GCU-EYE

Submitted by	**Ahtisham Tasawar** **Danish Mustafa Awan**
Roll. No	**0206-BSCS-21** **0065-BSCS-21**
Session	**2021 - 2025**
Supervised by	**Ma'am Qurra-Tul-Ann** **Lecturer**

BS(HONS)
IN
COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

Acknowledgements

In the name of Allah, the Most Merciful, the Most Compassionate.

All praise is due to Allah (SWT), the Lord of the worlds, whose boundless mercy, infinite wisdom, and unwavering guidance enabled us to complete this final year project. It is through His divine help that we found strength during challenges, clarity during confusion, and success at the end of this journey.

We would like to express our deep appreciation to our respected supervisor, Ma'am Qurra-Tul-Ann, for her continuous guidance, valuable insights, and unwavering support throughout the development of this project. Her encouragement, patience, and expert advice played a vital role in helping us transform our ideas into a functional and meaningful application.

We extend our heartfelt thanks to the Department of Computer Science, Government College University, for providing us with a conducive academic environment, essential resources, and mentorship that empowered us to explore and innovate. The skills, knowledge, and values we gained throughout our degree program have been instrumental in the successful completion of this work.

We are especially grateful to our parents, whose endless love, prayers, encouragement, and moral support have been the backbone of our academic journey. Their belief in our abilities gave us the motivation to overcome challenges and stay committed to our goals.

Lastly, we acknowledge all those — teachers, peers, and friends — who contributed in any way, directly or indirectly, to the success of this project. Your support and encouragement have been truly appreciated.

Dedication

This project is lovingly dedicated to the visually impaired students, whose resilience, courage, and determination in the face of daily challenges continue to inspire us. It is for them that this work was envisioned with the hope of enabling greater independence, safety, and dignity in navigating the academic world and beyond. We also dedicate this work to our beloved parents, whose unwavering support, sacrifices, and prayers have been the pillars of our academic and personal journey. Their constant encouragement gave us the strength to persevere through every obstacle. Lastly, we extend this dedication to our teachers and mentors, whose guidance, knowledge, and belief in our abilities have shaped our path and empowered us to turn our ideas into reality.

Abstract

Navigating a busy university campus without sight presents daily challenges for blind students—avoiding obstacles, locating classrooms, and moving independently. GCUeye is a mobile application developed for visually impaired students at GC University Lahore to address these challenges through intelligent, accessible technology.

The app turns any Android smartphone into a navigation assistant using a YOLOv8 object detection model hosted on a Flask server. By processing live camera input, it detects obstacles like stairs, people, and furniture, and provides real-time voice and vibration feedback. All interactions are gesture-based, eliminating the need for visual elements.

GCUeye also features a step-based VR indoor navigation system, allowing users to simulate movement between rooms on a graph-based map—completely offline and without GPS or Bluetooth beacons. This makes indoor routing possible in buildings where traditional systems fail.

Additional tools include a voice-controlled class reminder system, an SOS emergency function triggered by long press, and customizable settings for speech speed, pitch, and feedback preferences. Designed entirely around accessibility and independence, GCUeye combines AI, voice, and gesture to make campus life safer, smarter, and more inclusive for blind students.

Contents

List of Tables

List of Figures

Chapter 1

Requirement Specification

1.1 Project Overview

1.1.1 Purpose and Vision

Navigating a university campus is a routine task for most students—but for those who are blind or visually impaired, even simple activities like finding a classroom or walking to the library can be complex and overwhelming. The traditional reliance on human assistance or trial-and-error navigation not only reduces independence but also adds unnecessary cognitive and emotional stress to daily academic life.

GCUeye is a dedicated mobile application designed to address these challenges faced by blind students at Government College University (GCU), Lahore. The app leverages real-time object detection using YOLOv8, voice-guided feedback, haptic alerts, and voice command interfaces to help users confidently move around the university campus.

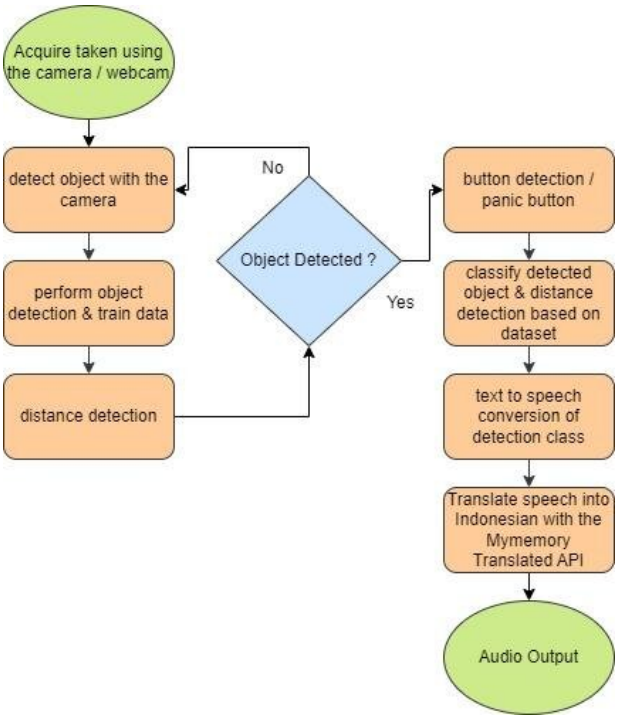


FIGURE 1.1: Blind student navigating campus using assistive mobile technology

The project envisions a context-aware assistive platform where users can:

- Use voice commands to input destinations.
- Receive obstacle alerts using audio and vibration.
- Be guided to university locations (e.g., classrooms, cafeterias, libraries) with step-by-step assistance.
- Get emergency help using a dedicated gesture-triggered SOS system.
- Manage schedules via voice reminders for classes or events.
- Navigate indoors using step-by-step simulated routes with audio instructions and gesture-based input, without relying on GPS.

This app is built with the specific needs of GCU's blind students in mind, ensuring that the interface, detection logic, and content are localized and adapted to the physical layout, academic structure, and safety requirements of the university environment.

1.1.2 System Architecture

The system architecture of GCUeye is modular and optimized for low-latency performance, on-device processing, and multimodal feedback. It is composed of three tightly integrated layers:

A. Perception Layer (Object Detection + Camera)

- Uses the device's camera to continuously capture live images every 1–2 seconds.
- Sends the frame to a locally hosted Flask server where the YOLOv8 model runs inference.
- Detects real-world objects (e.g., stairs, potholes, people, walls, chairs).
- Estimates object direction — left, center, or right — using bounding box coordinates.

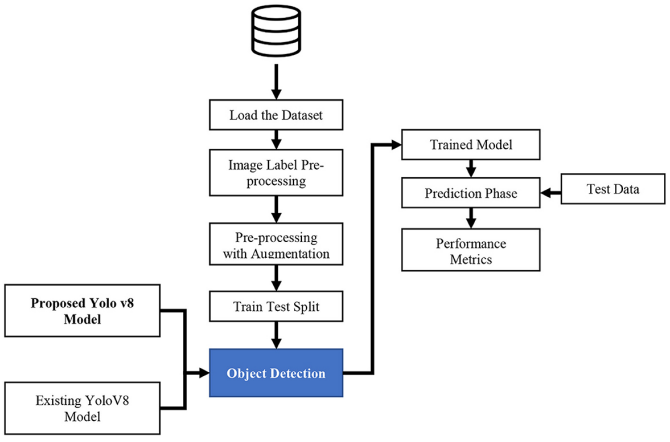


FIGURE 1.2: System architecture showing detection and feedback pipeline

B. Interaction Layer (Flutter Mobile App)

- Built in Flutter for high performance and beautiful UI across Android devices.
- Supports:

  - Gesture control: triple tap, long press, swipe navigation.
  - Text-to-speech audio feedback using FlutterTTS.
  - Haptic feedback for obstacle alerts using vibration APIs.
  - Voice-based reminders for class scheduling.
- Includes a custom homepage UI with animated navigation, icons, and swiping feature cards.

VR Navigation Layer (Graph-based Indoor Guidance)

- Allows users to select a source and destination using swipe gestures.
- Uses Dijkstra's algorithm to calculate the shortest route between locations on a graph-based map.
- Triggers step-by-step audio instructions based on user steps
- Alerts the user with special voice messages with custom instructions for each route he/she takes.

C. Intelligence + Communication Layer

- The Flask backend returns detected object data in structured JSON.
- The mobile app interprets this and decides:
  - What to speak.
  - How to vibrate.
  - When to alert the user.
- The system ensures safety and reliability even in areas with no internet coverage (e.g., campus basements or lecture halls) by use of graphs mapping that is totally offline.

1.1.3 Technical Innovation

Unlike existing apps that are general-purpose or cloud-reliant, GCUeye offers a highly customized, offline, and context-aware solution for blind students navigating a campus.

Area	**GCUeye Innovation**
Object Detection	YOLOv8 model trained on campus-specific obstacle and object classes
Offline and Online Capability	Object Detection runs on Flask server online and Indoor Navigation runs offline by graphs.
Campus Awareness	Custom location database (e.g., classrooms, library, cafeteria)
Haptic + Voice Feedback	Combined vibration + TTS for faster and more intuitive spatial alerts
Voice Navigation	Users can speak destination names, and the app guides them step-by-step
Emergency Contact System	Long-press gesture sends pre-written SMS + makes a phone call
Gesture-Based UI	Full navigation via gestures (no visual UI required)
Accessibility by Design	Uses large icons, high-contrast UI, semantic descriptions, and sound feedback
Indoor VR Navigation	Step-based audio guidance using graph traversal without GPS or Bluetooth

1.1.4 Application Framework

The GCUeye application follows a three-layered application framework:

1. **Input Interface**
    - Voice input for destinations and reminders.

- Camera input for object recognition.
- Gesture input for control (e.g., triple tap for detection, long press for SOS).

2. **Processing & Intelligence**
   - YOLOv8 processes camera frames on Flask server.
   - Bounding box data is converted to direction labels (left, center, right).
   - Voice command parser interprets instructions like "Take me to library."
   - Calculates indoor navigation paths using graph-based algorithms (Dijkstra).
3. **Output & Feedback**
   - Voice feedback: "Stairs ahead. Bag on the left."
   - Vibration feedback: Short buzz for left, double buzz for right.
   - Visual UI (for partially sighted or caregivers): swipeable cards, icons, neon theme.
   - Step-by-step spoken directions for indoor navigation (e.g., "Walk straight. Turn left after 3 steps.")

This layered structure ensures:

- Low-latency real-time processing.
- Scalability to add new objects, voice commands, or locations.
- Separation of concerns, making development and maintenance modular.

1.2 Problem Statement

Navigating large university campuses such as Government College University (GCU), Lahore is inherently challenging for students with visual impairments. University

environments are often dynamic, spatially complex, and not designed with inclusive mobility in mind. Blind students must rely on others for guidance, memorize spatial layouts, or risk accidents during solo movement — all of which hinder their academic independence and overall confidence.

GCUeye aims to solve this problem by creating a mobile-based real-time navigation and obstacle detection system that enhances mobility, safety, and autonomy for blind students within and around the GCU campus.

1.2.1 Real-World Challenges for Visually Impaired Students in Campus Environments

Despite policy-level awareness about accessibility, most university campuses remain difficult to navigate without sight. Visually impaired students at GCU face several persistent issues:

Spatial Disorientation

- Struggle to determine their location or direction.
- Cannot identify pathways vs. obstacles.
- Get lost while finding classrooms, offices, or restrooms.

Obstacle Hazards

- Chairs, bags, or desks placed in corridors.
- Staircases, potholes, or elevated platforms.
- People standing still or walking suddenly.
- Vehicles, trolleys, or scattered furniture.

Limited Accessibility in Conventional Navigation Tools

- Inaccurate GPS positioning indoors.
- No obstacle or object-level awareness.
- No haptic guidance or spatial voice feedback.
- Visually complex UI not suitable for screen readers or gesture navigation.

Dependence on Human Assistance

- Friends or classmates to guide them.
- Memorization of class locations and timings.
- Staff assistance for administrative navigation.

Lack of Emergency Communication

- Difficulty dialing a phone number or explaining a situation.
- No hands-free, gesture-based SOS system.

Lack of Accessible Scheduling and Reminder Tools

- Tracking class timings.
- Managing exam or assignment schedules.
- Receiving event notifications.

1.2.2 Limitations of Existing Approaches

While there are several accessibility apps and devices available globally, they are either:

- Too generic (not tailored to GCU or university use cases),
- Too expensive (require wearables or special sensors),
- Not offline (require constant internet),
- Or non-integrated (no all-in-one solution for detection + guidance + emergency + reminders).

Existing Tool	**Limitations**
Google Maps	Works poorly indoors, no object detection, no blind-specific interface.
SeeingAI / Lookout	Only labels objects, no spatial feedback, requires cloud connection.
BeMyEyes	Human volunteer-based, not real-time, not gesture-triggered.
Wearables (e.g., Smart Cane)	Expensive, single-function, uncomfortable for students.
Voice Assistants	Generic responses, no environmental understanding or object awareness.

1.2.3 Unique Challenges in Developing GCUeye

Real-Time Performance vs. Hardware Limitations

- Real-time inference is crucial, but mobile hardware has limited processing capability.
- Cloud introduces latency — local processing is required.

Offline Functionality

- Many parts of the GCU campus have poor or no internet.
- GCUeye has indoor navigation based on offline without requiring internet,gps, or bluetooth beacons.

Cognitive Load and Simplicity

- UI must avoid overwhelming the user.
- Gesture and voice interaction must be intuitive — no buttons, no menus, no reading.

Privacy and Safety

- Detection frames must not be stored or transmitted externally.
- Emergency systems must be secure and fast.

Natural Voice and Haptic Communication

- Feedback must be clear, fast, and non-confusing.
- Vibration must complement voice, not replace it.

Summary of the Problem

Blind students at GCU require a unified, intelligent, accessible, and offline mobile solution that helps them:

- Navigate both indoor and outdoor university spaces.
- Detect and avoid real-world objects and obstacles.
- Receive location-based and time-based reminders.
- Contact help in emergencies using intuitive, non-visual triggers.
- Regain academic independence, confidence, and mobility.

1.3 Literature Review

Over the past two decades, assistive technologies for the visually impaired have made significant strides, transitioning from simple mechanical tools to highly advanced AI-powered systems. Yet, despite the growing ecosystem of wearable devices, smart assistants, and computer vision-based tools, challenges remain in designing affordable, context-specific, and offline-capable systems for daily navigation and task management—especially within educational institutions. This review explores the research landscape across four major pillars that form the foundation of the **GCUeye** system: assistive navigation tools, YOLO-based object detection, multimodal feedback mechanisms, and gesture/voice-based interaction models for accessible mobile applications.

1.3.1 Assistive Navigation for the Visually Impaired

Traditional Approaches and Their Limitations

Visually impaired individuals have long relied on white canes, Braille signage, and guide dogs for navigation. While effective to a degree, these methods:

- Do not convey detailed spatial information.
- Cannot identify specific object types or contextual obstacles.
- Lack integration with digital environments or personalized schedules.

The white cane, first introduced in the 1920s, remains a foundational tool, but it's limited to detecting obstacles within one meter and only those at ground level. Similarly, guide dogs, while helpful, are costly (up to $50,000 per dog), require extensive training, and are not always feasible for all users.

Modern Technological Solutions

To overcome these limitations, researchers and developers have turned to digital tools that leverage computer vision, GPS, and sensor fusion. Some notable systems include:

- **NavCog (CMU, 2016–2019):** An indoor navigation app developed at Carnegie Mellon University using BLE (Bluetooth Low Energy) beacons installed throughout buildings to provide spoken directional cues to blind users [8]. While effective indoors, it relies heavily on pre-installed infrastructure, which limits scalability and deployment in new or public spaces.

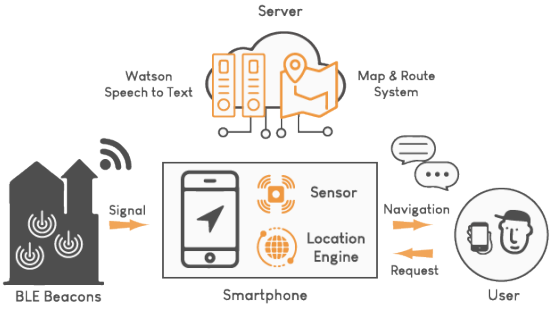


FIGURE 1.3: Blind users navigating using BLE Beacons with smartphone sensors

- **EyeMate:** A smart cane system combining GPS and infrared sensors to guide users in outdoor spaces. While it offers audio feedback, it fails in GPS-denied environments like classrooms or basements [13].

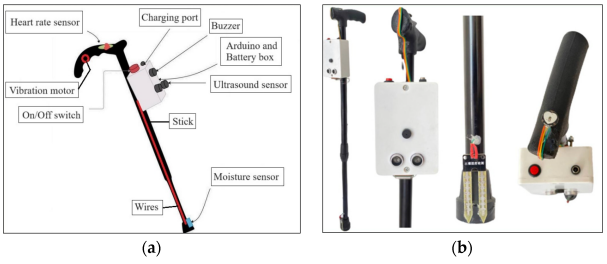


FIGURE 1.4: Blind users navigating using smart cane with GPS sensors

- **Microsoft Soundscape:** This innovative app used 3D spatialized audio to help blind users build mental maps of their surroundings. However, it required constant internet access and was discontinued in 2023 [3], leaving a gap in this domain.

