JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

AI-Based Real-Time Navigation and Obstacle Detection System for the Visually Impaired

[1]Dr. Suresha D, [2]Bhoomika P Sarvajna, [3]Lekhana Hugar, [4]Mamatha, [5]Pavithra N

Associate Professor, [2]Student, [3]Student, [4]Student, [5]Student
[1]Department of Computer Science and Engineering,
[1]Dr. Ambedkar Institute of Technology, Bengaluru, India
[1][suresha.cs@drait.edu.in], [2][bhoomikasarvajna@gmail.com], [3][lekhanahugar21@gmail.com], [4][mamatha27102003@gmail.com], [5][murthypavithra4@gmail.com]

***Abstract*—Moving around independently is one of the hardest challenges for people with visual impairment or people with very little to no vision. Everyday tasks like detecting objects, finding routes become risky and difficult. There are traditional solutions like white cane and trained dogs but they are costly, limited ranged. The current review compares various assistive technology and other solutions. This paper presents an Android application that combines obstacle detection, outdoor navigation, emergency mechanism together in one smartphone application. The user speaks the destination, mobile camera detects objects and announces the detected object and approximate distance using a pre-trained YOLOv8n model, and it obtains a walking route from OpenStreetMap-based services and announces each turn. For safety, the app sends an SMS to the saved contacts along with the user's location when an emergency is triggered and also when the battery drops below the threshold. This paper also describes the design and implementation of each feature, discusses the advantages and limitations of the application, and outline directions for future work.**

***Index Terms*—assistive technology, visually impaired, navigation, object detection, YOLOv8, voice interface, Android, OpenStreetMap, Artificial Intelligence, Obstacle Detection, Real-time Navigation.**

I. Introduction

Walking to an unfamiliar outdoor place requires continuous perception and decision-making so it can be a challenging task for visually impaired as they have limited environmental awareness. A sighted pedestrian combines landmarks, signs, traffic flow, behaviour of other people. But for visually impaired people these means of information becomes restricted or unavailable. They can resort to traditional existing means like white canes which doesn't work for distant object detection, trained dogs which are difficult to maintain and are not very efficient. There are other modern assistive devices that uses ultrasonic sensors, GPS, GSM modules and haptic devices but they are costly and unaffordable. This paper presents a cost-effective, user-friendly solution which is an Android application that runs on smartphone that already contains a camera, a microphone, a GPS receiver, a speaker and a means of sending messages. This work examines how far these can be combined in a single voice-operated application. One rule guided the design: the user should not have to move through many screens while walking. The application announces what it can do, listens for a short command, and returns most information as audio.

The application is written in Kotlin with Jetpack Compose and provides five functions:

1. Navigation: the user speaks a destination and the app gives spoken walking directions, turn by turn.
2. Obstacle detection: the phone camera detects objects ahead and says what they are, which side they are on and roughly how many steps away.
3. Saved places: the user can say the name of a place and be guided there after a quick yes or no.
4. Emergency: pressing and holding the screen for three seconds sends an SMS with the user's location to saved contacts.
5. Settings: a guardian can manage emergency contacts and saved places, and set the battery level at which a low-battery alert is sent.

The system is intended as an additional source of information, not as a replacement for a cane, a guide dog or mobility training. The paper provides a description of how the system works and a list of its weaknesses. Section II reviews related work, Section III states the design goals and Section IV describes the architecture. Section V covers the implementation, Section VI includes results and evaluation, and Section VII discusses the limitation and future enhancements. Section VIII covers the conclusion.

II. Literature Review

Development of assistive devices for people with impaired vision followed a number of development steps. Development of mobility aids started with a white cane and guide dogs and moved on to electronic travel aids with distance sensors, wearable devices with camera and deep learning capabilities, and finally smartphone applications. The current section reviews ten recent publications sorted according to the developed device type, and outlines the place of our application among them highlighting its strengths and weaknesses.

The simplest electronic aid device involves the use of distance sensors in the item the person uses already. A smart knee glove with an ultrasonic sensor, PIR sensor, and buzzer attached to the Arduino Uno was created by Ikram et al. [1]. Multiple machine learning classifiers were trained, used in a voting ensemble, and provided 98.34% of detection accuracy in a range up to 4 meters. The idea is also similar in smart canes but with some modifications. Muktha et al. [2] equipped a cane with ultrasonic sensors and developed an integrated navigation system using an online map service, providing feedback using voice and vibrations. Suhair et al. [3] designed a low-cost walking stick using an ESP8266 microcontroller, ultrasonic sensors, buzzer, and vibration motor to detect obstacles. Such devices are relatively cheap and easy to manufacture, and the use of buzzer or vibration motor allows receiving fast notifications. The drawback of such a method is that an ultrasonic sensor can detect objects but not determine their nature, while the person needs to carry and charge one more device. Only the cane from [2] has a routing function among three implementations mentioned above.

Wearable systems enhance the use of devices equipped with cameras and improved computational abilities. A smart jacket was designed by Bhatlawande et al. [4], which comprised five ultrasonic sensors, a web camera, and a GPS receiver. Machine learning techniques were used to detect obstacles and find paths. The device was tested on ten people who are blind and found to be helpful for 80% of them. Visual and auditory information can be integrated within a wearable system to facilitate navigation, target detection, and planning paths by using neural networks, according to Song et al. [5]. Loresco et al. [6] attached a camera to 3D-printed glasses and applied deep learning algorithms to localize and navigate the environment, detect and avoid obstacles using a Raspberry Pi and Google Coral Edge TPU. Testing took place in a controlled indoor environment using System Usability Scale. AI-SenseVision [7] runs trained models on a low-cost single-board processor to detect visual and sensory information related to objects and obstacles in order to provide real-time audio feedback. There are two points worth mentioning. First, as shown in [4] and [6], user testing of prototypes, even with small groups of people, is not only possible but also useful. Second, all mentioned systems use custom hardware components, while [5] and [6] are focused on indoor navigation, and the current system is aimed at outdoor walking environments.

