technologies

MDPI

Article

Artificial Vision System for Autonomous Mobile Platform Used in Intelligent and Flexible Indoor Environment Inspection

Marius Cristian Luculescu *, Luciana Cristea * and Attila Laszlo Boer *

Product Design, Mechatronics and Environment Department, Faculty of Product Design and Environment, Transilvania University of Brasov, 500036 Brasov, Romania

* Correspondence: lucmar@unitbv.ro (M.C.L.); lcristea@unitbv.ro (L.C.); boera@unitbv.ro (A.L.B.)

Abstract: The widespread availability of artificial intelligence (AI) tools has facilitated the development of complex, high-performance applications across a broad range of domains. Among these, processes related to the surveillance and assisted verification of indoor environments have garnered significant interest. This paper presents the implementation, testing, and validation of an autonomous mobile platform designed for the intelligent and flexible inspection of such spaces. The artificial vision system, the main component on which the study focuses, was built using a Raspberry Pi 5 development module supplemented by a Raspberry Pi AI Kit to enable hardware acceleration for image recognition tasks using AI techniques. Some of the most recognized neural network models were evaluated in line with the application's specific requirements. Utilizing transfer learning techniques, these models were further developed and trained with additional image datasets tailored to the inspection tasks. The performance of these networks was then tested and validated on new images, facilitating the selection of the model with the best results. The platform's flexibility was ensured by mounting the artificial vision system on a mobile structure capable of autonomously navigating indoor environments and identifying inspection points, markers, and required objects. The platform could generate reports, make decisions based on the detected conditions, and be easily reconfigured for alternative inspection tasks. Finally, the intelligent and flexible inspection system was tested and validated for its deep learning-based vision system performance.

Keywords: DL—deep learning; CNN—Convolutional Neural Network; YOLO—You Only Look Once; autonomous; navigation; mobile robot; intelligent; inspection

check for updates

Academic Editors: Manoj Gupta, Florian Ion Tiberiu Petrescu and Liviu Marian Ungureanu

Received: 5 March 2025
Revised: 6 April 2025
Accepted: 15 April 2025
Published: 16 April 2025

Citation: Luculescu, M.C.; Cristea, L.; Boer, A.L. Artificial Vision System for Autonomous Mobile Platform Used in Intelligent and Flexible Indoor Environment Inspection. *Technologies* **2025**, *13*, 161. https://doi.org/10.3390/technologies13040161

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

1.1. Automated Indoor Inspection

Automated indoor inspection can revolutionize how buildings and other controlled environments are managed and monitored without direct human intervention. In recent years, due to the implications and effects of this activity, this subject has become an increasingly important and complex concern through an approach based on the study and conception of the use of robots [1–4] and autonomous systems to analyze, monitor, and evaluate the conditions in an indoor environment [1,5], as well as detect anomalies, identify problems, and assess compliance with specific standards.

The success of indoor inspection depends on technological progress, addressing the challenges of navigation, detection, and autonomy, and the responsible integration of these solutions into society [3].

The most commonly implemented technologies for autonomous spatial monitoring include the mapping and analysis of spaces with the help of stereo cameras and LiDAR sensors [4,6], the detection and classification of elements using specialized algorithms for object recognition such as Convolutional Neural Networks (CNNs), the automatic collection of motion-related data [2–4,7] and environmental features, machine learning for predictions, anomaly detection, and automatic decision making based on artificial intelligence (AI) algorithms, the automatic inspection of spaces with autonomous mobile systems, and the detailed evaluation and analysis of spaces with reconstruction by 3D scanning systems [4,8].

The applications of such technologies vary from inspecting commercial and industrial spaces to smart houses, hospitals, and critical spaces such as laboratories or hazardous material warehouses [7].

1.2. Autonomous Indoor Inspection Systems

A high level of utility characterizes autonomous indoor inspection systems without human control. The first aspect considers the reduction in risks for humans. In hazardous environments, for example, spaces with toxic gases and inaccessible or unstable areas, mobile robots can replace humans, reducing the risk of accidents.

Autonomous systems can work continuously, reducing the time required for inspections with a human operator, but above all, eliminating the stress and monotony of repetitive activities and ensuring precise checks with a constant accuracy over time, which demonstrates the increased efficiency of these technologies [9].

The use of these systems can lead to increased accessibility to hard-to-use spaces. In the long term, operational costs are reduced compared to inspections performed with human operators.

An important aspect concerns the assurance of legal, compliance, and constancy requirements, elements that are respected in a reliable and documented way in the case of autonomous solutions.

Autonomous inspection is important not only from the perspective of safety and efficiency, but also for monitoring and optimizing energy consumption, the early detection of problems, and monitoring structural integrity, as well as for integration into IoT systems by creating intelligent ecosystems that can automatically manage conditions from the inside based on data collected by robots.

While autonomous inspection has immense potential, some significant challenges need to be addressed, as follows:

(a) Navigation and localization technologies that involve navigation in complex [2,10], unstructured, or dynamic environments and localization and mapping systems (SLAM—Simultaneous Localization and Mapping) that must be sufficiently accurate in low-light conditions or congested spaces [4];
(b) The detection and recognition of abnormal objects or conditions using advanced computer vision methods [1,2,4,11];
(c) Energy autonomy, i.e., practical solutions for energy management for the prolonged operation of robots;
(d) Interaction with the environment that considers adaptation to unpredictable obstacles and the ability to navigate safely among people, respecting ethical and legal norms [2,3,11,12];
(e) Data privacy and security must address issues related to people's privacy in indoor spaces and protection against cyber-attacks [4].