FIGURE 1.5: Blind users navigating using 3D Spatial Audio with audio sensors

- **PathVu Navigation Maps:** Designed to help pedestrians with mobility challenges, PathVu creates high-resolution maps of sidewalks and crosswalks. However, it lacks real-time object detection or personal navigation.
- **Waymap (UK, 2023):** Offers audio-based GPS navigation for public transit, but like others, depends on infrastructure and data connectivity.

Academic Environments Are Still Underserved

Despite the advancements, few systems address the specific needs of blind university students, who navigate between lecture halls, crowded corridors, staircases, and open grounds. Academic campuses pose unique challenges:

- Dense, obstacle-rich environments.
- Dynamic placement of objects like chairs, bags, or lab equipment.
- Need for class schedule reminders and emergency support.

GCUeye addresses this research gap by delivering a campus-specific navigation solution that works offline, detects movable obstacles (e.g., chairs, bins, bags, people), and integrates class reminders and an emergency SOS system, all using a smartphone.

1.3.2 Object Detection Using YOLO

Overview of YOLO Technology

Object detection is central to empowering blind users with environmental awareness. The YOLO (You Only Look Once) framework revolutionized object detection by offering real-time performance without sacrificing accuracy.

- **YOLOv1 – v2:** Early versions introduced single-shot detection from entire images, making inference fast but with lower accuracy on small objects.
- **YOLOv3/v4:** Achieved significant improvements in accuracy and robustness using deeper architectures and better upsampling.
- **YOLOv5:** A PyTorch-based implementation by Ultralytics that added auto-learning anchors, simplified deployment, and custom dataset support.
- **YOLOv8 (2023):** Introduced new features like export to ONNX/CoreML, improved modularity, lighter weight models, and faster training — making it highly suitable for mobile and embedded applications [17].

YOLO in Assistive Applications

Several projects have employed YOLO for blind assistance, but each has critical limitations:

- **Togacar et al. (2020):** Deployed YOLOv3 on Raspberry Pi for blind pedestrian navigation. The system could detect road elements like cars and traffic lights but lacked voice output and was not designed for indoor use [16].
- **Lohani et al. (2021):** Developed a wearable belt equipped with YOLOv5-powered cameras. The system offered object detection but required cloud connectivity and lacked feedback mechanisms for direction or urgency [11].
- **Envision AI / Seeing AI (Microsoft):** These commercial apps use YOLO-like backends but are cloud-dependent, requiring constant internet and often offering delayed response — impractical for real-time, offline academic environments [12].

GCUeye improves on these by:

- Training YOLOv8 on a custom dataset of GCU campus objects.
- Running the detection server offline using Flask.
- Providing instant multimodal feedback on detected object types and relative directions.

1.3.3 Multimodal Feedback: Voice and Haptics

Need for Multisensory Output

Voice-based systems dominate assistive technology, but research suggests that combining voice with vibration enhances spatial understanding and cognitive load handling.

- **Wobbrock et al. (2009):** Found that blind users perform better when tactile feedback is used for notifications and navigation [18].
- **Kammoun et al. (2012):** Demonstrated that direction-based vibration patterns (e.g., left shoulder buzz = turn left) improved reaction time and confidence [9].
- **González-Mora et al. (2020):** Developed a haptic belt that accurately directed users using vibration pulses. While effective, it was expensive, uncomfortable for long use, and impractical for students [7].

Google Lookout App: Although it narrates visual scenes, it lacks haptic feedback entirely and often overwhelms the user with verbal information.

GCUeye uses built-in phone vibration to indicate obstacle direction:

- 1 buzz = obstacle on left
- 2 buzzes = obstacle on right

Voice speaks the object type (e.g., "Bin ahead on right.").
This feedback is non-intrusive, intuitive, and works effectively in noisy areas like campuses.

1.3.4 Gesture and Voice Interfaces in Mobile Accessibility

Challenges with Traditional GUI Interfaces

Visually impaired users cannot interact with standard UI elements like buttons, menus, or dropdowns. Hence, interaction paradigms must shift from visual to gesture and voice-based systems.

- **Kane et al. (2008):** Identified preferred gesture types among blind users—triple tap, long press, and swipes were found to be natural and easy to memorize [10].
- **Apple VoiceOver & Google TalkBack:** These tools allow screen reading and voice control, but they are often cumbersome, requiring users to remember multiple commands and gestures. Furthermore, they don't support real-time object detection or environment awareness.
- **Be My Eyes and Aira:** These apps connect users to human volunteers via video call for help. However, they are not suitable for continuous navigation or offline use.
- **Voice AI Assistants (e.g., Siri, Google Assistant):** Support basic commands like checking time or setting reminders but do not integrate with real-world vision or emergency systems.

GCUeye builds an accessible interaction layer:

- **Triple tap:** Starts or stops object detection
- **Long press:** Activates offline SOS mode

- **Swipe:** Navigates between app screens (e.g., Detection, Reminder, SOS)
- **Voice input:** Allows class schedule reminders

This ensures blind users can fully operate the app with just gestures and short voice commands — all offline.

1.3.5 Summary of GCUeye's Research Positioning

Research Pillar	**Gap in Existing Solutions**	**GCUeye's Contribution**
Campus Navigation	No real-time system tailored to educational spaces	Built specifically for GCU's blind students
Object Detection	Generic, cloud-reliant models	Custom YOLOv8, online via Flask
Multimodal Feedback	Over-reliance on voice, no haptic integration	Combines voice + vibration for situational clarity
Accessible Interaction	Button-based, visual navigation	Gesture-first, voice-enabled interface
Scheduling Integration	Apps lack academic reminders	TTS-powered class reminders with tactile confirmation
SOS Features	Most require network/internet	Works offline using preconfigured contacts + SMS/call

Table 1.1: GCUeye's Research Contribution vs. Existing Limitations

1.4 Project Objectives

The GCUeye project is guided by a combination of technical, functional, and social impact objectives. The goal is not just to develop a software tool, but to deliver

a practical assistive solution that transforms the way blind students navigate and experience life on campus.

1.4.1 Primary Research and Development Goals

The broader vision of GCUeye is to demonstrate that low-cost, on-device AI paired with accessible mobile design can solve a real-world problem: enabling blind students at GC University Lahore to independently, safely, and efficiently navigate their university environment.

Key Research Goals:

1. Explore real-time object detection (YOLOv8) in dynamic indoor environments such as classrooms and corridors.
2. Investigate the effectiveness of multimodal feedback (audio + haptics) for spatial guidance.
3. Validate gesture-based mobile UIs for visually impaired users in real academic settings.
4. Integrate obstacle detection, emergency support, and class scheduling into a unified, intuitive app with no visual interaction.
5. Contribute a domain-specific campus-focused solution to the field of accessible navigation.
6. Design a step-triggered indoor navigation system using swipe input and graph traversal without relying on GPS or beacons.

1.4.2 Technical Implementation Objectives

A. Machine Learning: YOLOv8-Based Object Detection

- Train a YOLOv8 model on 17+ campus-specific object classes.

- Use tools like Roboflow for annotation.
- Optimize for mobile speed using YOLOv8n or YOLOv8s.
- Apply transfer learning with pretrained weights.

B. Backend Server: Flask for Local Inference

- Build Flask API to accept camera images in base64.
- Return object labels, confidence, bounding box coordinates.
- Classify object direction (left/center/right).
- Ensure response time under 500ms.

C. Mobile App: Flutter Interface for the Blind

- Gesture-based UI with:
  - Triple-tap: detection
  - Long-press: SOS
  - Swipe: screen navigation
- Integrate FlutterTTS for voice output.
- Implement vibration patterns:
  - Left = 1 buzz
  - Center = 2 buzzes
  - Right = 3 buzzes
- Design swipeable card-style homepage with large icons and neon theme.

D. Voice Command System for Scheduling

- Capture voice input like: “Remind me for class at 9 a.m.”

- Parse time and content.
- Store in SQLite or SharedPreferences.
- Trigger alerts using TTS and vibration.

E. SOS System

- Allow contact setup.
- Long press sends SMS and places a call.
- Offline-compatible with fallback if SIM/balance is missing.

F. Accessibility and UI/UX Standards

- Use high-contrast design.
- Add semantic labels.
- Follow Material Design Accessibility Guidelines.
- Avoid menus or button-heavy navigation.

1.4.3 Measurable Success Criteria

Model Performance Metrics

Metric	**Target**
mAP on test set	$\geq 77.3\%$
Inference time per image	≤ 440 ms
Bounding box accuracy	$\geq 89\%$
Direction estimation accuracy	$\geq 94\%$

Mobile Usability Metrics

Gesture detection reliability	$\geq 98\%$
Audio clarity (user tested)	$\geq 90\%$
Vibration recognition rate	$\geq 95\%$
App crash rate	0 in 48-hour test
TTS latency	≤ 1 second
SOS response time	≤ 2 seconds

Accessibility and UX Metrics

User satisfaction (Likert)	≥ 4.5
Onboarding time	≤ 10 minutes
Independent navigation success	$\geq 85\%$
Reminder accuracy	$\leq$ 2s deviation
Offline operation	100% for core features

Academic Milestone Integration

Milestone	**Deliverable**
Midterm Evaluation	Flask + Flutter integration with object detection
Final Year Exhibition	Full app with detection, SOS, reminders, gesture UI
Thesis Submission	Architecture, design, metrics, test results
Viva	Live demo with blind user simulation

1.5 Target Audience

Designing a mobile application for accessibility requires a deep understanding of the day-to-day challenges, expectations, and habits of the intended users. This blind app is not built for the general public or even for blind users in open city environments — it is developed specifically for visually impaired students of Government College University (GCU) Lahore, whose challenges are rooted in academic routines, on-campus mobility, and campus-specific social and infrastructural dynamics.

1.5.1 Primary End Users: Blind and Visually Impaired Students at GCU

The primary users of this app are registered blind or low-vision students currently enrolled in undergraduate and postgraduate programs at GCU. These students navigate a complex, multi-building campus that includes:

- Academic blocks with lecture rooms and labs
- Libraries, reading rooms, and study lounges
- Administrative offices and corridors
- Open outdoor areas, cafeterias, hostels, and gardens

For these students, independent movement is limited due to:

- Lack of tactile signposting
- Absence of auditory campus mapping
- Inaccessibility of visual room labels, schedules, and notices

Mobility Needs:

- Safe, real-time navigation from one campus building to another
- Obstacle awareness (e.g., stairs, walls, bags, crowded areas)
- Directional understanding (e.g., object on left vs. front)

Academic Needs:

- Timely reminders for classes, labs, and university events
- Ability to input schedules without visual interfaces

- Flexible rescheduling of tasks without text-based menus

Safety Needs:

- A quick, hands-free method to contact trusted contacts or security in emergencies
- Reliable communication even without internet access or visual confirmation

Interface Needs:

- No reliance on screen reading or visual confirmation
- Tactile and auditory cues for every action
- Non-intrusive controls that work with minimal training

The blind app directly responds to these needs by offering:

- Real-time object detection using the smartphone camera and YOLOv8
- Audio and vibration feedback to indicate direction and type of obstacles
- Voice-controlled scheduling, including reminders for academic tasks
- Gesture-based navigation, eliminating the need for reading or typing
- An SOS system that can be triggered silently and instantly via a long press

By focusing on this specific student population, the app maintains a laser-sharp scope and delivers functionality that's practical, not theoretical.

1.5.2 Secondary Stakeholders

While the app is built primarily for students, there are secondary users and stakeholders who indirectly benefit from its features.

University Staff and Academic Coordinators:

- Receive fewer requests for navigation help
- Can promote inclusivity without investing in expensive infrastructure like Braille maps or beacon networks

Parents and Guardians:

- Gain peace of mind knowing their children can safely and independently move around campus
- Receive SOS alerts in real time when students are in distress

Accessibility Researchers and Developers:

- Can study the effectiveness of gesture-based mobile interaction
- Use the app's modular codebase as a foundation for broader accessible navigation systems

1.5.3 User Demographics and Device Assumptions

Attribute	**Assumption**
Age Group	18–30 (typical university students)
Experience	Low to moderate smartphone familiarity
Device Type	Android smartphones with rear cameras
Connectivity	May have limited or no internet access
Vision Profile	Blind or low vision (screen not usable)
Language Preference	English and/or Urdu voice support

The app is designed to function under typical device and campus constraints without the need for expensive hardware, subscriptions, or ongoing internet service.

1.5.4 Inclusion Philosophy

Unlike commercial products that aim for scale and broad appeal, this blind app is a customized, locally deployable accessibility tool. It is designed with:

- **Empathy-first engineering** — understanding the student experience before choosing tools
- **Minimal friction** — every gesture or command is mapped to a high-impact action
- **Autonomy over assistance** — users don't need to ask for help or wait for volunteers

By placing the GCU blind student at the center of the development process, the app offers not just functionality but also dignity, safety, and academic empowerment.

1.6 Use Case Specification

This section outlines the core functional use cases of the blind app. Each use case represents a specific interaction that visually impaired students at Government College University (GCU) can perform using the mobile application. These interactions are designed to be executed through non-visual controls such as gestures and voice commands, and include object detection, SOS messaging, scheduling reminders, and navigating between app features.

The use cases are written using a standard software engineering template and include detailed descriptions of the app's behavior under various scenarios.

UC01 – Start Object Detection (Triple Tap)

Use Case ID	UC01
Actor	Blind student
Description	The user initiates the object detection system by performing a triple tap gesture anywhere on the screen. The app begins capturing camera frames, sends them to the server, and provides real-time voice and vibration feedback about detected obstacles.
Preconditions	Camera permission granted. App is open and in foreground.
Postconditions	Detection loop is active. Feedback is provided for each detected object with spatial orientation.
Main Flow	1. User performs a triple tap. 2. App activates the camera and starts sending frames to the Flask server. 3. Server detects objects using YOLOv8. 4. App parses object labels and directions. 5. TTS announces: "Person in front, bag on the left." 6. Vibration is triggered (e.g., one buzz for left). 7. Loop continues every 1–2 seconds until stopped.
Alternate Flow	• If server does not respond, app announces: "Detection unavailable." • If no object is detected, TTS says: "Clear path."
Exceptions	• Camera permission denied → TTS error prompt. • Frame not captured → skip frame and retry silently.

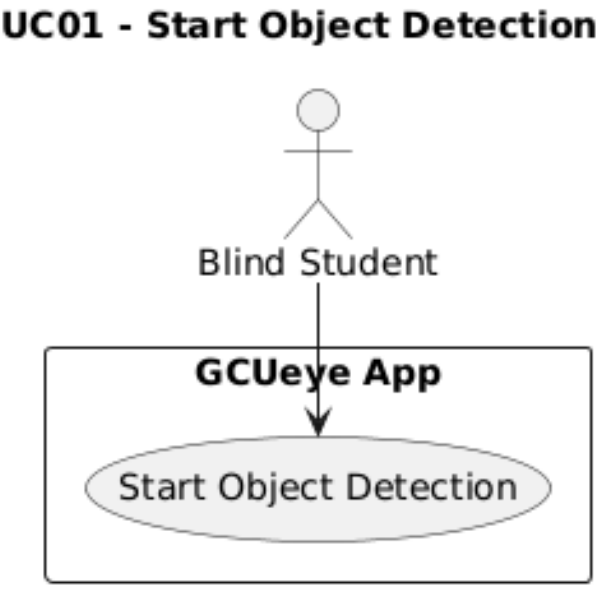


FIGURE 1.6: Use Case Diagram – Start Object Detection

UC02 – Stop Object Detection (Triple Tap Again)

Use Case ID	UC02
Actor	Blind student
Description	Detection mode is stopped when the user performs another triple tap gesture.
Preconditions	Detection loop is running.
Postconditions	Camera feed and detection requests stop. No further feedback is given.
Main Flow	1. User performs triple tap again. 2. App stops sending frames to server. 3. TTS says: "Detection stopped." 4. All background detection services terminate.

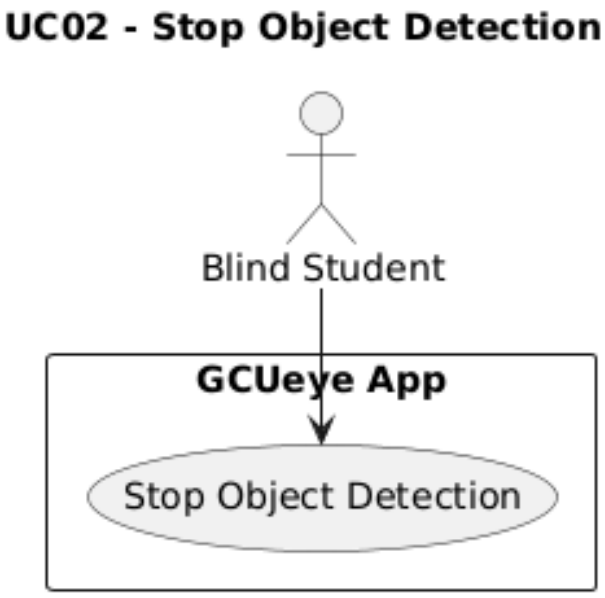


FIGURE 1.7: Use Case Diagram – Stop Object Detection

UC03 – Send Emergency SOS (Long Press)

Use Case ID	UC03
Actor	Blind student
Description	User sends an emergency SMS and places a call to a pre-configured contact by performing a long-press gesture anywhere on the screen.
Preconditions	Contact is saved in the app. Device has SIM/network service.
Postconditions	SMS is sent and phone call is placed to emergency contact.
Main Flow	1. User performs a long press. 2. App reads emergency contact. 3. App sends pre-configured SMS message. 4. App places call automatically. 5. TTS confirms: "Help message sent. Calling now."
Alternate Flow	If contact is not saved, app prompts: "Please add an emergency contact."
Exceptions	• SIM unavailable → TTS: "Unable to send message." • No credit → App tries call, fails silently, retries SMS if possible.

