

International Journal Research Publication Analysis

Page: 01-10

SIGHTSENSE: AN AI-POWERED REAL TIME VISUAL INTERPRETATION SYSTEM

¹Dr. Abhishek Kulkarni, ²Shivathej S. Gowda, ²Sreepriya B. M., ²Sudhiksha Prabhakar,
²Sunaina Ravi

¹Associate Professor, Department of Computer Science and Engineering, Jyothy Institute of Technology, Bengaluru, India.

²Department of Computer Science and Engineering, Jyothy Institute of Technology, Bengaluru, India.

Article Received: 17 September 2026

Article Revised: 07 October 2026

Published on: 27 October 2026

*Corresponding Author: Dr. Abhishek Kulkarni

Associate Professor, Department of Computer Science and Engineering, Jyothy Institute of Technology, Bengaluru, India.

DOI: <https://doi.org/10.1555/ijarp.1860>

ABSTRACT

Visual impairment is a significant global challenge affecting millions of individuals, severely limiting their ability to navigate, recognize objects, and interact independently with their environment. Traditional assistive tools such as white canes and guide dogs, while reliable, fail to deliver detailed real-time environmental information. With recent advancements in Artificial Intelligence (AI) and computer vision, intelligent assistive systems have emerged as transformative solutions. This paper presents the implementation of SightSense, an AI-based visual interpretation system built on a Raspberry Pi

5 with a camera module. On request, the system captures an image, sends it to a multimodal Gemini model for scene description or text reading, translates the result into one of ten supported languages, and delivers it as audio feedback using neural text-to-speech; a separate on-device YOLO-based mode provides object detection. The software is organized into independent Python modules so that each stage can be improved or replaced without changing the others. Functional testing showed that the prototype describes scenes, reads printed text and speaks in the selected language, and it exposed two limitations: reduced confidence in low light and a freeze when the cloud API usage limit is reached. Challenges widely reported in this field, including processing latency, reduced accuracy in cluttered environments, limited personalization and limited context awareness, remain relevant, and the paper outlines future work on offline fall-back, voice-based control and user studies. The

goal is to contribute to efficient, affordable and accessible AI-driven assistive technology that enhances the safety, confidence and independence of visually impaired individuals.

KEYWORDS: Assistive Technology, Ai-Based Visual Interpretation, Multilingual Speech Synthesis, Raspberry Pi, Scene Description, Text Recognition, Visually Impaired Users.

I. INTRODUCTION

People with visual impairment regularly need information that sighted people take for granted: what an object is, what a label says, or what lies ahead in an unfamiliar room. Research on visual question answering shows how common these needs are; the VizWiz challenge, for instance, is built from questions that blind people asked about photographs they took themselves [1]. White canes and guide dogs remain valuable, but they do not describe the surroundings, and many electronic aids depend on proprietary hardware, paid services or a remote human assistant.

Recent multimodal models make a different design possible. A single model can caption an image, classify its contents and answer questions about it without task-specific training [2]. Combined with a single-board computer, a camera and a speech engine, this allows a compact assistant to be assembled from widely available parts. SightSense is a prototype of this kind, developed in the Department of Computer Science and Engineering at Jyothy Institute of Technology, Bengaluru.

This paper concentrates on how the prototype is implemented. Its contributions are:

- 1) A modular python implementation on a raspberry pi 5 that couples still-image capture, cloud-based scene description and text reading, and spoken output;
- 2) Spoken output in ten languages (english and nine indian languages) through a translation step and language-specific neural voices;
- 3) A record of functional test outcomes, including two failure modes observed in practice; and
- 4) An explicit statement of limitations and the steps needed to move from prototype to evaluated assistive device.

Section II reviews related work, Section

III states the requirements and design approach, Section IV presents the architecture, Section V describes the implementation, Section VI reports testing, and Sections VII and VIII discuss limitations, future work and conclusions.

RELATED WORK

A. Detection-based assistive systems

Object detectors trade speed against accuracy. YOLO predicts boxes and classes in a single pass of one network [3], and SSD likewise detects objects in one pass across several feature scales [4]. The two-stage Faster R-CNN [5] is generally more accurate but computationally heavier, which is why single-stage detectors are favored on embedded hardware. Kumari and Hammady

[6] build smart glasses that combine YOLOv8 detection, distance estimation and cloud OCR, and report a detection precision of 0.90 and recall of 0.83. Systems of this kind return bounded labels and distances but do not describe a scene in sentences.

B. Wearable and language-model-based systems

AIris [7] is a wearable consisting of eyewear with a camera, earphones and a Raspberry Pi; it uses a distributed architecture in which local information is captured and transmitted for processing, and its modules include face detection, text reading, object recognition and money counting. Baig et al. [8] describe a hat-mounted camera connected to a Raspberry Pi 4 Model B that combines real-time object recognition with contextual understanding using large vision-language models. Navigation-focused systems such as NavCog [9] provide route guidance rather than describing what a camera sees.