In this context, the development of autonomous mobile systems equipped with artificial vision to ensure analysis and monitoring following the specifics of the application and

to ensure an increased efficiency is of particular importance for solving complex problems regarding the inspection and autonomous monitoring of indoor spaces and the accuracy, legal, compliance, and constancy requirements imposed.

In the work of Sanchez-Cubillo, J et al. [1], the suitability of such an autonomous system with computer vision is described in three inspection cases by introducing an AI algorithm capable of detecting, classifying, and following the characteristics of the environment. This study highlights the increased performance of autonomous inspection systems equipped with artificial intelligence.

Important studies have focused on the development of object detection based on AI techniques and its implementation on mobile platforms equipped with artificial vision systems. Compared to autonomous indoor inspection systems equipped with artificial vision, the great challenge is the analysis and high-precision processing of acquired image streams and the recognition of objects, people, and border situations.

Recent concerns aim to improve object detection and recognition performance by integrating artificial intelligence with artificial vision systems implemented in various applications. Bai, C. et al. [11] review the object detection of AI algorithms based on improvement methods such as multiscale, attention mechanism, and supervision. The study analyzes algorithms such as YOLO, SSD, and Transformer and states that the YOLO series algorithms are suitable for real-time object detection due to their fast speed and high accuracy, but with a poor performance for small objects. Algorithms in the SSD series provide a good performance in detecting small objects, but the detection accuracy is reduced [7].

In Song, Q. et al.'s [12] study, an improved multiscale parallel YOLO algorithm named YOLO-MSP is tested on pedestrian recognition. The proposed algorithm assures an increased accuracy and speed in pedestrian detection.

Halder et al. [4] review the usage of robots in inspecting and monitoring buildings and infrastructure. The report highlights that the research in this field is structured in the following eight directions: autonomous navigation, motion control systems, sensing, knowledge extraction, safety, multi-robot, human–robot collaboration, and data transmission [4].

Based on this study's results, there is a need to concentrate current research on solving the problems related to the realization of autonomous integrated mobile systems to inspect indoor spaces. This aspect involves solving problems related to autonomous navigation, optimal image selection and capture, and increasing the performance of the object recognition system, as well as decision making and data transmission [4].

The key contribution of this research includes integrating one of the latest advancements in edge computing for embedded artificial vision applications (Raspberry Pi AI Kit, launched in 2024) into an autonomous mobile inspection platform. To the best of our knowledge, this is the first embedded artificial vision system utilizing the Raspberry Pi AI Kit for an autonomous mobile inspection platform mentioned in a journal paper. A literature search in international databases such as Clarivate and Scopus found no published studies related to this AI accelerator. Other significant contributions include the design of a flexible system architecture enabling seamless adaptation to various mobile inspection platforms and implementing a specialized YOLO training methodology to improve detection accuracy and enhance generalization capabilities. All these contributions are outlined in the conclusions section.

In this context, the main objective of the research presented in this work was developing and implementing an autonomous mobile platform for intelligent and flexible indoor environment inspection to validate the concept and evaluate the performance of the artificial visual system.

2. Materials and Methods

The following specific objectives were established to achieve the main objective:

(a) **The development and implementation of a mobile platform (hardware)** capable of remote control and autonomous navigation based on sensor data;
(b) **The development and integration of a computer vision system (hardware)** to enhance the perception capabilities of the mobile platform;
(c) **The design and implementation of an object detection module (software)** utilizing artificial intelligence techniques;
(d) **The deployment of the object detection module** on the mobile platform, creating an artificial vision system;
(e) **The development and implementation of an autonomous navigation algorithm** leveraging object detection for enhanced mobility and decision making;
(f) **The development of a software module for managing the inspection process** in indoor environments, ensuring flexible missions and efficient data collection;
(g) **The comprehensive testing and validation of the artificial vision system** to assess its accuracy and operational efficiency under real-world conditions.

A detailed flowchart of the research is presented in Figure 1.