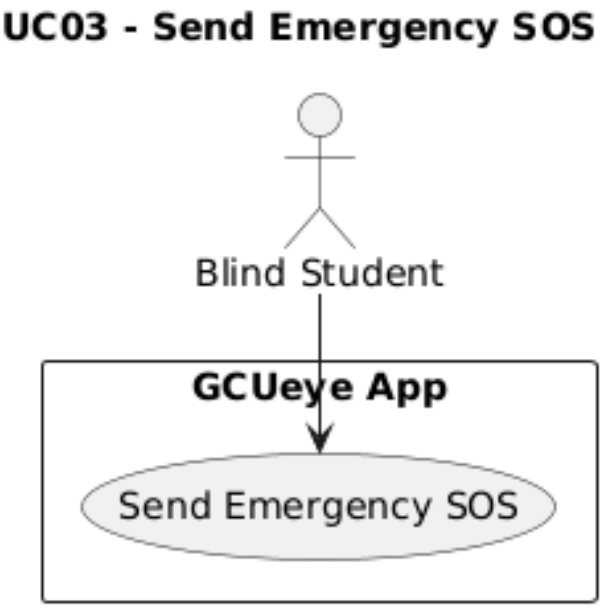


FIGURE 1.8: Use Case Diagram – Send Emergency SOS

UC04 – Set a Voice Reminder

Use Case ID	UC04
Actor	Blind student
Description	The user sets a class or task reminder using voice input (e.g., "Remind me for class at 10 a.m."). The app stores and schedules the reminder for TTS and vibration notification.
Preconditions	Microphone permission granted. Voice service enabled.
Postconditions	Reminder is stored locally and will trigger at scheduled time.
Main Flow	1. User taps the voice reminder icon or navigates to it via swipe. 2. TTS prompts: "Speak your reminder." 3. User says: "Remind me for class at 9 a.m." 4. App parses time and content. 5. Reminder is saved in local storage. 6. TTS confirms: "Class reminder set for 9 a.m."
Alternate Flow	If voice not understood, TTS asks: "Please repeat your reminder."
Exceptions	• No microphone access → TTS prompts for permission. • Parsing error → fallback to default time or ask user again.

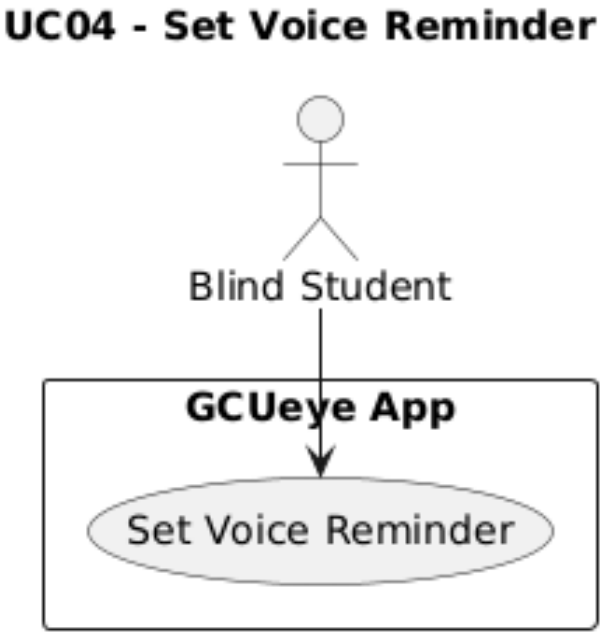


FIGURE 1.9: Use Case Diagram – Set Voice Reminder

UC05 – Receive Reminder Alert

Use Case ID	UC05
Actor	System
Description	At the scheduled time, the app alerts the user with speech and vibration to remind them of a scheduled class or task.
Preconditions	Reminder is saved. Device is on.
Postconditions	Reminder alert is spoken and vibration is triggered.
Main Flow	1. Reminder time is reached. 2. App triggers vibration (long pulse). 3. TTS speaks: "It's time for your class." 4. Optionally, user can dismiss with triple tap or long press.

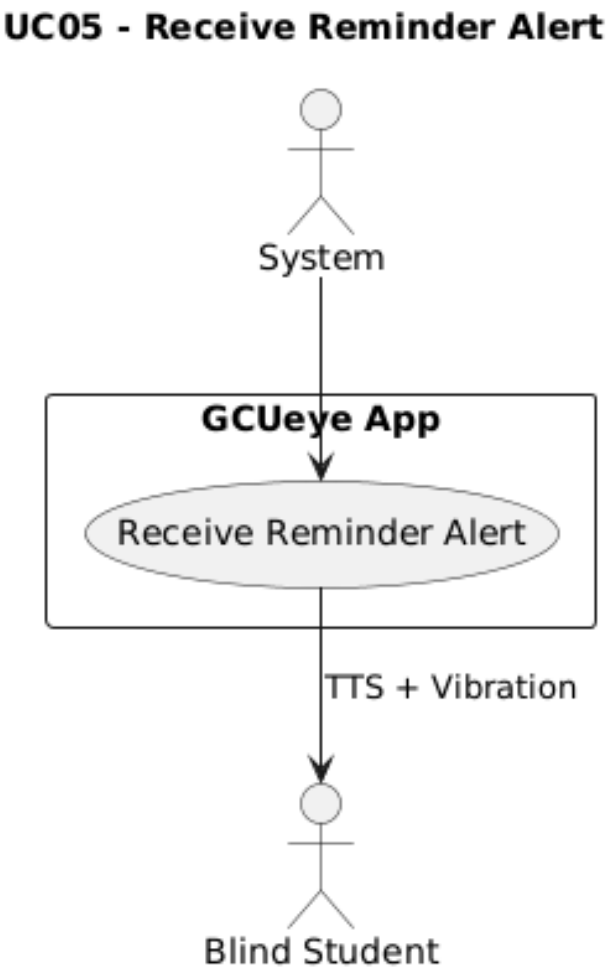


Figure 1.10: Use Case Diagram – Receive Reminder Alert

UC06 – Navigate App Features (Swipe Gesture)

Use Case ID	UC06
Actor	Blind student
Description	User navigates between the app's main functional screens (e.g., Camera, SOS, Reminders) by swiping left or right.
Preconditions	App is running. Gesture listener is active.
Postconditions	Selected screen is changed. TTS announces the name of the new feature.
Main Flow	1. User swipes left or right. 2. App navigates to next or previous feature screen. 3. TTS announces: "Camera View," "Reminder Section," or "SOS Page."
Alternate Flow	If already at first/last screen, TTS says: "No more screens."

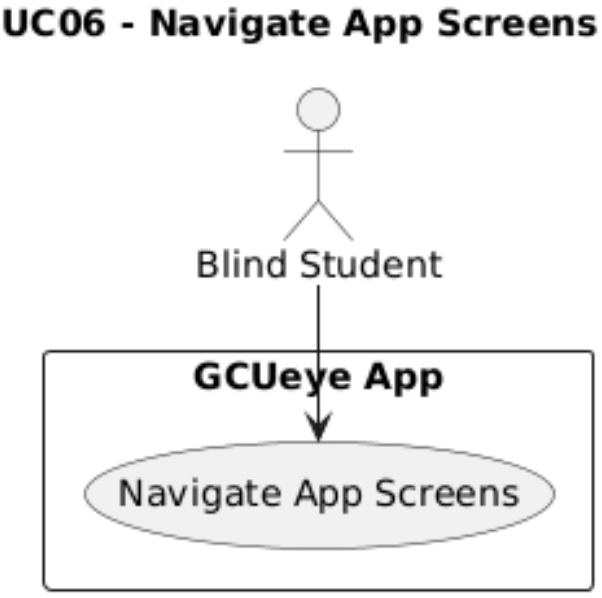


FIGURE 1.11: Use Case Diagram – Navigate App Features

UC07 – Indoor VR Navigation (Swipe + Steps)

Use Case ID	UC07
Actor	Blind student
Description	The user navigates from one indoor location to another by selecting source and destination nodes using swipe gestures. The app calculates a route and provides step-triggered voice instructions for each segment.
Preconditions	Pedometer active. Indoor map graph initialized.
Postconditions	User reaches destination with audio guidance.
Main Flow	1. User swipes to select source node. 2. User swipes to select destination node. 3. App calculates shortest route. 4. App begins navigation session. 5. Voice says: "Start walking straight." 6. After X steps → "Turn left into Corridor B." 7. App continues giving instructions until end node.
Alternate Flow	If nodes not selected, TTS says: "Please choose start and end locations."
Exceptions	• Step detection fails → fallback timer-based instruction. • Hazard on path → warning alert spoken.

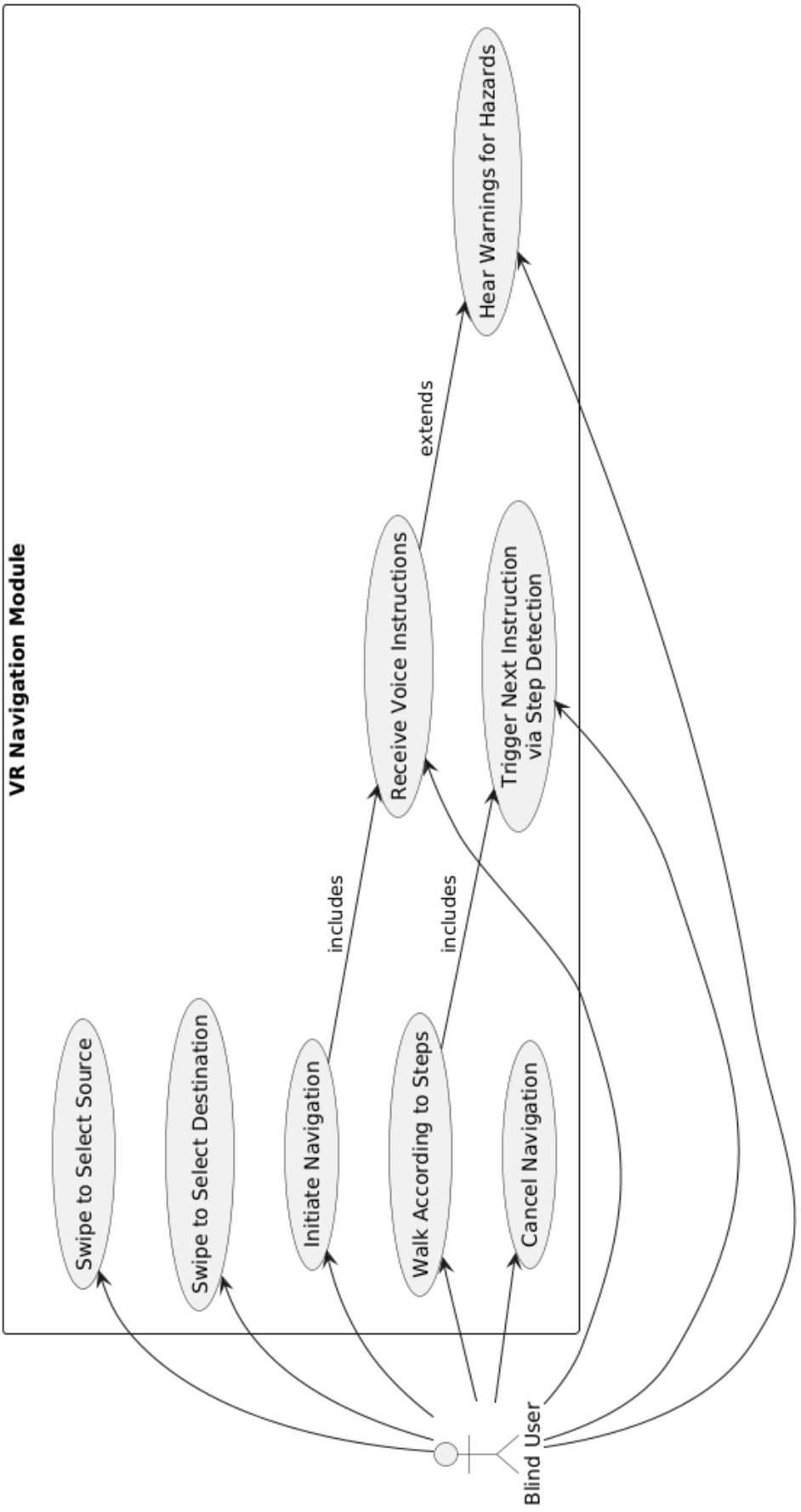


FIGURE 1.12: Use Case Diagram – Indoor VR Navigation (rotated)

UC08 – Configure Emergency Contacts (Voice Input)

Use Case ID	UC08
Actor	Blind student
Description	The user adds or updates emergency contact numbers by speaking the phone number aloud using long-press gesture input.
Preconditions	Microphone permission granted. Contact management screen active.
Postconditions	The emergency contact is saved locally and used during SOS triggers.
Main Flow	1. User navigates to the SOS Contact Setup screen. 2. Performs a long press. 3. TTS says: "Please speak the emergency contact number." 4. User says: "03xx-xxxxxxx" 5. App parses and stores the number. 6. TTS confirms: "Emergency contact saved."
Alternate Flow	If voice is not understood, TTS says: "Could not recognize the number. Please try again."
Exceptions	• Microphone permission denied → TTS: "Microphone access is required."

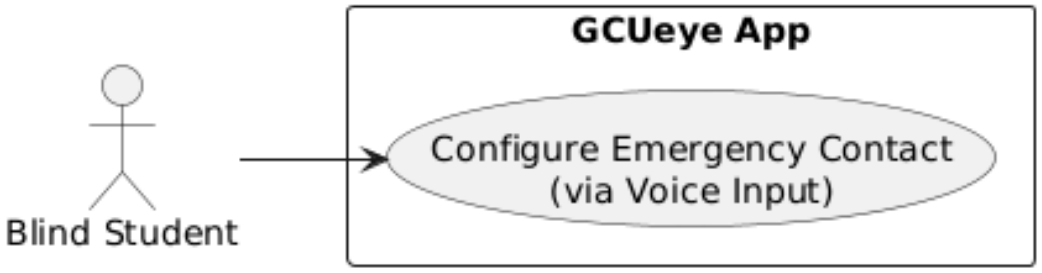


FIGURE 1.13: Use Case Diagram – Configure Emergency Contacts

UC09 – Adjust TTS Speed (Swipe + Tap)

Use Case ID	UC09
Actor	Blind student
Description	User adjusts the speech speed of TTS (Text-to-Speech) using gesture-based input.
Preconditions	Settings page active. TTS is enabled.
Postconditions	TTS speed value is updated and applied to all future voice feedback.
Main Flow	1. User swipes to reach "TTS Speed" setting. 2. TTS says: "TTS speed setting. Current: Normal." 3. User taps right to increase speed, or left to decrease. 4. TTS announces updated value: "Speed set to fast."
Alternate Flow	At min/max value → TTS says: "This is the slowest/-fastest speed."
Exceptions	• Setting fails to apply → TTS: "Unable to update speed."

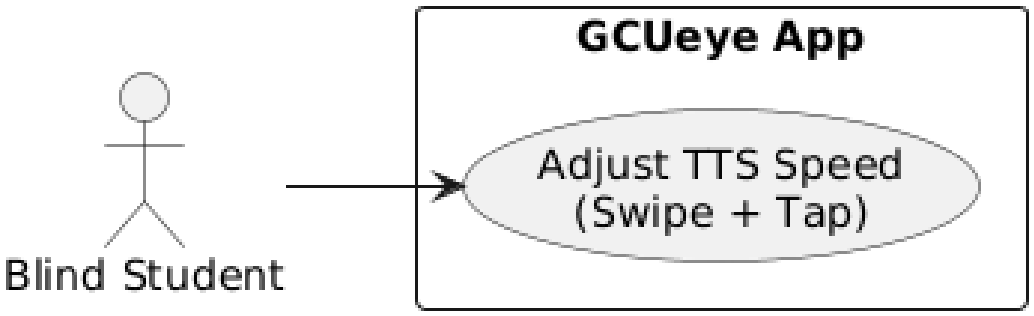


FIGURE 1.14: Use Case Diagram – Adjust TTS Speed

UC10 – Adjust TTS Pitch (Swipe + Tap)

Use Case ID	UC10
Actor	Blind student
Description	User modifies the pitch of the TTS voice using left and right screen taps.
Preconditions	Settings page active. TTS enabled.
Postconditions	Updated pitch is stored and applied to all future speech.
Main Flow	1. User swipes to reach "TTS Pitch" setting. 2. TTS says: "TTS pitch setting. Current: Medium." 3. User taps left to decrease, right to increase. 4. TTS says: "Pitch set to high."
Alternate Flow	At limit → TTS: "Pitch is already at maximum/minimum."
Exceptions	• Error applying pitch → TTS: "Failed to update pitch."

