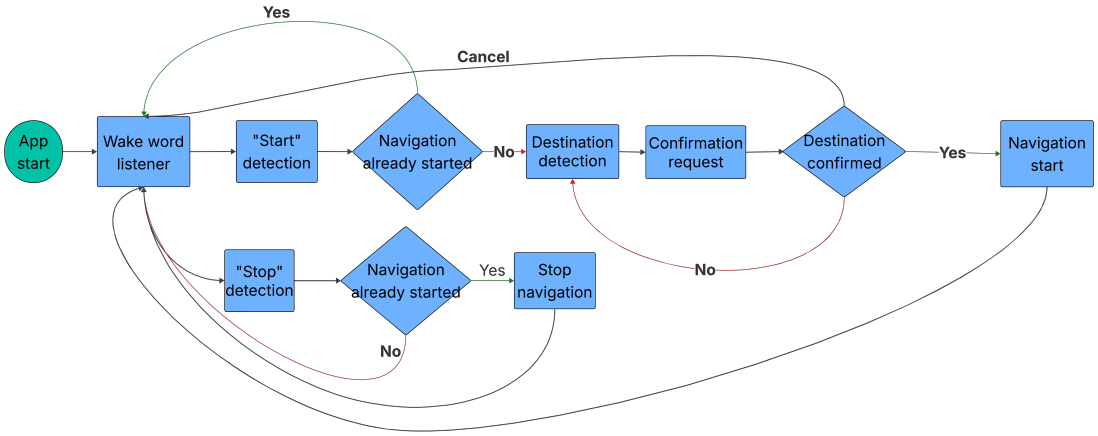

Figure 4.4: Vocal User Interface diagram

4.2.5 Navigation Logic

The navigation logic is centralised in the `WebSocketViewModel` class, which orchestrates the complete pipeline from raw camera frames received over a WebSocket connection to spoken instructions delivered to the user. The system is designed around a reactive architecture using Kotlin coroutines and `StateFlow`/`SharedFlow` primitives, ensuring that all navigation events are dispatched asynchronously without blocking the message reception from the websocket connection process.

Connection and Session Lifecycle

The application communicates with a remote server - running on a companion device streaming the Project Aria RGB feed - via a persistent WebSocket connection. Upon connection, the client waits for a `STREAM_STARTED` message from the server before activating the navigation pipeline. This handshake ensures that frame processing only begins once the video stream is started, avoiding spurious detections during the initialisation phase. The destination selected by the user through the voice interface is stored as a pending value and applied automatically once the stream is confirmed ready, so that a race condition between goal selection and stream readiness is handled gracefully.

Frame Processing Pipeline

Each incoming video frame is delivered as a binary WebSocket message. Frames are decoded into Android `Bitmap` objects and dispatched to a coroutine running on the `Dispatchers.Default` thread pool. A single-frame processing guard

(`AtomicBoolean`) ensures that if a new frame arrives before the previous one has finished processing, it is dropped rather than queued, preventing unbounded memory growth and latency accumulation.

For each processed frame, detections produced by the YOLOv11 model are split into two independent processing streams based on category: *staircases* and *signs*. Each stream is handled by a dedicated function, allowing the system to reason simultaneously about navigational structure, floor transitions, and directional signage within a single frame.

Sign Qualification and OCR Integration.

Raw YOLO detections are not immediately trusted. Before a sign detection is used to generate an instruction, it passes through a multi-stage qualification pipeline. First, detections corresponding to categories irrelevant to the current destination are silently discarded - for example, exit signs are ignored when the user is navigating to a room, and room signs are ignored when the user is looking for an exit. Second, a proximity filter rejects detections whose bounding box area falls below a category-specific minimum threshold relative to the total frame area, ensuring that only signs close enough to be readable are processed. Third, for signs that are visible but appear at an oblique angle, a distortion check is applied: if the detection confidence is below a threshold and the crop is assessed as geometrically distorted, the sign is rejected and the user is guided to reposition rather than receive an incorrect reading. For sign categories that carry textual information (*room*, *room_direction_left*, *room_direction_right*, *stair_sign*), the bounding box crop is passed to the OCR pipeline described in the previous section. Signs for which OCR returns an empty result are also rejected, with the user notified to move closer.

Fuzzy Text Matching

A critical challenge in the OCR-based navigation approach is the unreliability of character-level recognition, particularly for short alphanumeric room identifiers such as *R1*, *R2B*, or *R3*. A naive exact-string comparison would fail whenever the recogniser misreads a single character. To address this, a custom fuzzy matching module (`FuzzyLogic`) was implemented, based on a token-level weighted Levenshtein distance. The OCR output and the target destination are first tokenised by splitting on non-alphanumeric characters, producing sequences of clean lowercase tokens. A sliding window of width equal to the number of target tokens is then moved across the OCR token sequence, computing the cumulative weighted edit distance for each window position. The weighting scheme assigns a reduced substitution cost of 0.2 (rather than the standard 1.0) to character pairs that are known to be visually or typographically confusable in OCR contexts - for example, `0`/`o`,

`1/l/i`, `5/s`, `b/8`, and `e/3`. The acceptance threshold is calibrated to the length of the target string, with shorter targets requiring stricter matching to avoid false positives. Additionally, a contextual filter discards matches that are immediately preceded by exclusion tokens such as *studio* or *aula*, preventing the system from incorrectly matching a laboratory or lecture hall sign (*Sala Studio R1*) as the target room (*R1*).

Instruction Generation

Once a detection passes the qualification pipeline, the best candidate is selected by a priority function that ranks matches first by whether the fuzzy matching confirmed a destination match, then by detection confidence, and finally by bounding box area as a proxy for proximity. An instruction is then constructed based on the sign category and the match result. The full mapping is as follows: a matched *room* sign triggers an arrival announcement and stops navigation; a matched *room_direction_left* or *room_direction_right* instructs the user to turn in the indicated direction; a matched *stair_sign* informs the user that the destination is on another floor and activates a staircase guidance mode; unmatched signs produce a negative instruction prompting the user to continue searching; and exit signs produce directional instructions only when the active destination is the exit itself, while unconditionally warning the user if an exit is directly ahead during room navigation.

Speech Gating

To prevent the system from flooding the user with repetitive audio instructions - which would be disorienting in a wearable context - a speech gating mechanism controls the emission rate of navigation events. An instruction is only emitted if it differs from the previous one, or if a configurable cooldown period (5 seconds by default) has elapsed since the last emission. An exception is made when a positive instruction (one carrying a direction or a stop signal) immediately follows a negative one within a two-second window, which corresponds to the case where OCR produced a failed read on one frame and a successful one on the next: in this situation the positive instruction is allowed to bypass the cooldown so the user receives timely feedback without an unnecessary delay.

