ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

AI Navigation Glasses for the Visually Impaired

Team #45

SHENGNAN CAI JUNCHEN HE
YI SU MINGYAN GAO

TA: Ruolin Zhao

May 16, 2026

Abstract

This report presents the design, implementation, and verification of AI Navigation Glasses for the Visually Impaired, a wearable assistive prototype that provides real-time environmental awareness through computer vision, voice interaction, and spoken navigation prompts. The system uses an ESP32-S3 glasses unit for camera, microphone, speaker, and inertial sensing, while a graphics processing unit (GPU)-capable FastAPI backend performs higher-load perception tasks, including blind-path segmentation, crosswalk detection, obstacle filtering, and traffic-light recognition. A finite-state navigation workflow combines these perception outputs into concise audio guidance for blind-path following and street-crossing assistance.

Verification shows that the integrated prototype can stream camera and audio data, load the required perception models, execute the navigation state machine, block crossing on red-light evidence, and generate prioritized spoken prompts. The primary segmentation model runs in real time on the GPU backend after warmup, while central processing unit (CPU)-only profiling averages 885.3 ms per frame and is therefore insufficient for full-rate navigation. Remaining limitations include incomplete outdoor traffic-light validation, missing local prompt files, Wi-Fi reliability under extended field use, and the absence of tests with visually impaired participants. The prototype demonstrates the feasibility of a modular wearable navigation aid, but it should be treated as an engineering prototype rather than a deployment-ready mobility device.

Contents

1 Introduction

1.1 Problem Statement and Motivation

Visually impaired individuals face significant challenges in achieving safe and independent mobility because they lack continuous real-time awareness of their surrounding environment. Everyday navigation tasks such as following tactile paving (blind paths), avoiding pedestrians or obstacles, locating crosswalks, and interpreting traffic signals require persistent situational understanding that is difficult to obtain without visual feedback.

Traditional mobility aids such as white canes provide limited environmental perception and only detect nearby obstacles through physical contact. More advanced electronic travel aids often suffer from excessive auditory feedback, high latency, or poor adaptability to dynamic user motion. In practice, visually impaired users continuously move their heads while walking, causing wearable cameras to experience vibration, viewpoint drift, and rapid orientation changes. These effects significantly degrade the stability of computer vision outputs and may lead to incorrect navigation guidance.

In addition, most existing systems focus only on isolated perception tasks such as object detection or scene captioning. Few systems integrate environmental understanding, obstacle awareness, blind-path following, traffic-light recognition, and real-time user interaction into a unified wearable platform capable of operating continuously in outdoor environments.

To address these limitations, this project proposes an AI-based wearable visual navigation system implemented as an assistive smart glasses platform. The system combines real-time computer vision, multimodal AI interaction, voice feedback, and inertial sensing to provide stabilized navigation guidance for visually impaired users. Rather than overwhelming the user with raw sensory information, the system selectively extracts critical environmental cues and delivers concise, prioritized guidance through low-latency audio feedback.

The primary design goal of this project is to bridge the gap between environmental perception and real-time human navigation by integrating visual sensing, motion stabilization, and intelligent feedback into a single wearable assistive system. The final platform is intended to help users better understand their surroundings, maintain alignment with tactile paths, avoid hazards, and safely perform street-crossing tasks in real-world environments.

1.2 Proposed Solution and Core Functionalities

The proposed system is a wearable smart-glasses platform with embedded sensing hardware, a GPU-accelerated FastAPI backend, and a real-time audio feedback pipeline. The ESP32-S3 unit captures camera, microphone, speaker, and motion data, sends the streams over WebSocket or HTTP links, and leaves high-load AI inference to the backend [1, 2]. The main user-facing functions are voice command handling, scene description and visual question answering, blind-path guidance, obstacle warnings, crosswalk assistance, traffic-light recognition, low-latency speech feedback, and a browser monitor for debugging. Together these functions form a unified assistive perception platform rather than a collection of isolated detectors; the detailed architecture is described in Section 2.3 and Figure 1.

1.3 Subsystem Overview

At the top level, the project is divided into six connected subsystems: wearable sensing hardware, embedded firmware, wireless data transport, backend AI perception, navigation decision logic, and audio/user interaction. Figure 1 gives the top-level block diagram and shows the interconnects among these subsystems. The ESP32-S3 glasses act as the wearable front end, capturing images, audio, and IMU data and sending them to the backend through WebSocket streams. The backend decodes the streams, runs the vision and audio models, updates the navigation state machine, and returns spoken prompts through the WAV downlink. The browser monitor is not part of the user-facing assistive path, but it is connected to the

same backend state so that the team can verify masks, detections, prompts, and latency during tests. Table 1 summarizes the function and main interconnects of each subsystem.

Table 1: Subsystem overview and interconnect summary.

Subsystem	Purpose	Main interconnects
Wearable hardware	Camera, microphone, speaker, IMU, amplifier, battery, and glasses frame provide the physical sensing and feedback platform.	ESP32-S3 GPIO, I2S, PDM, camera connector, power rails
Embedded firmware	Schedules camera capture, audio upload, TTS playback, WiFi reconnection, and timing metadata.	`/ws/camera`, `/ws_audio`, `/stream.wav`, IMU JSON messages
Backend bridge	Receives live streams, maintains latest-frame buffers, and broadcasts debug state.	FastAPI WebSockets, HTTP WAV stream, browser monitor sockets
AI perception	Segments blind paths and crosswalks, detects traffic-light state, and filters navigation-relevant obstacles.	YOLO segmentation outputs, YOLO-E obstacle masks, traffic-light class history
Navigation logic	Converts perception outputs into a finite-state navigation workflow and prioritizes safety prompts.	Crosswalk geometry, path centerline, obstacle overlap, stable light count
Voice interaction	Routes voice commands to navigation modes and plays concise spoken feedback.	Paraformer ASR, command parser, prompt queue, I2S speaker output

1.4 High-Level Requirements

To ensure that the wearable navigation system provides reliable real-world assistance, the project defines the following high-level performance requirements. Each requirement is written with a measurable tolerance so that the verification results in Section 3 can be evaluated consistently.

1. **Perception Accuracy**

   The system shall identify blind paths, crosswalk regions, navigation-relevant obstacles, and traffic-light state on representative outdoor camera frames. For a labeled field-test set, blind-path/crosswalk segmentation and obstacle relevance decisions shall achieve at least 90% frame-level agreement with manual labels, while traffic-light state classification shall achieve at least 85% precision on visible signal frames.

2. **Real-Time Responsiveness**

   The camera pipeline shall maintain the configured $5\,\mathrm{fps}$ stream rate within ± 1 fps during controlled integration tests. On the GPU backend, primary blind-path/crosswalk segmentation shall complete in less than $100\,\mathrm{ms}$ after model warmup, and safety-critical spoken prompts shall be issued within $1000\,\mathrm{ms}$ of the triggering navigation event.

3. **Navigation Stability and Reliability**

   The navigation workflow shall prevent rapid prompt oscillation during head motion by using mask smoothing, centerline history, optical-flow tracking, and state-transition hysteresis. Consecutive stabilized blind-path centerline estimates should remain within $10\,\mathrm{px}$ under moderate walking motion, and traffic-light crossing decisions shall require at least five stable green frames before entering the crossing state.

4. **Audio and Voice Interaction**

   The system shall support hands-free navigation commands, including start navigation, stop navigation, cross-street mode, and traffic-light detection. At least 90% of predefined local prompts shall be

available for immediate playback, and missing prompt files shall be reported before demonstration.

