TED UNIVERSITY

Faculty of Engineering

CMPE 491– Senior Project 2

VAVI

LOW DESIGN REPORT

31/10/2025

Ceyda Kuşçuoğlu – 16348076072

Kıvanç Terzioğlu – 27233564574

Berkay Kaan Karaca – 68317070956

1. Introduction

This Low-Level Design (LLD) document describes the detailed design and implementation decisions for the **Real-Time Object Detection and Indoor Navigation System for Visually Impaired Users**. The system integrates on-device computer vision, directional audio feedback, and indoor localization (fusion of Wi-Fi fingerprinting and IMU-based dead-reckoning) to provide continuous navigation assistance in indoor environments. This document translates high-level architectural components into concrete classes, interfaces, data structures, algorithms, and messages to be implemented by the development team.

This LLD aims to:

- Provide precise class and interface specifications so developers can implement modules without additional architectural decisions.
- Define data formats, API endpoints, and message contracts for module interactions and backend integration.
- Specify algorithms used for localization (KNN/RF, Kalman/Complementary filter), path planning (A*, Dijkstra), and audio feedback mapping (direction and proximity → stereo balance + beep frequency).
- Describe testing criteria, performance targets, and error handling strategies.

Scope and limitations: This document assumes an Android native client (Kotlin) using CameraX and TensorFlow Lite for inference, an optional backend (FastAPI) for data aggregation and heavier computations, and a lightweight SQLite / PostgreSQL database for storing fingerprints and map graphs. Where trade-offs exist (e.g., model size vs. accuracy), chosen solutions and rationale are provided.

2. Object Design Trade-offs

This section explains design trade-offs made during object design, including alternatives considered and the reasons for final choices.

2.1 Vision Inference: On-Device vs. Server-Side

- **Option A — On-device (chosen):**
    - *Pros:* Works without network, lower latency for single-frame feedback, better privacy (video never leaves device).
    - *Cons:* Limits model size/complexity; relies on mobile CPU/NPU.
    - *Rationale:* User safety and offline reliability are priorities; therefore a lightweight YOLOv5n TFLite model is selected and quantized (float16/int8 when supported).

- **Option B — Server-Side (optional mode):**
    - *Pros:* Larger models, centralized logging, easier model updates.
    - *Cons:* Network dependency, latency, privacy issues.

2.2 Localization: Fingerprinting vs. Range-based Methods

- **Wi-Fi Fingerprinting (chosen primary):**
    - *Pros:* No extra infrastructure required if APs exist; robust indoors where GPS fails.
    - *Cons:* Requires mapping/wardriving to collect fingerprints; RSSI variability across time.
- **Range-based (TOA/TDOA) or BLE beacons (alternate):**
    - *Pros:* Potentially more accurate if infrastructure is provided.
    - *Cons:* More infrastructure cost and deployment effort.

2.3 Sensor Fusion Complexity

- **Complementary Filter vs. Kalman Filter:**
    - Complementary filter is simpler, lower computational cost; used for orientation smoothing.
    - Kalman filter (or extended Kalman) used for fusing Wi-Fi position estimates (discrete, lower rate) with high-rate IMU dead-reckoning for smoother trajectory estimation.

2.4 Path Planning

- **Dijkstra vs. A* (chosen):**
    - A* with Manhattan/Euclidean heuristic on node coordinates is used for faster route computation where heuristics are admissible. Dijkstra is available as fallback for graphs without meaningful coordinates.

3. Interface Documentation Guidelines

This section prescribes how interfaces are documented and what contracts modules must respect.

3.1 Class Interface Style

- Each method description must include: purpose, input types (with units), output, exceptions thrown, and complexity notes (if nontrivial).

3.2 Inter-Module Communication

- **In-app module communication:** Use Observer / Listener patterns or LiveData streams (Android) to decouple producers (CameraManager, WiFiScanner) from consumers (ObjectDetector, IndoorNavigator).
- **App ↔ Backend communication:** JSON messages sent over WebSocket for real-time streams. REST endpoints for initialization, fingerprint upload, and map/graph updates.

3.3 Versioning & Backwards Compatibility

- All JSON messages must include protocol_version and timestamp fields.
- Server must accept older versions for at least one minor upgrade window.

3.4 Error Handling Contract

- Methods return Result<T>-like objects or throw documented exceptions.
- Network failures trigger retry with exponential backoff; critical features (inference, audio) degrade gracefully (fallback to offline-only mode).

4. Engineering Standards

- **Design Documentation:** IEEE 1016 for SDD structure and content.
- **Requirements Reference:** IEEE 830 compliance for traceability between requirements and design elements.
- **Modeling Notation:** UML 2.0 for class, sequence, activity, and package diagrams.
- **Coding Standards:** Kotlin coding guidelines (naming, immutability where possible, coroutines for concurrency). For backend: PEP8 / FastAPI best practices.
- **Security & Privacy:** Follow ACM Code of Ethics; never transmit raw image frames unless explicitly consented; anonymize logs.
- **Accessibility:** Audio feedback design must follow accessibility principles — clear cues, adjustable volume/settings.

