



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: <https://doi.org/10.22214/ijraset.2026.85013>

www.ijraset.com

Call: ☎ 08813907089

E-mail ID: ijraset@gmail.com

BlindAssist: A Near-Real-Time Object Detection and Voice-Based Assistive System for Visually Impaired Users

Nishchay Sakure¹, Chanchal Pagare², Md. Eliyas Sheikh³, Sapna Debnath⁴, Saurabh Sharnagat⁵, Harsha Keche⁶, Mr. Vaibhav Shrivastava⁷

¹⁻⁷Department of Computer Science and Engineering, G H Rasoni University, Amravati, Maharashtra, India

Abstract: *BlindAssist is a Python prototype that uses a laptop webcam and a YOLOv8 object detector to tell a visually impaired user, by voice, which objects are in front of the camera and roughly where they are. Frames are captured with OpenCV and passed to a COCO-pretrained YOLOv8 model (Ultralytics), and the results are spoken offline with pyttsx3. Each announcement gives the object's class name and whether it lies to the left, ahead, or to the right, based on the horizontal position of its bounding box. For seven object classes an uncalibrated estimate of distance is computed from the box width and shown on screen; it is not spoken. Objects are tracked by class name, so that a class is announced when it first appears, when its direction changes, and once when it has been missing for three consecutive detection passes. Face modules using YuNet and SFace (OpenCV) and the DeepFace library recognize known people from reference photographs, and a DeepFace emotion model announces the dominant facial expression of each detected face; a custom pen-and-pencil model is under development. The paper describes the design, gives the algorithms and configuration, compares the system with published assistive systems, and describes the tests we plan to run. No labeled test set or user study has been completed, so no accuracy, speed, or user-study figures are reported.*

Keywords: *Assistive technology, visually impaired, object detection, YOLO, OpenCV, face recognition, facial emotion recognition, DeepFace, text-to-speech, announcement logic, Python.*

I. INTRODUCTION

Many everyday tasks depend on sight: finding a phone on a desk, knowing whether something is to the left or right, knowing who is standing nearby, or noticing a sharp object on a table. The World Health Organization reports that at least 2.2 billion people worldwide have some form of vision impairment [1], and a systematic review of global blindness and vision impairment is given in [2]. This project covers one small part of that problem: telling a person by voice what a camera can see.

Object detectors such as YOLO [3], [4] can find and name many everyday objects in a live video stream. In a basic detection script the result is drawn as labeled boxes on the video, which does not help someone who cannot see the screen. BlindAssist runs a YOLOv8 detector but turns the result into speech, so the user hears a sentence such as “cell phone detected on your right.”

BlindAssist is a local application written in Python. It was developed on a Windows laptop with a webcam, and once the models are downloaded it needs no internet connection because speech is generated offline. It is a prototype, and this paper describes it as one: what has been built, how it works, how it relates to published work, and what has not yet been tested.

Note on the term “near-real-time.” In this paper “near-real-time” means that the system works on a live webcam stream and speaks while it runs, instead of analyzing recorded video. A new detection pass starts only after at least 0.8 s have passed since the previous pass finished, so the time between passes is 0.8 s plus the time the pass itself takes. The face and expression stages run in the same thread and make each pass longer. Neither this interval nor the delay from an object appearing to speech has been measured (Section IX), so we do not claim hard real-time behavior.

A. Research Contribution

- A pipeline from webcam to speech that gives the name and direction (left, ahead, or right) of each detected object in one short sentence.
- Announcement logic that keeps track of which object classes are in view. A class is announced once when it appears, again if it changes direction, and once when it has been missing for three detection passes.
- Face-recognition modules (OpenCV YuNet and SFace, and the DeepFace library) that recognize people from stored reference photographs.

- A facial-expression stage that speaks the dominant expression label produced by the DeepFace emotion model for each detected face. It is implemented but not yet evaluated.
- A Roboflow dataset workflow for pen and pencil, which the standard COCO-trained model cannot detect.

The rest of the paper is organized as follows. Section II reviews related work, Section III states the research gap and objectives, Section IV describes the architecture, and Section V the implementation. Sections VI and VII give the algorithms and the configuration, Section VIII compares BlindAssist with published systems, Section IX defines the experimental methodology, and Sections X to XIII cover prototype status, limitations, future scope, and conclusion.

II. RELATED WORK

A. Surveys and Wearable Mobility Aids

Elmannai and Elleithy [5] give a comparative survey of wearable and portable assistive devices for visually impaired people, discussing the advantages, disadvantages, and accuracy of the main devices. Tapu et al. [6] survey wearable assistive devices and classify them using qualitative and quantitative measures defined with the help of the blind community. A large part of this literature is about mobility: detecting obstacles and guiding the user along a path. As an example, Poggi and Mattoccia [7] describe a wearable aid built on a custom RGB-D sensor based on passive stereo vision. It detects obstacles on an embedded computer, uses deep learning to categorize them, and gives audio messages and tactile feedback. Such systems rely on depth sensing and are aimed at obstacle avoidance. BlindAssist uses a single ordinary webcam, has no depth sensor, and does not perform obstacle detection or path guidance; its distance value is only an uncalibrated estimate for a few classes.

B. Camera-Based Object Recognition with Speech Output

Tapu et al. [8] present DEEP-SEE, which combines object detection, tracking, and recognition for navigational assistance of visually impaired users. Closer to our setting, Ahmed et al. [9] describe an assistive tool for a PC with a USB camera and headphones. It runs COCO-pretrained YOLOv7 on about one frame per second, converts the detected class labels into speech with pyttsx3, and compares YOLOv7 variants by parameters, average precision, and inference time. A master's thesis by Tavakoli Yarakhi [10] uses YOLOv8 with text-to-speech to give spatial information and object counts as audio, in several languages, for webcam, image, and video input; we have reviewed only the abstract of that thesis. These two works are the closest to BlindAssist in the components they use (YOLO, offline or local text-to-speech, a webcam or PC camera).

C. Face Recognition and General-Purpose Apps

Mocanu et al. [11] describe DEEP-SEE FACE, a mobile face-recognition system for visually impaired people. Microsoft's Seeing AI [12] is a smartphone app and research project that speaks short text, guides the capture of documents, scans barcodes, recognizes and describes people including facial expressions, describes scenes, and identifies currency. These systems cover a wider range of tasks than BlindAssist and run on mobile hardware. BlindAssist has a smaller scope: a local Python application that combines general object announcements with a known-person module and facial-expression announcements built from openly available libraries.

