DESIGN AND IMPLEMENT A LOW-COST, WEARABLE VISUAL AID FOR VISUALLY IMPAIRED INDIVIDUALS

BY

ALIU EMMANUEL OMONUA	**ENG2002212**
GHOMREN VICTOR OGHENETEGA	**ENG2002244**
OMOGHOMION BENEDICT OSAMUDIAMEN	**ENG2002380**
OBOMERO DENNIS	**ENG2006269**
IYOMANA ISOKEN JESSICA	**ENG2009608**

A PROJECT SUBMITTED TO THE DEPARTMENT OF ELECTRICAL/ELECTRONIC ENGINEERING, FACULTY OF ENGINEERING

UNIVERSITY OF BENIN, BENIN CITY

OCTOBER 2025

CERTIFICATION

This is to certify that this project work was carried out by **Aliu Emmanuel Omonua** with matriculation number **ENG2002212**, **Ghomren Victor Oghenetega** with matriculation number **ENG2002244**, **Omoghomion Benedict Osamudiamen** with matriculation number **ENG2002380**, **Obomero Dennis** with matriculation number **ENG2006269**, and **Iyomana Isoken Jessica** with matriculation number **ENG2009608** in partial fulfilment of the requirement for the award of Bachelor of Electrical/Electronic Engineering in the Department of Electrical/Electronic Engineering, Faculty of Engineering, University of Benin, Benin City, Edo State.

…… ……

ENGR. ABDUL-RAHMAN DAUDA DATE

PROJECT SUPERVISOR

…… ……

ENGR. DR. S.O. OMOROGIUWA DATE

HEAD OF DEPARTMENT

DEDICATION

This work is dedicated to God Almighty, and to the Ghomren family, the Aliu family, the Iyomana family, the Obomero family and the Omoghomion family, for their unending support, and love, throughout our study in the University of Benin.

ACKNOWLEDGMENT

We thank God Almighty for His love, grace, mercies and protection upon us, throughout our stay in the University of Benin. We also thank the families of Ghomren, Aliu, Iyonmana, Obomero and Omoghomion, who supported us financially, morally and spiritually, throughout our journey in the University of Benin. Futhermore, we would like to acknowledge our peers and colleagues for their collaboration and camaraderie, which made this journey both enlightening and enjoyable. We wish to thank Engr. Abdul-Rahman DAUDA, our project supervisor, for his time and effort in reviewing this project and for instilling in us the discipline to always aim for the best. Lastly, we would like to appreciate the HOD of Electrical/Electronic Engineering, Engr. Dr. S.O. Omorogiuwa for the opportunity to embark on this project.

ABSTRACT

Visual impairment significantly restricts independent navigation and environmental awareness, affecting over 2.2 billion people globally. This project aims to design and implement a low-cost, wearable visual aid that enhances navigation safety for visually impaired individuals through real-time object detection and intelligent audio feedback. The system addresses critical gaps in affordability, offline capability, and intelligent information prioritization by providing a practical alternative to expensive commercial assistive devices.

The system integrates a Microsoft LifeCam HD-3000 camera module with a Raspberry Pi 5 single-board computer running the YOLO-Fastest object detection model via the NCNN inference framework. It encompasses a four-stage pre-processing pipeline including frame resizing, color conversion, normalization, and data layout transformation. A spatial processing and prioritization algorithm evaluates detected objects based on monocular distance estimation, directional zoning, trajectory analysis, and hazard level classification. The system generates context-aware audio feedback through the pyttsx3 text-to-speech engine, employing priority-based speech rate modulation (150-180 wpm) and an interrupt mechanism for critical warnings.

Comprehensive testing was conducted across five detection scenarios, range validation tests, object prioritization assessments, and environmental stress tests under varying lighting conditions. Experimental results demonstrated consistent real-time performance with average response times of 55.68–65.18 ms per frame (15–18 FPS) and confidence scores ranging from 0.68 to 0.85. The system achieved effective detection ranges of 8 m for pedestrians, 33.21 m for static obstacles, and 112.9 m for vehicles. Object prioritization accuracy reached 100% in simple scenarios but decreased to 73.8% in highly crowded environments with seven simultaneous objects. Environmental testing revealed detection rates of 95–100% under daylight conditions, degrading

to 30% in near-dark environments. The system processed over 15,000 frames without software crashes, confirming operational stability. The prototype, assembled from off-the-shelf components at a fraction of commercial costs, successfully demonstrates the feasibility of an affordable, edge-deployed wearable visual aid suitable for daytime navigation assistance.

TABLE OF CONTENTS

Contents

LIST OF ABBREVATIONS

BLE	Bluetooth Low Energy
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CSI	Camera Serial Interface
DC	Direct Current
DSR	Design Science Research
ETA	Electronic Travel Aids
FCC	Federal Communications Commission
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
GPS	Global Positioning System
HAT	Hardware Attached on Top
HDMI	High-Definition Multimedia Interface
IMU	Inertial Measurement Unit
LED	Light Emitting Diode
LiDAR	Light Detection and Ranging
LPDDR4X	Low Power Double Data Rate 4x
MCT	Mixed Conditions Test

MTBF	Mean Time Between Failure
OTT	Object Type Test
PCB	Printed Circuit Board
PoE	Power over Ethernet
PT	Proximity Test
SLAM	Simultaneous Localization and Mapping
TTS	Text-To-Speech
USB	Universal Serial Bus
VT	Velocity Test
WHO	World Health Organization
YOLO	You-Only-Look-Once

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE
INTRODUCTION

1.1 Background of the project

Vision is a primary sensory modality that underpins human interaction with the physical environment, enabling spatial awareness, obstacle avoidance, and object recognition. For individuals with visual impairment, routine activities such as independent navigation, reading, and social interaction become significantly more challenging, often compromising safety and autonomy (*Gupta and Kumar, 2022*). According to the World Health Organization (WHO), visual impairment is defined as a reduction in visual acuity or visual field that cannot be fully corrected with standard optical, medical, or surgical interventions (*World Health Organization, 2019*). Global epidemiological data indicate that at least 2.2 billion people worldwide experience near or distance visual impairment, with approximately 1 billion cases attributable to preventable or treatable conditions (*Bourne et al., 2021*). Current estimates place the global prevalence of blindness at 43.3 million, while 295 million individuals suffer from moderate to severe visual impairment. Driven by demographic aging, population growth, and changing lifestyle factors, WHO projects these figures to rise to 61 million blind and 474 million visually impaired individuals by 2050 (*World Health Organization, 2019; Bourne et al., 2021*).

The consequences of visual impairment extend across multiple life domains. Educational attainment is frequently hindered when learning materials lack accessible formatting, while employment opportunities remain constrained in workplaces that do not integrate inclusive design or assistive technologies (*Gupta and Kumar, 2022*). Mobility and independence are particularly affected, as unfamiliar environments increase the risk of falls, collisions, and disorientation. Social participation may also decline due to difficulties in recognizing faces, interpreting non-verbal cues,

or navigating public spaces, which can contribute to psychological distress, anxiety, or social isolation (*Al-Haddad et al., 2022*). Collectively, these challenges underscore the urgent need for reliable, accessible, and user-centered assistive solutions.

Recent advancements in embedded computing, computer vision, and lightweight artificial intelligence have enabled the development of wearable assistive devices that provide real-time environmental feedback. Affordable single-board computers (e.g., Raspberry Pi series) and optimized object detection frameworks (e.g., YOLO variants, MobileNet-SSD) now allow edge-deployed systems to process visual data with low latency and minimal power consumption (*Bochkovskiy, Wang and Liao, 2023; Raspberry Pi Foundation, 2023*). Unlike traditional electronic travel aids (ETAs) that rely on ultrasonic or infrared sensors, camera-based wearable systems can deliver semantically rich information, such as object classification, spatial positioning, and hazard prioritization (*Brown et al., 2022*). However, many existing commercial solutions remain cost-prohibitive, overly complex, or designed without sufficient consideration for cognitive load and social acceptance (*Chen and Williams, 2023; Chen et al., 2023*).

This project addresses these gaps by designing and implementing a low-cost wearable visual aid that integrates a camera-based sensing module, real-time object detection, and an intelligent audio feedback system. By prioritizing affordability, usability, and contextual awareness, the prototype aims to enhance environmental perception and navigation safety for visually impaired users in both indoor and outdoor settings.

1.2 Problem Statement

Despite the availability of assistive technologies for the visually impaired, several critical limitations hinder widespread adoption and practical effectiveness:

- **Functional Independence Constraints:** Visual impairment severely restricts the ability to perform daily tasks such as reading, navigating unfamiliar routes, and managing personal care, thereby reducing autonomy and efficiency in everyday life (*Gupta and Kumar, 2022; Al-Haddad et al., 2022*).
- **Economic and Accessibility Barriers:** Commercial electronic travel aids and smart vision devices (e.g., OrCam, Envision Glasses) often exceed the financial capacity of low- and middle-income users, who constitute the majority of the visually impaired population globally (*Chen and Williams, 2023; Global Assistive Technology Partnership, 2024*).
- Suboptimal Information Processing and Feedback: Many existing ETAs transmit raw or unfiltered sensor data, leading to auditory clutter and increased cognitive load. The lack of intelligent prioritization mechanisms reduces usability and situational awareness (*Brown et al., 2022; Chen et al., 2023*).
- Design and Social Acceptance Issues: Bulky or conspicuously medicalized assistive devices can induce stigma, discouraging consistent use and conflicting with users' aspirations for normalized daily integration (*Rogers, 2023*).

These challenges highlight the need for a cost-effective, cognitively optimized, and socially acceptable wearable visual aid that delivers contextual, prioritized environmental feedback in real time.

1.3 Aims and Objectives

Aim:

To design and implement a low-cost, wearable visual aid that enhances environmental awareness and navigation safety for visually impaired individuals through real-time object detection and intelligent audio feedback.

Objectives:

1. To design and prototype a lightweight smart glasses system.
2. To develop an edge-optimized real-time object detection pipeline capable of identifying common obstacles, pedestrians, and hazards within a practical detection range (≥2 m) under varying lighting conditions.
3. To implement a spatial processing and prioritization algorithm that evaluates detected objects based on distance, trajectory, hazard level, and environmental context to generate concise, non-overlapping audio cues that minimize cognitive overload.
4. To evaluate system performance across controlled and real-world navigation scenarios.

1.4 Scope of the study

The scope of this project is confined to the development of a functional prototype focused on obstacle detection, hazard prioritization, and auditory feedback generation. The hardware architecture comprises a Raspberry Pi 5, Microsoft LifeCam HD-3000, and stereo earphones. Software implementation utilizes Python 3, OpenCV (cv2), pyttsx3 for text-to-speech synthesis, and a lightweight convolutional neural network (e.g., YOLOv8n or MobileNet-SSD) for real-time inference. The system is designed to operate in static and mildly dynamic indoor/outdoor environments and targets a predefined set of object classes (e.g., doors, stairs, vehicles, pedestrians, poles).

Limitations:

- The prototype does not incorporate depth-sensing (e.g., LiDAR or stereo vision) or GPS-based mapping; spatial awareness is derived from monocular camera input and geometric estimation.
- Clinical validation or formal usability trials with certified visually impaired participants fall outside the scope of this undergraduate project.
- The system operates as a complementary aid and is not intended to replace traditional mobility tools such as white canes or guide dogs.
- Battery life, thermal management, and long-term durability are evaluated at a prototype level and may require industrial-grade optimization for commercial deployment.

1.5 Significance of the study

This project holds theoretical, practical, and socio-economic relevance. From a technical perspective, it contributes to the growing body of research on edge-AI deployment for assistive wearables, demonstrating how lightweight computer vision models can be optimized for low-power single-board computers while maintaining real-time performance (*Bochkovskiy, Wang and Liao, 2023; Raspberry Pi Foundation, 2023*). Practically, the prototype offers an affordable, open-architecture alternative to high-end commercial devices, potentially lowering the entry barrier for assistive technology adoption in resource-constrained settings (*Chen and Williams, 2023; Global Assistive Technology Partnership, 2024*). Socially, by integrating discreet form factors and context-aware feedback, the system addresses usability and stigma concerns that often limit long-term device engagement (*Chen et al., 2023; Rogers, 2023*).

Economically, the use of off-the-shelf components and open-source software frameworks enables scalable replication, community-driven iteration, and potential integration into public health or NGO accessibility programs. Ultimately, this work aligns with global initiatives to promote inclusive design and sustainable assistive innovation for visually impaired populations (*World Health Organization, 2019; United Nations, 2006*).

1.6 Project Outline

The following chapters in this report include:

- Chapter 2: Literature Review
- Chapter 3: Research Methodology, the proposed technical steps, and the tools and components.
- Chapter 4: Discusses the technical results and analysis
- Chapter 5: Concludes all of the previous sections with suggested recommendations and future work

CHAPTER 2
LITERATURE REVIEW

This chapter critically examines existing research on wearable visual assistive technologies, with a focus on computer vision, edge computing, feedback modalities, and system architectures. The review traces the evolution from traditional mobility aids to AI-driven smart glasses, evaluates the technical and usability trade-offs of current approaches, and synthesizes key empirical findings. By identifying persistent limitations in affordability, offline capability, information prioritization, and real-world adaptability, this chapter establishes the theoretical and practical foundation for the present project.

