InnovAIte

AI ASSISTED NAVIGATION DEVICE

By: Yash Kalra

Research Report

High-Level Problem / Problem Description

JUSTIFICATION

Navigating unfamiliar environments is a significant challenge for visually impaired individuals. Traditional aids like canes or guide dogs are limited. According to the WHO, over 285 million people suffer from vision impairment globally. There is a growing demand for intelligent, affordable, and reliable indoor navigation systems. With recent developments in AI and IoT, we are well-positioned to design a wearable system that integrates real-time object detection, environmental awareness, and voice feedback.

PROBLEM STATEMENT

Visually impaired individuals lack access to real-time, intelligent assistive systems that can detect environmental objects, read signage, and provide navigational guidance. Most existing solutions either rely on GPS (which fails indoors), are costprohibitive, or provide only partial functionality (e.g., vibration-only alerts).

VISION

This project aims to build a smart, affordable, and real-time AI-powered assistive navigation device that:

- Detects indoor and mobility-relevant objects using a lightweight YOLO model
- Provides voice feedback using offline Text-to-Speech (TTS) libraries
- Integrates sensor data (ultrasonic + IMU) for obstacle and motion detection
- Supports text reading via OCR on signs and printed material
- Is deployable on a Raspberry Pi with Arduino integration

-

DIFFERENTIATION FROM EXISTING SOLUTIONS

- Runs offline (pyttsx3 + espeak-ng) unlike cloud-reliant solutions

  Combines vision-based object recognition with dynamic distance sensing and fall detection

- Integrates document reading and signage detection
- Uses lightweight deep learning models suitable for edge deployment
- Leverages SLAM and binaural cues for localization

Solution Overview

SYSTEM ARCHITECTURE Architecture Summary:

The system is designed to function as an intelligent, wearable or handheld assistive navigation device. It uses a multi-sensor and multi-model architecture. A camera module (ESP32-CAM or Pi Camera) captures live video input, which is processed by a Raspberry Pi using the YOLOv4-tiny model to detect objects in the environment. OCR (Tesseract) is used on visual frames to read printed or signboard text.
Simultaneously, Arduino-connected sensors (ultrasonic HC-SR04 and MPU6050 IMU) collect spatial awareness and orientation data. These inputs are fused and prioritized on the Pi using custom logic that determines the most critical information. The result is converted into audio using an offline Text-to-Speech engine (pyttsx3 or Coqui) and played back via earphones or speaker. All components are battery powered and integrated into a compact, mobile solution.

Components:

- **Camera Module** (ESP32-CAM/Pi Camera): Captures live frames
- **Raspberry Pi 4**: Object detection, OCR, TTS logic
- **Arduino UNO**: Sensor interfacing for MPU6050 & ultrasonic sensors
- **Text-to-Speech Module**: Offline TTS (pyttsx3 / Coqui)
- **Speakers/Headphones**: User interface
- **Battery Bank**: Power supply

MODEL & DATA FLOW

- Image capture → YOLOv4-tiny model → Object label + bounding box
- OCR scan on frames using OpenCV + Tesseract
- Sensor input (distance/orientation) via serial
- Prioritized fusion: object + proximity + text → voice output

DATASET (COCO + CUSTOM) COCO

2017 Dataset:

- 80 object categories, includes: chair, couch, table, laptop, person, tv, plant, microwave
- Does not include door, exit signs, handrails, or steps

Solution:

- Use LabelImg to label additional custom objects
- Augment dataset with custom photos from libraries/classrooms
- Train with YOLOv4-tiny using transfer learning in Google Colab

DYNAMIC SENSOR INTEGRATION

Handling Dynamic Data from Sensors:

Dynamic data from sensors is managed using a real-time serial communication pipeline between the Arduino UNO and Raspberry Pi. The Arduino continuously reads values from the ultrasonic sensors (distance) and the MPU6050 IMU (acceleration, gyroscope), and sends them over serial (UART) to the Raspberry Pi every 100 milliseconds. On the Raspberry Pi, a Python script listens to the serial port and parses the incoming data.