In summary, the reviewed work falls into navigation-oriented wearable devices, camera-based recognition tools with speech output, and general-purpose apps. Section VIII compares BlindAssist with representative examples from each group.

III. RESEARCH GAP AND OBJECTIVES

Object detection [3], [4], face detection and recognition, and text-to-speech are available as tools, and we did not train any new model for the main pipeline. The work here is connecting them so that the output is a short spoken sentence that a blind user can follow and that does not keep repeating. The gap below is stated only in terms of the published descriptions we reviewed, and it is not a claim that no other system addresses it.

A. Research Gap

- A plain detection script only draws boxes and labels on a screen, which a blind user cannot use. Speech output has been added in [9] and [10].
- A detector that runs repeatedly reports the same object again and again. In the description of [9] we found no mention of how repeated detections of the same object are handled before speech; in [10] we only reviewed the abstract. BlindAssist adds explicit announcement rules for arrival, direction change, and disappearance.

- The class name alone is not enough; the user also needs to know roughly where the object is. BlindAssist gives a three-region direction in every announcement.
- The standard YOLOv8 weights are trained on COCO [13]. COCO has no pen, pencil, keys, wallet, earphones, charger, glasses, or medicine box, and these are things a visually impaired user handles every day.
- Object detection, face recognition, and facial-expression analysis are separate models. In BlindAssist all three feed the same speech output.

B. Project Objectives

- Build a webcam-based application that detects objects and reports them by voice, with a configured minimum wait of 0.8 seconds after one detection pass finishes before the next one starts.
- Say where each object is (left, ahead, or right), and estimate the distance for a few known object types.
- Avoid repeating announcements for objects that stay in view.
- Recognize known people and announce their names.
- Announce the dominant facial expression of detected faces.
- Run on an ordinary laptop, using the GPU when CUDA is available and the CPU otherwise.
- Allow custom classes by loading a custom-trained YOLO model.

IV. SYSTEM ARCHITECTURE AND DESIGN

BlindAssist is organized as a short pipeline. OpenCV [16] reads frames from the webcam. A detection thread runs the YOLO model on the latest frame, and the results are filtered, compared with the objects already announced, and sent to a speech queue. A separate thread runs pytsx3, so speaking does not block the video window. The face module (YuNet and SFace) works on the camera frames and passes recognized names to the same speech output. The DeepFace library runs in the same detection thread for a second face-recognition branch and for facial-expression analysis (Section V-F), and its results are also spoken through the same queue. Fig. 1 shows the structure.

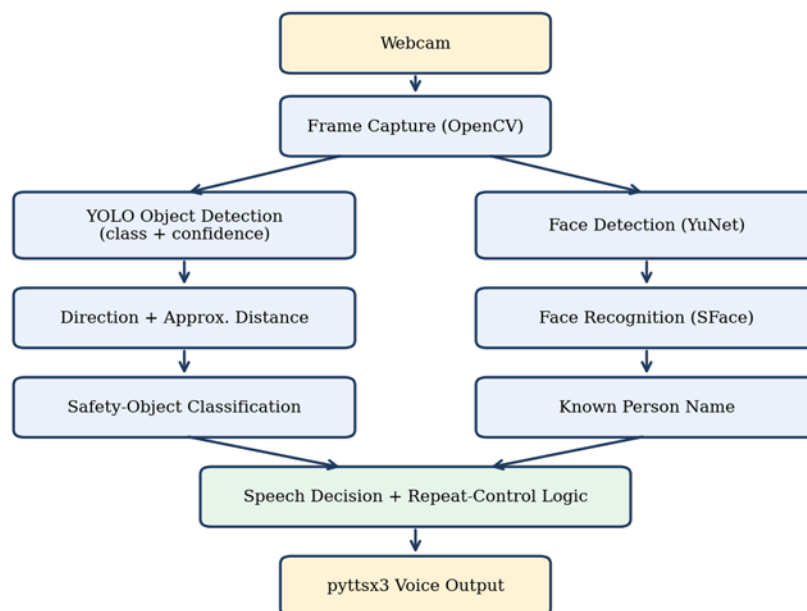


Fig. 1. BlindAssist system architecture showing the object-detection and face-recognition branches feeding a common speech-decision stage. The DeepFace recognition and expression stages (Section V-F) are not drawn in this figure.

A. Processing Flow

- The program is started with python main.py. The webcam is opened through OpenCV (DirectShow first, then the default backend, trying camera index 0 and then 1) and 15 warm-up frames are discarded.

- The main loop keeps reading frames and draws the latest boxes on them. A separate detection thread takes the most recent frame and runs YOLO on it, starting a new pass only after at least 0.8 seconds have passed since the previous pass finished; the face and expression stages run in the same thread after YOLO.
- Detections with a confidence below 0.5 are dropped.
- For each remaining detection, the direction is worked out from the horizontal center of the box. Distance is estimated from the box width for a few object types.
- A vote over the last three detection passes is applied, and then the announcement logic compares the current objects with the ones already announced.
- New objects, direction changes, and objects that have disappeared are put into a queue, and a speech thread reads them out with pyttsx3.
- An object is removed from the present list after it has been missing for three passes in a row, and it is announced as gone. If it comes back, it is announced as new.

V. METHODOLOGY AND IMPLEMENTATION

Table I lists the main tools. They were chosen so that the application can run locally on an ordinary laptop. The source code is in a public GitHub repository [20]. Sections V-A to V-G describe features that are implemented in the code; Sections V-H and V-I describe work that is still in progress.

TABLE I
TECHNOLOGY STACK OF BLINDASSIST

Layer	Technology	Purpose
Language	Python	Main programming language of the application.
Object detection	YOLOv8 / Ultralytics [4]	Object detection on video frames. Default weights: yolov8n.pt (COCO).
Deep-learning backend	PyTorch [17], CUDA	Runs the model. The GPU is used when CUDA is available, otherwise the CPU.
Image processing	OpenCV [16]	Webcam capture, drawing boxes and labels, and the video window.
Face detection	YuNet [14]	Face detection.
Face recognition	SFace [15]	Face feature extraction and matching against reference photographs.
Emotion recognition	DeepFace	Second face-recognition branch (VGG-Face) and dominant-expression analysis, run in the detection thread.
Voice output	pyttsx3 [18]	Offline text-to-speech (SAPI5 voices on Windows), run in its own thread.
Custom dataset	Roboflow [19]	Annotation and preparation of the custom pen and pencil dataset.
Development	VS Code, virtual environment, Git/GitHub [20], Jenkins	Coding, dependency isolation, version control, and CI (in progress).