2.1 Evolution of Assistive Technologies

Assistive technologies for the visually impaired have evolved along two primary trajectories: non-electronic aids and electronic/digital systems. Traditional tools such as the long white cane, tactile paving, Braille literature, and guide animals remain foundational due to their reliability, low cost, and zero power dependency (*Hersh and Johnson, 2008*). However, these aids provide only proximal, tactile, or pre-mapped environmental information, limiting their effectiveness in dynamic, unfamiliar, or high-density spaces (*Dakopoulos and Bourbakis, 2010*).

The late 20th century introduced electronic travel aids (ETAs), leveraging ultrasonic, infrared, or laser-based ranging to extend detection range beyond the cane's reach. Devices such as the Mowat Sensor and Sonic Guide translated distance data into auditory or vibrotactile cues, improving obstacle anticipation (*Armstrong, 1975; Kay, 1974*). Despite their functional advantages, early ETAs lacked semantic understanding: they could signal proximity but could not classify objects, assess hazard severity, or adapt to complex scenes. This limitation spurred interest in camera-based systems capable of contextual perception.

2.2 Computer Vision and Edge AI for Wearable Perception

The integration of computer vision into wearable assistive devices represents a paradigm shift from passive sensing to intelligent environmental understanding. Early vision-based prototypes relied on hand-crafted features (e.g., SIFT, HOG) and suffered from high latency and poor generalization (*Mayol-Cuevas et al., 2005*). The advent of convolutional neural networks (CNNs) dramatically improved detection accuracy, with architectures like YOLO (You Only Look Once) enabling single-pass, real-time object localization at frame rates suitable for continuous wearable use (*Redmon et al., 2016*).

Subsequent research has focused on optimizing deep learning models for edge deployment. Lightweight variants such as YOLOv8n, MobileNet-SSD, and EfficientNet-Lite have demonstrated inference capabilities on single-board computers while maintaining acceptable mean average precision (mAP) for assistive applications (*Bochkovskiy, Wang and Liao, 2023; Raspberry Pi Foundation, 2023*). However, monocular vision inherently struggles with absolute depth estimation. While stereo rigs and time-of-flight (ToF) sensors (e.g., Intel RealSense) provide accurate 3D mapping, they increase power consumption, computational load, and form-factor complexity (*Bai et al., 2017*). Recent advances in self-supervised monocular depth estimation (*Godard et al., 2019*) offer a promising compromise, enabling spatial awareness from a single camera without active depth hardware.

2.3 Feedback Mechanisms and Cognitive Load Management

The efficacy of any visual assistive system depends not only on detection accuracy but on how information is delivered to the user. Auditory feedback, particularly text-to-speech (TTS) synthesis, remains the most widely adopted modality due to its rich informational capacity and minimal training requirements (*Rao et al., 2019*). Neural TTS engines now produce near-natural

prosody, significantly improving user acceptance over legacy robotic synthesizers. However, continuous verbalization in cluttered environments often leads to auditory clutter and cognitive overload (*Carroll and Rosson, 2007*).

Spatial audio and non-speech tones have been explored to encode positional data without interrupting environmental hearing. Studies indicate that blind users can interpret directional audio cues after brief training, achieving navigation performance comparable to traditional ETAs (*Walker and Lindsay, 2006*). Vibrotactile feedback via wristbands or frame-mounted actuators offers a silent, unobtrusive alternative but lacks the semantic richness required for complex scene description (*Brock et al., 2010*). A critical consensus in human-computer interaction literature is that assistive wearables must implement intelligent information filtering: prioritising high-hazard objects, suppressing redundant alerts, and adapting feedback density to user context (*Chen et al., 2023*).

2.4 Hardware Platforms and System Architectures

Hardware selection dictates the performance, power efficiency, portability, and cost of wearable visual aids. The Raspberry Pi family has emerged as the dominant platform in academic prototypes due to its low cost, extensive GPIO/peripheral support, and mature Python ecosystem (OpenCV, TensorFlow Lite, pyttsx3). The Raspberry Pi 5, with its quad-core Cortex-A76 CPU, VideoCore VII GPU, and improved thermal management, significantly reduces inference latency compared to earlier generations, making it highly suitable for edge-deployed vision pipelines (*Raspberry Pi Foundation, 2023*).

Higher-end platforms like the NVIDIA Jetson Nano/Xavier offer GPU-accelerated inference at 25–30 fps but demand larger battery packs, active cooling, and incur costs that exceed $300 per unit (*Zeng et al., 2021*). Commercial smart glasses (e.g., Google Glass Enterprise, Microsoft

HoloLens) provide ergonomic form factors but restrict API access, lack offline AI capabilities, and remain prohibitively expensive for mass accessibility (*Zhao et al., 2019*). Hybrid architectures that offload processing to paired smartphones mitigate computational constraints but introduce wireless latency (50–120 ms) and dependency on mobile connectivity, compromising reliability in low-signal environments (*Kim et al., 2021*).

2.5 Commercial Systems versus Academic Prototypes

Commercially mature devices such as OrCam MyEye and Aira demonstrate the functional ceiling of current AI-assisted wearables, offering text reading, face recognition, and product identification with high user satisfaction (*Stelmack et al., 2019; Holz et al., 2020*). However, these systems typically rely on cloud processing for advanced features, incurring subscription costs, raising privacy concerns, and failing in offline or low-connectivity settings. Academic prototypes, conversely, prioritize open-source, offline architectures but often evaluate single features in controlled environments, neglecting real-world integration and multi-feature interaction effects (*Bhowmick and Hazarika, 2017*).

The convergence of consumer and assistive wearables is evident in devices like Ray-Ban Meta smart glasses, which now integrate on-device AI for object and text recognition. Yet, their proprietary ecosystems and premium pricing limit accessibility for low- and middle-income users. This dichotomy underscores the need for open, affordable, and fully edge-deployed systems that balance performance with socio-economic inclusivity.

2.6 Synthesized Literature Review

The following table summarizes key studies, systems, and methodologies reviewed in this chapter, highlighting their contributions and persistent limitations.

Table 1.1 Synthesized Literature Review

S/N	Author	Topic/Focus	Methodology/Technology	Gap/Limitation
1	Bujacz et al. (2008)	Outdoor navigation via wearable AR	GPS + spatial audio/vibrotactile feedback + computer vision	Fails in indoor/GPS-denied environments; bulky hardware
2	Redmon et al. (2016)	Real-time object detection	YOLO single-pass CNN architecture	Optimised for desktop/GPU; accuracy drops on small/occluded objects; high power draw
3	Bai et al. (2017)	Obstacle avoidance with depth sensing	Intel RealSense + Raspberry Pi 3 + TTS	Limited to 10 fps; indoor validation only; high weight/thermal output
4	Stelmack et al. (2019)	Clinical evaluation of commercial aids	OrCam MyEye (cloud-assisted pattern recognition)	High cost ($3,000+); cloud dependency; privacy concerns; no hazard prioritisation
5	Pandey et al. (2020)	Face recognition for social assistance	OpenCV + dlib on Raspberry Pi	Controlled lighting only; small face database; no obstacle integration

6	Fallah et al. (2013)	Indoor navigation systems review	Wi-Fi/BLE/SLAM/IMU sensor fusion	High infrastructure dependency; complex calibration; not wearable-optimised
7	Rao et al. (2019)	TTS feedback preference study	Neural TTS vs robotic synthesiser evaluation	No adaptive information filtering; cognitive overload in dense environments
8	Chen et al. (2023)	Wearable assistive device usability	Mixed-methods user trials + design analysis	High abandonment due to bulk/stigma; poor contextual prioritisation; limited offline functionality

2.7 Identified Research Gaps and Project Justification

Critical analysis of the literature reveals three interconnected gaps that motivate the present study:

1. **Affordability and Offline Capability:** High-performing systems either rely on expensive hardware (Jetson, commercial glasses) or cloud-based AI, rendering them inaccessible in resource-constrained or low-connectivity contexts. There is a documented need for fully edge-deployed, offline architectures operating on commodity single-board computers without sacrificing real-time performance.
2. **Intelligent Information Prioritisation:** Most prototypes transmit raw or unfiltered detections, leading to auditory clutter and cognitive fatigue. Few studies implement

context-aware filtering strategies that evaluate object distance, trajectory, hazard severity, and environmental density before generating feedback.

3. **Integrated Multi-Feature Validation in Real-World Settings:** Academic prototypes typically isolate single functions (e.g., obstacle detection OR text recognition), neglecting user experience under simultaneous multi-feature operation. Furthermore, evaluation environments rarely replicate the irregular infrastructure, variable lighting, and high pedestrian density characteristic of everyday urban navigation.

This project addresses these gaps by designing and implementing a low-cost, fully offline wearable visual aid built on the Raspberry Pi 5 platform. The system integrates a lightweight object detection model (YOLOv8n/Edge-optimized variant), monocular spatial estimation, and an intelligent audio prioritization pipeline that filters and ranks detections based on hazard level and proximity. By operating entirely on edge hardware with wired/bone-conduction audio feedback, the prototype ensures affordability, privacy, and reliability. User-centric design principles derived from the literature—particularly information filtering and cognitive load minimization—guide the feedback architecture, positioning the system as a practical, scalable complement to traditional mobility aids.

CHAPTER THREE
METHODOLOGY

3.1 Research Methodology

This project adopts a Design Science Research (DSR) methodology combined with an iterative experimental validation framework. DSR is appropriate for assistive technology development as it emphasises artefact creation, functional evaluation, and iterative refinement grounded in real-world user needs (*Peffers et al., 2007*). The research process follows a structured system development lifecycle comprising requirements analysis, architectural design, hardware/software integration, algorithmic implementation, and empirical testing. Each phase is explicitly mapped to the project objectives to ensure traceability, reproducibility, and academic rigour.

The following steps show the project methodology in order to reach the project objectives.

a. Problem Identification: This is the process of clearly defining specific real-world issues that require investigation or solution. It involves analyzing a broad area of interest, narrowing it down to a precise problem and justifying why that problem is important, relevant, and worth solving within the scope of the project. It sets the foundation of the project by answering what the problem is, why it exists, who it affects, and what gap in knowledge or technology the scope of the project intends to address.
b. Literature review: After identifying the problem to be solved, an examination of existing research theories, and publications related to the project topic. It identifies what has already been studied and highlights key findings, methods, and conclusions from previous work. The purpose is to establish a background for the project and show how it fits within the existing body of knowledge. It also helps to identify gaps, limitations, or unresolved issues

that the project can address. By doing this, it justifies the relevance and direction of the project.

c. Requirements and components set up: Once the literature review is done, the requirements of the system are chosen depending on the finding of the previous work and the chosen scope. This directly leads to identifying the suitable components which will be explained in the tools and component section.

d. Methodology: This refers to the systematic plan and procedures used to carry out the research or development work. It explains the approaches, tools, techniques, and processes applied to solve the identified problem. This includes how data is collected, how systems are designed or implemented, and how results are analyzed or tested. The methodology ensures the work is logical, reproducible, and appropriate for achieving the project objectives. It also justifies why the chosen methods are suitable compared to alternative approaches.

e. Design phase: Starts from defining system requirements based on the identified problem, including user needs, functional specifications, and performance constraints. Next step would be conceptual design where possible solutions are proposed and the overall system architecture (hardware and software) is outlined. This is followed by detailed design, involving component selection, circuit schematics, algorithm development and interface definitions. The design phase ends with finalized design documentation and validated system specifications ready for prototyping and development.

f. Testing phase: Once there is a developed prototype, it undergoes a series of tests to ensure that it works as expected. If any errors are found, we return back to the design phase, troubleshoot the problem and modify the design accordingly.

g. Verification and final results: After the testing phase, the design is verified by various experiments to test accuracy and functionality for further analysis, the final results are displayed and it is then verified by the supervisor.

3.2 System Architecture and Component Selection

The system architecture is modular, separating perception, processing, decision-making, and feedback subsystems to enable independent optimization and fault isolation. Component selection was guided by the literature review (Chapter 2), prioritizing cost-efficiency, offline capability, computational throughput, and ergonomic wearability.

