Universidade Tecnológica Federal do Paraná – UTFPR
Computer Engineering
Oficina de Integração 3 (ELEX22) - 2025.2

Technical Report

InLoco – In The Right Place

Gabriel Moro Conke – gabrielconke@alunos.utfpr.edu.br
Patrick Gonçalves Razzoto – patrickrazzoto@alunos.utfpr.edu.br
João Pedro Maciel de Souza – joaosouza.2020@alunos.utfpr.edu.br
Luiz Augusto Maués Noronha Filho – luizfilho.1999@alunos.utfpr.edu.br
Pedro Henrique Góes dos Anjos – pedrohenriqueanjos@alunos.utfpr.edu.br

November 27, 2025

Abstract

Indoor navigation is still a major barrier for blind and visually impaired people, especially in large public buildings where GPS is unavailable and physical accessibility infrastructure is often incomplete or inconsistent. This report presents InLoco – In The Right Place, an indoor navigation system that combines WiFi fingerprinting, a mapping robot and an accessible Android application to provide turn-by-turn audio guidance to visually impaired users. The system operates in two phases: in an offline setup phase, a mobile robot equipped with LiDAR, inertial sensors and a WiFi adapter generates an obstacle map of the environment and collects signal fingerprints; in the online phase, a smartphone app sends WiFi scans and heading information to a cloud server, which estimates the user's position, computes a route to a selected point of interest and returns spoken navigation instructions. The report describes the system requirements, architecture, technologies used and implementation, and discusses the prototype deployment in a university building, highlighting the strengths, current limitations and opportunities for future improvements of the InLoco approach.

1 Introduction

Indoor navigation in large public buildings remains challenging for blind and visually impaired people because GPS is unavailable indoors and visual landmarks are inaccessible. Existing solutions either require costly infrastructure (tactile paving), provide static information (Braille signage), or depend on human assistance, limiting autonomy and scalability [1, 2, 3].

InLoco addresses this gap with a low-cost, infrastructure-light approach that leverages existing WiFi access points and commodity smartphones. The system operates in two phases. In an offline setup, a small robot builds a 2D obstacle map with LiDAR and collects WiFi fingerprints across navigable areas. In the

online phase, an Android app periodically scans nearby WiFi networks and reads the phone's heading; a cloud service estimates the user's position via fingerprint matching, computes a route to a selected Point of Interest (POI), and returns turn-by-turn audio guidance [4, 5, 6, 7].

The architecture integrates embedded robotics for mapping, a spatial database for storing maps, fingerprints, and POIs, and server-side localization and A* routing exposed to a voice-first Android interface. We deployed a prototype in a university building and evaluated end-to-end performance. Results show that the full pipeline—mapping, fingerprint collection, routing, guidance, and re-routing—works reliably, with average WiFi-based localization error around 4 meters. While this validates feasibility, further accuracy improvements are needed to meet doorway-level guidance for blind users. Subsequent sections summarize requirements, implementation and results.

2 System Requirements

This section presents the functional requirements of the InLoco. Non-functional and anti-requirements are available on the project blog [8]. The system is organized into three layers — Mapping, Data, and Application.

2.1 Mapping Layer Functional Requirements

M-FR-1 The robot must be able to navigate the environment via remote control in order to generate the "Obstacle Map".

M-FR-2 The robot must generate a two-dimensional occupancy grid map of the environment's static objects with an Intersection over Union (IoU) index greater than 0.75, constituting the "Obstacle Map".

M-FR-3 The robot's navigation and mapping functions are designed to operate exclusively on flat, planar surfaces.

M-FR-4 After an Obstacle Map is generated, the system must create a route for the robot that covers the navigable area with a grid of measurement points. The collected WiFi data at these points will constitute the "WiFi Signals Map" layer [5, 9].

M-FR-5 The robot must be able to move autonomously during the creation of the "WiFi Signals Map", covering every point defined.

M-FR-6 The robot must be able to scan at least 5 and at most 10 WiFi signals at the same time in each defined point, capturing the BSSID and RSSI for all detectable networks with a signal strength greater than -80 dBm [10, 11].

M-FR-6.1 If more than ten WiFi signals are detected, the ten strongest will be selected.

M-FR-7 The robot must store the captured Wi-Fi and mapping data in its internal memory.

2.2 Data Layer Functional Requirements

D-FR-1 The robot must be able to transmit the scanning and measurement data to a cloud server after finishing the mapping and measurements.

D-FR-1.1 Upon successful completion of the mapping tasks, the robot shall automatically attempt to connect to a pre-configured WiFi network and transmit all collected data (map, Wi-Fi fingerprints) to the cloud server endpoint.

D-FR-1.2 If transmission fails, the robot must retry up to 5 times at 2-minute intervals. If it still fails, the data will be held locally, and an error status will be indicated.

D-FR-1.3 In the event of persistent transmission failure, the system must provide a documented method for manually extracting the data from the robot's local storage.

D-FR-2 The system must have desktop software with a graphical user interface (GUI) for defining points of interest (POIs) visually on the map using a computer mouse.

D-FR-2.1 The system will enforce that POIs have names limited to alphanumeric characters and spaces, and must be between 4 and 50 characters.

D-FR-3 The system must store the map scanned by the robot, the points of interest, and the measurements in a database.

D-FR-3.1 The database schema must support storing the occupancy grid map, a list of POIs with their (x,y) coordinates and unique names, and the associated WiFi fingerprint data (BSSID, RSSI) for each measurement point.

2.3 Application Layer Functional Requirements

A-FR-1 The system must have an Android mobile application as the user interface.

A-FR-2 The mobile application must transmit the orientation of the smartphone's compass and all Wi-Fi networks detected by the device to the server to estimate the user's position on the map, with the device held in hand, facing forward [4, 5].

