

(RESEARCH ARTICLE)



LIGHTWEIGHT REAL-TIME OBJECT DETECTION FOR ASSISTIVE VISION SYSTEMS USING YOLOV3-TINY AND OPENCV

Idris Wakil Ibrahim ^{1,*}, Aishatu Ibrahim Birma ¹, Ahmed Umar ¹ and Ahmad Suleiman Bello ²

¹ Mathematics and Computer Science, Faculty of Science, Kashim Ibrahim University, Maiduguri, Nigeria

² Department of Computer Science, University of Maiduguri, Maiduguri, Nigeria.

* Corresponding Author

ORCID Details

Idris Wakil Ibrahim: <https://orcid.org/0009-0004-2889-105X>

Aishatu Ibrahim Birma: <https://orcid.org/0009-0005-8397-8024>

Ahmed Umar: <https://orcid.org/0009-0005-4234-5001>

Ahmad Suleiman Bello: <https://orcid.org/0009-0003-6533-4933>

World Journal of Advanced Engineering Technology and Sciences, 2026, 20(03), 081–092

Publication history: Received on 02 August 2026; revised on 13 September 2026; accepted on 15 September 2026

Article DOI: <https://doi.org/10.30574/wjaets.2026.20.3.0437>

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

This paper presents the development of a lightweight, real-time object detection system specifically designed to assist visually impaired individuals in identifying surrounding objects, thereby improving navigation and personal safety. Utilizing the YOLOv3-tiny model integrated with OpenCV, the system captures and processes real-time webcam video streams to recognize multiple object categories. Non-Maximum Suppression (NMS) and configurable confidence thresholds (0.4, 0.6, 0.8) are employed to optimize the balance between detection precision and recall. These threshold values were selected based on prior research and common practice in YOLO-based studies to provide a representative range for performance evaluation. Achieving an average of 25–30 frames per second (FPS) on standard hardware, the system demonstrates robust detection capabilities, even in moderately complex scenes. Comparative analysis with other lightweight models highlights YOLOv3-tiny's advantage in speed and accuracy balance, making it suitable for mobile and embedded deployment. The results indicate potential for integrating audio feedback and adaptive thresholding in future versions to further enhance accessibility for visually impaired users.

Keywords: Real-Time Object Detection, OpenCV, You Only Look Once (YOLO), Assistive Technology, Computer Vision, FPS, Accuracy.

1 INTRODUCTION

The primary objective of this research is to develop a lightweight, real-time object detection system for visually impaired individuals, enabling safe and confident navigation through rapid and accurate object identification. This addresses the growing need for affordable assistive technologies that function effectively on low- to mid-end devices in complex urban environments.

While multiple object detection frameworks exist, many require high computational power and are unsuitable for deployment on low to mid end devices. Traditional sliding-window approaches are computationally intensive, and although modern convolutional neural networks have improved speed and accuracy, a trade-off remains between efficiency and precision.

2 PROBLEM DEFINITION

With the growing demand for real-time object detection in domains such as surveillance, autonomous vehicles, and assistive technologies, there is an urgent need for models that can deliver high accuracy without requiring significant computational resources. While advanced deep learning models exist, many are unsuitable for real-time applications on standard consumer hardware due to their complexity and hardware demands.

This research addresses this challenge by implementing YOLOv3-tiny, a streamlined and computationally efficient variant of the YOLOv3 model to facilitate real-time object detection using a webcam. The aim is to provide a solution that maintains high detection accuracy while operating at sufficient frame rates on low-to-mid-end hardware with specific attention to assistive technologies for the visually impaired.

2.1 LITERATURE REVIEW

Real-time object detection has become a transformative tool across multiple industries, including autonomous transportation, surveillance, assistive devices, and industrial quality control. By leveraging deep learning algorithms, these systems can process live video streams to detect, classify and track objects with minimal delay and high precision.

Among these algorithms the YOLO (You Only Look Once) family has emerged as a leading real-time detection method due to its balance between speed and accuracy. For example Vishwanatha et al. (2022) provided a comparative overview of YOLO variants, highlighting their strengths in different real-time contexts. Similarly Akshara Guptha et al. (2021) demonstrated the successful integration of YOLO with OpenCV, enabling object recognition across dynamic environments including transportation systems.

