

ARTIFICIAL INTELLIGENCE (AI)

YOLO-Based Real-Time Object Detection Using Computer Vision for Robotics

Aafareen Sikandar Shaikh, Archana Nilesh Pawar

Department of Artificial Intelligence and Machine Learning, Navashyadri College of Engineering, Pune, Maharashtra, India

Published October 2026

Abstract

Robots operating in dynamic environments require rapid and reliable perception to identify people, tools, obstacles, and other objects before making movement or manipulation decisions. Conventional vision pipelines may require separate stages for region proposal, classification, and localization, which can increase computational cost and latency. You Only Look Once (YOLO) formulates object detection as a single-network prediction task and has become a practical family of detectors for real-time computer-vision applications. This paper presents a research-oriented framework for integrating a lightweight YOLO detector with a robot-mounted camera. The framework covers image acquisition, preprocessing, object detection, confidence filtering, coordinate interpretation, and communication with a robot controller. It also discusses model selection, deployment constraints, evaluation metrics, safety considerations, and a reproducible testing plan. Rather than reporting unperformed experiments, the paper defines how a prototype should be evaluated using precision, recall, mean average precision, inference latency, frame rate, and end-to-end response time. The analysis indicates that compact YOLO variants are suitable starting points for resource-constrained robotic platforms, while reliable navigation requires additional depth or distance sensing and a safety controller. The proposed approach can support obstacle awareness, object following, pick-and-place assistance, and indoor mobile-robot perception.

Keywords: computer vision; object detection; robotics; real-time inference; YOLO

1 Introduction

Robotics is increasingly used in manufacturing, agriculture, logistics, healthcare, and service environments. To work safely and effectively, a robot must perceive its surroundings and distinguish relevant objects from background regions. A camera provides rich visual information, but raw images do not directly tell a robot which objects are present or where they appear. Computer-vision algorithms convert image data into structured information that can be used by a control system.

Object detection identifies both the class and location of objects in an image. A typical detector returns a class label, a confidence score, and a bounding box for each detected instance. The YOLO approach introduced by Redmon et al. performs detection in a single forward pass of a neural network, making speed a central design objective [1]. Later implementations, including the Ultralytics YOLO family, provide model variants and workflows for prediction, validation, training, and deployment [2, 3].

In robotics, the value of detection is not limited to drawing boxes around objects. The detections must be delivered quickly enough to influence motion, remain sufficiently reliable under changing lighting and viewpoints, and be translated into useful robot actions. This paper outlines a practical YOLO-

based perception pipeline and highlights the distinction between image-space object detection and full three-dimensional obstacle avoidance.

2 Problem statement and objectives

A robot that relies on manual observation or simple fixed-threshold vision may fail when objects vary in shape, colour, size, or background. A robotics-oriented detector should recognize multiple object categories in video while maintaining acceptable latency on the selected hardware. The main problem considered here is how to integrate a real-time YOLO detector into a camera-based robotic perception system without treating detections alone as a complete navigation solution.

The objectives of the proposed framework are:

1. detect selected object classes from a live camera feed;
2. return bounding boxes, class names, and confidence values for each frame;
3. provide detection outputs to a robot-control layer with minimal delay;
4. define measurable evaluation criteria for accuracy, speed, and robustness; and
5. identify limitations and safety measures required before real-world deployment.

3 Related work

3.1 Yolo as a single-stage detector

Redmon et al. framed detection as direct prediction of bounding boxes and class probabilities from the full image. Their original YOLO system demonstrated high processing speed compared with contemporary detection approaches, while also noting localization errors and limitations for small objects [1]. This design established the core speed-oriented principle behind the YOLO family.

3.2 Modern YOLO implementations

Current YOLO implementations offer model sizes that trade computational demand against accuracy. Ultralytics documentation describes a workflow for inference, training, validation, and export, and lists compact variants intended for faster processing [2, 3]. A small model can be a sensible starting point for a robot with limited compute, but the best model depends on target classes, image resolution, accelerator availability, and the required response time.