A-FR-2.1 The mobile application must transmit a data packet containing the device's compass heading (in degrees) and a list of scanned Wi-Fi BSSIDs and their RSSI values to the server every 2 seconds.

A-FR-3 The mobile application must allow voice commands to request the calculation of a route from the user's position to a point of interest.

A-FR-4 The server must calculate a route and transmit it to the user's application.

A-FR-4.1 The server must calculate the route and transmit the next instruction from the user's estimated position to a selected POI within 5 seconds of receiving the initial request.

A-FR-4.2 The server must generate a static waypoint graph from the map and POI data before any route calculation.

A-FR-4.2.1 The nodes of the waypoint graph shall consist of the (x,y) coordinates of all defined Points of Interest (POIs).

A-FR-4.2.2 An edge connecting two nodes shall be created in the graph if and only if the straight-line path between them does not intersect with any inflated obstacle cells on the pre-processed map.

A-FR-4.3 For each navigation request, the system must identify the waypoint graph node closest to the user's current estimated position to serve as the starting point for the A* pathfinding algorithm.

A-FR-5 The app provides the user with directions in meters, indicating the distance in meters to the next turn.

A-FR-6 The app must demand the user to stop if the user goes out of the instructed path; after that the server must recalculate the route based on the current position of the user within 5 seconds of detecting the deviation.

A-FR-7 The application must indicate the direction of the point of interest at the end of the route.

A-FR-8 The application shall support specific voice commands such as "Where am I?", "Take me to [destination]", "Repeat", "Let's go", "Cancel" and "What's near me?".

A-FR-9 The app shall confirm the selected destination with the user before starting navigation.

A-FR-10 In the event of duplicate POIs with the same name, the app shall consider only the one closest to the user's current position.

3 Technologies Used

3.1 Hardware

The Mapping layer is implemented by a custom two-wheel differential drive robot designed for indoor environments. Its main hardware components are:

Single-board computer (SBC) Raspberry Pi 5 – 4 GB RAM, used as the main onboard computer, running the high-level navigation and data collection stack [12].

Microcontroller ESP32-C3, responsible for low-level motor control and real-time sensor reading, under supervision of the Raspberry Pi [13].

Motion and drive Two DC motor with magnetic encoders integrated + wheel assemblies providing differential drive locomotion; DC motor driver L9110; and one caster wheel for passive support and balance.

Sensors YDLIDAR X2L 2D LiDAR for 360° obstacle detection and occupancy grid generation; IMU GY-BNO055 (9-axis absolute orientation sensor); IR sensor TCRT5000 facing downward to detect stairs and drops; and a USB

Wi-Fi Adapter Comfast CF-922AC to scan surrounding access points and measure RSSI.

Power supply Four 21700 Li batteries powers the Raspberry Pi 5 and digital electronics, while four rechargeable batteries Ni-MH AA with 1,2V supply the motor power stage, separating logic and actuation power.

3.2 Software

Operating system Linux distribution for Raspberry Pi 5.

Robotics framework ROS 2, used as middleware for node-based development.

Core packages and nodes `slam_toolbox` to perform 2D SLAM and generate the obstacle map [14]; NAV2 stack to autonomously drive the robot through a lawnmower-style measurement grid [15]; a custom WiFi scanner node to record BSSID and RSSI at each grid point; and a custom data logger/uploader node to store mapping and fingerprint data locally and send it to the cloud server.

Low-level firmware Custom firmware on the ESP32-C3 DevKit for motor control and sensor acquisition.

Server Python-based backend running on the Render cloud platform, with a PostgreSQL.

APP Android application written in Kotlin

4 System Overview

The InLoco system is organized into three main layers: Mapping, Data, and Application. In general terms, the process begins by constructing a robot that is remotely controlled by an operator. As the robot moves through the environment, it creates a map of the space. After mapping, the robot performs an autonomous navigation routine, scanning the Wi-Fi signals it detects and uploading this information to a cloud server. This stage is known as the offline mapping phase.

Once this phase is complete, a user can interact with a mobile app using sonic commands which will guide the user through the environment. This stage is known as the online navigation phase.

4.1 Offline Mapping Phase

In the offline phase, an operator deploys the robot in the target indoor environment to build two map layers:

Obstacle Map generation. The operator first teleoperates the robot through the environment while the onboard LiDAR, encoder and IMU, processed by an algorithm called `slam_toolbox`, generates a 2D occupancy grid map. This map represents walls and static obstacles.

WiFi Signals Map generation. Once map has been obtained, the system automatically defines a measurement grid covering the free space using a lawnmower coverage pattern. The robot then autonomously navigates to each grid point. At every point, it performs a WiFi scan with the WiFi adapter, collecting the BSSID and RSSI of all visible networks. These measurements are stored along with the corresponding (x, y) coordinates, forming the WiFi Signals Map or fingerprint database [5, 9, 10, 11].

When the scanning is complete, the robot uploads the obstacle map and WiFi fingerprint data to the cloud server. Using a desktop GUI, an administrator then defines Points of Interest (e.g., classrooms and bathrooms) [16].

4.2 Online Navigation Phase

In the online phase, the mobile robot is no longer needed. Visually impaired users interact exclusively with their smartphone.

Initialization and localization. Upon entering the mapped environment, the user launches the InLoco app and issues a voice command such as "Where am I?" or "Take me to [destination]". The app periodically scans the surrounding WiFi networks and reads the device's compass heading, assuming the phone is held in the walking direction. This information is sent to the server, which uses a K-Nearest Neighbors (KNN) algorithm over the WiFi fingerprint database to estimate the user's current position [11, 4].

Route planning. Once a destination is confirmed, the server projects the user's estimated position onto the waypoint graph and runs an A* pathfinding algorithm to compute the shortest path to the chosen POI. The resulting path is converted into a sequence of high-level navigation instructions. [6, 7].

