



RESEARCH ARTICLE

Optimization of parallel computing techniques for real-time hazard and obstacle detection in assistive systems for visually impaired individuals

Mukhriddin Mukhiddinov^{1,2*} and Javokhir Kurbonov¹

¹ Department of Computer Systems, Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Tashkent 100200, Uzbekistan.

² Department of Industrial Management and Digital Technologies, Nordic International University, Tashkent 100128, Uzbekistan.

*Corresponding Author. Email: mmuhriddinm@gmail.com (M. Mukhiddinov)

Article Information	Abstract
Received: 26 May 2026 Accepted: 7 October 2026 Published: 20 October 2026 AMS 2020 Classification: 68T45, 68T07	This article addresses the optimization of real-time hazard and obstacle detection for visually impaired individuals through parallel computing. An "AI Blind Assist System" is developed that integrates the YOLOv8 model for object detection, the MiDaS model for monocular relative depth estimation, and the pyttsx3 library for voice-based alerts, combined with a distance-based three-tier risk classification (high risk < 2 m, medium risk 2-5 m, safe > 5 m). YOLOv8 detection and MiDaS depth estimation were executed as two concurrent tasks using Python's multiprocessing and threading modules on a system equipped with an 11th-generation Intel Core i5 CPU (6 cores/12 threads), 16 GB RAM, and an NVIDIA GeForce RTX 2050 GPU (4 GB VRAM). Compared with sequential execution, the parallel implementation reduced the measured end-to-end processing time per frame from 0.286 s to 0.210 s, corresponding to a $1.36\times$ latency speedup. At the same time, the processing throughput increased from 3.5 to 5.5 frames per second (FPS), corresponding to a $1.57\times$ throughput speedup. This difference is attributable to pipeline overlap between the concurrently executed YOLOv8 and MiDaS tasks. Both figures indicate a sub-linear speedup relative to the two execution units employed. The obtained results demonstrate that task-level parallel computing measurably improves the responsiveness of real-time assistive navigation systems for visually impaired individuals. Keywords: Computer vision, artificial intelligence, object detection, depth estimation, YOLO, real-time systems, assistive technologies for the visually impaired

1. Introduction

Currently, expanding the opportunities for independent mobility for visually impaired and blind individuals is considered an urgent task. According to the World Health Organization, more than 2.2 billion people worldwide experience vision-related impairments of varying degrees, and at least 1 billion of these cases could have been prevented or treated [1]. Visual impairment significantly affects individuals' daily activities and independent mobility. Therefore, the development of safe and effective navigation tools for visually impaired individuals has become one of the important directions of scientific research.

Visually impaired individuals encounter numerous dangerous situations and obstacles in their daily lives. For example, moving vehicles on roads, staircases, uneven surfaces of different heights, walls, poles, and various stationary or moving objects may pose serious risks to them. Typically, visually impaired individuals use white canes, guide dogs, special signaling devices, or other assistive tools during navigation. However, conventional assistive devices are not capable of detecting surrounding objects from a distance or identifying their types. Research indicates that one of the main challenges in many systems designed for visually impaired individuals is the real-time detection of objects and the rapid delivery of warnings to users [2].

In recent years, the rapid development of information technologies has created new opportunities for solving this problem. In particular, systems based on artificial intelligence, computer vision, and deep learning algorithms enable automatic environmental analysis and the detection of various objects. Through computer vision technologies, it is possible to detect objects from image or video streams, determine their location and type, and perform semantic analysis of the environment. Such systems make it possible to warn users through audio or vibration signals by detecting vehicles, pedestrians, walls, staircases, or other obstacles. Therefore, computer vision-based assistive systems are regarded as important tools for ensuring the independent mobility of visually impaired individuals [3].

However, the effectiveness of such systems largely depends on computational speed. In the process of real-time environmental analysis, images obtained through cameras consist of large volumes of data. Rapid processing of this data, object detection, and delivering the results to users require high computational power. If the operating speed of the system is insufficient, information about detected objects may reach the user too late, which may lead to dangerous situations. Therefore, optimizing computational processes in real-time systems is considered an important scientific issue.