3.3 Yolo in mobile robotics

Recent work has explored YOLO-based obstacle detection and mobile-robot navigation. For example, a 2024 study combined YOLO object recognition with a control strategy for mobile-robot positioning and obstacle avoidance [4]. Other work has specifically investigated lightweight YOLO-based collision avoidance in unstructured environments [5]. These studies support the relevance of real-time detection for robotics, but they also show that perception must be integrated with a motion-planning or control method.

The research gap addressed by this paper is practical integration: a clear, reproducible pipeline that connects a camera, a YOLO model, detection post-processing, and a robot controller, while explicitly measuring end-to-end performance and acknowledging the limitations of monocular image detections.

4 Proposed methodology

The proposed system uses a camera mounted on a mobile robot or robotic platform. Video frames are passed to a pretrained or task-trained YOLO model. The detector identifies objects of interest and returns their bounding boxes, class labels, and confidence scores. A post-processing module filters low-confidence detections and converts accepted detections into simple robot-level cues. The controller then decides whether to continue, slow down, stop, turn, follow an object, or request a higher-level decision.

4.1 System architecture

Camera / video stream → frame resizing and normalization → YOLO inference → confidence filtering and non-maximum suppression (where used by the implementation) → object-to-action logic → robot controller → motors or actuator interface. A separate distance sensor, depth camera, or LiDAR may be used to estimate range and improve collision avoidance.

4.2 Image acquisition and preprocessing

The camera captures frames at a chosen resolution and frame rate. Frames are resized to the model input size and converted to the expected tensor format. Stable camera mounting, adequate exposure, and a suitable field of view are important because motion blur, glare, darkness, and occlusion can reduce detection quality.

4.3 Yolo inference

A compact pretrained model can first be tested on common classes. If the robot must detect domain-specific objects not represented in the pretrained class set, a custom labelled dataset should be collected and used for fine-tuning. The dataset should contain different distances, viewing angles, lighting conditions, backgrounds, and partially occluded examples. Training, validation, and test images must be separated to avoid data leakage.

4.4 Detection filtering

For each predicted object, the system receives a category, confidence score, and bounding-box coordinates. Detections below a selected confidence threshold are discarded. The threshold should be tuned on validation data: a very high threshold can miss hazards, while a low threshold may create false alarms. Bounding boxes can be grouped into image regions such as left, centre, and right to create simple directional cues.

4.5 Robot-level decision logic

If a relevant object appears in a predefined region, the controller can issue a high-level command such as 'stop and inspect' or 'turn left'. These rules are only a demonstration strategy; they do not calculate physical distance from image position alone. For safe movement, the controller should combine detections with range measurements, speed limits, emergency-stop logic, and fail-safe behaviour when camera frames or model outputs become stale.

5 Implementation plan

A prototype can be implemented in Python using OpenCV for camera access and display, and an Ultralytics YOLO model for inference [2, 3]. The software should separate the perception module from

the robot-control interface so that the detector can be tested without moving hardware. The initial prototype should run on recorded video or a stationary camera before it is connected to a moving robot.

Recommended implementation stages are:

1. install Python, OpenCV, and the selected YOLO package in a virtual environment;
2. run inference on sample images to verify class labels and bounding boxes;
3. test a live webcam feed and record processing time per frame;
4. select the object classes relevant to the robot's task;
5. fine-tune on a custom dataset if pretrained classes are insufficient;
6. connect detections to a simulated controller or low-risk robot test; and
7. repeat tests under varied lighting, distance, background, and object motion.

The robot interface may use serial communication, ROS/ROS 2 topics, or a platform-specific API. The communication layer should include timestamps and validity checks. A controller should not continue a previous movement command indefinitely if the vision process stops responding.

6 Performance evaluation

A valid experimental evaluation should measure detection quality and system responsiveness on the actual target hardware. Results should be recorded rather than inferred from published benchmark values, because performance varies with model version, input size, hardware, software configuration, and scene complexity.

