RESEARCH PROGRESS IN MECHANICAL AND MANUFACTURING ENGINEERING
e-ISSN: 2773-4765

RPMME

Vol. 7 No. 1 (2026) 188-194
https://publisher.uthm.edu.my/periodicals/index.php/rpmme

You Only Look Once (YOLO) – Based Human Detection for a Scuttle Bot

Ahmad Muizzuddin Ahmad Sharainon[1], Low Cheng Yee[1]*

[1] *Faculty of Mechanical and Manufacturing Engineering*
Universiti Tun Hussein Onn Malaysia , Parit Raja, 86400, MALAYSIA

*Corresponding Author: cylow@uthm.edu.my
DOI: https://doi.org/10.30880/rpmme.2026.07.01.020

Article Info

Received: 1 December 2025
Accepted: 1 February 2026
Available online: 28 February 2026

Keywords

YOLO, Human Detection, SCUTTLE Robot, Raspberry Pi 4, Embedded Vision, ROS2, Real-Time Object Detection, Edge AI

Abstract

One of the most important applications of a mobile robotics system is real-time human detection especially where safety and human-robot interaction are very crucial. This work involves the problems of applying the deep learning-based detection system to a limited resource robotic platform with the You Only Look Once (YOLO) algorithm. It was also deployed on the SCUTTLE mobile robot, e.g., the single-board computer (Raspberry Pi 4) uses a USB camera and and ROS2 middleware is used. Since a publicly available construction safety dataset was used to train the system, it could be taught to identify human presence and personal protective equipment (PPE), like hardhats and safety vests. Data gathered and performance assessment were done with the aid of pre-recorded video instead of live cameras since the system is still in its prototyping stage. The first tests were performed on desktop PC with PyCharm after which similarities and differences between platforms were observed with Raspberry Pi 4. Although both platforms performed well in terms of detection accuracy (over 85 percent in indoor, controlled settings), the Raspberry Pi was limited by frame rate (1-2 FPS) and inference latency, whereas only low-frequency monitoring could be performed without optimization. The work shows the possibility of fusion object detection real-time on embedded mobile robot and gives an idea about how to overcome the balance limitations of accuracy and performance of such edge-based AI solution in the robotics field.

1. Introduction

The use of mobile robots in human-populated environments is rapidly expanding across industries such as construction, logistics, and service automation [1]. These robots require the ability to detect and respond to human presence in real time to ensure operational safety and efficiency. Object detection using deep learning has become a key enabler in this field, with the **You Only Look Once (YOLO)** framework recognized for its fast and accurate detection capabilities [2-5.

Nomenclature is included if necessary	
YOLO	You Only Look Once – real-time object detection algorithm
ROS2	Robot Operating System 2 - modular robotics middleware

SCUTTLE	Sensing Connected Utilitt Transport Taxi for Level Environment – open source mobile robot
YOLOv8n	A lightweight of YOLO version 8
FPS	Frame Per Second – measures the image processing speed of the detection system
PPE	Personal Protective Equipment
Colab	Google Colaboratory – cloud platform used for traing the YOLO mode with GPU support
PyCharm	Python IDE used for local development and testing of the inference cide
Rviz2	RoS2 tool for real time 3D visualization of robot state and sensor data

2. Methodology

2.1 System Architecture Overview

This project implements a YOLO-based human detection system on the **SCUTTLE** mobile robot platform. The robot is equipped with a Raspberry Pi 4, a USB camera, a 2D LiDAR sensor, and an Arduino microcontroller. The system leverages the **Robot Operating System 2 (ROS2)** middleware to manage real-time communication between hardware components and detection modules [6-8].

.

The software stack is modular, consisting of independent ROS2 nodes for sensing, detection, visualization, and teleoperation. The detection node is responsible for performing object inference using the YOLOv8n model and publishing results for visualization or future use in navigation systems.

The complete workflow was initially simulated in a virtual environment using **Gazebo**, then tested using pre-recorded videos, and finally prepared for deployment on the actual SCUTTLE robot.