A buffer system with timestamps ensures synchronization between the video frame being processed and the corresponding sensor data. Data smoothing is applied using a moving average filter to eliminate noise from raw sensor values. Threshold-based logic is used to trigger specific events — such as playing an audio warning when an obstacle is detected within 1 meter, or when tilt exceeds 30 degrees.

Fusion logic then decides which event to prioritize: obstacle proximity, detected object, or fall alert. Only the most critical message is forwarded to the Text-toSpeech engine to avoid overloading the user with redundant information.

-

From research:

Ultrasonic HC-SR04 calculates object proximity, ideal for stair detection, narrow space alerts

- MPU6050 detects tilt/fall; ideal for triggering rest alerts
- Combined with object detection to prioritize dynamic warnings (e.g., "Person ahead, 2 meters")

HARDWARE–SOFTWARE INTEGRATION

The integration between hardware and software in this project is managed through a modular pipeline that ensures real-time data processing with minimal latency. Arduino UNO handles all low-level sensor interfacing (e.g., ultrasonic HC-SR04, MPU6050) and transmits data via serial communication to the Raspberry Pi. This raw data includes distance measurements and orientation status every 100ms.

On the Raspberry Pi, a Python-based software stack processes visual input (from ESP32-CAM or Pi Camera) using the YOLOv4-tiny model and Tesseract OCR. Simultaneously, it listens to the Arduino's sensor data, processes it with filters, and merges it with the object detection and text output. Logic prioritizes which data needs to be converted into voice prompts or alerts.

This integration enables the device to function as a responsive, real-time system capable of making decisions based on fused multimodal data. The use of serial communication (UART), timestamp buffers, and Python control scripts ensures that both visual and physical sensor data are processed and acted upon synchronously.

TESTING STRATEGY

To evaluate the effectiveness and reliability of the AI-Assisted Navigation Device, we will conduct a structured testing phase in a simulated real-world environment. This testing will take place in a designated building at Deakin University.

Environment Setup:

- A classroom or lab space will be transformed to simulate common indoor navigation challenges.
- Objects like chairs, desks, computers, bookshelves, and signage will be strategically arranged to represent a realistic indoor layout (similar to a library or classroom).

Participants:

-

Team members will participate by simulating pedestrian motion and interacting with the device.

- All participants will provide informed consent before any form of recording is conducted.

Testing Objectives:

- Evaluate object detection accuracy (YOLOv8) in varying lighting and obstacle positions.
- Validate ultrasonic + IMU sensor readings for proximity and tilt.
- Ensure that voice guidance is context-aware and non-repetitive.
- Confirm OCR readability of printed signs and paper.

Output:

- Observational logs will be taken.
- Team feedback will be recorded to refine model thresholds and message logic.

This plan ensures comprehensive testing of the prototype in conditions that closely reflect real usage scenarios.

INDOOR NAVIGATION LOGIC

To support directional movement (e.g., navigating from a bookshelf to a chair), the AI-Assisted Navigation Device can be extended with basic indoor navigation logic that leverages object detection and relative positioning.

Navigation Strategy:

- **Object Anchoring**: The YOLOv8 model detects static features like shelves, desks, and chairs. Each object is assigned a reference position in a temporary local map based on detection order and angle.
- **IMU-Aided Position Tracking**: The MPU6050 sensor provides acceleration and orientation, allowing for basic step counting and direction estimation (dead reckoning).

-

Voice Query Support: Users can issue simple spoken commands like "Go to the chair" or "Guide me to the exit." The system identifies the object class and selects the most recently detected instance.

- **Turn-by-Turn Feedback**: Based on real-time updates, the system guides the user using proximity and orientation cues. Example voice prompts include "Turn slightly left... move forward two steps... you are near the chair."

