AI-BASED DISASTER VICTIM DETECTION AND RESCUE ASSISTANT SYSTEM USING YOLOV8

Varun Gowda S V[1], BHM Sharath Kumar Shastri[2], Tejas R[3], Sangam Gupta H T[4], Dr. Sridhar C S[5]

[1] *Department of Computer Science and Engineering (AI & ML), Akash Institute of Engineering and Technology, Devanahalli, India*

[2] *Department of Computer Science and Engineering (AI & ML), Akash Institute of Engineering and Technology, Devanahalli, India*

[3] *Department of Computer Science and Engineering (AI & ML), Akash Institute of Engineering and Technology, Devanahalli, India*

[4] *Department of Computer Science and Engineering (AI & ML), Akash Institute of Engineering and Technology, Devanahalli, India*

[5] *HOD, Department of Computer Science and Engineering (AI & ML), Akash Institute of Engineering and Technology, Devanahalli, India*

Abstract — Searching for survivors after earthquakes, floods and building collapses is slow and dangerous when it depends on human observers. This paper presents a prototype disaster-response assistant that applies the YOLOv8 object detector to live camera video. Frames from a laptop webcam or an IP-streamed smartphone camera are passed to a custom-trained model with one class, "victim", and only detections with confidence above 0.6 are accepted. For every accepted detection the system draws a bounding box, shows the confidence score, and estimates the camera-to-person distance from the bounding-box height using a pinhole-camera proportion with an assumed height of 1.7 m and a focal-length constant of 350. Approximate latitude and longitude are obtained from an IP-based geolocation service and shown on screen, and an audio alarm sounds while a valid detection is present and stops when it is not. The paper explains the architecture, the governing equations and the test plan. Quantitative detection metrics and completed test outcomes are not available for this prototype, and this is stated openly. The system is intended as a low-cost decision-support aid for rescue teams, not as a replacement for established search-and-rescue procedures.

Index Terms — YOLOv8, Disaster Response, Victim Detection, Search and Rescue, Distance Estimation, Real-Time Computer Vision, Alert System.

I. INTRODUCTION

Natural disasters such as earthquakes, floods, landslides and building collapses leave people trapped, injured or missing. In the hours that follow, the speed at which victims are located strongly influences their chances of survival. Conventional search depends on rescuers who either walk into unstable areas or watch camera footage for long periods. This is time-consuming, tiring, hazardous and prone to missed sightings, especially when visibility is poor or several victims are present.

Computer vision and deep learning can take over part of this visual search. Single-stage detectors in the YOLO family treat detection as a single regression problem and predict bounding boxes and class probabilities in one network pass, which is why they suit live video [1]. YOLOv8, developed by Ultralytics, continues this line with an anchor-free, decoupled detection head and a Python interface that works directly with OpenCV [2].

Detection alone, however, tells a rescuer little about where to go. A usable aid should also give a sense of range, some location context, and a signal that attracts attention without constant screen watching. This work therefore combines four functions in one application: real-time detection, distance estimation from bounding-box geometry, display of approximate coordinates, and an audible alert linked to the detection state. The contribution is an integrated, inexpensive pipeline built from open-source tools and commodity hardware, rather than a new detection architecture.

Research Objectives

- To detect possible victims in live camera video using YOLOv8.
- To display a bounding box and confidence score for each accepted detection.
- To estimate the approximate distance between the camera and the detected person.
- To display latitude and longitude associated with the monitoring setup.
- To activate an audio alarm on detection and deactivate it when no victim is present.
- To document the testing plan used to verify the individual modules and the complete system.

II. RELATED WORK

Research relevant to this project spans object detection, automated person detection for search and rescue, and training strategies for disaster imagery.

A. Real-Time Object Detection with YOLO. Redmon et al. reformulated detection as a single regression from image pixels to boxes and class probabilities, giving the speed needed for real-time processing [1]. YOLOv8 builds on that idea. Its backbone uses convolutional layers, C2f modules and Spatial Pyramid Pooling Fast, its neck fuses multi-scale features, and its anchor-free head predicts boxes and classes separately at three scales. Training uses CIoU and Distribution Focal Loss for box regression and binary cross-entropy for classification [2].