5. **Fail-Safe Crossing Behavior**

   The system shall never instruct the user to cross from a single-frame traffic-light result. Red or yellow states shall block crossing immediately, and any development-only no-light fallback shall be documented as unresolved for deployment.

1.5 Block-Level Design Changes

Several block-level changes were made during the semester as testing exposed the practical limits of the first prototype. The camera block was upgraded from the original low-resolution module to an OV5640-class $5\,\mathrm{MP}$ module because the early image stream did not preserve enough detail for distant tactile-path and crosswalk masks. The audio block was revised by replacing the first speaker with a louder unit driven by the MAX98357A amplifier so that navigation prompts could be heard during walking tests. The communication block was also changed: instead of allowing camera and audio uploads to compete equally for Wi-Fi time, the firmware gives audio higher scheduling priority and drops stale camera frames through a latest-frame queue. These changes preserved the original high-level architecture while improving sensing quality, prompt audibility, and real-time behavior.

2 Design

2.1 Equations and Simulations

2.1.1 Camera Streaming

The ESP32-S3 camera is configured at a target of 5 frames per second (fps), giving a nominal frame period of

$$T_{\text{frame}} = \frac{1000\,\text{ms}}{\text{FPS}} = \frac{1000}{5} = 200\,\text{ms}. \tag{1}$$

A single-depth FreeRTOS queue ensures only the newest frame is processed; stale frames are dropped rather than queued. Each transmitted packet carries timing metadata embedded in a binary header:

$$t_{\text{cap-to-send}} = t_{\text{send\_start}} - t_{\text{capture}}, \tag{2}$$

$$t_{\text{queue-wait}} = t_{\text{send\_start}} - t_{\text{enqueue}}. \tag{3}$$

These values are logged by the backend to profile per-frame end-to-end latency without requiring a separate instrumentation framework.

2.1.2 Audio Sampling

The microphone uplink transmits 16 kHz, 16-bit mono PCM in 20 ms chunks:

$$B_{\text{up}} = f_s \times t_{\text{chunk}} \times 2\,\tfrac{\text{bytes}}{\text{sample}} = 16{,}000 \times 0.020 \times 2 = 640\,\text{bytes}. \tag{4}$$

The speaker downlink streams 8 kHz mono PCM16 WAV:

$$B_{\text{down}} = 8{,}000 \times 0.020 \times 2 = 320\,\text{bytes per 20 ms}. \tag{5}$$

Before output to the MAX98357A I2S amplifier the firmware converts mono 16-bit samples to stereo 32-bit by left-shifting and duplicating:

$$s_{32} = \text{gain} \times (s_{16} \ll 16), \qquad s_{\text{left}} = s_{\text{right}} = s_{32}. \tag{6}$$

2.1.3 Segmentation Mask Geometry

YOLO segmentation probability maps are binarised at threshold 0.5:

$$m(x,y) = \begin{cases} 1 & \text{if } p(x,y) > 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The following geometric quantities are derived from the binary mask and used throughout all navigation modules (image size $H \times W$):

$$\rho_{\text{area}} = \frac{\#\{(x,y) : m = 1\}}{H \times W}, \tag{8}$$

$$\bar{x}_{\text{centre}} = \frac{\sum\limits_{m=1} x}{\#\{m=1\}} \Big/ W, \tag{9}$$

$$\bar{y}_{\text{bot}} = \frac{\max\limits_{m=1} y}{H}, \qquad \rho_{\text{height}} = \frac{\max\limits_{m=1} y - \min\limits_{m=1} y + 1}{H}. \tag{10}$$

2.1.4 Blind-Path Centerline and Guidance

For each sampled image row y, the horizontal span of the tactile-path mask gives:

$$x_c(y) = \frac{x_{\min}(y) + x_{\max}(y)}{2}, \qquad w(y) = x_{\max}(y) - x_{\min}(y). \tag{11}$$

A width-weighted second-order polynomial is fitted to the centerline samples:

$$x(y) = ay^2 + by + c, \qquad \text{weights} = w(y). \tag{12}$$

Coefficients from successive frames are temporally smoothed with exponentially decaying history weights $w_i = 0.7^{N-i-1}$ (normalised) to reduce frame-to-frame jitter.

Heading error and path curvature are derived analytically from the fit:

$$\kappa = |a|, \qquad \theta_{\text{err}} = \arctan\big(2a\, y_H + b\big), \tag{13}$$

where y_H is the bottom image row index. A curvature-dependent lookahead point controls how far ahead the guidance target is placed:

$$y_{\text{target}} = \begin{cases} 0.60\, H & \kappa > 5 \times 10^{-5}, \\ 0.40\, H & \text{otherwise.} \end{cases} \tag{14}$$

The lateral offset of the lookahead point from the image centre is

$$\delta = \frac{|x(y_{\text{target}}) - W/2|}{W}. \tag{15}$$

Guidance commands are issued according to the following priority hierarchy:

$$\text{command} = \begin{cases} \text{turn left} & \theta_{\text{err}} > +10^\circ, \\ \text{turn right} & \theta_{\text{err}} < -10^\circ, \\ \text{adjust laterally} & \delta > 0.15, \\ \text{straight ahead} & \text{otherwise.} \end{cases} \tag{16}$$

2.1.5 Turn Detection

Sharp turns in the tactile path are detected by comparing local tangent directions computed over two short windows along the fitted centerline:

$$\Delta\theta = \big|\theta_{\text{front}} - \theta_{\text{back}}\big|, \qquad \theta = \operatorname{atan2}(\Delta x,\, \Delta y). \tag{17}$$

A row is marked as a turn candidate when $\Delta\theta > 30^\circ$. A secondary confirmation, based on straight-line fits before and after the candidate point, requires $\Delta\theta > 45^\circ$ before a turn prompt is issued to the user.

2.1.6 Crosswalk Proximity and Crossing

The system transitions to crosswalk-waiting mode when all three spatial conditions are met simultaneously:

$$\rho_{\text{area}} \geq 0.20, \quad \bar{y}_{\text{bot}} \geq 0.80, \quad \rho_{\text{height}} \geq 0.35. \tag{18}$$

Directional alignment during far approach is estimated by Principal Component Analysis (PCA) on the crosswalk mask pixels; the dominant eigenvector angle ϕ and lateral offset δ are thresholded at $|\phi| < 15^\circ$ and $|\delta| < 0.20$ respectively.

During active crossing, Canny edge detection and Hough line fitting estimate the stripe direction in each frame. Multiple Hough line estimates are fused with weighted circular averaging to avoid wrap-around errors:

$$\bar{\phi} = \operatorname{atan2}\Big(\sum_i r_i \sin\phi_i,\ \sum_i r_i \cos\phi_i\Big), \tag{19}$$

where r_i is the Hough vote count for line i.

Crossing completion is declared when the crosswalk has moved mostly out of the camera field of view:

$$\bar{y}_{\text{top}} > 0.70 \ \wedge \ \bar{y}_{\text{bot}} > 0.85 \ \wedge \ \rho_{\text{area}} < 0.08. \tag{20}$$

If the crosswalk mask disappears entirely for more than 10 seconds, crossing completion is also announced as a fallback.