Future Expansion:

- Integration of SLAM (Simultaneous Localization and Mapping) or Wi-Fi/BLE beacons could provide more precise room-level navigation in complex environments.

This approach ensures that even without a pre-mapped environment, users can be guided intelligently between key objects using AI perception and basic spatial awareness.

PRIVACY AND ETHICAL CONSIDERATIONS

To ensure the ethical integrity of the project and the protection of all participants, the following privacy and ethical considerations are embedded into the project design:

Consent and Transparency:

- All team members or volunteers participating in prototype testing will be informed in advance.
- Written consent will be obtained from each participant before any data is collected, especially in cases involving recording or analysis.

Data Protection:

- No personally identifiable information (PII) will be stored or shared.
- All data captured (e.g., video frames, sensor logs) will be anonymized and stored locally on secured Raspberry Pi storage.
- The system does not connect to the internet during use, ensuring offline processing for enhanced privacy.

-

Purpose Limitation:

Collected data is used solely for performance evaluation of the navigation system.

- Any media or logs collected during testing will be deleted after the analysis is complete.

Accessibility and Inclusion:

- The project has been developed with the input of peers to ensure inclusive design for visually impaired users.
- TTS prompts and guidance systems are tested for clarity, timing, and comfort.

By adhering to these principles, the project aims to ensure responsible development and deployment of AI technologies in assistive domains.

RESOURCES

The following tools, platforms, and open-source projects have been used or referenced in the development and planning of this navigation device. Each resource is directly relevant to the technical implementation or benchmarking of system features.

Resource Link	Description
Ultralytics YOLO App	Mobile app that allows testing, running, and deploying YOLO models quickly. Useful for prototyping object detection on smartphones.
CameraX + OpenCV Android Demo	Android app demo combining CameraX API and OpenCV. Helps visualize live edge detection and frame processing.

OpenCV FR Demo	Demonstrates face recognition and OpenCV vision capabilities on Android. Useful for vision feature exploration.

Microsoft Seeing AI	AI-based narration app by Microsoft. Serves as a benchmark for OCR, object reading, and real-time guidance UX.
Coqui TTS	Open-source TTS framework offering high-quality, neural speech synthesis. Can be integrated for customizable audio output.
ESPnet	Toolkit for end-to-end speech processing. Useful for future extensions involving speech recognition or interaction.
pyttsx3 Documentation	Official documentation for the offline Python TTS library used in this project. Fully compatible with Raspberry Pi.

Each team member is encouraged to review these platforms to understand how similar systems function and where enhancements can be applied in our solution.

USER INTERFACE (UI) INTEGRATION

To further enhance accessibility and usability for visually impaired users and testers, the system includes a graphical User Interface (UI) hosted on the Raspberry Pi or accessible via a local web interface.

Features of the UI:

- **Live Object Detection Feed**: Displays bounding boxes and labels for detected objects in real-time.
- **Sensor Status Dashboard**: Visual indicators for proximity distance and orientation (MPU6050 values).
- **Feedback Logs**: Shows recent TTS output and interpreted commands.

- **Configuration Panel**: Enables users or testers to adjust thresholds for obstacle alerts, toggle OCR or voice prompts, and enable haptic feedback.

Technical Implementation:

- The UI can be developed using Tkinter for a native interface or Flask + HTML/CSS for a browser-based dashboard.
- Sensor and detection data will be passed through local buffers or MQTT to the UI in real time.
- For voice input and control, libraries like Whisper or Vosk can be integrated to accept simple spoken commands.

Future Enhancements:

- Add buttons or tiles for commonly requested actions (e.g., "Guide to exit", "Read sign")
- Use user profiles to store preferences (audio/haptic balance, verbosity level)

This UI not only supports daily users but also assists developers in testing, debugging, and demonstrating the system in a clear and transparent manner.