B. Person Detection for Search and Rescue. Gotovac et al. compared human reviewers with a machine-learning detector on imagery captured by unmanned aerial systems. Their study supports automated detection as a way to reduce manual image analysis, while noting that performance depends on image quality, lighting and viewing conditions [3].

C. Detection of Disaster Victims. Disaster scenes pose a data problem: victims are often partly buried or hidden, and

real training images are scarce. Zhang et al. addressed this by training a detection network on harmonious composite images, which improved recognition of partially visible victims [4].

D. Observed Gaps. The project documents also reviewed studies using thermal imagery, drones with GPS navigation, YOLOv5 people counting, YOLOv3 action recognition and YOLOv8 transfer learning. Many of these depend on specialised hardware such as drones or thermal cameras, and few place detection, distance and location cues together in a lightweight tool. Computational cost is another recurring concern. These observations motivate the present work.

E. Other Approaches Reviewed in the Project Report. The project report surveyed a broader set of UAV and drone-based studies, summarised in Table I. They share a common pattern: strong detection results are obtained by adding hardware such as thermal cameras, drones or autonomous navigation, and each study names this dependence, or its sensitivity to image quality and datasets, as a limitation. Full bibliographic details of these studies are given in the project report and are not repeated in the reference list of this paper.

TABLE I. APPROACHES REVIEWED IN THE PROJECT REPORT

Approach (year)	Technology	Limitation noted
UAS person detection, humans vs. ML (2020)	UAS imagery, machine learning	Image quality, lighting, viewing conditions
UAV thermal survivor detection (2020)	YOLOv3, MobileNet-based models, thermal imagery	Needs thermal cameras and computing resources
UAV search-and-rescue case study (2021)	UAV, smartphone-based ML platform	Depends on camera quality and processing power
Drone-based autonomous human identification (2021)	Custom TensorFlow network, GPS	Needs drone hardware and autonomous navigation
UAV real-time survivor detection (2021)	YOLOv3, thermal image datasets	Needs thermal equipment and onboard processing
Disaster victim detection, composite images (2022)	Deep learning, UAV images	Needs specialised training data preparation
UAV people counter (2022)	YOLOv5, thermal camera	Depends on UAV and thermal camera availability
Human detection and action recognition (2023)	YOLOv3, drone video	Affected by poor visibility, complex backgrounds
UAV survivor detection with transfer learning (2024)	YOLOv8, real and synthetic data	Depends on dataset quality and diversity
Indoor survivor detection with low-cost MAVs (2025)	YOLOv8x, thermal imaging, mapping	Needs thermal cameras and autonomous aerial systems

III. RESEARCH GAP AND PROBLEM STATEMENT

Existing studies offer strong individual solutions: detectors for aerial imagery, thermal-camera pipelines, and training methods for occluded victims. What is less common is a simple, low-cost tool that runs on ordinary hardware and presents detection, an approximate distance, location context and an alert in a single interface. The research gap addressed here is the integration of these functions into one real-time workflow that a small team could deploy without specialised equipment.

Problem Statement. In a conventional setup the pipeline is simply: Camera → Human Observer → Decision. Detection depends entirely on sustained human attention. The question addressed in this work is: *How can a YOLOv8-based system detect victims in live video, indicate an approximate distance and location context, and alert the operator automatically, using commodity hardware and open-source software?*

Existing Practice and Its Drawbacks. Existing victim-search practice relies mainly on manual search and conventional techniques, with cameras, drones and physical inspection used by rescue teams. Basic image-processing and object-detection approaches have been used, but they struggle under poor lighting, complex backgrounds and debris, and continuous manual monitoring adds to the workload of rescuers. Traditional setups seldom bring real-time detection, approximate distance, location information and automatic alerts together on one platform. Table II contrasts this with the proposed system.

IV. PROPOSED SYSTEM

The proposed system follows the sequence: Camera Input → Frame Capture → Preprocessing → YOLOv8 Detection → Confidence Filtering → (Distance Estimation, Location Overlay, Alarm Control) → Display Output. It contains six functional components: video acquisition, model loading and detection, distance estimation, location identification, audio alarm, and visual output with resource management.