2.1.7 Obstacle Detection

YOLO-E obstacle detections pass the acceptance filter when confidence ≥ 0.25 and the obstacle mask overlaps the navigable-path mask:

$$A_{\text{intersect}} \geq 100\,\text{px} \quad \text{and} \quad \frac{A_{\text{intersect}}}{A_{\text{obstacle}}} \geq 0.01. \tag{21}$$

Detections where the mask covers more than 70% of the image area are rejected as false positives (e.g. a mislabelled ground plane). Consecutive-frame stability is maintained with Lucas-Kanade optical flow; the current and warped previous masks are blended as:

$$\hat{m} = 0.8\, m_{\text{curr}} + 0.2\, m_{\text{prev,warped}}. \tag{22}$$

2.1.8 Traffic-Light Detection

The YOLO traffic-light model uses a confidence threshold of 0.25. A detected state is considered stable when the same valid class appears in at least 2 of the last 5 frames (3 of 5 in standalone monitoring mode). The system will not initiate crossing until the green state has been continuously stable for:

$$N_{\text{green}} \geq 5 \text{ consecutive stable frames.} \tag{23}$$

An HSV colour fallback classifies the dominant colour in the detected bounding-box region using fixed hue/saturation/value ranges. The minimum pixel-ratio threshold for any colour class is 0.03.

2.1.9 Offline Profiling Simulation

The script `yolo_seg_profile.py` benchmarks YOLO segmentation outside the live WebSocket loop. Table 2 reports results from a 3-run CPU profiling session on a representative camera frame (480×640, no CUDA available).

Table 2: YOLO segmentation CPU inference profiling (3 runs, 1 warmup run).

Metric	**Value**
Hardware	CPU only (CUDA unavailable)
Input resolution	$480 \times 640 \times 3$
Average latency	885.3 ms
Minimum latency	828.0 ms
Median (p_{50})	859.0 ms
90th percentile (p_{90})	947.0 ms
Maximum latency	969.0 ms
Effective throughput	≈ 1.1 fps

The camera target rate is 5 fps (200 ms per frame), so running full segmentation on every frame in CPU-only mode is slower by a factor of:

$$\frac{885.3\,\text{ms}}{200\,\text{ms}} \approx 4.4\times . \tag{24}$$

This result quantitatively motivates the hybrid inference schedule described in Section 2.2: crosswalk segmentation runs every 4 frames, obstacle detection every 15 frames, and optical-flow mask tracking fills the gaps between heavy model calls. GPU acceleration is available through the `AIGLASS_DEVICE` environment variable and automatic mixed precision (AMP); the profiling session used a CPU-only host for portability.

2.2 Design Alternatives

2.2.1 ESP32 Local Inference vs. Server-Side Inference

An early design alternative was to run all AI models directly on the ESP32-S3. The blind-path segmentation model occupies approximately 144 MB and the traffic-light model approximately 175 MB—both far exceeding the 8 MB PSRAM available on the ESP32-S3. The selected design offloads all YOLO inference, ASR, and language-model interaction to a Python FastAPI backend, keeping the firmware lightweight and deterministic. This split also allows optional GPU acceleration on the backend host with no firmware changes, and makes it straightforward to swap or upgrade models independently of the embedded system.

2.2.2 Raw Video Stream vs. JPEG WebSocket

Sending uncompressed 640×480 YUV frames would require approximately 460 MB/s of WiFi bandwidth, which is infeasible on a 2.4 GHz link shared with audio traffic. The selected design JPEG-encodes each frame on the ESP32 hardware encoder (quality setting 12) and transmits it over a binary-framed WebSocket (`/ws/camera`) together with a compact timing metadata header. A queue depth of one ensures the backend always processes the most recent frame rather than a backlog of stale images. The connection is automatically re-established when a single send stalls for more than 300 ms, preventing silent hangs.

2.2.3 YOLO Traffic-Light Detector vs. HSV Colour Heuristic

A pure HSV colour heuristic was evaluated as a lightweight alternative for traffic-light state detection. The heuristic can confuse bright advertising signs, vehicle headlights, and reflective surfaces with traffic lights because it lacks spatial context and object localisation. The selected design uses the YOLO `trafficlight.pt` model as the primary detector to obtain class-level localisation, and falls back to the HSV heuristic only when the model produces no valid output. Multi-frame majority voting (requiring 2 of 5 consecutive frames to agree) further reduces false positives from both pathways. The system never initiates road crossing on a single-frame detection.

2.2.4 Summary of Design Issues and Corrective Actions

Three design issues were corrected during development. First, the initial speaker was too quiet outdoors, so it was replaced with a higher-output unit compatible with the MAX98357A amplifier. Second, the OV2640 camera did not provide enough detail for tactile-path segmentation at distance, so the hardware was upgraded to an OV5640-class 5 MP module and the firmware pin map was updated. Third, simultaneous PCM16 audio and JPEG camera uploads caused Wi-Fi contention; audio was given priority in the FreeRTOS scheduler and camera send operations were deferred when the audio queue was active.

2.3 Design Description and Justification

The system is decomposed into six independent modules that communicate through well-defined network and function-call interfaces. This modularity allows each subsystem to be developed, tested, and replaced independently without affecting the others.

2.3.1 Hardware and Firmware Module

The ESP32-S3 firmware (`compile/compile.ino`, `camera_pins.h`) manages three concurrent FreeRTOS tasks: camera capture and send, microphone capture and upload, and TTS audio playback. Separating these into independent tasks with dedicated queues prevents any single operation from blocking the others. Camera frames are JPEG-encoded by the ESP32 hardware encoder, reducing WiFi load by approximately $30\times$ compared with raw YUV transmission. Per-frame timing metadata embedded in each camera packet allows the backend to compute decode, inference, encode, and broadcast latencies at runtime without a separate instrumentation layer.

2.3.2 Backend I/O Bridge

`app_main.py` and `bridge_io.py` host the FastAPI server and maintain a latest-frame buffer so that AI modules always operate on the most recent image regardless of how long inference takes. Runtime metrics (receive fps, navigation-used fps, per-stage latency) are broadcast to connected browser monitors in real time, enabling debugging without interrupting the navigation loop.

2.3.3 Navigation Orchestrator

`navigation_master.py` implements an eight-state FSM, illustrated in Figure 2, covering chat, blind-path navigation, crosswalk seeking, traffic-light waiting, active crossing, next-path search, recovery, and standalone traffic-light detection. State transitions require repeated evidence before committing; for example, five stable green-light frames are required before entering CROSSING. A TTS cooldown timer prevents overlapping audio prompts.

2.3.4 Blind-Path Navigation Module

`workflow_blindpath.py` segments the tactile ground surface with the YOLO blind-path model (every 4 frames), fits the width-weighted polynomial centerline described in Section 2.5, and derives heading error θ_{err} and lateral offset δ. Obstacle detections (YOLO-E, every 15 frames with optical-flow inter-frame tracking) are filtered to those whose masks intersect the computed walking path, suppressing warnings about objects alongside rather than ahead of the user. Guidance thresholds (10° heading, 15% lateral offset, $30^\circ/45^\circ$ turn angles) were tuned through on-device testing to balance responsiveness against false-alarm rate [3].

2.3.5 Cross-Street Navigation Module

`workflow_crossstreet.py` manages the complete crossing sequence: detecting crosswalk proximity via the three-threshold gate, aligning the user using PCA or Hough-line angle estimates, waiting for stable green, monitoring stripe direction and obstacles during crossing, and detecting completion from mask geometry. The three-threshold proximity gate ensures the crosswalk is genuinely close and fills a substantial portion of the frame before the user is directed to stop and prepare to cross. Tighter alignment thresholds ($|\phi| < 5^\circ$, $|\delta| < 0.08$) are applied once crossing has begun, keeping the user perpendicular to the road even as the view angle changes mid-crossing.