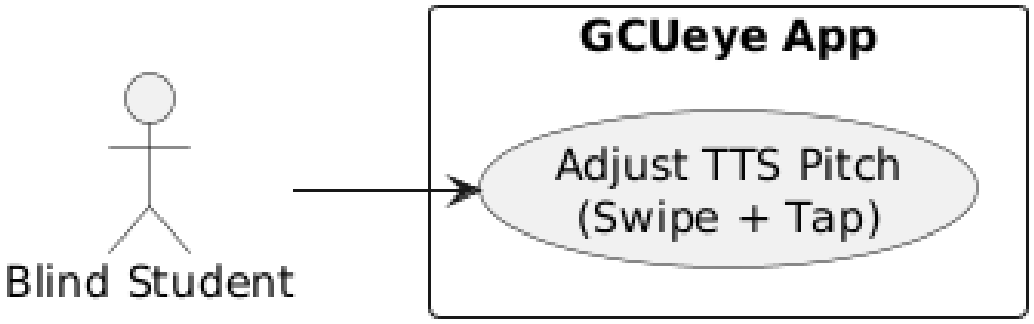


FIGURE 1.15: Use Case Diagram – Adjust TTS Pitch

UC11 – Toggle Voice Instructions On/Off

Use Case ID	UC11
Actor	Blind student
Description	The user enables or disables voice instructions for non-critical prompts.
Preconditions	Settings page active.
Postconditions	Voice instructions are toggled based on user input and saved.
Main Flow	1. User swipes to "Voice Instructions" setting. 2. TTS announces current state: "Voice instructions are ON." 3. User taps left to disable or right to enable. 4. TTS confirms: "Voice instructions turned OFF."
Alternate Flow	Toggle again → TTS reflects new state.
Exceptions	• Setting fails → TTS: "Unable to update setting."

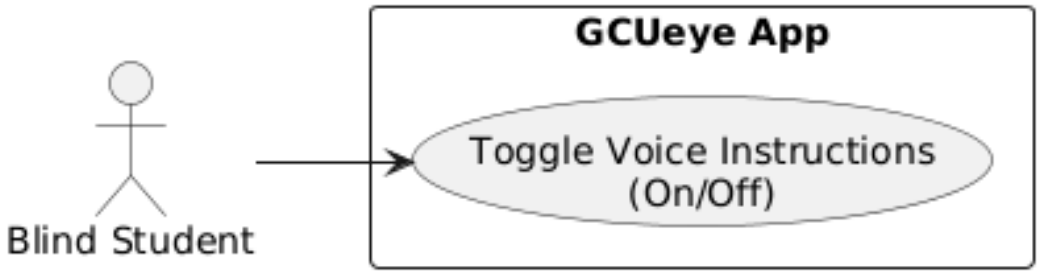


Figure 1.16: Use Case Diagram – Toggle Voice Instructions

UC12 – Toggle Vibration Feedback On/Off

Use Case ID	UC12
Actor	Blind student
Description	The user toggles vibration feedback used in object detection and alerts.
Preconditions	Settings page active. Device supports vibration.
Postconditions	Vibration feedback is turned on or off.
Main Flow	1. User swipes to "Vibration Feedback" setting. 2. TTS says: "Vibration is currently ON." 3. User taps left to disable or right to enable. 4. TTS confirms: "Vibration turned OFF."
Alternate Flow	No motor detected → TTS: "Vibration not supported on this device."
Exceptions	• Setting fails → TTS: "Unable to update."

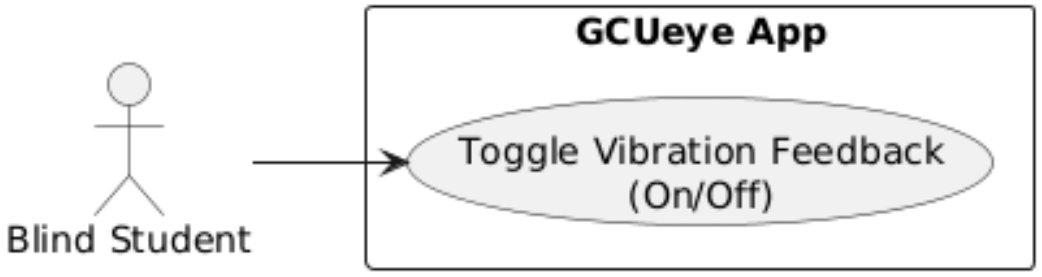


FIGURE 1.17: Use Case Diagram – Toggle Vibration Feedback

UC13 – Reset All Settings to Default

Use Case ID	UC13
Actor	Blind student
Description	User resets all settings (TTS speed, pitch, toggles) to factory defaults using a long press or combined tap.
Preconditions	User is on "Reset Settings" section of the Settings page.
Postconditions	All settings are reverted to their original values.
Main Flow	1. User swipes to "Reset Settings." 2. Performs a long press or simultaneous left/right tap. 3. App resets all configurable options to default. 4. TTS confirms: "All settings have been reset to default."
Alternate Flow	Accidental trigger → TTS warns: "Hold again to confirm reset."
Exceptions	• Reset fails → TTS: "Unable to reset settings."

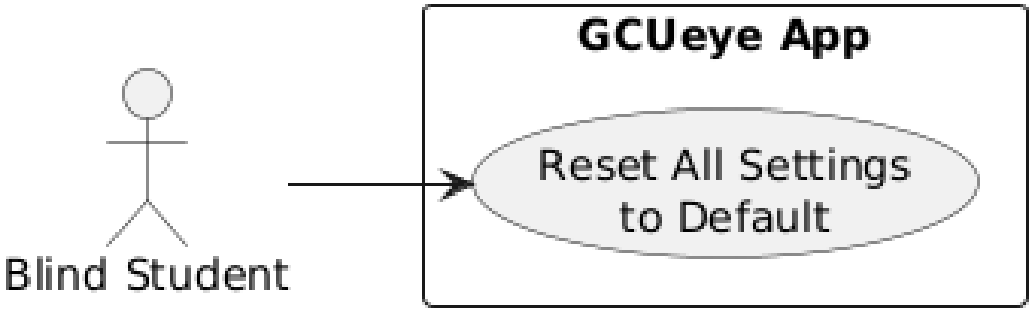


FIGURE 1.18: Use Case Diagram – Reset All Settings to Default

1.6.1 Use Case Diagram of the GCUeye Application

These use cases collectively define how a blind student can fully operate the app — without visual support — for daily navigation, schedule management, and safety.

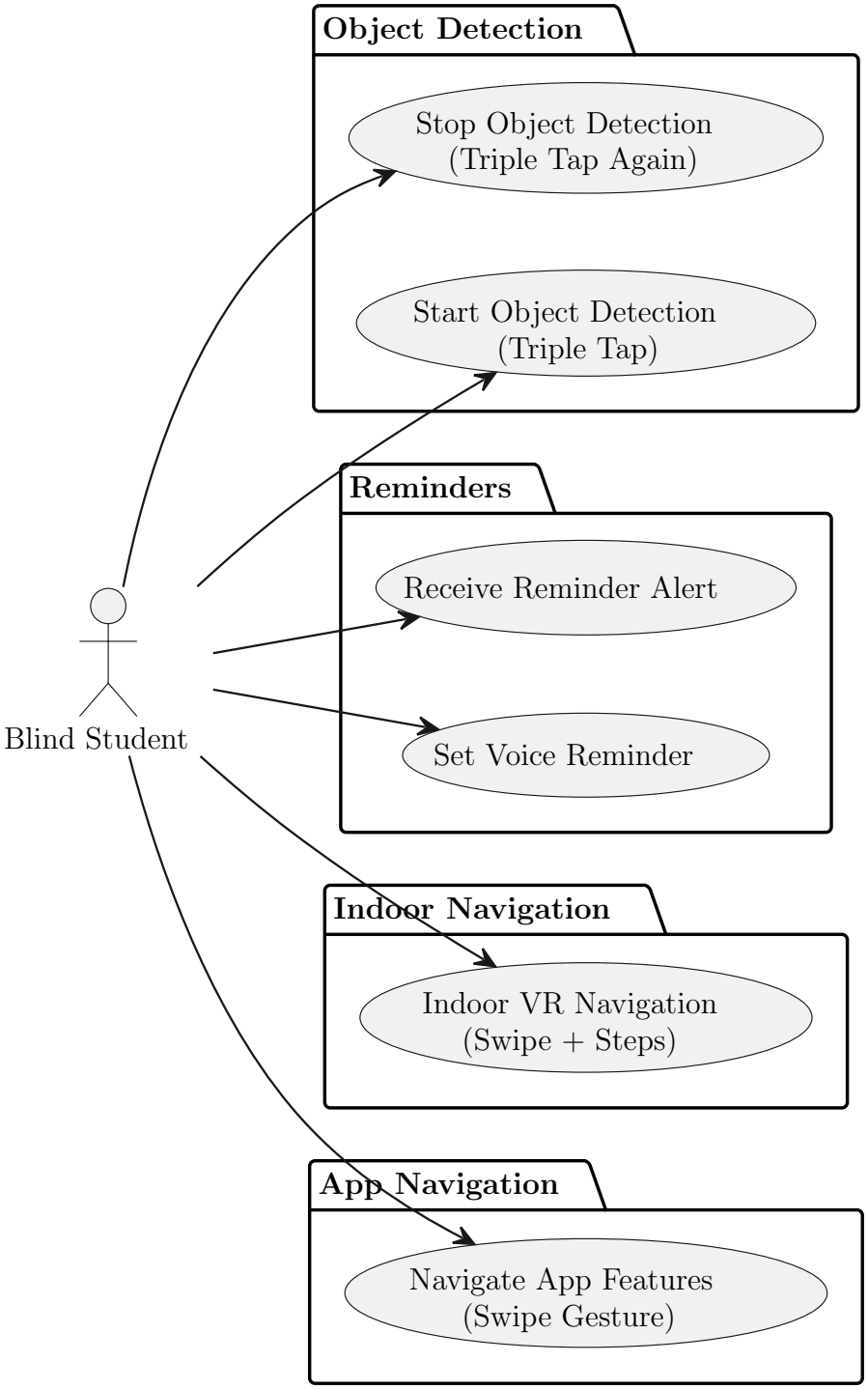


FIGURE 1.19: Use Case Diagram – GCUeye App (Part 1)

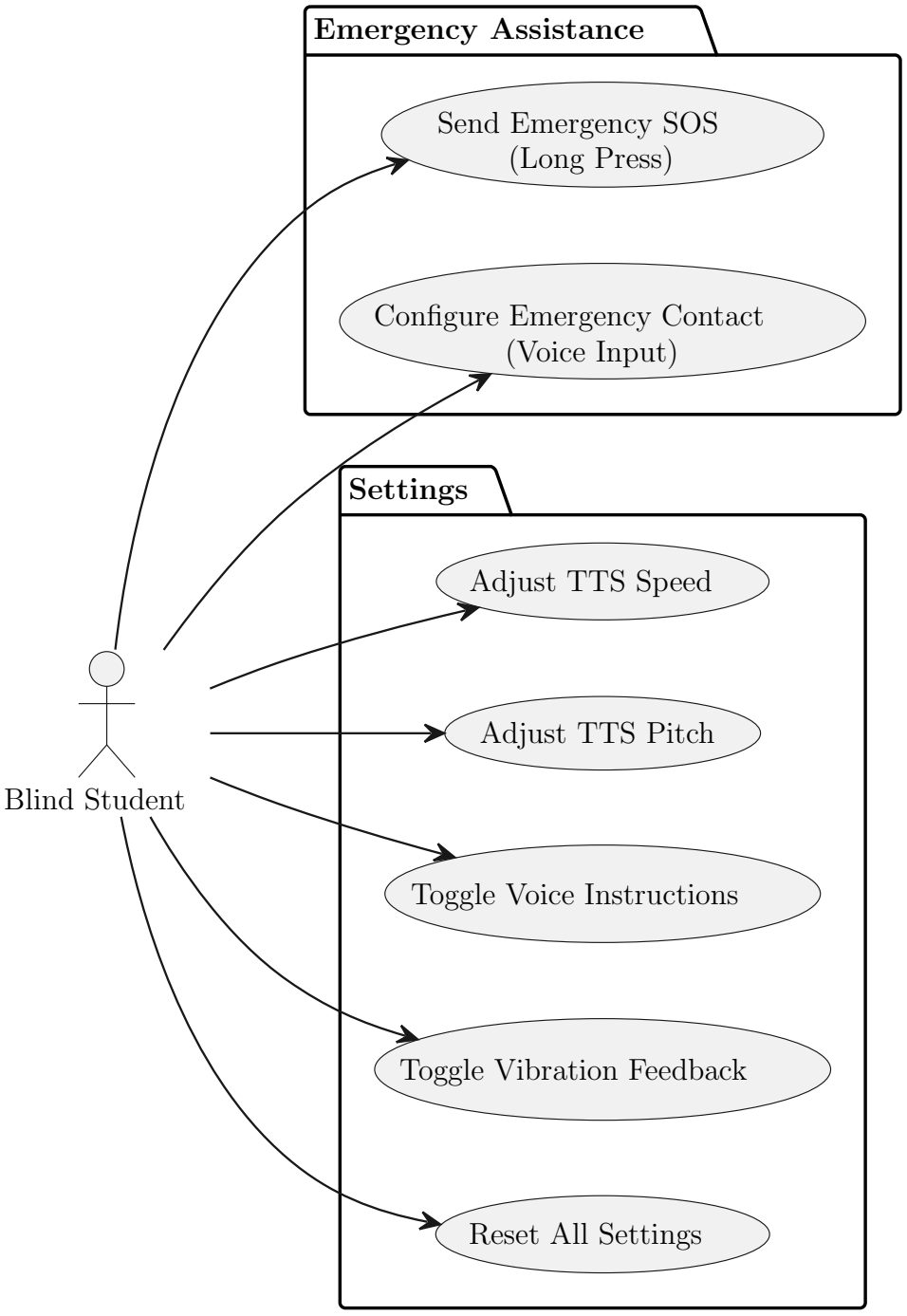


FIGURE 1.20: Use Case Diagram – GCUeye App (Part 2)

Chapter 2

Requirements Specification

This chapter formally defines the core requirements for the development and deployment of the blind navigation app. It includes:

- Functional requirements (what the system should do), and
- Non-functional requirements (how well it should perform).

The requirements were derived from problem analysis, user interviews, and testing assumptions made during the initial prototype phase.

2.1 Functional Requirements

Functional requirements define the core capabilities the blind navigation app must offer to fulfill its purpose: to help visually impaired students safely and independently navigate the GC University campus through non-visual interaction. This includes outdoor object detection, gesture-based controls, voice reminders, and indoor VR-style navigation.

Each requirement below describes what the app should do, how the user interacts with it, and what the system must ensure to deliver a seamless experience.

2.1.1 Camera-Based Object Detection

The system must be capable of performing real-time object detection using the smartphone's camera and an AI model hosted locally. This feature is foundational to the app's ability to assist blind users in understanding and navigating their environment.

- The app must access the rear-facing camera to periodically capture images, ideally every 1–2 seconds.
- Each captured frame must be encoded and transmitted via HTTP to a locally running Flask server [6], which hosts a trained YOLOv8 [17] object detection model.
- The model must return a structured list of detected objects, including their class labels (e.g., "chair," "person," "pothole") and their bounding box coordinates.
- The bounding box data must be processed within the app to determine the spatial location of each object (i.e., left, center, or right side of the frame).
- The app must interpret and describe these detections to the user using a combination of:
  - Spoken output using Text-to-Speech [5](e.g., "Chair on the left"),
  - Directional vibration cues (e.g., one buzz = left).

This functionality must be consistent, responsive, and continue in a loop until manually stopped by the user. Performance must be optimized so that the time from image capture to feedback remains under 1.5 seconds in total.

2.1.2 Direction-Based Spatial Interpretation

The object detection system alone is insufficient without directional interpretation, which enables the user to understand where exactly the object is located relative to their current viewpoint.

- Based on the horizontal position of an object's bounding box (typically the x-coordinate of its center), the app must assign the object to one of three spatial zones:
  - Left: Bounding box center is within the left third of the screen.
  - Center: Object appears directly ahead (middle third).
  - Right: Object is in the right third of the field of view.
- The direction must be appended to the spoken feedback, forming a descriptive phrase such as:
  - "Bag on the right,"
  - "Stairs ahead,"
  - "Wall on the left."
- This logic must be applied for every object detected in a frame, and the app should prioritize speaking only the most relevant or closest objects, to prevent overwhelming the user with too much information.

The success of this function will directly impact the student's mobility safety, making it essential for every detected object to be accompanied by clear spatial feedback.

2.1.3 Gesture-Based Feature Activation

The app must operate exclusively through gestures, ensuring blind users can access all major functions without relying on on-screen navigation or visual components.

The following gestures must be supported across all screens:

- **Triple Tap:**
  - Acts as a toggle to start or stop object detection.
  - On activation, the app should capture and analyze frames continuously until another triple tap is detected.
  - On deactivation, all detection-related processes must be safely terminated.
- **Long Press:**
  - Triggers the SOS feature, which includes both a help message (SMS) and a voice call to the configured emergency contact.
  - Must be designed to work regardless of current screen context or network state.
- **Horizontal Swipe (Left/Right):**
  - Allows users to navigate between the main screens of the app: Detection View, Reminder Setup, and SOS Configuration.
  - After every swipe, a voice prompt must announce the currently active screen, e.g., "You are now on the Reminder screen."

These gestures must be processed instantly and with high reliability. A misfire or delay in recognition could lead to confusion, missed alerts, or failure to respond to hazards.

2.1.4 Multimodal Feedback (Voice + Haptic)

Blind users benefit most from systems that combine multiple forms of feedback. This app must therefore implement dual-channel output: speech (via TTS) and tactile vibration feedback, especially during object detection and critical events.