Guidance and re-routing. The mobile app receives the instructions, converts them into speech using Text-to-Speech and guides the user step by step [17]. .

5 Development / Implementation

This section will describe in detail the mechanical, electronic, software and integration aspects of the project implementation.

5.1 Robot

5.1.1 Mechanical

We began by developing the robot's mechanical design with a focus on simplicity, stability, and clean sensor visibility. The chassis uses a two-layer layout: the lower deck houses the drivetrain, power, and electronics, while the upper deck

isolates the LiDAR. For mobility, we selected a differential-drive configuration with two powered wheels and a rear caster, balancing ease of implementation and mechanical stability. Reference photos and detailed blueprints of the initial design are provided below.

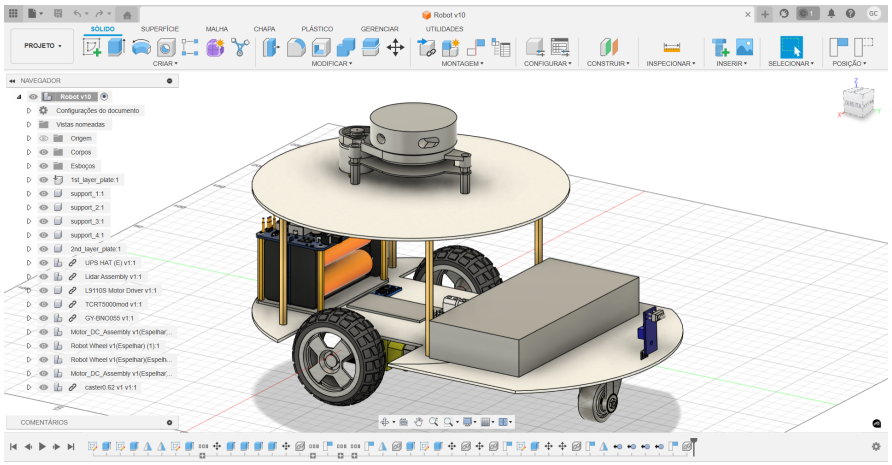


Figure 1: Mechanical overview design

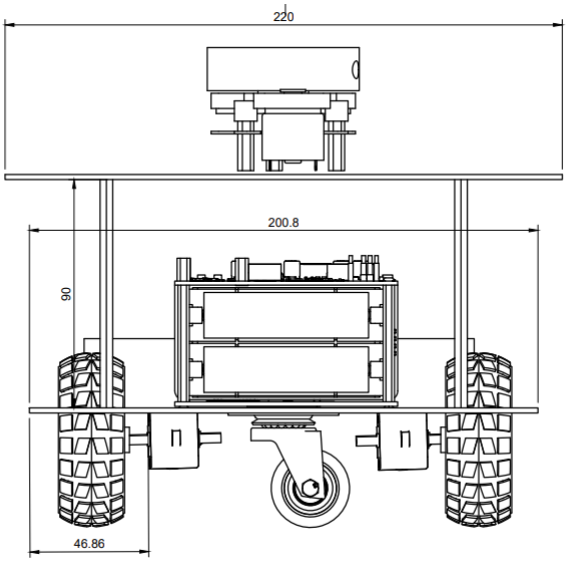


Figure 2: Mechanical rear blueprint

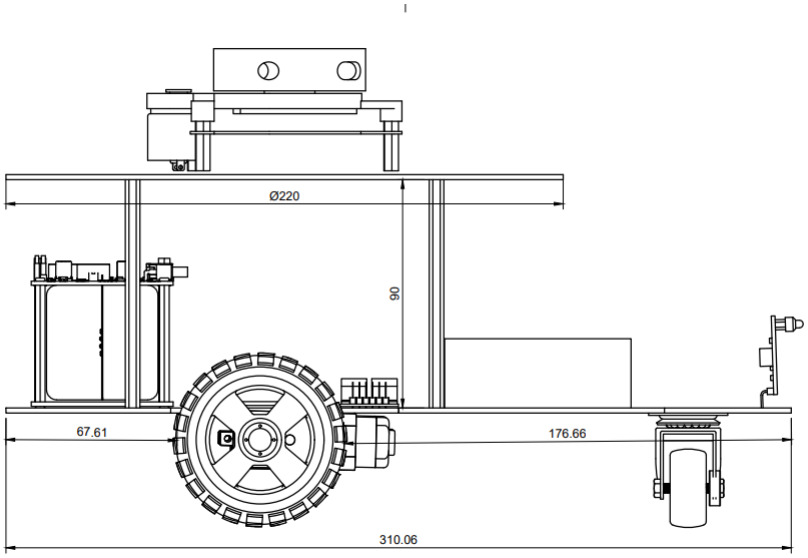


Figure 3: Mechanical side blueprint

During construction, we decided to increase the size of the robot's structure and plates by approximately 1.3× to make electronic connections and circuits easier to assemble. The final version of our mechanical design is shown in the image below.

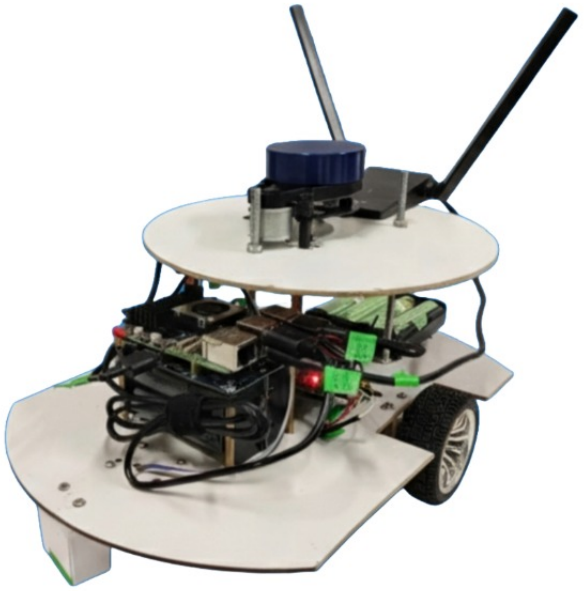


Figure 4: Final mechanical design

5.1.2 Electronic

The electronic design of the robot is based on a dual-power architecture combined with a distributed processing scheme. A Raspberry Pi 5 operates as the main computation unit, responsible ROS2 management, while an ESP32 is used for motor actuation.