Precision and recall are defined in Equations (1) and (2):

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

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

where TP denotes true positives, FP false positives and FN false negatives. Mean average precision (mAP) summarizes precision-recall performance across classes and, depending on the definition used, intersection-over-union thresholds. Frames per second (FPS) measures processed frames per second, while inference latency measures the time taken for one model prediction. End-to-end latency should include frame acquisition, inference, post-processing, communication and controller response.

A suggested test matrix includes: (i) static objects in good lighting, (ii) moving objects, (iii) partial occlusion, (iv) low-light or backlit scenes, (v) multiple objects in one frame, and (vi) target-hardware operation under sustained load. Report the model name and version, input resolution, confidence threshold, hardware, dataset size, class list, and number of test samples. Include average and worst-case latency where possible.

No new hardware experiment or quantitative benchmark is claimed in this paper. The proposed test matrix is intended to make a future prototype evaluation transparent and reproducible. Once tests are completed, measured results can be inserted into a table with columns for scenario, precision, recall, mAP, FPS, and end-to-end latency.

7 Discussion

The main benefit of YOLO for robotics is that object detection can be performed within a single neural-network inference stage, which can reduce pipeline complexity and support responsive perception [1]. Smaller variants may be more suitable for embedded or low-power platforms, whereas larger variants may provide better accuracy at a higher computational cost [2]. Model selection should therefore be based on the task's safety requirements and measured hardware performance rather than on model size alone.

Several limitations remain. First, bounding boxes describe image regions, not exact three-dimensional positions. Second, a detector may fail on unfamiliar objects, unusual weather, motion blur, or heavy occlusion. Third, high FPS does not automatically guarantee safe robot behaviour: camera delay, communication delay, controller tuning, and braking distance also matter. Fourth, models trained on general datasets may not perform well for a specialised industrial or agricultural environment without additional data.

For practical deployment, YOLO should be treated as one component of a perception-and-control system. A depth sensor or range sensor can supply distance information; tracking can help maintain object identity over time; and a safety controller should override motion commands when an obstacle is too close or the perception system becomes unreliable.

8 Applications

The framework can be adapted to: (1) mobile-robot obstacle awareness, (2) object following and human-aware service robots, (3) robotic pick-and-place where objects are visually localized, (4) warehouse robots that identify boxes or tools, and (5) agricultural robots that recognize plants, produce, or visible obstacles. Each application requires a task-specific class list and testing in its intended environment. Applications involving people or safety-critical movement require stronger validation and independent protective systems.

9 Conclusion

This paper presented a YOLO-based framework for real-time object detection using computer vision in robotics. The proposed pipeline connects camera acquisition, preprocessing, YOLO inference, detection filtering, and robot-level decision logic. It also defines an evaluation procedure using precision, recall, mAP, FPS, and end-to-end latency. YOLO provides a practical starting point for responsive visual perception, but image-space detections alone are insufficient for reliable three-dimensional navigation. Future work should implement the framework on a selected robot, evaluate it on a representative dataset, compare compact model variants, and integrate depth sensing and fail-safe control.

References

- [1] Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*; 2016. p. 779–788. <https://doi.org/10.1109/CVPR.2016.91>
- [2] Ultralytics. *Object detection with Ultralytics YOLO*. Ultralytics documentation. <https://docs.ultralytics.com/tasks/detect/> (accessed 16 October 2026).

-
- [3] Ultralytics. *Explore Ultralytics YOLOv8*. Ultralytics documentation. <https://docs.ultralytics.com/models/yolov8/> (accessed 16 October 2026).
 - [4] Top A, Gökbulut M. Real-time deep learning-based position control of a mobile robot. *Engineering Applications of Artificial Intelligence*. 2024;138(Part A):109373. <https://doi.org/10.1016/j.engappai.2024.109373>
 - [5] Abood SH, Al-Khafaji HMH, Al-Khafaji MMH. Enhancing collision avoidance in mobile robots using YOLOv5: A lightweight approach for unstructured environments. *Journal of Robotics and Control*. 2025;6(2):769–778. <https://doi.org/10.18196/jrc.v6i2.25856>