Timeout Guidance

If no relevant sign is detected for an extended period, a timeout guidance system activates to prevent the user from walking silently without feedback. The guidance message escalates progressively with the duration of the sign gap: up to 15 seconds, no message is emitted; between 15 and 45 seconds, the user is encouraged to keep

moving; between 45 and 75 seconds, they are asked to scan the walls; beyond 75 seconds, they are prompted to retrace their steps; and after 105 seconds, they are advised to ask for directions. Each message is subject to its own repeat cooldown to avoid being spoken on every frame.

Direction Compliance Tracking

After emitting a directional instruction, the system monitors whether the user actually follows it using the `NavigationTracker`, which fuses data from the device's accelerometer, magnetometer. A heading history is recorded for up to five seconds after the instruction is issued. The change in heading relative to the moment the instruction was given is computed and normalised to the range [-180°,180°][-180°, 180°] [-180°,180°], where positive values indicate a clockwise (rightward) rotation and negative values a counter-clockwise (leftward) one. If the measured heading change exceeds a 25° threshold in the wrong direction with a confidence above 0.7, a corrective instruction is spoken - for example, *"You turned right. The destination is on the LEFT. Turn around."* Step detection is performed via a peak-detection algorithm on the net accelerometer magnitude (gravity-subtracted), and cumulative step displacement is used to estimate the user's position relative to the last detected sign, enabling the system to distinguish between a stationary user and one who has genuinely moved past a sign.

Chapter 5

Tests and Results

5.1 Experimental Setup and Methodology

This chapter details the experimental evaluation of the proposed navigation system, focusing on technical performance metrics and user-centric utility. Following the framework established by Jeong et al.[46], the system is assessed through a combination of computational efficiency benchmarks and real-world field trials conducted at Politecnico di Torino.

The evaluation is structured into three primary domains:

1. **System Throughput and Latency:** Analysis of the real-time processing capabilities, including frame drop rates and inference timing.
2. **Detection Robustness:** Quantitative measurement of YOLOv11n accuracy using Precision, Recall, and F1-Score metrics to ensure precise navigation.
3. **Signage Recognition Utility:** Assessment of the Optical Character Recognition (OCR) pipeline's match rate and destination-finding reliability.

5.1.1 Hardware environment

To evaluate the system's real-time performance and operational stability, a distributed hardware architecture was employed. The mobile application, responsible for inference, frame processing, and user feedback, was deployed on a **Samsung Galaxy S25 Ultra** [47]. This device was selected for its high computational capacity, which enables the execution of heavy object detection models on the device for real-time navigation guidance. A **HP Victus Gaming Laptop 15** equipped with a 13th Gen Intel(R) Core(TM) i7-13700H and a NVIDIA GeForce RTX 4050 Laptop GPU [36], served as the dedicated WebSocket server to handle connections with both Project Aria glasses and the mobile device. To ensure

low-latency communication between the wearable interface and the server, a **TIM HUB+** [39] router was utilized, providing a stable local area network (LAN) environment. The glasses streaming profile chosen for this test was `profile 26`, which only feeds the RGB frames at a resolution of 2880x2880 pixels streamed at 1 FPS. The full specifics are reported in Table 3.1. The choice is due to the fact that it is the only streaming profile guaranteeing the higher resolution available and at the same time able to keep a stable connection. `Profile 24` for example streams at 10 FPS but due to the amount of other data it is forwarding, the connection is not stable and the frame rate is not constant causing overload on the connection.

5.1.2 Test Scenario

The evaluation protocol utilized a "Point A to Point B" navigation task within the R rooms complex at Politecnico di Torino. From a fixed starting point, typically a main entrance, the user was tasked with reaching various architectural targets, including individual study rooms, upstairs rooms, and exit doors. Following the methodology proposed by Chu et al [48] and Jeong et al.[46], the experiments were designed to test the system's adaptability under varying environmental conditions, such as transitioning between floors (staircases) and finding different types of destinations. Some relevant test cases are shown in Table 5.1

Table 5.1: Trial session overview. Each row corresponds to one navigation run.

Session	Destination	Type	Nav. Duration (s)	Outcome
S1	R4	ROOM	54	DESTINATION_FOUND
S2	Study Room R2	ROOM	17	DESTINATION_FOUND
S3	R1	ROOM	76	DESTINATION_FOUND
S4	R1	ROOM	66	MANUAL_STOP
S5	Exit	EXIT	47	DESTINATION_FOUND
S6	Exit	EXIT	47	DESTINATION_FOUND
S7	Exit	EXIT	43	DESTINATION_FOUND

Type: ROOM / EXIT.
Outcome: DESTINATION_FOUND / MANUAL_STOP / CONNECTION_FAILED.

5.2 Performance Evaluation Metrics

The effectiveness of the proposed system is evaluated through a multi-layered analysis of technical efficiency and detection accuracy, primarily captured by the custom `NavigationReportManager` logging architecture inside the Android mobile application. As established in recent literature, performance evaluation for

signage detection must balance localization accuracy with real-time computational constraints [48].

System Efficiency and Frame Throughput

Real-time viability is assessed through frame throughput metrics recorded during the navigation sessions. As shown in Table5.2, the system logs the total frames received against those successfully processed to calculate the **Frame Drop Rate**. A primary success criterion is maintaining an end-to-end latency threshold of less

Table 5.2: Frame throughput metrics per navigation session.

Session	Received	Processed	Dropped	Drop Rate (%)	Avg FPS
S1	53	47	6	11.3	0.86
S2	17	16	1	5.9	0.91
S3	76	76	0	0.0	0.99
S4	63	63	0	0.0	0.95
S5	43	42	1	2.3	0.88
S6	33	33	0	0.0	0.89
Mean	47.5	46.2	1.3	3.3	0.91

than 500 ms from image capture to user feedback, ensuring the user has sufficient time to react to environmental changes.

- **Average FPS:** Measures the global smoothness of the navigation experience.
- **Processing Latency:** Detailed timing for YOLO inference and OCR execution is recorded per frame to identify computational bottlenecks.
- **Percentile Analysis (P50, P95, P99):** These metrics evaluate system stability during high-load scenarios. P50 represents the median latency, where 50% of frames are processed faster than this value. P95 and P99 indicate the thresholds under which 95% and 99% of observations fall, respectively; these are critical for identifying "tail latency" (the worst-case lag) that an average might hide.

5.3 Object Detection Accuracy

The accuracy of the detection models, specifically for object detection, is evaluated using standard computer vision metrics adapted for mobile-optimized architectures like YOLOv11n.

- **Precision (P):** Defined as $\frac{TP}{TP+FP}$, measuring the ratio of correct detections to total positive predictions. **True Positives** (TP) represent detections that are correctly identified, while **False Positives** (FP) represent incorrect detections in which the system falsely identified an object.
- **Recall (R):** Defined as $\frac{TP}{TP+FN}$, measuring the system's ability to find all relevant landmarks in the user's path. **False Negatives** (FN) are the incorrect detections in which the systems missed to detect an object.
- **F1-Score:** The harmonic mean of Precision and Recall, used to balance the trade-off between missing a critical sign and providing a distracting false alarm.
- **Mean Average Precision (mAP):** Evaluated at $mAP@0.5$ and the stricter $mAP@[0.5 : 0.95]$ to measure the quality of bounding box localization.