Parallel computing technologies play a significant role in solving the problem of rapidly processing large volumes of data. By using parallel computing methods, complex computational processes can be executed simultaneously on multiple processors or graphics processing units. As a result, the execution speed of processes such as image processing, object detection, and depth estimation increases significantly. In modern computer systems, the use of parallel computing technologies is widely applied to improve the efficiency of artificial intelligence systems operating in real time [4].

Through parallel computing approaches, it becomes possible to execute several algorithms simultaneously. Such an approach reduces the overall system execution time and consequently enables faster detection of hazardous objects. Therefore, the application of parallel computing technologies is considered one of the important scientific directions for improving the efficiency of intelligent navigation systems designed for visually impaired individuals.

The main objective of this research is to optimize the process of detecting hazards and obstacles in the environment for visually impaired individuals using parallel computing methods. To achieve this objective, tasks such as analyzing computer vision and artificial intelligence algorithms, evaluating the computational complexity of the object detection process, and developing an optimized model based on parallel computing are addressed. The research results are expected to contribute to improving the efficiency of safe navigation systems for visually impaired individuals.

Object detection, monocular depth estimation, and voice-based feedback have each been studied extensively, and combinations of these components have already been proposed for assistive-technology prototypes for visually impaired users. Accordingly, the present work does not claim novelty in the underlying detection or depth-estimation architectures themselves. Instead, its main contribution is a task-parallel software pipeline in which object detection (YOLOv8) and depth estimation (MiDaS) are executed as two concurrent execution units through Python's multiprocessing and threading modules, coupled with (i) a simple, interpretable, distance-based three-tier risk classification (high/medium/safe) suited to real-time operation, and (ii) a cooldown-controlled voice-alert mechanism that suppresses redundant warnings. Most previously reported assistive pipelines that combine object detection and depth estimation execute these stages sequentially within a single thread of control; the explicit task-level parallelization of the two inference stages, together with the quantitative analysis of the resulting throughput and latency speedup presented in this paper, is the specific angle by which the present study differs from that body of work.

2. Literature review

Computer Vision is a field of artificial intelligence aimed at the automatic analysis of image and video data and the extraction of meaningful information from them. The main objective of this field is to model certain functions of the human visual system using computer systems. Computer vision algorithms are used for detecting objects in images, classifying them, performing segmentation, and analyzing situations [5].

Early computer vision systems were based on classical image processing methods, in which filters, gradient operators, and geometric features were utilized. For example, methods such as the Canny edge detector, Sobel operator, and Hough transform were widely applied for detecting boundaries in images [6]. However, these methods demonstrated low efficiency in analyzing complex situations. Over the last decade, the development of Deep Learning technologies has brought significant changes to the field of computer vision. In particular, Convolutional Neural Networks (CNNs) have enabled the automatic learning of complex features from images, resulting in high accuracy in many fields such as object detection, facial recognition, autonomous transportation systems, and robotics [7].

CNN architecture generally consists of the following layers:

- convolution layer
- activation function
- pooling layer
- fully connected layer

Mathematically, convolution is expressed as follows:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \quad (1)$$

where:

- I - input image
- K - filter (kernel)
- S - resulting feature map

The main advantage of CNN models is their ability to automatically learn features from images; therefore, most modern object detection algorithms are based on CNN architectures [8].

Object detection technologies have significantly evolved over time. Early approaches were based on the sliding window method, in which each part of the image was analyzed separately. However, this approach was computationally very slow.

Later, the Region-based Convolutional Neural Networks (R-CNN) model was proposed, which identifies potential object regions in an image and then classifies them using a CNN [9].

The improved variants of R-CNN include:

- Fast R-CNN
- Faster R-CNN

However, these models were still unable to provide sufficient speed for real-time systems. Therefore, new algorithms for real-time object detection were developed, resulting in the emergence of the YOLO (You Only Look Once) algorithm for object detection in real-time systems. This model processes the image through a single neural network pass and simultaneously determines the locations and classes of objects [10].