A. Object Detection

Frames are passed to a YOLOv8 model through the Ultralytics library. The default weights are yolov8n.pt, which is pretrained on COCO. Inference uses an image size of 640 and a confidence threshold of 0.5. Each detection gives a box, a class name, and a confidence score. Detection runs in its own thread, which starts a new pass only after at least 0.8 seconds have passed since the previous pass finished, while the video window keeps updating at the camera frame rate using the most recent boxes. The model runs on the GPU if CUDA is available and on the CPU if not.

B. Direction Estimation

Direction depends only on the horizontal position of the box. The center of the box is compared with the frame width: below 33% of the width is left, above 66% is right, and anything in between is ahead. Vertical position is not used. A phone whose box center is in the left third is announced as “cell phone detected on your left,” and one in the middle as “cell phone detected ahead of you.” Direction is also written on the label of each box in the video window (Fig. 2). It is a position in the camera image, not a measured position in space.

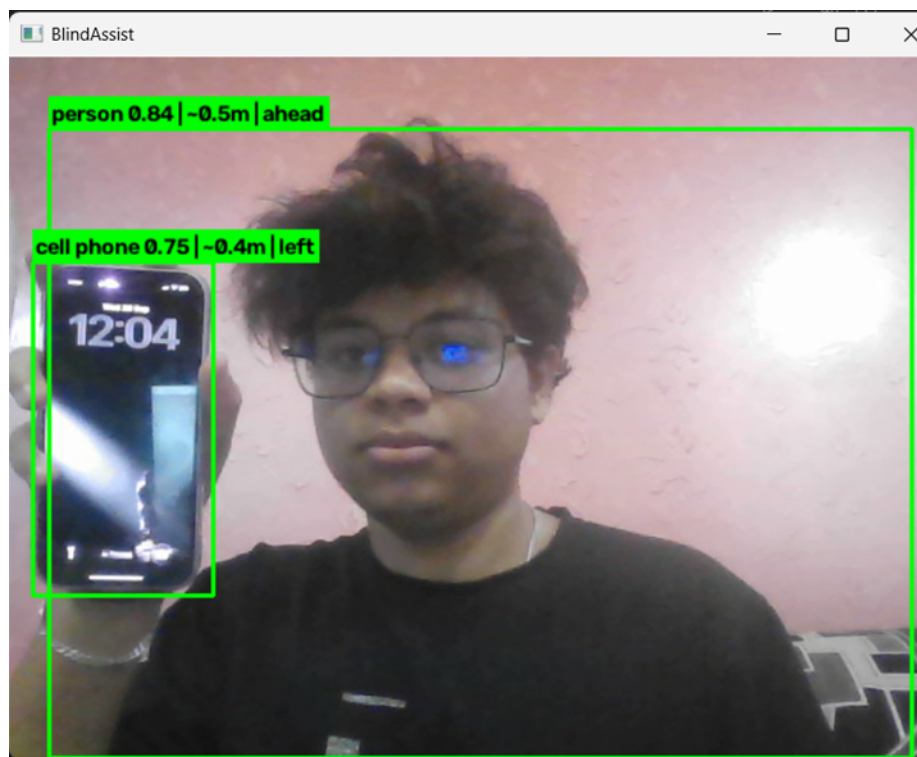


Fig. 2. Video window during a development run. Each box is labeled with the class, confidence, estimated distance (for supported classes), and direction. The distance values are uncalibrated estimates.

C. Approximate Distance Estimation

Distance is calculated only for seven classes whose real width is stored in the program: person, cell phone, bottle, laptop, chair, backpack, and cup. It uses the pinhole camera relation, $\text{distance} = (\text{real width} \times \text{focal length}) / \text{box width in pixels}$, with the focal length set to 700 pixels. This value was set by hand and the camera has not been calibrated, so the result is only a rough estimate and must not be read as a measured physical distance. The box width also changes with the angle of the object and when it is partly out of view, and the stored real widths are fixed values while real objects vary. For all other classes no distance is given. In the current code the distance appears only on the label in the video window (as ~x m) and is not spoken. Calibrating the focal length, for example with a standard camera-calibration procedure [21], and comparing the estimates with measured distances is future work.

D. Voice Output

Announcements are short sentences spoken with pyttsx3, which uses the SAPI5 voices on Windows. A normal object gives “cell phone detected on your right.” and an object in the middle gives “cell phone detected ahead of you.” The speech rate is set to 170. One pyttsx3 engine runs in its own thread and reads sentences from a queue. An earlier version created a new engine for every sentence, which caused crashes and garbled speech. The voice quality depends on the voices installed on the computer.

E. Repeat Control and Detection Stability

The program keeps a set of objects that are currently present, tracked by class name. A class that is not in the set is announced and added to it. While it stays visible nothing more is said, unless its direction changes, in which case the program says, for example, “cell phone is now left.” If a class is not detected in a pass, a miss counter goes up, and after three misses in a row it is announced as

gone (“cell phone gone.”) and removed from the set. If it appears again later, it is announced as new. The counter stops a single missed detection from causing a “gone” message followed immediately by a new announcement. Because tracking is by class name and not by individual object, two objects of the same class, two phones for example, are treated as one, and only the direction of the last box is kept. Each event is also printed in the console as [SPEAK], [MOVE], [MISS], or [GONE].

Before the announcement logic, a simple vote is applied. The labels found in each of the last three detection passes are counted, and a label is kept if it appears at least `vote_threshold` times. The threshold is now 1. It was 2 in an earlier version, but objects were dropped and never announced, so it was lowered. With a threshold of 1 the vote has no filtering effect, because every label in the current pass is counted at least once, so every detection above 0.5 confidence is passed on. A higher value would remove some false detections but could delay or lose real ones. The best setting still has to be tested.