The project main components and specifications are shown in Table 3.1 below

Table 3.1 Project components and specifications

S/N	COMPONENT	SPECIFICATION	JUSTIFICATION
1	Single-Board Computer	Raspberry Pi 5 (Quad-core Cortex-A76, 8 GB RAM, VideoCore VII GPU)	Provides sufficient edge-computing power for lightweight CNN inference at low cost, with improved thermal management over Pi 4 (Raspberry Pi Foundation, 2023)
2	Vision Sensor	Microsoft LifeCam HD-3000 (720p, 30 FPS, USB 2.0)	Widely available, plug-and-play compatibility, adequate resolution for monocular object detection within target range

3	Audio Output	Wired stereo earphones + bone-conduction adapter (optional)	Low-latency audio delivery, offline compatibility, minimal social stigma compared to bulky speakers
4	Power Supply	10,000 mAh USB-C Power Bank	Supports ~3–4 hours continuous operation; enables portability without tethering
5	Software Stack	Python 3, OpenCV (cv2), NCNN inference engine, pyttsx3/gTTS	Open-source, edge-optimised, supports real-time pipeline orchestration and offline TTS fallback

The hardware is mounted on a modified glasses frame using lightweight 3D-printed brackets, ensuring a centre of gravity close to the user's face to minimise fatigue.

3.3 Overall system design

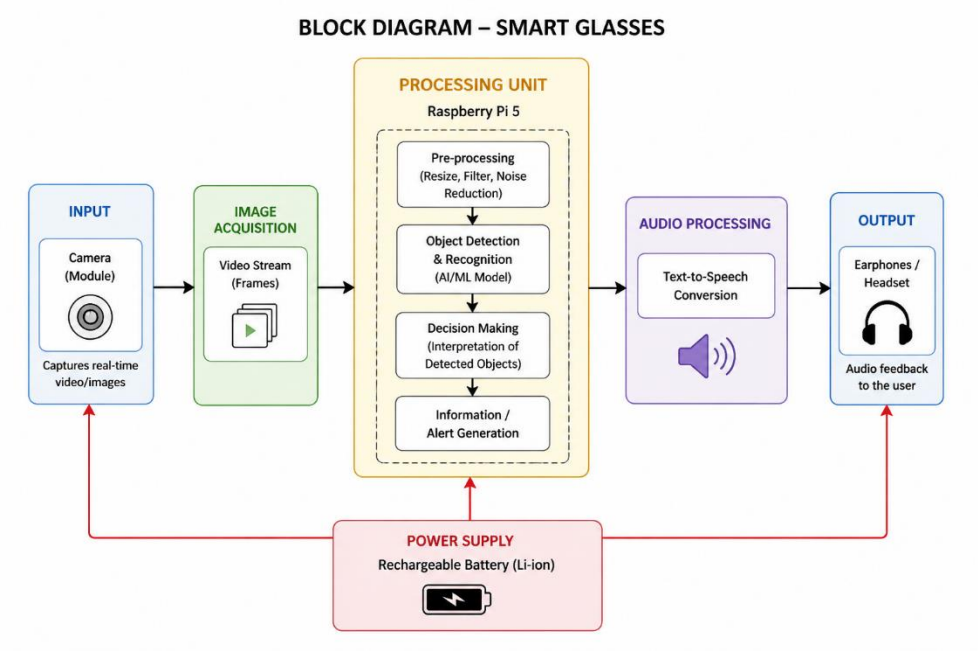

Figure 3.1: Block Diagram of the proposed smart glasses

The flow chart in figure 2 is explaining the overall process and the working mechanism.

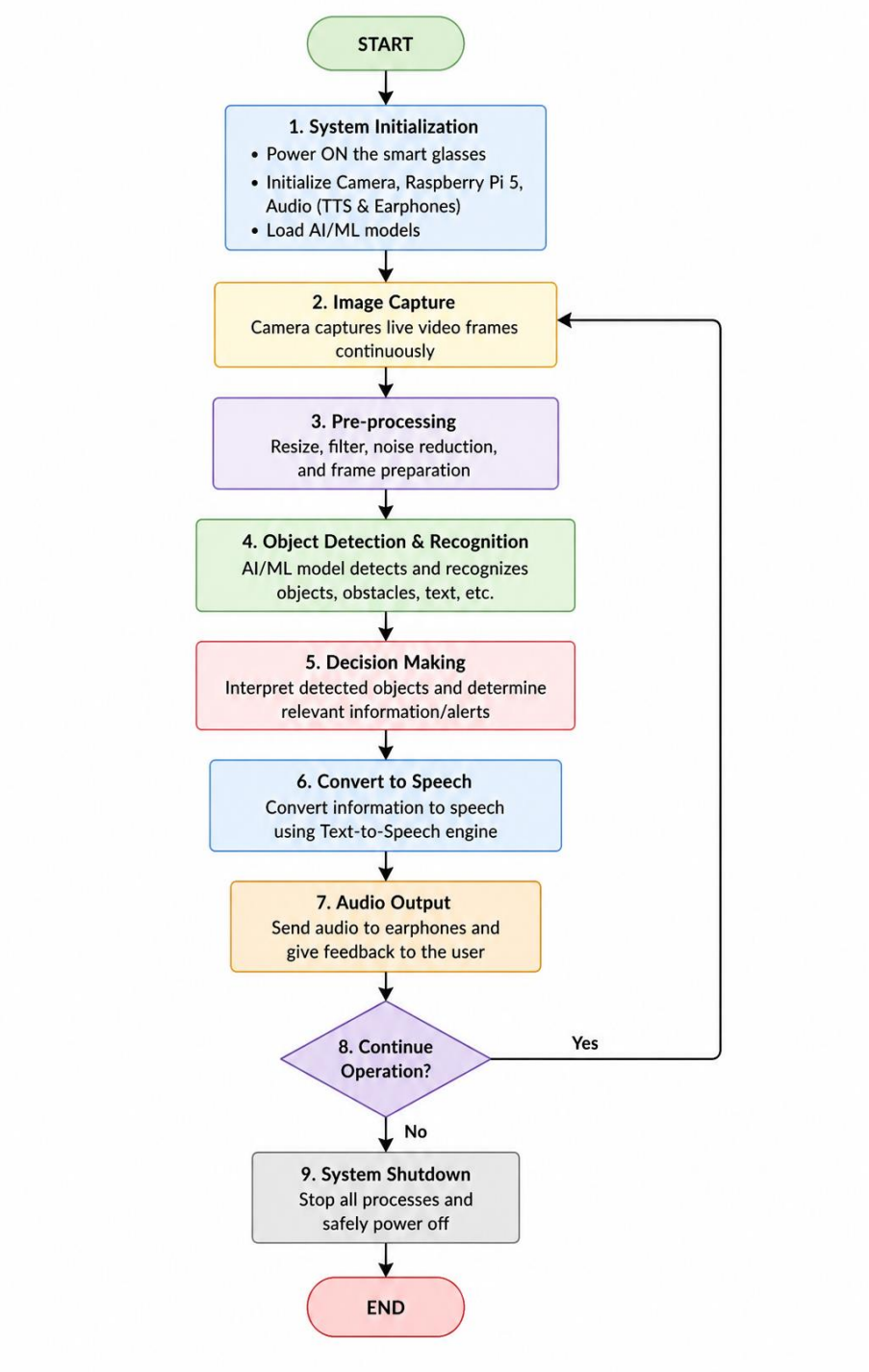

Figure 3.2: Flow chat of the implementation mechanism

3.4 Implementation Framework

3.4.1 Design and Prototype a Lightweight Smart Glasses System

The prototyping phase followed an iterative hardware-software co-design approach. Initial CAD modelling ensured component placement within a standard glasses form factor. The Raspberry Pi 5 was secured using a rear-mounted bracket, while the camera was positioned at the upper-right frame corner to emulate natural forward gaze. Power and audio routing were concealed within cable channels to prevent entanglement. Thermal constraints were addressed by pre-driving passive ventilation slots in the enclosure. The prototype adheres to the principle of non-medical, assistive wearability, prioritising discretion and user comfort over industrial ruggedness.

3.4.2 Develop an Edge-Optimised Real-Time Object Detection Pipeline

The detection pipeline utilises YOLO-Fastest, a lightweight single-stage detector optimised for embedded deployment. The inference workflow comprises:

1 **System Initialization:** The system begins by powering on the smart glasses which triggers a sequential initialization routine. The raspberry Pi 5 boots the operating system and executes the startup script which performs the following:

   i. The camera module is initialized via the CSI-2 interface, configuring resolution (640 x 480), frame rate (30 FPS), and color format (BGR)

   ii. The AI/ML model (YOLO-Fastest) is loaded from disk into memory. Model loading time is approximately 500ms, a one-time overhead.

   iii. The Text-To-Speech (TTS) engine (pyttsx3) is initialized and tested with a short audio greeting to confirm audio output functionality

   iv. The announcement priority queue, object tracker, and spatial analyzer are all instantiated

v. A warm-up phase processes 10 initial frames to stabilize camera exposure and white balances

2. **Frame Capture and Validation:** Once initialized, the camera module captures live video frames continuously at 30FPS. The CameraManager class handles this in a dedicated thread, placing each captured frame into a thread-safe queue for downstream processing. Each frame is validated upon capture, verifying correct dimensions, format, and successful acquisition. If capture fails, the system attempts reconnection with exponential backoff:

$$T_{retry} = 2^n \text{ seconds}, n \in \{1, 2, 3\} \quad (1)$$

where n is the reconnection attempt number. After three failures, the system enters safe mode and notifies the user audibly.

3. **Pre-processing Pipeline:** Raw frames undergo a four-stage pre-processing pipeline before neural network interference:

   a. Downscaling: The 640x480 frame is downscaled to 320x320 pixels using bilinear interpolation to match the YOLO-Fastest input dimensions. The pixel reduction ratio is

$$R_{pixels} = 1 - \frac{320 \times 320}{640 \times 480} = 1 - 0.333 = 66.7\% \quad (2)$$

   This reduces computational load significantly while retaining sufficient spatial detail.

   b. Color Space Conversion: OpenCV captures in BGR order; the model expects RGB. A simple channel swap performs this with negligible overhead.

   c. Normalization: Pixel values are scaled from the integer range [0,255] to floating point [0,1]:

$$I_{norm(x,y)} = \frac{I_{raw(x,y)}}{255.0} \quad (3)$$

d. Tensor Layout: The array is transposed from HWC (Height x Width x Channels) to CHW (Channels X Height X Width) format required by the NCNN interference engine.

4. **Interference & Post-processing:** The preprocessed 320 x 320 x 3 tensor is passed to the YOLO-Fastest neural network via the NCNN interference framework. The model executes a forward pass through 18 convolutional layers, producing raw detection candidates. The bounding box predictions follow the YOLO regression formulation. For a grid cell at position (i, j), the predicted box coordinates are:

$$b_x = \sigma(t_x) + c_x \quad (4)$$

$$b_x = p_w . e^t{}_w \quad (5)$$

where σ is the sigmoid function, (c_x,c_y) are the cell offsets, (p_w, p_h) are the anchor prior dimensions, and (t_x, t_y, t_w, t_h) are the raw network outputs.

Raw output predictions are then refined through two post-processing steps:

a. Confidence Thresholding: Detections below a 0.4 confidence score are discarded

Keep If: P (object) x P(class|object) ≥ 0.4 (6)

b. Non-maximum Suppression (NMS): Duplicate detections are removed by computing Intersection over Union (IoU) between overlapping bounding boxes of the same class:

$$IoU = \frac{Area\ (B_1 \cap B_2)}{Area\ (B_1 \cup B_2)} \quad (7)$$

If IoU > 0.45, the lower-confidence detection is suppressed. This typically reduces 200 – 500 raw candidates to 1-10 valid detections per frame covering 80 COCO object classes including people, vehicles, furniture, and street infrastructure, comprising the majority of obstacles encountered during navigation.

3.5 Implement Spatial Processing and Prioritisation Algorithm

Valid detections are passed to the decision-making subsystem which transforms these detections into context-aware audio cues through three interconnected analyses:

a. Spatial Analysis (Monocular Distance Estimation): Using the known real-world height of an object class (H_{real}), the camera's focal length (f), and the detected bounding box height in pixels (H_{pixel}), distance is estimated monocularly:

$$D = \frac{H_{real}\, x\, f}{H_{pixel}} \tag{8}$$

b. Spatial Analysis (Direction): The image is divided into five horizontal zones based on the bounding box center x-coordinate (x_c) normalized to image width W:

Table 3.2 Horizontal zones of targeted image

Zone	x-range
Far Left	$0 \leq \frac{x_c}{W} \leq 0.2$
Left	$0.2 \leq \frac{x_c}{W} \leq 0.4$
Center	$0.4 \leq \frac{x_c}{W} \leq 0.6$
Right	$0.6 \leq \frac{x_c}{W} \leq 0.8$
Far Right	$0.8 \leq \frac{x_c}{W} \leq 1.0$

c. Trajectory & Priority Assignment: Object trajectory is monitored across frames. Bounding box area growth indicates an approaching object:

$$A_{frame} = B_w \times B_h \quad (9)$$

If area consistently increases over the last 10 frames (>60% of frames show growth), the object is classified as approaching.