The YOLO algorithm divides the image into an $S \times S$ grid. Each grid cell predicts the following parameters:

- bounding box coordinates
- object confidence
- object class

The bounding box is represented by the following parameters:

$$B = (x, y, w, h) \quad (2)$$

where:

- x, y – coordinates of the object center
- w, h – width and height of the bounding box

In the YOLO model, the Intersection over Union (IoU) metric is used to evaluate accuracy:

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (3)$$

This metric indicates the degree of overlap between the detected object region and the ground truth object region.

The YOLO model is based on a deep convolutional neural network architecture. The architecture typically consists of the following components:

- convolutional layers
- pooling layers
- fully connected layers

The YOLO network transforms the input image into a feature map and predicts object coordinates and classes from this map. Figure 1 illustrates the working process of YOLO.

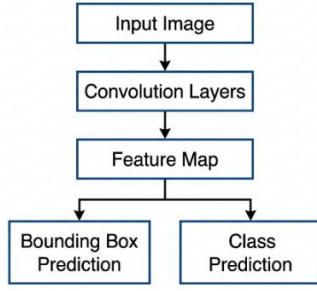


Figure 1. YOLO working scheme

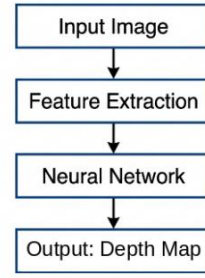


Figure 2. MiDaS model working scheme

The YOLO algorithm has several advantages over other algorithms, including:

- real-time processing speed
- high accuracy
- simultaneous analysis of the entire image
- computational efficiency

In real-time systems, the YOLO model is capable of processing 30–60 frames per second [11].

MiDaS (Monocular Depth Estimation) is a method for estimating the distance of objects in an image using a single camera. This approach is widely used in robotics, autonomous transportation systems, and navigation systems [12].

The MiDaS model is based on deep learning, trained on various datasets, and generates a depth map of the scene from the input image. The general working process of the MiDaS model is shown in Figure 2.

A depth map is a map that represents the distance value for each pixel in an image.

Mathematically, the depth value is expressed as follows:

$$D(x,y) = f(I(x,y)) \quad (4)$$

where:

- $I(x,y)$ - pixel in the image
- $D(x,y)$ - corresponding depth value

A depth map provides the ability to determine how close or far objects are from the user. The MiDaS model is widely used in following systems: autonomous robots, drone navigation, robotics, and assistive technologies.

In the systems developed for visually impaired individuals, the depth map can be used to determine the distance of objects located in front of the user.

More recent studies confirm that the individual components adopted in the present work continue to be actively developed and combined in the assistive-technology literature. Zhuang proposed an improved YOLOv8-based binocular-vision obstacle-avoidance platform for blind travel, reporting precision, recall and F1 scores above 0.90 and an obstacle-avoidance success rate exceeding 90% [21]. Kumari and Hammady compared several YOLOv8 variants combined with audio feedback in a smart-glasses assistive system, evaluating the trade-off between detection accuracy and inference cost [22]. Ouyang et al. likewise investigated visual-aid algorithms for indoor navigation of visually impaired users based on deep-learning object detection [23]. These studies demonstrate that YOLO-family detectors, depth or distance estimation, and audio feedback are established building blocks in this domain; unlike the present study, however, none of them reports an explicit task-level parallelization of the detection and depth-estimation stages or an associated throughput/latency speedup analysis, which is the specific aspect examined here.

3. Methodology

With the development of information technologies, the problem of fast and efficient processing of large volumes of data has become increasingly important. In particular, in the fields of artificial intelligence, computer vision, Big Data, and real-time systems, the need for rapid execution of complex computational processes is growing. To address such challenges, parallel computing technologies play a crucial role [13]. Parallel computing is a computational approach based on executing complex tasks simultaneously using multiple processors or computational cores. This approach enables fast processing of large-scale data and significantly improves the overall system performance efficiency [14].

In conventional computing systems, tasks are typically executed sequentially, i.e., each task starts only after the previous one has finished, and all computational operations are performed step by step by a single processor. While this method may be sufficient for simple problems, when dealing with large volumes of data or complex algorithms, it leads to a significant increase in computation time [15].