Implementation Plan HARDWARE SUMMARY

Component	Role
Pi Camera	Visual input (live video feed)
Raspberry Pi 4	Central AI & ML processor
Arduino	Sensor control
HC-SR04	Distance measurement

MPU6050	Motion/orientation detection
Headphones	Audio output

SOFTWARE STACK

Tool/Library	Purpose
Python	Control logic & ML
OpenCV	Video processing
YOLOv4-tiny/Darknet	Object detection
Tesseract OCR	Text recognition (signage)
pyttsx3 / Coqui TTS	Offline voice generation
Arduino IDE (C++)	Microcontroller programming

ALGORITHM SUMMARY

The project incorporates a suite of machine learning models and rule-based algorithms tailored to real-time embedded deployment. Below is a breakdown of each algorithm, its role, and implementation.

Task	Algorithm	Description
Object Detection	YOLOv4-tiny / YOLOv8	YOLO (You Only Look Once) is a fast, onestage object detection algorithm. We start with YOLOv4-tiny for lightweight edge deployment and plan to migrate to YOLOv8 for improved accuracy and training flexibility.

Text Reading (OCR)	Tesseract OCR	Open-source OCR engine that extracts text from video frames. Used to identify signage, labels, and printed material.

Speech Generation	Concatenative TTS (pyttsx3/Coqui)	Converts detected object names or OCR output into spoken words using offline TTS models. Pyttsx3 supports pitch/speed configuration. Coqui provides higherquality neural synthesis.
Proximity Alerts	Rule-based fusion	Simple if-else logic based on ultrasonic sensor values. For example: if distance < 100cm, trigger TTS warning. Integrated with object detection priority.
Orientation Alerts	MPU6050 threshold logic	Gyroscope data is filtered and thresholded to detect tilts or possible falls. Alerts are generated if roll/pitch exceed ±30°.

YOLO Model Details:

- **YOLOv4-tiny**: Ideal for real-time detection on Raspberry Pi with minimal compute resources.
- **YOLOv8 (Ultralytics)**: Used for training with custom datasets, supports flexible export formats (PyTorch, ONNX, CoreML) and advanced augmentation. **OCR Pipeline:**

1. Capture frame
2. Apply preprocessing (e.g., grayscale, thresholding)
3. Feed into Tesseract engine
4. Return extracted text for TTS output

Sensor Fusion Workflow:

- Incoming data from ultrasonic and IMU sensors is buffered with timestamps.
- Fusion logic compares distance, object type, and tilt in parallel.
- The most urgent event is selected for speech output.

These models and algorithms form the backbone of the AI-assisted navigation system, ensuring timely, relevant, and accurate assistance for visually impaired users.

PROGRAMMING LANGUAGES

Language	Purpose
Python	YOLO, OCR, TTS, Fusion Logic
C++	Arduino-based sensors
Shell	Raspberry Pi setup scripts

DEPLOYMENT PLAN

Model Training Workflow (YOLOv8):

1. **Data Collection**: Use the COCO dataset along with custom indoor object photos (e.g., door, shelf, sign).
2. **Annotation**: Annotate images using Roboflow or LabelImg in YOLO format.
3. **Environment Setup**: Set up a training environment using Google Colab or local GPU with the Ultralytics YOLOv8 package (pip install ultralytics).
4. **Configuration**: Prepare your dataset YAML file, specifying class labels and paths.
5. **Model Training**: Use the yolo task=detect mode=train model=yolov8n.pt data = dataset.yaml epochs = 50 imgsz = 640 command to train.
6. **Evaluation**: Track training progress with loss, mAP, and confusion matrices.
7. **Export**: Export the best .pt weights file using yolo export model=best.pt format=onnx or keep in PyTorch format.
8. **Deployment**: Transfer the trained model to the Raspberry Pi and run inference using Python with OpenCV and Ultralytics API.