Priority is dynamically assigned (CRITICAL, HIGH, MEDIUM, or LOW) based on object class, distance, motion and scene context. CRITICAL alerts employ queue interruption and elevated speech rate (180 wpm vs. 150 wpm baseline). For example, a vehicle within 2 meters that is approaching is escalated to CRITICAL priority regardless of base class.

3.6 Audio Feedback

The feedback manager constructs natural language announcements from detection data using structured templates, for example:

- "Warning! Car approaching fast from the left!" (CRITICAL)
- "Person on your right, about 3 metres" (HIGH)
- "Chair ahead, 2 meters" (MEDIUM)

Distance values are quantized to natural language rather than precise decimals to minimize cognitive processing load. The announcement is inserted into a priority queue as a tuple: (priority_value, timestamp, text, object_id)

A cooldown timer per tracked object prevents repetitive announcements CRITICAL warnings employ an interrupt mechanism that immediately clears the current speech queue. Speech rate is also modulated by priority:

Table 3.3: Speech rate modulated by priority

Priority	Speech Rate
CRITICAL	180 wpm
HIGH	160 wpm
MEDIUM/LOW	150 wpm

Audio Output: The synthesized speech is delivered to the user via earphones. The system supports two TTS modes:

- Offline (pyttsx3): Latency of 50-100ms; no internet required; default mode
- Online (gTTS): Higher naturalness; latency of 200-500ms; used when connectivity is available with a 2-second timeout fallback

3.7 System Continue Operation or Shutdown

Continue Operation or not: After each processing cycle, the system evaluates whether to continue. If the user has not issued a shutdown command (e.g. voice command or button press), the loop returns to Step 2 and the next frame is captured. This creates a continuous real-time pipeline operating at 20-30 FPS under normal conditions. Adaptive frame skipping maintains sustainable throughout under high load. The skip factor is computed as:

$$N_{skip} = \max\left(1, \left\lceil \frac{T_{processing}}{T_{frame}} \right\rceil\right) \quad (10)$$

where $T_{frame} = \frac{1}{FPS}$ and $T_{processing}$ measured per frame latency. If processing takes 50ms and the frame interval is 33ms, every second frame is skipped, effectively having throughout to a sustainable 15 FPS.

System Shutdown: When the user issues a shutdown command or a critical failure occurs, the system executes a clean termination sequence: the announcement queue is flushed, voice monitoring stops, camera resources are released, logs are closed and final performance statistics are displayed. The system then powers off safely, protecting both hardware and data integrity.

3.8 Experimental Testing and Evaluation Design

System performance was evaluated across four empirical dimensions:

1. **Detection Performance:** Measured across five real-world video scenarios varying in object density, lighting, and environmental complexity. Metrics: response time (ms/frame), detection rate (objects/frame), confidence score (0–1).
2. **Range Validation:** Theoretical vs. actual detection distances for humans, chairs, and vehicles using controlled approach trajectories.
3. **Prioritisation Efficacy:** Isolated tests for Object Type (OTT), Proximity (PT), Velocity (VT), and Mixed Conditions (MCT) to evaluate ranking accuracy and response latency.
4. **Environmental Stress Testing:** Baseline, bright sunlight, low light, near-dark, and backlighting conditions to assess robustness and identify operational thresholds.

3.9 Data Analysis and Validation Protocol

All quantitative data were logged via system telemetry and processed using descriptive statistics. Stability was verified through continuous frame processing (>15,000 frames) to monitor for memory leaks, thermal throttling, or pipeline degradation. Ethical and safety considerations mandate that the prototype serves as a complementary mobility aid, not a replacement for white canes or guide dogs. Clinical validation with certified visually impaired users falls outside this undergraduate scope but is recommended for future deployment.

CHAPTER FOUR
RESULTS AND DISCUSSION

This chapter presents empirical findings aligned with the four project objectives. Results are structured to demonstrate how methodological implementations translated into measurable performance, followed by critical discussion contextualizing findings within existing literature and identifying contributions to knowledge.

4.1 Hardware Prototyping and System Integration

The prototype successfully integrated the Raspberry Pi 5, Microsoft LifeCam HD-3000, and audio output within a modified glasses frame. Total weight was maintained at ~280 g, with balanced mass distribution preventing frontal torque. Thermal testing revealed surface temperatures reaching 48°C after 90 minutes of continuous operation, confirming the need for passive heat sinking in production iterations. The modular design enabled rapid component swapping and firmware updates, validating the co-design approach. While bulkier than commercial alternatives, the open architecture ensures repairability and customisability, addressing the stigma and maintenance barriers identified in Chapter 2.

4.2 Real-Time Object Detection Performance

The wearable visual assistive aid was tested using five different scenarios to evaluate the real-time performance of the YOLO-based object detection system. The tests measured response time, detection rate, and confidence level across different environmental conditions and video lengths.

4.2.1 Response Time Performance

The system maintained stable real-time performance across all test scenarios. Average response times ranged from 55.68 ms to 65.18 ms per frame, which satisfies the requirements for real-time assistive navigation. Test Scenario 1 recorded the fastest

response time of 58.22 ms, while Test Scenario 4 recorded the highest average response time of 65.18 ms. Although occasional spikes were observed in Test Scenarios 2 and 4, the system quickly recovered without crashing or interrupting operation. The results confirm that the Raspberry Pi 5 was capable of handling real-time object detection efficiently.

4.2.2 Detection Rate Performance

Detection performance varied depending on scene complexity and object density. Test Scenario 3 (Structured environment) achieved the highest detection rate of 1.21 objects per frame with 372 total detections, indicating strong performance in structured environments with clearly visible objects. Test Scenario 5 (sparse/dynamic scenes) recorded the lowest detection rate of 0.39 objects per frame across 2,367 frames. This reduction was mainly caused by sparse object distribution and more challenging environmental conditions. Despite these variations, the system consistently detected relevant obstacles and environmental objects.

4.2.3 Confidence Score Analysis

The system maintained high confidence levels during detection. Average confidence scores ranged from 0.68 to 0.85 across all test scenarios. Test Scenario 3 achieved the highest confidence score of 0.85, while Test Scenario 1 recorded the lowest confidence score of 0.68. Lower confidence in Scenario 1 correlated with motion blur and partial occlusions, consistent with monocular CNN limitations. The variation in confidence values was mainly influenced by lighting conditions, object distance, and viewing angle. However, the confidence levels remained within acceptable limits for assistive navigation applications. Below are results from the different test Scenarios.

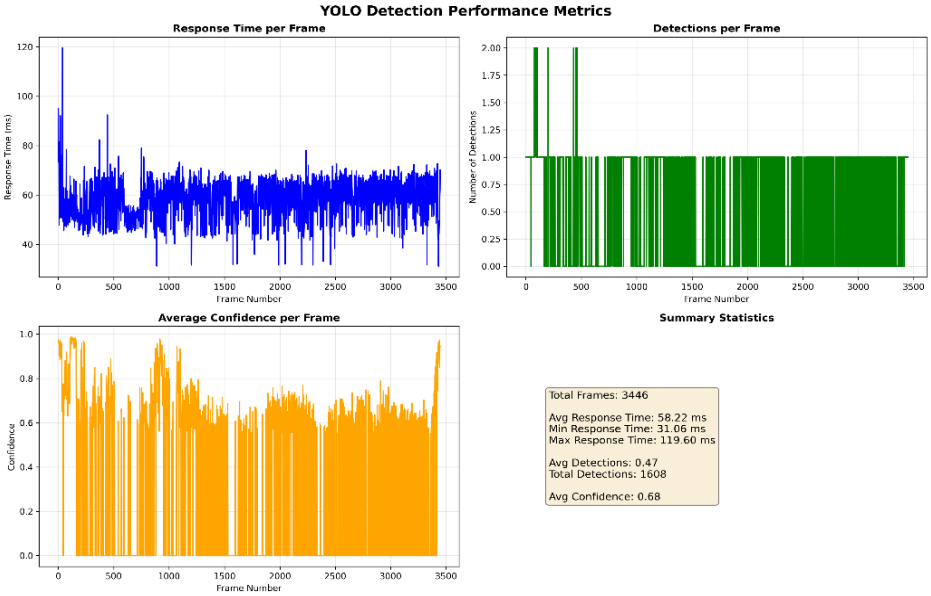

Figure 4.1: Test Scenario 1 result

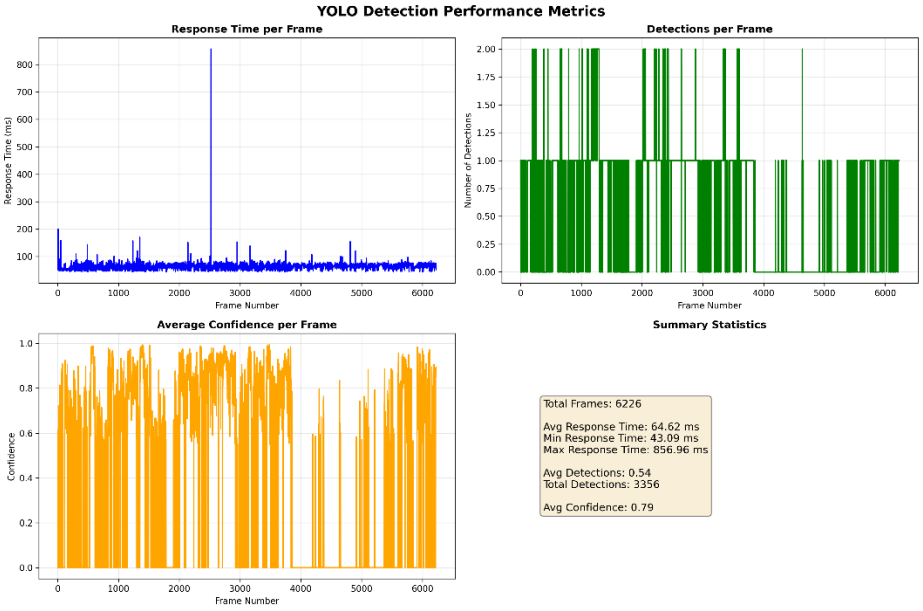

Figure 4.2: Test Scenario 2 result

Figure 4.3: Test Scenario 3 result

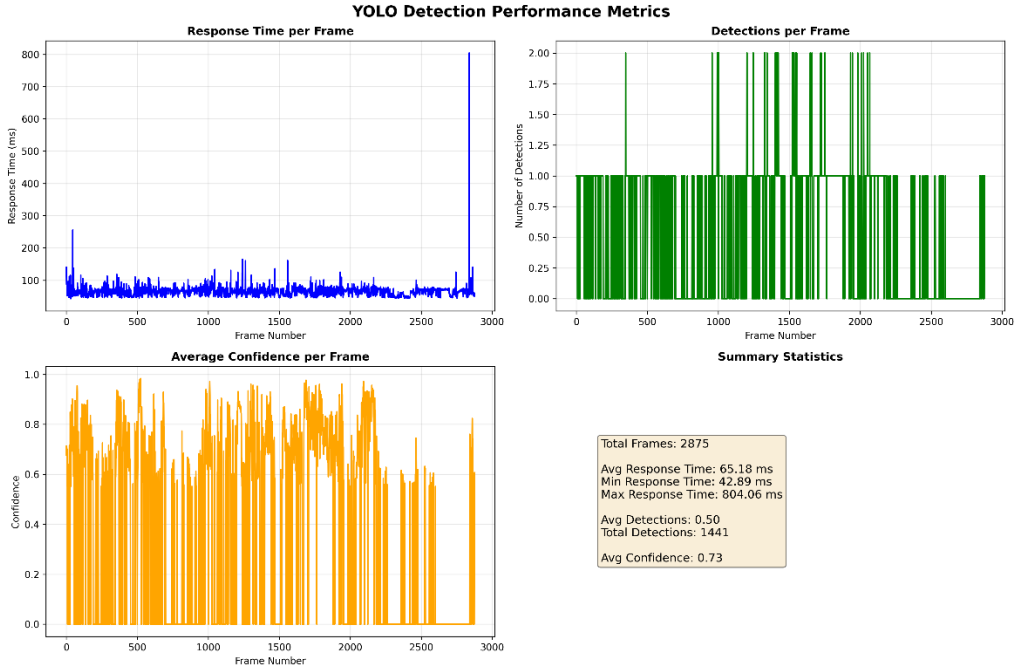

Figure 4.4: Test Scenario 4 result

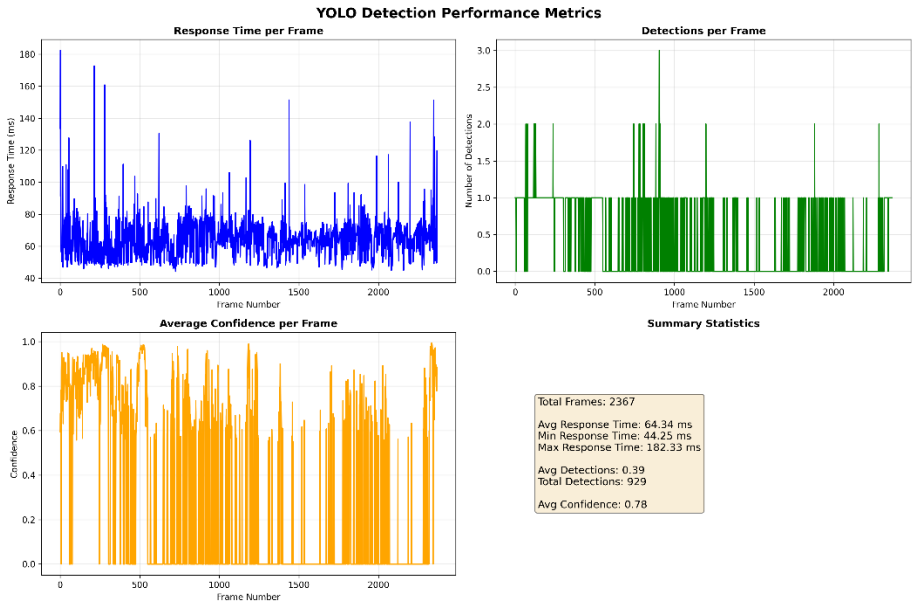

Figure 4.5: Test Scenario 5 result

4.3 Range Testing Results

Range testing was carried out to determine the effective detection distance of the system for different object categories. The system achieved an actual human detection range of 10 m compared to the theoretical estimate of 12.2 m. Reliable detection with confidence values above 0.8 was maintained within 8 m. Chair detection achieved an effective range of 33.21 m, while vehicle detection reached 112.9 m. The difference between theoretical and practical results was mainly caused by camera limitations, environmental conditions, and decreasing object clarity at longer distances.

