Visual Obstacle Detection System for Blind Navigation

Abstract

In this report, a real-time visual obstacle detection and navigation system designed for people with visual impairments will be presented. The detection system uses computer vision, deep learning object detection, and edge server computing to give users immediate awareness of their surroundings. A Raspberry Pi captures visual data and sends it to a server running a YOLOv8 object detection model. The detection system checks obstacles for how close and in which direction they are, then notifies the user with audio and LED feedback. By focusing on low latency, accuracy, and portability, this affordable solution works well both indoors and outdoors.

1. Introduction

Independent and safe navigation remains one of the major obstacles faced by individuals with visual impairments. There are some well-known traditional mobility aids, such as white canes and guide dogs. However, these aids provide limited information about obstacles beyond immediate physical contact. There have been some recent advances in computer vision and embedded systems which enabled the development of intelligent assistive devices. Those new advances extended surrounding awareness beyond tactile range.

This project proposes a real-time obstacle detection and navigation aid that utilizes camera based perception and deep learning to identify obstacles, estimate their proximity and convey actionable feedback to the user. In addition, The system is designed to be lightweight, affordable and adaptable, with a strong emphasis on minimizing false alarms while maintaining timely warnings.

2. Project Objectives

2.1 Primary Objectives

The primary goals of this project are:

-Enhancing mobility independence by enabling visually impaired users to navigate in unfamiliar environments in a more safe and confident way.

-Provide real time obstacle detection to increase situational awareness.

-Deliver intelligent alerts that differentiate between minor obstacles and immediate hazards.

-Support multimodal feedback, combining audio and tactile cues to improve accessibility and usability.

2.2 Design Principles

The system was developed according to several guiding principles:

-Low latency: Minimize the delay between obstacle detection and user notification.

-Accuracy over raw speed: Prioritize reliable detection and warning stability.

-Environmental adaptability: Operate effectively in diverse indoor and outdoor settings.

-Resource efficiency: Support battery powered, portable hardware suitable for wearable deployment.

3. System Architecture

3.1 Architectural Overview

The system has a client-server architecture. The architecture separates image gathering from computationally intensive inference tasks. The Raspberry Pi client captures video frames, then transmits them wirelessly to a locally run server, which performs object detection and navigation logic before returning feedback commands.