2.3.6 Traffic-Light Module

`trafficlight_detection.py` loads `model/trafficlight.pt` and applies majority voting over a five-frame sliding window. Red and yellow states always block crossing regardless of elapsed wait time. If no valid detection is produced for a configurable number of frames, a cautious fallback (based on elapsed waiting time and HSV colour) is applied. The YOLO detector and HSV heuristic are complementary: YOLO provides object-level localisation, while HSV provides robustness to model failure on unusual or non-standard light fixtures.

2.3.7 Obstacle Module

`obstacle_detector_client.py` uses YOLO-E open-vocabulary segmentation to detect a configurable set of pedestrian hazards (persons, vehicles, poles, cones, animals, bollards, etc.) without requiring a separate trained model for each object class. Only obstacles whose masks overlap the navigable-path mask are forwarded to the FSM, which substantially reduces irrelevant warnings from objects alongside the walking lane.

2.3.8 Voice and AI Interaction Module

Audio travels a dedicated pipeline (Figure 4): PDM microphone → 16 kHz PCM WebSocket → Paraformer ASR → command parser → navigation FSM or Qwen/Omni language model → TTS prompt queue → 8 kHz WAV HTTP stream → I2S speaker. Older messages are dropped from the TTS queue when it fills, ensuring that safety-critical guidance (obstacle alerts, traffic-light state changes) is announced without delay. Voice is the only output channel available to the visually impaired user; all navigation commands are therefore designed to be unambiguous and timed to precede the required action window.

2.4 Subsystem Diagrams and Schematics

2.4.1 System Architecture Overview

Figure 1 shows the top-level system architecture. The ESP32-S3 glasses connect to the FastAPI backend over WiFi via three uplink WebSocket endpoints and one downlink HTTP WAV stream. The backend distributes decoded frames to the AI and navigation layer and streams annotated frames to the browser-based debug monitor.

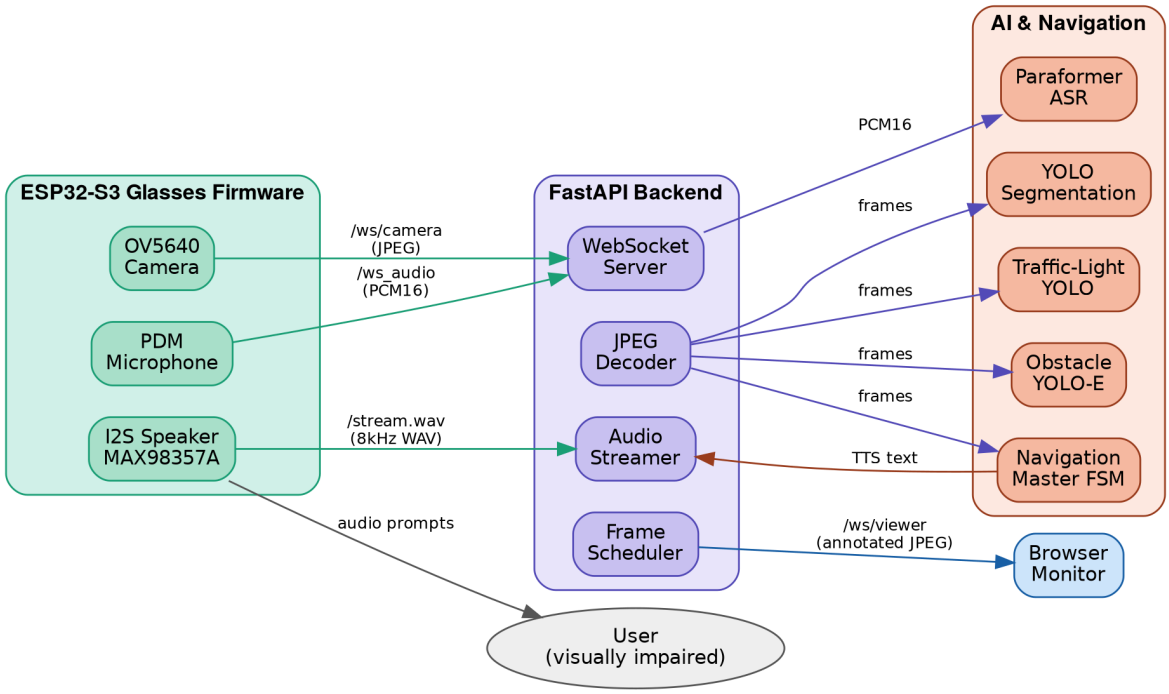

Figure 1: Top-level system architecture. Teal: ESP32-S3 hardware subsystem. Purple: FastAPI backend. Coral: AI and navigation modules. Blue: browser monitor.

2.4.2 Navigation State Machine

Figure 2 illustrates the eight-state FSM in `navigation_master.py`. Solid arrows represent primary navigation transitions; dashed arrows represent optional stop commands and recovery paths.

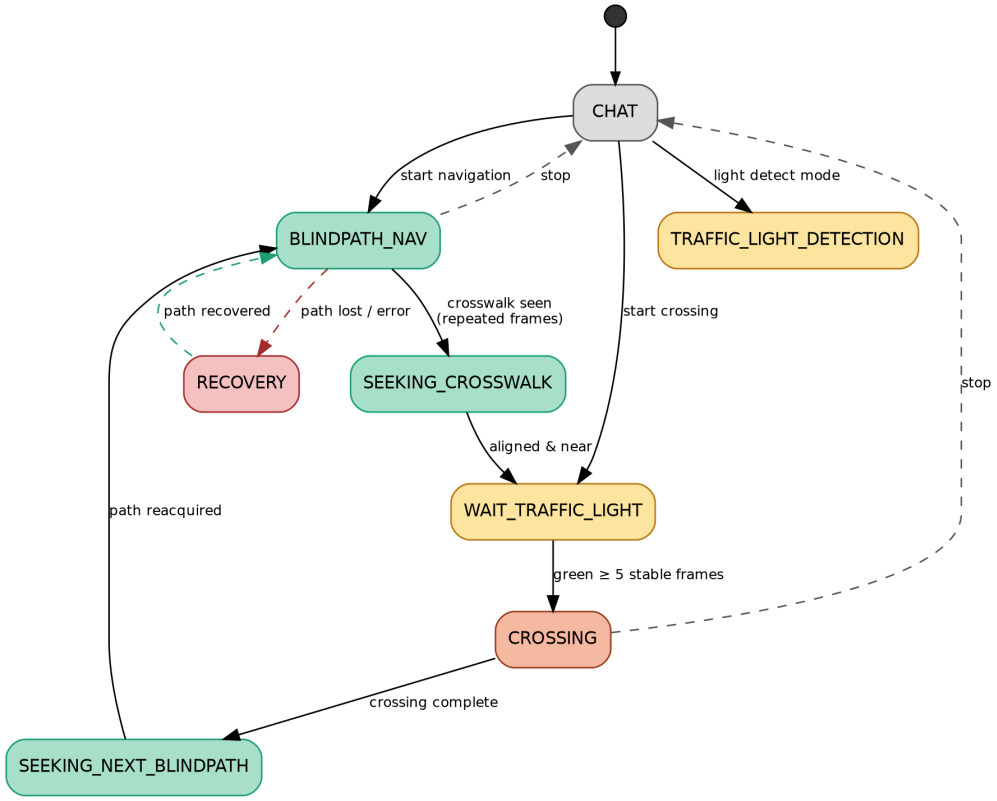

Figure 2: NavigationMaster finite-state machine. Green: navigation flow states. Amber: waiting states. Coral: active road-crossing state. Red: recovery state. Dashed arrows: optional or recovery transitions.