F. Face Recognition and Facial Expression

The face stage has two parts. The first uses two models available through OpenCV. YuNet [14] finds faces in the frame, and SFace [15] computes a feature vector for each face. The vector is compared with vectors computed from the reference photographs kept in the `known_faces` folder, using cosine similarity with a threshold of 0.363 and a minimum face-detection score of 0.8. A face is assigned to the smallest person box that contains its center, and when a match is found the class label “person” is replaced by the person’s name, so the name is announced by the same announcement logic as any other class. A recognized name is kept for 3 s if the face is briefly not visible. The second part uses the DeepFace library [22], [23] in the same detection thread. It extracts faces with its OpenCV detector, compares each face with the photographs in the `known_faces` folder using the VGG-Face model [24], and speaks “<name> detected” when the set of names changes or after 5 s. In the current code this branch returns a name for one enrolled person only, whose name is written into the program, and returns “Unknown person” otherwise; generalizing it to all enrolled people is future work. For each detected face DeepFace also analyzes facial emotion and returns the dominant expression class (by default angry, disgust, fear, happy, sad, surprise, or neutral), which is spoken as “<name> appears to be <emotion>.” Emotion analysis runs at most once every 4 s, and the label is spoken each time it runs, so an unchanged expression is repeated. The label is an estimate of the expression shown on the face, not a determination of the person’s actual emotional state. How well these parts work depends on the lighting, the viewing angle, and the number and quality of the reference photographs. They were run informally during development (Fig. 3), and their accuracy has not been measured.

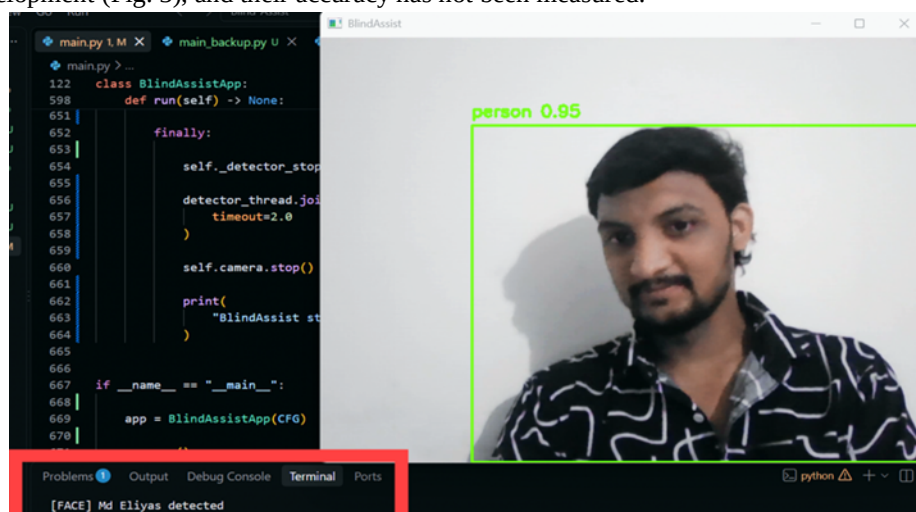


Fig. 3. Screenshot from a development run of the face module. The console line reports that a known person was detected.

G. Objects of Interest and Default-Model Coverage

The default COCO weights contain some objects that matter to a visually impaired user and lack others. Table II lists groups of objects of interest and whether the default model can detect them. The class names knife and scissors are in COCO, so these sharp objects are detected and announced like any other class, with no separate warning and no announcement priority. Such a warning mechanism is not implemented and is listed under future work. Names are matched exactly against the class names of the loaded model, so a class only works if the model has it, and the other objects in Table II need a custom model.

TABLE II
OBJECT GROUPS OF INTEREST AND THEIR AVAILABILITY IN THE DEFAULT MODEL

Category	Objects	Availability
Daily-use objects	Book, notebook, remote, keys, wallet, earphones, charger, mug, glasses, umbrella.	Book, remote, and umbrella are COCO classes. The rest need a custom model.
Pen and pencil	Pen, pencil.	In development; requires a custom-trained model.
Health-related objects	Medicine box.	Custom model required.
Sharp or hazardous objects	Knife, scissors, blade, cutter, needle, syringe, broken glass, hammer.	Knife and scissors are COCO classes. The rest need a custom model.
Other detected classes	Any class in the loaded model, for example person and cell phone.	Works with COCO. No special message.

Classes are matched by name, so a class only works if the loaded model has a class with exactly that name. Groups that need a custom model are marked as such and are not claimed to work.

H. Custom Object Detection (Under Development)

The default COCO weights [13] cannot detect pen, pencil, keys, wallet, earphones, charger, glasses, notebook, or medicine box, so these need a custom-trained model. We started with pen and pencil, because a general detector cannot tell them apart. The dataset is being prepared and annotated in Roboflow [19], with pictures taken at different angles, distances, backgrounds, lighting conditions, and positions in the frame. The trained weights are not yet part of the running program. A custom weights file would be loaded through the model path in the configuration.

I. Development Automation Using Jenkins (In Progress)

Jenkins is being connected to the GitHub repository. The planned job fetches the latest code, sets up the Python environment, installs the dependencies, and runs basic checks. This is part of the development workflow and has no effect on detection accuracy.

VI. ALGORITHMS

Algorithm 1 gives the announcement logic that runs once per detection pass, and Algorithm 2 gives the direction and distance rules. SPEAK(.) places a sentence in the speech queue; it does not wait for the speech to finish. The message templates are those in Section V: “<class> detected <direction>.” for a new object, “<class> is now <direction>.” for a direction change, “<class> gone.” for a disappeared object; the same templates apply when a recognized name replaces the class label. Algorithm 1 covers object classes only; the DeepFace face and expression messages follow the separate rules of Section V-F.