Two independent power sources were implemented, one dedicated to the DC motors and another exclusively for the Raspberry Pi and logic-level electronics. The motors introduce electrical noise and current spikes, especially during acceleration, direction change and sudden load. If both systems shared the same supply, these fluctuations could propagate to the CPU and potentially result in instability or unexpected resets.

Between the battery and the distribution network, a voltage regulation module was placed to ensure a constant output. This protects the ESP32 from undervoltage events and ensures stable CPU operation under continuous processing load, minimal electrical noise affecting peripheral readings, motor current peaks without disturbing logic voltage levels. Figure 5 shows the full electronic topology

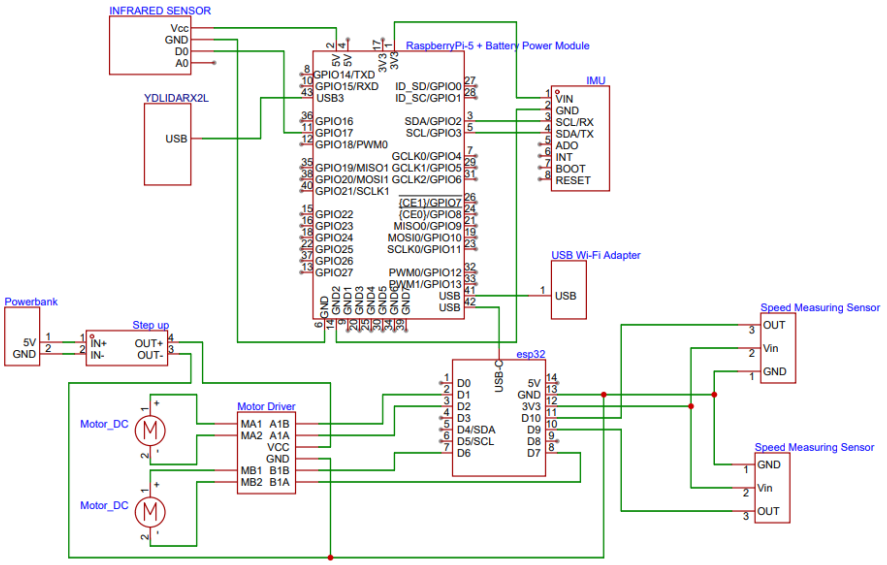


Figure 5: Complete electronic diagram

5.1.3 Software

The L9110S interfaces the microcontroller and DC motors, reversing polarity for forward/backward motion and accepting PWM directly on its inputs. PWM (Pulse-Width Modulation) provides speed control and the duty cycle (ON time percentage) sets the motor's average power.

The goal is to make motor speed follow a target setpoint. The code reads an encoder, computes error (setpoint measurement), and drives PWM duty via a PID controller in a fixed-rate loop for fast response. Robot motion is coordinated by a Raspberry Pi 5 running ROS 2, integrating high-level planning with low-level control [12, 18].

The robot operates in a three-phase workflow: map creation, autonomous navigation and data collection. First, operator manually drive the robot through the environment using keyboard control while SLAM Toolbox creates detailed maps in real-time. Once maps exist, the robot navigates autonomously to waypoints, collecting WiFi fingerprints at each location [14, 15]. Besides that, the robot has a IR sensor that prohibit it to go down stais.

SLAM Toolbox is a ROS2 package that performs graph-based SLAM, creating maps by: matching current LiDAR scans to previous scans to track movement, detecting when robot returns to previously visited locations, optimizing the entire map to correct accumulated errors and creating a 2D grid map showing free space, obstacles, and unknown areas [14].

The robot maintains awareness of its position through a combination of methods: (1) Map-Based Localization: the robot uses a pre-built map of the environment and matches its current LiDAR scan against this map to determine its position. [14, 15]. (2) Odometry Tracking. (3) Initial Pose: when starting, the robot must be told its starting position. (4) Position Updates: the robot's position is updated at approximately 10-20 Hz, combining odometry with localization.

After the map being created, it's necessary to run a script to create measurements points. In summary, this points are locations where the robot will stop during autonomous navigation and scan wifi signals (waypoints). Nav2 converts the map into motion. Given a goal, it plans a global path on the occupancy grid and uses local controllers to follow it. The global planner proposes the path; the local controller adjusts speed and heading for corners and narrow passages. [15, 18].