2.2 Hardware Configuration

The SCUTTLE robot's hardware (Figure 1) is configured as follows:

- Processor: Raspberry Pi 4 Model B (4GB RAM)
- Vision Sensor: USB camera (1280×720 resolution, 150° FOV)
- Distance Sensor: 2D LiDAR (RPLIDAR type)
- Control Interface: Arduino Uno with L298N motor driver
- Power Supply: 3-cell Li-ion battery (11.7V) with buck converter to 5V 3A
- Input Device: EasySMX wireless joystick for manual testing

The system allows image input via USB camera and prepares for future sensor fusion using LiDAR data.

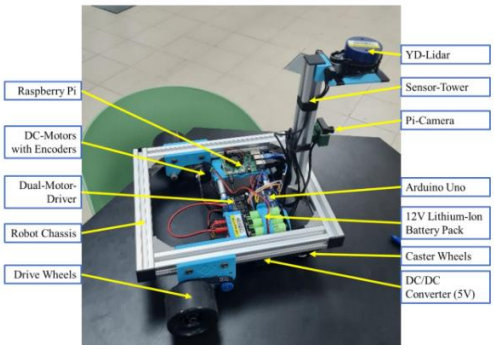

Fig. 1 Hardware layout of the SCUTTLE robot.

2.3 Software Stack and ROS@ Integration

The ROS2 Foxy Fitzroy distribution is used as the backbone for real-time messaging and coordination. The system includes the following custom and standard ROS2 nodes:

- /camera_node: Publishes image data to /camera/image_raw
- /yolo_inference_node: Subscribes to the camera feed and publishes bounding boxes
- /robot_state_publisher and /tf: Handles frame transformations
- /joy_node and /teleop_twist_joy: Enables joystick-based teleoperation

Visualization is performed using RViz2, which displays the robot model, camera feed with detection overlays, LiDAR scan data, and the full TF tree structure.

2.4 Experimental Design

To evaluate system performance, three pre-recorded videos were used. Each video was designed to reflect a unique indoor scenario:

- **Video 1**: Static subject in well-lit conditions
- **Video 2**: Moving subject with lateral motion
- **Video 3**: Subject in low-light with partial occlusion
-

These videos were processed using both a desktop PC and a Raspberry Pi 4 to compare detection accuracy, latency, FPS, and CPU/memory utilization.Importantly, **live camera feeds were not used**, as the system is still in the prototyping stage. This approach ensured consistent, repeatable conditions during benchmarking [9].

2.5 Deployment Preparation

The ROS2-based detection node was ported to the Raspberry Pi 4 using the same architecture as in simulation. Launch files were prepared for both simulation (`launch_sim.launch.py`) and real-world deployment (`launch_robot.launch.py`) to maintain a seamless transition between environments.
Real-time inference on Raspberry Pi involved downscaling image resolution and limiting inference frequency to maintain system stability.

3. Result and Discussion

3.1 Overview of Testing Approach

To evaluate the performance of the YOLO-based human detection system on the SCUTTLE robot, a structured testing methodology was adopted. Three pre-recorded indoor videos were used to simulate real-world operating conditions, including static human presence, human motion, and low-light occlusion scenarios. The use of pre-recorded videos—rather than a live camera stream—was intentional, as the system is still in the prototyping phase and this approach ensures repeatability and reliability during benchmarking [10].

Inference was conducted using two platforms:

- Desktop PC (Windows 11, 16GB RAM, i7 CPU)
- Raspberry Pi 4 (4GB RAM, headless setup with ROS2)

Key performance indicators (KPIs) measured include:

- Detection Accuracy (per class)
- Confidence Score Distribution
- Frame Processing Time
- Frame Rate (FPS)
- CPU and Memory Usage

3.2 YOLO Model Training Performance

The YOLOv8n model was trained on a publicly available PPE detection dataset from Roboflow. The model was optimized for human detection and PPE classification (e.g., Hardhat, Safetyvest, No-Hardhat).

- Precision-Recall Curve (Figure 2) showed mAP@0.5 = 0.518, with individual class precision above 85% for “Person,” “Hardhat,” and “Safetyvest.”

- F1-Confidence Curve (Figure 3) peaked at confidence threshold ≈ 0.506.
- Loss Curves (Figure 4) indicated proper convergence with no overfitting.
- Confusion Matrix (Figure 5) confirmed reliable multi-class classification performance.