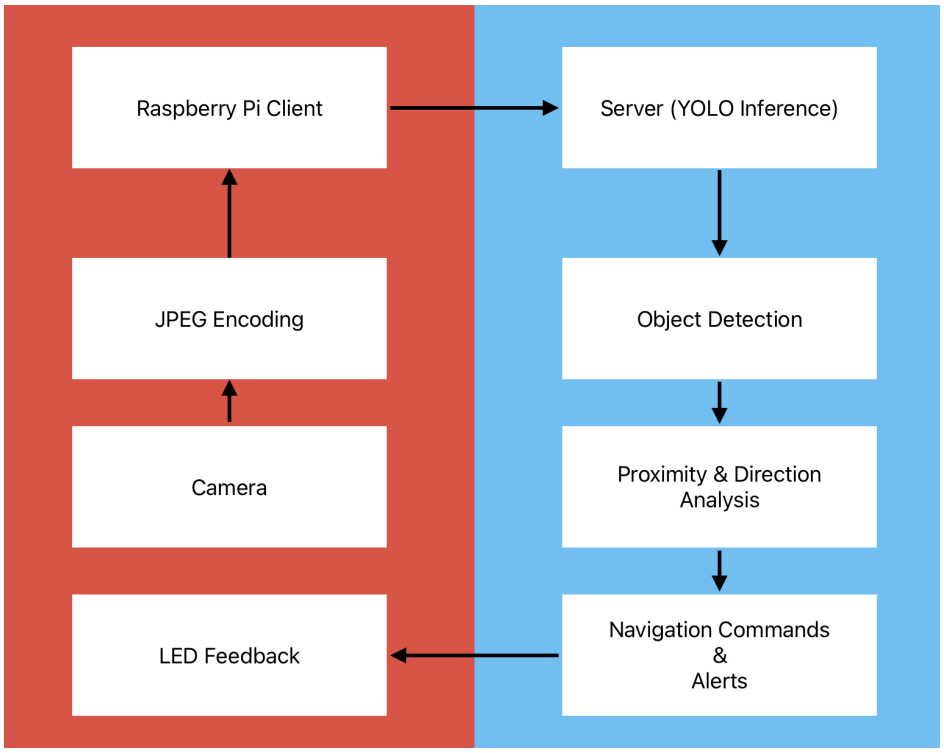

Figure 1: System Architecture

This separation allows the system to remain lightweight on the client side while benefiting from scalable server-side computation.

4. Hardware and Software Components

4.1 Hardware Components

Raspberry Pi: Serves as the client device, capturing video frames at a resolution of 320×240 using an IMX219 camera module.

Server Machine: Executes the object detection pipeline. This is a locally run environment not a Docker container because of the speak module.

Network: A local WiFi connection enables communication between the client and server using a static IP configuration.

4.2 Software Stack

Client (Raspberry Pi):

-Python 3

-NumPy and PIL for image handling

-Requests library for HTTP communication

-Linux sysfs interface for LED control

Server:

-Flask for RESTful API services

-YOLOv8 for object detection

-OpenCV for image decoding and visualization

-Windows PowerShell-based text-to-speech for audio feedback

5. System Pipeline and Workflow

5.1 Frame Capture and Preprocessing

The Raspberry Pi captures RGB frames continuously and it uses the camera utility. Frames are temporarily stored as raw binary files and afterwards converted into JPEG format with 70% compression quality to reduce bandwidth usage. To make sure there is real-time responsiveness, only the two most recent frames are retained.

5.2 Frame Transmission

The client identifies the most recent unprocessed frame and transmits it to the server via an HTTP POST request. A persistent HTTP session is maintained to reduce connection overhead and improve throughput.

5.3 Server-Side Inference

After receiving a frame, the server decodes the image and performs object detection using YOLOv8. Detected objects are filtered based on configurable criteria, including object class, confidence level and minimum bounding box area. There is a custom proximity metric. In addition, that metric is computed for each detection, and the most critical obstacle is selected for navigation decisions.

5.4 Feedback Generation

The server returns a structured JSON response containing navigation commands, object class information, proximity scores and latency metrics. Audio alerts are generated using text-to-speech, while LED blink patterns provide tactile feedback corresponding to obstacle direction and urgency.

6. Object Detection Model

6.1 Model Selection

The YOLOv8 nano (YOLOv8n) model was selected. Because, it has a favorable balance between inference speed, detection accuracy and model size. Its lightweight architecture makes it suitable for near real-time applications in assistive technologies.

6.2 Training Dataset

The model utilizes pre-trained weights obtained from training on the COCO dataset, which contains 80 common object categories relevant to indoor and outdoor navigation scenarios. No additional fine tuning was required, since the pre-trained model demonstrated sufficient performance for obstacle detection tasks.

6.3 Detection Modes

Two operational modes are supported:

-Any-object mode: Detects all COCO classes.

-Person-only mode: Restricts detection to human obstacles for crowded environments.

7. Proximity Estimation

7.1 Near Score Metric

A custom proximity metric, which is referred as the nearScore, estimates obstacle distance without relying on depth sensors because we had no depth sensor in our Raspberry Pi:

nearScore = 0.7 * areaRatio + 0.3 * bottomRatio

This formulation uses ideas of perspective geometry, where closer objects appear larger and lower in the image frame.