Algorithm 1. Announcement logic for one detection pass

Input: detections D (class, box, confidence); frame width Wf

State: present[class] → direction; miss[class] → count; hist = last 3 label sets

Constants: CONF = 0.5, WINDOW = 3, vote_threshold = 1, MISS_LIMIT = 3

1: cur ← empty map

2: for each d in D with d.conf ≥ CONF do

3: cur[d.class] ← DIRECTION(d.box, Wf) (last box of a class is kept)

4: append labels(cur) to hist; keep only the last WINDOW sets

5: V ← { c in cur : c occurs in at least vote_threshold sets of hist }

6: for each c in V, new classes in alphabetical order do

7: miss[c] ← 0

8: if c not in present then present[c] ← cur[c]; SPEAK(new-object message)

9: else if cur[c] ≠ present[c] then present[c] ← cur[c]; SPEAK(direction-change message)

10: for each c in present with c not in V do

11: miss[c] ← miss[c] + 1

12: if miss[c] ≥ MISS_LIMIT then SPEAK(“c gone.”); remove c from present and miss

Algorithm 2. Direction and approximate distance for one detection

Input: box (x1, y1, x2, y2), class, frame width Wf

Constants: F = 700 px (set by hand); REAL_WIDTH stored for 7 classes

1: $xc \leftarrow (x1 + x2) / 2$; $r \leftarrow xc / Wf$

2: if $r < 0.33$ then direction $\leftarrow$ left

3: else if $r > 0.66$ then direction $\leftarrow$ right else direction $\leftarrow$ ahead

4: if class in REAL_WIDTH then

5: $Z \leftarrow REAL_WIDTH[class] \cdot F / (x2 - x1)$ (approximate, uncalibrated; displayed only)

6: return direction, Z (if defined)

VII. SYSTEM CONFIGURATION

Table III lists the parameters of the current implementation. They are values set in the program or its configuration; none of them has been tuned on a labeled test set.

TABLE III
ACTUAL SYSTEM PARAMETERS AND CONFIGURATION

Parameter	Current value	Note
Detector weights	yolov8n.pt (COCO-pretrained)	Set through the model path; a custom weights file can be loaded the same way.
Inference image size	640	Passed to the Ultralytics predictor.
Confidence threshold	0.5	Detections below it are dropped.
Detection interval	0.8 s minimum wait	Minimum wait after a pass finishes before the next one starts; the real interval between passes is this wait plus the processing time of the pass (the face and expression stages run in the same thread) and has not been measured.
Compute device	CUDA GPU if available, else CPU	Selected at start-up.
Camera indexes tried	0, then 1	First index that opens is used.
Capture backends tried	DirectShow, then default	Windows development machine.
Warm-up frames discarded	15	Before processing starts.
Direction rule	< 33% left; > 66% right; otherwise ahead	Horizontal box center relative to frame width; vertical position unused.
Distance estimation	Focal length 700 px; 7 classes	Person, cell phone, bottle, laptop, chair, backpack, cup. Uncalibrated; displayed only, not spoken.
Vote window / threshold	3 passes / 1	Threshold 1 gives no filtering effect.
Miss limit for “gone”	3 consecutive passes	Then the class is removed from the present set.
Tracking key	Class name	Two objects of one class are treated as one.
Face matching (OpenCV)	Cosine threshold 0.363; face score 0.8; name held 3 s	YuNet + SFace; a matched name replaces the label “person”.

Parameter	Current value	Note
Face and emotion (DeepFace)	VGG-Face recognition; emotion every 4 s	Face names repeat after 5 s and emotion labels every 4 s; both run in the detection thread.
Speech engine / rate	pyttsx3 (SAPI5), 170	One engine in its own thread, fed from a queue.
Face models	YuNet + SFace; DeepFace	Reference photographs in the known_faces folder; DeepFace runs in the detection thread.

VIII. COMPARISON WITH EXISTING SYSTEMS

Table IV compares BlindAssist with four published systems from the groups identified in Section II. The comparison is functional. It uses only what each cited source states; “not described” means we did not find the item in the material we reviewed, and it does not mean the system lacks it. It is not a performance comparison, because no performance figures have been measured for BlindAssist.

TABLE IV
FUNCTIONAL COMPARISON WITH PUBLISHED SYSTEMS

System	Platform and input	Output	Spatial information	Repeat handling	Face recognition	Evidence reported in source
Ahmed et al. [9]	PC, USB camera, headphones; COCO-pretrained YOLOv7; about 1 frame/s	Offline speech (pyttsx3) of detected class labels	Not described	Not described	Listed as future work	Inference time and COCO AP of YOLOv7 variants; abstract also mentions subjective assessments
Tavakoli Yarak [10] (thesis)	Webcam, images, video; YOLOv8 (abstract)	Text-to-speech, several languages; object counts	Spatial location information (abstract)	Not stated in abstract	Not stated in abstract	Abstract reports “promising results”; full text not reviewed
Seeing AI [12]	Smartphone camera; Microsoft research project	Spoken text, documents, barcodes, people, scenes, currency	Not stated on project page	Not stated on project page	Recognizes and describes people, including expressions	None on the cited page
Poggi and Mattoccia [7]	Wearable; custom RGB-D (passive stereo) sensor; embedded computer	Audio messages and tactile feedback	Obstacle detection from 3-D data	Not stated in abstract	Not part of described system	Experimental results on obstacle detection and categorization (abstract)

System	Platform and input	Output	Spatial information	Repeat handling	Face recognition	Evidence reported in source
BlindAssist (this work)	Laptop webcam (single camera); YOLOv8, default COCO weights	Offline speech (pyttsx3): class and direction; face names; facial-expression labels	Left / ahead / right from box center; uncalibrated distance shown on screen only	Announces arrival, direction change, and disappearance after 3 missed passes (object classes only)	Known people (YuNet + SFace; DeepFace VGG-Face); expression label (DeepFace); not measured	None yet: no measured results

Table IV shows two things. First, BlindAssist and the systems in [9] and [10] use the same kind of components, so the difference is in what gets spoken and when: the announcement rules of Algorithm 1, the known-person module, and the facial-expression announcements. Whether these rules give a better listening experience than plain label speech is an open question that the user study in Section IX is meant to answer; we make no such claim now. Second, BlindAssist does not offer what the other groups provide: depth-based obstacle detection as in [7], or the range of functions of [12]. It should be read as a small aid and not as a replacement for either.

IX. EXPERIMENTAL METHODOLOGY

This section describes the tests we plan to run. None of them has been carried out yet, so no results are reported. We will decide the number of trials before collecting data, and any figure we report later will state its test conditions and number of trials.

A. Test Setup and Logging

Each experiment will record the computer (CPU, GPU, RAM, operating system), the webcam model and resolution, Python and library versions, the weights file, and the configuration of Table III, so that the results can be reproduced.