2.4.3 Vision Processing Pipeline

Figure 3 shows the per-frame vision processing pipeline. YOLO segmentation runs every 4 frames; obstacle detection runs every 15 frames with optical-flow mask tracking filling the intervening frames.

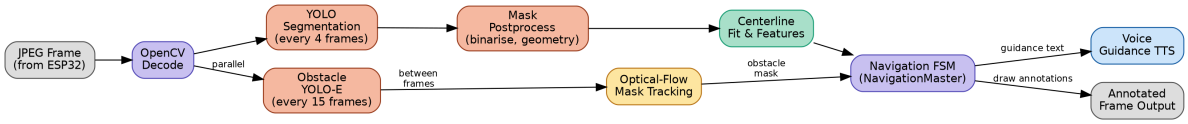

Figure 3: Vision processing pipeline from JPEG frame input to annotated frame output and voice guidance. Parallel branches handle path segmentation (top) and obstacle detection with inter-frame tracking (bottom).

2.4.4 Audio Pipeline

Figure 4 shows the complete audio pipeline from the PDM microphone through Paraformer ASR to the I2S speaker.

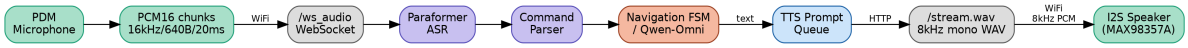

Figure 4: Audio pipeline. Uplink (top row): PDM microphone → PCM16 → `/ws_audio` WebSocket → Paraformer ASR → command parser → navigation FSM. Downlink (bottom row): TTS queue → `/stream.wav` HTTP WAV → I2S speaker.

2.4.5 Hardware Wiring Schematic

Figure 5 shows the hardware wiring between the XIAO ESP32-S3 and the MAX98357A I2S audio amplifier (right, I2S interface). The colour-coded wires correspond to power (red/black) and I2S audio lines (yellow/blue/green).

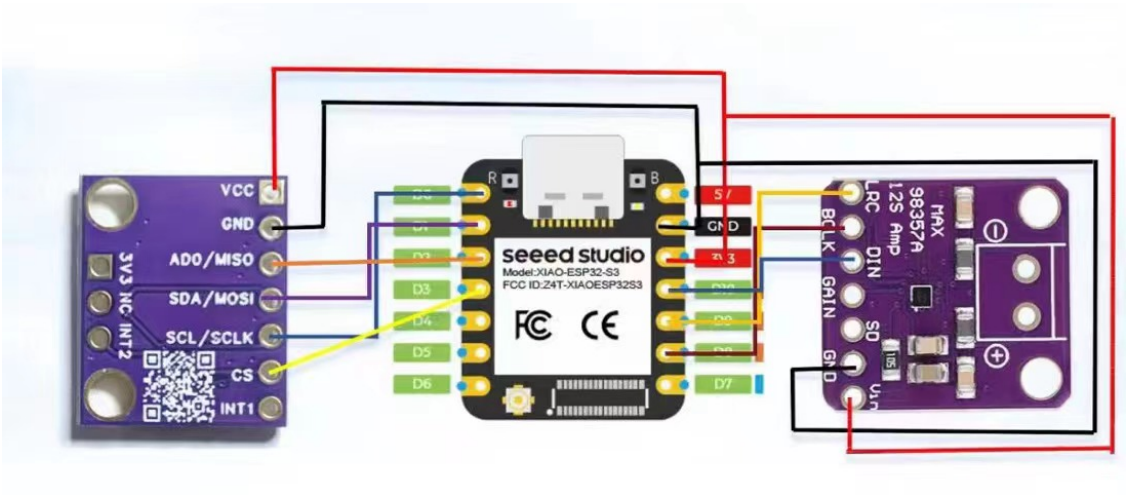

Figure 5: Hardware wiring schematic. Centre: XIAO ESP32-S3 (Seeed Studio, Model XIAO ESP32S3). Right: MAX98357A I2S amplifier connected via I2S (BCLK, LRC, DIN). Red/black lines carry 3.3 V power and ground respectively.

2.4.6 Network Endpoint Summary

Table 3 lists all network endpoints used between the ESP32-S3, backend, and browser monitor.

Table 3: Network endpoints.

Direction	Endpoint	Protocol	Payload
ESP32 → backend	`/ws/camera`	WebSocket	JPEG + timing header
ESP32 → backend	`/ws_audio`	WebSocket	PCM16, 16 kHz
Backend → ESP32	`/stream.wav`	HTTP WAV	PCM16, 8 kHz mono
Backend → browser	`/ws/viewer`	WebSocket	Annotated JPEG
Backend → browser	`/ws`	WebSocket	UI state

3 Requirements and Verification

The verification plan evaluates the system at the same boundaries used in the design: sensing and communication, AI perception, navigation logic, audio feedback, and total system cost. The tests combine source-level evidence from the final implementation, startup and warmup logs, offline model profiling, and controlled integration tests with the ESP32-S3 camera stream. The final integrated log used for this section contains 185 cross-street debug records, 79 spoken-prompt records, 45 traffic-light state records, and complete backend startup evidence for the segmentation, obstacle-detection, traffic-light, audio, and camera subsystems. Because this system is intended as an assistive device, the verification results distinguish between completed engineering validation and remaining safety limitations.

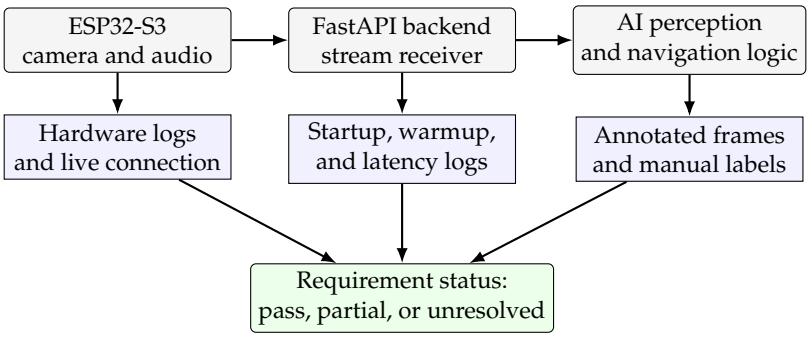

Figure 6: Verification evidence flow used to connect hardware tests, backend timing logs, model outputs, and requirement-level conclusions.

3.1 Sensing and Communication Verification

The sensing subsystem was verified by isolating the camera, microphone, and speaker before running the full navigation stack. Early tests showed that temporary wiring caused intermittent camera behavior, so the hardware connections were soldered before later integration testing. After this correction, the ESP32-S3 firmware successfully initialized the camera, PDM microphone, I2S speaker output, Wi-Fi client, WebSocket camera uplink, WebSocket audio uplink, and HTTP WAV downlink.

The camera firmware uses a latest-frame queue so that old frames are dropped instead of accumulating latency. The microphone uplink sends 16 kHz PCM audio in 20 ms chunks. This verifies that the hardware is capable of supplying the backend with the required camera and audio streams. In the integrated run, the backend repeatedly detected the ESP32 camera and reinitialized the navigator without reloading the full stack, with 26 connection records and 25 disconnection records appearing during extended testing. These reconnects did not prevent the navigation object from being reused, but they show that a longer camera-reliability test remains necessary before claiming sustained outdoor robustness. Table 4 summarizes the sensing and communication tests used to support this requirement.

Table 4: Sensing and communication verification.