7.2 Distance Categories

Near scores are mapped to discrete distance categories as it can be seen in Figure 2 (far, medium, close, very close) to simplify decision-making and feedback generation.

```
def getDistanceCategory(nearScore):
    #converts our near_score to interpretable distance levels
    if nearScore is None or nearScore < NEAR_OUT_SCORE:
        return "far"
    elif nearScore < 0.35:
        return "medium"
    elif nearScore < 0.50:
        return "close"
    else:
        return "very close"
```

Figure 2: Distance categories

7.3 Stability Mechanisms

Hysteresis thresholds meaning a "sticky" state to prevent rapid flickering between "obstacle detected" and "clear" which are found using NEAR_IN_SCORE as 0.20 (object must reach score >= 0.20 to enter the "near" state) and NEAR_OUT_SCORE as 0.17 (Object must drop below 0.17 to exit the "near" state) and a three-frame moving average is applied to prevent rapid change between warning states and to reduce jitter caused by bounding box fluctuations. The reason of the 0.03 gap in the detection is because if the threshold was exactly 0.19 then if the score was:

-0.188 -> 0.191 -> 0.189 -> 0.192

- clear -> obstacle -> clear -> obstacle (rapid flickering)

With our current setup with the stability mechanisms applied:

-0.21 -> 0.19 -> 0.18 -> 0.16

-obstacle -> still near -> still near -> now clear

8. Direction Classification

Obstacle direction is determined by the normalized horizontal position of the bounding box center. The system divides the camera's field of view into three horizontal zones using left and right thresholds (LEFT_THRESH = 0.33, RIGHT_THRESH = 0.67) enabling directional warnings that correspond to the user's forward field of movement. Objects that has center positioned in the left third of the frame (normalized x-position < 0.33) are classified as "obstacle_left," those in the right third (x-position > 0.67) as "obstacle_right," and those in the central third (0.33 =< x =< 0.67) as "obstacle_front." as can be seen from Figure 3. Critically,

only obstacles classified as "obstacle_front" trigger voice announcements, ensuring users receive warnings specially for obstacles directly in their path while avoiding confusion from objects positioned to the sides.

```
# we determine direction based on horizontal position based on 0.33 and 0.67 thresholds
if bestNormCx < LEFT_THRESH:
    rawCmd = "obstacle_left"
elif bestNormCx > RIGHT_THRESH:
    rawCmd = "obstacle_right"
else:
    rawCmd = "obstacle_front"
```

Figure 3: Horizontal categories of the frame

9. Feedback Systems

9.1 Audio Feedback

Text-to-speech announcements are generated only for front-facing obstacles to reduce cognitive load. A cooldown period and state-change detection mechanisms prevent repetitive or overlapping audio alerts.

9.2 Haptic Feedback

The Raspberry Pi's onboard LED provides tactile feedback through distinct blink patterns. The number of blinks encodes obstacle direction, while blink speed reflects urgency based on proximity.

10. Running and Streaming the Frames

To be able to see the live stream buffer one can go to http://127.0.0.1:8000/stream and also before running the Pi client code one needs to make sure to change the IP to the server machine's IP.

11. Performance Optimization

System performance was optimized through reduced image resolution, JPEG compression, persistent network sessions and early filtering of insignificant detections. End-to-end latency measurements indicate a response time of approximately 200-500 ms, which is acceptable for walking-speed navigation.

12. Design Rationale

Key design decisions include adopting a client–server architecture to overcome edge-device limitations, using a heuristic proximity metric instead of expensive depth sensors and prioritizing front-facing audio alerts to improve usability. These choices were made by hardware constraints, cost considerations and user experience in mind.

13. Limitations and Future Work

13.1 Current Limitations

-Lack of true depth perception

-Sensitivity to lighting conditions

-Latency constraints in fast-moving environments

13.2 Future Enhancements

Proposed improvements include on-device inference using edge accelerators, integration of depth sensors or IMUs, extensive user studies and power optimization for wearable deployment.

14. Conclusion

This project demonstrates a practical and cost effective assistive navigation system for visually impaired users. By combining deep learning based object detection with a novel proximity heuristic and multimodal feedback, the system delivers meaningful environmental awareness in real time. The proposed architecture provides a strong foundation for future development of wearable assistive technologies and highlights the potential of computer vision in improving accessibility and quality of life.

Appendix:

Figure 4: Our moving “cane” project setup

Figure 5: Camera is taped to hold still