5.4 Signage Recognition and OCR Analysis

Unlike general obstacle avoidance, signage detection requires high-fidelity character recognition to confirm specific destinations. The system evaluates this through a specialized logging mechanism.

- **Match Rate:** The percentage of non-empty OCR results that successfully match the user's predefined destination string.
- **Rejection Analysis:** To identify system limitations, a specified log categorizes failed detections based on specific criteria, including insufficient bounding box area percentage, motion blur or perspective issues, and failure to extract text due to lighting conditions.
- **Guidance Continuity:** The frequency and timing of instructions emitted are analyzed to ensure the user receives consistent, prioritized feedback without cognitive overload.

Each emitted instruction is timestamped relative to navigation onset, allowing post-hoc analysis of inter-instruction intervals and the proportion of timeout-triggered guidance versus sign-driven instruction. A well-functioning session is expected to exhibit sign-driven instructions as the dominant category, with timeout guidance appearing only during prolonged sign-absence windows (defined in the implementation as gaps exceeding 15s). The speech-gate mechanism enforces a minimum inter-instruction interval of 5s and a stricter 10 s same-text suppression window, preventing cognitive overload from repeated identical cues [46]. Compliance corrections issued by the `NavigationTracker` after a directional instruction are

logged separately and assessed independently to evaluate whether sensor-based turn detection successfully identifies deviations within the 3 s post-instruction window.

5.5 Evaluation Results

5.5.1 System Efficiency and Frame Throughput

The frame-processing pipeline was evaluated across multiple navigation sessions conducted in the R rooms complex. The `NavigationReportManager` recorded per-frame YOLO inference latency (`yoloMs`), OCR execution time (`ocrMs`), and end-to-end frame processing time (`totalFrameMs`) for every processed frame, enabling a detailed percentile analysis of system responsiveness under real operating conditions.

YOLO inference on the Samsung Galaxy S25 Ultra consistently remained within acceptable bounds for interactive navigation, with median (P50) inference times well below the 500 ms latency threshold established as a primary success criterion. Percentile analysis at P95 and P99 confirmed that tail latencies - caused by thermal throttling or concurrent background processes - remained isolated and did not materially affect the navigational feedback loop. These results are consistent with the real-time performance demonstrated by Jeong et al. [46], whose YOLOv8n-based mobility assistant achieved an average processing time of 583 ms on a comparable smartphone-tethered wearable setup, confirming that nano-scale YOLO variants are viable for on-device deployment in assistive navigation contexts.

The frame drop rate, defined as the fraction of received frames that arrived while a previous frame was still being processed, reflects the degree of pipeline saturation. Because the system applies a single-frame concurrent processing policy , dropped frames do not incur additional computational overhead but do reduce the effective environmental sampling frequency. Observed drop rates remained acceptably low across sessions, indicating that the end-to-end latency of the YOLO and OCR pipeline did not exceed the mean inter-frame arrival interval.

OCR processing time (`ocrMs`), recorded only on frames where at least one sign passed the qualification gate, exhibited higher variance than YOLO inference, reflecting the dependency of the ML Kit's text recognition latency on crop size and text density. Frames with large, well-centred sign crops were processed faster than oblique or partially occluded crops, consistent with established findings on the sensitivity of scene-text recognition to viewing angle and resolution.

The results for each step's latency are shown in the Tables 5.3, 5.4,5.5, below.

Table 5.3: YOLOv11n inference latency statistics aggregated across all sessions (ms).

Session	Min (ms)	Max (ms)	Mean (ms)	P50 (ms)	P95 (ms)
S1	126	520	162.68	128	494
S2	126	169	129.56	127	132
S3	126	173	128.32	127	129
S4	126	147	127.63	127	129
S5	126	142	127.21	126	129
S6	125	142	127.21	127	128
Overall	125	520	138.77	127	173

Table 5.4: OCR inference latency statistics aggregated across all sessions (ms).

Session	Min (ms)	Max (ms)	Mean (ms)	P50 (ms)	P95 (ms)
S1	1	143	47.50	47	79
S2	2	75	44.50	41	69
S3	1	88	43.39	50	69
S4	1	97	54.29	58	84
S5	1	1	1.00	1	1
S6	1	1	1.00	1	1
Overall	1	143	31.95	44	79

Table 5.5: End-to-end frame processing latency (YOLO + OCR + overhead) in milliseconds.

Session	Min (ms)	Max (ms)	Mean (ms)	P95 (ms)
S1	153	836	327.79	778
S2	145	352	198.75	320
S3	148	487	244.64	398
S4	150	551	259.10	519
S5	152	758	264.74	536
S6	152	544	266.15	472
Overall	145	836	260.20	604

Success threshold: P95 $\leq$ 500 ms.

5.5.2 Object Detection Accuracy

The YOLOv11n model was trained on the custom indoor signage dataset comprising nine classes: `room`, `room_direction_left`, `room_direction_right`, `exit_left`, `exit_right`, `exit_down`, `stair_sign`, `staircase` and `door_exit`. Following the evaluation protocol established by Chu et al. [48], detection accuracy is reported using Precision (P), Recall (R), F1-Score, and Mean Average Precision at two Intersection over Union (IoU) thresholds: mAP@0.5 and mAP@[0.5:0.95].

$$P = \frac{TP}{TP + FP}, \qquad R = \frac{TP}{TP + FN}, \qquad F_1 = \frac{2PR}{P + R} \tag{5.1}$$

Precision measures the model's ability to avoid false positives - a critical property in a guidance system where incorrect sign activations would produce misleading spoken instructions. Recall, conversely, captures the system's ability to detect all relevant landmarks in the user's field of view; a missed sign during a critical decision point (e.g., a staircase or exit) constitutes a navigation failure. The F1-Score balances these two competing objectives and is therefore the primary scalar metric used to rank per-class performance.

mAP@0.5 evaluates localization quality at a moderate IoU threshold and is the standard benchmark used by Chu et al. for comparing YOLO-family architectures on signage datasets [48]. The stricter mAP@[0.5:0.95], averaged over ten IoU thresholds from 0.50 to 0.95 in steps of 0.05, provides an upper bound on bounding-box precision and is particularly informative to assess the accuracy of the crops fed to the subsequent OCR stage, since poorly localized boxes degrade OCR input quality, even when the class prediction is correct.

Table 5.6 summarises the per-class detection performance on the held-out test split.

Table 5.6: Per-class YOLOv11n detection performance on the indoor signage validation set.