---

5. Packages

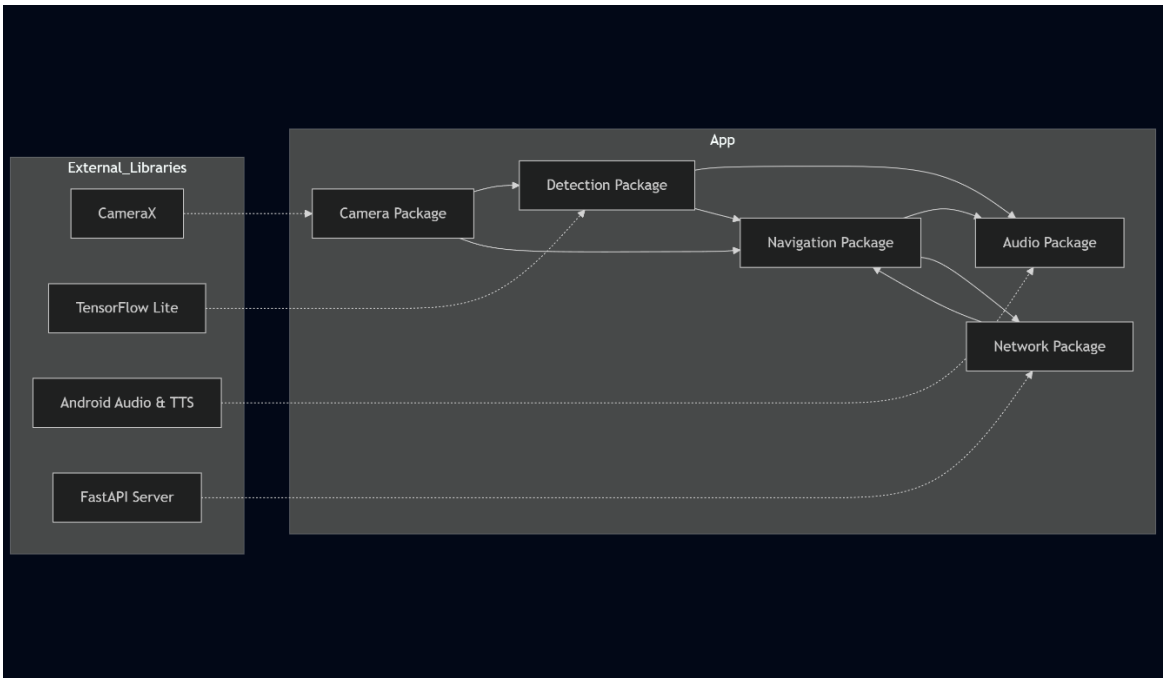


Figure1 – UML Package Diagram

Package: camera

- Responsibilities: configure CameraX, manage lifecycle, control frame rate/resolution, perform YUV→RGB conversion, maintain buffer pool, and hand off frames via a nonblocking queue to the detection pipeline.
- Public interface highlights: startCamera(), stopCamera(), setFrameListener(listener), setResolution(width,height)

Package: detection

- Responsibilities: load TFLite Interpreter, manage interpreter thread(s), pre/post-process images (resize, normalization, NMS), provide DetectionResult stream with bounding boxes, class labels, confidences and estimated depth.
- Performance goals: <150 ms inference per frame on supported device (or <66 ms for 15 fps target).

Package: audio

- Responsibilities: map detection center X to stereo balance, convert detection distance to beep frequency, manage AudioTrack / OpenSL ES to generate low-latency beeps, support TTS for navigation instructions.
- Public: notifyDetections(list<DetectionResult>), notifyNavigationStep(step), setVolume(level)

Package: network

- Responsibilities: manage WebSocket sessions, define JSON message schemas, encrypt messages (TLS), queue telemetry when offline, upload fingerprint datasets.

- API Examples (to include in report): /api/v1/fingerprint/upload, /api/v1/map/get, WebSocket path /ws/live/{device_id}

Package: navigation

- Responsibilities: maintain indoor graph, run localization module, compute path, produce human-friendly step instructions, interface with audio module for navigation cues.
- Public: setTarget(nodeId), cancelNavigation(), getCurrentPosition()