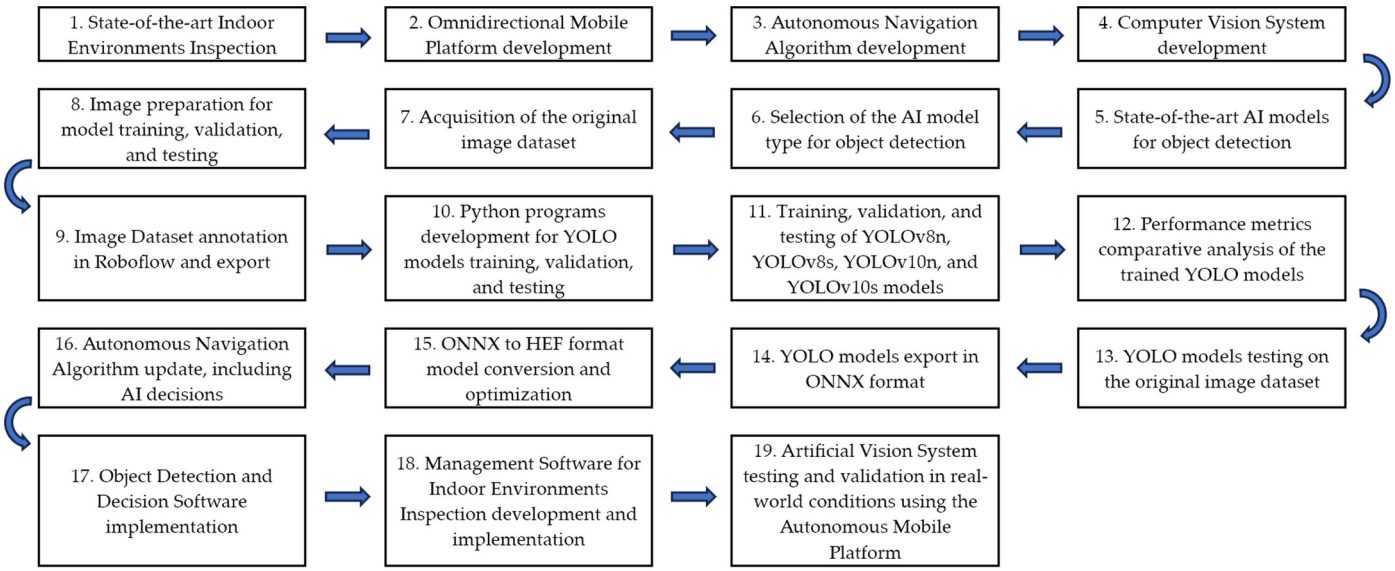


Figure 1. The research flowchart.

The steps covering the introduction and addressing all the specified objectives are detailed in the following sections of the paper.

2.1. Description of the Inspection Process

The inspection process is designed to verify the presence of specific objects within indoor environments and detect potentially undesirable situations, such as open windows or doors. To facilitate object presence verification, designated observation points are defined and marked. These locations are hereinafter referred to as Inspection Points (IPs).

At each IP, the autonomous mobile platform must stop and utilize the artificial vision system to determine whether the objects associated with that location are present or absent. Additionally, if the system detects open windows or doors during its movement, it immediately reports the event to the central server, triggering an alert.

The inspection process follows a structured sequence of steps, as follows: **system initialization**—the system starts from a predefined START point based on an inspection

mission; **autonomous navigation**—the platform moves autonomously, leveraging sensor-based obstacle avoidance while simultaneously scanning its surroundings to identify IP markers; **IP detection and approach**—once an IP marker is identified, the system orients itself towards the designated location; **positioning at the IP**—the platform stops precisely at the identified IP; **object detection and verification**—the artificial vision system scans the area to identify and confirm the presence of predefined objects, and the inspection results are then transmitted to the central system; and **continuation of navigation**—the platform proceeds either to the next IP or towards the predefined STOP point.

Upon reaching the STOP point, the system may initiate a new inspection mission or return to its initial location. The START and STOP points are marked similarly to the IPs, ensuring a structured and efficient inspection workflow.

2.2. Structure of the Inspection System

Figure 2 presents the system's structure, comprising hardware and software components.

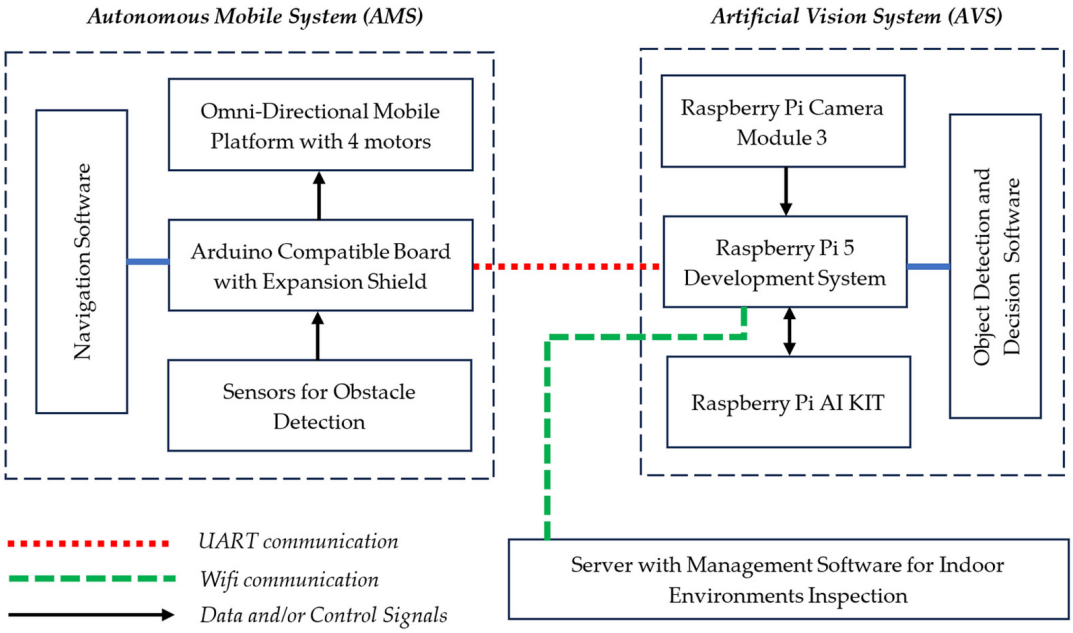


Figure 2. The structure of the autonomous mobile platform for intelligent and flexible indoor environment inspection.