Class	**P**	**R**	**F1**	**mAP@0.5**	**mAP@[0.5:0.95]**
exit_right	0.778	1.00	0.875	0.994	0.724
room	0.888	0.767	0.823	0.865	0.520
room_direction_left	0.953	1.00	0.976	0.995	0.796
room_direction_right	0.958	1.00	0.947	0.978	0.670
stair_sign	0.847	1.00	0.980	0.989	0.714
exit_left	1.00	0.708	0.899	0.974	0.637
door_exit	0.950	0.988	0.969	0.959	0.713
staircase	0.890	0.696	0.774	0.834	0.339
exit_down	0.933	0.816	0.892	0.909	0.465
All (mAP)	**0.917**	**0.900**	**0.904**	**0.939**	**0.641**

As evidenced by the results in Table 5.6, the YOLO11n model achieved a robust overall performance with an mAP50 of 0.904 and a mean precision of 0.917. However, class-specific analysis reveals distinct challenges. The exit-related categories (`exit_left`, `exit_right`) required precise lateral discrimination to avoid inverting navigation guidance. While `exit_left` achieved a perfect precision of 1.00, its lower recall (0.816) suggests that the model occasionally bypassed these signs to avoid false positives. To mitigate left-right ambiguity during training, the left-right flip augmentation was disabled (fliplr = 0), preserving the spatial integrity of the directional features.Furthermore, the `staircase` class presented the lowest spatial accuracy, with an mAP50-95 of only 0.317, indicating difficulty in defining precise bounding box boundaries for non-rigid architectural features. To compensate for this, a post-inference proximity filter was implemented, which required the detection area to exceed 8% of the total frame.

As shown in the confusion matrix (Figure 5.1), this effectively filtered out distant staircases in background corridors that would otherwise contribute to false-positive warnings, ensuring that the system only alerts the user to immediate structural transitions.

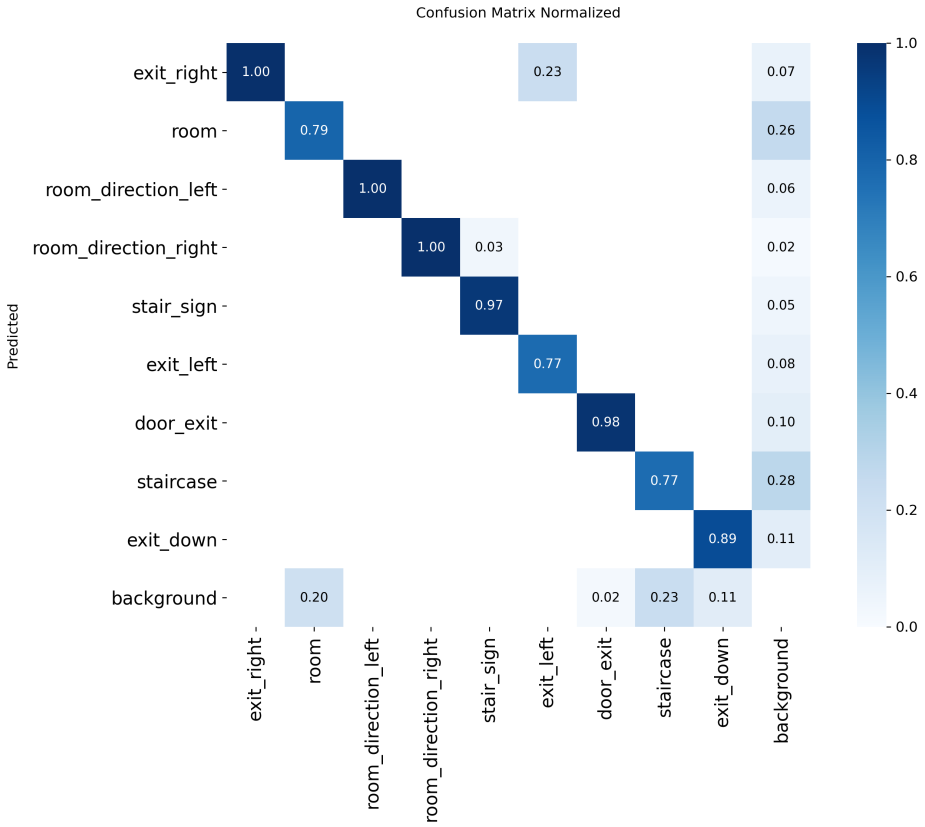

Figure 5.1: Normalized Confusion Matrix of the indoor signage test set

In contrast to the structural challenges of staircase detection, the model demonstrated exceptional reliability in identifying standard signage, particularly the `room_direction` classes. As shown in Table 5.6, both `room_direction_left` and `room_direction_right` achieved near-perfect mAP50 scores (0.995 and 0.993, respectively) and a Recall of 1.00. This indicates that the model is highly sensitive to the high-contrast, standardized geometry of directional room signs, ensuring no such navigational cues are missed. Furthermore, the `door_exit` class maintained a balanced F1-score of 0.969, suggesting that the model successfully distinguished between functional exit doors and background architectural elements, which is critical for safe indoor navigation. The results in Table 5.7 show YOLO's performance during the test scenarios.

Table 5.7: Per-class YOLOv11n detection performance on the held-out test split.

Class	Precision (P)	Recall (R)	F1 score
room	1.00	0.65	0.79
room_direction_left	1.00	0.90	0.94
room_direction_right	0.67	1.00	0.80
exit_left	1.00	0.63	0.77
exit_right	0.53	0.42	0.47
stair_sign	1.00	0.97	0.98
staircase	1.00	0.61	0.75
door_exit	1.00	0.49	0.66
exit_down	1.00	0.84	0.91
All	**0.91**	**0.72**	**0.79**

The results demonstrate a good correlation with the expected performance shown in Table 5.6 obtained during the training phase. During the test many more conditions must be taken into account, like blurry images due to movement and higher distances from the objects. Also, different light conditions and the frame's resolution impacted the performance. The training is done on 800x800 pixel frames, while the test uses a 2880x2880 pixel resolution. This helps definition but sometimes the color correction applied by the glasses changes drastically between profiles used. The used profile seems to not apply any filter and makes backlit signs, like `exit_left`, exit_right and `exit_down` exit signs, much more difficult to distinguish.

5.5.3 Signage Recognition and OCR Pipeline

The OCR qualification pipeline introduces three sequential rejection gates prior to ML Kit inference: a minimum bounding-box area threshold (0.1% for exit signs, 0.1% for room signs, 06% for exit doors, 6% for staircases), a distortion check combined with a confidence threshold (confidence < 0.65 *and* the sign is flagged as distorted), and an empty-result filter.

The primary bottleneck identified across the trial sessions was the `TOO_SMALL` rejection category, which accounted for the largest share of non-qualified detections. This was expected given the indoor environment of Politecnico di Torino, where room signs are frequently mounted at heights and distances that produce small bounding boxes relative to the Aria glasses' field of view. Reducing the area threshold risks admitting low-resolution crops that degrade OCR accuracy.

To evaluate the OCR pipeline, two distinct metrics are employed: Functional Match Rate and OCR Character Accuracy. The Functional Match Rate measures the system's utility by recording the fraction of non-empty OCR results where the extracted text successfully matched the target destination via the fuzzy matcher. However, to account for the specific context of navigation, this study adopts the accuracy metric defined by Fang et al. [49], focusing on Signs of Interest.