Some of these systems conduct data processing on-device instead of doing it via a remote server, which is important to mention within the context of the current design. The models of Loresco et al. [6] are deployed on a Coral Edge TPU embedded in eyewear, whereas AI-SenseVision [7] works on a single-board processor of low cost, and in both cases, object recognition happens on-device without help from the server. Costs are yet another factor worth mentioning. Suhair et al. [3] mention that their device is affordable, and the authors of [7] say that their device is of low cost, which means that the question of costs is important for researchers in the same way as it is for users. This aspect is implemented in the smartphone concept in the most straightforward way since the camera, processor, GPS receiver, and speaker are located right in the pocket of a person. The downside of this approach is that the processor and battery of the device must work simultaneously on all those tasks, which entails additional costs.

Closely related to our approach is the implementation of detection and spoken output on regular hardware devices. For instance, Vision Enhancer developed by Jyotsna [8] consists of both object detection and depth calculation modules which are able to detect obstacles and the distance to them, as well as provide audio alerts and its accuracy is 98% according to the paper. Mathur et al. [9] introduced a deep learning-based visual assistance robot. Their system replaces conventional obstacle detection hardware with a software-centric model that employs CNNs and YOLOv5 for real-time object tracking and automated path optimization. In both cases, spoken output is considered the natural way of warnings, in accordance with the approach chosen for our application. However, in contrast to our approach, the distance calculation in both cases requires extra knowledge, either through additional model or camera and object knowledge.

The methods of interaction between these systems and users are different. In the case of the glove presented in [1] and the stick from [3], an audible buzzer is used, with the addition of the stick vibrating motor for indicating distance but nothing else. In case of the cane from [2], Vision Enhancer [8], and the bot from [9], the speech interface is used, thus providing a possibility of more complex messages about the nature of the objects and their lateral placement. In these cases, the speech is also the means of command routing. Therefore, the system which uses both interfaces should establish the order of the messages. This point is not addressed in the abstracts considered, and in this particular application, this issue is resolved by a single priority queue where the warning messages are spoken before the others and waiting messages receive increasing priority. The research carried out by El-Taher et al. [10] offers the clearest picture of user needs. The questionnaire survey was distributed among 49 visually impaired people in Ireland, and the findings were that existing journey navigation apps cannot provide enough information about traffic lights, intersections, and obstacles. 63% or more of those surveyed said they had experienced injury during journey navigation. This suggests that a route alone is not enough, and neither is a detector without a route.

There is significant diversity of approaches to the evaluation process used in the studies, making comparison impossible in numeric terms. Detection accuracies of 98.34% and 98%, according to Ikram et al. [1] and Jyotsna [8], correspond to distinct devices, activities, and conditions, so comparison is not possible. Bhatlawande et al. [4] and Loresco et al. [6] go beyond mere evaluation of performance by involving users in testing, conducting it with ten fully blind people and using a System Usability Scale, respectively. The contribution made by El-Taher et al. [10] represents another user-oriented perspective. Two implications follow from the above analysis. Figures related to accuracy become meaningful only under explicit definition of conditions under which tests were conducted and a system intended for use by visually impaired users should be tested from the user's point of view at the appropriate stage. Tests described in Section VI only evaluate speed and routing time on a single smartphone.

In conclusion, the existing literature leads to four design implications. First, speech is a viable form of output for warnings [2], [8], [9]. Second, on-device processing allows for instant feedback without any dependency on a server [6], [7]. Third, detection and navigation focus on the different sides of the same problem, which is evident from the survey in [10], and thus should be included into one application. Lastly, regardless of the actual technology used, findings based on users' experience have more importance than mere detection performance [4], [6]. All the above implications were considered while setting the design goals in Section III, except for one point, due to the initial platform being a smartphone, the design does not use sensors that other devices possess.

Putting these papers side by side, three gaps stand out. Most of the systems use extra hardware, such as a glove, a cane, a jacket, glasses or a small robot, which raises the cost and the effort of carrying it. In the papers we read, camera-based detection is usually judged on detection accuracy or in controlled settings, and it is not always combined with routing to a destination. And we did not find a safety net for when something goes wrong, such as a message to a trusted contact, described in any of them. Our application is an attempt to cover these gaps on one smartphone. It uses the phone's camera for on-device YOLOv8n detection, gets walking

routes from OpenStreetMap-based services, speaks both through one priority queue, and sends an SOS message and a low-battery alert to saved contacts. We also have to be honest about what it lacks compared with these works. Our distance estimate is rough, the detector knows only the COCO classes. The related work is thus considered as evidence for the inclusion of detection, navigation, and emergency functionalities in one program, as well as a reminder of the significant amount of testing and evaluation that is still needed.

III. Design Goals

These goals guided the design.

- **Voice first.** The main walking flow, from the welcome message to arriving at the destination, works through spoken prompts and voice commands.
- **Low cost, no extra hardware.** Only sensors present in an ordinary Android handset (minimum API level 26) may be assumed.
- **Little effort while walking.** The app keeps reading the GPS position and announces turns on its own, so the user does not have to refresh or tap anything.
- **On-device detection.** Camera frames are processed on the phone and never uploaded, so obstacle detection works without internet.
- **Guardian safety net.** The user must be able to send their position to chosen contacts with a simple gesture, and those contacts should be warned automatically if the phone is about to run out of power.
- **Easy to use.** Instead of managing many screens at a time, this app provides all features in one interface.
- **Modest safety scope.** The application can miss objects, GPS can be inaccurate and maps can be out of date, so it is an aid and not a replacement for a white cane or guide dog.