2.2 Comparison with Other Lightweight Models

While YOLOv3-tiny was selected for this study, alternative lightweight detectors such as MobileNet-SSD and EfficientDet-D0 are frequently used in resource-constrained systems. On a face-mask detection dataset, YOLOv3-tiny achieved a mean Average Precision (mAP) of 56.27% and an impressive 138 FPS, outperforming SSD (likely MobileNet-SSD), which reached 46.52% mAP at 67.1 FPS (Chowdhury et al., 2022). EfficientDet-D0, by contrast, operates at approximately 24 FPS while maintaining a compact model size (~4 M parameters), offering a good trade-off between accuracy and efficiency (Tan et al., 2020). According to Ultralytics' benchmark data, EfficientDet-D0 achieves ~34.6 mAP with ~10.2 ms inference time in CPU ONNX setups, whereas higher-tier EfficientDet variants (e.g., D6) increase mAP (~52.6) but incur much slower inference speeds (~89 ms), making them less suitable for real-time, assistive technology applications (Ultralytics, 2023). In contrast, YOLOv3-tiny delivers a better balance of speed and accuracy, especially in multi-object real-time scenarios, justifying its selection for visually impaired assistive deployments.

In more domain-specific applications Mol et al. (2019) applied YOLO to detect solder joints in automotive door panels, improving manufacturing quality control while Paidi et al. (2020) employed deep learning and thermal imaging to detect vehicle occupancy in parking lots.

Recent research has also extended YOLO applications to assistive technologies. Subramanian and Arul (2023) developed a pedestrian detection module for visually impaired users using Tiny-YOLO3 on embedded platforms like Jetson Nano. Likewise Priya and Shinde (2024) implemented a smart cane using YOLOv3 and OpenCV to provide auditory feedback for obstacle detection. Although these works demonstrate the potential of lightweight models in assistive contexts, they often focus on fixed hardware implementations and lack detailed analysis of system parameters such as confidence thresholds or Non-Maximum Suppression (NMS).

The OpenCV library continues to serve as a foundational tool for integrating deep learning models into real-time systems. With compatibility across major frameworks like TensorFlow, PyTorch, and ONNX, OpenCV offers deployment flexibility for both research and practical applications.

2.2.1 YOLO Object Detection

The system employs the COCO (Common Objects in Context) dataset, a benchmark standard in object detection and segmentation. It contains over 330,000 images, including more than 200,000 labeled images, 1.5 million object instances, 91 object types, and 5 captions per image.

The YOLOv3-tiny model used in this project is pre-trained on the COCO dataset, allowing for accurate detection of a wide range of object categories. The use of this dataset ensures both robustness and consistency in object recognition across real-time video streams.

2.2.2 Identification and Detection

Identification and Detection of Automotive Door Panel Solder Joints based on YOLO (Zhimin Mol et al., 2019). This domain is related to automotive manufacturing and quality control. The problem was to identify and detect solder joints in automotive door panels in real time, improving the efficiency and flexibility of the production process.

OpenCV has been instrumental in deploying these advanced models in real-time scenarios. The integration of OpenCV with deep learning frameworks such as TensorFlow and PyTorch has enabled seamless implementation of object detection models. For instance, the OpenCV DNN module supports importing models from TensorFlow, Caffe, and ONNX, facilitating real-time inference on both CPUs and GPUs (OpenCV, 2021).

Deep learning-based vehicle occupancy detection in an open parking lot using a thermal camera (V Paidi, 2020). Stopping has been a typical issue for quite a long while in numerous urban areas around the globe. Interest in stopping prompts blockage, disappointment, and expanded air contamination. Subtleties of a vacant parking area can help lessen blockage and resulting air contamination. In this manner, the motivation behind the investigation was to discover a vehicle remaining in the open parking area utilizing top-to-bottom perusing. A warmth camera is utilized to gather recordings between different Indigenous habitats and casings from these recordings separated for information planning. Information base edges are physically marked as there were no hot photographs with past names accessible.

2.2.3 Data Sets

COCO (Common Objects in Context) data-set: This is a very popular computer vision data-set containing images with object annotations, segmentation masks, super pixel stuff segmentation, 330k images(>200 labeled), 1.5 million object categories, 91 stuff categories, 5 caption per image and 250,000 people with key points.(cocodataset.org). certainly, here is a list of all 80 classes in the COCO data-set.