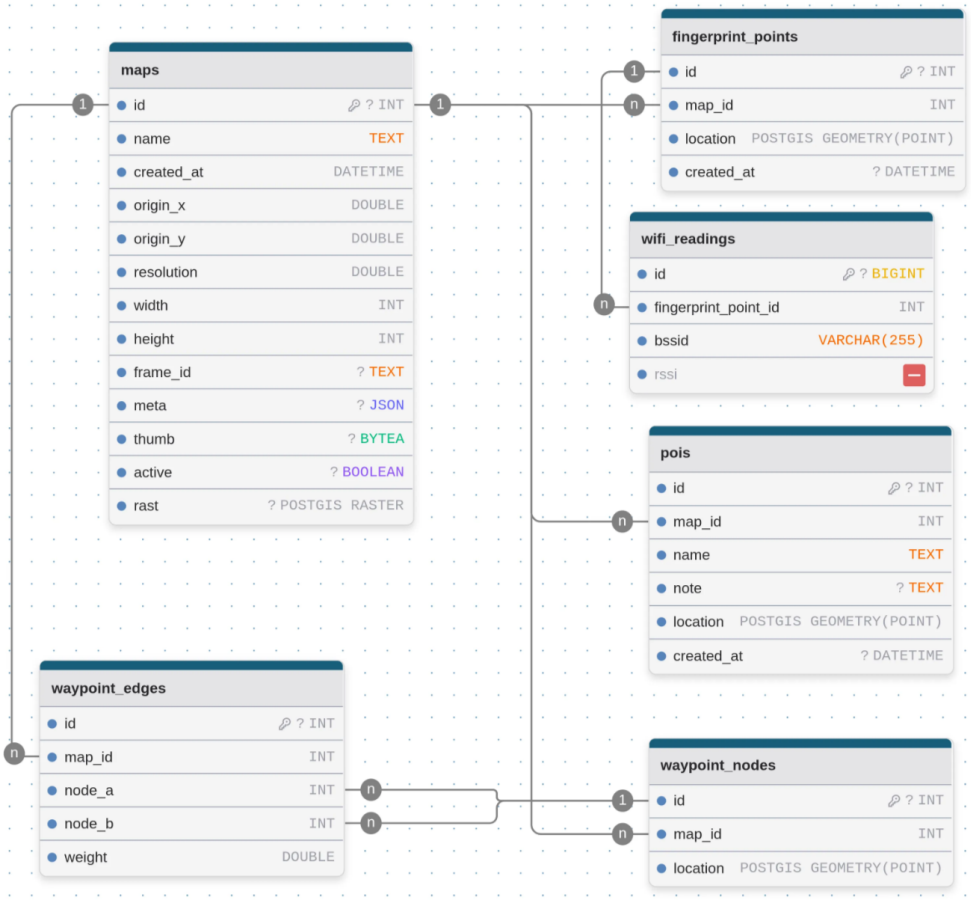
WiFi scanning is synchronized with navigation. The Nav2 confirms arrival within tolerance and a full stop, so the robot records the strongest access points. This stop–scan–resume routine ties each fingerprint to a precise, repeatable map coordinate, improving localization accuracy [10, 11]. After all the points scanned, the robot upload the map and wifi informations collected to server to be used later on.

A desktop GUI is provided to allows administrators to create and manage Points of Interest (POIs) [16]. These point will be used in the future to be the possible destinations from a navigation.

5.2 Server

5.2.1 Database Model

Render was chosen as the cloud server because it met our requirements while remaining free. Once defined, the database was modeled, so a foundation was established upon which the server could be built. PostgreSQL was chosen as the database system due to its support for geographic data through PostGIS.



5.2.2 System Architecture

The server implements a three-layer architecture: (1) Controller Layer (FastAPI routers): handles HTTP request/response lifecycle, input validation via Pydantic and error handling.(2) Service Layer: contains core algorithms and domain logic. (3) Repository Layer (data access): abstracts database operations and query construction. [19].

5.2.3 Core API Endpoints

- Localization (POST /app/locate): recieve WiFi scan as array of BSSID/RSSI pairs, applies RSSI normalization, executes COSINE similarity and TOP5 KNN metrics in parallel, computes confidence-weighted, applies grid constraint to snap position to walkable areas within 3m of calibration points and returns position with confidence metric [5, 9, 11].

- Navigation Initialization (POST /navigation/start_mission): (1) recieve current WiFi scan and destination POI name. (2) Localize current position. (3) Query POI coordinates. (4) Generate waypoint grid from occupancy map. (5) Build graph with orthogonal + diagonal connections validated by line-of-sight checks. (6) Execute A* pathfinding. (7) Initialize PathFollower and LandmarkNavigator for decision-point instructions. (8) Create in-memory navigation session with UUID. (9) Calculate required compass heading. (10) Returns session ID, initial instruction and required heading. [15, 19].

- Position Update (POST /navigation/update_position): (1) Accepts session ID and current WiFi scan. (2) Localize current position. (3) Apply confidence filter (reject if <15%). (4) Validate velocity (reject if movement exceeds reasonable walking speed). (5) Project 2D WiFi position onto 1D path distance using PathFollower. (6) Check off-route condition (distance from path >6m for 3 consecutive readings triggers recalculation). (7) Query LandmarkNavigator for instruction at current distance along path. (8) Return instruction only if at turn or arrival, otherwise return distance to next landmark.

- Calibration Data Upload (POST /wifi-scan): bulk upload endpoint for WiFi fingerprint collection.

5.2.4 Localization Algorithms

The localization system implements confidence-weighted averaging of the COSINE similarity and TOP5 strongest signals KNN metrics. When COSINE produces low confidence due to weak signal ambiguity, TOP5's strong signal focus dominates the weighted average. Conversely, when TOP5 confidence drops due to AP set changes, COSINE's matching stabilizes the position estimate.[5, 9, 4].

Let P_c and P_t represent position estimates from COSINE and TOP5 with confidences γ_c and γ_t. The ensemble is

$$P_{ensemble} = \frac{P_c \times \gamma_c + P_t \times \gamma_t}{\gamma_c + \gamma_t},$$

The COSINE metric treats WiFi scans as vectors in high-dimensional RSSI space and measures angular similarity between the user scan and database fin-

gerprints. RSSI values undergo normalization using database-wide extrema:

$$RSSI_{norm} = \frac{RSSI - global\_min}{global\_max - global\_min}.$$

Each WiFi scan transforms into a sparse vector where dimensions correspond to unique BSSIDs in the union of the user scan and database fingerprints. Missing BSSIDs (visible in the database but not in the current scan, or vice versa) are zero-padded.