IV. System Architecture

The application is a single Android activity with three layers. The first layer contains ten screens developed using Jetpack Compose: splash, home, voice destination, navigation, obstacle detection, saved places, emergency, settings, manage contacts, and manage saved places. Awareness about which screen is displayed at a particular moment is tracked through a simple state variable along with the back stack, where returning to the Home screen leads to clearing the stack. The second layer contains several small modules responsible for performing a certain job. The third layer contains the dependencies of the application, such as Android services for speech, camera, location, and SMS, online map services, and model file in the application (refer to Fig. 1).

The speech output system is controlled by a dedicated audio manager that wraps around the text-to-speech engine and a priority queue. This helps in ensuring that all the screens use a common channel so as to avoid overlapping of route instructions and obstacle warnings. Speech input is done by the recogniser class that listens actively for a single phrase at a time. In terms of obstacle detection, CameraX uses the camera frames and passes them to the detector where the YOLOv8n algorithm is run locally. The GPS location is obtained using the location helper while the route service converts the place name from the speech input to a destination and corresponding route. Two modules deal with safety: the SOS helper sends SMS messages, and a background service watches the battery. Contacts, saved places and the battery threshold are stored in the application settings.

Two flows run side by side during a walk. In the first, the user speaks a destination. The route service finds it, requests a route, and the guidance loop then reads the GPS position every two seconds and sends turn instructions to the audio manager. In the second, the camera sends every third frame to the detector, which finds the closest object and sends a short warning to the same audio manager. Because both flows end in one queue, the audio manager decides what is heard first.

Three choices shape the structure. First, only one audio manager speaks, so that urgent warnings are heard before routine information. Second, detection runs on the phone while searching and routing use the internet, so obstacle warnings do not depend on a network connection. Third, the low-battery service runs separately from the screens, so it keeps working when the application is closed, and it restarts after a reboot. The project has 33 Kotlin files with 5,768 lines. Table 1 lists the main files and Table 2 lists the tools used.

Table 1 Principal Source Modules

Module	Responsibility
MainActivity	Screen state, splash, volume shortcut
LocationHelper	One-shot and continuous location, last known position
RouteService	Parsing, geocoding, OSRM routing, steps
Routegeometry	Haversine, projection, route trimming
NavigationScreen	Guidance loop, turn and arrival logic, live detection
MapPreview	Tiles, markers, polyline, follow mode
CameraManager	Camera lifecycle, frame throttling
ObjectDetectorHelper	Model load, preprocessing, inference, NMS, guidance
ObstacleDetectionScreen	Detection display and speech
AudioManager	Priority speech queue
SpeechRecognizerManager	Voice input
EmergencyScreen, Emergerncysoshelper	SOS gesture, SMS composition
Batteryalertservice	Low-battery alert, restart after boot
AppSettings	Contacts, places, threshold, last location

Table 2 Implementation Stack

Layer	Technology
Language, UI	Kotlin, Jetpack Compose, Material 3
Location	Google Play services fused location
Map	osmdroid 6.1.20, OpenStreetMap tiles
Routing	OSRM (foot profile), Nominatim, Photon, Overpass
Camera	CameraX 1.3.4
Detection	YOLOv8n (float I/O), yolov8n.tflite (12.7 MB)
Runtime	LiteRT 1.4.1 interpreter
Audio	SpeechRecognizer, TextToSpeech
Emergency	SmsManager, foreground service
Storage	Shared preferences
SDK	minSdk 26, targetSdk 37, Java 11

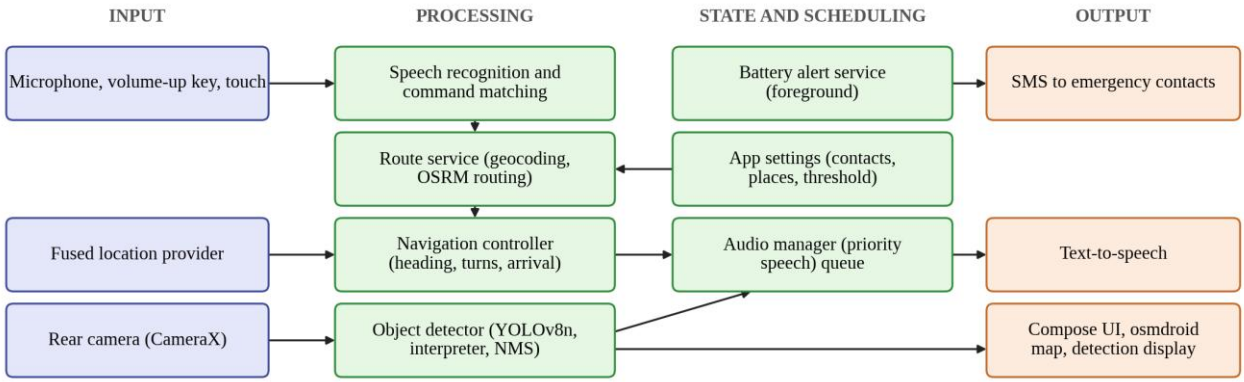


Fig. 1. Logical architecture of the system.

V. IMPLEMENTATION

A. Voice Interaction and Screens

After the welcome message, the home screen lists five spoken commands: Start Navigation, Detect Obstacle, Saved Places, Emergency and Settings. Buttons are also provided for users who prefer to tap. The microphone listens once permission is given and restarts after each result or error. The recognized text is matched with simple keyword rules; for example, a phrase containing "Start Navigation" opens navigation, and one containing "Settings" opens settings option. Key-word matching is used instead of a language model because the command list is small and the rules are easy to inspect. A separate function matches the words home, back, return, exit and main menu.

On the voice destination screen, the application first states that it is finding the current location. If this succeeds, it speaks the place name found by reverse geocoding; if it fails, it asks the user to check that GPS is on. It then asks where the user would like to go, listens, finds a route, shows a map preview and asks the user to say yes or no. Any other answer is followed by a repeated question. The confirmation step is included because speech recognition can make mistakes and a journey toward the wrong place is undesirable.

Pressing the volume-up key twice within 400 ms stops any speech and returns to the Home screen. This provides a recovery path for a user who cannot find a button. Fig. 2 shows how the screens connect.