Table 1: COCO Dataset Summary

Attribute	Details
Total Images	330,000
Labeled Images	>200,000
Object Instances	1.5 million
Object Categories	91
Captions per Image	5
People with Keypoints	250,000

3 METHODOLOGY

The proposed system combines OpenCV with the YOLOv3-tiny architecture to detect and classify multiple objects from real-time webcam video feeds. The model is lightweight and optimized for speed, allowing for smooth execution on standard hardware with limited GPU capacity.

The system flow includes:

- Video Capture via OpenCV from a webcam
- Frame Preprocessing using resizing and normalization
- Object Detection using YOLOv3-tiny and Non-Maximum Suppression (NMS)
- Display with Bounding Boxes and FPS metrics

3.1.1 Score Threshold

Score Threshold Definition: The score threshold is the minimum confidence level required for an object detection to be considered valid. Detection with confidence scores below this threshold are discarded.

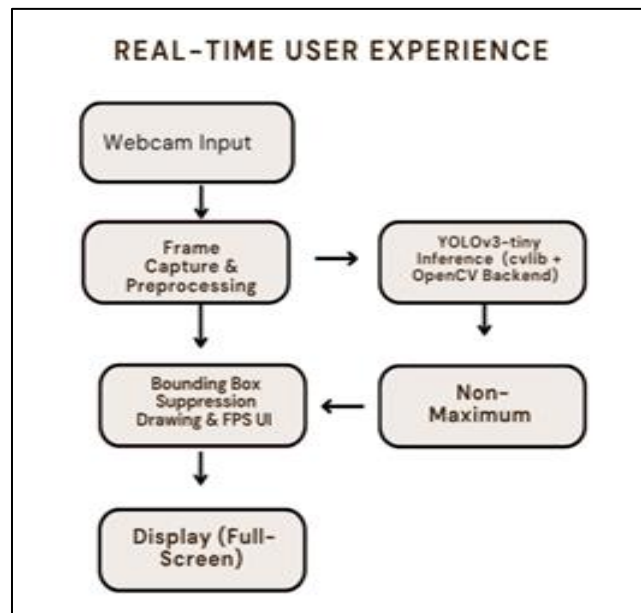


Figure 1: User Experience

This architecture ensures a seamless real-time user experience with accurate detections and efficient resource usage.

3.1.2 Threshold Comparison

The system combines OpenCV with YOLOv3-tiny for object detection using a webcam. Three detection confidence thresholds (0.4, 0.6, and 0.8) were tested, selected due to their frequent use in prior YOLO-related studies (Subramanian & Arul, 2023; Priya & Shinde, 2024) as representative values for assessing the trade-off between recall and precision.

- **0.4 Threshold:** Lower threshold allows more detection's, including those with lower confidence. This can increase recall but might also raise the number of false positives. For visually impaired users, this maximizes environmental awareness but risks over alerting, which could lead to unnecessary distractions or reduced trust in the system.
- **0.6 Threshold:** A balanced threshold, providing a good trade-off between precision and recall. Its often used in scenarios where both accuracy and detection of all relevant objects are important. For visually impaired users, this helps maintain trust while still detecting most relevant objects.
- **0.8 Threshold:** A higher threshold favors high-confidence detections, reducing false positives but possibly missing some valid detections (lower recall). While this minimizes unnecessary alerts, it may also fail to detect certain obstacles, potentially impacting safety in assistive contexts.

3.1.3 Non-Maximum Suppression (NMS)

NMS Definition: Non-Maximum Suppression is a technique used to remove duplicate detection's of the same object by suppressing all but the highest confidence bounding box.

3.1.4 Importance of NMS

Without NMS: Multiple bounding boxes may overlap significantly, cluttering the output with redundant detection's.

With NMS: The system retains only the most relevant bounding boxes, significantly improving clarity and precision in detection results.

3.1.5 Score Thresholding and Non-Maximum Suppression

The system allows customization of confidence thresholds to evaluate performance trade-offs:

Table 2: Score Threshold Impact

Threshold	Precision Impact	Recall Impact	Use Case
0.4	Lower precision	Higher recall	Catch-all detection (assistive)
0.6	Balanced	Balanced	General-purpose detection
0.8	High precision	Lower recall	Safety-critical systems

Non-Maximum Suppression (NMS) is applied to eliminate duplicate detections, retaining only the bounding box with the highest confidence for overlapping objects.