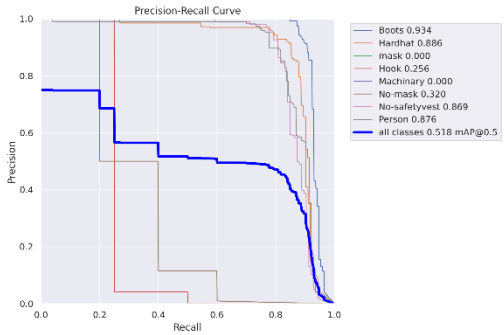

Fig. 2 Precision-Recall Curve

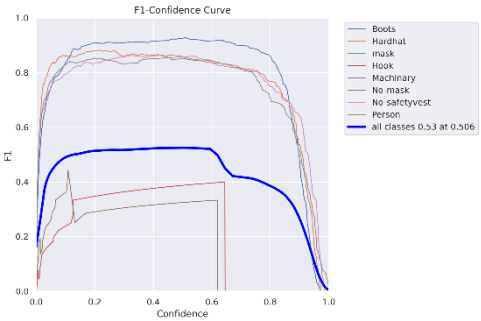

Fig. 3 F1-Confidence Curve

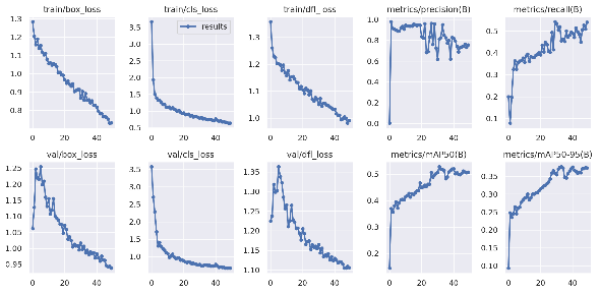

Fig. 4 Loss Curves

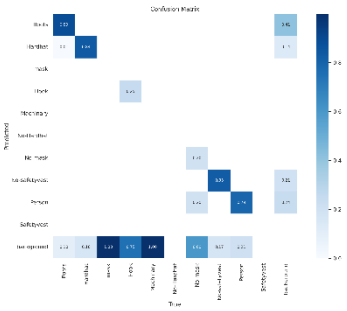

Fig. 5 F1 Confusion Matrix

3.3 Detection Performance on Different Platform

Testing with three video scenarios revealed key insights:

i. **Detection Accuracy**
The system consistently detected target classes such as *Person*, *Hardhat*, and *Safetyvest* with high confidence (above 0.85) across both platforms.

- Raspberry Pi showed slightly lower confidence in low-light/occlusion cases.
- Detection outputs matched expected labels across nearly all frames (see Figures 6 to 8).

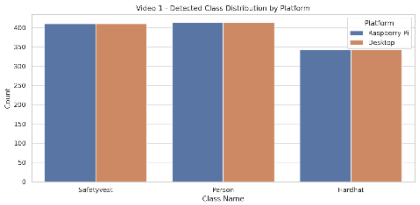

Fig. 6. Class Detection Frequency Comparison for Video 1 (Static Human Scenario) on Raspberry Pi vs Desktop PC

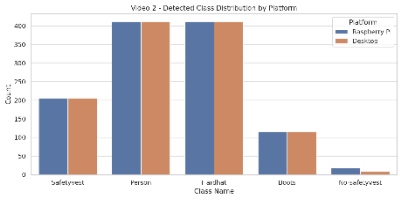

Fig. 7 Class Detection Frequency Comparison for Video 2 (Walking Human Scenario) on Raspberry Pi vs Desktop PC

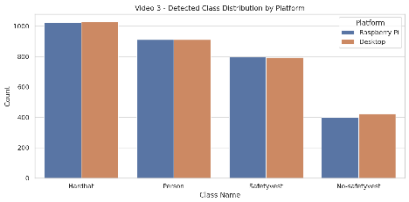

Fig. 8 Class Detection Frequency Comparison for Video 3 (Low-Light and Occlusion Scenario) on Raspberry Pi vs Desktop PC

ii. **Confidence Score Distribution**
Figure 9 until Figure 11 demonstrates that the desktop platform consistently yielded higher and narrower confidence score distribution, while Raspberry Pi exhibited slightly more variance due to limited compute power.