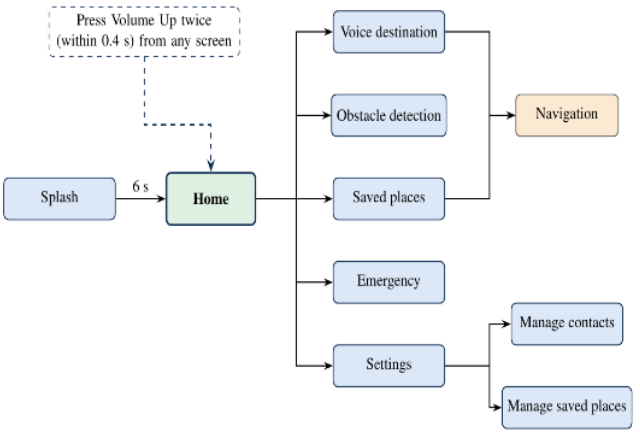


Fig. 2. Primary screen flow.

B. Destination and Route

Spoken place names are often unclear or incomplete, and one search service rarely finds them all. The route service tries several steps in order. It searches Nominatim near the user, then without a location bias, then with a region hint from reverse geocoding. It then tries Photon, then Nominatim again after removing words such as "restaurant" or "near me", and then an Overpass name search. If these fail, other commercial geocoders (Mappls, Mapbox, Geoapify, OpenCage, LocationIQ and Google Places) can be used (if

the API keys are set). As a last attempt, the service searches only the final two significant words, and if that succeeds the application states that it is guiding to the general area.

A separate function reads latitude–longitude pairs and Google Maps links, including short links that need redirects. It is called only from the Manage Saved Places screen, where a guardian types or pastes the text. The voice destination screen has no text box and sends the spoken words directly to the geocoder, so coordinates and links cannot be used as a spoken destination. Routes are requested from the public OSRM server with the walking profile. If that fails, the driving profile is tried, and if that fails too, a straight line is drawn and the route is marked as approximate. The time estimate from the server also assumes a normal walking pace, so it is recalculated from the distance at 0.9 m/s which is assumed in the project and not measured.

C. Walking Guidance

During the initial phase of navigation, the app will ask for a precise location. In case the above requirement is not satisfied, the app alerts the lack of this feature and stops itself from making a path with an undefined origin point. Once the walk begins, the app starts to request the location updates every three seconds, stores the last received position in the memory as the last known location, and asks the question about that location every two seconds in a loop. No compass is used. The heading is calculated based on two consecutive GPS locations and it is changed only when the user moves more than three meters in order to prevent the false heading in case the user is stationary. The distances are calculated using the haversine formula.

For the turning step, the system gives distance and direction information upon approaching the turn at 80, 40, and 15 meters. The repetition of the instruction three times is important because there is a chance that it may go unnoticed in a noisy street setting. The user's current orientation is compared to the orientation toward the next waypoint, and the following instructions are given: "turn around," "turn left or right," or "keep going straight." The step gets updated once the user is less than 15 meters away from its destination. Arrival is announced once the user has two successive readings less than 15 meters, thus avoiding the possibility of an error with one GPS location and finishing the journey too early. Figure 3 shows one such iteration of the loop.

The map is generated through osmdroid, showing the current location of the user, the destination, and the route line. In follow mode, the map will zoom in and rotate based on the orientation of the user, and the portion of the route that has been covered so far will be cleared out.

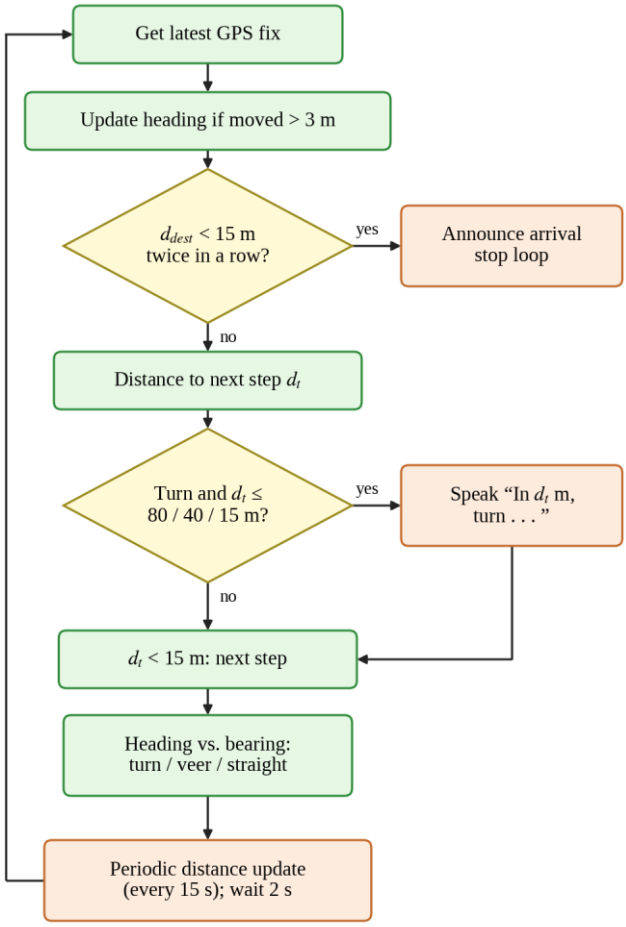


Fig. 3. One iteration of the guidance loop in the navigation screen.

D. Obstacle Detection

Object detection runs in two places. The Detect Obstacle screen shows the camera view and speaks about what is found. The Navigation screen also shows a live camera view with detection boxes and speaks the same warnings during the walk, using the same audio queue as the route instructions.

The model is a YOLOv8n file (yolov8n.tflite, about 12.7 MB) exported at 320 × 320 pixels with INT8 turned off; the application requires float32 input and output tensors. It is pre-trained on the 80 COCO classes and can therefore name objects such as people, bicycles, cars, buses, benches and chairs. A small general-purpose model was chosen because it is readily available and small enough for a phone. It is not trained for pedestrian hazards, as discussed in Section VII.