Parallel computing addresses this problem by dividing a task into several smaller parts and executing them simultaneously on different processors as shown in Figure 3. As a result, the total computation time is significantly reduced.

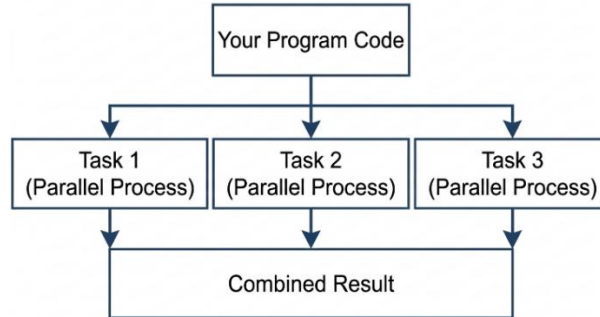


Figure 3. Parallel execution of tasks

In parallel computing systems, tasks are generally organized in two ways: data parallelism and task parallelism. In data parallelism, the same operation is applied simultaneously to different parts of a large dataset. For example, in image processing, different regions of an image can be analyzed in parallel on separate processors [16]. In task parallelism, different tasks are distributed across separate computational resources. For instance, one processor may handle data acquisition, another may process the data, while another performs result storage or delivers the output to the user.

To evaluate the efficiency of parallel computing, various mathematical models are used in the scientific literature. One of the most well-known is Amdahl's Law [17]. According to this law, the speedup of a parallel system is expressed by the following formula:

$$S = \frac{1}{(1-P) + \frac{P}{N}} \quad (5)$$

where:

- S - system speedup coefficient
- P - the fraction of the algorithm that can be executed in parallel
- N - the number of processors used

This formula is important for evaluating the efficiency of parallel computing systems. If a large portion of the algorithm can be executed in parallel, the overall system performance increases significantly.

In modern computer systems, various hardware architectures are used to implement parallel computing. The most common ones include multi-core central processing units (multi-core CPUs), graphics processing units (GPUs), and cluster computing systems [18]. Multi-core processors consist of several independent computational cores that can execute different tasks simultaneously. This technology is widely used in modern personal computers and servers. Graphics processing units, on the other hand, contain thousands of small computing cores and provide high performance for parallel processing of large-scale data. Therefore, GPU technologies are widely used in artificial intelligence, deep learning, and computer vision systems.

Parallel computing technologies are particularly important in the fields of computer vision and artificial intelligence. In image processing tasks, a large amount of pixel data is analyzed. For example, processing high-resolution video streams in real time requires significant computational power. With the help of parallel computing, different parts of an image can be analyzed simultaneously, which significantly increases the system's processing speed [18].

Modern algorithms widely used in computer vision, such as object detection and depth estimation models, also rely heavily on parallel computing technologies. Object detection algorithms identify different objects within an image and determine their locations. Since such algorithms are often based on deep neural networks, they require fast processing of large amounts of data. Parallel computing technologies accelerate these processes and make real-time application possible.

Parallel computing also enables the development of real-time navigation and assistive systems. For example, in modern assistive systems designed for visually impaired individuals, images captured by a camera are analyzed in real time, and users are warned about dangerous objects in the environment. In such systems, object detection,

distance estimation, and result delivery to the user must be performed simultaneously. Parallel computing technologies make it possible to execute these processes quickly and efficiently.

3.1. Hardware and implementation details

All experiments reported in this study were conducted on a single workstation equipped with an 11th-generation Intel Core i5 CPU (6 physical cores, 12 logical threads), 16 GB of RAM, and an NVIDIA GeForce RTX 2050 GPU with 4 GB of VRAM, running Windows 11. YOLOv8 and MiDaS inference were performed with CUDA acceleration on the GPU where available, while the task-level parallelization of the two models was implemented on the CPU side. YOLOv8 object detection and MiDaS depth estimation were treated as two independent computational tasks and executed concurrently using two execution units created with Python's built-in multiprocessing and threading modules, rather than being distributed across a larger cluster or a higher number of processes; consequently, only two of the six available CPU cores were exercised for task-level parallelism in the present implementation, and no multi-node cluster computing was used.