This structure is further detailed in the following subsections, focusing on the materials used in the study and the methods applied to implement, test, and validate the platform.

2.3. Hardware Subsystem

The hardware subsystem comprises the following three main components: an omnidirectional mobile platform, a motion control subsystem, and a computer vision subsystem.

The development process starts with acquiring a Tscinbuny ESP32 Robot for Arduino Uno Starter Kit [13], which serves as the basis for the inspection platform to ensure a reliable mobility within indoor environments.

Figure 3 presents a comparative view of the initial mobile platform and the enhanced version resulting from system modifications.

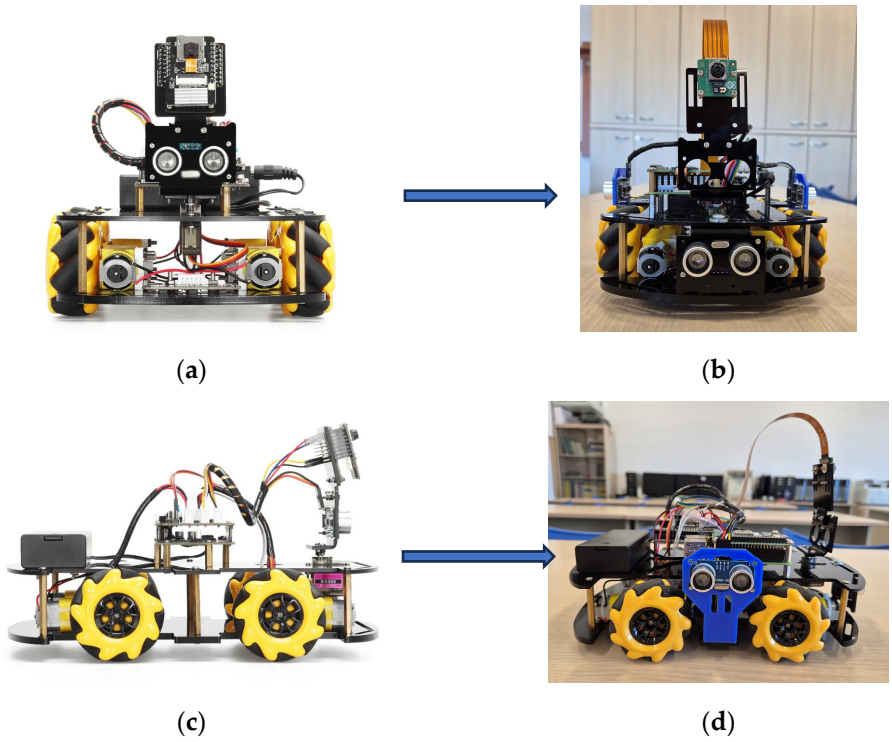


Figure 3. The autonomous mobile platform used for indoor environment inspection. (**a**) Tscinbuny ESP32 Robot for Arduino Uno—front view [13]; (**b**) modified platform—front view; (**c**) Tscinbuny ESP32 Robot for Arduino Uno—lateral view [13]; and (**d**) modified platform—lateral view.

2.3.1. Initial Structure of the Mobile Platform

The Tscinbuny Omni-Directional Mobile Platform contains a chassis equipped with the following components: four gear motors enabling Four-Wheel Drive (4WD); four Mecanum wheels facilitating omnidirectional movement; a tracking module featuring three line-following sensors positioned at the front of the robot; a servo motor that actuates the support for an ultrasonic sensor with a detection range of 2–400 cm; an ESP32 CAM module; and the Motion Control Subsystem (MSC). The system is powered by a module consisting of two 18,650 batteries, and video streaming is facilitated via the ESP32 CAM's Wi-Fi interface.

The MCS is implemented using an Arduino-compatible development board mounted with an Expansion Shield (ES). The ES incorporates drivers and connectors for each of the four DC motors and connectors for the sensors and the servo.

2.3.2. Modified Structure of the Mobile Platform

Several modifications and improvements are implemented to create a mobile platform capable of remote control and autonomous movement based on sensors for wall tracking or obstacle avoidance.

Remote control is facilitated through a connection to a web server hosted on the ESP32 CAM, with command transmission occurring via the Wi-Fi communication interface. The received commands are sent to the MCS through an asynchronous serial communication interface.

For obstacle avoidance and/or wall tracking, two additional ultrasonic sensors are installed on the lateral sides of the platform, and the position of the existing front-facing ultrasonic sensor is adjusted. The required distance to be maintained from walls or obstacles can be configured through the graphical interface of the inspection process management system.

2.3.3. Artificial Vision System

The ESP32 CAM module is replaced with a more powerful subsystem based on a Raspberry Pi 5 development board with 8 GB of RAM [14], paired with a Raspberry Pi Camera Module 3 [15] featuring a Sony IMX708 sensor, to develop and implement a computer vision system (hardware) for the mobile platform.

The camera offers a resolution of 11.9 Mpx (4608 × 2592 px) and a 75° field of view. The camera is mounted on a support that can be oriented with the help of a servo motor. The platform scans the environment during movement to identify markers for inspection points, START, or STOP. The scanning involves positioning the camera successively at −90°, −5°, 0°, +45°, and +90° relative to the direction of motion, capturing images from each position, and processing them to identify the markers.