Deployment:

- Google Colab (YOLOv4-tiny)
- COCO + Custom annotated dataset

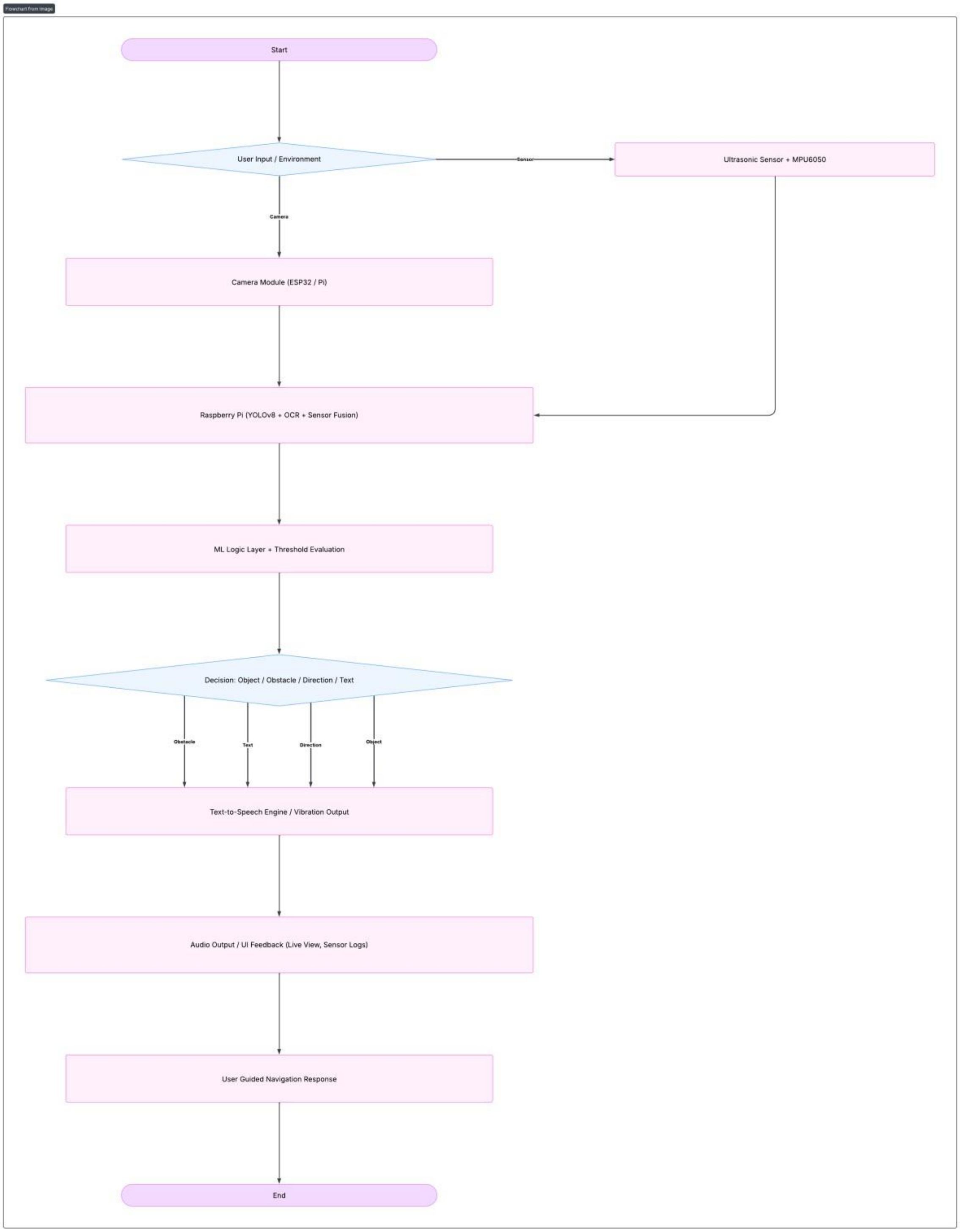
Deployment:

- YOLOv8 model weights (.pt) uploaded to Pi
- detect.py runs object recognition
- Audio feedback triggered via pyttsx3 or Coqui
- OCR runs per interval for signage detection
- Sensor data parsed every 100ms from Arduino