Test item	Procedure	Result	Status
Camera streaming	Connect ESP32-S3 to the server and monitor `/ws/camera` frame arrival and backend viewer output.	Camera frames reached the FastAPI backend and browser monitor. Firmware logs frame capture, sent, dropped, and WebSocket failure counts.	Pass, with longer reliability test needed
Audio uplink	Enable microphone stream and confirm 16 kHz PCM chunks reach `/ws_audio`.	Firmware sends 20 ms audio chunks through the audio WebSocket. Backend ASR module receives the stream for command recognition.	Pass
Audio downlink	Start backend WAV stream and confirm ESP32 speaker task parses WAV headers and writes I2S samples to MAX98357A.	The HTTP `/stream.wav` path and I2S output path are implemented. Prompt preloading reduces runtime playback delay.	Pass

3.2 Computer Vision Verification

The computer vision subsystem includes three model pathways. The first pathway uses a YOLO segmentation model for blind-path and crosswalk masks. The second uses YOLO-E open-vocabulary segmentation to detect pedestrian obstacles and filter them by overlap with the navigable path. The third uses a traffic-light detector with temporal voting. The backend successfully loads the blind-path segmentation model on `cuda:0` when a GPU is available, loads YOLO-E with a 29-class obstacle whitelist, and initializes the traffic-light model with the required signal classes.

The strongest measured timing evidence is the contrast between CPU-only profiling and the final GPU startup log. As shown previously in Table 2, CPU-only execution averaged 885.3 ms per segmentation frame and therefore cannot satisfy the real-time requirement alone. In the final integrated run, the blind-path segmentation model loaded on `cuda:0`. The fused YOLOv8x-seg model reported 125 layers, 71.7 million parameters, and 327.9 GFLOPs, with two output classes: `road_crossing` and `blind_path`. The first warmup pass took 766.5 ms, while the following four passes averaged approximately 12.5 ms. For this reason, the implemented system runs blind-path and crosswalk segmentation on the GPU-enabled backend when available. Interval scheduling is then mainly used to prevent heavier auxiliary perception tasks, especially obstacle detection, from blocking the navigation loop.

Obstacle detection was verified functionally but not yet as a full real-time pipeline. The YOLO-E model loaded successfully on `cuda:0`; the fused YOLOe-11l-seg model reported 227 layers, 35.1 million parameters, and 134.1 GFLOPs. The system precomputed text embeddings with shape `[1, 29, 512]` for the 29-class whitelist, whose first ten classes were bicycle, car, motorcycle, bus, truck, animal, scooter, stroller, dog, and pole. The startup self-test completed successfully, although the test frame produced zero detections. Because this pathway is heavier than the primary segmentation loop, the final system does not run obstacle detection on every frame. It uses a 15-frame default detection interval, cached results, and path-mask overlap filtering so that expensive obstacle reasoning does not block every navigation update. Table 5 summarizes the computer-vision verification status for segmentation, obstacle detection, and traffic-light recognition.

Table 5: Computer vision verification results.

Module	Procedure	Result	Interpretation
Blind-path and crosswalk segmentation	Load segmentation model and run warmup/inference on representative frames.	Backend loads the segmentation model on `cuda:0` when available; final warmup after the first pass averaged about 12.5 ms; CPU-only profiling averaged 885.3 ms.	Primary blind-path and crosswalk segmentation should use GPU acceleration for real-time navigation.
Obstacle detection	Load YOLO-E model and detect whitelist objects on path-overlap masks.	Model loaded on `cuda:0`; 29-class whitelist available; text-feature tensor shape was `[1, 29, 512]`; startup detection self-test succeeded.	Functional, but must be interval-scheduled and cached.
Traffic-light detection	Load traffic-light model and process frames with temporal voting.	Classes include stop/go/countdown states. Standalone mode uses 3-of-5 voting; cross-street mode accepts a stable state from repeated detections and requires five stable green frames before crossing.	Functional, but outdoor precision still needs a labeled field-test set.

3.3 Navigation Logic Verification

Navigation behavior was verified through state-machine inspection, live-frame testing, and controlled scenario reasoning. The top-level `NavigationMaster` prevents conflicting guidance modes from running simultaneously. Blind-path navigation, crosswalk seeking, traffic-light waiting, active crossing, recovery, and standalone traffic-light detection are separated into explicit states. This structure makes safety-critical transitions testable because each transition requires a clearly defined perception condition.

For blind-path navigation, the verification procedure checks whether the system identifies the tactile path mask, fits the centerline, and converts the result into one voice command. A valid frame should produce a straight-ahead guidance prompt or an obstacle warning. The obstacle warning has higher priority than ordinary path guidance. This behavior is implemented in the voice-priority logic and prevents a lower-priority straight-ahead command from masking a collision warning.

For cross-street navigation, the final workflow uses three major states: crosswalk seeking, traffic-light waiting, and active crossing. The transition into traffic-light waiting requires crosswalk nearness to satisfy all three geometry checks:

$$\rho_{\text{area}} \geq 0.20, \qquad \bar{y}_{\text{bot}} \geq 0.80, \qquad \rho_{\text{height}} \geq 0.35. \tag{25}$$

The crossing transition is allowed only after the traffic-light detector reports a stable green state for five frames:

$$\text{crossing allowed} = (\text{stable light} = \texttt{go}) \wedge (N_{\text{green}} \geq 5). \tag{26}$$

If the red state is detected, the state machine remains in the traffic-light waiting state and instructs the user to stop and wait. A no-light fallback exists for development testing so that the crossing workflow can be exercised when the light is outside the camera field of view. This fallback is not treated as final safety validation and should be disabled or require user confirmation in a deployment-oriented version.

The integrated log provides one representative successful green-light sequence and one red-light blocking sequence. In the green-light sequence, the system entered traffic-light waiting at frame 8 when the crosswalk geometry reached $\rho_{\text{area}} = 0.214$, $\bar{y}_{\text{bot}} = 0.999$, and $\rho_{\text{height}} = 0.354$. A green light was detected at frame 9, became stable at frame 10, accumulated four stable green counts by frame 13, and triggered active crossing at frame 14. The same run produced the spoken message that crossing was finished at frame 93, when the crosswalk area ratio had dropped to 0.072. In a separate red-light sequence, the system entered traffic-light waiting at frame 24 and remained blocked by `stop` detections from frame 25 through at least frame 80,

including the spoken instruction "Red light. Stop and wait." at frame 45. Table 6 lists the representative log events used to verify these transitions.

Table 6: Representative cross-street state-machine evidence extracted from the integrated test log.

Log frame	State or event	Measured evidence	Verification meaning
8	Enter traffic-light waiting	Crosswalk near check passed with area ratio 0.214, bottom ratio 0.999, and height ratio 0.354.	Geometry threshold works on live frames
9–14	Stable green accumulation	`go` detections accumulated until the workflow announced "Green light is stable. Start crossing."	Green transition confirmed
93	Crossing completion prompt	Crosswalk area ratio decreased to 0.072 and the system announced sidewalk preparation.	Exit cue generated
25–80	Red-light waiting	Stable `stop` state persisted; frame 45 generated "Red light. Stop and wait."	Red state blocks crossing

Table 7 converts the same navigation logic into scenario-level verification outcomes for blind-path following, obstacle priority, crosswalk approach, light-gated crossing, and development-only no-light handling.

Table 7: Navigation scenario verification matrix.