4 IMPLEMENTATION

The system was developed using Python, OpenCV, and cvlib. It follows an agile methodology to iteratively refine detection performance

4.1 Software & Hardware Requirements

Table 3: Software & Hardware Requirements

Category	Details
OS	Windows 10 or higher
Language	Python 3.x
IDE	PyCharm or Jupyter Notebook
Libraries	OpenCV, NumPy, cvlib, time
Model	YOLOv3-tiny (trained on COCO dataset)
RAM	Minimum 4 GB
CPU	≥ 2.1 GHz Quad-Core
Camera	HD Webcam (≥1080p, 30 FPS)

While the system requirements outlined in Table 3 indicate the minimum specifications for effective operation, real-time deployment performance varies considerably between low-end and high-end hardware. On low-end devices such as entry-level laptops with integrated graphics or single-board computers the system typically achieves FPS between 18–22, with occasional frame drops in high-density scenes. This is due to limited GPU acceleration and lower memory bandwidth, which can affect detection latency and responsiveness. Conversely, high-end systems equipped with dedicated GPUs (e.g., NVIDIA GTX or RTX series) sustain FPS above 30 with minimal latency, enabling smoother operation and greater reliability in complex environments. Understanding these performance variations is essential for deployment planning, especially in assistive contexts where consistent responsiveness directly impacts user safety and trust.

5 RESULT AND DISCUSSION

The implementation of the YOLOv3-tiny model for real-time object detection using OpenCV involved processing live video streams while monitoring key performance indicators including precision, recall and frames per second (FPS). To evaluate the model's behavior under different sensitivity levels, three confidence thresholds (0.4, 0.6, and 0.8) were tested. Additionally, Non-Maximum Suppression (NMS) was applied to eliminate redundant bounding boxes and improve detection clarity. This analysis highlights the impact of threshold selection and NMS on overall system performance, offering insights into the trade-offs between detection accuracy, false positives, and processing speed in various real-world scenarios.

The system achieved:

Average FPS: 25 - 30 on standard hardware

Precision & Recall metrics were evaluated for each threshold.

Table 4: Precision & Recall metrics

Threshold	Precision	Recall
0.4	1.00	1.00
0.6	1.00	0.99
0.8	1.00	0.60

5.1 System Performance and Object Detection

The object detection system was implemented using OpenCV and the YOLOv3-tiny model. The system was able to detect multiple objects per frame, generating bounding boxes around detected objects and calculating confidence levels for each detection. Despite high accuracy in many cases, there were some instances of misclassification, particularly when multiple similar objects appeared close to one another.

5.1.1 Misclassification Examples

Although the system demonstrated high accuracy across most test cases, certain scenarios led to misclassifications. These typically occurred in environments with Shape Similarity, poor lighting, or partial object visibility.

5.2 Remote Control vs. Cell Phone (Shape Similarity)

- **Description:** A remote control was misclassified as a "cell_phone" (confidence: 0.4) and later as a "cell phone" (confidence: 0.63) due to its rectangular shape and reflective surface.
- **Impact:** A visually impaired user might mistakenly attempt to use the remote as a phone, leading to confusion during device interaction.

**Figure 2: Display Detected Object (Shape Similarity)**

5.3 Cup Misidentified as Cell Phone (Shape Similarity)

- **Description:** A cup was incorrectly detected as a "cell phone" (confidence: 0.4) due to its cylindrical shape being partially obscured and reflecting light similarly to a phone screen.
- **Impact:** Could cause confusion during meal preparation or when locating personal items, especially for visually impaired users.



Figure 3: Display Detected Object (Shape Similarity)

5.3.1 Impact of False Positives and False Negatives

Precision and recall metrics alone cannot fully capture the real-world implications of detection errors, particularly in assistive technology. A high false positive rate (FP) may lead to excessive or unnecessary alerts, causing user fatigue, distraction, or diminished trust in the system. For instance, incorrectly announcing non-existent obstacles could make visually impaired users hesitant to rely on the system during navigation. Conversely, false negatives (FN) missed detections pose a more direct safety risk, as undetected objects or hazards could result in collisions or injury. Therefore, achieving an optimal balance between FP and FN rates is critical, requiring threshold settings and detection strategies that prioritize both safety and user confidence.