- Every detected object should be:
  - Spoken aloud using Text-to-Speech in a short, precise sentence.
  - Accompanied by a vibration pattern that encodes directional context:
    - One short buzz = Left
    - Two medium buzzes = Center
    - Three short buzzes = Right
- In addition to obstacle alerts, the app must use voice prompts to:
  - Announce screen names after a swipe.
  - Confirm actions such as "Reminder saved" or "SOS triggered."
- The vibration motor must be triggered in sync with voice output, providing non-redundant support in noisy environments or for users with dual impairments (e.g., partial hearing loss).

2.1.5 Voice-Controlled Reminder Scheduling

Blind students often lack access to accessible tools for scheduling their day. To solve this, the app must support speech-based reminder creation.

- The app must offer a voice prompt asking, "What would you like to be reminded about?"
- Users can respond naturally with phrases like:
  - "Remind me for class at 10 a.m."
  - "Remind me to submit the assignment at 5 p.m."
- The app must:
  - Parse the spoken sentence,
  - Extract both the time and the content of the reminder,

  - Store it securely and locally,
  - Confirm via voice: "Reminder for class at 10 a.m. saved successfully."

- At the scheduled time:
  - The app must trigger a TTS alert (e.g., "This is your 10 a.m. class reminder.")
  - Trigger a vibration notification as backup.

Reminders must persist across sessions and function offline. Users should not be required to manually type or set timers.

2.1.6 VR-Based Indoor Navigation

Blind students often struggle with navigating indoor environments where GPS does not work (e.g., corridors, classrooms). To address this, the app must include a virtual indoor navigation system based on a graph of connected nodes and step-based progress tracking.

- Users must be able to select a start and end location using horizontal swipe gestures.
- The app must calculate the shortest route using Dijkstra's algorithm on a predefined building graph.
- Navigation instructions should be provided using Text-to-Speech, such as:
  - "Walk straight for 4 steps,"
  - "Turn left into Corridor A,"
  - "Room 103 is on your right."
- Each instruction must be triggered using pedometer-based step detection, not location services.

- Hazards like stairs, blocked paths, or uneven floors must trigger custom audio alerts (e.g., “Caution: Stairs ahead”).
- The graph data must be editable and loaded dynamically, allowing future building maps to be supported.

This feature must work offline, require no Wi-Fi or GPS, and function using only gestures, audio, and pedometer sensors.

2.1.7 Emergency Communication System (SOS)

In high-stress or emergency situations, blind users must have an instant method of reaching help — without unlocking their device, searching through contacts, or navigating visual menus.

The app must include a gesture-triggered emergency feature that performs the following:

- A long press triggers the SOS system.
- The system must immediately:
  - Send a prewritten SMS message (e.g., “I need help. Please come to my location.”) to a configured emergency contact.
  - Initiate a voice call to the same contact or a secondary number.
- All actions must occur in sequence without requiring any user confirmation.
- If the system fails (e.g., no network), a voice message should alert the user: “Unable to send SOS at this time.”

This feature must be available even if the app is not on the SOS screen, making it a global safety shortcut.

2.1.8 Screenless Interface Navigation

Blind users should never be forced to look at, read, or interact with visual screen components. Thus, the app must support full screenless operation.

- Each page of the app (Camera View, Reminder, SOS Setup) should be reachable through horizontal swipe gestures.
- Upon landing on a screen, the app must:
  - Announce the screen name via voice (e.g., "Camera Detection View").
  - Provide a brief hint (e.g., "Triple tap to start detection.")
- The app must not require visual confirmation, labels, or form entry at any point.

This design ensures that the user can interact with the app purely by touch and sound, in line with universal accessibility principles.

2.1.9 Persistent Local Storage of User Data

To support daily use, the app must securely store:

- Emergency contact numbers,
- All created reminders.

This data should:

- Be saved locally (not in the cloud),
- Be retained across app restarts and phone reboots,
- Be editable using voice or via a simplified caregiver UI if needed,
- Be protected from accidental deletion or background process interference.

2.2 Non-Functional Requirements

While functional requirements describe what the system must do, non-functional requirements define how well the system must perform these tasks. They represent the quality attributes that influence the system's performance, usability, reliability, scalability, and overall user experience — especially in the context of a mobile assistive solution designed for visually impaired users.

Given that this blind navigation app will be used by students in real-time and often in challenging conditions (e.g., outdoor spaces, noisy corridors, offline environments), these non-functional properties are critical to its success.

2.2.1 Performance and Responsiveness

The system must deliver a consistently fast and smooth experience, especially during object detection and feedback cycles. Delays or lag in a real-time assistive app could lead to safety risks and user frustration.

- The object detection pipeline — from camera capture to feedback delivery — must complete in under 1.5 seconds, with an ideal target of 1 second.
- Voice feedback must be generated and spoken immediately after detection results are received, without noticeable lag. The user should not experience long pauses or buffering.
- Vibration feedback must be triggered in parallel with the voice output, and ideally within 200 milliseconds of object classification.
- The app must maintain this performance consistently even on mid-range Android devices, ensuring it does not require high-end hardware to function properly.
- In VR navigation mode, each step-triggered instruction must be spoken within 300 milliseconds of the detected step to maintain navigational clarity.

2.2.2 Reliability and Offline Functionality

The app must function reliably in a wide range of real-world conditions — especially without internet access, which may be unavailable in parts of the university (e.g., basements, older buildings).

- All core features — including object detection, vibration and voice feedback, reminder alerts, and emergency communication — must operate offline.
- The object detection server (Flask + YOLOv8) must be accessible via a local network (e.g., mobile hotspot or LAN), eliminating reliance on cloud services.
- The SOS module must use standard SMS and telephony, so that even with no Wi-Fi, users can still reach out for help via the mobile network.

This ensures independence from internet services, improves speed, protects privacy, and makes the system suitable for use in unpredictable connectivity environments.

2.2.3 Usability and Accessibility

The blind navigation app is designed for users who cannot rely on sight, and therefore must follow strict usability and accessibility principles tailored for screenless interaction.

- The app must support gesture-only navigation and offer spoken feedback for all key actions and events. The user should never need to read or see anything on the screen.
- Each screen or feature must include a clear audio description, including prompts for how to interact (e.g., "Swipe right for SOS options.").
- Audio feedback must use a neutral, natural voice, at a medium pace, and in clear language (English and optionally Urdu).

- Vibration patterns must be simple, recognizable, and repeatable — ideally differing between left/center/right so the user can build muscle memory around them.

- In VR navigation, users must be guided via short, simple spoken directions like “Turn left,” “Go straight,” or “You have arrived,” triggered only through step detection.

- A new user should be able to:

  – Understand the basic features within 10 minutes of guided onboarding,

  – Use detection, SOS, and reminders without external help or training,

  – Navigate confidently between screens based solely on touch and sound.

2.2.4 Security and Privacy

Since the app stores personal emergency contact details and private schedule information, it must enforce privacy protection, even if all storage is local.

- Emergency contact numbers must be stored in protected local storage (e.g., SharedPreferences [4] or encrypted SQLite [15]), inaccessible to other apps or processes.

- No personally identifiable data — such as user name, ID, or live camera footage — should ever be transmitted or stored online.

- SOS messages must be generalized and respectful of privacy. For example:

  “This is a help alert from a blind user. Please contact them urgently.”

This ensures that while the app provides safety features, it does not compromise the user’s personal information or autonomy.

2.2.5 Compatibility and Device Requirements

The system must be compatible with a broad range of Android devices, particularly those used by students, many of whom may not have the latest phones.

- The app must function on devices running Android 10 or higher.
- It must support various screen sizes (from small 4-inch phones to large 6.5-inch devices).
- Hardware requirements must be minimal, limited to:
  - A working rear camera (5 MP),
  - A microphone (for voice reminders),
  - A vibration motor (for haptic feedback),
  - Basic telephony and SMS support.
- The app must not demand high RAM or CPU usage and should remain responsive even under heavy multitasking or low battery conditions.

2.2.6 Scalability and Maintainability

While the current version is designed for GCU-specific use, the system must be flexible enough to scale or be repurposed for other campuses, user groups, or future features.

- The detection model must be retrainable or replaceable. If new objects are added or accuracy is improved, the new `.pt` model should be easily deployable on the same server.
- The VR navigation graph should be modular, allowing additional nodes and paths to be added for other indoor environments without rewriting navigation logic.

- The app codebase must be modular and maintainable, with clearly separated logic for:
  - Camera handling,
  - Gesture recognition,
  - Voice/text feedback,
  - Network communication.
- Future features like GPS navigation, indoor mapping, or user analytics must be integratable without refactoring the core detection-feedback loop.

This ensures long-term sustainability of the system, enabling upgrades and customizations without starting from scratch.

2.2.7 Fault Tolerance and User Feedback

Mobile apps can fail due to network drops, camera errors, or user missteps. The system must be fault-tolerant and provide clear, user-friendly responses during such events.

- If the Flask server is disconnected, the app should announce: *"Detection temporarily unavailable. Please check connection."*
- If voice recognition fails to understand a reminder, the app should respond with: *"I didn't catch that. Please repeat your reminder."*
- If SOS fails due to network issues or SIM unavailability, the app must say: *"Unable to send message. Please move to an open area."*
- The system must never crash silently or leave the user unsure about what happened. Every exception must be paired with meaningful, accessible feedback.

Chapter 3

Project Design

This chapter presents the design architecture, methodologies, data pipelines, and component-level breakdown of the blind navigation mobile application. It focuses on how the integration of advanced technologies like YOLOv8, Flask, and Flutter is used to deliver a real-time, offline, and gesture-based solution tailored specifically for visually impaired students at Government College University Lahore. Every aspect of the system was designed to emphasize accessibility, low latency, modularity, and real-world usability under diverse campus conditions.

3.1 Development Methodology

The system was developed using a modular, user-centered, and incremental approach. This methodology supported rapid prototyping, early integration, and continuous feedback, making the system adaptable and accessible from early stages of development.

3.1.1 Incremental Development Approach

The development cycle was broken down into testable and manageable sprints, each composed of four key phases:

1. **Requirement Finalization and Scoping:** User scenarios and environmental constraints were analyzed to identify essential features.
2. **Feature Prototyping and Isolated Testing:** Each module (e.g., object detection, feedback) was developed independently.
3. **Module Integration and Real-Device Testing:** Integrated modules were tested on Android devices to ensure compatibility and reliability.
4. **Accessibility Testing and Simulated Usage:** The app was tested by blindfolded users in realistic campus environments to validate usability.

This approach minimized bugs and enhanced usability by validating modules both in isolation and in context.

3.1.2 Phase-Wise Implementation Strategy

- **Phase 1 – Planning and Prototyping:**
  - Defined gestures: triple tap, long press, swipes.
  - Designed interaction flowcharts for screenless navigation.
  - Configured a Flask server backend for model hosting.
- **Phase 2 – YOLOv8 Dataset Training:**
  - Combined multiple datasets into one labeled for 17+ classes relevant to campus navigation.
  - Used YOLOv8n (nano variant) with transfer learning for speed and efficiency.
  - Tuned detection thresholds to minimize false positives.
- **Phase 3 – Client-Server Integration:**
  - Created a Flask API endpoint `/predict` to receive image data and respond with detection results.

  - Implemented an HTTP client in Flutter to send images and process JSON responses.
  - Added fallback logic for dropped frames or server inaccessibility.

- **Phase 4 – Feature Expansion:**
  - Added voice-controlled reminder scheduling.
  - Integrated a long-press-based SOS system (SMS + call).
  - Rebuilt UI into a swipe-driven interface for easy navigation.

- **Phase 5 – Accessibility Enhancements:**
  - Implemented TTS feedback for all user interactions.
  - Used vibration patterns to indicate obstacle direction.
  - Conducted final usability testing under blindfolded conditions.

- **Phase 6 – VR Navigation Integration:**
  - Modeled the GCU building as a graph of nodes and weighted edges.
  - Implemented Dijkstra's algorithm to calculate optimal indoor routes.
  - Built gesture-based source/destination selection and step-based instruction delivery.
  - Added TTS instructions like "Turn left" or "You have reached your destination."
  - Integrated hazard alerts using audio cues (e.g., "Stairs ahead").

3.2 System Architecture Overview

3.2.1 High-Level Architecture

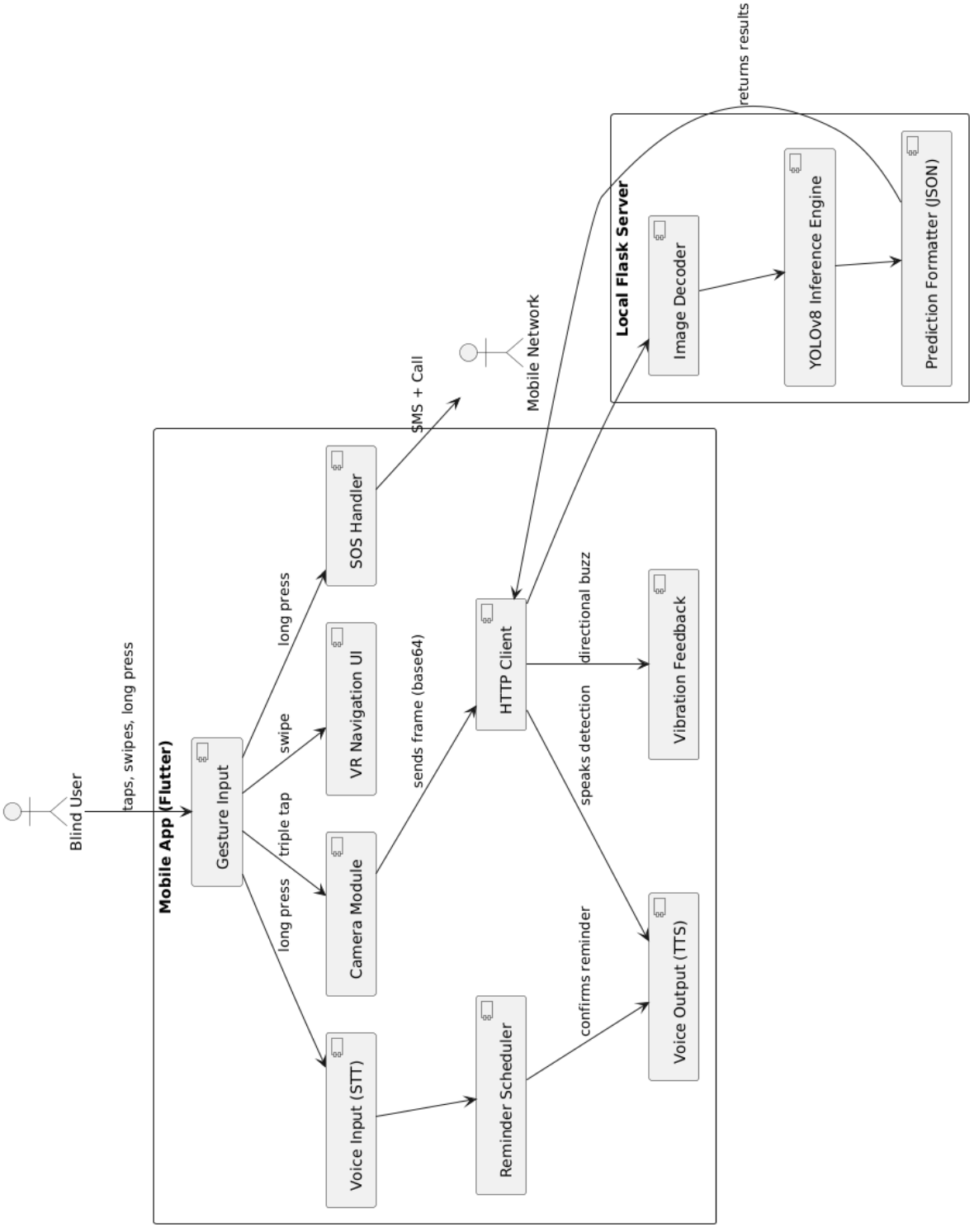


FIGURE 3.1: System Architecture Diagram of GCUeye (Rotated and Expanded)

3.2.2 Data Flow Pipeline

1. **Frame Capture:** Rear camera captures a frame every 1–2 seconds.
2. **Image Encoding:** Frame is compressed and encoded in base64.
3. **HTTP Request:** Flutter sends encoded image via POST to `/predict`.
4. **YOLOv8 Inference:** Server decodes image, performs detection, returns objects with labels, scores, and bounding boxes.
5. **Direction Classification:**
   - Left: $x_{center} < 0.33$
   - Center: $0.33 \leq x_{center} \leq 0.66$
   - Right: $x_{center} > 0.66$
6. **Feedback Execution:** TTS speaks detections (e.g., "Stairs ahead"), and vibration patterns are triggered (1 buzz = left, etc.).
7. **Indoor Navigation Trigger:** Based on step count input, next route instruction is fetched from path graph and played using TTS.