6. Detailed Class Interfaces

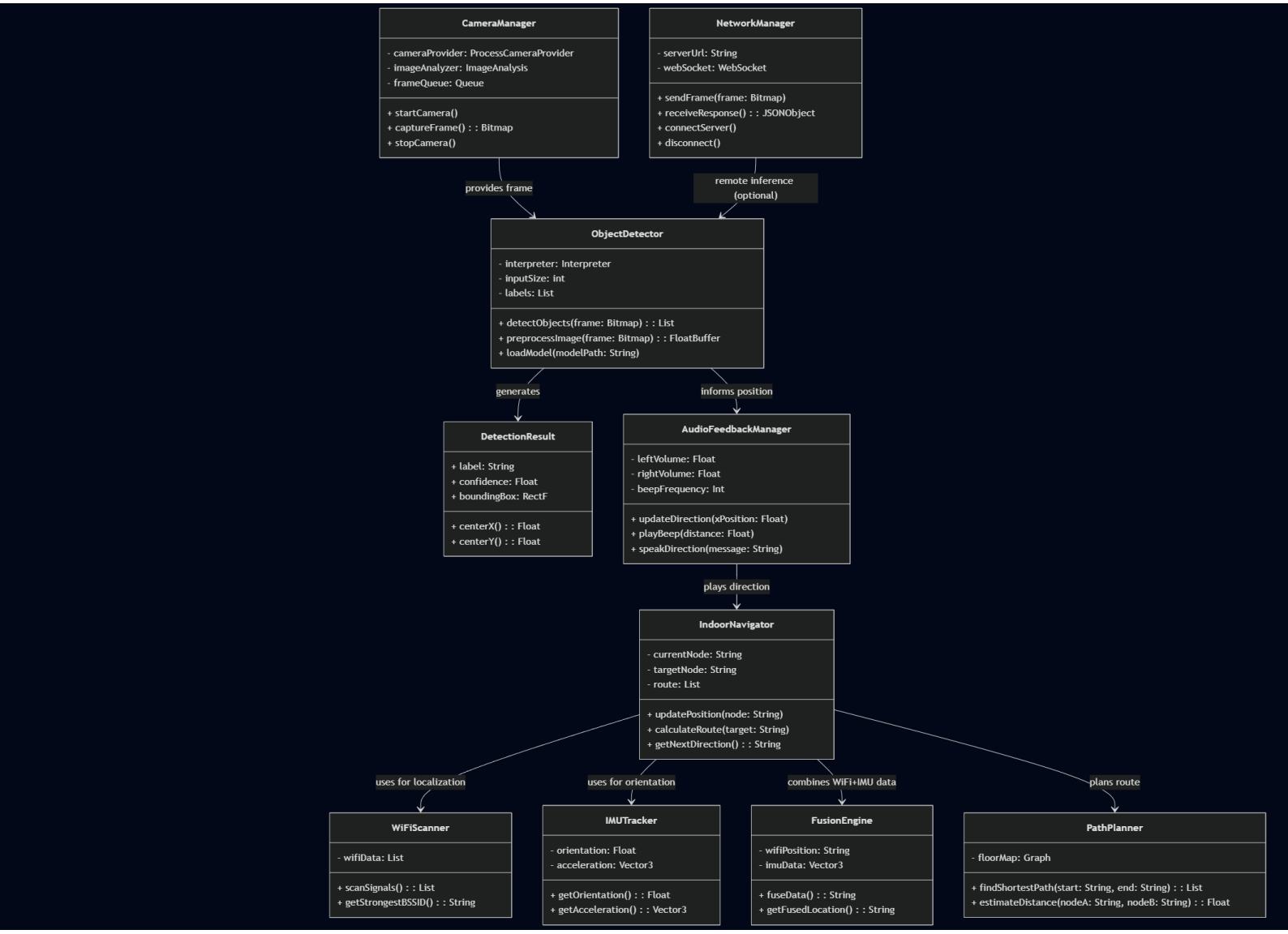
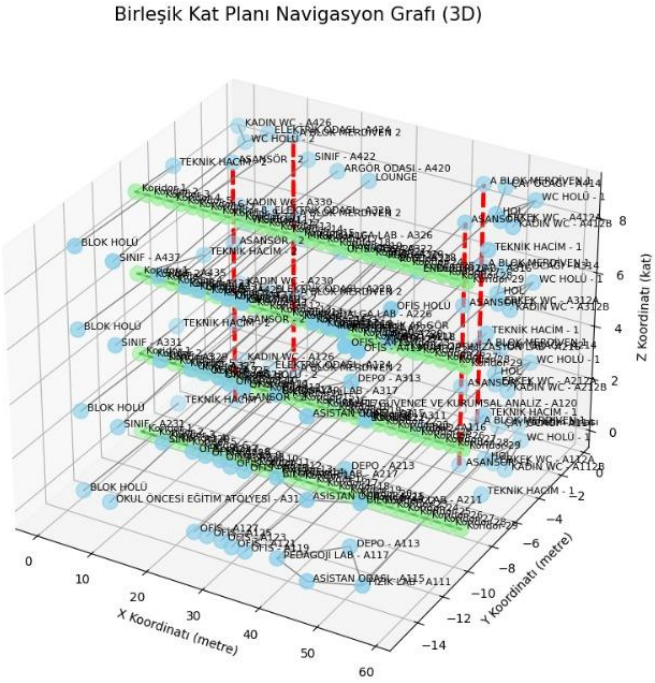
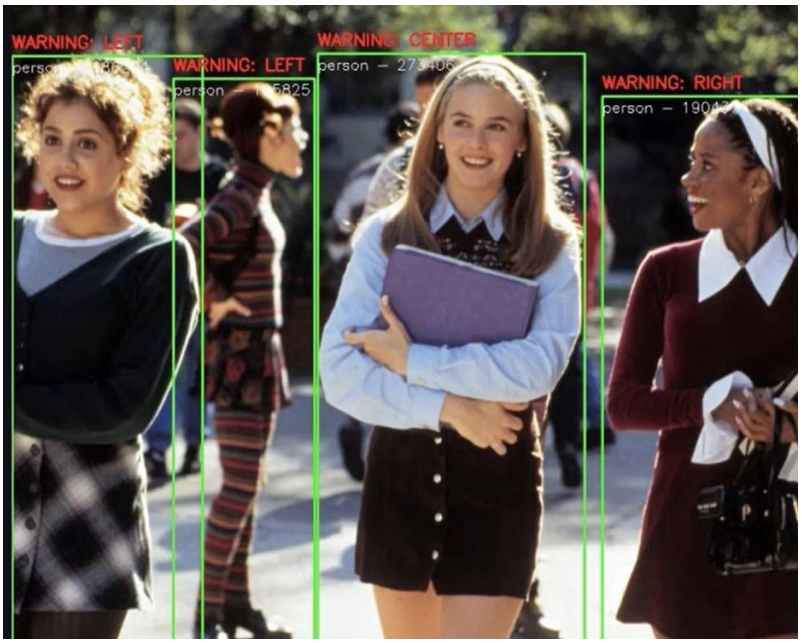