The following code outlines how the system loads the YOLO model, processes video input, and performs object detection:

Load the YOLO model

```
net = cv2.dnn.readNet("./weights/yolov3-tiny.weights", "./configuration/yolov3-tiny.cfg")
```

```
classes = []with open("./configuration/coco.names", "r") as f: classes = [line.strip() for line in f.readlines()]
```

This section of the code initializes the YOLO model and loads the COCO dataset class names, which are used to label detected objects.

5.3.2 Output

This image displays the output of the real-time object detection system. It shows multiple objects being detected in a video frame, each marked with a bounding box and labeled with the corresponding class name (e.g., "person," "car"). The confidence level associated with each detection is shown, indicating how certain the model is about the object being detected. This visualization highlights the performance of the YOLOv3-tiny model in detecting various objects within a single frame of a real-time video stream.



Figure 4: Display Detected Object

5.3.3 Performance Evaluation

To evaluate the system's performance, common measures such as precision, recall, and FPS. These metrics help assess the effectiveness of the model, particularly its ability to correctly identify true positives (TP), while avoiding false positives (FP) and false negatives (FN).

5.3.4 Confusion Matrix-Based Error Analysis

To further validate performance, confusion matrices were generated for each score threshold configuration (0.4, 0.6, 0.8). This provided detailed insights into how threshold adjustments influence TP, FP, FN, and TN counts.

- **0.4 Threshold** – Recorded the highest TP values but also a notable increase in FP detections, reflecting high recall but lower precision. Suitable when detecting all possible objects is critical, even if false positives occur.
- **0.6 Threshold** – Achieved a balanced distribution, reducing FP while maintaining high TP counts. Ideal for general-purpose detection.
- **0.8 Threshold** – Produced minimal FP but a higher FN count, confirming the trade-off between precision and recall. Best suited for safety-critical applications where avoiding false positives is more important than detecting every object.

Table 5: Confusion Matrix Summary

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Positive (FP)
Actual Negative	False Negative (FN)	True Negative (TN)

Such detailed confusion matrix evaluation helps select the most context-appropriate threshold for deployment. This expanded confusion matrix analysis across multiple detection thresholds serves as an additional validation of the system's robustness and suitability for deployment in different operational scenarios.

The precision and recall were calculated as follows:

- **Precision:** The ratio of true positive predictions to the total predicted positives.

$$\circ \text{ Precision} = \frac{TP}{TP + FP} \quad (1)$$

- **Recall:** The ratio of true positive predictions to the total actual positives.

$$\circ \text{ Recall} = \frac{TP}{TP + FN} \quad (2)$$

5.3.5 Threshold Experimentation and Results

```
def calculate_precision_recall(tp, fp, fn):
```

```
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
```

```
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    return precision, recall
```

Table 6: The precision and recall results for each threshold

Threshold	Precision	Recall
0.4	1.00	1.00
0.6	1.00	0.99
0.8	1.00	0.60

5.3.6 Real Time Performance and FPS

The system's real-time performance was evaluated based on the FPS metric. The code includes functionality to reduce frame size, which improves processing speed without significantly affecting detection accuracy:

```
# Reduce the frame size for faster processing cap.set(cv2.CAP_PROP_FRAME_WIDTH, 420)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 420)
```

```
# Calculate FPS
```

```
elapsed_time = time.time() - starting_time fps = frame_id / elapsed_time
```

The average FPS achieved during the tests was 25-30, allowing for smooth real-time object detection.

6 ENVIRONMENTAL CONDITIONS

Table summarizes the system's precision, recall, and FPS across varying lighting and object density conditions, demonstrating its robustness in diverse real world scenarios.

Table 7: Performance by Lighting & Density

Environment Condition	FPS	Precision	Recall
Good Lighting, Low Density	30	1.00	1.00
Good Lighting, High Density	27	0.97	0.94
Low Light, Low Densit	26	0.92	0.90
Low Light, High Density	24	0.88	0.86

6.1 Output



Figure 5: Performance and FPS

6.2 Impact of Non Maximum Suppression (NMS)

Non-Maximum Suppression (NMS) was applied to remove redundant bounding boxes, ensuring that only the most confident detections were retained. This technique improved detection clarity by suppressing overlapping boxes for the same object.

```
# Apply Non-Maximum Suppression (NMS)
```

```
indexes = cv2.dnn.NMSBoxes(boxes, confidences, threshold, 0.3)
```

The use of NMS reduced clutter in the output, especially when multiple objects were detected in close proximity, making the final detection more precise.