The computer vision subsystem evolves into an artificial vision system when the Raspberry Pi 5 runs AI applications. Given the requirement for real-time object detection, the integration of a supplemental AI accelerator must be considered. A study evaluated the available AI accelerators for embedded systems on the market. Three models with the best performance are presented in Table 1.

Table 1. Comparison between AI accelerators [16].

Feature	Coral USB Accelerator	Coral M.2 Accelerator with Dual Edge TPU	Raspberry Pi AI Kit
Release Year	2020	2021	2024
Form Factor	USB 3.0 Type-C	M.2-2230-D3-E	M.2 2242
AI Chip	Edge TPU	Dual Edge TPU	Hailo-8L
Performance	4 TOPS	8 TOPS	13 TOPS
Connectivity	USB 3.0 Type-C	Two PCIe Gen2 × 1 interfaces	PCIe 3.0 (Single Lane, 8 Gbps)
Compatibility	Linux, macOS, Windows	Debian-based Linux, Windows 10	Raspberry Pi 5
AI Applications	AI NVR systems, home automation, vision projects	Industrial AI, edge AI development, traffic systems	Computer vision, object detection, smart home automation
Price	USD 59.99	USD 39.99	USD 70

Due to the 13 tera-operations per second (TOPS) neural network inference accelerator based on the Hailo-8L chip and the use of the Raspberry Pi 5 development board, the **Raspberry PI AI Kit** is selected for the artificial vision system [17].

2.4. Software Subsystem

The software subsystem consists of the following three components: the Platform Autonomous Navigation Software implemented on the MCS; the Object Detection and Decision Software (ODDS) implemented on the Raspberry Pi 5 development board; and the Management Software for Indoor Environments Inspection implemented on the server.

2.4.1. Platform Autonomous Navigation Software

Depending on the characteristics of the inspection environment, the platform can be programmed to navigate along a wall at a specified distance or to move freely. In both cases, obstacles are detected using ultrasonic sensors and avoided. Additionally, the camera of the

artificial vision system is controlled to scan for markers corresponding to IPs. The software is loaded onto an Arduino-compatible development board, which sends commands to the four motors. Through various combinations of commands applied to the motors, the omnidirectional platform can move in the following directions: forward, backward, left, right, top left, top right, bottom left, bottom right, clockwise rotation, and counterclockwise rotation. Navigation is also influenced by the commands received from the Object Detection and Decision Software.

2.4.2. Object Detection and Decision Software

This program runs on the Raspberry Pi 5 and utilizes the Raspberry Pi AI Kit accelerator for real-time object detection and classification.

During the mobile platform's movement, the MCS commands the servo to position the camera mount, then sends a command via the UART (Universal Asynchronous Receiver-Transmitter) interface to the AVS to capture an image.

The ODDS receives the command, captures the image, and analyzes it using the AI module. If an IP marker is detected in the image, the ODDS calculates its position relative to the center of the image and sends a command via the UART to the MCS to direct the mobile platform towards the IP marker. If no IP marker is found, the ODDS sends a "Detection process finished" message to the MCS, prompting the camera to rotate to the next position.

When the mobile platform is positioned in front of the IP marker, with the camera oriented towards it, the MCS commands the platform to rotate 180° around its center. This action ensures that the platform's rear faces the marker, and the camera is directed towards the inspection area, allowing image capture for the inspection process. The ODDS then verifies if the required objects are in the inspection area and reports the results to the Management Software for Indoor Environments Inspection implemented on the server. Subsequently, the ODDS sends a command to the MCS to continue the inspection process and locate the next IP marker.

A detailed discussion of the motion and obstacle avoidance algorithms and the positioning at the IPs, including the challenges encountered, tests performed, the algorithm's performance in complex indoor environments with dynamic obstacles, such as moving people or objects, and the results, is presented in a separate paper focused on autonomous navigation.

2.4.3. AI Models Used for Object Detection

This subsection focuses on the object detection problem utilizing AI techniques [18–20]. While various solutions exist [18], the most promising results are typically achieved using Convolutional Neural Networks (CNNs) [5,21]. The object detection problem involves object recognition and the localization of objects within an image [22]. These models can be categorized based on their approach to generating region proposals, as follows: two-stage and one-stage detectors [23].

Two-stage detectors first generate potential regions of interest (ROIs), which represent possible object locations, and in the second step, they classify and refine the bounding boxes of these proposals to achieve more precise localization. In contrast, **one-stage detectors** predict object classes and bounding boxes in a single pass through the network.

Table 2 presents a comparison between these two categories based on their different features and the recommended applications and devices for deploying these models.

Table 2. Comparison between two-stage and one-stage models for object detection.

Feature	One-Stage (OS)	Two-Stage (TS)
Accuracy (mAP)	Good, but lower than TS	Higher than OS
Speed (FPS)	Fast–Very fast	Very slow–Medium
Computational Cost	Low–Medium	Medium–High
Complexity	Medium	High
Recommended devices	Edge, Embedded	With GPU
Recommended applications	Real-time apps, autonomous driving, video surveillance, robotics	High-precision object detection, medical imaging

The most important features are as follows: *Accuracy*, expressed by Mean Average Precision (mAP), a common metric for object detection (a higher mAP indicates a better accuracy); *Speed*, expressed by frames per second (FPS), indicating how many images the model can process per second (a higher FPS means faster processing); *Computational cost*, referring to the amount of processing power (CPU/GPU) (a higher cost generally means more demand on hardware); and *Complexity*, which describes the difficulty of implementing and training the model.