Fig. 4 shows the processing steps applied to each frame. The camera keeps just the latest frame, and the processing happens every third frame. It involves converting the frame to a bitmap, rescaling it to fit the model input through gray padding and normalizing. The interpreter works on four threads. Candidate detections with scores under 0.40 are discarded, and overlapping boxes of the same classes are removed by non-maximum suppression at 0.45 and up to five detections kept. There is one lock that ensures the screen does not close the model while the camera thread uses it; any errors make the detector unavailable. In this case, the application does not say anything because saying "Path clear" without having a detector is not appropriate.

The detection with the highest bounding box is assumed to be the closest object. The centre of the bounding box defines its horizontal position that can be left, centre, or right (the frame width is divided into thirds). The box height to the frame height ratio gives an approximate distance estimate in discrete steps (see Table 3). Distance estimation is done solely based on the size of the detected object and does not include any depth sensing;

An object estimated within three steps is spoken at a higher priority than one farther away. The message depends on the side and the distance. A new object label is spoken if at least 1.5 seconds have passed since the last obstacle message. The same label is repeated after 4 seconds, or after 2 seconds if it is within three steps. If nothing is detected, "Path clear" is spoken at most every 20 seconds. No filter ignores irrelevant classes, and detections are not smoothed over time

Table 3 Box-Height Ratio to Estimated Steps

Box height / frame height	Estimated steps
> 0.60	2
0.45–0.60	4
0.30–0.45	6
0.15–0.30	10
≤ 0.15	15

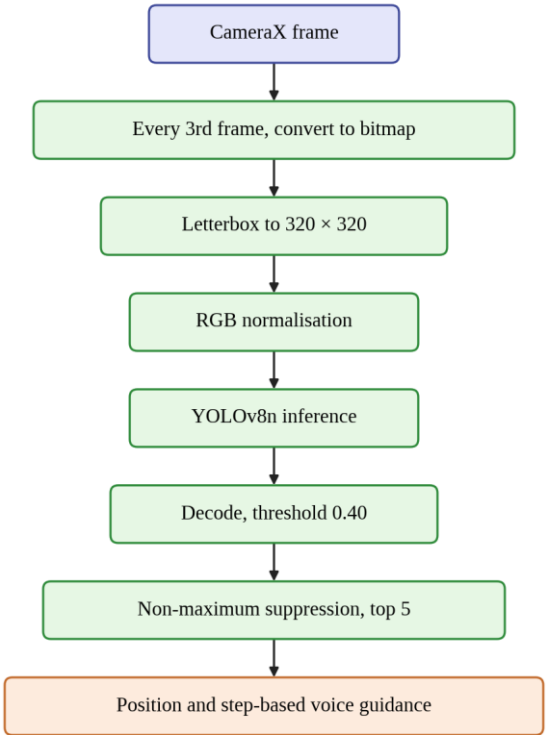


Fig. 4. On-device object-detection pipeline.

E. Speech Queue

All speech outputs are managed by one audio manager which uses six priority levels in a hierarchy: emergency, close obstacle, navigation command, navigation update, obstacle info, and general info. The emergency message empties the speech queue and interrupts current vocalization. Messages may override queued messages on the same priority level, and no more than one general info message may be in the queue at once. To avoid delay of low-priority messages behind the high-priority ones, any waiting message is raised by one priority level after a 2.5 second interval.

F. Saved Places and Settings

The guardian can save up to 20 places and 10 emergency contacts. To add a new place, the guardian gives it a name and either types or speaks the address or paste Google Map URL and confirms with the voice command "Confirm and Save". Inside the Saved Places screen, the user may speak the name of one of places; the application asks for a confirmation in yes/no format, and the navigation begins based on the saved coordinates without the need of additional search.

G. Emergency SOS

The emergency screen asks the user to press and hold anywhere for three seconds to send the alert. If a location is available, an SMS with a Google Maps link is sent to every contact.

H. Low-Battery Alert

A foreground service will be triggered from the home screen if the user permits SMS permissions. It will track battery level changes and compare the current reading with the threshold set on Settings (default setting 20%). In case the battery level is lower or equal to the threshold and there is no notification sent out in the current low period, then the service will obtain the latest location for a maximum period of 5 seconds and send the notification message to the selected contacts.

I. Fixed Values

Table 4 lists the fixed values found in the source code.

Table 4 Constants Found in the Source

Constant	Value
Model input size	320 × 320
Confidence threshold	0.40
NMS IoU threshold	0.45
Maximum detections returned	5
Letterbox padding value	114
Interpreter threads	4
Frames analysed	every 3rd
Minimum gap, new obstacle label	1.5 s
Repeat cooldown, same label	4 s (2 s if urgent)
Urgent distance	≤ 3 estimated steps
“Path clear” interval	20 s
Location request interval	3 s target, 2 s minimum
Guidance loop period	2 s
Heading update distance	3 m
Turn warning distances	80, 40, 15 m
Arrival radius	15 m, two readings
Distance update interval	about 15 s
Walking speed for time estimate	0.9 m/s
Map redirect limit (links)	5
Speech aging step	2.5 s
Volume double-press window	400 ms
SOS hold time	3 s
Default low-battery level	20%
Service location timeout	5 s
Splash duration	6 s
Maximum contacts / saved places	10 / 20

Table 5 Representative Spoken Messages

Situation	Message (abridged)
Splash	“Welcome to AI Navigation Assistant”
No location at start	“I could not detect your current location... . ”
Destination prompt	“Okay. Where would you like to go?”
Unclear confirmation	“Please say Yes or No.”
Saved place chosen	“Navigating to *name*.”
SOS hold complete	“Sending emergency alert now.”
SMS accepted	“Emergency message sent to your contact.”
SMS failed	“Could not send the message. Please try again.”
Centered, ≤ 3 steps	“Stop. *label* directly ahead. Move right, then continue.”
Nothing detected	“Path clear.”
Off heading	“Turn around”, “Turn left/right”, “Veer slightly”, “Continue straight”

VI. Results and Evaluation

A. Test Setup

We ran two small tests on one phone, a vivo V2355 with Android 16 (API 36) and 7.3 GB of RAM, using the debug build of the application. The first test times the object detector, and the second checks destination lookup and routing. Neither test involved users, and neither measure how accurately objects are detected. We therefore treat the results below as a pilot evaluation of speed and of the navigation pipeline, not as a study of accuracy or safety.