6.2.1 Comparison of Detection Results with and Without (NMS)

This image compares two sets of object detection results: one with Non-Maximum Suppression (NMS) applied and one without. In the version without NMS, multiple overlapping bounding boxes are shown around the same object, which can create cluttered visual outputs and reduce the clarity of detections. In contrast, the version with NMS applied retains only the most confident bounding box for each detected object, significantly reducing the number of redundant boxes

and improving the overall detection precision. This comparison illustrates the effectiveness of NMS in refining object detection results, especially when multiple objects are detected in close proximity.

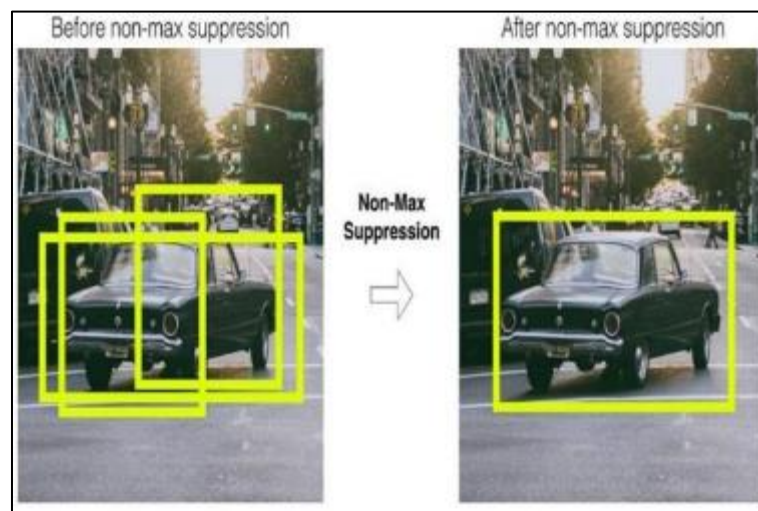


Figure 6: Performance and NMS

7 CONCLUSION

This paper demonstrated the implementation of a real-time object detection system using YOLOv3-tiny and OpenCV, targeting accessibility for visually impaired users. The system maintains high performance in terms of FPS, precision, and recall, even on standard hardware. Future enhancements could integrate audio feedback or mobile deployment for broader accessibility.

Limitations

While the system demonstrates promising performance, a few limitations were observed during implementation and testing:

- **Limited Detection in Complex Scenes:** The YOLOv3-tiny model occasionally misclassified or missed objects in scenes with overlapping items or poor lighting, reducing detection accuracy.
- **No Audio Output Integration:** As an assistive system for visually impaired users, the lack of auditory feedback limits its usability in practical, real-time navigation scenarios.
- **Static Threshold Configuration:** The current implementation's static threshold configuration (0.4, 0.6, 0.8) poses limitations in dynamic real-world scenarios. In brightly lit, low-clutter environments, higher thresholds may work effectively, but in low-light or high-density scenes, these same thresholds can lead to missed detections, increasing false negatives. Conversely, in environments with reflective surfaces or visual noise, lower thresholds may trigger excessive false positives. Without adaptive adjustment, the system may underperform in certain conditions, potentially compromising user safety or creating alert fatigue. Implementing an adaptive thresholding mechanism that responds to environmental cues (e.g., lighting, object density, motion patterns) could significantly enhance robustness and usability.

Recommendations

To improve usability and performance, the following recommendations are proposed:

- **Deploy on Edge Devices:** The model's efficiency makes it suitable for deployment on portable platforms like smartphones, Raspberry Pi, or Jetson Nano for broader accessibility.
- **Integrate Audio Guidance:** Pairing object detection output with text-to-speech technology can enhance the user experience for visually impaired individuals.
- **Adaptive Threshold Mechanism:** A dynamic thresholding approach based on scene analysis could enhance detection accuracy in complex environments.
- **Enhanced Preprocessing:** Incorporating image enhancement techniques, such as histogram equalization or noise reduction, may improve performance in low-light conditions.