The code listing below shows a simplified representation of the parallel processing structure used to clarify the implementation. It illustrates the concurrent execution of YOLOv8 and MiDaS as two execution units communicating through a process-safe queue; implementation-specific preprocessing, post-processing, and voice-alert functions are omitted for readability.

Code-1:

```
import multiprocessing as mp

def yolo_worker(frame_queue, result_queue, weights):
    detector = YoloDetector(weights)
    while True:
        item = frame_queue.get()
        if item is None:
            break
        frame_id, frame = item
        detections = detector(frame)
        result_queue.put((frame_id, detections))

def midas_worker(frame_queue, result_queue):
    estimator = MidasEstimator()
    while True:
        item = frame_queue.get()
        if item is None:
            break

        frame_id, frame = item
        depth_map = estimator(frame)
        result_queue.put((frame_id, depth_map))

if __name__ == "__main__":
    ctx = mp.get_context("spawn")
    yolo_queue = ctx.Queue()
    midas_queue = ctx.Queue()
    yolo_results = ctx.Queue()
    midas_results = ctx.Queue()
    yolo_process = ctx.Process(
        target=yolo_worker,
        args=(yolo_queue, yolo_results, "yolov8n.pt")
    )
    midas_process = ctx.Process(
        target=midas_worker,
        args=(midas_queue, midas_results)
    )
    yolo_process.start()
    midas_process.start()

    # The same frame is sent to both processes
    frame_id = 0
    yolo_queue.put((frame_id, frame))
    midas_queue.put((frame_id, frame))

    # Results are combined by the main process
    # for distance estimation and risk classification.
    yolo_queue.put(None)
    midas_queue.put(None)
    yolo_process.join()
    midas_process.join()
```

The listing above is a didactic simplification intended to convey the two-execution-unit task-parallel structure; the authors' production implementation additionally includes frame synchronization, exception handling, model warm-up, and the cooldown-controlled voice-alert logic, and is planned for release as an open-source repository once the revision process is complete (see the Data and Code Availability section).

3.2. Dataset

The experimental evaluation used monocular video sequences containing common environmental objects and obstacles, including persons, chairs, vehicles, and stationary objects, captured in both indoor and outdoor scenes under normal lighting conditions. The evaluation focused on processing speed (FPS), relative depth estimation, and distance-based risk classification rather than on detection-accuracy benchmarking; consequently, standard object-detection accuracy metrics (e.g., mAP, precision/recall) for YOLOv8 on this data are not reported, and this is acknowledged as a limitation in the Discussion section.

4. Experimental tests

In Python programming, parallel computing can be implemented using libraries such as multiprocessing, threading, joblib, and Dask [19].

One of the main approaches to implementing parallel computing in Python is the multiprocessing module. This module divides a program into multiple independent processes, each of which is executed on a separate processor core. This approach is particularly effective for large-scale computational problems. The multiprocessing module allows bypassing the Global Interpreter Lock (GIL) limitation and thus enables efficient utilization of multi-core processors [20].

Another method of implementing parallel computing in Python is the threading module. Using the threading module, multiple threads are created within a program, allowing them to execute different tasks simultaneously. However, due to the Global Interpreter Lock (GIL) in the Python interpreter, the threading module is mainly effective for I/O-bound tasks [20].

In this study, we used the YOLO and MiDaS algorithms in Python, along with threading and multiprocessing tools for parallel computing as shown in Figure 4. The YOLO algorithm was used for object detection, while MiDaS was used for depth (distance) estimation.

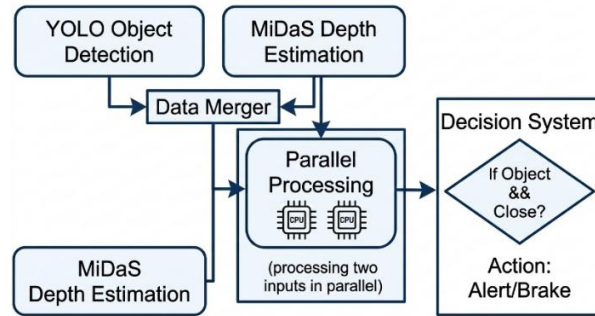


Figure 4. Parallel execution of YOLO and MiDaS algorithms

As a result of the parallel execution of these algorithms, the overall system processing time was significantly reduced (Table 1).