C. Reading text in natural scenes

Text in everyday photographs is hard to detect and recognize. The COCO-Text benchmark was created to study this problem [10], and open-source engines such as Tesseract are widely used for printed text [11]. Multimodal language models offer an alternative route, because the request that describes a scene can also ask for the visible text [2].

D. Positioning of this work

SightSense is closest in spirit to [7] and [8]: it also pairs a Raspberry Pi with a camera and audio output and relies on modern vision-language capability. Its distinguishing implementation choices are a Requirement

Low cost, portable

Modularity

Tolerance of errors

How it is addressed

Raspberry Pi 5, camera module, power bank

One Python module per stage

Original text spoken if translation fails

menu-driven design that exposes

IV. SYSTEM ARCHITECTURE

description, text reading and detection as separate modes, and a translation and neural-voice stage covering ten languages, nine of them Indian. The contribution is an integrated, documented implementation rather than a new algorithm.

II. REQUIREMENTS AND DESIGN APPROACH

The design was driven by a short list of requirements, shown with the way each is addressed in Table I. Three principles guided the work: keep hardware inexpensive and widely available, keep every stage in its own module so that it can be replaced, and keep spoken output short and in the user's language.

TABLE I. Requirements and how they are addressed

The prototype is a command-line application organized in layers. The input layer is the camera. The processing layer is either a Gemini model reached over the network (scene description and text reading) or a local YOLO detector. The language layer translates text when a language other than English is selected. The output layer synthesizes speech and plays it. Fig. 1 shows the pipeline shared by the description and text-reading modes, and Table II lists the modules.

Requirement

Capture on demand

Describe the scene

Read printed text

Detect objects

Spoken multilingual output

How it is addressed

Picamera2 still capture to a JPEG file

Gemini model through the Google GenAI client

Same client with a constrained prompt

Separate on-device YOLO mode

Translator module and neural voices for ten languages

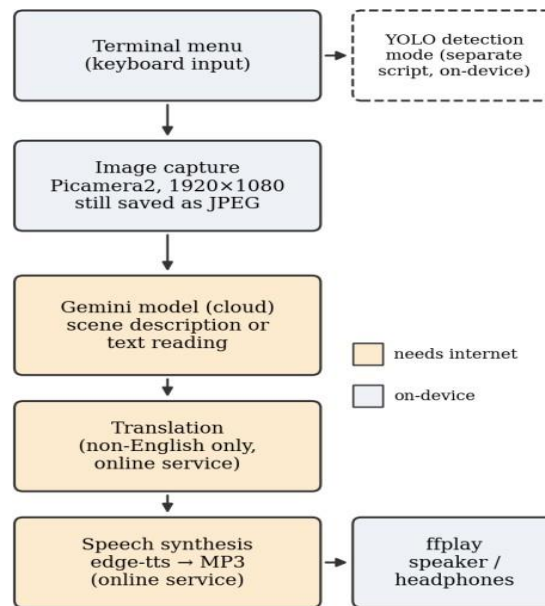


Fig. 1. SightSense processing pipeline.

Shaded blocks need an internet connection.

TABLE II. Software modules.

Module	Responsibility and main libraries
Main menu	Keyboard-driven loop that launches the selected mode; speaks a farewell on exit
Capture	Picamera2 still capture to a JPEG file
Scene description	Google GenAI client, Pillow, python-dotenv; loops until stopped
Text reading	Same client; constrained prompt; speaks the result
Translator	deep-translator (online); returns the original text on error
Speech	edge-tts for synthesis, ffplay for playback

stops the camera. A confirmation message is printed once the file is saved.

C. Scene Description

The scene description module reads the API key from an environment variable loaded with python-dotenv, so the key is not stored in the source code. It opens the saved image with Pillow and sends it, with a text prompt, to a Gemini model through the Google GenAI client. The returned text is translated if needed and spoken. In one test run the system reported a large light-pink cylindrical object nearby, a label on the right showing the text “System Ma,” “Processor” and “Brand,” and a light tiled floor. After speaking, the module captures the next

image and repeats until the user interrupts it with Ctrl+C (Algorithm 1).

Algorithm 1. Scene-description loop

```
1 load API key from environment
2 create Gemini client
3 repeat
4   capture image (1920x1080, 2 s settle)
5   text <- describe(image, prompt)
6   if language is not English then
7     text <- translate(text, language)
8   (original text kept on error)
9   speak(text, language)
10 until user interrupts (Ctrl+C)
```

V. IMPLEMENTATION

A. Hardware and Environment

The prototype runs on a Raspberry Pi 5 with a Raspberry Pi camera module, powered from a supply or a power bank, with a speaker or headphones for output. Software is written in Python. During development the board was operated from a laptop over a VNC session. An internet connection is required for scene description, text reading, translation and speech synthesis.