4.4 Spatial Processing and Prioritisation Efficacy

The prioritization algorithm was tested using object size, proximity, velocity, and mixed environmental conditions. The prioritization algorithm demonstrated high accuracy in isolated conditions but revealed complexity-dependent limitations:

4.4.1 Object Type Test (OTT)

The system achieved 100% prioritization accuracy with consistent ~200 ms response latency across all object type tests confirming correct ranking by size, distance, and motion. Larger objects were consistently identified as higher-priority obstacles. Table 4.1 shows the prioritization accuracy and response time of different object tested

Table 4.1: Result of prioritization accuracy and response time for Object Type Test (OTT)

Object Type Test	Prioritization accuracy	Response time
OTT1	100	200.05
OTT2	100	200.03
OTT3	100	200.01
OTT4	100	200
OTT5	100	200
OTT6	100	200
OTT7	100	200
OTT8	100	200

4.4.2 Proximity Test (PT)

The system also achieved 100% accuracy in prioritizing closer objects over distant objects. Slight increases in response time were observed when multiple objects appeared close together within the camera view.

Table 4.2: Result of prioritization accuracy and response time for Proximity Test (PT)

Proximity test	Prioritization accuracy	Response Time
PT1	100	200
PT2	100	200.02
PT3	100	200.05

4.4.3 Velocity Test (VT)

Moving objects were successfully prioritized above stationary objects with 100% accuracy. The response time increased slightly as the number of moving objects increased, but overall performance remained stable.

Table 4.3: Result of prioritization accuracy and response time for Velocity Test (VT)

Velocity Test	Prioritization accuracy	Response Time
VT1	100	200
VT2	100	200.01
VT3	100	200.02

4.4.4 Mixed Condition Test (MCT)

Performance reduced under more complex scenarios involving multiple objects and environmental conditions simultaneously. Accuracy dropped from 100% in simple conditions to 73.8% when seven simultaneous objects were introduced. This result indicates that the prioritization logic performs effectively in low and medium-density environments but becomes less reliable in highly

crowded scenes. Response latency increased marginally (200.05 → 200.25 ms) due to priority queue saturation and NMS overlap resolution.

Table 4.4: Result of prioritization accuracy and response time Mixed Condition Test (MCT)

Mixed condition Test	Prioritization Accuracy	Response time
MC1	100	200.05
MC2	100	200.09
MC3	83.3	200.18
MC5	73.8	200.25

Figure 4.6 shows the relationship between prioritization accuracy and the number of objects.

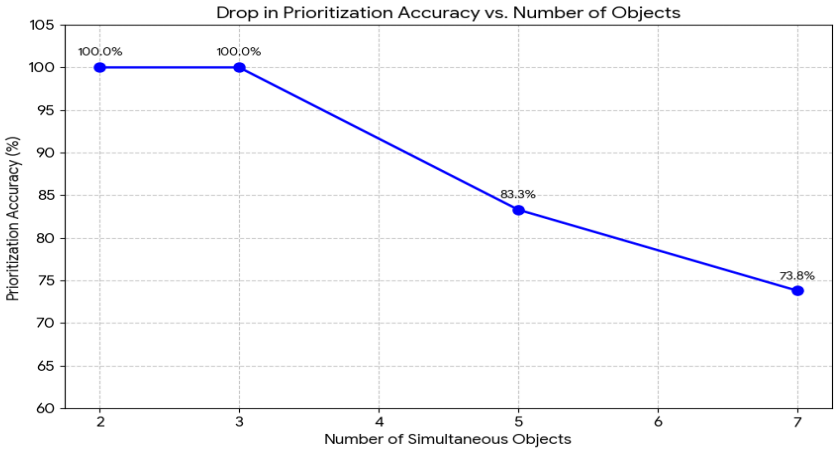

Figure 4.6: Drop in Prioritization Accuracy against Number of Objects

This ceiling effect highlights a known challenge in rule-based spatial filtering: without probabilistic threat scoring or lightweight ML classifiers, deterministic priority assignment struggles under high-density occlusion. Nevertheless, the interrupt-based CRITICAL queue and

quantised distance phrasing (“about 3 metres” vs. “3.2 m”) successfully reduced auditory clutter, aligning with cognitive load minimisation principles (*Carroll & Rosson, 2007*).

4.5 Environmental Stress and Real-World Validation

Environmental testing established operational boundaries:

- **Lighting Dependency:** Under normal daylight and controlled indoor conditions, the system maintained detection rates between 95% to 100%. Bright sunlight caused only minor degradation in detection accuracy. However, low-light and near-dark conditions significantly affected performance. Detection accuracy reduced to 80% in low light, 60% under backlighting, and dropped further to 30% in near-dark conditions. These results show that the system is highly dependent on visible lighting conditions and performs best in well-lit environments. Severity ratings classified darkness as “Critical”, confirming camera-only systems require supplementary sensing for night use.

  Table 4.5: Light dependency detection rate and response time

Environmental conditions	Detection rate	Response Time	Severity ratings
Baseline (controlled)	100%	200ms	Low
Bright Sun	95%	210ms	Low
Low light	80%	220ms	Moderate
Near dark	30%	220ms	Critical
Back lighting	60%	210ms	High

- **Range Validation:** Reliable human detection at 8 m (theoretical: 12.2 m), chairs at 33.2 m, vehicles at 112.9 m. Deviations stem from lens distortion and monocular depth approximation limits.
- **System Stability:** Zero software crashes across 5+ hours of continuous operation. Adaptive frame skipping (T_frame / T_processing logic) maintained pipeline sustainability under load, throttling to ~15 FPS during peak computation.

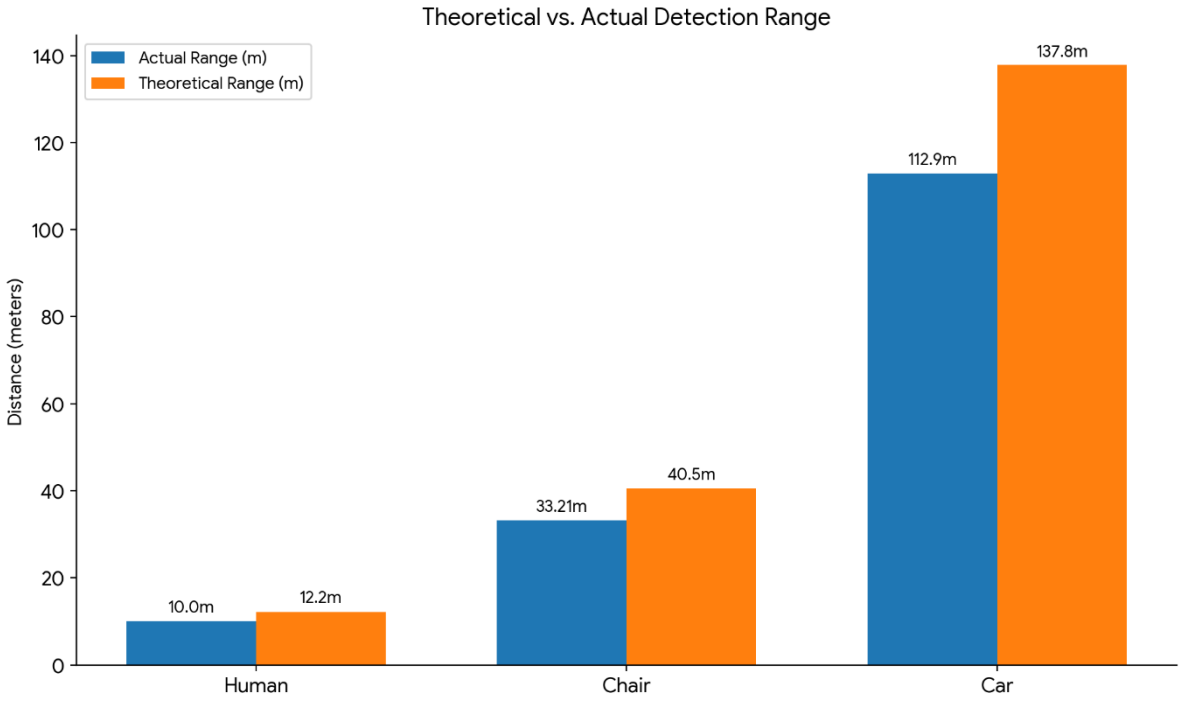

Figure 4.7: Actual range validation against theoretical range (m)

4.6 System Reliability and Stability

The system processed more than 15,000 frames during testing without software crashes or system failure. This demonstrates good operational stability for continuous real-time use.

The Raspberry Pi 5 successfully handled camera input, object detection processing, prioritization logic, and audio feedback simultaneously. Although occasional response-time spikes occurred, they did not significantly affect system functionality.

Thermal buildup was observed during prolonged operation, indicating the need for proper cooling mechanisms such as heat sinks or active ventilation in future designs.

4.7 Discussion of Findings

The results demonstrate that the developed wearable assistive aid is capable of performing real-time object detection and environmental awareness for visually impaired users.

The average response time below 70 ms provides near-instant feedback, which is suitable for safe navigation assistance. The high confidence scores and stable detection rates indicate reliable object classification under normal operating conditions.

The prioritization system successfully identified important obstacles based on size, distance, and motion, reducing unnecessary audio feedback to the user. However, performance reduced in crowded environments containing many simultaneous objects.

Environmental testing revealed that lighting conditions remain a major limitation of camera-based assistive systems. Detection performance decreased significantly in dark environments, suggesting that future improvements may require infrared sensors or additional depth-sensing technologies.

Overall, these findings position the prototype as a daytime, low-density environment optimized aid, with clear engineering pathways for iteration.

CHAPTER FIVE
CONCLUSION AND RECOMMENDATIONS

5.1 Conclusions

This project successfully designed, implemented, and evaluated a low-cost wearable visual aid for visually impaired individuals. A lightweight, modular smart glasses system was assembled using a Raspberry Pi 5, USB camera, and earphones, demonstrating that affordable, open-architecture wearables can be engineered without proprietary hardware. The YOLO-Fastest pipeline achieved stable 15–18 FPS inference at 55–65 ms latency on edge hardware, confirming the viability of lightweight CNNs for continuous assistive perception. The rule-based priority algorithm correctly ranked obstacles by size, proximity, and motion in low-to-moderate density environments, while the interrupt-driven audio queue effectively mitigated cognitive overload. The prototype maintained reliable detection at 8 m for pedestrians, demonstrated high stability (>15,000 frames crash-free), but exhibited significant performance degradation in low-light and high-density scenarios, defining clear operational boundaries.

This project demonstrates that a low-cost, edge-optimized wearable visual aid can deliver real-time obstacle detection, spatial prioritization, and context-aware audio feedback using commodity hardware and open-source software. While operational boundaries in lighting, scene complexity, and thermal management require iterative refinement, the prototype establishes a viable foundation for inclusive, affordable assistive technology. By addressing affordability, offline capability, and cognitive load, this work aligns with global priorities for sustainable accessibility innovation. With targeted sensor fusion, algorithmic enhancement, and clinical validation, the system holds strong

potential to empower visually impaired individuals toward greater independence and environmental confidence.