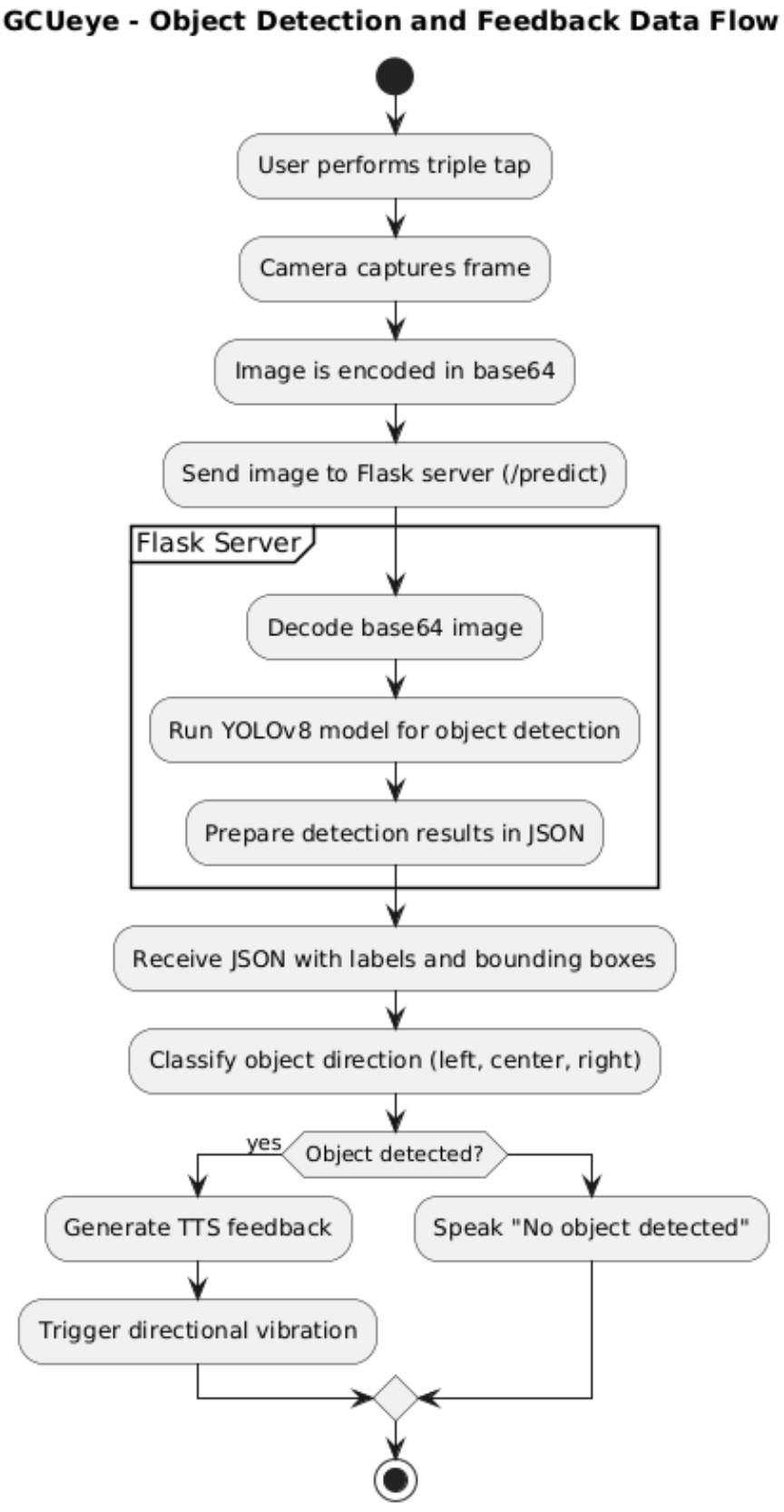


FIGURE 3.2: GCUeye - Object Detection and Feedback Data Flow

3.3 Component-Level Design

3.3.1 YOLOv8 Detection Module (Flask Server)

Component	Description
Framework	Flask (Python 3.10)
Model	YOLOv8n `.pt` file, loaded on server boot
Endpoint	`/predict` — accepts image via POST
Response	JSON: label, confidence, bounding box
Inference Time	200–400 ms per image

3.3.2 Flutter Frontend (Mobile Application)

Module	Functionality
Camera View	Captures camera frames, encodes and sends to server
TTS Feedback	Uses `flutter_tts` [1] to narrate detections and prompts
Vibration API	Executes buzz patterns based on object direction
Gesture Recognition	Triple tap (detect), swipe (navigate), long press (SOS)
Voice Input	Parses natural language reminders via `speech_to_text` [2]
Reminder Scheduler	Triggers TTS + vibration at scheduled time
SOS Handler	Sends emergency SMS and calls predefined contact

3.3.3 User Interaction Flow

1. App starts → TTS: “Welcome. Swipe to choose a feature.”
2. User triple-taps → Detection begins.
3. Frame captured → Detection result → “Wall on the left” + buzz.
4. User swipes → Reminder screen → Says: “Remind me for exam at 3 p.m.”
5. At 3 p.m. → Buzz + TTS: “It’s time for your exam.”
6. User long-presses → SOS message sent and call placed.

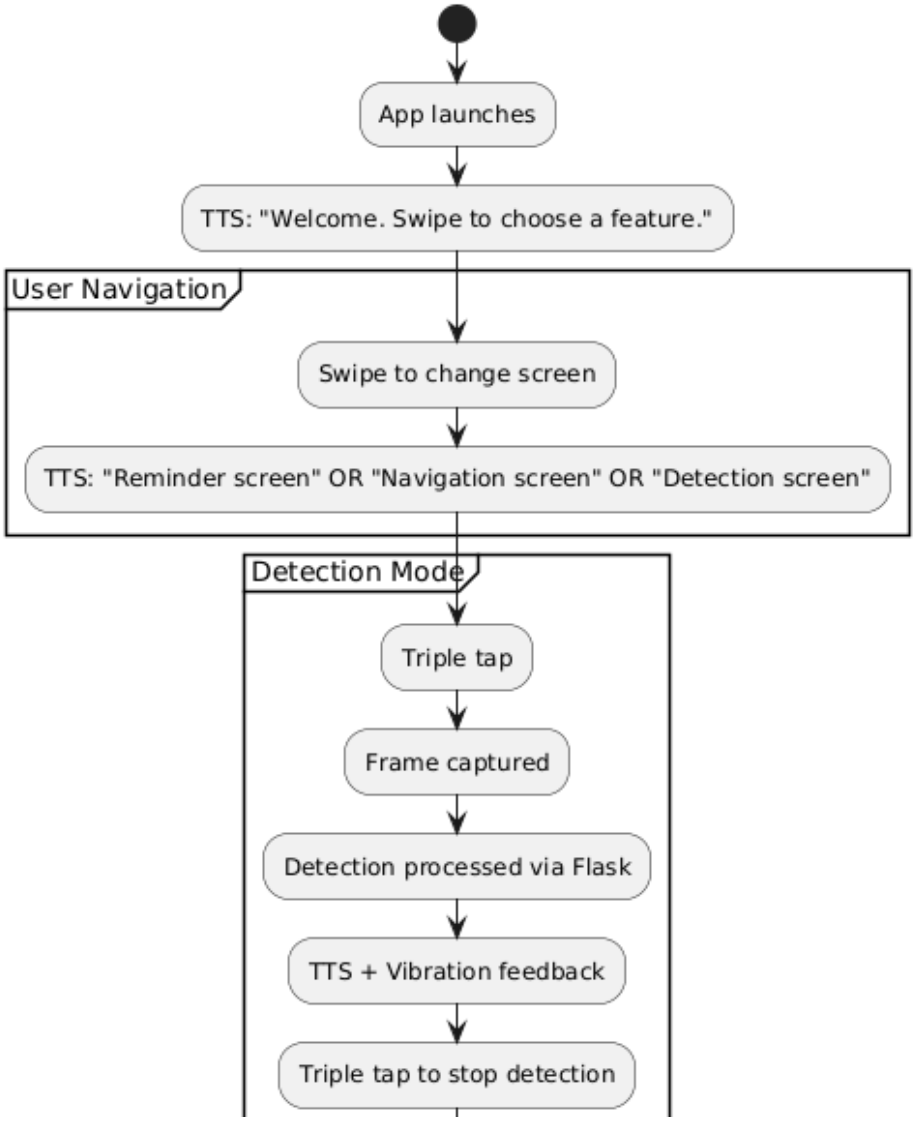


Figure 3.3: User interaction flow - Part 1

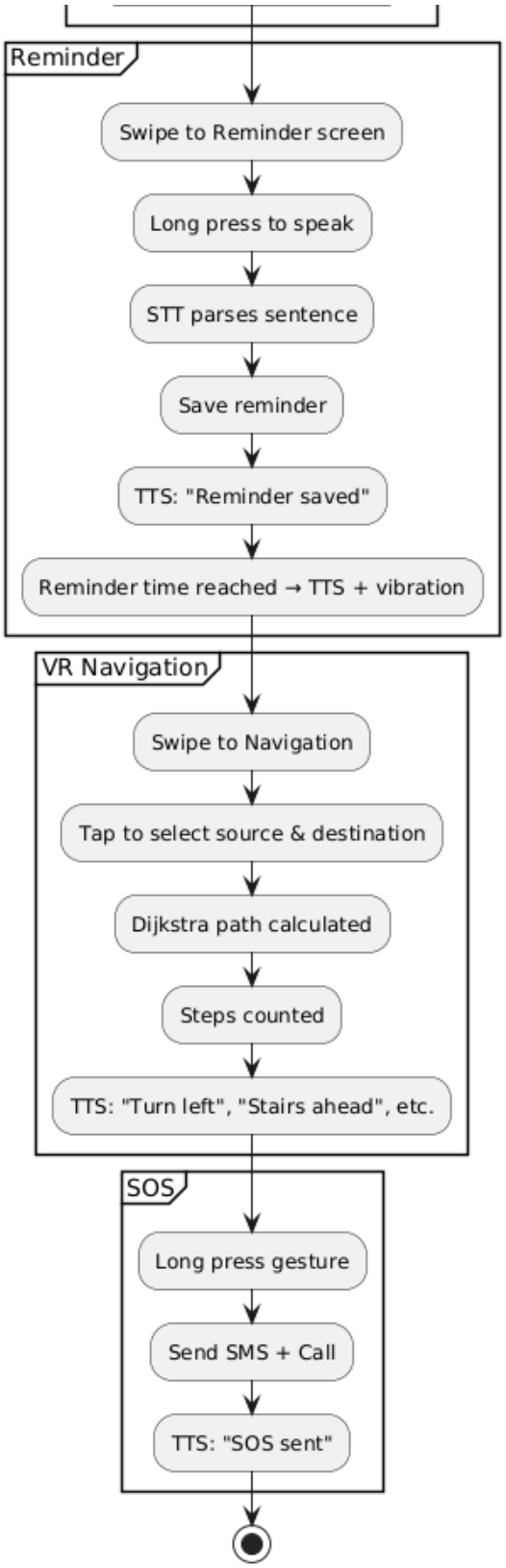


Figure 3.4: User interaction flow – Part 2 (continued)

3.3.4 VR Indoor Navigation Design

The VR Navigation system allows blind users to navigate indoor environments like corridors, classrooms, and stairs by simulating real-world movement using step counts and gesture input. It is designed to work entirely offline, without GPS or Wi-Fi.

- The building layout is modeled as a graph, where each node represents a physical location (e.g., "Corridor A", "Room 101") and edges represent walkable paths.
- The user swipes to select source and destination locations using voice feedback.
- Upon confirmation, the system uses Dijkstra's algorithm to calculate the shortest path.
- Each segment of the path includes metadata such as direction, hazard warnings, and step count.
- The app listens to the pedometer or the user tap on screen to confirm the steps taken, to track steps and trigger the next instruction (e.g., "Turn right into Room 102").
- If a danger zone (e.g., stairs) is nearby, a TTS alert like "Caution: Stairs ahead" is played.

This component offers safe and intelligent indoor movement simulation using just TTS, gestures, and the step sensor—empowering blind users to navigate campus buildings independently.

3.3.5 Entity Relationship Diagram (ERD)

The ERD represents the logical structure of the data used within the GCUeye mobile application.

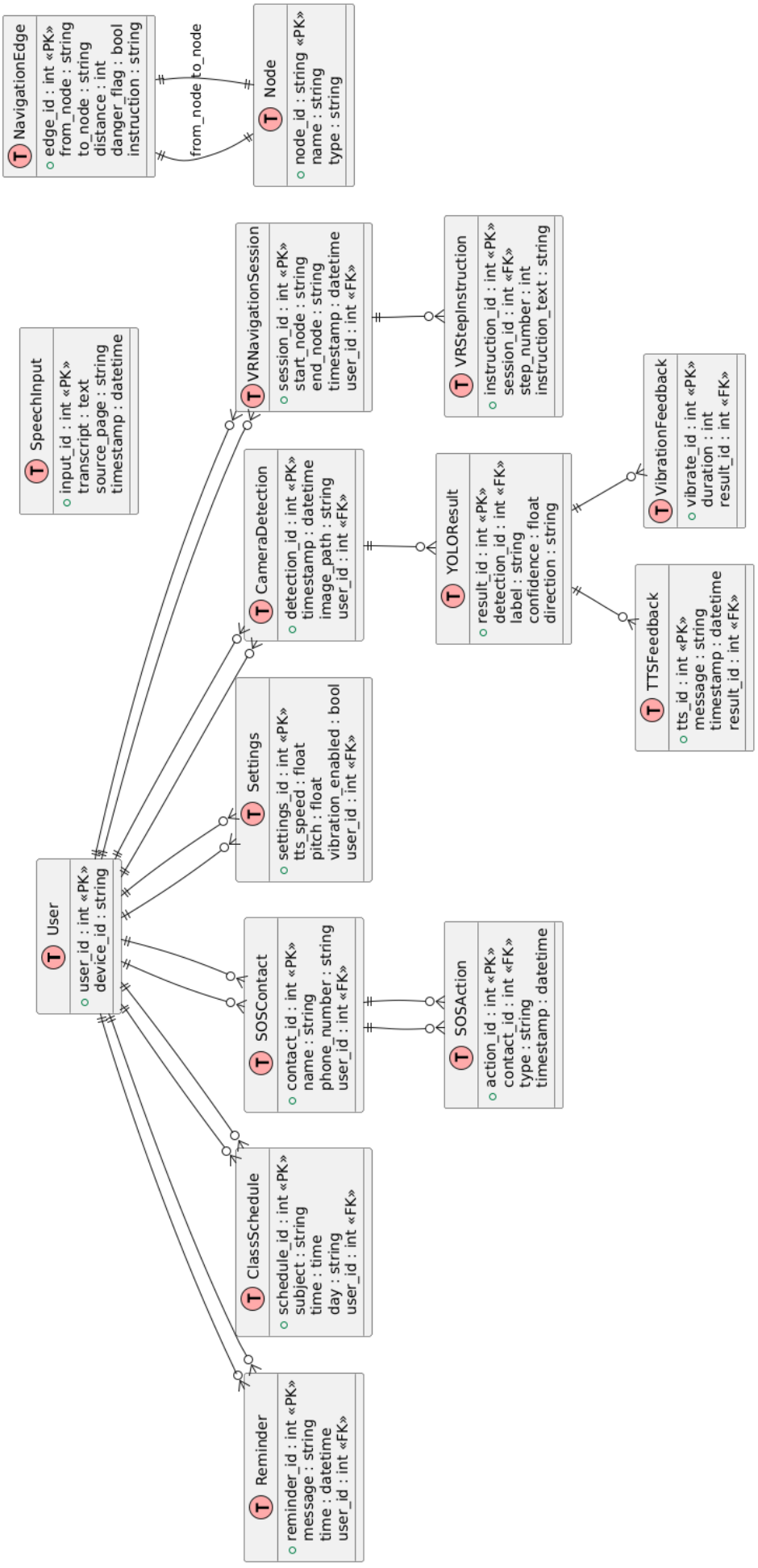


FIGURE 3.5: Entity Relationship Diagram for GCUeye System (Rotated)

This ERD shows that the blind user interacts with modules like reminders, class schedule, camera detection, VR navigation, and SOS contacts. YOLO-based detection results are linked to TTS and vibration feedback, while VR navigation relies on node-edge graphs and step tracking. Each module stores structured data to ensure modularity, testability, and persistence.

3.3.6 Data Flow Diagrams (DFD)

Data Flow Diagrams illustrate how data travels between user actions, system processes, and storage units. GCUeye relies on a gesture and voice-driven input system, supported by a Flask backend and modular feedback mechanisms.

- **DFD Level 1 – System Overview:** Presents the overall flow of data between major system modules such as object detection, reminders, VR navigation, and feedback. It also includes interactions with the YOLOv8 model via Flask API.
- **DFD Level 2 – VR Navigation Module:** Breaks down the internal logic of the VR navigation process, including gesture-based source/destination selection, Dijkstra pathfinding, step counting, and hazard alerts.