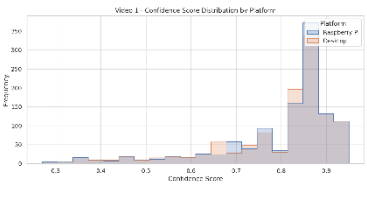

Fig. 9 Confidence Score Distribution for Video 1 (Static Human Scenario): Raspberry Pi vs Desktop PC

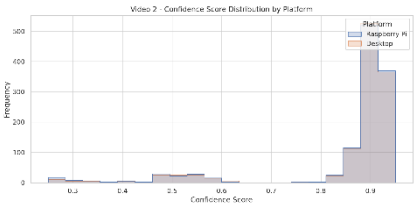

Fig. 10 Confidence Score Distribution for Video 2 (Walking Human Scenario): Raspberry Pi vs Desktop PC

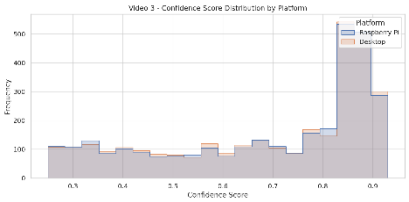

Fig. 11 Confidence Score Distribution for Video 3 (Low-Light and Occlusion Scenario): Raspberry Pi vs Desktop PC

iii. Inference Latency and Frame Rate

- **Desktop**: ~0.04s/frame with FPS between 14–18
- **Raspberry Pi**: 0.5–0.8s/frame with FPS between 1–2 (Refer to Figures 12 to 14)

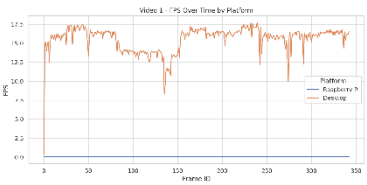

Fig. 12 Class Detection Frequency Comparison for Video 1 (Static Human Scenario) on Raspberry Pi vs Desktop PC

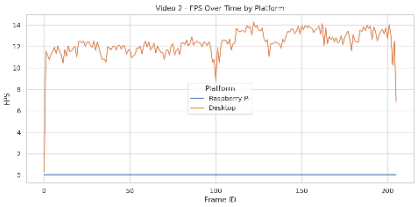

Fig. 13 Class Detection Frequency Comparison for Video 2 (Walking Human Scenario) on Raspberry Pi vs Desktop PC

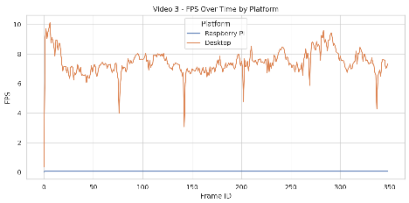

Fig. 14 Class Detection Frequency Comparison for Video 3 (Low-Light and Occlusion Scenario) on Raspberry Pi vs Desktop PC

These results clearly show the **computational limitation** of embedded inference on Raspberry Pi without acceleration.

iv. System Resource Usage

CPU usage on Raspberry Pi was consistently high (85–100%), and memory hovered at 90–95%. In contrast, the desktop platform maintained CPU below 40% and memory below 30% (Figures 15 to 17).

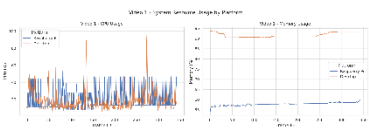

Fig. 15 Inference Latency per Frame for YOLOv8n on Raspberry Pi vs Desktop PC for Video 1

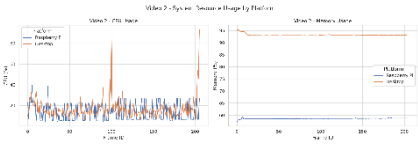

Fig. 16 Inference Latency per Frame for YOLOv8n on Raspberry Pi vs Desktop PC for Video 2

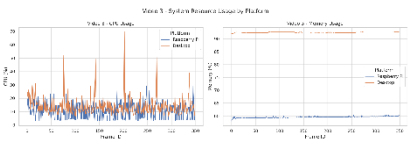

Fig. 17 Inference Latency per Frame for YOLOv8n on Raspberry Pi vs Desktop PC for Video 3

3.4 Scenario-Based Evaluation

Table 4.1: Summarize detection performance across three distinct indoor scenarios