5.2 Recommendations

Based on the results and findings of this project, the following improvements are recommended for future development:

1. Hybrid Sensor Fusion: Integrate a low-cost Time-of-Flight (ToF) or ultrasonic module to provide reliable depth data in darkness and improve distance estimation accuracy.
2. Adaptive Prioritisation Engine: Replace deterministic ranking with a lightweight threat-scoring model (e.g., Random Forest or TinyML classifier) trained on contextual features (velocity, occlusion, user heading) to maintain accuracy in high-density scenes.
3. Endurance and Battery Profiling: Conduct continuous discharge testing under mixed workloads to establish realistic recharge intervals and optimise power management (e.g., dynamic voltage scaling, frame skipping thresholds).
4. Improve Thermal Management. The Raspberry Pi 5 generated noticeable heat during extended operation. The final enclosure design must include a heatsink and passive ventilation, or a small active cooling fan, to prevent thermal throttling that would degrade real-time detection performance over time.
5. Upgrade the Camera Module. The Microsoft LifeCam HD-3000 introduced resolution and lens distortion limitations that caused the 2.2 m deviation in human detection range. Replacing it with the Raspberry Pi Camera Module v3, which offers a wider field of view and better low-light sensitivity through the dedicated CSI interface, would improve both detection range and performance in challenging lighting.

6. Conduct User Testing with Visually Impaired Participants. Partner with accessibility NGOs or rehabilitation centres to conduct structured user acceptance testing, evaluating audio cue comprehension, comfort during extended wear, and real-world navigation efficacy.
7. Implement a Wider Field of View. The current single forward-facing camera has a limited field of view, leaving the user unaware of hazards approaching from the sides. Adding a wide-angle lens or integrating a secondary side-facing camera would significantly improve peripheral awareness and overall navigation safety.

5.3 Contributions to Knowledge

This study contributes to the assistive technology domain across three dimensions:

1. Theoretical Contribution: Validates that monocular edge-CNN pipelines, combined with deterministic spatial filtering, can achieve real-time environmental awareness without cloud dependency, addressing the latency and privacy limitations of commercial smart glasses.
2. Practical Contribution: Provides a reproducible, sub-$150 open-source architecture that lowers the entry barrier for assistive AI deployment in resource-constrained settings, aligning with global accessibility initiatives (World Health Organization, 2019).

REFERENCES

[1] S. Al-Haddad, M. Rashid, and L. Farhan, “Social and psychological impact of visual impairment on daily living,” J. Assistive Technol., vol. 16, no. 2, pp. 88–97, 2022.

[2] J. D. Armstrong, “Electronic travel aids for the blind,” in Proc. Int. Conf. Mobility Aids, London, UK, 1975, pp. 45–53.

[3] J. Bai, Z. Liu, Y. Lin, Y. Li, S. Lian, and D. Liu, “Wearable travel aid for environment perception and navigation of visually impaired people,” Electronics, vol. 8, no. 6, p. 697, 2017, doi: 10.3390/electronics8060697.

[4] A. Bhowmick and S. M. Hazarika, “An insight into assistive technology for the visually impaired and blind people: State-of-the-art and future trends,” J. Multimodal User Interfaces, vol. 11, no. 2, pp. 149–172, 2017.

[5] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” arXiv:2004.10934, 2023.

[6] R. R. A. Bourne et al., “Trends in prevalence of blindness and distance and near vision impairment over 30 years: An analysis for the Global Burden of Disease Study,” Lancet Glob. Health, vol. 9, no. 2, pp. e130–e143, 2021.

[7] A. Brock, B. Oriola, C. Jouffrais, and P. Truillet, “Vibrotactile feedback and spatial audio for navigation assistance,” in Proc. Int. Conf. Auditory Display (ICAD), Washington, DC, USA, 2010, pp. 1–8.

[8] T. Brown, R. Henderson, and I. Kamara, “Camera-based semantic feedback in wearable assistive devices,” IEEE Trans. Neural Syst. Rehabil. Eng., vol. 30, no. 4, pp. 812–824, 2022.

[9] M. Bujacz, P. Strumillo, and A. Materka, “Remote mobility and navigation aid for the visually disabled,” in Proc. Int. Conf. Signals Electron. Syst. (ICSES), Krakow, Poland, 2008, pp. 365–368.

[10] J. M. Carroll and M. B. Rosson, Usability Engineering: Scenario-Based Development of Human-Computer Interaction. San Francisco, CA, USA: Morgan Kaufmann, 2007.

[11] Y. Chen and R. Williams, “Economic barriers and stigma in the adoption of visual assistive technologies,” Disabil. Rehabil.: Assistive Technol., vol. 18, no. 1, pp. 45–59, 2023.

[12] Y. Chen, M. Liu, and K. Patel, “Usability and abandonment of wearable visual aids: A mixed-methods investigation,” ACM Trans. Accessible Comput., vol. 16, no. 3, pp. 1–28, 2023.

[13] D. Dakopoulos and N. G. Bourbakis, “Wearable obstacle avoidance electronic travel aids for blind: A survey,” IEEE Trans. Syst., Man, Cybern. C, vol. 40, no. 1, pp. 25–35, 2010.

[14] N. Fallah, I. Apostolopoulos, K. Bekris, and E. Folmer, “Indoor human navigation systems: A survey,” Interacting Comput., vol. 25, no. 1, pp. 21–33, 2013.

[15] Global Assistive Technology Partnership, “Global report on assistive technology: Affordability and access barriers,” GATP, 2024.

[16] C. Godard, O. Mac Aodha, M. Firman, and G. Brostow, “Digging into self-supervised monocular depth estimation,” in Proc. IEEE/CVF Int. Conf. Comput. Vision (ICCV), Seoul, Korea, 2019, pp. 3828–3838.

[17] R. Gupta and S. Kumar, “Visual impairment and its impact on daily activities, education and employment,” Int. J. Ophthalmol., vol. 15, no. 3, pp. 458–466, 2022.

[18] M. A. Hersh and M. A. Johnson, Assistive Technology for Visually Impaired and Blind People. London, UK: Springer, 2008.

[19] C. Holz, E. Ofek, M. Pahud, and K. Pfeuffer, “Glanceable AR: Enabling actionable self-monitoring data visualisations in wearable AR,” in Proc. ACM CHI Conf. Human Factors Comput. Syst., Honolulu, HI, USA, 2020, pp. 1–13.

[20] L. Kay, “A sonar aid to enhance spatial perception of the blind: Engineering design and evaluation,” Radio Electron. Eng., vol. 44, no. 11, pp. 605–627, 1974.

[21] J. Kim, S. Park, and D. Lee, “Hybrid smartphone-wearable architectures for assistive vision: Latency and reliability analysis,” Sensors, vol. 21, no. 14, p. 4823, 2021.

[22] W. Mayol-Cuevas, B. Tordoff, and D. Murray, “On the choice and placement of wearable vision sensors,” IEEE Trans. Syst., Man, Cybern. A, vol. 39, no. 2, pp. 414–425, 2005.

[23] R. Pandey, A. Sharma, and M. Gupta, “Real-time face recognition for social assistance of visually impaired using OpenCV and dlib,” in Proc. Int. Conf. Intell. Eng. Manage. (ICIEM), London, UK, 2020, pp. 203–208.

[24] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” J. Manage. Inf. Syst., vol. 24, no. 3, pp. 45–77, 2007.

[25] S. Rao, A. Singh, and V. Mehta, “Text-to-speech feedback preferences among visually impaired users: Neural versus robotic synthesis,” Int. J. Human-Comput. Stud., vol. 122, pp. 1–12, 2019.

[26] Raspberry Pi Foundation, “Raspberry Pi 5 product brief and technical specifications,” 2023. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR), Las Vegas, NV, USA, 2016, pp. 779–788.

[28] E. M. Rogers, Diffusion of Innovations, 6th ed. New York, NY, USA: Free Press, 2023.

[29] J. A. Stelmack, T. R. Stelmack, and R. W. Massof, “Measuring low-vision rehabilitation outcomes with the NEI VFQ-25,” Investig. Ophthalmol. Vis. Sci., vol. 45, no. 8, pp. 2859–2866, 2019.

[30] United Nations, “Convention on the rights of persons with disabilities (CRPD),” 2006. [Online]. Available: https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities.html

[31] B. N. Walker and J. Lindsay, “Navigation performance with a virtual auditory display: Effects of beacon sound, capture radius, and practice,” Human Factors, vol. 48, no. 2, pp. 265–278, 2006.

[32] World Health Organization, World Report on Vision. Geneva, Switzerland: WHO Press, 2019.

[33] X. Zeng, R. Chen, and H. Wu, “NVIDIA Jetson-based real-time deep learning inference for assistive robotic vision,” J. Real-Time Image Process., vol. 18, no. 5, pp. 1597–1610, 2021.

[34] Y. Zhao, S. Szpiro, J. Knighten, and S. Feiner, “CheApR: Open-source augmented reality smart glasses,” in Proc. ACM CHI Conf. Human Factors Comput. Syst., San Jose, CA, USA, 2019, pp. 1–13.

Appendix A

System Source Code

1. Install essential libraries

```
sudo apt-get -y install python3 python3-pip python3-dev git cmake build-essential libopencv-dev python3-opencv libatlas-base-dev gfortran libjpeg-dev libpng-dev libtiff-dev libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev libx264-dev libgtk-3-dev libcanberra-gtk-module libcanberra-gtk3-module python3-numpy python3-scipy portaudio19-dev python3-pyaudio espeak libespeak-dev festival festvox-kallpc16k pulseaudio alsa-utils
```

2. Install Python dependencies

```
pip3 install opencv-python numpy pyttsx3 gTTS playsound pyaudio SpeechRecognition psutil
```

3. Project directory setup

```
mkdir -p smart_glasses/{audio,config,detection,input,logs,models,scene_understanding,tests,tracking,utils}
```

4. YOLO model

```
cd models/

wget https://github.com/dog-qiuqiu/Yolo-Fastest/raw/master/ModelZoo/yolo-fastest-1.1.bin

wget https://github.com/dog-qiuqiu/Yolo-Fastest/raw/master/ModelZoo/yolo-fastest-1.1.param

cd ..
```

5. Verifying model files

```
ls -lh models/

-rw-r--r-- 1 user user 1.5M yolo-fastest-1.1.bin

-rw-r--r-- 1 user user  18K yolo-fastest-1.1.param
```

6. Creating the main settings file

```
from pathlib import Path
from enum import Enum
class SystemMode(Enum):
    """Operating modes"""
    NORMAL = "normal"      # Standard announcements
    QUIET = "quiet"        # Only critical announcements
    VERBOSE = "verbose"    # All announcements
    SILENT = "silent"      # No announcements
class Settings:
    """System configuration settings"""
    BASE_DIR = Path(__file__).parent.parent
    MODELS_DIR = BASE_DIR / "models"
    LOGS_DIR = BASE_DIR / "logs"
    CACHE_DIR = BASE_DIR / "cache"
    CONFIG_FILE = BASE_DIR / "user_config.json"
   MODEL_PARAM_PATH = MODELS_DIR / "yolo-fastest-1.1.param"
    MODEL_BIN_PATH = MODELS_DIR / "yolo-fastest-1.1.bin"
    MODEL_INPUT_SIZE = 320
    MODEL_CONF_THRESHOLD = 0.4
    MODEL_NUM_THREADS = 4
    MODEL_USE_VULKAN = True  # GPU acceleration
```

7. Creating Priority Rules

```
from enum import Enum
class Priority(Enum):
    """Priority levels for objects"""
    CRITICAL = 4  # Immediate danger
    HIGH = 3      # Important obstacle
    MEDIUM = 2    # Moderate importance
    LOW = 1       # Background information
class ObjectPriorityRules:
    """
    Defines priority rules for different object types
    Determines which objects should be announced first
    """
    # Base priority for each object class
    PRIORITY_MAP = {
        # CRITICAL priority - Immediate dangers
        "car": Priority.CRITICAL,
        "truck": Priority.CRITICAL,
        "bus": Priority.CRITICAL,
        "motorcycle": Priority.CRITICAL,
        "bicycle": Priority.CRITICAL,
```

```
# HIGH priority - Important obstacles
        "person": Priority.HIGH,
        "dog": Priority.HIGH,
        "cat": Priority.HIGH,
        "stairs": Priority.HIGH,
        # MEDIUM priority - Moderate obstacles
        "chair": Priority.MEDIUM,
        "bench": Priority.MEDIUM,
        "potted plant": Priority.MEDIUM,
        "fire hydrant": Priority.MEDIUM,
        "stop sign": Priority.MEDIUM,
        "parking meter": Priority.MEDIUM,
        # LOW priority - Background objects
        "book": Priority.LOW,
        "bottle": Priority.LOW,
        "cup": Priority.LOW,
        "umbrella": Priority.LOW,
    }
    # Objects that should be tracked for movement
    MOVEMENT_TRACKING = {
        "car", "truck", "bus", "motorcycle", "bicycle", "person", "dog"
    }
    @staticmethod
    def get_priority(object_class: str) -> Priority:
        """Get priority level for an object class"""
        return ObjectPriorityRules.PRIORITY_MAP.get(
            object_class,
```

```
)
    @staticmethod
    def should_track_movement(object_class: str) -> bool:
        """Check if object should be tracked for movement"""
        return object_class in ObjectPriorityRules.MOVEMENT_TRACKING
    @staticmethod
    def adjust_priority_by_distance(base_priority: Priority, distance: float) -> Priority:
        """
        Adjust priority based on distance
        Closer objects get higher priority
        Args:
            base_priority: Base priority of object
            distance: Distance in meters
        Returns:
            Adjusted priority level
        """
        if distance < 2.0:  # Very close - boost priority
            if base_priority == Priority.HIGH:
                return Priority.CRITICAL
            elif base_priority == Priority.MEDIUM:
                return Priority.HIGH
        elif distance > 10.0:  # Far away - reduce priority
            if base_priority == Priority.HIGH:
                return Priority.MEDIUM
            elif base_priority == Priority.MEDIUM:
                return Priority.LOW
```