3.3.6.1 DFD Level 1 – System Overview

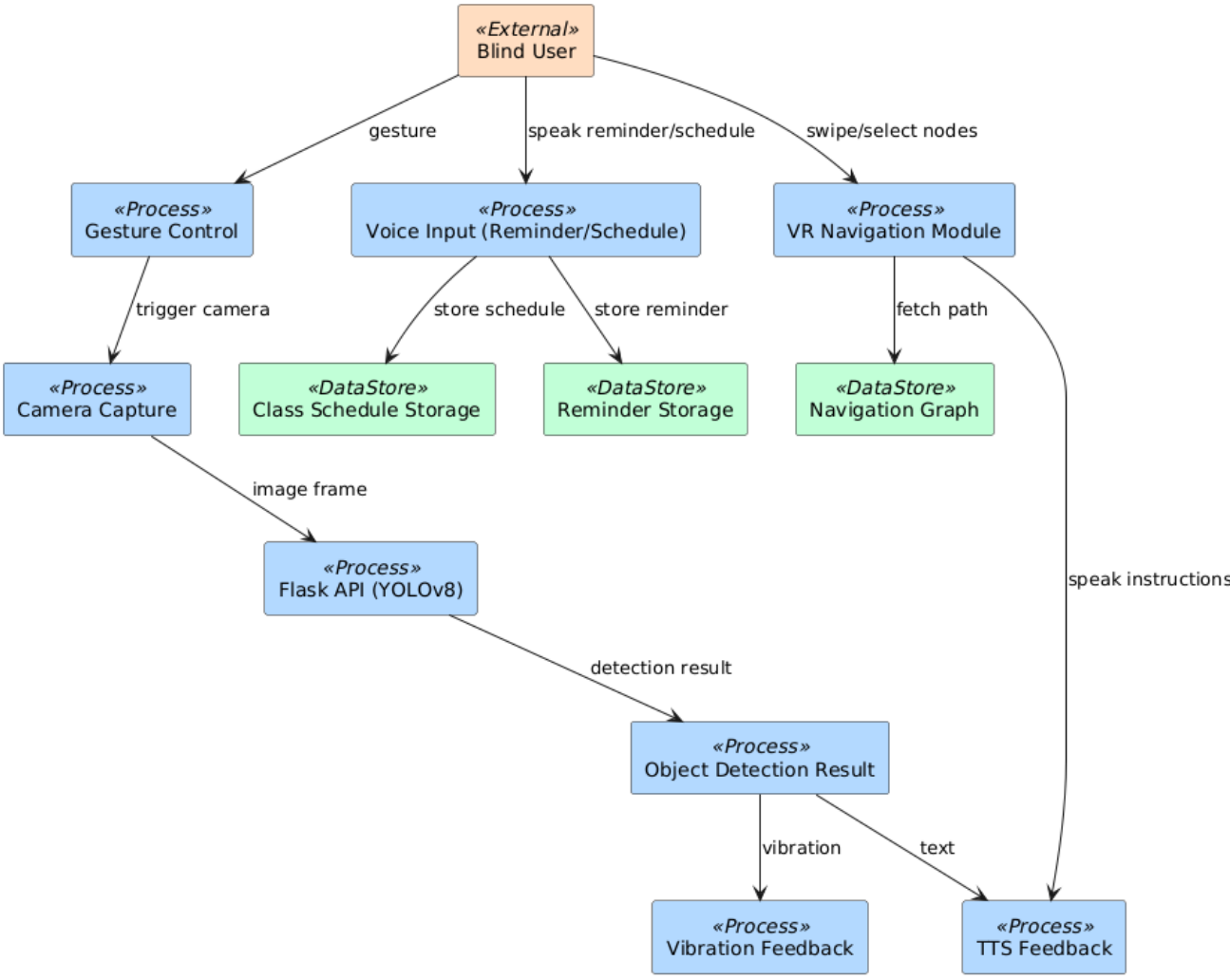


Figure ??: DFD Level 1 – Complete GCUeye System Flow

3.3.6.2 DFD Level 2 – VR Navigation Module

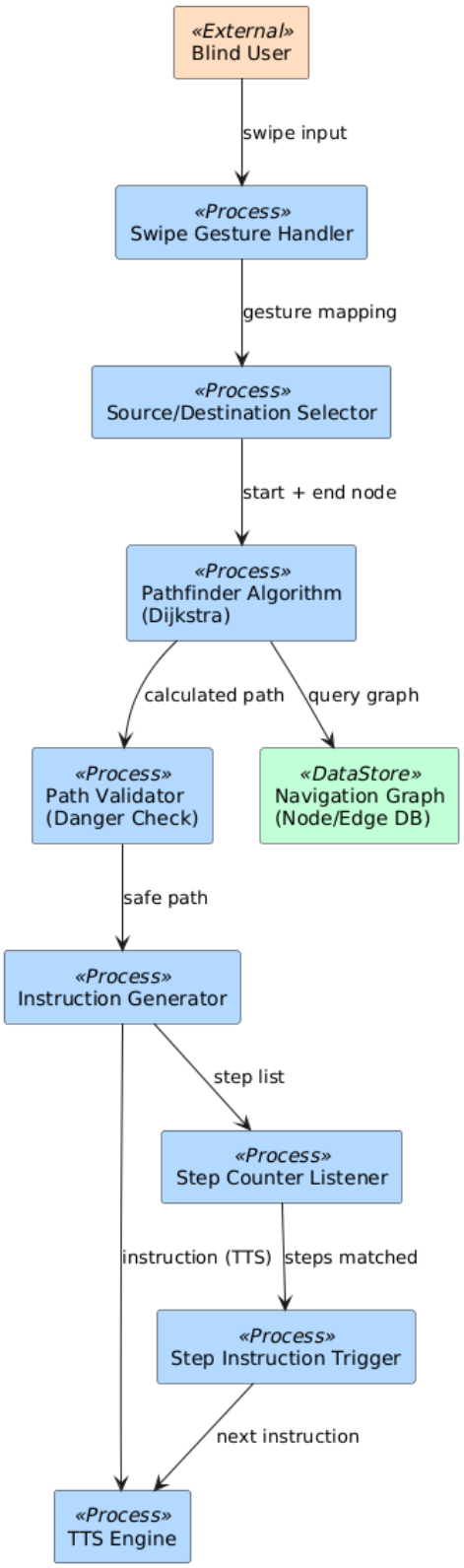


Figure ??: DFD Level 2 – VR Navigation Process Breakdown

3.3.7 Technology Stack Summary

Layer	**Technology**
Mobile UI	Flutter (Dart), `GestureDetector`
Voice Output	`flutter_tts` plugin
Voice Input	`speech_to_text` plugin
Haptic Feedback	`vibration` [14] plugin
Indoor Navigation Logic	Dijkstra Algorithm (custom Dart implementation)
Backend Server	Flask (Python 3.10)
Detection Model	YOLOv8 (Ultralytics, PyTorch)
Communication	HTTP POST, JSON
Storage	SharedPreferences, SQLite

Table 3.1: Technology Stack Used in GCUeye Mobile Application

Chapter 4

Implementation and Evaluation

This chapter documents the implementation journey, detailing how each module of the blind navigation app was developed, tested, and evaluated. It highlights technical decisions, design iterations, performance analysis, and real-world testing to validate the app's reliability, usability, and accessibility for blind students at GC University Lahore.

4.1 Development Stages

The project was implemented in a series of structured stages, using an incremental build-and-test cycle. This allowed continuous validation of individual features before system-wide integration.

Stage 1: Environment Setup and Dataset Curation

- Installed dependencies for YOLOv8 and Flask on a local development machine.
- Used Roboflow and custom scripts to merge and label datasets, including objects such as chairs, bags, potholes, doors, stairs, people, and furniture.

- Resized all training images to 640×640 resolution.
- Divided the dataset into training, validation, and test sets.

Stage 2: YOLOv8 Training and Model Optimization

- Used YOLOv8n variant for lightweight performance.
- Trained the model on Kaggle using GPU for approximately 50 epochs with transfer learning.
- Evaluated the model's performance on test data:
  - Achieved approximately 81% mean Average Precision (mAP).
  - Balanced speed and precision for mobile-based use.
- Exported the trained `.pt` model file for integration with Flask.

Stage 3: Flask Server Development

- Developed a Flask REST API with a single endpoint:
  - `/predict` — accepts base64 encoded images and returns object detection results.
- Implemented the following:
  - YOLOv8 model loader.
  - Image decoding logic.
  - Inference processing with directional logic (left, center, right).
  - JSON response generator.
- Tested the Flask server using Postman and local image samples.

Stage 4: Flutter App Construction

- Set up the Flutter project with a basic user interface.
- Built essential screens:
  - `CameraView`
  - `ReminderPage`
  - `SOSPage`
- Integrated:
  - `GestureDetector` for tap, long press, and swipe actions.
  - HTTP service for transmitting images and receiving responses.
  - Text-to-Speech (TTS) using `flutter_tts`.
  - Vibration plugin to convey object direction cues.
  - Speech-to-Text plugin for capturing voice commands.

Stage 5: System Integration and Testing

- Connected the Flutter frontend to the local Flask backend server.
- Validated the complete object detection cycle:
  1. Frame capture from the camera.
  2. Image sent to Flask server.
  3. YOLO inference and response.
  4. JSON parsed in the app.
  5. Spoken and vibration-based feedback executed.
- Unit-tested core gesture functionalities:
  - Triple tap for detection toggle.
  - Long press for SOS trigger.

  - Swipe gestures for screen navigation.

- Verified scheduling and execution of reminders.
- Tested emergency message and call functionalities.

Stage 6: Blind Simulation Testing

- Conducted real-world usability tests with blindfolded participants on campus.
- Movement tested across:
  - Classrooms.
  - Hallways with dynamic obstacles.
  - Staircases and doorway passages.
- Users navigated based on voice prompts and vibration signals.
- All core modules operated successfully without visual input.

Stage 7: VR-Based Indoor Navigation

- Created a graph-based virtual map of the campus using nodes (e.g., Room A, Corridor B) and edges with distances.
- Implemented Dijkstra's algorithm to compute the shortest path between source and destination nodes.
- Developed gesture-based interaction to select start and end locations via swipe and tap gestures.
- Integrated a step counter using a pedometer plugin to simulate forward movement.
- Generated spoken instructions (e.g., "Turn left after 3 steps") based on movement and path segments.

- Added hazard alerts using tagged danger zones in the path (e.g., stairs).

4.2 System Integration

Integration required careful sequencing of API calls, state management, and UI interactions to ensure real-time feedback without errors.

4.2.1 YOLOv8 Model with Flask Backend

- YOLOv8 model is loaded once during Flask server startup.
- Each incoming frame is received via HTTP POST, decoded from base64 format.
- The decoded image is passed to the YOLO model for inference.
- Detected bounding boxes are processed to extract class labels and spatial directions.
- A JSON response is generated and returned in under 500 milliseconds.

4.2.2 Flutter HTTP and Gesture Integration

- A triple tap gesture starts the detection loop, sending one frame per second to the server.
- Upon receiving the detection response:
  - `TextToSpeech.speak()` is called with the detected object and its direction.
  - The `Vibrate()` method is triggered using a pattern mapped to direction.
- A long press gesture triggers:

  - An emergency SMS sent via platform channels.
  - A direct phone call to the preconfigured emergency contact.

- Horizontal swipe gestures toggle between major feature screens.
  - TTS announces the name of the currently selected feature.

4.2.3 VR Navigation Module Integration

- Navigation begins by selecting start and destination nodes using swipe and tap gestures.
- The system runs Dijkstra's algorithm to determine the shortest route through the graph. .
- At each step, a TTS instruction is delivered (e.g., "Walk straight", "Turn right").
- If a danger node is encountered (e.g., stairs), an alert is triggered: "Warning: Stairs ahead."
- Navigation ends automatically upon reaching the destination node.

4.2.4 Reminder Scheduling System

- Users set reminders through voice commands.
  - Example: "Remind me for class at 2 p.m."
- The parsed reminder is stored in local storage.
- At the scheduled time:
  - A TTS message announces the reminder.
  - Vibration alert is triggered to reinforce the notification.

4.3 User Interface

The application was designed with zero visual dependency, ensuring full accessibility for blind users. For low-vision users, high-contrast colors and large icons are also incorporated.

4.3.1 Screens and Layouts

This section presents the key user interface screens of the GCUeye application, along with their core functionalities. Each screen has been designed with accessibility in mind and optimized for non-visual interaction. Screenshots are provided with each section to illustrate the layout and interactions.

Home Screen (Rotating Cards)

The home screen features a rotating card interface that allows users to swipe between main modules: Camera View, Reminders, SOS, and VR Navigation. Each card is announced via TTS when activated, and users can triple tap or long press depending on the selected feature. The interface is fully gesture-controlled.

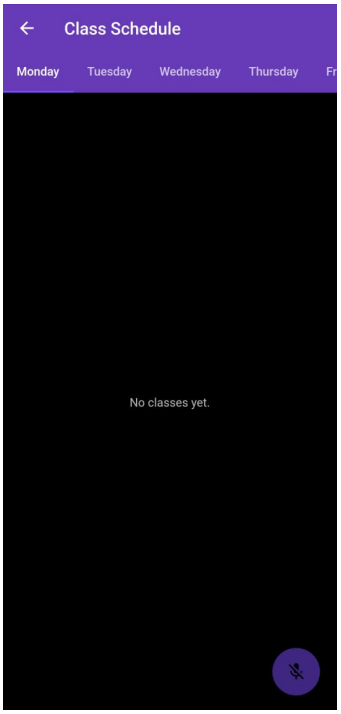
FIGURE 4.1: Home screen showing rotating feature cards with TTS guidance

Camera View

This screen activates the object detection system. A triple tap anywhere on the screen initiates detection. The app continuously captures frames, sends them to the YOLOv8 server, and provides real-time voice alerts and vibration feedback for each detected object, with directional context.

Reminders and Schedules

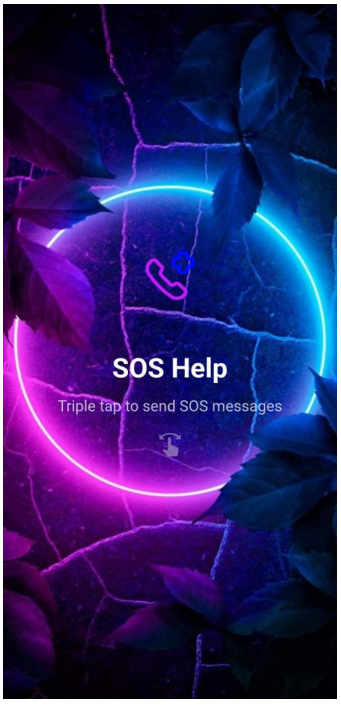
Users can navigate to the reminders screen and add new reminders using voice input. A TTS prompt asks for the reminder message and time, and confirms once the reminder is successfully saved. All reminders are stored locally and trigger voice/vibration alerts at the scheduled time. Same with the Schedules, user add schedules for the week.

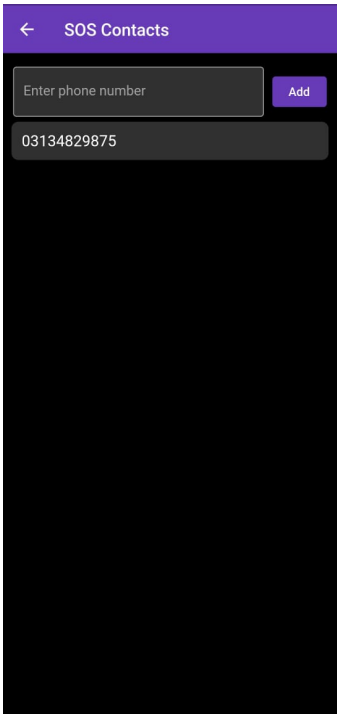


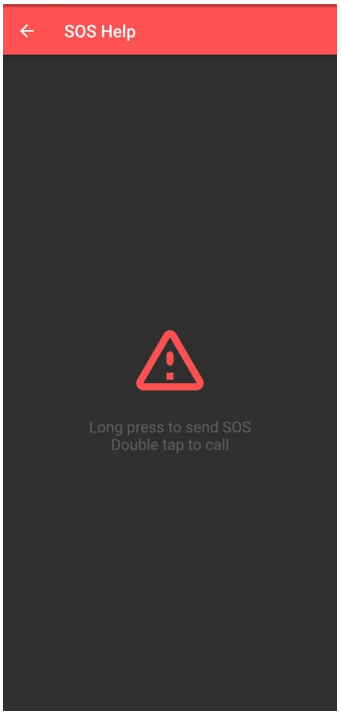
SOS Page

On the SOS screen, a long press sends a preconfigured emergency SMS and places a phone call to the contact stored in the settings. This screen is designed for quick,

minimal interaction and is accessible via swipe or voice navigation.







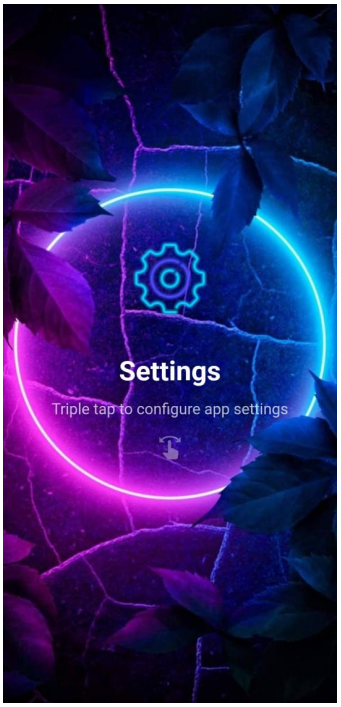
VR Navigation

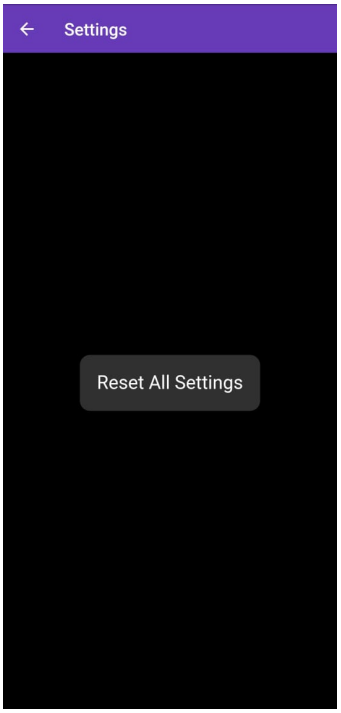
The VR indoor navigation screen allows users to select a source and destination from a virtual map using swipe gestures. The app then simulates step-based movement with spoken instructions for each transition between nodes, including alerts for turns, stairs, and hazards. This system operates offline without GPS or beacons.