Scenario	Expected behavior	Observed or implemented behavior	Status
Blind path visible and centered	Generate straight-ahead guidance while maintaining mask and centerline overlays.	Blind-path workflow computes mask geometry and centerline position; low-priority straight prompts are rate-limited.	Pass
Obstacle overlaps walking path	Obstacle warning overrides ordinary path guidance.	Detected obstacles are filtered by overlap with the path mask and assigned higher voice priority.	Pass, latency-limited
Far crosswalk visible	Remain in crosswalk-seeking state and provide alignment guidance.	Crosswalk workflow estimates angle and offset using PCA or Hough-line geometry before declaring the crosswalk near.	Pass
Near crosswalk with stable green light	Enter traffic-light waiting, count five stable green frames, then enter crossing.	Five-frame green threshold implemented; in the integrated log, frame 14 entered crossing after repeated `go` detections.	Pass in logic; field validation needed
Near crosswalk with red light	Remain stopped and do not enter crossing.	Red/stop state blocks crossing and produces a stop-and-wait instruction; the integrated log remained in waiting from frame 25 through at least frame 80 under `stop`.	Pass
Near crosswalk with no detected light	Avoid indefinite lockup during development tests while documenting safety risk.	Fallback transition after repeated no-light frames is implemented for demo/debug use only.	Dev pass

3.4 Audio Feedback and Voice-Interaction Verification

The audio-feedback subsystem was verified by checking prompt preloading, prompt prioritization, and duplicate-speech prevention. Startup logs show that 77 voice mappings were merged and 72 audio files were loaded, so 93.5% of predefined local prompts were available and the 90% prompt-availability requirement was met. Compression reduced the loaded prompt files from 3768.0 KB to 942.6 KB, or 25.0% of the original size, saving 2825.4 KB. This supports the design decision to preload common prompts so safety-critical messages can play without waiting for a network-generated TTS response. The same log also reported five missing prompt files, including `crossing_mode_started.wav` and `crosswalk_ahead.wav`; these missing files should still be regenerated before final demonstration so that all state transitions have local audio.

The final implementation centralizes speech decisions at the navigation-workflow level. Traffic-light detection does not speak independently during cross-street navigation; instead, it returns state information to the crossing workflow. This prevents overlapping red-light, green-light, obstacle, and direction prompts. Warning prompts are also assigned higher priority than routine guidance, which is necessary because a delayed obstacle warning is more dangerous than a delayed straight-ahead prompt.

Voice command routing was partially verified through command-report logs. The system successfully routed commands such as `Start navigation.`, `Stop navigation`, and `Detect traffic light.` to the corresponding navigation states. However, the general chat pathway is not yet verified: 13 requests failed with `Completions.create()` receiving an unsupported `modalities` argument. This does not invalidate local navigation prompts, but it means open-ended voice interaction should be marked unresolved until the OpenAI client call is updated and retested.

4 Cost

All costs in this section are listed in RMB. Labor cost is estimated using the required formula

$$\text{ideal salary} \times \text{actual hours spent} \times 2.5.$$

Table 8 summarizes the partner labor estimate. Electronics assembly and debugging are included in Shengnan Cai's hours. No machine-shop hours were used because the prototype used an off-the-shelf glasses frame and hand-built mounts.

Table 8: Labor cost estimate.

Partner	Main work	Rate (yuan/hr)	Hours	Cost (yuan)
Shengnan Cai	Wearable hardware, electronics integration, and field support	120	115	34,500
Junchen He	Backend streaming, deployment, and system logging	120	125	37,500
Yi Su	Perception models, navigation logic, and crossing workflow	120	130	39,000
Mingyan Gao	Verification planning, test evidence, and report integration	120	105	31,500
Total			**475**	**142,500**

Table 9 lists the prototype parts cost, including both retail cost and the amount paid by the team or department. Lab-owned or existing resources would be listed as free; in this prototype, the paid cost mainly came from purchased wearable parts and backend rental time.

Table 9: Prototype parts cost and bulk-purchase estimate.

Item	Retail unit	Qty.	Paid	Bulk unit	Note
Seeed Studio XIAO ESP32-S3 / ESP32-S3 Sense controller	110	1	110	80	Main wearable controller
OV5640-class 5 MP camera module	126	1	126	95	Upgraded visual input
MAX98357A I2S amplifier and speaker	25	1	25	15	Audio feedback output
PDM microphone module	18	1	18	10	Voice-command input
Battery, regulator, wiring, headers, solder, and small PCB materials	200	1	200	130	Power and assembly materials
Glasses frame and mechanical mounting materials	2	1	2	2	Off-the-shelf frame
Backend compute rental	400	1	400	N/A	GPU time for AI models
Miscellaneous replacement parts and testing materials	200	1	200	120	Spares and test fixtures
Total prototype parts cost			**1081**	**452**	Bulk estimate excludes backend rental

The total estimated project cost is therefore 143,581 yuan, including 142,500 yuan of labor and 1,081 yuan of paid prototype parts. For a commercial version, the wearable bill of materials could fall to about 452 yuan per unit under bulk purchasing, excluding backend or cloud-compute service.

5 Conclusion

5.1 Accomplishments

The project delivered an integrated prototype that combines ESP32-S3 sensing, FastAPI-based AI processing, YOLO segmentation, traffic-light detection, obstacle filtering, voice commands, and spoken navigation prompts. The system demonstrates the intended end-to-end workflow: it can stream camera and audio data, identify blind paths and crosswalk regions, manage crossing states, block crossing on red-light evidence, prioritize obstacle warnings, and expose live debugging information through the browser monitor.

Verification confirmed that the primary segmentation loop runs in real time on the GPU backend after warmup, while CPU-only execution remains too slow for full-rate guidance. The final architecture therefore uses GPU acceleration when available and interval scheduling for heavier auxiliary perception tasks. The navigation state machine also reduced prompt conflicts by centralizing traffic-light, crossing, obstacle, and routine guidance decisions.

5.2 Uncertainties and Unresolved Issues

Important limitations remain. First, CPU-only inference averaged 885.3 ms per segmentation frame, so the system depends on a GPU-capable backend for full responsiveness. Second, traffic-light recognition still needs a larger labeled outdoor test set, especially at night where brake lights, reflections, and non-standard fixtures may cause false positives. Third, worn crosswalk markings can fall below the mask-area threshold and keep the workflow in SEEKING_CROSSWALK. Finally, Wi-Fi range and packet loss still limit sustained outdoor use; reconnect logic exists, but no robust degraded-mode behavior is implemented.

5.3 Future Work and Design Alternatives

Future revisions should move inference onto a stronger edge platform or a wearable companion device with an NPU, add depth sensing for metric obstacle distance, retrain crosswalk segmentation on faded and occluded markings, and add haptic feedback so routine left/right corrections do not compete with spoken safety alerts.

5.4 Broader Impacts

The broader societal value of the project is improved access to independent mobility for visually impaired users, especially in outdoor settings where tactile paving, crosswalks, traffic signals, and moving obstacles must be interpreted together. Economically, the prototype suggests that useful navigation assistance can be built from low-cost wearable sensing hardware when expensive AI computation is separated from the glasses frame. That design choice also limits deployment readiness, because a backend computer or future edge accelerator remains necessary for reliable real-time inference. Environmentally, the small wearable unit has modest material cost, but future versions should use replaceable batteries, repairable mounts, and model compression to reduce unnecessary hardware replacement. The project therefore has positive accessibility potential, but only if safety claims remain conservative and future validation includes users from the community the system is meant to assist.

5.5 Ethical Considerations

The design and deployment of the AI Navigation Glasses were evaluated against the IEEE Code of Ethics [4]. Because the project provides mobility assistance to a potentially vulnerable user group, the main ethical requirement is to make truthful safety claims, reduce foreseeable risk, and avoid presenting a prototype as a certified mobility aid. Table 10 maps the main ethical concerns to the mitigations implemented or documented in this prototype.