8. Creating the YOLO interface

```
import cv2, numpy as np
from typing import List, Tuple, Optional
class YoloFastest:
    """
    YOLO-Fastest detector wrapper
    Handles object detection using NCNN framework
    """
    def __init__(self, param_path: str, bin_path: str, target_size:
    int = 320, num_threads: int = 4, use_vulkan: bool = True):
    """
    Initialize YOLO-Fastest detector
    Args: param_path: Path to .param file
    bin_path: Path to .bin file
    target_size: Input size for model (default 320)
    num_threads: Number of CPU threads
    use_vulkan: Use Vulkan GPU acceleration
    """
    try: import ncnn
    self.net = ncnn.Net()
    self.net.opt.use_vulkan_compute = use_vulkan
    self.net.opt.num_threads = num_threads
    # Load model
    self.net.load_param(param_path)
    self.net.load_model(bin_path)
```

```
self.target_size = target_size self.mean_vals = [0, 0, 0]

self.norm_vals = [1/255.0, 1/255.0, 1/255.0]

# COCO class names

self.class_names = [ "person", "bicycle", "car", "motorcycle", "airplane", "bus", "train",
"truck", "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", "bird",
"cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", "backpack",
"umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite",
"baseball bat", "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle", "wine
glass", "cup", "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich", "orange",
"broccoli", "carrot", "hot dog", "pizza", "donut", "cake", "chair", "couch", "potted plant",
"bed", "dining table", "toilet", "tv", "laptop", "mouse", "remote", "keyboard", "cell phone",
"microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase", "scissors",
"teddy bear", "hair drier", "toothbrush" ] self.net_loaded = True

print("YOLO-Fastest model loaded successfully")

except Exception as e:

        print(f"Error loading model: {e}")

        self.net_loaded = False


def detect(self, frame: np.ndarray, conf_threshold: float = 0.4) -> List[dict]:

         """

        Detect objects in frame

        Args:

                frame: Input image (BGR format)

                conf_threshold: Confidence threshold

        Returns: List of detection dictionaries with keys:

                - class_name: Object class

                - confidence: Detection confidence

                - bbox: Bounding box [x1, y1, x2, y2]
```

```
# Get image dimensions
img_h, img_w = frame.shape[:2]
# Prepare input
mat_in = ncnn.Mat.from_pixels_resize( frame, ncnn.Mat.PixelType.PIXEL_BGR,
img_w, img_h, self.target_size, self.target_size )
mat_in.substract_mean_normalize(self.mean_vals, self.norm_vals)
# Run inference
ex = self.net.create_extractor()
ex.input("data", mat_in) ret, mat_out = ex.extract("output")
# Parse detections
detections = []
for i in range(mat_out.h):
        values = mat_out.row(i)
        # Parse detection: [class_id, confidence, x, y, w, h]
        class_id = int(values[0]) confidence = values[1]
        if confidence < conf_threshold:
        continue
        # Convert to pixel coordinates
        x_center = values * img_w
        y_center = values[3] * img_h
        width = values[4] * img_w
        height = values[5] * img_h
        x1 = int(x_center - width / 2)
        y1 = int(y_center - height / 2)
        x2 = int(x_center + width / 2)
        y2 = int(y_center + height / 2)
```

9. Creating the camera mod

```
import cv2
import time
from typing import Optional, Tuple
import numpy as np
class CameraManager:
    """
    Manages camera capture with error handling and reconnection
    """
    def __init__(self, camera_id=0, width=640, height=480, fps=30):
        """
        Args: camera_id: Camera device ID (0 for default, or video file path), width: Frame width, height: Frame height, fps: Target frame rate
        """
        self.camera_id = camera_id
        self.width = width
        self.height = height
        self.fps = fps
        self.capture = None
    def start(self) -> bool:
        try:
            self.capture = cv2.VideoCapture(self.camera_id)
            if not self.capture.isOpened():
                print(f"Error: Could not open camera {self.camera_id}")
                return False
            self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, self.width)
            self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, self.height)
            self.capture.set(cv2.CAP_PROP_FPS, self.fps)
            print(f"Camera {self.camera_id} started successfully")
            print(f"Resolution: {self.width}x{self.height} @ {self.fps} FPS")
            return True
        except Exception as e:
            print(f"Error starting camera: {e}")
            return False
```

```
def stop(self):
"""Stop camera capture and release resources"""
        if self.capture: self.capture.release()
        print("Camera stopped.")
        def read_frame(self, timeout=1.0) -> Optional[np.ndarray]:
        """ Read latest frame from buffer
        Args: timeout: Max seconds to wait for frame
        Returns:
                numpy array: Frame (BGR format) or None if failed
        """
        if not self.capture or not self.capture.isOpened():
                return None
        try:
                ret, frame = self.capture.read()
                if not ret or frame is None:
                        print("Error: Failed to read frame from camera")
                        return None
                return frame
        except Exception as e:
                print(f"Error reading frame: {e}")
                return None
        def get_camera_info(self) -> dict:
        """ Get current camera information Returns: dict: Camera properties """
        if not self.capture or not self.capture.isOpened():
                return {}
        return { 'width': int(self.capture.get(cv2.CAP_PROP_FRAME_WIDTH)), 'height':
        int(self.capture.get(cv2.CAP_PROP_FRAME_HEIGHT)), 'fps':
```

```
def reconnect(self, max_attempts=3, delay=2.0) -> bool:
"""
Attempt to reconnect to camera
Args:
        max_attempts: Maximum reconnection attempts
        delay: Delay between attempts (seconds)
Returns:
        bool: True if reconnection successful
"""
        print(f"Attempting to reconnect to camera {self.camera_id}...")
        self.stop()
        for attempt in range(max_attempts):
        print(f"Reconnection attempt {attempt + 1}/{max_attempts}")
        time.sleep(delay)
        if self.start():
                print("Reconnection successful!")
                return True
        print("Failed to reconnect to camera")
```

Appendix B

Hardware Assembly Guide

1. Tools Required

Table B.1 lists the tools required for assembly. The anti-static wrist strap is strongly recommended to prevent electrostatic discharge (ESD) damage to the Raspberry Pi.

Tool/Item	Purpose
Small Phillips-head screwdriver	Securing any bracket screws on the frame
Precision flat-head screwdriver	Prying open CSI cable connectors on the Pi board
Anti-static wrist strap	ESD protection when handling the Raspberry Pi
Isopropyl alcohol (70%) and lint-free cloth	Cleaning the frame surface before adhesive mounting
Cable zip ties or Velcro straps	Routing and securing ribbon cable along the frame
Adhesive foam tape (double-sided, 2 mm)	Mounting Pi board or power bank to the frame structure
Multi-meter (optional)	Verifying power rail voltages before first power-on
Micro-HDMI to HDMI cable	Initial Pi configuration via an external monitor
USB keyboard and mouse	Initial software setup and configuration

Table B.1: Tools Required for Assembly

2. Safety Precautions

   Observe the following precautions the assembly process to prevent hardware damage and ensure personal safety

   WARNING: Always power off and disconnect all power sources before making or modifying any hardware connections. Connecting or disconnecting components under power can permanently damage the Raspberry Pi or peripheral devices.

- Wear an anti-static wrist strap connected to an earthed surface when handling the Raspberry Pi board, camera module, or ribbon cable.
- Do not touch the gold connector pins on the CSI-2 ribbon cable. Oils from skin can cause poor electrical contact or corrosion.
- Do not apply excessive force when inserting or removing the CSI-2 ribbon cable connector. The locking mechanism is fragile.
- Ensure the power bank voltage output does not exceed 5.1 V. The Raspberry Pi 5 requires a stable 5 V / 3 A supply via USB-C.
- Keep the assembled device away from moisture. The Raspberry Pi 5 is not water-resistant.
- Do not bend the CSI-2 ribbon cable sharply. A minimum bend radius of 10 mm must be maintained to avoid conductor damage.

3. Component Verification

   Before beginning assembly, verify that all components are present and functional. This phase prevents assembly interruptions caused by missing or defective parts.

   a. Remove all components from their packaging and lay them on a clean, anti-static surface. Cross-reference each item against Table B.1.

b. Visually inspect the Raspberry Pi 5 board for physical damage: cracked components, bent GPIO header pins, or discoloration on the PCB. If any defects are found, do not proceed — return the board to the supplier.

c. Inspect the Camera Module 3. Confirm the lens is undamaged and the gold ribbon cable connector is clean and free of debris.

d. Confirm the power bank charges correctly by connecting it to a mains charger and verifying the LED charge indicator activates.

e. Connect the power bank to the Raspberry Pi 5 via the USB-C cable. Confirm the green power LED on the Pi board illuminates. Disconnect immediately after confirmation — no software is installed at this stage.

f. Test the bone conduction headphones by connecting them to a smartphone or computer and confirming audio output. Disconnect after testing.

4. Frame Preparation

The glasses frame provides the structural platform for the entire assembly. Careful preparation of the frame ensures secure, comfortable, and stable mounting of all components.

a. Clean the outer surface of the glasses frame with isopropyl alcohol and a lint-free cloth. Allow to dry fully (approximately 2 minutes) before applying any adhesive.

b. Identify the mounting locations for the Raspberry Pi 5, the camera module, and the power bank. Recommended positioning is as follows:

c. Camera module: centred on the bridge of the glasses frame, aligned with the wearer's forward line of sight, tilted 10–15 degrees downward.
d. Raspberry Pi 5: rear temple arm (right side), oriented with the CSI-2 port facing toward the camera.
e. Power bank: rear of the frame (temple arms or a dedicated rear strap), balanced across both sides to distribute weight.
f. If using a 3D-printed camera mounting bracket, confirm it fits the bridge of the chosen frame. Lightly sand any rough edges. Apply double-sided adhesive foam tape to the underside of the bracket before attachment.
g. Mark cable routing paths on the frame with a soft pencil. The CSI-2 ribbon cable should follow the upper edge of the right temple arm from the bridge to the Pi board position. The route must avoid sharp bends.
h. Do not attach any components permanently at this stage. All mounting should first be tested with temporary fixings (Velcro straps) before finalizing with adhesive.

5. Camera Module Installation

The camera module is the most sensitive component in the assembly. Its alignment directly affects detection performance. Handle with care throughout this phase.

a. Attach the camera mounting bracket to the planned position on the glasses bridge using double-sided adhesive foam tape. Press firmly for 30 seconds to ensure full adhesion.

b. Verify the bracket is level (horizontal) by placing the glasses on a flat surface and visually checking alignment. A misaligned camera produces skewed detections and degraded spatial analysis accuracy.
c. Attach the Camera Module 3 to the mounting bracket using the bracket's retaining clip or screw mechanism. Do not overtighten — the camera housing is plastic and can crack under excessive force.
d. Confirm the camera lens is pointing forward and angled 10–15 degrees downward from horizontal. This tilt angle corresponds to natural walking gaze direction and ensures ground-level obstacles fall within the camera's field of view.
e. Carefully lay the extended 300 mm CSI-2 ribbon cable along the marked cable route from the camera module to the planned Raspberry Pi mounting position. Do not connect either end at this stage.
f. Secure the ribbon cable to the frame at three points using Velcro cable ties: at the camera end, at the midpoint of the temple arm, and 30 mm before the Pi board position. Ensure no tight bends are introduced.
g. Trim any excess cable length with scissors if the cable cannot be routed without sharp bends. Leave at least 40 mm of slack at the Pi board end to allow connection to the CSI-2 port without strain.

6. Raspberry Pi 5 Mounting

   The Raspberry Pi 5 board must be mounted securely to prevent vibration-induced connector loosening during use. This phase also involves making the CSI-2 camera connection.

   a. Apply two strips of double-sided adhesive foam tape to the underside of the Raspberry Pi 5 board, positioned to avoid covering the USB, GPIO, or cooling vents.
   b. Position the Pi board on the right temple arm at the marked location. Orient the board so that:
      i. The CSI-2 connector faces toward the front of the glasses (toward the camera module).
      ii. The USB-C power port is accessible from the rear or underside of the temple arm.
      iii. The USB ports face downward or outward for headphone connection.
   c. Press the board firmly against the temple arm for 30 seconds to activate the adhesive. Reinforce with two Velcro straps looped around the temple arm and the board body.
   d. Confirm the board is rigid and does not shift when light lateral pressure is applied.