Visualization & Diagrams

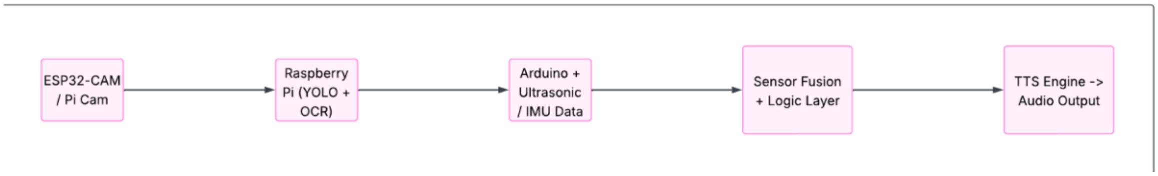
FULL SYSTEM FLOWCHART (AI + UI + SENSORS + OUTPUT)

The following flowchart illustrates the complete pipeline of how hardware, machine learning models, UI components, and dynamic sensor data integrate to deliver realtime navigation assistance for visually impaired users:


Flowchart from Image
Start
User Input / Environment
Sensor
Ultrasonic Sensor + MPU6050
Camera
Camera Module (ESP32 / Pi)
Raspberry Pi (YOLOv8 + OCR + Sensor Fusion)
ML Logic Layer + Threshold Evaluation
Decision: Object / Obstacle / Direction / Text
Obstacle
Text
Direction
Object
Text-to-Speech Engine / Vibration Output
Audio Output / UI Feedback (Live View, Sensor Logs)
User Guided Navigation Response
End

This flowchart reflects the live feedback loop and coordination between visual data, physical sensors, processing logic, and multi-modal output to assist the user safely and efficiently.

Flowchart: Device Interaction


ESP32-CAM / Pi Cam
Raspberry Pi (YOLO + OCR)
Arduino + Ultrasonic / IMU Data
Sensor Fusion + Logic Layer
TTS Engine -> Audio Output

Research Insights & References

GOOGLE DRIVE ARCHIVE

All research papers, references, and technical resources used to prepare this report have also been stored securely and shared in a publicly accessible Google Drive folder:

Google Drive Link: AI Navigation Device – Research Collection

This folder includes:

- Full-text PDFs of all research papers
- Screenshots and image diagrams used in the report
- Dataset planning notes
- Annotated articles for TTS, OCR, SLAM, YOLOv8, and indoor navigation

Team members and stakeholders are encouraged to explore the folder for further learning, validation, and citation purposes.

Research papers incorporated:

The following research papers were studied and referred to during the development of this project. Their insights helped in areas like object detection, OCR integration, sensor fusion, TTS logic, UI design, and testing approaches:

- Object detection design was influenced by *Blind Assistive System Based on Real-Time Object Recognition*, *Efficient Multi Object Detection and Smart Navigation*, and *VOICE-ASSISTED-REAL-TIME-OBJECT-DETECTIONUSING-YOLO-V4-TINY*.
- TTS integration was inspired by *AI-Based Assistance for Visually Impaired People Using TTS* and benchmarked against *Microsoft Seeing AI*.
- OCR functionality was guided by *An Assistive Reading System for Visually Impaired using OCR and TTS*, *Finger-Eye: A Wearable Text Reading Assistive System*, and *Smart Application for OCR and Text Detection*.

- Navigation logic and fall detection were informed by *A Deep Learning Based Model to Assist Blind People in Their Navigation* and *Mahendran et al., CVPRW 2021*.
- Sensor fusion design (ultrasonic + IMU) and proximity logic were modeled after *Obstacle Avoidance for Blind People* and *IRJET-V11I7142*.
- UI considerations and hardware-software coordination were shaped by concepts in *Microsoft Seeing AI* and *Intelligent Technologies and Applications*.

These works provided critical understanding and direction in shaping both the technical implementation and usability goals of our assistive navigation system.