A detailed analysis of the two categories highlights the most important object detection models, summarized in Table 3 [5,8,11,19,21,24–26].

Table 3. Two-stage and one-stage model features for object detection.

Model Type	Algorithm	Accuracy (mAP)	Speed (FPS)	Computational Cost	Complexity
Two-stage	R-CNN	High	Very slow	High	Complex
	Fast R-CNN	High	Faster	Moderate	Moderate
	Faster R-CNN	Very high	Moderate	Moderate	More complex than Fast R-CNN
	Cascade R-CNN	Highest	Slower	High	Most complex
One-stage	YOLO	High	Very fast	Low	Relatively simple
	SSD	Moderate	Fast	Moderate	Moderate
	RetinaNet	High	Moderate	Moderate	Moderate
	EfficientDet	High	High	Moderate	Moderate

2.4.4. AI Model Analysis and Selection

The decision for model selection is based on the considerations outlined above, as well as specific constraints such as the real-time processing requirements, hardware compatibility, limitations, and particularities of the inspection process.

Since our platform is mobile, the artificial vision system must be implemented on an embedded system. Therefore, regarding the real-time processing requirements, a model with a low computational cost is crucial. This is why a one-stage detector has to be selected. Comparing the existing solutions, YOLO seems to be optimized for GPU, SSD has a moderate resource consumption and can run efficiently even on embedded devices with limited capabilities, and RetinaNet requires a powerful GPU, making it less suitable for real-time applications with limited resources.

As shown in Tables 2–4, the YOLO, SSD, and RetinaNet models analyzed exhibit significant differences in FPS across different hardware platforms. Depending on the version and hardware used, YOLO achieves between 30 and 161 FPS. SSD provides an intermediate performance, ranging between 20 and 60 FPS. RetinaNet achieves the lowest FPS, between 5 and 30 FPS.

In the analysis of object recognition models, the criteria Accuracy, expressed by Mean Average Precision (mAP), and Complexity, which describes the difficulty of implementing and training the model, are also considered.

Regarding accuracy, YOLO has a high accuracy, as can be observed in Table 4, depending on the version and dataset used. The latest versions (YOLOv8, YOLOv9, and YOLOv10) offer significant improvements over previous ones, maintaining a speed over 100 FPS. SSD has a moderate accuracy, with an mAP between 30% and 45%, performing worse than YOLO and RetinaNet. RetinaNet is the most accurate of the three.

Analyzing these models based on the difficulty of implementing and training the model, YOLO, despite requiring a powerful GPU for training, is relatively easy to implement due to extensive documentation and the availability of pre-trained models. SSD is the simplest to train and implement, with an architecture that is easier to optimize for resource-limited devices. RetinaNet is the most complex to train and implement, being more challenging than YOLO or SSD.

Regarding applicability in indoor environments, YOLO is a high-speed object detection model that is well-adapted to variable lighting conditions and can be optimized for edge computing. However, it has limitations in detecting very small or overlapping objects. SSD detects objects of various sizes well, but may struggle with small and overlapping objects, and RetinaNet provides the highest accuracy for small objects and improved detection in complex environments. However, it is too slow for autonomous mobile platforms and requires powerful hardware.

Choosing the appropriate model involves a trade-off between speed, accuracy, and the available hardware resources.

Given its high accuracy and fast processing speed requirements, the YOLO (You Only Look Once) model is selected, as it is one of the highest-performing models in terms of these features. YOLO's reduced complexity and low computational cost align with the system's needs.

The ODDS runs on the Raspberry Pi 5 platform, which is paired with the Raspberry Pi AI Kit—a new AI accelerator solution based on the Hailo-8L chip. The Hailo-8L chip can only execute HEF (Hailo Executable Format) binary files, and YOLO is one of the models that can be compiled and converted into HEF.

It is important to note that YOLO is not a single model, but a family of open-source models developed by various authors. Since the release of YOLOv1 in 2015, 11 versions have been released, up to YOLOv11 by the end of 2024, along with 8 variants [6,27–29]. Table 4 provides an overview of these versions based on the official GitHub repositories for each YOLO release and some references.

Table 4. YOLO family models [1,11,27,28,30,31].

Model	Release Year	Accuracy (mAP) %	Speed (FPS)	Model Size (MB)	Parameters (Million)	Key Features and Improvements
YOLOv1	2015	63.4	45	193	62	Introduced real-time object detection with a single neural network
YOLOv2	2016	76.8	40	193	67	Improved accuracy with batch normalization, high-resolution classifier, and anchor boxes
YOLOv3	2018	78.6	30	236	62	Added feature pyramid networks for better detection at different scales

Table 4. *Cont.*

Model	Release Year	Accuracy (mAP) %	Speed (FPS)	Model Size (MB)	Parameters (Million)	Key Features and Improvements
YOLOv4	2020	82	62	244	64	Incorporated CSPDarknet53, PANet, Mish activation, and Mosaic data augmentation
YOLOv5	2020	88	140	27	7.5	Focused on ease of use, smaller model sizes, and faster inference
YOLOv6	2022	52	123	17	15.9	Optimized for industrial applications and improved training strategies
YOLOv7	2022	56.8	161	72	36.9	Introduced extended efficient layer aggregation networks
YOLOv8	2023	53.9	111	68	43.7	Featured a new backbone and neck architecture and improved benchmark performance
YOLOv9	2024	54	112	69	44	Further optimized architecture and enhanced feature extraction
YOLOv10	2024	55	115	70	45	Better handling of small objects and refined training procedures
YOLOv11	2024	57	120	72	46	Enhanced feature extraction and optimized for efficiency and speed

The input image size for YOLO models can vary depending on the specific model and implementation, with common resolutions including 416 × 416, 640 × 640, and 1280 × 1280.