Table 1. Performance comparison

Processing Method	YOLO Detection Time (s)	MiDaS Depth Time (s)	Total Time (s)	FPS	Speedup
Sequential	0.105	0.181	0.286	3.5	1.0x
Parallel	0.090	0.170	0.210	5.5	1.57x

As can be seen from the program results, the use of parallel computing with these algorithms resulted in a 1.57× speedup. This value was computed from the measured processing rate,

$$S_{FPS} = \frac{FPS_{parallel}}{FPS_{sequential}} = \frac{5.5}{3.5} \approx 1.57x$$

i.e., from throughput rather than from a single-frame timing measurement. Reviewer comments requested that the speedup also be verified using the time data reported in Table 1. Computed directly from the measured end-to-end processing time, the latency speedup is:

$$S_{time} = \frac{T_{parallel}}{T_{sequential}} = \frac{0.286}{0.210} \approx 1.36x$$

The two figures differ because they measure different quantities: S_{FPS} reflects sustained throughput over a continuous stream of frames, in which the two execution units can overlap their workloads across successive frames (pipeline overlap), whereas S_{time} reflects the end-to-end latency of an individual frame. With two execution units employed ($N=2$), the ideal linear throughput speedup would be $2\times$. The measured throughput speedup of $1.57\times$ is therefore sub-linear, which is expected due to inter-process communication overhead, GPU resource sharing between the concurrently running models, and the fact that only two of the six available CPU cores were used for task-level parallelism in the current implementation. The lower latency speedup of $1.36\times$ reflects the different nature of the end-to-end latency metric and should not be interpreted as a direct measure of the pipeline's throughput scaling.

5. Results

In this study, the developed real-time “AI Blind Assist System” model integrates object detection (YOLOv8) and depth estimation (MiDaS) technologies to determine hazard levels and provide voice-based alerts. The experimental results were analyzed based on detected object distance, risk level, and system response time as shown in Table 2.

Table 2. Object detection result

Detected Object	Estimated Distance	Risk Level	Voice Alert
Person	1.5 m	High Risk	Yes
Chair	3.2 m	Medium Risk	No
Car	7.8 m	Low Risk	No

During the experiments, the system evaluated detected objects in terms of distance and hazard level as follows:

- An object located at a distance of 1.5 meters (a person) was classified as “high risk,” and the system immediately issued a voice alert.
- An object at 3.2 meters was classified as “medium risk” and was indicated only with a visual alert (colored bounding box).
- An object at 7.8 meters was classified as “low risk” and was only monitored by the system.

These results show that the system correctly segments hazard levels based on distance and makes appropriate decisions in real time as shown in Table 3.

Table 3. Risk classification

Distance Range	Risk Level	Bounding Box Color	System Action
< 2 m	High Risk	Red	Voice Warning
2 – 5 m	Medium Risk	Yellow	Visual Warning
> 5 m	Safe	Green	Monitoring Only

The system’s response mechanism to different hazard levels is based on three stages:

- High risk (< 2 m): the object is highlighted in red, and an immediate voice alert is triggered.
- Medium risk (2–5 m): the object is displayed in yellow, and a visual warning is provided.
- Low risk (> 5 m): the object is marked in green, and the system operates in monitoring mode only.

This mechanism allows users to receive fast and intuitive information about hazard levels and improves safety in real-time environments. The obtained results indicate that the developed system:

- performs distance-based segmentation effectively,
- detects hazard levels in real time,
- and, most importantly, provides both visual and audio feedback simultaneously.

In addition, the system’s voice alert mechanism is optimized using a “cooldown” approach, which prevents repetitive alerts for the same object. This significantly improves the user experience.