A. Video Acquisition. The application reads frames from a laptop webcam or from a smartphone camera served over an IP stream and opened with OpenCV. Frames are read one at a time. If a frame cannot be read, the acquisition loop ends.

B. Model Loading. The trained weights file (best.pt) is loaded with the Ultralytics YOLO library. The dataset configuration defines train, validation and test folders and a single class, victim (nc: 1).

C. Victim Detection. Each frame is passed to the model, and each candidate detection is kept only if its confidence satisfies

$$c_i > \tau, \quad \tau = 0.6 \tag{1}$$

A per-frame flag, victim_detected, is reset at the start of every frame and set to true when at least one detection passes Eq. (1). The flag therefore reflects the current frame only.

D. Distance Estimation. Using the pinhole-camera similar-triangles relation, the distance D is computed from the bounding-box height h, an assumed real-world height H and a focal-length constant f:

$$h = y_2 - y_1 \tag{2}$$

$$D = (H \times f) / h \tag{3}$$

The implementation uses H = 1.7 m and f = 350, and sets D = 0 when $h \leq 0$. With these constants D = 595/h metres. As an illustration only (not a measured result), a box 200 pixels tall gives about 3.0 m. Table IV tabulates Eq. (3) for several box heights to show how the estimate behaves.

The table shows the inverse relationship: the estimated distance falls as the person occupies more of the frame. Because the relationship is inverse, a given pixel error in the box height produces a larger distance error for far-away people than for near ones.

E. Location Identification. A single request is sent to the IP-API geolocation service, and the lat and lon fields of the response are stored and overlaid on every frame. If the request raises an exception, both values are set to "Unknown". The result is derived from the network's public IP address, so it indicates approximate network location, not the exact position of the camera or the victim.

F. Alarm and Alert Mechanism. An audio file (alarm.wav) is loaded with the Pygame mixer if it exists. If a valid detection is present and the alarm is not already playing, playback starts. If no valid detection is present, playback stops. Checking the busy state prevents the sound from restarting on every frame.

G. Visual Output and Resource Release. OpenCV draws a green bounding box and a label of the form "Victim conf | distance m" and displays the annotated frame in a window titled "Smart Rescue AI System". Pressing ESC ends the loop, releases the camera, closes the windows and stops the audio.

H. Module Summary. The implementation is divided into the modules listed in Table III. A small launcher offers two modes, Victim Detection and a separate Security Monitoring module, and runs the selected program; only the first is within the scope of this paper.

TABLE III. IMPLEMENTED MODULES

Module	Library	Function
Mode selection	Python (os)	Launches the chosen mode
Video acquisition	OpenCV	Connects to the camera stream and reads frames
Model loading	Ultralytics YOLO	Loads best.pt
Dataset configuration	YAML file	Paths of train/valid/test sets, one class "victim"
Victim detection	Ultralytics YOLO	Inference and confidence filtering (0.6)
Distance estimation	Python	Applies Eq. (3) to box height
Location identification	Requests	Queries ip-api.com for lat and lon
Visual output	OpenCV	Draws box, label and coordinates; shows window
Audio alarm	Pygame	Plays or stops alarm.wav
Resource management	OpenCV, Pygame	Releases camera, windows and audio on ESC

I. Complete Algorithm. The steps executed by the application are:

1. Display the modes and select Victim Detection.
2. Load the trained YOLOv8 model and the alarm file.
3. Open the camera stream and request the latitude and longitude once; set both to "Unknown" if the request fails.
4. Read the next frame; if it cannot be read, leave the loop.
5. Run YOLOv8 on the frame and reset victim_detected to false.
6. For each detection with confidence above 0.6, set victim_detected to true, compute the box height and the distance from Eq. (3), draw the box and the label.
7. Overlay the latitude and longitude on the frame.
8. If victim_detected is true and the alarm is not playing, start it; if it is false, stop the alarm.
9. Display the annotated frame.
10. If ESC is pressed, release the camera, close the windows, stop the audio and end; otherwise return to step 4.