Figure2 - UML Class Diagram





CameraManager (camera package)

Class CameraManager

Attributes:

- cameraProvider: ProcessCameraProvider
- analysisUseCase: ImageAnalysis
- previewUseCase: Preview
- frameQueue: ConcurrentLinkedQueue<Bitmap>
- frameRate: Int
- resolution: Pair<Int,Int>

Methods:

+ startCamera(context: Context): Unit

- Initializes CameraX with specified resolution and frameRate.

+ stopCamera(): Unit

+ setFrameListener(listener: (Bitmap, Long) -> Unit): Unit

- Provides frames as (bitmap, timestamp).

+ convertYuvToRgb(image: ImageProxy): Bitmap

+ setFrameRate(fps: Int): Unit

ObjectDetector (detection package)

Class ObjectDetector

Attributes:

- interpreter: Interpreter (thread-safe wrapper)

- inputSize: Int (e.g., 640)

- labels: List<String>

- detectionThreshold: Float

- iouThreshold: Float

Methods:

+ loadModel(filePath: String, useNnApi: Boolean = false): Unit

+ runInference(bitmap: Bitmap): List<DetectionResult>

- Returns list of DetectionResult sorted by confidence.

+ postProcess(rawOutputs: Array<FloatArray>): List<DetectionResult>

+ estimateDistance(boxHeightPx: Float): Float

- Simple inverse-proportion mapping or use stereo calibration.

DetectionResult (value object)

Class DetectionResult

Attributes:

- label: String

- confidence: Float (0.0 - 1.0)
- bbox: RectF (left, top, right, bottom) in image coordinates
- timestamp: Long (ms since epoch)
- centerX: Float (relative 0..1)
- centerY: Float (relative 0..1)
- estimatedDistanceMeters: Float (optional)

AudioFeedbackManager (audio package)

Class AudioFeedbackManager

Attributes:

- leftGain: Float
- rightGain: Float
- baseBeepIntervalMs: Int
- lastBeepTs: Long

Methods:

+ notifyDetections(results: List<DetectionResult>): Unit
+ updateDirection(centerX: Float): Unit
+ updateFrequency(distanceMeters: Float): Unit
+ playBeep(): Unit
+ speak(text: String): Unit
  - Uses TTS for navigation instructions

IndoorNavigator (navigation package)

Class IndoorNavigator

Attributes:

- graph: Graph<Node,Edge>
- currentEstimate: PositionEstimate (nodeId, x, y, covMatrix)

- fusionEngine: FusionEngine

- pathPlanner: PathPlanner

Methods:

+ loadMap(mapJson: JSONObject): Unit

+ setTarget(targetNodeId: String): Boolean

+ updateSensorData(wifiSamples: List<WiFiSample>, imu: IMUReading): PositionEstimate

+ computePath(): List<Node>

+ nextNavigationInstruction(): NavigationInstruction

7. Algorithms

7.1 Wi-Fi Fingerprinting (KNN)

- **Training data:** For each node, collect multiple Wi-Fi scans; represent each scan as a vector over canonical AP list (BSSID order). Missing APs → fill with -100 dBm.
- **Model:** KNN with Euclidean distance on normalized RSSI vectors. Optionally use Random Forest classifier on discriminative features (top-N APs).