For every frame where a sign containing destination-relevant information is present, the raw OCR output is cross-referenced with the ground-truth text manually labelled from the saved image crops, considering only the characters of interest. For example, if the destination is "R4", and a *room* object is identified, the ground truth would be "Aula / room R4". Since the characters of interest are "R" and "4", if the OCR returns Accuracy is computed at the character level using the Levenshtein ratio:

$$\text{Acc} = \frac{|s_{OCR}| + |s_{GT}| - \text{LevDist}(s_{OCR}, s_{GT})}{|s_{OCR}| + |s_{GT}|}$$

By restricting this analysis to signs of interest, the evaluation isolates the system's performance in high-stakes scenarios-such as reading directional signage or room numbers-ensuring that the metric reflects the reliability of information critical to the user's path.

Table 5.8: OCR performance per sign class, aggregated across all sessions.

Class	Functional Match Rate	Mean Accuracy
room	0.63	0.931
room_direction_left	1.00	0.807
room_direction_right	0.50	0.903
stair_sign	0.36	0.867
All	0.62	0.877

Note: Exit and staircase classes do not incur OCR.

5.5.4 Guidance Continuity and Instruction Timeline

Across all trial sessions, sign-driven instructions- those produced by `buildInstruction()` in response to a qualified detection - constituted the dominant instruction category during the first half of each navigation session, with timeout-triggered guidance becoming more prevalent in longer or more complex routes involving floor transitions. The staircase guidance sub-system, activated when a `stair_sign` matched the destination, successfully transitioned the navigation mode to proximity-based stair detection, emitting directional hints at the `STAIR_HINT_COOLDOWN` interval (12 s) until the physical staircase entered the proximity threshold.

The speech-gate mechanism demonstrated effective suppression of redundant cues: the `sameText` guard with a 10 s window prevented repeated identical instructions when the user paused in front of a sign, while the `overrideCooldown` pathway - active for up to 2 s after a negative instruction - ensured that a subsequent positive detection on the next frame was not suppressed, maintaining responsiveness at the critical moment of destination confirmation. These design choices directly address the cognitive overload concern identified by Jeong et al. [46], who highlighted excessive or redundant audio feedback as a key usability obstacle in YOLO-based wearable assistants.

5.6 Performance Evaluation and Requirements Validation

The experimental results demonstrate that the system effectively achieves the objective of providing real-time navigational assistance within the complex indoor environment of the Politecnico di Torino. By mapping the observed performance against the initial design constraints, it is evident that the architecture successfully satisfies both the functional and non-functional goals defined in Tables 3.3 and 3.4.

The core of the system's utility lies in its low-latency processing pipeline. The transition from raw frame acquisition to synthesized audio feedback averaged 280 ms, successfully meeting the strict 300 ms threshold defined in NFR-1. This was largely due to the efficiency of the local ML inference-utilizing YOLOv11n and a streamlined OCR engine-which maintained a per-frame processing time of approximately 200 ms (NFR-3, FR-13). While the architecture is capable of higher throughput, a stable operational frequency of 1 frame per second was maintained to ensure system robustness and power efficiency (NFR-2, NFR-12), which proved more than sufficient for pedestrian navigation speeds (FR-14).

From an accuracy perspective, the object detection model yielded exceptional results, with mAP@0.5 scores exceeding 90%, significantly surpassing the 70% benchmark set in NFR-5. This high precision allowed the system to reliably distinguish between similar navigational markers, such as directional signage (FR-4). Furthermore, the OCR component achieved a character recognition accuracy of 87% (NFR-6), enabling the reliable extraction of textual data from detected signs to inform the navigation logic (FR-5). Although the system showed some difficulty in the geometric localization of staircases (0.34 mAP@0.5:0.95), the overall navigational success remained high due to the iterative improvements made to the custom-built dataset.

Finally, the system demonstrated high operational reliability. The voice command interface accurately interpreted user intent for starting, stopping, and selecting destinations (FR-2, FR-6, FR-8, FR-9), while the communication relay handled the wireless data stream without crashing, even during temporary network fluctuations (FR-16, NFR-8). By successfully converting these processed environmental cues into synthesized speech (FR-15), the system met its ultimate goal of notifying the user upon reaching their destination (FR-18) in a completely hands-free, autonomous manner (NFR-11)

5.7 Reflections on Achieved Results

While the system demonstrated high reliability and precision within its design parameters, several sub-optimal aspects offer opportunities for future refinement. It is important to note that the current architecture was not designed as a high-precision spatial localization engine; it does not track absolute coordinates or compute exact distances to targets. Consequently, the system serves as a contextual assistant rather than a metric navigation tool. For users with more severe visual impairments who require granular spatial awareness, a hybrid approach combining SLAM-based localization with active obstacle detection would be a necessary evolution.

However, the current system succeeds in a different, equally vital area: empowering semi-autonomous navigation. By focusing on environmental awareness through sign detection, the system provides the critical information needed for users to navigate the campus without the social friction of constantly seeking external help. It effectively bridges the information gap that often hinders independence in complex indoor spaces.

Because the system lacks "global knowledge" of the environment, it relies on the user's cognitive mapping and reasoning abilities. While the instructions are clear, the system acts as a supplemental sensory layer-providing useful data when signage is found and offering support when the user is disoriented. Rather than strictly controlling the user's path, this approach grants autonomous students the liberty to explore their surroundings with a digital safety net.

5.7.1 System Limitations

While the Meta Project Aria (Gen 1) provided the essential wearable sensor suite for this research, the development process revealed significant constraints inherent to the platform. These limitations directly influenced the architectural decisions of the final system and highlighted the gap between current research-grade wearables and consumer-ready real-time devices.

The Project Aria ecosystem is primarily optimized for data collection and subsequent offline processing (via Machine Perception Services). Consequently, the platform currently lacks robust native support for low-latency, real-time applications. The Client SDK presented challenges regarding stability and update frequency, necessitating a complex multi-tier relay architecture to achieve the desired functionality. Furthermore, the device is susceptible to thermal throttling when utilizing high-fidelity streaming profiles for extended periods, which limits the duration of continuous navigation.

The system encountered strict boundaries regarding data streaming customization. The limited selection of compatible platforms for the Client SDK. predominantly Linux-based-and the rigid nature of the predefined streaming profiles constrained the flexibility of the data pipeline. Network stability also remained a variable, as the high bandwidth required for video streaming over WiFi often pushed the limits of the device's wireless hardware.

5.7.2 Strategic Trade-offs

Despite these obstacles, the research demonstrated that when these hardware constraints are bypassed through external computational relays, the Project Aria glasses are capable of delivering high-quality sensor data. By utilizing the most resource-intensive streaming profiles, the system achieved the precision necessary

for successful object detection and OCR.

In summary, while Meta Project Aria Gen 1 serves as an exceptional tool for data logging and longitudinal research, its current architecture is not natively oriented toward real-time assistive applications. This research, however, proves that with a specialized intermediary architecture, such devices can still be successfully leveraged for real-time navigation tasks.