J. Data Flow and Design Views. The project report describes the system with a flowchart, use-case and sequence diagrams, data-flow diagrams at three levels, and activity diagrams. At Level 0 the camera supplies frames and the operator starts, stops and selects the camera source, while the system returns detections, distance, location and the alarm. At Level 1 the processes are capture and preprocessing, detection, calculation of box, confidence and distance, location retrieval, alert and display. Level 2 expands the detection stage, in which the model weights are read from a data store and the results feed the alarm, location and display processes. The diagrams themselves are in the project report.

The overall process can be summarised as: Capture → Detect → Filter → Estimate → Locate → Alert → Display.

V. EXPERIMENTAL METHODOLOGY

A. Development Environment. The prototype was written in Python on Windows 10/11 using Visual Studio Code. The libraries were Ultralytics YOLO (detection), OpenCV (capture, drawing, display), PyTorch (model execution), NumPy, Requests (location lookup) and Pygame (audio). The recommended hardware is an Intel Core i5 or equivalent, 8 GB RAM and 256 GB SSD storage. The camera source is a laptop webcam or a smartphone camera application streaming over IP.

B. Dataset and Training. A human/victim image dataset was annotated with bounding boxes and split into training, validation and test sets. The trained weights were saved as best.pt. Input frames are resized to 640 × 640 pixels before inference. The documents do not give the dataset's name or source, the image or instance counts, the annotation tool, any augmentation, or the training hyperparameters (epochs, batch size, pretrained initialisation). These details are not reported here and must be added by the authors.

C. Test Plan. Testing was planned at three levels, summarised in Table V, and the individual system-level scenarios are listed in Table VI.

TABLE V. TEST PLAN SUMMARY

Level	Cases	Coverage
Unit testing	13 (UT-01 to UT-13)	Camera connection, frame acquisition, camera failure, model loading, victim and no-victim detection, confidence threshold, distance, location success and failure, alarm start and stop, resource release
Integration testing	9 (IT-01 to IT-09)	Camera–model, detection–distance, detection–display, detection–audio, location–display, camera and audio release on termination
System testing	14 (ST-01 to ST-14)	Start-up, mode selection, detection with and without a victim, distance and location display, audio, ESC termination, camera/model/location/alarm-file failures, continuous monitoring, repeated detection

D. Evaluation Metrics. The standard measures for a detector of this kind are precision, recall, F1-score and mean average precision, together with frame rate and distance error against measured ground truth. The supplied documents do not report any of these measures.

E. Functional Requirements. The system is required to: capture live video from a webcam or an IP-streamed phone camera; process frames with OpenCV; detect people with YOLOv8; draw bounding boxes; display a confidence score; estimate approximate distance from box height; retrieve latitude and longitude through an IP geolocation service; raise an audio alert on detection; and present the processed video, confidence, distance and location in a simple interface.

F. Non-Functional Requirements. The system should respond with minimum delay (performance), identify persons reliably although accuracy may vary with lighting, camera quality, distance and background (accuracy), operate consistently during monitoring (reliability), present results clearly to non-expert users (usability), make efficient use of CPU, RAM and GPU (efficiency), and allow models and libraries to be updated (maintainability). None of these qualities has been quantitatively measured in the available documentation.

VI. RESULTS AND DISCUSSION

A. Observed System Behaviour. The prototype opens a single window showing the live feed. For each accepted detection it overlays a green bounding box with the confidence score and estimated distance, and it shows the latitude and longitude on the frame. The alarm plays while a valid detection is present and stops when none is. The project presentation includes screenshots of the IP-camera connection, real-time detection and location display. In the supplied report, Figures 8.1, 8.3, 8.4 and 8.5 are still placeholders, so the authors must insert their own screenshots here.

B. Test Outcomes. Every test case in the report has "To be recorded" as its actual result and "Pending" as its status. No pass/fail outcome can therefore be claimed (Table VII).