For user vector V_u and database vector V_d, compute

$$similarity = \frac{V_u \cdot V_d}{\|V_u\| \times \|V_d\|},$$

with result in $[-1, 1]$. Convert similarity to distance via

$$distance = (1 - similarity) \times 50.$$

Distances are computed to all fingerprints and the five nearest neighbors are selected. The position estimate uses

$$weight_i = \frac{1}{distance_i^2 + \varepsilon},$$

with $\varepsilon = 10^{-6}$, giving much higher weight to closer neighbors [5, 9].

The TOP5 algorithm focuses on the 5 strongest access points per scan, removing weak signals. It computes Euclidean distance on normalized RSSI values over the intersection of top-5 BSSIDs between user scan and fingerprint and applies a penalty when there are fewer than 5 common signals. [11]. After ensemble averaging, the resulting (x, y) position may lie in non-walkable areas. The grid constraint system snaps positions to nearby fingerprint points [4, 5].

5.2.5 Route Planning

For each WiFi-derived position U and path segment from P_1 to P_2, compute

$$t = \frac{(U - P_1) \cdot (P_2 - P_1)}{\|P_2 - P_1\|^2}, \quad t \in [0, 1]\ (clamped),$$

then project onto the segment as

$$P_{proj} = P_1 + t\,(P_2 - P_1).$$

Route planning applies A* on a waypoint graph generated from the occupancy grid. Waypoints are sampled in free space at $0.7\,m$ spacing with a conservative occupancy threshold and a 5-pixel safety margin from walls. Orthogonal and diagonal edges are admitted only if line-of-sight is clear, verified via a Bresenham-like raster check. A* uses

$$f(n) = g(n) + h(n),$$

with h the Manhattan-distance heuristic, while edge cost is the Euclidean distance augmented by a fixed turn penalty of 100 units whenever the travel direction changes between consecutive edges, encouraging smoother routes [15]. Graph connectivity is validated via BFS; disconnected components indicate configuration issues and the system retains the largest connected component for planning.

Turn-by-turn guidance is produced at landmarks (turns and arrival) which are detected where the angle between consecutive segment directions exceeds 30°. Off-route is detected if the perpendicular distance from U to the path exceeds 6 m for three consecutive readings [4].

5.2.6 Software Complexity

Fingerprint data load on the first localization request (cold start ≈ 5–8 s) and are then cached, yielding warm responses in 100–200 ms. For navigation, each route loads the occupancy grid and POIs once; computed polylines and landmarks remain in session memory so subsequent updates use cached artifacts. Per request, fingerprint distance evaluations are $O(n)$ with K-NN selection $O(n \log K)$; A* on the waypoint graph runs in $O(m \log m + e)$. Projecting a WiFi position onto a route polyline is $O(s)$ [4, 15].

5.3 APP

InLoco has a voice-controlled Android application that enables indoor navigation using WiFi fingerprinting [4, 5, 11]. The application operates through spoken commands, which is essential for accessibility [6, 7].

The app general flow is: (1) User taps screen, (2) App displays "Listening..." feedback, (3) User speaks a command (4) App converts speech to text (5) System interprets and executes the appropriate action (6) Response is spoken back [17]. When the user speaks, the system analyzes the text normalizing it. If the system doesn't understand a command, it responds with: "Sorry, I didn't understand. Please repeat."

There is a list of voice commands supported by the app:

- **Location Queries:** - "Where am I?" - App scans WiFi, determines location, and announces nearest point of interest. "What's near me?" - Lists the 5 closest POIs
- **Navigation Commands:** - "Take me to [x]" - Initiates navigation to specified location - "Cancel" - Stops active navigation session - "Repeat" - Repeats the last spoken instruction

The diagram bellow shows what happens in each one of the possible commands.