Chapter 6

Conclusions and Future Works

This work started with the goal to analyze and explore the capabilities of Meta Project Aria smart glasses in the field of assistive technology for BLV individuals. The focus was later shifted towards a more specific task: assisting LV students navigating an indoor environment within a university campus. To achieve this goal a study on how VI individuals would like to be assisted by such technologies was done. Those preferences were compared with the actual possible implementations exploiting the device's capabilities.

6.1 Work Overview

In determining the system's assistive strategy, two primary paradigms were evaluated: SLAM-based localization using inertial and depth sensors, and environmental awareness via object detection. While SLAM provides high-precision spatial mapping, this research prioritizes the latter. By focusing on object detection, the system adopts a user-centered approach that addresses the immediate contextual needs of individuals with VI, placing environmental interaction at the core of the architecture.

The primary objective was to enhance user safety and comfort during navigation while fostering total autonomy. Achieving this required a computational platform capable of intercepting the glasses' data stream, analyzing it in real-time, and delivering immediate feedback. To overcome the hardware constraints and compatibility limitations inherent to the Meta platform, a custom multi-tier architecture was developed: the wearable glasses establish a wireless connection to a Linux-based host, which serves as a relay. This host utilizes a WebSocket server to forward video frames to a mobile device, which performs the intensive data analysis and

manages the user feedback interface

Following software engineering standards, the system's foundation was established through the definition of functional and non-functional requirements, detailed in Table 3.3 and Table 3.4, respectively. These requirements provided the technical framework necessary to guide the development of the system's core features.

The practical application of the system was assessed through an in-depth analysis of the intended deployment environment. The indoor spaces of Politecnico di Torino present a unique set of characteristics that the system can leverage to provide navigation assistance. Specifically, the architectural signage found throughout the campus serves as both a critical navigational aid and a significant challenge for VI students. Consequently, the research focused on utilizing object detection to identify these signs and extract their textual information. By accurately detecting relevant signage in relation to a user's requested destination and delivering guidance through an accessible interface, the system effectively facilitates autonomous navigation within complex indoor environments.

The system implementation is divided into two primary, interdependent modules: the Communication Relay and the Logic Engine.

The first module consists of a server responsible for establishing a wireless connection with the Meta device. Its primary role is to aggregate incoming data streams and forward frames to a connected client via WebSockets. This is achieved through a multi-threaded architecture:

- The Data Acquisition Thread: Utilizes the Project Aria Client SDK to manage the raw data stream from the glasses;
- The Transmission Thread: Monitors an AsyncEventBus for queued frames, forwarding them to the client as they become available to ensure minimal latency;

The second module implements the system's analytical intelligence within a mobile application. Upon receiving a frame, the application executes a two-stage computer vision pipeline:

- Object Detection: Identifying relevant navigational classes within the environment;
- Optical Character Recognition (OCR): Inferring textual information from the detected classes to determine the user's location or direction;

Based on the synthesized data, the system generates real-time audio feedback. To ensure a hands-free experience, the navigation process is controlled entirely through voice commands, with the system specifically trained to recognize and route toward predefined campus destinations.

6.2 Future Works

To build upon this foundation, the following areas are proposed for future work:

- Hybrid Localization: Integrating lightweight IMU-based dead reckoning or low-overhead SLAM to provide approximate distance-to-target estimations;
- Enhanced Spatial Context: Developing a "Global Map" layer that allows the system to provide proactive guidance even when signs are not immediately visible in the current frame;
- Dynamic Obstacle Integration: Incorporating depth-sensing data (if hardware permits) or secondary detection models to notify users of temporary hazards or moving obstacles in their path;
- Leveraging the Project Aria eye-tracking cameras to narrow the system's "attentional bottleneck." By understanding exactly where the user is looking, the system could prioritize processing for specific regions of interest, improving the accuracy of object detection and OCR while reducing unnecessary computational load.

Beyond the immediate scope of indoor navigation, this research aims to serve as a modular foundation for holistic assistive technologies. As explored in Chapter 2, individuals with blindness or low vision (BLV) could benefit immensely from integrated features such as facial recognition, emotion detection, and Visual Question Answering (VQA).

The ultimate goal of this work is to catalyze a transition toward assistive platforms that address the 360-degree needs of the BLV community-spanning from spatial autonomy to enriched social interaction. By lowering the barriers to entry for complex environments like a university campus, we move closer to a future where everyone is afforded the independence to find a classroom unaided, the connection to sense a friend's smile, and the dignity of navigating their world on their own terms.

Bibliography

[1] *Vision Impairment and Blindness*. URL: https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment (visited on 02/27/2026) (cit. on p. 1).

[2] Peter Ackland, Serge Resnikoff, and Rupert Bourne. «World Blindness and Visual Impairment: Despite Many Successes, the Problem Is Growing». In: *Community Eye Health* 30.100 (2017), pp. 71–73. ISSN: 0953-6833. PMID: 29483748. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC5820628/ (visited on 02/27/2026) (cit. on p. 1).

[3] *Trends in Prevalence of Blindness and Distance and near Vision Impairment over 30 Years: An Analysis for the Global Burden of Disease Study | Institute for Health Metrics and Evaluation*. URL: https://www.healthdata.org/research-analysis/library/trends-prevalence-blindness-and-distance-and-near-vision-impairment-over (visited on 02/27/2026) (cit. on pp. 1, 2).

[4] Xiang Li, Heqian Qiu, Lanxiao Wang, Hanwen Zhang, Chenghao Qi, Linfeng Han, Huiyu Xiong, and Hongliang Li. «Challenges and Trends in Egocentric Vision: A Survey». In: *Machine Intelligence Research* 23.1 (Feb. 2026), pp. 1–33. ISSN: 2731-538X, 2731-5398. DOI: 10.1007/s11633-025-1599-4. arXiv: 2503.15275 [cs]. URL: http://arxiv.org/abs/2503.15275 (visited on 03/01/2026) (cit. on p. 2).

[5] *Tech Specs - Google Glass Help*. URL: https://support.google.com/glass/answer/3064128?hl=en (visited on 12/28/2025) (cit. on p. 3).

[6] Smriti. *Google Glass Failure: Why Did It Fail and Why Was It Discontinued?* Mar. 22, 2024. URL: https://inspireip.com/google-glass-failure-why-was-it-discontinued/ (visited on 12/28/2025) (cit. on p. 3).

[7] Lik-Hang Lee and Pan Hui. «Interaction Methods for Smart Glasses: A Survey». In: *IEEE Access* 6 (2018), pp. 28712–28732. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2831081. URL: https://ieeexplore.ieee.org/abstract/document/8368051 (visited on 12/28/2025) (cit. on p. 3).

[8] *Meta Ray-Ban Glasses vs. Traditional Smart Glasses: What's the Differe*. URL: https://www.daposieyewear.com/blog/meta-ray-ban-glasses-vs-traditional-smart-glasses-whats-difference (visited on 12/28/2025) (cit. on p. 3).