TABLE VII. RESULTS STATUS

Measure	Status in supplied documents
Precision / Recall / F1 / mAP	Not reported
Confusion matrix, training curves	Not reported
Frames per second, latency	Not reported
Distance error vs. ground truth	Not reported
Unit, integration and system tests	Pending (not recorded)

C. Discussion. The value of the prototype lies in integration. Detection, a range cue, a location cue and an audible alert are delivered together using open-source tools and ordinary hardware, in contrast to approaches that rely on drones or thermal cameras [3], [4]. The supporting modules are approximate by construction. The distance in Eq. (3) assumes an upright 1.7 m person and an uncalibrated focal-length constant, so a trapped, lying or partly hidden victim will not match the assumption. The coordinates come from IP geolocation, which needs internet access and may be unavailable in a disaster area. The fixed 0.6 threshold has not been tested for sensitivity, and because the alarm follows the single-frame flag it will stop in any frame where the detector briefly misses the person. No frame-to-frame smoothing is described. These points should be treated as hypotheses to test, not as findings.

D. Intended Applications and Impact. The project documents list the intended users as disaster-response teams, military rescue units and emergency services, and the intended benefits as faster rescue, reduced human effort and risk, and better detection. These are design goals. Without measured results they cannot yet be presented as demonstrated outcomes. The listed deliverables are the trained detection model, the real-time detection system, the distance-estimation module, the location feature, the alarm system, a working camera demonstration and the project documentation.

VII. ABLATION STUDY (PLANNED)

No ablation experiments were carried out. An ablation study would help show the contribution of each design choice. Suggested experiments are listed in Table VIII.

VIII. LIMITATIONS

1. No quantitative detection metrics and no completed test outcomes are available.
2. The dataset's source, size and training configuration are undocumented.
3. Distance estimation relies on an assumed person height and an uncalibrated focal-length constant.
4. Location is derived from IP geolocation, not GPS, is requested once, and needs an internet connection.
5. The detection threshold and per-frame alarm logic have not been evaluated.
6. Performance in smoke, darkness, debris, water or heavy occlusion has not been tested.
7. Alerts reach only the local operator. The emergency-message and rescue-status features shown in the report's design diagrams are not part of the described implementation.

IX. CONCLUSION

This paper presented a YOLOv8-based prototype that detects possible victims in live video, estimates an approximate distance from bounding-box height, displays network-derived coordinates and drives an audio alarm from the detection state. The modules run together in one real-time application on commodity hardware. The contribution is a practical integration rather than a new algorithm. Because the distance and location components are approximate and the available documentation contains no measured accuracy or completed test results, the system should be regarded as a proof of concept that still requires validation, and as a support tool for, not a substitute for, professional rescue teams.

X. FUTURE WORK

1. Report precision, recall, F1 and mAP on a documented held-out test set, with frame rate and latency.
2. Train on larger and more varied disaster data, including rubble, flooding, low light and partial occlusion.
3. Calibrate the camera and validate distance estimates against measured ground truth.
4. Replace IP geolocation with device GPS and add live map integration.
5. Add multi-frame smoothing to stabilise detection and alarm behaviour.
6. Support drone-mounted and thermal cameras.
7. Send automatic SMS, e-mail or app alerts to rescue teams.

REFERENCES

[1] Ultralytics, “YOLOv8 Documentation,” 2023. Available: https://docs.ultralytics.com/models/yolov8/

[2] G. Jocher, A. Chaurasia, and J. Qiu, “Ultralytics YOLOv8,” Ultralytics, 2023. Available: https://github.com/ultralytics/ultralytics

[3] G. Jocher et al., “Ultralytics YOLO,” GitHub Repository. Available: https://github.com/ultralytics/ultralytics

[4] OpenCV, “OpenCV Documentation,” Open Source Computer Vision Library. Available: https://docs.opencv.org/

[5] PyTorch, “PyTorch Documentation,” Meta AI. Available: https://docs.pytorch.org/

[6] Python Software Foundation, “Python 3 Documentation.” Available: https://docs.python.org/3/

[7] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” arXiv preprint arXiv:2004.10934, 2020.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788.

[9] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 7263–7271.

[10] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” arXiv preprint arXiv:1804.02767, 2018.

[11] GitHub, “RescueVision-YOLOv8: AI Based Disaster Victim Detection and Rescue Assistant System,” Source Code Repository. Available: https://github.com/varungowda18/RescueVision-YOLOv8

[12] IP-API, “IP Geolocation API Documentation.” Available: https://ip-api.com/docs/