B. Image Acquisition

Images are captured with Picamera2, the libcamera-based Python library for Raspberry Pi cameras [12]. The capture module creates a still configuration with a 1920×1080 main stream, starts the camera, waits two seconds so that exposure and white balance can settle, writes the frame to a JPEG file in a local images folder, and

D. Text Reading

The text reading module uses the same client with a more restrictive prompt. The prompt asks the model to read only text that is visible, to keep the original reading order as far as possible, and to return a fixed sentence stating that no readable text was found when none is present. Constraining the request in this way is intended to discourage the model from inventing text. The trimmed response is printed and spoken.

E. Object Detection Mode

The third menu option launches a separate script that performs YOLO-based object detection [3] on the device. It gives class labels without the cloud round trip needed for description, which makes it a candidate fall-back when the network or API quota is unavailable.

F. Multilingual Output

Supported languages are English, Hindi, Kannada, Tamil, Telugu, Malayalam, Marathi, Bengali, Gujarati and Punjabi. The model's answer is produced in English and, for any other language, passed to the translator module. It uses the deep-translator package [14] with automatic source-language detection and a language-code table, and a one-second pause precedes each request. If translation fails, the error is printed and the original English text is returned, so the user still hears a response. The selected language is currently a constant in the source code.

G. Speech Output

The speech module maps each language to a neural voice with an Indian locale and synthesizes the text with the edge-tts package, which accesses Microsoft Edge's online text-to-speech service without an API key [13]. The audio is saved as an MP3 file and played through ffplay in a subprocess with the display disabled, automatic exit and quiet logging.

H. User Interaction

The menu is read with a standard keyboard prompt (Fig. 2). It offers four choices: describe surroundings, read text, detect objects and exit. An invalid entry prints a message and returns to the menu, and choosing exit speaks a farewell. This interface suits development and demonstration, but it is not yet usable without sighted help, a point revisited in Section VII.

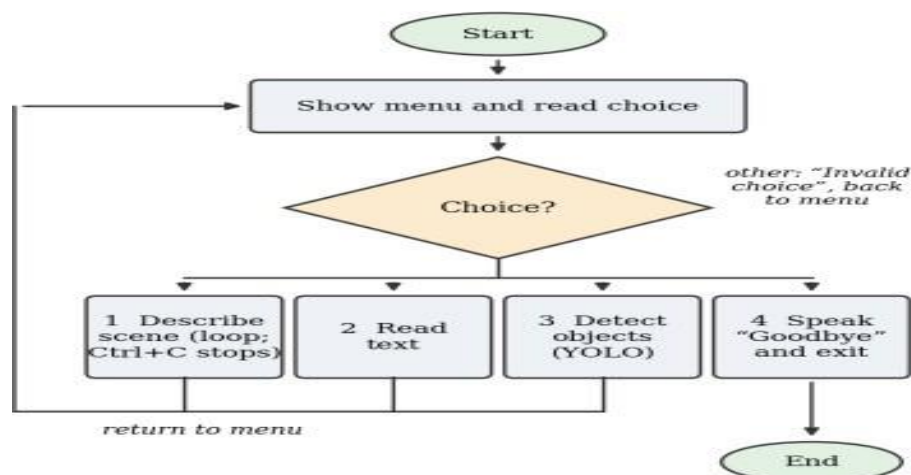


Fig. 2. Menu-driven control flow of the application.

VI. TESTING AND RESULTS

A. Test Setup

Tests were carried out by the development team on the Raspberry Pi 5 prototype in an indoor laboratory environment. Each test case had an expected outcome defined beforehand and was marked pass or fail against the observed behavior. Table III lists the outcomes.

TABLE III. Functional test outcomes.

Test case	Observed outcome	Status
Detection of common objects in the camera view	Class label, bounding box and score returned	Pass
Frame captured in low light	Objects detected with fairly low confidence; better training and prompting needed	Pass
Same object in consecutive frames	Output summarized identical objects as a count (e.g., “2 same objects”)	Pass
Text converted to speech	Clear speech in the selected language	Pass

Test case	Observed outcome	Status
OCR on printed text	Text extracted and spoken	Pass
Continuous operation	No crash, but the system froze once the API usage limit was reached	Pass

B. Observations

Scene descriptions were sentence-level and spatially phrased (for example, which side of the user an object was on), and printed labels in the scene were picked up in the same response. Low-light frames still produced output but with lower confidence. The most significant operational finding was the freeze at the API usage limit: the prototype depends on a quota-limited cloud service and currently has no graceful behavior when that quota runs out.