Overall, the results confirm that AI-based assistive systems can serve as an effective solution for real-time monitoring and support for visually impaired users.

6. Discussion and limitations

The “AI Blind Assist System” developed in this study demonstrated effective performance by integrating real-time object detection, distance estimation, and hazard level classification. Using the YOLOv8 model, objects were detected qualitatively reliably across the tested indoor and outdoor scenes, while the MiDaS model provided relative depth information, enabling the system to form spatial awareness; formal quantitative accuracy metrics (e.g., mAP, precision/recall) were not computed in this study and are noted as a limitation below.

The results show that the system successfully performs distance-based risk classification. In particular, immediate voice alerts for objects within 2 meters create an important safety mechanism for visually impaired users.

In addition, the application of parallel processing (YOLO + MiDaS) significantly improved system processing speed. This is a critical factor in real-time systems, as latency directly affects user safety. The obtained results indicate that the parallel approach provides a noticeable speed advantage compared to sequential execution.

Furthermore, the audio feedback mechanism based on the pyttsx3 library enables intuitive and rapid information delivery to the user. The implemented “cooldown” mechanism also improves usability by preventing redundant repeated alerts.

Although the system achieved promising results, it still has several limitations:

- The MiDaS model estimates only relative depth; therefore, the actual distance in meters is not precise. As a result, the formula $\text{distance} = 1000 / \text{depth}$ provides only an approximate estimation.
- Both YOLOv8 and MiDaS models may experience reduced accuracy under low-light or overly bright conditions.
- Even with parallel processing, the system may achieve low FPS when running on CPU alone; GPU acceleration is required for optimal performance.
- When objects partially overlap, the bounding box center may produce incorrect depth values.
- The pyttsx3 library is not the fastest text-to-speech engine for real-time applications, which may result in slight delays in voice alerts.
- Because MiDaS produces only a relative, unscaled depth map, the empirical distance formula ($\text{distance} = 1000 / \text{depth}$) was not validated against ground-truth measurements (e.g., LiDAR or stereo-calibrated distances); the reported distance values should therefore be interpreted as approximate rather than metrologically verified.
- The present evaluation was conducted on a laptop-class CPU/GPU workstation rather than on a resource-constrained embedded or wearable platform (e.g., Raspberry Pi or NVIDIA Jetson Nano); computational requirements and achievable frame rates on such devices were not measured in this study and are expected to be substantially lower without model compression (e.g., quantization or pruning).
- No real-world evaluation with visually impaired users or field navigation trials was carried out; the effectiveness and safety of the proposed system in real environments therefore remain to be established through dedicated user studies.

7. Conclusions and future research

This study presented a real-time “AI Blind Assist System” that integrates YOLOv8 for object detection and MiDaS for depth estimation, combined with parallel computing to improve performance. The experimental results show that the system can successfully detect objects, estimate their relative distance, classify risk levels, and provide timely audio-visual feedback for visually impaired users. The use of parallel processing significantly improved system speed, making real-time operation more efficient compared to sequential execution.

However, the system still has limitations in depth accuracy, lighting sensitivity, and real-time performance on CPU-based devices; in addition, no ground-truth validation of the distance estimates and no real-world evaluation with visually impaired users were performed in the present study, as detailed in the Discussion section. In future work, these issues can be addressed by improving depth estimation accuracy against ground-truth measurements, optimizing and compressing the models (e.g., quantization or pruning) for deployment on lightweight embedded or wearable platforms such as Raspberry Pi or NVIDIA Jetson Nano, and enhancing GPU-based parallel processing. Planned future work also includes structured evaluation with visually impaired users in real navigation scenarios to assess the effectiveness and safety of the system in practice. Additionally, more advanced and faster text-to-speech systems can be integrated to further reduce response delay and improve user experience.

Acknowledgments

The authors would like to express their sincere gratitude and appreciation to the supervisor Young-Im Cho for her support, comments, remarks and engagement over the period in which this manuscript was written. Moreover, the

authors would like to thank the editor and anonymous referees for the constructive comments in improving the contents and presentation of this paper.

Funding

None.