B. Object Detection

A labeled test set of the target classes will be collected with the same webcam, under different lighting, distances, backgrounds, and partial occlusion, and kept separate from any data used to choose parameters. Precision, recall, F1-score, and mean average precision will be computed per class and overall at the deployed confidence threshold. The custom pen and pencil model will be evaluated separately on its own held-out images.

C. Direction

An object will be placed at known positions in the left, middle, and right thirds of the frame, including positions near the 33% and 66% boundaries. The spoken direction will be compared with the true region and summarized in a 3×3 confusion matrix.

D. Approximate Distance

Each supported object will be placed at measured distances, and the estimate of Algorithm 2 will be compared with the measured value using absolute and relative error per class. The focal length will then be calibrated [21] and the error compared before and after calibration.

E. Announcement Logic

Scripted scenarios will be recorded and annotated with the true events: an object enters, stays, moves, leaves, and returns; and two objects of the same class are present. For each scenario we will count the spoken messages per appearance, missed or false “gone” messages, and missed direction changes. The same recordings will be replayed with different vote thresholds and miss limits to choose these two parameters on evidence.

F. Face and Expression Recognition

Known people will be enrolled with reference photographs, and genuine and impostor attempts will be tested under different lighting, angles, and numbers of reference photographs. True and false recognition rates will be reported for each condition. Facial-expression labels will be evaluated by asking participants to show posed expressions and comparing the spoken label with the requested one, reported per class; posed expressions may differ from spontaneous ones.

G. Timing

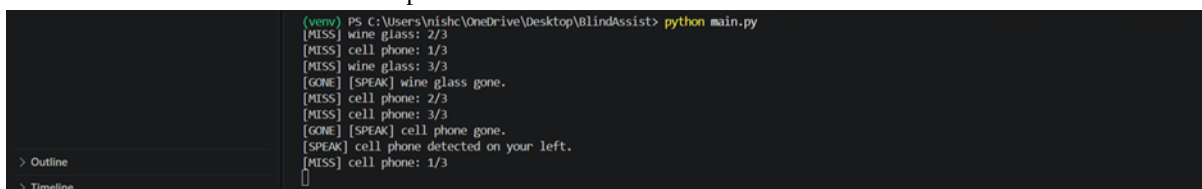
Timestamps will be logged at frame capture, at the end of each detection pass, and at the start of speech. From these we will report the detection pass rate, the delay from an object appearing to speech, and the video display rate, on CPU-only and GPU systems.

H. User Study

A study with visually impaired volunteers will be conducted only after institutional ethics approval and informed consent. Tasks will include locating and naming everyday objects in a room. Measures will include task completion, misunderstood or unwanted announcements, and structured feedback. Results from sighted or blindfolded participants, if used in a pilot, will be reported as such and not as evidence for blind users.

X. PROTOTYPE STATUS AND DEVELOPMENT OBSERVATIONS

BlindAssist has been developed and tested on a Windows laptop with a webcam. The development laptop has an AMD Ryzen 7 7735HS processor, 16 GB of RAM, and an NVIDIA GeForce RTX 3050 6 GB Laptop GPU. The code requests a frame size of 680 × 480 pixels from the camera. During development we ran the program and checked, from the video window and the console messages, that detection, direction, voice output, and the announcement logic behave as described in Section V (Figs. 2 and 4). The face and expression stages were also run informally during development (Fig. 3). These were informal checks and not measurements. A labeled test set and a user study have not been completed, so we report no accuracy, speed, or user-study results. We prefer to leave the results out rather than report numbers we did not measure.



```
(venv) PS C:\Users\nishc\OneDrive\Desktop\BlindAssist> python main.py
[MISS] wine glass: 2/3
[MISS] cell phone: 1/3
[MISS] wine glass: 3/3
[GONE] [SPEAK] wine glass gone.
[MISS] cell phone: 2/3
[MISS] cell phone: 3/3
[GONE] [SPEAK] cell phone gone.
[SPEAK] cell phone detected on your left.
[MISS] cell phone: 1/3
```

Fig. 4. Console output during a development run: miss counters ([MISS]), a spoken “gone” event, and a spoken new-object announcement with direction ([SPEAK]).

Table V separates what is implemented from what is under development and what is future work. “Implemented” means the feature is in the code and has been run during development; it does not mean it has been measured.

TABLE V
IMPLEMENTATION STATUS OF BLINDASSIST FEATURES

Status	Feature	Notes
Implemented	Webcam capture and display (OpenCV)	Camera index and backend fallback, 15 warm-up frames.
Implemented	YOLOv8 object detection (COCO weights)	Class, box, and confidence; threshold 0.5.
Implemented	Direction estimation	Left, ahead, or right from the box center.
Implemented	Offline voice output	pyttsx3 in a separate thread with a queue.
Implemented	Repeat control	Arrival, direction change, and gone after 3 missed passes; tracked by class name.
Implemented	Face recognition of known people	YuNet and SFace inside person labelling, plus a DeepFace VGG-Face branch; not measured.

Status	Feature	Notes
Implemented, with limits	Approximate distance	Seven classes; uncalibrated; shown on screen only.
Implemented	Facial emotion recognition (DeepFace)	Dominant expression spoken every few seconds; not measured.
Under development	Custom pen and pencil model	Roboflow dataset in preparation; weights not yet in the running program.
Under development	Jenkins CI	Being connected to the GitHub repository; a development aid only.
Not yet done	Labeled test set, accuracy evaluation, user study	Protocol in Section IX.
Future work	See Section XII	Not part of the current implementation.