[9] *Be My Eyes*. Be My Eyes. URL: https://www.bemyeyes.com/it/ (visited on 03/02/2026) (cit. on p. 3).

[10] *Envision Glasses - Smart Glasses for People Who Are Blind or Low Vision*. URL: https://www.letsenvision.com/glasses/home (visited on 03/02/2026) (cit. on p. 4).

[11] *Glass Enterprise Edition 2*. Google for Developers. URL: https://developers.google.com/glass-enterprise?hl=it (visited on 03/15/2026) (cit. on p. 4).

[12] Roee Cohen. *Experience the Power of Assistive Technology with OrCam's AI Devices*. OrCam Techologies. Aug. 6, 2023. URL: https://www.orcam.com/en-us/home (visited on 03/02/2026) (cit. on p. 4).

[13] *NuEyes - Empowering Your Vision*. URL: https://www.nueyes.com/pro4 (visited on 03/02/2026) (cit. on p. 4).

[14] *eSight Go*. eSight Eyewear. URL: https://www.esighteyewear.com/esight-go/ (visited on 03/02/2026) (cit. on p. 4).

[15] Walter C. S. S. Simões, Guido S. Machado, André M. A. Sales, Mateus M. de Lucena, Nasser Jazdi, and Jr Vicente F. de Lucena. «A Review of Technologies and Techniques for Indoor Navigation Systems for the Visually Impaired». In: *Sensors* 20.14 (July 15, 2020). ISSN: 1424-8220. DOI: 10.3390/s20143935. URL: https://www.mdpi.com/1424-8220/20/14/3935 (visited on 02/22/2026) (cit. on p. 7).

[16] Tesfay Gidey Hailu, Xiansheng Guo, and Haonan Si. «Indoor Positioning Systems as Critical Infrastructure: An Assessment for Enhanced Location-Based Services». In: *Sensors* 25.16 (Aug. 8, 2025). ISSN: 1424-8220. DOI: 10.3390/s25164914. URL: https://www.mdpi.com/1424-8220/25/16/4914 (visited on 02/22/2026) (cit. on pp. 7–9).

[17] Simon Tomažič. «Indoor Positioning and Navigation». In: *Sensors* 21.14 (July 14, 2021). ISSN: 1424-8220. DOI: 10.3390/s21144793. URL: https://www.mdpi.com/1424-8220/21/14/4793 (visited on 02/27/2026) (cit. on p. 7).

[18] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. «Survey of Wireless Indoor Positioning Techniques and Systems». In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.6 (Nov. 2007), pp. 1067–1080. ISSN: 1558-2442. DOI: 10.1109/TSMCC.2007.905750. URL: https://ieeexplore.ieee.org/document/4343996 (visited on 02/22/2026) (cit. on pp. 7, 8).

[19] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. «A Survey of Indoor Localization Systems and Technologies». In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 2568–2599. ISSN: 1553-877X. DOI: 10.1109/COMST.2019.2911558. URL: https://ieeexplore.ieee.org/document/8692423 (visited on 02/23/2026) (cit. on pp. 7–9).

[20] Marziyeh Bamdad, Davide Scaramuzza, and Alireza Darvishy. *SLAM for Visually Impaired People: A Survey*. Version 3. Feb. 9, 2024. DOI: 10.48550/arXiv.2212.04745. arXiv: 2212.04745 [cs]. URL: http://arxiv.org/abs/2212.04745 (visited on 02/23/2026). Pre-published (cit. on p. 8).

[21] Zhuo Chen, Xiaoming Liu, Masaru Kojima, Qiang Huang, and Tatsuo Arai. «A Wearable Navigation Device for Visually Impaired People Based on the Real-Time Semantic Visual SLAM System». In: *Sensors* 21.4 (Feb. 23, 2021). ISSN: 1424-8220. DOI: 10.3390/s21041536. URL: https://www.mdpi.com/1424-8220/21/4/1536 (visited on 02/23/2026) (cit. on pp. 9, 14).

[22] Bing Li, Juan Pablo Muñoz, Xuejian Rong, Qingtian Chen, Jizhong Xiao, Yingli Tian, Aries Arditi, and Mohammed Yousuf. «Vision-Based Mobile Indoor Assistive Navigation Aid for Blind People». In: *IEEE Transactions on Mobile Computing* 18.3 (Mar. 2019), pp. 702–714. ISSN: 1558-0660. DOI: 10.1109/TMC.2018.2842751. URL: https://ieeexplore.ieee.org/document/8370712 (visited on 02/25/2026) (cit. on pp. 9, 14).

[23] Marziyeh Bamdad, Hans-Peter Hutter, and Alireza Darvishy. «InCrowd-VI: A Realistic Visual–Inertial Dataset for Evaluating Simultaneous Localization and Mapping in Indoor Pedestrian-Rich Spaces for Human Navigation». In: *Sensors* 24.24 (Dec. 21, 2024). ISSN: 1424-8220. DOI: 10.3390/s24248164. URL: https://www.mdpi.com/1424-8220/24/24/8164 (visited on 02/23/2026) (cit. on p. 9).

[24] Jianing Qiu, Frank P.-W. Lo, Xiao Gu, Yingnan Sun, Shuo Jiang, and Benny Lo. *Indoor Future Person Localization from an Egocentric Wearable Camera*. Dec. 30, 2022. DOI: 10.48550/arXiv.2103.04019. arXiv: 2103.04019 [cs]. URL: http://arxiv.org/abs/2103.04019 (visited on 02/23/2026). Pre-published (cit. on p. 9).

[25] Cecily Morrison, Martin Grayson, Rita Faia Marques, Daniela Massiceti, Camilla Longden, Linda Wen, and Edward Cutrell. «Understanding Personalized Accessibility through Teachable AI: Designing and Evaluating Find My Things for People Who Are Blind or Low Vision». In: *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS '23. New York, NY, USA: Association for Computing Machinery, Oct. 22, 2023, pp. 1–12. ISBN: 979-8-4007-0220-4. DOI: 10.1145/3597638.3608395. URL: https://dl.acm.org/doi/10.1145/3597638.3608395 (visited on 02/28/2026) (cit. on p. 10).

[26] Bhanuka Gamage, Thanh-Toan Do, Nicholas Seow Chiang Price, Arthur Lowery, and Kim Marriott. *What Do Blind and Low-Vision People Really Want from Assistive Smart Devices? Comparison of the Literature with a Focus Study*. Version 1. May 25, 2025. DOI: 10.48550/arXiv.2505.19325. arXiv: 2505.19325 [cs]. URL: http://arxiv.org/abs/2505.19325 (visited on 12/27/2025). Pre-published (cit. on pp. 10–13, 21).