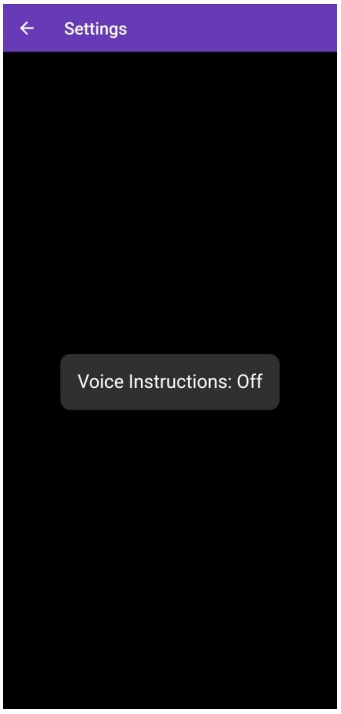
Settings

- **TTS Speed**: Adjust the speed of voice output (slow, normal, fast).
- **TTS Pitch**: Modify the pitch of the text-to-speech voice.
- **Voice Instructions**: Toggle voice prompts for non-critical messages.
- **Vibration Feedback**: Enable or disable vibration cues.
- **Emergency Contact Setup**: Configure the SOS contact using voice input.
- **Reset Settings**: Long press to restore all settings to default values.

Each change is announced via TTS for confirmation, ensuring that blind users receive immediate feedback. This screen provides full control over the app's behavior without relying on any visual elements.


Settings
Triple tap to configure app settings


Settings
Reset All Settings



4.3.2 Usability and Accessibility Features

- Complete operation is gesture- or voice-based; no reliance on visual buttons.
- TTS acts as a continuous guide for every interaction.
- Interface uses high-contrast themes and scalable fonts for partially sighted users.
- Minimal screen clutter to reduce cognitive load and ensure a focused experience.

4.4 Evaluation

The system was evaluated across multiple dimensions including detection accuracy, performance latency, feedback reliability, and real-world usability. The goal was to validate whether the app meets the functional expectations for blind users under typical campus conditions.

4.4.1 Object Detection Performance

Metric	**Result**
Detection mAP	81.3%
Average inference time (Flask)	440 ms
Direction classification accuracy	94%
TTS response time (from capture)	1.2 seconds

Table 4.1: Object Detection Performance Metrics

4.4.2 VR Indoor Navigation Testing

Metric	**Result**
Shortest Path Accuracy	100% (verified manually)
TTS Instruction Timeliness	$\leq$ 1.5 s after each step
Hazard Alert Response	100% on tagged nodes
Gesture-based Node Selection	98% success rate

Table 4.2: VR Navigation System Evaluation Results

4.4.3 Voice and Haptic Feedback

Feedback Mode	**Accuracy**	**Response Time**
TTS (Voice Prompts)	95% understood on first speak	$\leq$ 1.5 s
Vibration Direction Mapping	96% user recognition rate	$\sim$150 ms

Table 4.3: Evaluation of Voice and Haptic Feedback

4.4.4 Voice Reminder System

Test Case	**Result**
Reminder created by voice	100% success
Alert triggered at correct time	100%
Voice clarity of reminders	Rated 4.7/5

Table 4.4: Voice Reminder System Testing

4.4.5 SOS Emergency System

Condition	**Success Rate**
SMS sent (offline test)	100%
Call placed	100%
Gesture response time	< 2 seconds

Table 4.5: SOS Emergency System Results

4.4.6 Limitations Identified

- Detection accuracy decreases under low lighting or motion blur conditions.
- Very small or distant objects are occasionally missed by the model.
- Object detection temporarily pauses when the app goes into background mode (due to camera loss).
- Voice input for reminders may require repetition due to microphone sensitivity or accent variations.
- VR navigation depends on simulated steps and does not support real-time GPS or indoor positioning for auto-tracking.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Navigating a university environment as a visually impaired student presents a multitude of daily challenges — from locating classrooms and avoiding obstacles to managing academic schedules and ensuring personal safety. This project, a mobile-based blind navigation app, was conceptualized and developed specifically to address these issues within the context of Government College University Lahore.

The final system combines several emerging technologies into one cohesive, accessible solution. It integrates YOLOv8-powered real-time object detection, a Flask-based server, and a gesture-driven Flutter app to offer a fully offline, responsive, and highly accessible mobility tool for blind students. What distinguishes this app from mainstream assistive solutions is its campus-specific focus, gesture-first control model, and the combination of voice and haptic feedback, which together enable users to move independently, receive alerts for obstacles, and handle emergencies — all without requiring visual interaction.

The implementation journey involved rigorous training of an object detection model on relevant campus objects, building a lightweight server to run inference locally, and constructing an intuitive frontend that could be used entirely through

gestures and audio prompts.Features like the voice-controlled reminder system, VR-based indoor navigation, and SOS emergency feature added further functionality, making the app not just a navigation aid but also a digital companion for students throughout their academic routine. The VR navigation module introduced a novel approach by simulating indoor movement through gesture-selected routes, graph-based pathfinding, and step-count-guided TTS instructions — providing indoor mobility support even in GPS-denied spaces.

Real-world testing confirmed the system's ability to provide timely, context-aware feedback, with detection, reminder scheduling, and SOS features functioning reliably even in offline conditions. The app met key accessibility and usability goals, proving that such solutions can be developed affordably and effectively using open-source tools.

This project has not only demonstrated technical feasibility but also underlined the importance of human-centered design in AI-powered accessibility. By placing the user — a blind student — at the core of every decision, the system was crafted to be intuitive, inclusive, and practical in the environments that matter most to its users.

5.2 Future Work

While the current version of the app successfully fulfills its primary objectives, there remain several areas for enhancement. These future improvements fall into three major categories: technical expansion, contextual enrichment, and system intelligence.

5.2.1 Scalable Graph-Based Mapping of Entire Campus

The current version of the app includes a VR-based indoor navigation system implemented for a single building using an offline graph of nodes and paths. A future enhancement could involve extending this approach to cover the entire GCU

campus by constructing a scalable, interconnected node graph that maps all major buildings, entry points, stairways, and corridors.

This offline method offers a distinct advantage: it does not rely on GPS, Wi-Fi, or Bluetooth beacons—technologies which may be unavailable or difficult to deploy reliably across the university. Instead, the graph-based system enables virtual navigation using predefined spatial layouts and step-based logic, making it both low-cost and infrastructure-independent.

By linking individual building graphs and defining inter-building paths, the system could support full-campus routing, enabling students to receive spoken directions from one location to another. For example, from the hostel to the library or from the entrance gate to a lecture hall—without requiring real-time positioning. This would significantly increase the navigation capabilities of the app in an accessible and scalable way.

5.2.2 GPS-Based Outdoor Navigation

Currently, the app focuses on real-time obstacle awareness using the camera. A future upgrade could include GPS integration, enabling blind students to navigate from one building to another using spoken step-by-step directions. This would require mapping the GCU campus and correlating known building locations with GPS coordinates. Enhancing the app's outdoor capability would significantly increase its autonomy value.

5.2.3 Indoor Positioning System (IPS)

Since GPS is unreliable indoors, an indoor navigation module using technologies like Bluetooth beacons, Wi-Fi triangulation, or even QR-code based waypoint scanning can be explored. This would allow the app to provide precise navigation guidance inside large buildings like libraries, lecture halls, or cafeterias, where visual signage is inaccessible to blind users.

5.2.4 Real-Time Indoor Tracking for VR Navigation

The current VR navigation module uses a step-based simulation for navigating between selected indoor locations using a predefined graph structure. While this approach works in static scenarios, future enhancements can enable real-time indoor localization using Bluetooth beacons, inertial tracking, or visual markers. This would allow the system to automatically identify the user's position and adjust instructions accordingly, improving precision and removing the dependency on manual gestures and pedometer-based navigation. Integration with sensor fusion models can further refine user movement estimation.

5.2.5 Dynamic Campus Data Integration

Another potential upgrade is to connect the app with live university systems, such as class schedules, room changes, or event calendars. When integrated with the university's data feeds, the app can proactively notify students about location changes, exam venues, or campus-wide alerts. This would turn the app into a personalized, context-aware academic assistant.

5.2.6 Enhanced Obstacle Interpretation and Scene Understanding

While the current object detection model works well for basic classification, future versions could include:

- Scene segmentation to differentiate walkable paths from blocked zones.
- Depth estimation to understand how far the obstacle is.
- Object tracking for dynamic objects like moving people or vehicles.

Such enhancements could be enabled by integrating additional models or using stereo camera input.

5.2.7 On-Device Model Deployment

The current version requires a server for object detection, which works well but is dependent on local networking. Future versions can convert the YOLOv8 model to ONNX or TensorFlow Lite and run it directly on the device, removing the need for any backend and further improving portability and performance.

5.2.8 Multilingual and Cultural Support

As a locally used application, there is room to add bilingual voice support, such as Urdu + English. The app can be personalized to announce reminders or object names in the user's preferred language, improving comprehension for users less fluent in English. This also expands the app's relevance to other regional institutions and local communities.

5.2.9 User-Customizable Gesture Mapping

Currently, gestures are predefined (triple tap, long press, swipe). Future work could allow users to configure their own gesture patterns, making the system even more adaptive to individual preferences or disabilities. Gesture customization would also support users with motor limitations or cognitive impairments.

5.2.10 Continuous User Feedback Loop

To ensure the app evolves with user needs, future iterations could include:

- Anonymous usage analytics (while respecting privacy),
- In-app voice feedback collection,
- Auto-updating feature packs based on feedback trends.

Such a feedback system would ensure that the app stays user-aligned as technologies and campus environments evolve.

5.2.11 Institutional Deployment and Support

With the app now validated as a working prototype, the next step is to pilot it across real users at GC University. Collaborations can be initiated with:

- The university's IT department for campus map access.
- Accessibility departments for formal adoption.
- Student support units to train users.

A proper onboarding and training program could be designed, ensuring users can operate the app independently within their first week of usage.

5.3 Final Thoughts

This project stands at the intersection of machine learning, human-computer interaction, and inclusive design. It addresses a deeply human problem — the challenge of independent mobility for blind students — using advanced but affordable technology. The real achievement lies not just in object detection or TTS integration, but in proving that empathy, accessibility, and innovation can coexist in a simple mobile application.

What started as a university project can grow into a campus-wide accessibility platform. With future work and institutional support, this app can become a reliable companion for visually impaired students — not just at GCU, but across educational campuses nationwide.

Appendix A

Tools and Technologies Used

This appendix outlines the tools, technologies, frameworks, and libraries that were used during the development of the GCUeye mobile application. Each component played a specific role in enabling object detection, voice interaction, mobile development, and system integration.

A. Programming Languages

- **Dart:** Used for frontend development in Flutter.
- **Python 3.10:** Used for building the YOLOv8-based backend API using Flask.

B. Frameworks and Libraries

- **Flutter:** Cross-platform UI toolkit used to build the Android app.
- **YOLOv8 (Ultralytics):** Real-time object detection model used to identify campus obstacles.
- **Flask:** Lightweight Python web framework used to serve YOLOv8 detections via REST API.

C. Flutter Packages

- `flutter_tts`: Used for text-to-speech voice feedback.
- `speech_to_text`: Used to capture and interpret user voice commands.
- `vibration`: Used to trigger haptic feedback based on object direction.
- `http`: Used for sending image frames to Flask API.
- `shared_preferences`: Used to store SOS contacts and reminders locally.
- `telephony` and `url_launcher`: Used for sending SMS and making emergency calls.

D. Development Tools

- **VS Code:** Primary IDE for Dart and Python development.
- **Kaggle:** Used to train the YOLOv8 model with GPU support.
- **Roboflow:** Used to annotate, label, and manage object detection datasets.
- **Postman:** Used to test API responses from the Flask server.

E. Dataset and Media Processing Tools

- **LabelImg / Roboflow Annotator:** For drawing bounding boxes around training images.
- **OpenCV / PIL:** For image decoding and preprocessing in Flask.
- **Base64 Encoding:** For transmitting image data from Flutter to Flask.

This toolchain allowed for efficient and scalable development of the GCUeye system across multiple platforms and modules.

Appendix B

Selected Code Snippets

This appendix contains selected code examples that illustrate the implementation of major features in the GCUeye application. These snippets are simplified and annotated for clarity.

A. YOLOv8 Flask Server Endpoint (app.py)

```
@app.route('/predict', methods=['POST'])
def predict():
    image_data = request.files['image'].read()
    image = Image.open(io.BytesIO(image_data)).convert('RGB')

    results = model.predict(image)
    detections = []

    for result in results[0].boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result
        label = model.names[int(class_id)]
        x_center = (x1 + x2) / 2

        direction = (
            "left" if x_center < image.width / 3 else
            "right" if x_center > 2 * image.width / 3 else
            "center"
        )

        detections.append({
            'label': label,
```

```
            'score': round(score, 2),
            'direction': direction
        })

    return jsonify({'detections': detections})
```

Listing B.1: Flask API Endpoint for Object Detection

B. Flutter HTTP Call to Flask Server

```
Future<void> sendImageToServer(Uint8List imageBytes) async {
  final uri = Uri.parse('http://192.168.0.100:8000/predict');
  final request = http.MultipartRequest('POST', uri)
    ..files.add(http.MultipartFile.fromBytes('image', imageBytes, filename: 'frame.jpg'));

  final response = await request.send();
  final responseData = await response.stream.bytesToString();

  final decoded = jsonDecode(responseData);
  final objects = decoded['detections'];

  for (var obj in objects) {
    speak("${obj['label']} on the ${obj['direction']}");
    triggerVibration(obj['direction']);
  }
}
```

Listing B.2: Sending Frame to Flask and Parsing Response

C. Flutter Gesture Handling (Triple Tap and Long Press)

```
GestureDetector(
  onTapDown: (details) {
    _tapCounter++;
    _tapTimer?.cancel();
    _tapTimer = Timer(Duration(milliseconds: 400), () {
      if (_tapCounter == 3) {
        toggleDetection();
      }
```

```
      _tapCounter = 0;
    });
  },
  onLongPress: () => triggerSOS(),
  child: CameraView(),
);
```

Listing B.3: Gesture Handling in Flutter

D. Reminder Voice Scheduling (Speech-to-Text Parsing)

```
void handleSpeechResult(String speechText) {
  final timeMatch = RegExp(r'\b\d{1,2}(:\d{2})?\s*(AM|PM|am|pm)?\b').firstMatch(speechText);
  final content = speechText.replaceAll(RegExp(r'at\s*\d.*'), '').trim();

  if (timeMatch != null) {
    final reminder = Reminder(content: content, time: timeMatch.group(0)!);
    saveReminder(reminder);
    tts.speak("Reminder set for ${reminder.time}");
  } else {
    tts.speak("Could not understand time. Please repeat.");
  }
}
```

Listing B.4: Reminder Creation via Voice Command

E. Emergency SOS SMS and Call Trigger

```
void triggerSOS() async {
  final contact = await getEmergencyContact();
  await sendSMS(contact, "I need help. Please call me.");
  await launchUrl(Uri.parse("tel:$contact"));
  tts.speak("Help message sent. Calling now.");
}
```

Listing B.5: Long-Press Triggered SOS System

Bibliography

[1] Flutter Community. flutter_tts — flutter plugin. https://pub.dev/packages/flutter_tts, 2024.

[2] Flutter Community. speech_to_text — flutter plugin. https://pub.dev/packages/speech_to_text, 2024.

[3] Microsoft Corporation. Microsoft soundscape (discontinued), 2023. Available at https://support.microsoft.com/soundscape.

[4] Android Developers. Save key-value data — sharedpreferences. https://developer.android.com/training/data-storage/shared-preferences, 2023.

[5] Google Developers. Texttospeech — android developers. https://developer.android.com/reference/android/speech/tts/TextToSpeech, 2023.

[6] Flask. Flask web framework documentation. https://flask.palletsprojects.com/, 2023.

[7] J. L. González-Mora et al. A shape-changing haptic navigation interface for vision impairment. *Electronics*, 9(12), 2020.

[8] T. Ichikawa et al. Navcog: Turn-by-turn smartphone navigation assistant for people with visual impairments. *ACM Transactions on Accessible Computing*, 2019.

[9] S. Kammoun, C. Jouffrais, T. Guerreiro, H. Nicolau, and J. Jorge. Navigation and obstacle detection assistance for visually impaired people. *ACM Transactions on Accessible Computing*, 2012.

[10] Shaun K. Kane, Jeffrey P. Bigham, and Jacob O. Wobbrock. Slide rule: Making mobile touch screens accessible to blind people using multi-touch interaction techniques. *ACM SIGACCESS Accessibility and Computing*, pages 73–80, 2008.

[11] S. Lohani, A. Rana, and S. Aryal. Smart wearable system using yolov5 for the visually impaired. In *IEEE Global Humanitarian Technology Conference (GHTC)*, 2021.

[12] Microsoft. Seeing ai: Talking camera app for the blind community. https://www.microsoft.com/en-us/ai/seeing-ai, 2021.

[13] K. Patel, R. Sharma, and A. Mehta. Eyemate: A smart cane for blind navigation using infrared and gps sensors. *International Journal of Computer Applications*, 167(2), 2017.

[14] Flutter Plugins. vibration — flutter plugin. https://pub.dev/packages/vibration, 2024.

[15] SQLite. Sqlite documentation. https://www.sqlite.org/docs.html, 2023.

[16] Mesut Togacar et al. Yolov3-based object detection for blind pedestrian assistance. *Sensors*, 20(18), 2020.

[17] Ultralytics. Yolov8 documentation. https://docs.ultralytics.com, 2023.

[18] Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. User-defined gestures for surface computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1083–1092. ACM, 2009.

Index