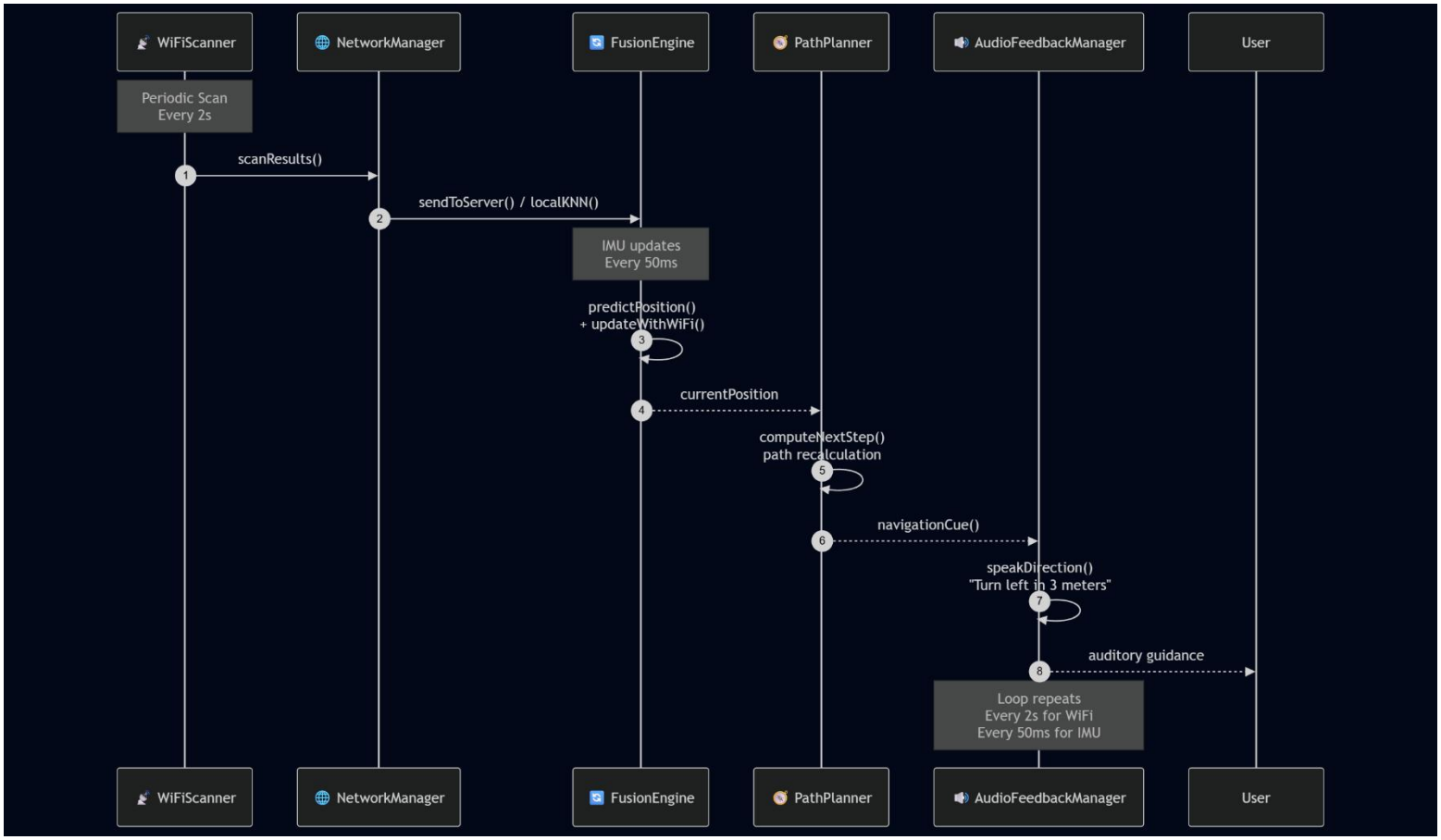


Figure 3 - Sequence Diagram: Camera → Detection → Audio

7.2 IMU Dead-Reckoning (step detection + heading)

- **Step detection:** Use vertical acceleration peaks and thresholding.
- **Heading:** Use fused magnetometer/gyroscope/accelerometer with complementary filter.
- **Position update:** x += step_length * cos(heading), y += step_length * sin(heading)

7.3 Fusion Engine (Kalman Filter high-level)

- **State vector:** [x, y, vx, vy] or discrete node probability vector (if using particle / Bayes).
- **Measurement:** Wi-Fi position estimate (x_meas, y_meas) with covariance R; IMU provides control inputs (delta movement) with covariance Q.

7.4 Path Planning (A*)

- Graph nodes have (x,y). Heuristic = Euclidean distance.
- Pseudocode is standard A* (include in report).