End-to-end latency was not instrumented in this study, so no timing figures are reported. The code does contain fixed delays: two seconds for camera settling in each capture, plus one second before each translation request. Network round trips to the model, translation and speech services add to this, so the system is best described as near-real-time rather than real-time.

C. Assessment Against Requirements

Against Table I, the capture, description, text reading, detection, multilingual output and modularity requirements were met in functional testing. Tolerance of errors is only partly met: a failed translation falls back to English text, but an exhausted API quota still freezes the system. Cost and portability were not quantified, because no bill of materials or battery-life

measurement is included in this study.

VII. LIMITATIONS AND FUTURE WORK

Several limitations should be stated plainly. First, the system relies on internet access and on third-party services; images are sent to a cloud model, and text is sent to translation and speech services, which raises privacy considerations for a user's surroundings. Second, generated descriptions can be incomplete or wrong, and the prototype provides no distance measurement or certified obstacle warning, so it must not be used as a substitute for a cane or other mobility aid. Third, the menu is keyboard-based and the language is fixed in code, so a visually impaired user cannot yet operate or configure the device independently. Fourth, this paper includes no user study with visually impaired participants and no quantitative accuracy or latency benchmark.

Planned work addresses these points: physical buttons or voice commands for input, request time-outs with an automatic fall-back to on-device detection and offline speech, a language selector, measured benchmarks of end-to-end latency, OCR accuracy and detection quality, and an evaluation with visually impaired users under appropriate ethical review. Distance sensing, richer context awareness and a wearable form factor are further extensions.

VIII. CONCLUSION

SightSense shows that a Raspberry Pi 5, a camera and a small set of Python modules can deliver spoken scene descriptions and text reading in ten languages. The modular design keeps each stage replaceable, and testing identified the dependence on a cloud quota as the main practical weakness. Quantitative benchmarking, offline fall-backs and a user study are the next steps before the system can be judged as an assistive device.

ACKNOWLEDGMENT

The authors thank their project guide, Dr. Abhishek Kulkarni, and the Department of Computer Science and Engineering, Jyothy Institute of Technology, for their guidance and support.

REFERENCES

1. D. Gurari et al., "VizWiz Grand Challenge: Answering visual questions from blind people," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2018, pp. 3608–3617.
2. Google, "Image understanding," Gemini API documentation. [Online]. Available:

- <https://ai.google.dev/gemini-api/docs/generate-content/image-understanding> (accessed Oct. 22, 2026).
3. Redmon, S. Divvala, R. Girshick, and Farhadi, “You only look once: Unified, real-time object detection,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788.
 4. W. Liu et al., “SSD: Single shot multibox detector,” in Computer Vision – ECCV 2016, 2016, pp. 21–37.
 5. S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 28, 2015.
 6. P. Kumari and R. Hammady, “Assisting blind people with AI and audio using smart glasses: System design with YOLOv8 variants comparisons,” *Multimedia Systems*, vol. 32, no. 1, Art. no. 73, 2026, doi: 10.1007/s00530-025-02139-z.
 7. D. D. Brilli, E. Georgaras, S. Tsilivaki, N. Melanitis, and K. Nikita, “AIris: An AI-powered wearable assistive device for the visually impaired,” arXiv:2405.07606, 2024.
 8. M. S. A. Baig et al., “AI-based wearable vision assistance system for the visually impaired: Integrating real-time object recognition and contextual understanding using large vision-language models,” arXiv:2412.20059, 2024.
 9. D. Ahmetovic, C. Gleason, C. Ruan, K. Kitani, H. Takagi, and C. Asakawa, “NavCog: A navigational cognitive assistant for the blind,” in Proc. 18th Int. Conf. Human-Computer Interaction with Mobile Devices and Services (MobileHCI), 2016, pp. 90–99.
 10. A. Veit, T. Matera, L. Neumann, J. Matas, and S. Belongie, “COCO-Text: Dataset and benchmark for text detection and recognition in natural images,” arXiv:1601.07140, 2016.
 11. R. Smith, “An overview of the Tesseract OCR engine,” in Proc. 9th Int. Conf. Document Analysis and Recognition (ICDAR), 2007, pp. 629–633.
 12. Raspberry Pi Ltd., “The Picamera2 Library,” Release 3, 2026. [Online]. Available: <https://github.com/raspberrypi/picamera> 2
 13. Rany2, “edge-tts: Use Microsoft Edge’s online text-to-speech service from Python,” GitHub. [Online]. Available: <https://github.com/rany2/edge-tts> (accessed Oct. 22, 2026).
 14. Nidhaloff, “deep-translator: Translate between different languages in Python,” GitHub. [Online]. Available: <https://github.com/nidhaloff/deep-translator> (accessed Oct. 22, 2026).