B. Detector Speed

The speed test measures the time the detector needs for one frame, from resizing the input to decoding the boxes. It does not include converting the camera image, drawing the boxes or speaking. After 10 warm-up runs, we timed 100 frames of a fixed test frame. The model loaded in 87.2 ms. Table 6 and Fig. 5 give the results of the first run. A frame took 105.8 ms on average and 97.8 ms at the median, and 95 of the 100 frames finished within 147.2 ms. The slowest frame took 386.0 ms, and a few slow frames like

this one are the reason the standard deviation is as high as 33.8 ms. We repeated the test, and the second run gave a similar median of about 97 ms, a 95th percentile of 102 ms and a slowest frame of 321.8 ms.

Table 6 Detector Time per Frame

Metric (ms)	Run 1	Run 2
Model load time	87.2	–
Mean per frame	105.8	–
Median per frame	97.8	≈ 97
Fastest frame	96.5	–
95th percentile	147.2	102
Slowest frame	386.0	321.8
Standard deviation	33.8	–
Maximum rate (from mean)	9.4 per s	–

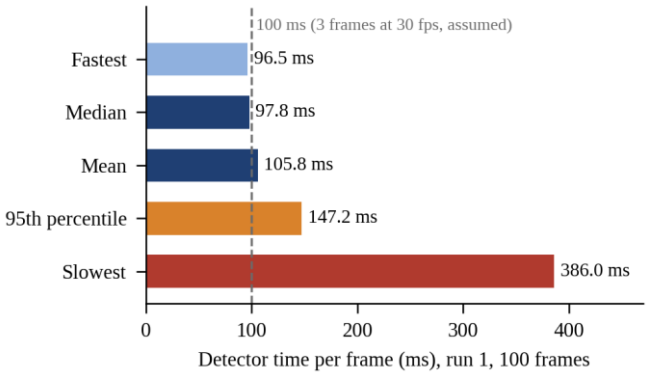


Fig. 5. Detector time per frame in run 1 (100 frames). The dashed line marks 100 ms, the interval between analysed frames for a 30 fps camera, which is an assumption about the device.

The mean time corresponds to at most 9.4 detections per second. At the walking speed of 0.9 m/s that the application assumes, the user moves only about 9.5 cm between two detections. The application analyses every third camera frame, so a camera running at 30 frames per second (again an assumption) would offer a new frame about every 100 ms, which is close to the median time per frame. On this phone the detector therefore keeps pace with the frame stride, but without much margin. The slow frames may come from other work running on the phone, but we did not investigate this. The real application also converts images, draws the boxes and speaks, so its actual rate will be somewhat lower than these figures.

C. Destination Lookup and Routing

The second test entered five real destinations in Bengaluru and one meaningless text, and recorded the time for geocoding and for the walking route (Table 7, Fig. 6). The geocoder returned coordinates for all five real destinations, after 1.73 s on average and 4.51 s at most. A walking route followed after 1.77 s on average and 6.00 s at most, so a route was ready about 3.5 s after the destination text on average. All five routes came from the walking profile, and none fell back to the driving profile or to a straight line. The meaningless text was rejected, but only after 6.9 s.

Table 7 Destination Lookup and Routing Test

Destination	Geocode (ms)	Route (ms)	Straight (km)	Route (km)	Steps
Kengunte circle	1167	792	9.85	11.9	19
Shankar Nag bus stop, Kamala Nagar	4513	1014	3.12	3.9	18
Dr. Ambedkar Inst. of Technology	294	224	9.57	11.9	21
Majestic bus stand	898	6003	2.54	3.7	12
Lalbagh botanical garden	1775	809	18.69	24.8	17
Meaningless text	6931	–	not found	–	–

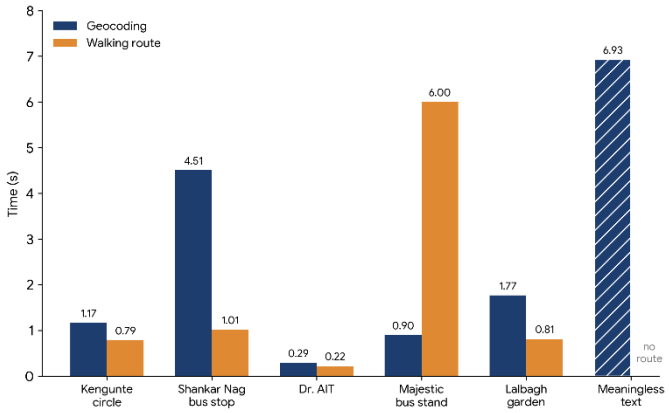


Fig. 6. Time for geocoding and for the walking route, per destination. The meaningless text was rejected after 6.93 s and had no route.

The results fell between 2.5 km and 18.7 km from our starting point. The automated geocoding naturally handles varied real-world landmarks, some destinations like Lalbagh garden showed a wider radius, yet our routing engine handled every scenario seamlessly. Across the board, the walking routes proved highly reliable, consistently coming out between 1.2 and 1.45 times the length of the straight-line distance (Fig. 7).