Some YOLO models have multiple variants—nano (n), small (s), medium (m), large (l), and extra-large (x)—each differing in size and complexity, which directly impacts speed, accuracy, and resource requirements [32].

For our system, the following two YOLO models developed by Ultralytics are considered for evaluation: YOLOv8, a state-of-the-art model for object detection, classification, and instance segmentation [6,28], and YOLOv11, the latest model in the series, which supports multiple tasks, including image classification, object detection, semantic segmentation, instance segmentation, keypoint detection, and Oriented Bounding Box (OBB) [29].

A conversion from ONNX (Open Neural Network Exchange) format to HEF (Hailo Executable Format) is necessary to deploy a trained YOLO model on the Raspberry Pi AI Kit [33]. Unfortunately, during our testing, the YOLOv11 model could not be directly converted into HEF format. As a result, we opt to replace it with YOLOv10.

The ODDS uses a YOLO model for object detection and classification. When an IP marker is detected with a confidence score exceeding a predefined threshold, its position in the image is calculated relative to the camera's center. The system determines the necessary adjustments by correlating this position with the camera's orientation angle relative to the platform's direction of movement. Then, it sends a command to the Motion Control Subsystem (MCS) to ensure that the platform moves directly toward the detected IP marker.

2.4.5. Dataset Used for Training, Validation, and Testing

The initial training, validation, and testing of the pre-trained YOLO models were conducted using a dataset consisting of 1033 images in .jpg format, containing objects categorized into nine classes. These were **IP markers** (four classes): *Inspection_1*, *Inspection_2*, *Inspection_3*, and *Inspection_4*; **Final position marker** (one class): *Stop_Sign*

(indicating the end of the inspection process); **Target object for presence verification** (one class): *Fire_Extinguisher*; and **Structural elements for anomaly detection** (three classes): *Door_Open_DI3* (open door detection), *Window_Open_DI3* (open window detection in room areas), and *Window_Open_Hall* (open window detection in hallways). These high-resolution images were captured using the camera of a Samsung S24+ smartphone under varying lighting conditions and from multiple angles, including the presence of shadows and other surrounding objects, in order to ensure a certain level of dataset diversity.

From the initial 1033 images, smaller regions containing the objects of interest were cropped and used for training, validating, and testing the models. This preprocessing step aimed to enhance detection accuracy by focusing on relevant features. However, the final evaluation was conducted on the original, uncropped images to ensure the model's ability to generalize to unconstrained environments and to assess the model's real-world performance in detecting objects within full-scene contexts.

Figure 4 provides sample cropped images representing each class.

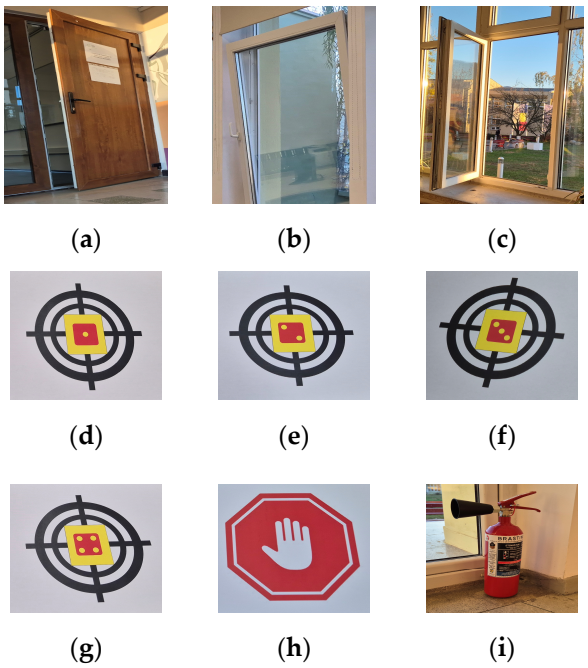


Figure 4. Image samples for the dataset classes. (**a**) Door_Open_DI3; (**b**) Window_Open_DI3; (**c**) Window_Open_Hall; (**d**) Inspection_1; (**e**) Inspection_2; (**f**) Inspection_3; (**g**) Inspection_4; (**h**) Stop_Sign; and (**i**) Fire_extinguisher.

Transfer learning was applied to train the model using pre-trained nano and small versions of YOLOv8 and YOLOv10. These models were imported and fine-tuned on the dataset using custom Python 3.12 scripts.

Image annotation was performed using Roboflow. Each uploaded image was manually annotated by drawing a bounding box around the target object and assigning it to one of the nine predefined classes. Once all 1033 images were labelled, the dataset was finalized.

The images were split automatically by Roboflow into the following three subsets: **training subset**—70% of the images (725 images); **validation subset**—20% of the images (208 images); and **testing subset**—10% of the images (100 images). Table 5 presents a detailed overview of the dataset structure.

Table 5. Dataset structure: number of images for model training, validation, and testing.