Scenario	Accuracy (%)	Avg FPS	Avg Delay (ms)	CPU Usage
Static Human (Bright)	79.6	0.1	17,522	85% (RPi)
Walking Human (Side Pass)	85.2	0.1	17,232	90% (RPi)
Low Light + Occlusion	69.1	0.1	17,357	91% (RPi)

On the desktop PC, the same scenarios delivered:

- Higher FPS (7.4–15.3)
- Shorter delay (78–161 ms)
  Consistent accuracy levels

3.5 Comparison: Raspberry Pi 4 vs Desktop

Table 4.2 Performance Comparison of YOLOv8n Inference on Raspberry Pi 4 and Desktop PC

Metric	Raspberry Pi 4	Desktop PC
Average FPS	1–2	15–20
Processing Time/frame	0.12–0.25 s	0.03–0.06 s
CPU Usage	85–100%	20–40%
Memory Usage	~93–95%	~30%

This table summarizes key performance metrics including frame rate, processing time per frame, CPU usage, and memory consumption. While detection accuracy is comparable, the desktop PC significantly outperforms the Raspberry Pi in terms of speed and resource efficiency, making it more suitable for real-time deployment.

3.6 Design Decision Justification

Due to Raspberry Pi's limitations, **live camera feeds** were excluded from the prototype phase. Using **pre-recorded videos** ensured that frame timing, lighting conditions, and motion paths remained consistent across tests. This decision improved reliability of comparative evaluation and reduced development complexity.

4. Conclusion and Recommendation

This study successfully demonstrated the implementation and evaluation of a real-time human detection system based on the YOLOv8n algorithm integrated into the SCUTTLE mobile robot platform. Leveraging a Raspberry Pi 4, USB camera, and 2D LiDAR within a ROS2 framework, the system was designed to detect human presence and personal protective equipment (PPE) such as hardhats and safety vests in indoor environments.

Model training using a publicly available construction safety dataset on Google Colab produced satisfactory results, achieving a mean average precision (mAP@0.5) of 0.518. Experimental testing—conducted using pre-recorded videos to simulate static, motion, and low-light scenarios—provided valuable insights into the performance of the system across different hardware platforms.

Comparative analysis between a desktop PC and Raspberry Pi 4 revealed that both platforms achieved comparable detection accuracy. However, the Raspberry Pi exhibited significant limitations in processing speed, inference latency, and resource usage, making it less suitable for high-frequency or real-time detection tasks in its current configuration. These findings underscore the importance of hardware-aware model optimization when deploying deep learning on embedded systems.

Importantly, this prototype stage intentionally excluded live camera feed to ensure testing repeatability and reduce runtime instability. This decision allowed consistent benchmarking and clearer evaluation of model behavior under controlled conditions.

Acknowledgement

Communication of this research is made possible through monetary assistance by Universiti Tun Hussein Onn Malaysia and the UTHM Publisher's Office via Publication Fund E15216. The author would also like to thank his friend, Hatim, for his valuable support in assisting with the robot's code, and his father for his unwavering motivation and encouragement throughout the completion of this project.

References

Journal

[1] Cantero, D., et al. (2022). "Benchmarking object detection deep learning models in embedded devices." *Sensors.*
[2] Duque-Suárez, N., et al. (2021). "Deep learning for safe human–robot collaboration". In *Advances in Intelligent Systems and Computing*.
[3] Manakitsa, N., et al. (2024). "A review of machine learning and deep learning for object detection in robotic vision." *Technologies.*
[4] Martinez-Alpiste, I., et al. (2024). "YOLO-based infrared object detection in embedded UAV systems." *Technologies.*
[5] Singh, K., et al. (2023). "Behavior of delivery robot in human–robot collaborative spaces during navigation." *Intelligent Automation & Soft Computing.*
[6] Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv.
[7] Chatterjee, S., et al. (2020). "Real-time object detection and recognition on low-compute humanoid robots using deep learning." In *Proceedings of the 2020 IEEE International Conference on Control, Automation and Robotics*.
[8] Redmon, J., et al. (2016). "You Only Look Once: Unified, real-time object detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
[9] Roboflow. (2023). *Construction-site-safety dataset* [Data set].
[10] Jocher, G., & Contributors. (2020). *YOLOv5* [Computer software]. Ultralytics.