Table 10: Ethical risk mapping and mitigation plan.

IEEE concern	Project relevance	Mitigation
Public safety, health, and welfare	Incorrect crossing or obstacle prompts could place a user in danger.	Crossing requires repeated green evidence; red/yellow blocks crossing; unresolved field limits are disclosed.
Honesty and realistic claims	The prototype has limited field validation and no clinical/user study approval.	The report separates verified results from unresolved limitations and does not claim deployment readiness.
Competence and technical limitations	AI models may fail under night lighting, faded markings, occlusion, or weak WiFi.	The design uses confidence thresholds, temporal voting, reconnect logic, and documented future work.
Privacy and data handling	Camera and microphone streams can contain bystanders and personal speech.	Processing is local to the backend during testing; logs emphasize timing and model metadata instead of personal content.
Fair treatment and accessibility	Assistive systems should not exclude users through unsafe assumptions or inaccessible feedback.	The interface uses concise voice prompts, stop commands, and future haptic feedback is proposed for lower audio load.

Public safety and risk mitigation. The system is an assistive aid, not a replacement for a cane, guide, or user judgment. It never initiates crossing on a single traffic-light detection; crossing requires crosswalk alignment, repeated green evidence, and user control. Obstacle warnings are prioritized over routine guidance, while a stop command returns the system to CHAT.

Competence, privacy, and honesty. Testing was limited to team members and simulated low-vision trials because no IRB approval was obtained for trials with visually impaired participants. Camera and microphone data are processed locally on the backend, the browser monitor defaults to localhost access, and logs store timestamps and inference metadata rather than personal content. Reported results are based on available logs and are separated from unresolved limitations instead of being presented as deployment-ready safety validation.

References

[1] Espressif Systems, "ESP32-S3 Series Datasheet," https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf, 2024, accessed: 2026-05-15.

[2] FastAPI, "FastAPI Documentation," https://fastapi.tiangolo.com/, 2026, accessed: 2026-05-15.

[3] Ultralytics, "Ultralytics YOLO Documentation," https://docs.ultralytics.com/, 2026, accessed: 2026-05-15.

[4] IEEE, "IEEE Code of Ethics," https://www.ieee.org/about/corporate/governance/p7-8.html, 2016, accessed: 2026-05-15.

A Requirement Traceability Appendix

This appendix records how the high-level requirements in Section 1.4 map to the verification evidence in Section 3. It is included to make the final report easier to audit against the format checklist and to separate completed engineering verification from deployment limitations. Table 11 gives the section-level traceability summary, and Table 12 gives the full requirement and verification matrix.

Table 11: Traceability between high-level requirements and verification evidence.

Requirement	**Verification evidence**	**Current status**
Perception accuracy	Model startup, segmentation warmup, obstacle whitelist loading, traffic-light class history, and representative frame-level checks are summarized in Table 5.	Partially verified; larger labeled outdoor set remains needed.
Real-time responsiveness	GPU warmup results and CPU-only profiling are compared in Table 2 and Table 5.	Pass on GPU backend after warmup; CPU-only mode is unresolved.
Navigation stability and reliability	State-machine transitions, crosswalk geometry gates, green-light accumulation, and red-light blocking behavior are shown in Table 6.	Pass for controlled integration tests; extended field reliability remains needed.
Audio and voice interaction	Prompt preload counts, command routing, and missing prompt files are described in Section 3.	Pass for local prompts and command routing; open-ended chat path is unresolved.
Fail-safe crossing behavior	The workflow requires repeated green evidence and blocks immediately on red or yellow states, as documented in Table 7.	Pass in implemented logic; development-only no-light fallback is not deployment-ready.

Table 12: Complete requirement and verification matrix.

Requirement	Verification procedure	Acceptance criterion	Measured or observed result	Status
Perception accuracy	Run segmentation, obstacle filtering, and traffic-light modules on representative outdoor frames; compare outputs with manual frame labels.	At least 90% frame-level agreement for path, crosswalk, and obstacle relevance; at least 85% precision for visible traffic-light state.	Models load and produce annotated outputs; limited frame checks are available, but a sufficiently large labeled outdoor set is not complete.	Partial
Real-time responsiveness	Measure camera stream rate, model warmup, segmentation inference time, and prompt-event delay from backend logs.	Camera stream remains near 5 fps; segmentation is below 100 ms after warmup on the GPU backend; safety prompts are issued within 1000 ms.	GPU segmentation averaged about 12.5 ms after first warmup passes; CPU-only profiling averaged 885.3 ms and cannot support full-rate guidance.	Pass on GPU
Navigation stability and reliability	Inspect state-machine transitions under blind-path and cross-street scenarios, including crosswalk entry, red-light wait, green-light crossing, and completion.	Centerline estimates remain stable under walking motion; crossing requires repeated green evidence; red or yellow blocks crossing.	Controlled logs show crosswalk entry at frame 8, crossing after repeated green evidence at frame 14, completion at frame 93, and red-light blocking from frame 25 through at least frame 80.	Pass
Audio and voice interaction	Check prompt preloading, local prompt availability, command routing, audio uplink, and WAV downlink.	At least 90% of predefined local prompts are available; start, stop, traffic-light, and crossing commands route correctly.	72 of 77 mapped prompt files were loaded, or 93.5%; command routing works for local navigation, but five prompt files and open-ended chat remain unresolved.	Pass for local prompts
Fail-safe crossing behavior	Test crossing decisions under red, yellow, green, and no-light conditions in the navigation workflow.	No crossing instruction from a single-frame light result; red/yellow block crossing immediately; green requires at least five stable frames.	The workflow implements repeated green counting and red/yellow blocking. A no-light fallback remains for development and is not acceptable for deployment without user confirmation or removal.	Partial

B Project Schedule Appendix

Table 13 records the semester work distribution and major milestones. It is placed in the appendix because the ECE 445 final report outline prioritizes design, verification, cost, and conclusions in the main text.

Table 13: Complete semester project schedule.

Wk.	Shengnan Cai	Junchen He	Yi Su	Mingyan Gao	Milestone
1	Hardware feasibility	Backend plan	Vision task scope	Requirements notes	Scope fixed
2	Parts selection	Backend plan	Model comparison	Test plan	Architecture selected
3	Camera bring-up	Camera WebSocket	YOLO baseline	Test plan	First stream test
4	Audio wiring	Audio paths	Mask tuning	Test logging	Sensing prototype
5	Camera upgrade	Monitor buffering	Mask tuning	Report diagrams	Better camera input
6	Mounting and power	Queue handling	Obstacle prototype	Midpoint evidence	Walking-test setup
7	Mounting and power	Reconnect logic	Guidance smoothing	Demo coordination	Midterm prototype
8	Prompt playback	ASR/TTS routing	Traffic-light voting	Verification update	Voice modes connected
9	Wearable stability	Mode manager	Crossing states	Scenario design	Crossing workflow
10	Wearable stability	Cached scheduling	Crossing states	Safety review	Real-time logic stable
11	Camera repair	Startup logs	Optical-flow smoothing	Evidence collection	Stability improved
12	Voice-file check	Startup logs	Obstacle filtering	Verification draft	Subsystems verified
13	Demo hardware	GPU warmup logs	Crossing tests	Cost update	Integrated log collected
14	Field-test support	PDF build checks	Frame evidence	Final tables	Report assembled
15	Final hardware check	Final build	Requirement review	Submission checklist	Final package completed