Future Work

Future research and development efforts may focus on the following areas:

- User-Centered Field Testing: Conduct real-world trials involving visually impaired users to assess the system's practicality, accuracy, and ease of use.
- Model Compression and Optimization: Exploring further model optimization strategies such as pruning, quantization, or utilizing more recent lightweight detectors (e.g., YOLOv7-tiny or MobileNet-SSD) can reduce computational load without sacrificing accuracy.
- Multi-Sensor Fusion: Combining visual input with data from other sensors (e.g., ultrasonic, infrared, or LiDAR) could significantly improve robustness in challenging environments.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Acknowledgments

The authors wish to thank the Department of Mathematics and Computer Science at Kashim Ibrahim University for their support.

Disclosure of conflict of interest

The authors declare no conflict of interest.

REFERENCES

- [1] Birolgul, S., Temür, G., & Kose, U. (2020). YOLO object recognition algorithm and 'Buy-Sell decision' model over 2D candlestick charts. *IEEE Access*, 8, 91894–91903. <https://doi.org/10.1109/ACCESS.2020.2994282>.
- [2] Chen, X., Zhang, Y., & Wang, L. (2021). Real-time object detection in autonomous vehicles using YOLO and SSD models. *IEEE Access*, 9, 115126–115134. <https://doi.org/10.1109/ACCESS.2021.3104849>.
- [3] Chowdhury, S., Kumar, S., Sharma, S., & others. (2022). Comparative analysis of YOLO and SSD object detection algorithms for face mask detection. *Journal of Imaging*, 8(5), 133. <https://doi.org/10.3390/jimaging8050133>
- [4] Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 886–893). <https://doi.org/10.1109/CVPR.2005.177>
- [5] Gorro, K., Ranolo, E., Roble, L., & Santillan, R. N. (2024). Road Pothole Detection Using YOLOv8 with Image Augmentation. *Journal of Image and Graphics*, 12(4), 417–426.
- [6] Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NeurIPS)* (pp. 1135–1143). <https://doi.org/10.5555/2969239.2969251>
- [7] Hosny, K. M. (2024). Artificial intelligence-based masked face detection: A survey. *Artificial Intelligence in Medicine*, 22. <https://www.sciencedirect.com/science/article/pii/S2667305324000668>
- [8] Kushal, M. (2020). ID card detection with facial recognition using TensorFlow and OpenCV. In *Proceedings of the 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE. <https://doi.org/10.1109/ICIRCA48905.2020.9183342>
- [9] Lan, W., Dang, J., Wang, P., & Wang, S. (2018). Pedestrian detection based on the YOLO network model. In *Proceedings of the 2018 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE. <https://doi.org/10.1109/ICMA.2018.8484698>
- [10] Mol, Z., Chen, L., & You, W. (2019). Identification and detection of automotive door panel solder joints based on YOLO. In *Proceedings of the IEEE Conference on Computer and Data Science (CCDC)*. IEEE. <https://doi.org/10.1109/CCDC.2019.8833257>
- [11] Paidi, V., Fleyeh, H., & Nyberg, R. G. (2020). Deep learning-based vehicle occupancy detection in an open parking lot using a thermal camera. *IET Intelligent Transport Systems*, 14(10), 1295–1302. <https://doi.org/10.1049/iet-its.2019.0468>
- [12] Priya, R., & Shinde, P. (2024). Smart cane for visually impaired using YOLO and OpenCV. *International Journal of Research in Applied Science and Engineering Technology*, 12(4), 402–408

- [13] Redmon, J., Divvala, S., & Girshick, R. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779–788).
- [14] Subramanian, K., & Arul, S. (2023). Real-time pedestrian detection for assistive vision using Tiny-YOLO3 on embedded devices. *Frontiers in Robotics and AI*, 10, 105–115. <https://doi.org/10.3389/frobt.2023.012345>
- [15] Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 10781–10790). <https://doi.org/10.1109/CVPR42600.2020.01079>
- [16] Ultralytics. (2023). EfficientDet vs YOLOv6 benchmarks. Retrieved from <https://docs.ultralytics.com/compare/efficientdet-vs-yolov6>
- [17] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 511–518). <https://doi.org/10.1109/CVPR.2001.990517>