Data availability

The hardware configuration, dataset description, and implementation details necessary to reproduce the reported experiments are provided in the Methodology section. The source code implementing the parallel processing pipeline (task-level parallel execution of YOLOv8 and MiDaS, distance-based risk classification, and the cooldown-controlled voice-alert mechanism) is publicly available in a GitHub repository at <https://github.com/JavohirQurbonov/Scientific>

Conflict of interest

There is no conflict of interest to disclose.

Author contributions

Mukhiddin Mukhiddinov: Conceptualization, Methodology, Formal analysis, Validation, Resources, Supervision, Project administration, Funding acquisition, Writing – original draft, Writing – review & editing.
Javokhir Kurbonov: Data curation, Software, Investigation, Formal analysis, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of using AI tools

The authors declare that they have not used any type of generative artificial intelligence for the writing of this manuscript, nor for the creation of images, graphics, tables, or their corresponding captions.

References

- [1] World Health Organization, World Report on Vision. Geneva, Switzerland: World Health Organization, 2019.
- [2] K. Potdar, C. Pai, and S. Akolkar, “A convolutional neural network based live object recognition system as blind aid,” *Int. J. Comput. Appl.*, 2018.
- [3] S. Chandra, U. Sharma, and D. Khilnani, “Computer vision and depth sensor-powered smart cane for real-time obstacle detection,” 2025.
- [4] M. J. Shafiee, B. Chywl, F. Li, and A. Wong, “Fast YOLO: A fast You Only Look Once system for real-time embedded object detection in video,” 2017. 10.15353/vsnl.v3i1.171
- [5] R. Szeliski, *Computer Vision: Algorithms and Applications*. Cham, Switzerland: Springer, 2022. 10.1007/978-3-030-34372-9
- [6] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Pearson, 2018.
- [7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
- [8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 28 2015. 10.1038/nature14539
- [9] R. Girshick, “Rich feature hierarchies for accurate object detection,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [10] J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” *arXiv preprint*, 2018.
- [11] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” *arXiv preprint*, 2020.
- [12] R. Ranftl, A. Bochkovskiy, and V. Koltun, “MiDaS: Robust monocular depth estimation,” in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [13] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*. Pearson Education, 2003.
- [14] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*. New York, NY, USA: McGraw-Hill, 2004.
- [15] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*. Pearson, 2015.
- [16] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Cambridge, MA, USA: Morgan Kaufmann, 2019.
- [17] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” in *Proc. AFIPS Spring Joint Computer Conf.*, 1967. 10.1145/1465482.1465560
- [18] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors*. Cambridge, MA, USA: Morgan Kaufmann, 2016.
- [19] Python Software Foundation, “Python documentation: Concurrent execution and threading.” [Online]. Available: <https://docs.python.org>
- [20] Python Software Foundation, “Multiprocessing module documentation.” [Online]. Available: <https://docs.python.org/3/library/multiprocessing.html>

- [21] Y. Zhuang, “Intelligent obstacle avoidance method for blind travel based on an improved YOLO algorithm and binocular vision,” *Promet – Traffic&Transportation*, vol. 38, no. 5, pp. 1192-1205, 2026. 10.7307/ptt.v38i5.1295
- [22] P. Kumari and R. Hammady, “Assisting blind people with AI and audio using smart glasses: System design with YOLOv8 variants comparisons,” *Multimedia Syst.*, vol. 32, no. 1, p. 73, 2026., doi:
- [23] Y. Ouyang, R. Zhang, Y. Liu, and Y. Peng, “Research on indoor visual aid algorithms for visually impaired people,” *Laser Technology*, vol. 49, no. 2, pp. 166–174, 2025.



All open access articles published in Transactions on Computational Modelling and Intelligent Systems (<http://tcmis.org>) are distributed under the terms of the CC BY-NC 4.0 license (Creative Commons Attribution Non-Commercial 4.0 International Public License as currently displayed at <http://creativecommons.org/licenses/by-nc/4.0/legalcode>) which permits unrestricted use, distribution, and reproduction in any medium, for non-commercial purposes, provided the original work is properly cited.