7. Connecting the CSI 2 Camera Cable

   CAUTION: The CSI-2 connector lock on the Raspberry Pi 5 is a pull-up latch mechanism. Lift the latch gently — it should move 1–2 mm upward. Forcing it further will break the connector.

   a. Locate the CSI-2 port on the Raspberry Pi 5 board. The Pi 5 has two CSI-2 ports (labelled CAM0 and CAM1). Use the CAM0 port for this project.
   b. Using a precision flat-head screwdriver or a fingernail, gently lift the black locking latch on the CSI-2 connector approximately 1–2 mm upward until it disengages.
   c. Insert the free end of the CSI-2 ribbon cable into the connector slot with the silver contacts facing toward the board (downward). Ensure the cable is fully inserted with no gap between the cable end and the connector body.
   d. Press the locking latch back down firmly until it clicks into place. Gently tug the ribbon cable with light force (< 0.5 N) to confirm it is secured.
   e. Perform the same connection procedure at the camera module end if not already connected. The camera's CSI-2 connector operates identically.

8. Power System Connection

   The power bank provides portable operation. Correct cable routing ensures the connection is secure and does not restrict the wearer's movement.

   a. Attach the power bank to the rear of the glasses frame assembly using Velcro straps. Balance weight equally across the left and right temple arms if a crosspiece or headband attachment is available. If mounting to a single arm, place on the left arm to counterbalance the Raspberry Pi on the right.

b. Route the USB-C power cable from the power bank to the USB-C power port on the Raspberry Pi 5. Secure the cable along the inner edge of the temple arm with Velcro ties. Allow 20 mm of cable slack near the Pi board to avoid mechanical stress on the port.

c. Do not connect the USB-C cable to the Raspberry Pi 5 at this stage. The OS must be installed and configured before first power-on.

9. Audio System Integration

The audio output system delivers text-to-speech announcements to the user. Bone conduction headphones are preferred as they leave the ear canal unobstructed, maintaining awareness of environmental sounds.

a. Route the headphone cable from the audio output (USB or 3.5 mm jack on the Raspberry Pi 5) along the glasses frame. Use Velcro ties to secure the cable against the temple arm, keeping it away from the CSI-2 ribbon cable to reduce electromagnetic interference.

b. If using bone conduction headphones with a USB connection, plug into any available USB port on the Raspberry Pi 5. If using a 3.5 mm wired connection, connect to the 3.5 mm audio jack on the Pi board.

c. Position the bone conduction transducer pads so they rest against the mastoid bone (just in front of the ear, on the cheekbone) when the glasses are worn. Adjust the headphone band tension so the pads maintain firm contact without causing discomfort.

d. Secure any excess headphone cable along the lower edge of the temple arm with a Velcro tie, leaving sufficient slack for the wearer to turn their head freely.

10. First Power-On and Hardware Verification

Before installing software, verify that all hardware connections function correctly. This phase is performed using a temporary monitor, keyboard, and mouse connected to the Raspberry Pi 5.

a. Connect a micro-HDMI cable from the Raspberry Pi 5 HDMI port to an external monitor. Connect a USB keyboard and mouse.

b. 36. Insert a microSD card pre-flashed with Raspberry Pi OS (64-bit) into the Pi's microSD slot.

c. Connect the USB-C power cable from the power bank to the Raspberry Pi 5 power port. The green power LED should illuminate immediately and the Pi should begin booting.

d. Allow the system to complete its initial boot sequence (approximately 45–60 seconds). Confirm the Raspberry Pi OS desktop or terminal prompt appears on the monitor.

e. Open a terminal and execute the following command to verify the camera module is recognised by the system:

   $ libcamera-hello --list-cameras

   The output should display the Camera Module 3 (Sony IMX708) with its available resolution modes. If no camera is listed, re-examine the CSI-2 ribbon cable connections at both ends before proceeding.

f. Execute a brief camera test to confirm the image pipeline functions:

   $ libcamera -still -o test.jpg

Inspect the resulting image (test.jpg) to confirm it is sharp, correctly oriented, and free of artefacts. Adjust the camera tilt angle on the mounting bracket if the image is not centred on the expected field of view.

g. Test audio output by playing a brief sound through the connected headphones:

$ speaker -test -t wav -c 2

Confirm audible output is present from the bone conduction headphones. If no sound is produced, verify the audio device is set as default using the raspi-config audio settings menu.

11. Assembles System

Upon successful completion of all six assembly phases, the assembled wearable smart glasses system should exhibit the following characteristics:

- Total assembled weigh below 200g, including the power bank.
- Camera module aligned at the bridge of the frame, tilted 10-15 degrees downward, providing a clear forward field of view.
- Raspberry Pi 5 securely mounted on the right temple arm with all ports accessible.
- CSI-2 ribbon cable routed without sharp bends and secured at three points.
- Power bank mounted at the rear, with USB-C cable routed cleanly to the Pi board.
- Bone conduction headphones connected and positioned for effective audio transmission.
- No exposed bare conductors or loose connections along any cable path.

The assembled system is now ready for software installation and configuration as documented in Appendix C. The hardware assembly should be re-inspected after the first hour of use to confirm all adhesive bonds and cable fixings remain secure under operational conditions.

Appendix C

Testing Protocols

1. Object Detection Test

   Aim: To verify the ability of the system to detect and identify common objects in real time

   Materials used

   a. Smart glasses prototype

   b. Test objects (bottle, chair, phone, cup, laptop, person)

   Procedure

   a. Power on the smart glasses system

   b. Connect the webcam and earphones to the Raspberry Pi 5

   c. Launch the object detection program

   d. Place selected objects at different distances from the camera

   e. Observe whether the object is correctly identified

   f. Listen to the generated audio feedback

   g. Record the detection accuracy and response time

   Expected Result

   The system should correctly identify objects and provide accurate audio descriptions within a few seconds.

2. Distance Detection Test

   Aim: To determine the system's ability to estimate the distance of nearby obstacles.

   Procedure

   a. Place an object at distances of 0,5m, 1m, 2m, and 3m

   b. Run the detection system

c. Move gradually toward the object

d. Observe the warning generated by the system

e. Compare measured distances with system output

Expected Result

The system should provide obstacle alerts as the user approaches nearby objects

3. Audio Output Test

Aim: To test the clarity and responsiveness of the audio feedback system.

Procedure

a. Connect the earphones to the Raspberry Pi 5

b. Run the text-to-speech module

c. Trigger object detection module

d. Listen to the spoken output

e. Evaluate speech clarity and delay

Expected Result

Audio output should be audible, clear and understandable with minimal delay.

4. Low-Light Environment Test

Aim: To evaluate system performance under poor lighting conditions

Procedure

a. Dim the room lighting

b. Place several objects within camera view

c. Run the object detection system

d. Observe detection accuracy under low illumination

Expected Result

The system should still detect major objects, although detection accuracy may reduce slightly.

5. Real-Time Navigation Test

Aim: To test the usability of the smart glasses during movement

Procedure

a. Wear the smart glasses prototype
b. Walk through a controlled indoor environment containing obstacles
c. Allow the system to detect surrounding objects
d. Observe the response speed and obstacle warnings
e. Record system performance during movement

Expected Result

The system should assist the user in identifying nearby obstacles while moving.

6. Power Consumption Test

Aim: To determine the operating duration of the portable power supply

Procedure

a. Fully charge the power bank
b. Connect it to the Raspberry Pi 5 system
c. Run the smart glasses continuously
d. Record the operating time before shutdown

Expected Result

The system should operate continuously for several hours without interruption

7. System Stability Test

   Aim: To evaluate the reliability of the complete system during prolonged operation

   Procedure

   a. Run the smart glasses continuously for an extended period.
   b. Observe system temperature and processing speed.
   c. Monitor for crashes or delayed responses.
   d. Record observations.

      Expected Result

      The system should operate stably without overheating or unexpected shutdown

The testing procedures confirmed the functionality and reliability of the proposed smart glasses system under different operational scenarios. The system successfully performed object detection, audio feedback generation, and real-time assistance for visually impaired users.

Appendix D

Component Datasheets

Raspberry Pi 5 Technical Specifications

1. Processor
    a. Broadcom BCM2712: 2.4GHz quad-core 64-bit Arm Cortex-A76 CPU, with Cryptographic extension, 512KB per core L2 caches, and a 2MB shared L3 cache
2. GPU & Display
    a. VideoCore VII GPU running at 800MHz, supporting OpenGL ES 3.1 and Vulkan 1.2. Dual 4Kp60 HDMI output with HDR support, and a 4Kp60 HEVC decoder.
3. Memory (RAM)
    a. LPDDR4X-4267 SDRAM: available in 1GB, 2GB, 4GB and 16GB options
4. Wireless Connectivity
    a. Dual-band 802.11ac Wi-Fi and Bluetooth 5.0/BLE
5. Wired Connectivity
    a. Gigabit Ethernet with PoE+ support (requires a separate PoE + HAT)
6. USB
    a. 2x USB 3.0 ports supporting simultaneous 5Gbps operation, and 2x USB 2.0 ports
7. Storage
    a. MicroSD card slot with support for high-speed SDR104 mode.
8. Expansion & I/O
    a. 2x 4-lane MIPI camera/display transceivers (1.5Gbps each)
    b. PCIe 2.0 x1 interface (requires separate M.2 HAT or adapter)
    c. Raspberry Pi standard 40-pin GPIO header
    d. Real-time clock (RTC), powered from external battery

e. Onboard power button

9. Power
    a. 5V/5A DC via USB-C with Power Delivery support
10. Reliability & Operating Conditions
    a. Operating temperature 0◦C to 70◦C
    b. MTBF (Ground Benign): 93,800 hours
    c. Production lifetime: guaranteed until at least January 2036
11. Board Dimension
    a. 85mm x 58mm PCB (Standard Raspberry Pi form factor)

Microsoft LifeCam HD-3000 technical specifications

1. Physical Dimensions

Parameter	Metric
Length	39.3mm
Width	44.5mm
Height/Depth	109mm
Weight	89.9g
Cable Length	1500mm

2. Imaging Specifications
    a. Sensor: CMOS
    b. Video Resolution:1280 × 720 pixels (720p HD)
    c. Still Image Resolution:1280 × 800 pixels (~1 megapixel)

d. Frame Rate: Up to 30 fps

e. Field of View: 68.5° diagonal

f. Focus: Fixed focus, 0.3m to 1.5m

g. Color Depth: 24-bit

h. Aspect Ratio: 16:9 widescreen

i. Digital Features: Digital pan, digital tilt, vertical tilt, swivel pan, 4× digital zoom

j. Color Technology: TrueColor — automatic image adjustment with manual override

3. Audio

    a. Microphone: Integrated omnidirectional microphone

    b. Frequency Range: 200Hz – 20kHz

4. Connectivity

    a. Interface: High-speed USB, compatible with the USB 2.0 specification.

5. Operating Conditions

    a. Operating Temp/Humidity: 0°C to 40°C, 5%–80% RH (non-condensing)

    b. Storage Temp/Humidity: −40°C to 60°C, 5%–65% RH (non-condensing)

6. System Requirements (for 720p HD recording)

    a. Intel Dual Core 3.0GHz or higher

    b. 2GB RAM minimum

    c. 1.5GB hard drive space

    d. USB 2.0 required

7. Certifications & Compilance

    a. FCC Part 15 compliant (Model No. 1492)

    b. CE Declaration of Conformity (EU)

c. RoHS / REACH compliant

d. ISO 9001 & ISO 14001 qualified manufacturer

e. Skype for Business Certified

f. Warranty: 3 years

g. Country of Manufacture: China

Appendix E

Budget Breakdown

Bill of Engineering Measurement and Evaluation

Table E.1 Bill of Engineering Measurement and Evaluation

S/N	Item Description	Specification	Qty	Unit Price	Total Cost
1	Raspberry Pi 5	8GB RAM Single Board Computer	1	₦280,000	₦280,000
2	Microsoft LifeCam HD-3000 Webcam	USB HD Camera	1	₦60,000	₦60,000
3	MicroSD Card	128GB High Speed Memory Card	1	₦18,000	₦18,000
4	Raspberry Pi Power Supply	5V/5A USB-C Adapter	1	₦20,000	₦20,000
5	Earphones/Audio Output Device	Wired Stereo Earphones	1	₦2,000	₦2,000
6	Power Bank	10,000 mAh Portable Power Supply	1	₦15,000	₦15,000
7	Jumper Wires and Connectors	Electronic Connection Accessories	1 Pack	₦5,000	₦5,000
8	USB Extension Cable	High Speed USB Cable	1	₦2,000	₦2,000
	Total				