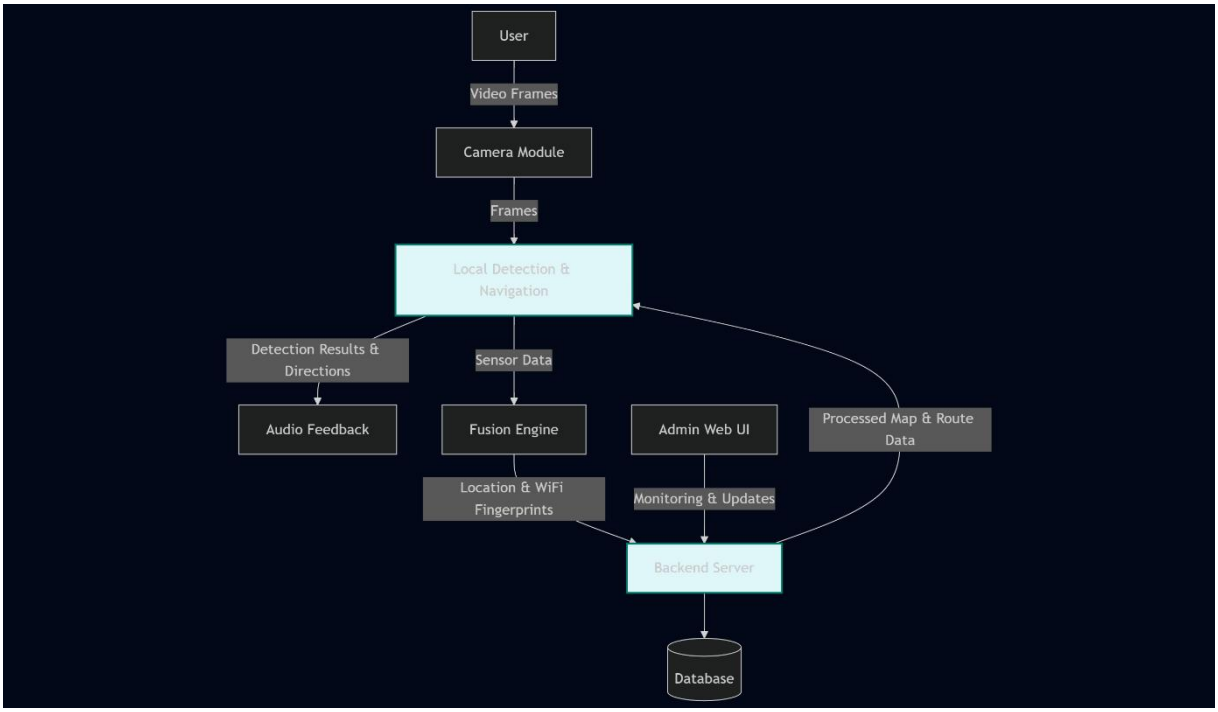


Figure 4 - Sequence Diagram: Navigation Update Cycle

8. Data Formats & JSON Schemas

8.1 Wi-Fi Sample JSON

```
{
  "device_id": "device123",
  "timestamp": 1690000000000,
  "samples": [
    {"bssid": "c8:bf:4c:fc:c6:45","ssid":"FiberHGW_1","rssi": -47},
    {"bssid": "f4:5c:89:aa:bb:cc","ssid":"eduroam","rssi": -78}
  ],
  "location_label": "node_12"  // optional for training uploads
}
```

8.2 Detection Telemetry JSON (WebSocket)

```
{
  "protocol_version": "1.0",
  "device_id": "device123",
  "timestamp": 1690000001000,
  "detections": [

{"label":"person","confidence":0.93,"bbox":[0.12,0.24,0.48,0.92],"center":[0.30,0.58],"distance_m":2.4}
  ]
}
```

8.3 Navigation Instruction (from server → device)

```
{
  "protocol_version":"1.0",
  "device_id":"device123",
  "timestamp":1690000002000,
  "instruction": {
    "type":"turn_right",
    "distance_m":3.2,
```

```
  "next_node":"node_15",
  "eta_seconds": 12
}
```