XI. LIMITATIONS

- 1) The system needs a working camera, and poor lighting lowers detection quality.
- 2) Partly hidden objects may be missed.
- 3) The default yolov8n.pt (COCO) weights cannot detect most objects in Table II. Pen and pencil need a custom model trained on enough varied images.
- 4) Sharp objects such as knives and scissors are announced like any other class, with no separate warning or priority.
- 5) Distance works for only seven classes and depends on a focal length set by hand (700 pixels) and fixed real widths. It is not spoken and must not be treated as a measurement.
- 6) Objects are tracked by class name, so two objects of the same class are treated as one and only one direction is reported.
- 7) A new detection pass starts only after at least 0.8 seconds have passed since the previous pass finished, and the face and expression stages run in the same thread, so the real interval between passes is this wait plus the processing time of the pass, which is longer when faces are in view. It depends on the CPU or GPU and has not been measured, so a fast-moving object may be announced late or missed.
- 8) With a vote threshold of 1, a false detection above 0.5 confidence is announced.
- 9) Face recognition and expression labels depend on image quality, lighting, angle, and the quality of the reference photographs, and have not been measured.
- 10) The DeepFace recognition branch currently returns a name for one enrolled person only, with the name written into the program, and "Unknown person" otherwise.
- 11) The repeat control of Algorithm 1 applies to object classes only. Face names (every 5 s) and expression labels (every 4 s) are repeated while a face stays in view, and a known person can be announced twice, once as an object and once by the DeepFace branch.
- 12) Expression labels estimate the expression on the face, not the person's actual emotional state, and posed and spontaneous expressions may be classified differently.
- 13) Speech is queued without a limit, so announcements can lag behind the scene when many are generated.
- 14) The comparison in Section VIII is functional and based on published descriptions; no performance comparison has been made.
- 15) The system is only an aid and should not be a user's only safety measure.

XII. FUTURE SCOPE

- 1) Collect a labeled test set, run the evaluation of Section IX, and test with visually impaired users.
- 2) Finish the custom pen and pencil model, add more object classes, and track individual objects across frames.
- 3) Calibrate the camera, extend distance estimation to more classes, and speak the distance. A depth sensor or stereo camera could also be tried.

- 4) Evaluate and improve facial-expression recognition: measure its accuracy on more participants, lighting conditions, and face angles, compare alternative models, test spontaneous as well as posed expressions, and decide whether and how expression labels should be spoken.
- 5) Add a warning message and announcement priority for dangerous objects such as knives and scissors, generalize the DeepFace branch to all enrolled people, and merge the face and expression announcements with the object announcements.
- 6) Add voice commands, adjustable speech speed, and multilingual speech, including Hindi and Marathi. Support recognition of several people at once and a user-managed face database.
- 7) Port the system to wearable or embedded hardware such as smart glasses, a Raspberry Pi, a mobile phone, or a wearable camera with Bluetooth earphones.
- 8) Longer-term ideas beyond the current scope: obstacle detection and path guidance, indoor navigation and GPS, scene description, text reading (OCR), currency recognition, and product recognition.

XIII. CONCLUSION

BlindAssist is a webcam-based prototype that turns YOLOv8 detections into spoken sentences giving the object's name and direction. A class is spoken once when it appears, again if it moves, and once when it is gone. Face modules can name known people, and a DeepFace emotion stage announces the dominant facial expression. Distance is estimated for seven classes only and is shown on screen but not spoken. Compared with the published systems reviewed here, the prototype uses similar components but adds explicit announcement rules, a known-person module, and facial-expression announcements; it offers neither depth-based obstacle detection nor the breadth of a general-purpose app. It is a small aid and not a navigation system for blind users.

The main limits are the COCO-only default model, the uncalibrated distance, and the lack of any measured results. The next steps are to build a labeled test set, calibrate the distance estimate, train the pen and pencil model, and test with visually impaired users before making any claims about accuracy or usefulness. The Jenkins setup is also still in progress.

XIV. ACKNOWLEDGMENT

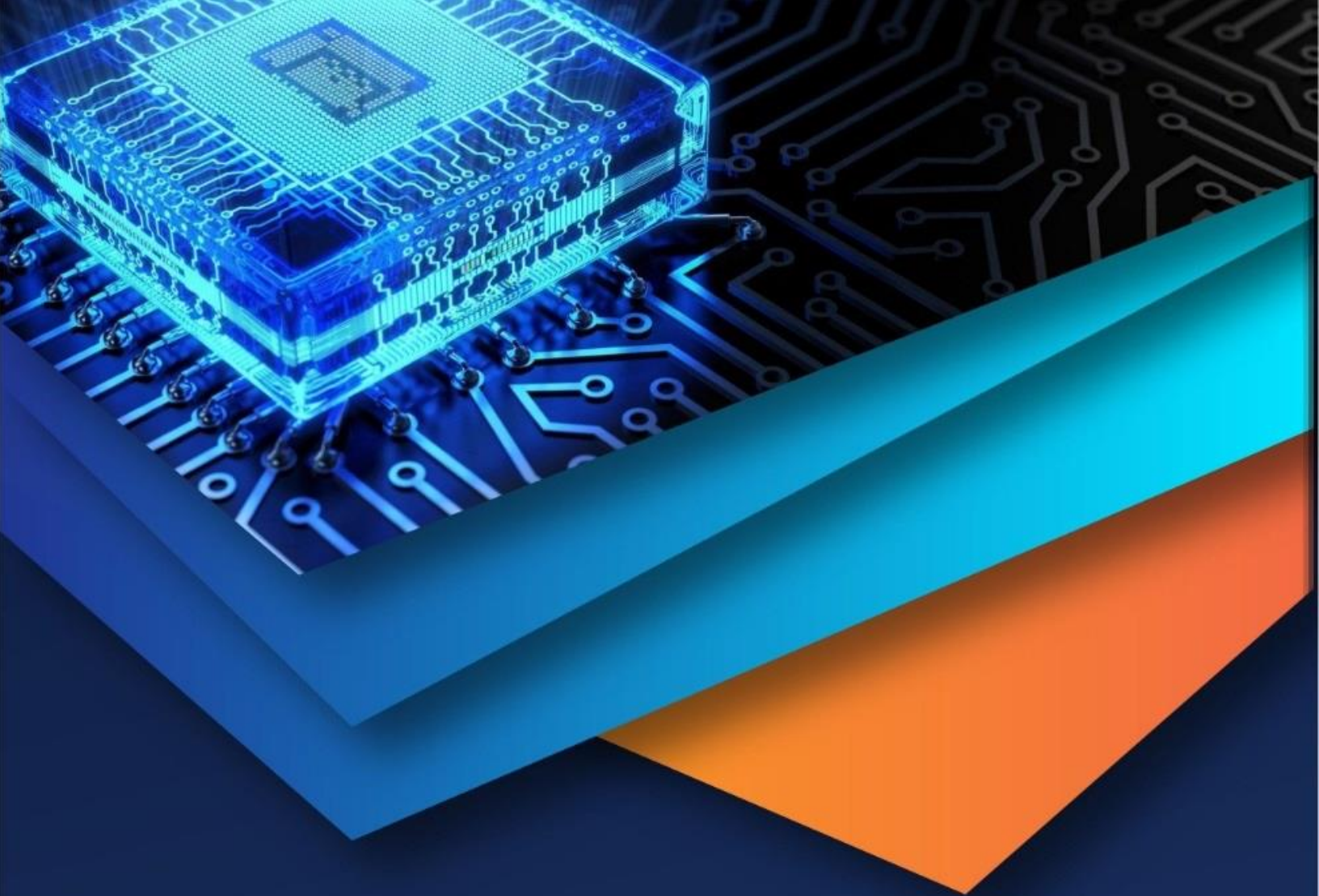
The authors would like to thank Mr. Vaibhav Shrivastava for guidance and support during the development of BlindAssist, and the Department of Computer Science and Engineering, G H Raison University, Amravati, for the academic support and facilities provided.