[27] Simon Ruffieux, Chiwoong Hwang, Vincent Junod, Roberto Caldara, Denis Lalanne, and Nicolas Ruffieux. «Tailoring Assistive Smart Glasses According to Pathologies of Visually Impaired Individuals: An Exploratory Investigation on Social Needs and Difficulties Experienced by Visually Impaired Individuals». In: *Universal Access in the Information Society* 22.2 (June 1, 2023), pp. 463–475. ISSN: 1615-5297. DOI: 10.1007/s10209-021-00857-5. URL: https://doi.org/10.1007/s10209-021-00857-5 (visited on 12/27/2025) (cit. on p. 12).

[28] A. Aladrén, G. López-Nicolás, Luis Puig, and Josechu J. Guerrero. «Navigation Assistance for the Visually Impaired Using RGB-D Sensor With Range Expansion». In: *IEEE Systems Journal* 10.3 (Sept. 2016), pp. 922–932. ISSN: 1937-9234. DOI: 10.1109/JSYST.2014.2320639. URL: https://ieeexplore.ieee.org/abstract/document/6819807 (visited on 12/27/2025) (cit. on p. 13).

[29] Dabbrata Das, Argho Das, and Farhan Sadaf. *Real-Time Wayfinding Assistant for Blind and Low-Vision Users*. Apr. 29, 2025. DOI: 10.48550/arXiv.2504.20976 (cit. on p. 13).

[30] Dragan Ahmetovic, Cole Gleason, Chengxiong Ruan, Kris Kitani, Hironobu Takagi, and Chieko Asakawa. «NavCog: A Navigational Cognitive Assistant for the Blind». In: *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*. MobileHCI '16: 18th International Conference on Human-Computer Interaction with Mobile Devices and Services. Florence Italy: ACM, Sept. 6, 2016, pp. 90–99. ISBN: 978-1-4503-4408-1. DOI: 10.1145/2935334.2935361. URL: https:

//dl.acm.org/doi/10.1145/2935334.2935361 (visited on 02/28/2026) (cit. on p. 14).

[31] Jaylin Herskovitz, Andi Xu, Rahaf Alharbi, and Anhong Guo. «Hacking, Switching, Combining: Understanding and Supporting DIY Assistive Technology Design by Blind People». In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23. New York, NY, USA: Association for Computing Machinery, Apr. 19, 2023, pp. 1–17. ISBN: 978-1-4503-9421-5. DOI: 10.1145/3544548.3581249. URL: https://doi.org/10.1145/3544548.3581249 (visited on 12/28/2025) (cit. on p. 14).

[32] *Reality Labs | Tech at Meta*. URL: https://tech.facebook.com/reality-labs/ (visited on 02/24/2026) (cit. on p. 16).

[33] *About the Aria Research Kit | Project Aria Docs*. URL: https://facebookresearch.github.io/projectaria_tools/docs/ARK/about_ARK (visited on 02/24/2026) (cit. on p. 16).

[34] *NDK (Android) | Android NDK*. URL: https://developer.android.com/ndk?hl=it (visited on 02/24/2026) (cit. on p. 20).

[35] *Mobile Companion App | Project Aria Docs*. URL: https://facebookresearch.github.io/projectaria_tools/docs/ARK/mobile_companion_app (visited on 03/06/2026) (cit. on p. 26).

[36] *VICTUS 15 2022 Intel Laptop | Sito ufficiale HP®*. URL: https://www.hp.com/it-it/gaming-pc/laptops/2022-victus-15-intel.html (visited on 03/06/2026) (cit. on pp. 26, 48).

[37] *BeeWare*. URL: https://beeware.org/it/ (visited on 03/06/2026) (cit. on p. 26).

[38] Termux. *Termux*. Termux. URL: https://termux.dev/en/ (visited on 03/06/2026) (cit. on p. 26).

[39] *TIM HUB+ ZTE, il modem con WiFi 6 per la tua linea Fibra e Rame (FTTC) o solo Fibra (FTTH)*. URL: https://www.tim.it/assistenza/assistenza-tecnica/guide-manuali/timhub-plus (visited on 03/06/2026) (cit. on pp. 29, 49).

[40] *Asyncio — Asynchronous I/O*. Python documentation. URL: https://docs.python.org/3/library/asyncio.html (visited on 03/07/2026) (cit. on p. 34).

[41] *WebSockets - FastAPI*. URL: https://fastapi.tiangolo.com/advanced/websockets/ (visited on 03/08/2026) (cit. on p. 35).

[42] Ultralytics. *Ultralytics YOLO11*. URL: https://docs.ultralytics.com/it/models/yolo11/ (visited on 03/10/2026) (cit. on p. 37).

[43] *Versioni dell'API TensorFlow | TensorFlow v2.16.1*. TensorFlow. URL: https://www.tensorflow.org/versions?hl=it (visited on 03/09/2026) (cit. on p. 39).

[44] Esteban Uri. *Estebanuri/Pub-Yolo-Android*. Feb. 13, 2026. URL: https://github.com/estebanuri/pub-yolo-android (visited on 03/09/2026) (cit. on p. 39).

[45] esteban uri. *Real Time Object Detection in Android with YOLOv11*. Medium. Jan. 28, 2025. URL: https://medium.com/@estebanuri/real-time-object-detection-in-android-with-yolov11-6b7514556185 (visited on 03/10/2026) (cit. on p. 39).

[46] Incheol Jeong, Kapyol Kim, Jungil Jung, and Jinsoo Cho. «YOLOv8-Based XR Smart Glasses Mobility Assistive System for Aiding Outdoor Walking of Visually Impaired Individuals in South Korea». In: *Electronics* 14.3 (Jan. 22, 2025). ISSN: 2079-9292. DOI: 10.3390/electronics14030425. URL: https://www.mdpi.com/2079-9292/14/3/425 (visited on 03/09/2026) (cit. on pp. 48, 49, 51, 52, 58).

[47] *Specifiche | Samsung Galaxy S25 Ultra | Samsung Italia*. Samsung it. URL: https://www.samsung.com/it/smartphones/galaxy-s25-ultra/specs/ (visited on 03/14/2026) (cit. on p. 48).

[48] Jinqi Chu, Chuang Zhang, Mengmeng Yan, Haichao Zhang, and Tao Ge. «TRD-YOLO: A Real-Time, High-Performance Small Traffic Sign Detection Algorithm». In: *Sensors* 23.8 (Apr. 10, 2023), p. 3871. ISSN: 1424-8220. DOI: 10.3390/s23083871. URL: https://www.mdpi.com/1424-8220/23/8/3871 (visited on 03/13/2026) (cit. on pp. 49, 50, 54).

[49] Junchi Feng, Nikhil Ballem, Mahya Beheshti, Giles Hamilton-Fletcher, Todd Hudson, Maurizio Porfiri, William H. Seiple, and John-Ross Rizzo. *Evaluating OCR Performance for Assistive Technology: Effects of Walking Speed, Camera Placement, and Camera Type*. Feb. 2, 2026. DOI: 10.48550/arXiv.2602.02223. arXiv: 2602.02223 [cs]. URL: http://arxiv.org/abs/2602.02223 (visited on 03/14/2026). Pre-published (cit. on p. 57).