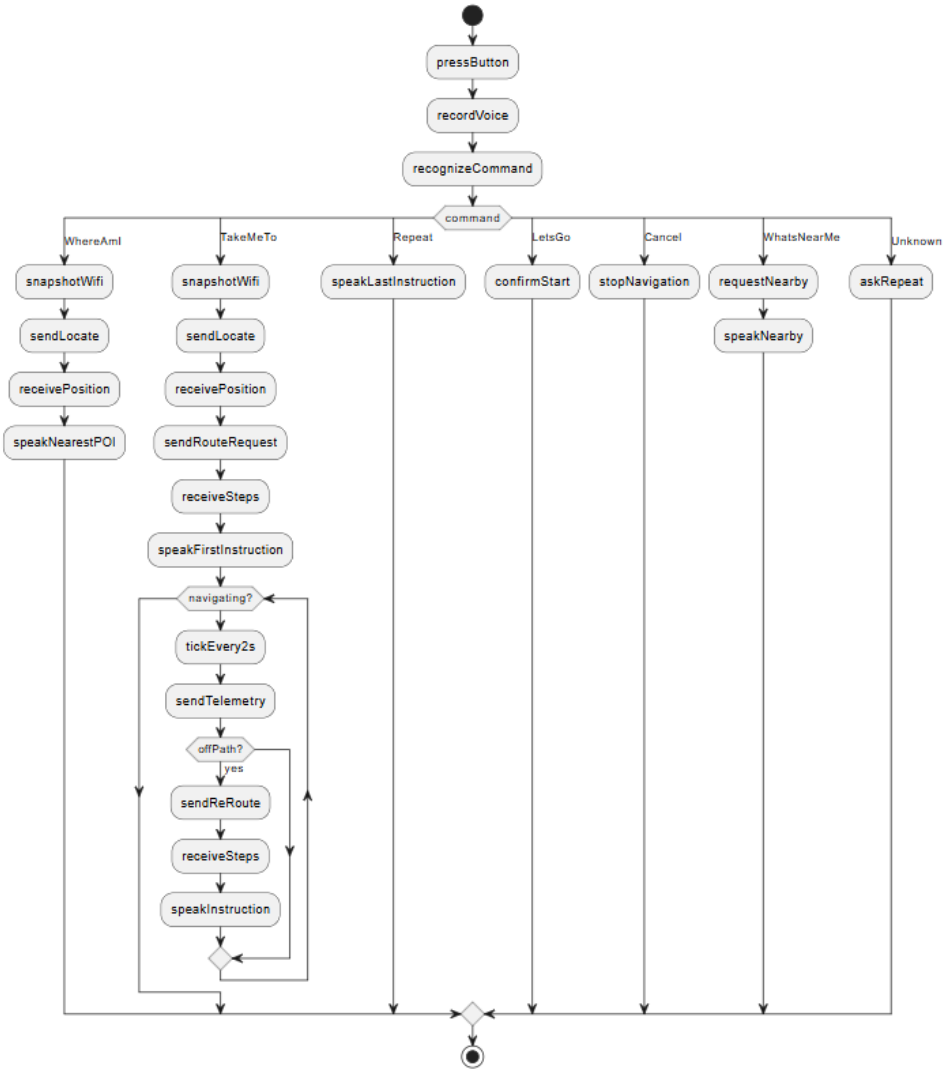


Figure 6: APP - Process Diagram

User's initial orientation affects navigation, so it was implemented a alignment step before starting. System tells the user how many degrees they need to rotate to face the correct starting direction. Instructions are given in 30-degree increments, with a tolerance of ±15 degrees. When the app needs to determine location, it activates phone's WiFi radio, detects all nearby WiFi access points signals, select the 15 strongest and send to server.

InLoco is built to be simple and responsive: a single main screen coordinates everything. Instead of running on timers, it reacts to events: when speech finishes or new readings arrive, the next step happens immediately. This keeps things fast, consistent, and easy to follow.

The app is written in Kotlin and uses WiFi and sensors for positioning and text-to-speech for guidance. The interface is lightweight—simple XML layouts. To talk to the server, it sends small JSON messages over HTTPS using a reliable networking library. An image off app interface can be seen below

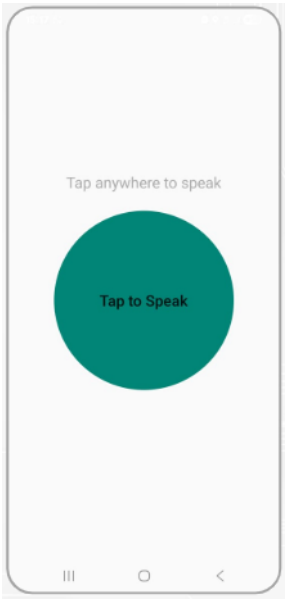


Figure 7: APP - Final interface

6 Tests and Results

Integration and field tests evaluated end-to-end behavior across the three layers. Mechanically, the robot was robust: chassis, drivetrain, and LiDAR placement remained stable during teleoperated mapping and autonomous coverage. Electronics were dependable; after organizing wiring and separating power domains on a universal PCB, no hardware faults occurred, improving assembly cleanliness and maintainability.

On sensing and navigation, wheel encoders were the most time-consuming issue. The initial infrared encoder was unsuitable and inconsistent; switching to magnetic encoders resolved signal quality and enabled accurate closed-loop control. Synchronizing encoders, LiDAR, IMU, and Nav2 required iterative calibration. Google Cartographer yielded slightly better SLAM maps, but SLAM Toolbox integrated more smoothly with Nav2 and was adopted. The robot then executed lawnmower-style coverage to collect WiFi fingerprints.

On the server and data side, receiving fingerprints, building maps/waypoint graphs, and answering localization/routing requests worked as designed. Route computation met latency expectations, and polylines were reliably transformed into concise, turn-by-turn instructions. In mobile app, voice interaction was

straightforward. Phone compass required per-device calibration and exhibited device-specific biases; after calibration, headings were precise.

Quantitatively, WiFi localization with cosine similarity alone achieved mean error 4 m, with worse outliers at specific locations. To reduce worst-case errors, a confidence-weighted blend of cosine and TOP5-strongest-signals was used; this improved maximum-error behavior but slightly increased mean error relative to cosine alone. Route generation and periodic WiFi scans stayed on schedule; instruction delivery, re-routing triggers, and session management behaved as planned. Overall, nearly all planned features worked end-to-end, fulfilling about 95% of requirements. However, average localization precision around 4 m remains insufficient for safe, production-grade, turn-by-turn guidance for blind users; tighter positioning is needed to ensure instructions occur exactly where required.

7 Challenges and Limitations

The main challenges centered on synchronizing sensors and achieving practical indoor localization: while the mechanics and electronics were reliable, the first infrared encoder proved inadequate, prompting a switch to magnetic encoders and careful decoding, and integrating encoders, LiDAR, IMU, and Nav2 required iterative calibration to align timing and behavior [14, 15]. For SLAM, Cartographer produced slightly sharper maps, but SLAM Toolbox offered smoother Nav2 integration, so we prioritized system stability over marginal map quality [14, 18]. On the app side, voice interaction was straightforward, but heading estimation exposed a real-world constraint: smartphone compasses need user calibration and exhibit device-specific biases, which, despite yielding precise headings after calibration in tests, adds friction and variability that can affect instruction correctness across different phones [4].

The core limitation is localization precision from WiFi fingerprinting in our current setup. Despite reliable route generation and instruction logic, average errors around 4 meters—and occasional larger spikes at particular points—make it difficult to trigger turns and landmark instructions with the spatial fidelity required by blind or low-vision users [3, 5, 9]. The ensemble of cosine and TOP5 reduced worst-case outliers but did not sufficiently drive average error into the sub-meter-to-2-meter range generally needed for confident doorway-level decisions. Consequently, even with the full feature set functioning and responsive, the present accuracy does not yet meet the usability threshold for independent indoor navigation by blind users [4, 11].