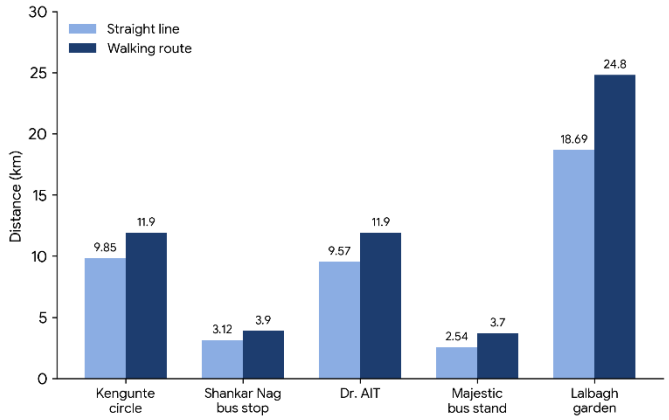


Fig. 7. Straight-line distance and walking-route length for each destination on a linear scale, using the correct local coordinates.

D. Comparison with Existing Apps

Table 8 compares the application with five widely used apps for blind and low-vision users, based on the vendors' public descriptions and third-party accessibility reviews [11]–[15]. Each app is strong in its own area: Google Maps in walking directions, Lookout and Seeing AI in recognizing objects and text, Be My Eyes in human and AI visual assistance, and BlindSquare in describing places and intersections. In the descriptions we found, none of them combines walking directions, camera-based obstacle warnings and an emergency message in one program, which is the combination our application targets. The table compares functions and not quality. "Not described" means that the source does not mention the feature.

Table 8 Feature Comparison with Existing Apps

App	Platform	Walking directions	Camera recognition	Emergency alert
Google Maps [11]	Android, iOS	Yes; detailed voice guidance for walking	Not described	Not described
Lookout [12]	Android	Not described	Objects, people, text; Find mode	Not described
Seeing AI [13]	iOS, Android	Not described	Text, products, people, scenes	Not described
Be My Eyes [14]	iOS, Android	Not described	AI photo description; volunteer video	Not described
BlindSquare [15]	iOS	Yes; distance and direction to a destination	Not described	Not described
This app	Android	Yes; OSRM walking route, turn announcements	YOLOv8n on device; side and steps	SOS and low-battery SMS

E. Discussion

Taken together, the two tests give a mixed but useful picture. On the positive side, the typical detection takes about 98 ms, and the destination pipeline returned a walking route for all five real destinations within a few seconds, without needing the driving profile or the straight-line fallback. On the other side, both tests show how thin the margins are. The detector has only about 2 ms to spare at the median, and routing delay varied from under a second to almost seven seconds with no clear pattern. Speed alone also says little about usefulness. A detector that is fast but names the wrong object, or a route that is found quickly but leads to the wrong place, would not help the user.

F. Analysis of Results

Although the tests were small, they can be read against the design goals in Section III. Two things are supported. First, on the test phone the detector usually finishes processing one frame (about 98 ms) before the next frame is due (about every 100 ms), so it keeps up with the camera. Second, the destination pipeline found and routed all five real destinations, and the meaningless input was rejected instead of being guessed.

The two halves of the system are limited by different things. Detection is limited by the phone's processing and repeats on every analysed frame, so any delay is felt continuously during a walk. Routing depends on public online services and happens once, before the walk begins. The mean time to a ready route (3.5 s) is more than thirty times the median detection time (about 98 ms), but in our view the two are not comparable in impact. A few seconds of waiting before a trip, with the user told what is happening, is easier to accept than a delay that repeats while the user is moving. This also explains why the design keeps detection on the device and leaves only search and routing to the network.

Some parts of the design were not exercised by this run. All five routes came from the walking profile, so the driving and straight-line fallbacks were never needed. We also did not record which search step found each destination, so we cannot yet say how often the fallback searches help, and logging this in future runs would be an easy way to find out. Likewise, the detector and the routing were timed separately, whereas a real walk runs them together with the camera, the guidance loop and speech, so a combined run is needed to see how the phone copes with the full load. It is also worth being clear about what kind of result this is. The accuracies of 98.34% in [1] and 98% in [8] describe how often objects were detected correctly, while our figures describe how fast the detector runs, so the two cannot be compared directly. Speed is a sensible first check, because a detector that cannot keep up with the camera would be of little use on a walk, however accurate it is. Accuracy on labelled pavement images, and testing with visually impaired users as done in [4] and [6], are the natural next steps and are part of our future work. Table 9 summarises what the tests support and what remains open.

Table 9 Summary of Evidence from the Pilot Tests

Aspect	What was done	Outcome	Status
Detector speed	100 frames, two runs, one phone	Median about 97–98 ms; 95th percentile 102–147 ms	Supported on one phone
Lookup and routing	5 real destinations	All found; 0.5–6.9 s to a ready route	Supported, small sample
Invalid input	1 meaningless text	Rejected after 6.9 s	Supported, one case
Not measured	–	Step-distance accuracy, turn-announcement timing, usability with visually impaired users	Open (future work)

G. Scope of the Tests

These tests do not measure detection accuracy, the accuracy of the step estimate, the timing of the turn announcements, battery use, or usability with visually impaired users. They also used one phone, one network and a small number of destinations, so the numbers are indicative only. The missing tests are part of our future work.

VII. Limitations and Future Work

The application is a prototype, and it has several limitations that can be clearly improved. Currently, the object detector detects only the 80 objects present in the COCO dataset, and each detected object is notified regardless of its relevancy in assisting pedestrians navigate through the area. Fixing this problem would involve fine-tuning the model using a dataset consisting of pavement hazards and eliminating unnecessary classes. Calculation of distance in steps is done using the bounding box height alone, without calibrated thresholds, and the highest bounding box is considered to be the closest object. A depth estimate is planned to improve this.

The accuracy of GPS reduces in areas close to tall structures; thus, the natural step here would be to combine the compass data from the device with GPS. Geocoding and routing require the use of servers, which might be slow, limit the rate of requests, or simply give erroneous results; there is no offline navigation available. Offline maps with a hosted routing solution would resolve this issue. With regards to speech recognition, incorrect recognition and ambiguous phrases might result in unintended commands, as command recognition is based on keywords and only English is supported now.