Nishchay Sakure developed the overall system, including YOLOv8-based object detection, OpenCV webcam integration, voice output, and the object announcement logic, carried out testing, and prepared the paper. Chanchal Pagare contributed to project research, the literature review, analysis of existing assistive systems, documentation, and testing of the prototype. Md. Eliyas Sheikh contributed to object-detection testing, preparation and organization of object samples, system observations, and documentation of the development work. Sapna Debnath contributed to research documentation, analysis of system requirements, testing of different detection scenarios, and preparation of the paper content. Saurabh Sharnagat contributed to system testing, identification of practical issues during development, analysis of detection and voice-feedback behavior, and project documentation. Harsha Keche contributed to testing, collection and organization of supporting material, documentation, and review of the paper and project presentation. Mr. Vaibhav Shrivastava supervised the project, provided technical guidance, reviewed development progress, and suggested improvements to the system and the research work.

REFERENCES

- [1] World Health Organization, World Report on Vision. Geneva, Switzerland: WHO, 2019.
- [2] R. R. A. Bourne et al., "Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment: a systematic review and meta-analysis," *The Lancet Global Health*, vol. 5, no. 9, pp. e888–e897, 2017, doi: 10.1016/S2214-109X(17)30293-0.
- [3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.
- [4] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics>
- [5] W. Elmannai and K. Elleithy, "Sensor-based assistive devices for visually-impaired people: Current status, challenges, and future directions," *Sensors*, vol. 17, no. 3, Art. no. 565, 2017, doi: 10.3390/s17030565.
- [6] R. Tapu, B. Mocanu, and T. Zaharia, "Wearable assistive devices for visually impaired: A state of the art survey," *Pattern Recognition Letters*, vol. 137, pp. 37–52, 2020, doi: 10.1016/j.patrec.2018.10.031.
- [7] M. Poggi and S. Mattocchia, "A wearable mobility aid for the visually impaired based on embedded 3D vision and deep learning," in *Proc. IEEE Symp. Computers and Communications (ISCC)*, 2016, pp. 208–213, doi: 10.1109/ISCC.2016.7543741.

- [8] R. Tapu, B. Mocanu, and T. Zaharia, "DEEP-SEE: Joint object detection, tracking and recognition with application to visually impaired navigational assistance," *Sensors*, vol. 17, no. 11, Art. no. 2473, 2017, doi: 10.3390/s17112473.
- [9] G. J. Ahmed, F. H. Khorsheed, and F. K. Zaidan, "Designing an assistive tool for visually impaired people based on object detection technique," *J. Informatics, Network, and Computer Science (JOINCS)*, vol. 8, no. 2, 2025. [Online]. Available: <https://joincs.umsida.ac.id>
- [10] A. Tavakoli Yarak, "Seeing with sound: Object detection and localization by YOLOv8 and audio feedback for blind individuals," M.S. thesis, Univ. of Padua, Padua, Italy, 2024. [Online]. Available: <https://thesis.unipd.it/handle/20.500.12608/66485>
- [11] B. Mocanu, R. Tapu, and T. Zaharia, "DEEP-SEE FACE: A mobile face recognition system dedicated to visually impaired people," *IEEE Access*, vol. 6, pp. 51975–51985, 2018, doi: 10.1109/ACCESS.2018.2870334.
- [12] Microsoft, "Seeing AI." [Online]. Available: <https://www.microsoft.com/garage/wall-of-fame/seeing-ai>
- [13] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Computer Vision – ECCV 2014*, LNCS vol. 8693. Springer, 2014, pp. 740–755, doi: 10.1007/978-3-319-10602-1_48.
- [14] W. Wu, H. Peng, and S. Yu, "YuNet: A tiny millisecond-level face detector," *Machine Intelligence Research*, vol. 20, no. 5, pp. 656–665, 2023, doi: 10.1007/s11633-023-1423-y.
- [15] Y. Zhong, W. Deng, J. Hu, D. Zhao, X. Li, and D. Wen, "SFace: Sigmoid-constrained hypersphere loss for robust face recognition," *IEEE Trans. Image Processing*, vol. 30, pp. 2587–2598, 2021, doi: 10.1109/TIP.2020.3048632.
- [16] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
- [17] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.
- [18] pyttts3 contributors, "pyttts3: Offline text-to-speech library for Python." [Online]. Available: <https://pytpi.org/project/pyttts3/>
- [19] Roboflow, "Roboflow: Computer vision developer framework." [Online]. Available: <https://roboflow.com>
- [20] N. Sakure, C. Pagare, Md. E. Sheikh, S. Debnath, S. Sharnagat, H. Keche, and V. Shrivastava, "BlindAssist," GitHub repository, 2026. [Online]. Available: <https://github.com/CoderNishchay/Blind-Assist>
- [21] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000, doi: 10.1109/34.888718.
- [22] S. I. Serengil and A. Ozpinar, "LightFace: A hybrid deep face recognition framework," in *Proc. Innovations in Intelligent Systems and Applications Conf. (ASYU)*, 2020, pp. 23–27, doi: 10.1109/ASYU50717.2020.9259802.
- [23] S. I. Serengil and A. Ozpinar, "HyperExtended LightFace: A facial attribute analysis framework," in *Proc. Int. Conf. Engineering and Emerging Technologies (ICEET)*, 2021, pp. 1–4, doi: 10.1109/ICEET53442.2021.9659697.
- [24] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proc. British Machine Vision Conf. (BMVC)*, 2015, pp. 41.1–41.12, doi: 10.5244/C.29.41.



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)