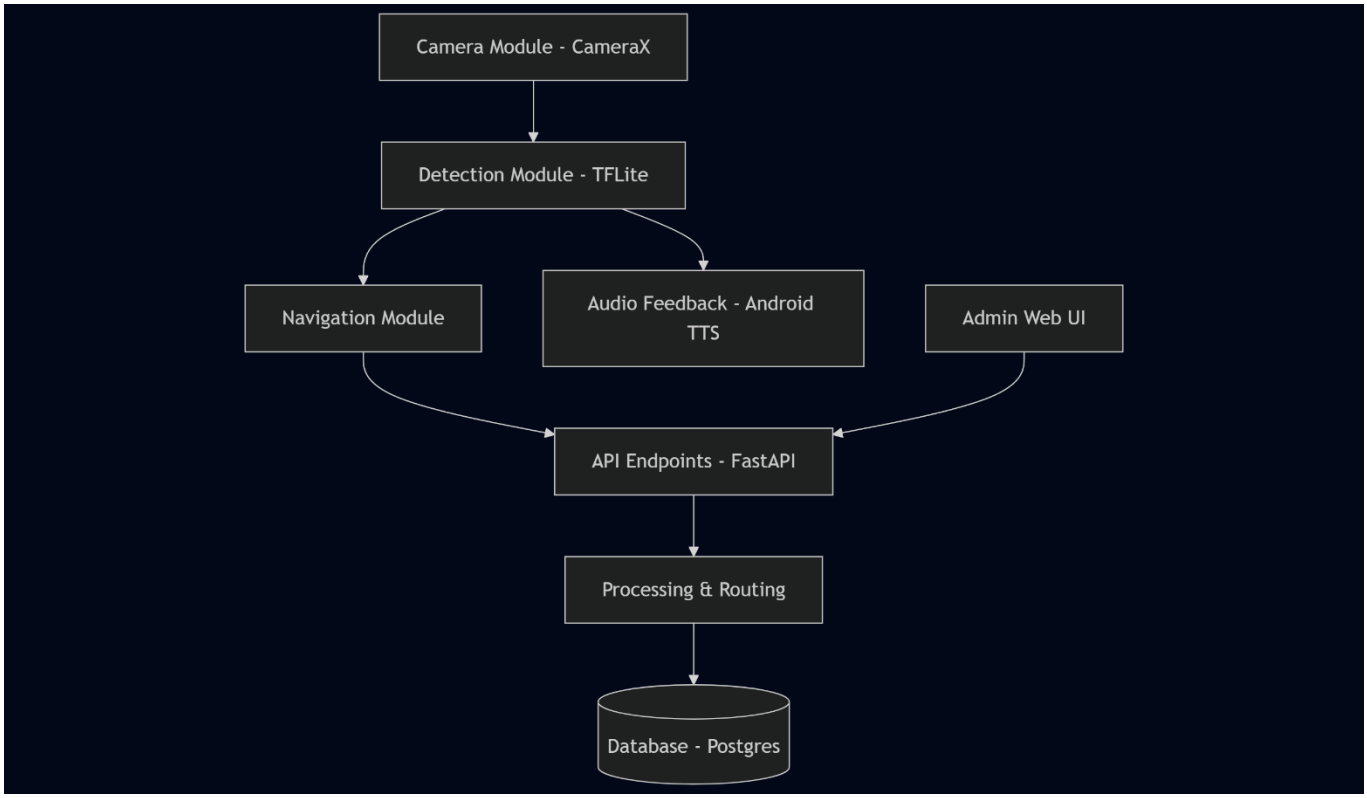



Figure 5 - Data Flow Diagram

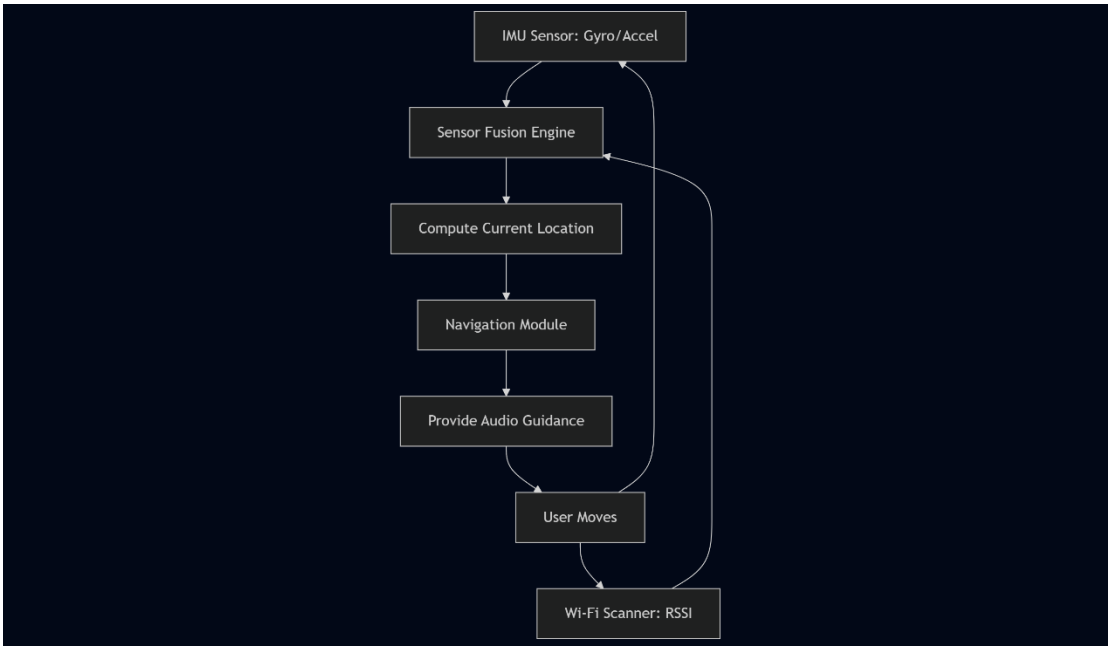


Figure 6 - System Architecture Diagram

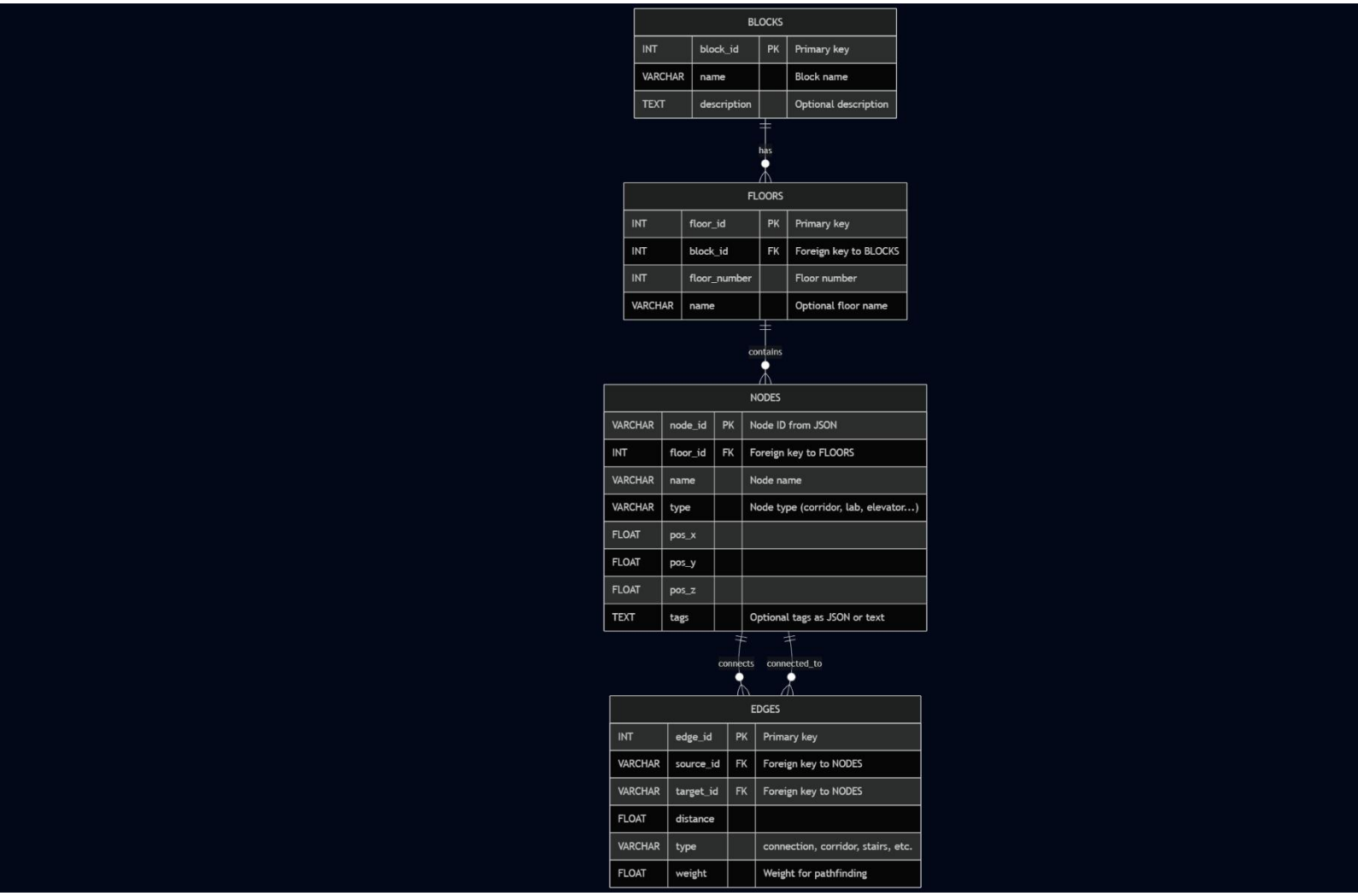


Figure 7 - Sensor Fusion Flowchart

9. API Endpoints

- POST /api/v1/fingerprint/upload — upload labeled Wi-Fi scans (for training).
- GET /api/v1/map/{map_id} — retrieve map graph JSON.
- POST /api/v1/position/estimate — send Wi-Fi sample (and optionally IMU); returns estimated position.
- WebSocket /ws/live/{device_id} — real-time telemetry and navigation messages.

Document expected request/response bodies (include schemas above).

---

10. Error Handling & Failure Modes

- **Camera unavailable:** notify user via TTS "Camera is not available" and fall back to navigation-only mode (if GPS/Wi-Fi data exist).
- **Model loading failure:** attempt reload; if fails, switch to conservative audio-only guidance (e.g., beep patterns triggered by user input).
- **Network loss:** buffer telemetry locally, send when connected. Server should be stateless for real-time messages.
- **IMU drift:** mitigate via periodic Wi-Fi corrections; if Wi-Fi unavailable, warn user about increased uncertainty.

11. Performance Targets & Testing Plan

11.1 Performance Targets

- **Vision pipeline latency:** end-to-end (camera capture → audio cue) ≤ 250 ms for single detection on supported devices.
- **Localization update rate:** IMU updates at ≥ 20 Hz; Wi-Fi fingerprinting estimates every 2–5 s.
- **Navigation responsiveness:** instruction updates whenever user deviates or every 1–2 seconds.

11.2 Unit & Integration Tests

- Unit tests for:
    - Wi-Fi vector normalization and KNN classifier.
    - Kalman filter predict/update correctness.
    - Bounding box NMS correctness.
- Integration tests:
    - Camera→Detector→Audio end-to-end latency measurement (instrument timestamps).
    - Navigation: simulate recorded Wi-Fi/IMU traces and validate position accuracy (RMSE).
- Field tests:
    - Collect wardriving data in target building: at least 20 samples per node, across different times of day.
    - Walk tests: measure path guidance correctness and user-perceived utility (qualitative).

11.3 Evaluation Metrics

- **Detection:** mAP (mean Average Precision) on person class for mobile model variant (measured on small test set).
- **Localization:** mean localization error (meters), 50th/90th percentile.
- **Latency:** median and 95th percentile of inference→audio latency.

---

12. Security, Privacy & Ethics

- **Privacy:** Raw camera frames are not uploaded by default. If server upload is enabled for debugging/training, explicit user consent must be captured and frames anonymized (blur faces or strip metadata).
- **Data Protection:** Use TLS for all server comms; encrypt sensitive payloads at rest if stored.
- **Ethics:** Ensure system alerts are non-misleading and clearly communicate uncertainty ("estimation might be imprecise").
- **User Control:** Provide settings to adjust beep volume, detection sensitivity, and to opt out of telemetry.

---

13. Deployment & Maintenance (Expanded)

- **Client Releases:** Use semantic versioning; keep backwards compatibility for navigation instruction schema (support older protocol_version).
- **Model Updates:** Distribute model updates via app release or optional model download; ensure atomic swap and rollback on failure.
- **Backend:** Containerize FastAPI with Docker; maintain CI/CD pipeline for tests and deployment.
- **Monitoring:** Collect anonymized telemetry for performance monitoring (inference latency, dropped frames) with opt-in.