In case there is no location at all, the manual SOS message is not sent out. In the same manner, the battery alert can have the wrong last known position in it. A fallback solution for sending the SOS message with the last known position will be incorporated. Contacts, saved positions, and the last known position are saved without any encryption, implying that encryption is another task for the future.

The destination test showed that ambiguous place names are a weak point, and a single yes-or-no confirmation is not enough when the wrong match is plausible. A stronger step is planned: the application will compare the distance of each candidate from the user, and when the best match is far away or when two candidates lie in different areas, it will ask the user to choose between them (for example "the one near you or the one in another city") before a route is requested.

VIII. Conclusion

This paper presents a smartphone application that integrates walking navigation, object detection, and SOS capability in one device for the benefit of visually impaired users. The application allows for voice command entry to determine the destination and offers turn-by-turn navigation with audible guidance. Object detection in real time is achieved using the YOLOv8n model running on the device, which gives notifications about nearby objects and distance estimation. Furthermore, the application is capable of sending SOS and low battery alerts, including the user's position to selected contacts. Speech processing is handled using a single priority queue ensuring that urgent messages get priority. The application is still under experimental phase where the detector works with knowledge of only the COCO classes, distance estimation is rough (in steps), and navigation uses public web services. Future

plans include creating pavement obstacles detection, depth estimation, off-line maps capability, and extensive testing with the target audience.

REFERENCES

[1] S. Ikram, I. S. Bajwa, A. Ikram, I. de la Torre Díez, C. E. Uc Ríos, and A. K. Castilla, "Obstacle detection and warning system for visually impaired using IoT sensors," IEEE Access, vol. 13, pp. 35309–35321, 2025, doi: 10.1109/ACCESS.2025.3543299.

[2] D. S. Muktha, G. Niveditha, N. A. Pinto, and S. Sinha, "Enhancing mobility: A smart cane with integrated navigation system and voice-assisted guidance for the visually impaired," in Proc. 13th IEEE Int. Conf. Commun. Syst. Netw. Technol. (CSNT), 2024, pp. 1124–1129, doi: 10.1109/CSNT60213.2024.10546111.

[3] K. Muhammed Suhair, S. Malayil, A. V. Deepak, and T. P. Faslu Rahman, "Affordable navigation aid: A smart walking stick for the visually impaired," in Proc. IEEE Recent Adv. Intell. Comput. Syst. (RAICS), 2024, doi: 10.1109/RAICS61201.2024.10689812.

[4] S. Bhatlawande, R. Borse, A. Solanke, and S. Shilaskar, "A smart clothing approach for augmenting mobility of visually impaired people," IEEE Access, vol. 12, pp. 24659–24671, 2024, doi: 10.1109/ACCESS.2024.3364915.

[5] Y. Song, Z. Lian, G. Liu, B. Wang, M. Zhu, and P. Shi, "Multi-sensory visual-auditory fusion of wearable navigation assistance for people with impaired vision," IEEE Trans. Autom. Sci. Eng., 2023, doi: 10.1109/TASE.2023.3340335.

[6] P. J. Loresco, R. J. C. Cruz, K. Z. Ramones, J. A. D. Zafra, and K. R. G. Ramirez, "Indoor navigation glasses for the visually impaired with deep learning and audio guidance using Google Coral Edge TPU," in Proc. TENCON 2024 – IEEE Region 10 Conf., 2024, pp. 842–845, doi: 10.1109/TENCON61640.2024.10902929.

[7] R. C. Joshi, M. K. Dutta, N. Singh, A. K. Sharma, and R. Burget, "AI-SenseVision: A low-cost artificial-intelligence-based robust and real-time assistance for visually impaired people," IEEE Trans. Human-Mach. Syst., 2024, doi: 10.1109/THMS.2024.3375655.

[8] Jyotsna, "Vision Enhancer: A voice-driven system for obstacle detection," in Proc. Int. Conf. Innovation and Novelty in Engineering and Technology (INNOVA), 2024, doi: 10.1109/INNOVA63080.2024.10847032.

[9] T. Mathur, N. Jain, U. Jaiswal, R. Kothiyal, P. Sharma, and R. Bathla, "Advancements in assistive technology: A blind aid bot for enhanced spatial awareness and navigation," in Proc. Int. Conf. Advances in Computation, Communication and Information Technology (ICAICCIT), 2024.

[10] F. E.-Z. El-Taher, L. Miralles-Pechuán, J. Courtney, K. Millar, C. Smith, and S. McKeever, "A survey on outdoor navigation applications for people with visual impairments," IEEE Access, vol. 11, pp. 14647–14666, 2023, doi: 10.1109/ACCESS.2023.3244073.

[11] Google, "Voice guidance in Maps, built for people with impaired vision," Google Blog, Oct. 2019. [Online]. Available: https://blog.google/products/maps/better-maps-for-people-with-vision-impairments (accessed Oct. 13, 2026).

[12] Google, "Use Lookout to explore your surroundings," Google Accessibility Help. [Online]. Available: https://support.google.com/accessibility/android/answer/9031274 (accessed Oct. 13, 2026).

[13] Perkins School for the Blind, "Microsoft Seeing AI and low vision review," Perkins.org. [Online]. Available: https://www.perkins.org/resource/microsoft-seeing-ai-and-low-vision-review/ (accessed Oct. 13, 2026).

[14] Be My Eyes, "Getting started with Be My Eyes," Be My Eyes Support. [Online]. Available: https://support.bemyeyes.com/hc/en-us/articles/360005528557-Getting-started-with-Be-My-Eyes (accessed Oct. 13, 2026).

[15] BlindSquare, "What is BlindSquare?" BlindSquare. [Online]. Available: https://www.blindsquare.com (accessed Oct. 13, 2026).