The limiting factor is localization accuracy, not functionality. Future work should prioritize improvements to positioning. Addressing this single bottleneck would likely unlock the full value of the solution and make the guidance precise enough for real-world, accessibility-first navigation.

8 Budget

Table 1: Bill of Materials

Component	**Full Cost**
Raspberry Pi 5 – 8 GB RAM	R$420,00
YDLIDAR X2L	R$318,00
USB Wi-Fi Adapter – TP Link T4U Plus	R$165,00
18650 Batteries	R$80,00
Raspberry Pi Battery Power Module	R$250,00
21700 Batteries	R$120,00
ESP32-C3 Devkit	R$20,00
Motor + Wheel	R$120,42
Caster Wheel	R$4,90
Driver motor DC duplo – L9110S	R$10,00
IMU GY-BNO055	R$56,00
IR Sensor TCRC5000	R$12,40
Total	**R$1 576,72**

9 Schedule and Effort

Table 2: Planned vs. Worked Hours by Area

Area	**Planned (h)**	**Worked (h)**	**Delta (h)**
Mechanical	17	33	+16
Electronic	33	48	+15
Embedded	124	205	+81
Software	190	135	-55
Total	**380**	**444**	**+64**

Overrun concentrated in the Embedded stream, caused by encoder replacement and tuning together with closed-loop motor control and the synchronization of encoders, LiDAR, IMU, and Nav2. Software finished under budget because voice interaction and service endpoints stabilized early. Time planned and spend for each task can be seen with more details in the project blog [8].

10 Conclusion

This work delivered an end-to-end prototype for indoor navigation that combines robotic mapping, WiFi fingerprinting, cloud services, and an accessible Android application. The platform met approximately 95% of the requirements and validated the feasibility of a low-cost, infrastructure- light approach tailored to large public buildings.

Testing showed that the full feature set—mapping, fingerprint collection, localization, routing, re-routing, and voice guidance—operated reliably. However, the average localization error around 4 m remains insufficient for safe, doorway-level instructions for blind users, despite improvements from combining cosine similarity with a TOP5 ensemble.

Future work should prioritize accuracy: denser and better-controlled fingerprinting, stronger map matching, temporal filtering, and optional infrastructure such as BLE or UWB in critical zones. With localization precision improved to the 1–2 m range, the architecture and user experience demonstrated here can translate into practical, independent indoor navigation.

References

[1] Donatella Pascolini and Silvio Paolo Mariotti. Global estimates of visual impairment: 2010. *British Journal of Ophthalmology*, 96(5):614–618, 2012.

[2] World Health Organization. Global data on visual impairments 2010. Technical report, World Health Organization, Geneva, Switzerland, 2012. WHO/NMH/PBD/12.01.

[3] Darius Plikynas, Antanas Zvironas, et al. Indoor navigation systems for visually impaired persons: Mapping the features of existing technologies to user needs. *Sensors*, 20(3):636, 2020.

[4] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials*, 21(3):2568–2599, 2019.

[5] Shuo Shang et al. Overview of wifi fingerprinting-based indoor positioning. *IET Communications*, 16(7):689–706, 2022.

[6] Dragan Ahmetovic, Cole Gleason, Shiri Azenkot, et al. Navcog: A navigational cognitive assistant for the blind. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 90–99. ACM, 2016.

[7] Daisuke Sato, Uran Oh, Hironobu Takagi, et al. Navcog3: An evaluation of a smartphone-based blind indoor navigation assistant with semantic features in a large-scale environment. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 270–279. ACM, 2017.

[8] InLoco Team. Inloco – in the right place project blog. `https://www.notion.so/InLoco-In-The-Right-Place-250687c31ecb80bca899df1fb842360a`, 2025. Accessed: 26 Nov. 2025.

[9] Sheng Xia, Yong Li, Guojun Luo, et al. Indoor fingerprint positioning based on wi-fi: An overview. *ISPRS International Journal of Geo-Information*, 6(5):135, 2017.

[10] Paramvir Bahl and Venkata N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *Proceedings of IEEE INFOCOM*, pages 775–784. IEEE, 2000.

[11] Samaneh Amirisoori, Salwani Mohd Daud, and N. A. Ahmad. Wi-fi based indoor positioning using fingerprinting methods (knn algorithm) in real environment. *International Journal of Future Generation Communication and Networking*, 10(9):23–36, 2017.

[12] Raspberry Pi Ltd. *Raspberry Pi 5 Product Brief*, 2023. `https://www.raspberrypi.com/products/raspberry-pi-5/`.

[13] Espressif Systems. *ESP32-C3 Series Datasheet*, 2020. `https://www.espressif.com/en/products/socs/esp32-c3`.

[14] Steve Macenski et al. slam_toolbox: Real-time slam for ROS 2. `https://github.com/SteveMacenski/slam_toolbox`, 2025. Accessed: 26 Nov. 2025.

[15] Nav2 Contributors. Navigation2: The ros 2 navigation framework. `https://nav2.org/`, 2025. Accessed: 26 Nov. 2025.

[16] InLoco Team. Poi selector desktop gui. `https://poi-selector-desktop.onrender.com/`, 2025. Accessed: 26 Nov. 2025.

[17] Android Developers. Texttospeech class. `https://developer.android.com/reference/android/speech/tts/TextToSpeech`, 2025. Accessed: 26 Nov. 2025.

[18] Open Robotics. Ros 2 documentation. `https://docs.ros.org/en/rolling/`, 2025. Accessed: 26 Nov. 2025.

[19] PostGIS Project. *PostGIS 3 Reference Manual*, 2023. `https://postgis.net/documentation/`.