Class Number	Class Name	Training Count	Validation Count	Testing Count	Total Count
0	Door_Open_DI3	81	23	11	115
1	Fire_extinguisher	56	16	8	80
2	Inspection_1	56	16	8	80
3	Inspection_2	56	16	8	80
4	Inspection_3	57	16	7	80
5	Inspection_4	56	16	8	80
6	Stop_Sign	56	16	8	80
7	Window_Open_DI3	140	40	20	200
8	Window_Open_Hall	167	49	22	238
	TOTAL images	725	208	100	1033

After creating the dataset, it was exported in the YOLOv8-specific format, which included three folders (*train*, *valid*, and *test*) and a *data.yaml* configuration file.

Each folder contained an images subfolder with the dataset images and a labels subfolder with corresponding annotation files.

A label text file was generated for each image, containing the assigned class number and the normalized bounding box coordinates. All images were resized to 640 × 640 pixels using Roboflow to ensure compatibility with YOLOv8.

This dataset and custom Python scripts were used for training, validation, and testing. The results are presented in the following section.

2.4.6. Management Software for Indoor Environments Inspection

The entire inspection process is managed by a dedicated software module running on a server connected to the same network as the inspection area. This module provides the following functionalities: **Mission Planning**—users can create an inspection mission, define the inspection path, and specify Inspection Points (IPs) along with the objects that need to be verified at each location; **Navigation Mode Selection**—the system allows the user to choose whether the mobile platform navigates autonomously, following walls, or moves freely to locate IPs; **Automated Alerts**—if enabled, the system can receive alerts from the mobile platform when open doors or windows are detected, correlating these notifications with the last identified IP; and **Manual Control and Video Streaming**—the software includes an option for manual operation of the mobile platform and provides real-time video streaming from the onboard camera.

3. Results

This section details the evaluation methodology for model performance, the results obtained during the training process of YOLOv8 and YOLOv10, and their performance on the following two datasets: a test set of 100 cropped images and the entire dataset of 1,033 uncropped images.

All training and validation experiments were conducted on a laptop. The models were trained using the following two approaches: **fixed training** with 200 epochs and **adaptive training**, where the number of epochs was dynamically adjusted based on performance plateau detection to prevent overfitting.

After training, the models were exported in ONNX format and converted into HEF for deployment on the Raspberry Pi AI Kit to enable real-time object detection in the inspection process. Finally, the results obtained during the testing and validation of the artificial vision system are presented.

3.1. Model Performance Evaluation

In the context of YOLO-based object detection, model predictions involve the following two critical aspects: **object detection**, which determines the bounding box location, and **classification**, which assigns the correct class label to the detected object.

Object localization accuracy is evaluated using **Intersection over Union** (*IoU*), a key metric that measures how well the predicted bounding box overlaps with the ground truth bounding box. *IoU* is defined as follows:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{1}$$

where the *Area of Overlap* is the shared area between the predicted box and the ground truth box, while the *Area of Union* is the total area covered by both boxes combined (including the overlap). *IoU* values range from zero to one, with higher values indicating a better localization accuracy.

The classification performance of YOLO models is evaluated using the ***prediction score***, which measures the confidence level of the model in assigning an object to a specific class. It is defined as follows:

$$Prediction\ score = OC \times CP = (PO \times IoU) \times CP \tag{2}$$

where *OC* is the *object confidence* that measures the probability that an object exists within the predicted bounding box, *CP* is the *class probability* computed by the model to indicate the likelihood that the detected object belongs to a specific class, and *PO* is the probability computed by the model that the bounding box contains an object.

For each detected object, a prediction score is computed across all possible classes, and the highest score determines the assigned class label.

During training, the YOLO model is optimized to maximize the score for correct predictions and minimize it for incorrect predictions. Confidence is used in the loss function, which penalizes incorrect bounding boxes and class predictions.

When the model is evaluated, thresholds are set for IoU (typically IoU = 0.5) and for the prediction score so that the object can be classified in a specific class (typically conf = 0.25 for YOLO models). Based on these thresholds, YOLO model predictions are classified as follows:

- **True Positive** (TP) if the model correctly detects an object (IoU $\geq$ threshold) and classifies it into the correct category;
- **False Positive** (FP) if a bounding box is predicted but it does not sufficiently overlap with a ground truth box (IoU < threshold), or the class prediction is incorrect;
- **False Negative** (FN) if an object is present in the ground truth box, but the model fails to detect it, or the predicted bounding box is of the wrong class;
- **True Negative** (TN) if the model correctly identifies an image region as not containing any object of interest.

In object detection, true negatives (TNs) are generally not considered the same way as in classification problems, since YOLO models focus on detecting objects rather than confirming their absence. Consequently, TN is typically not computed for bounding boxes.

The performance of an object detection model is commonly evaluated using the following metrics: precision, recall, accuracy, F1-score, average precision, and mean average precision computed with the formulas presented below.

Precision measures the proportion of correct detections among all predicted detections, as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

A high precision score indicates that most detected objects are correctly classified, minimizing false positives.

Recall quantifies the proportion of actual objects in the dataset that were successfully detected, as follows:

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

A high recall score indicates that the model successfully detects most objects, minimizing false negatives.

Accuracy represents the proportion of correct predictions (both *TP* and *TN*) relative to all predictions, as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \tag{5}$$

F1-Score is the harmonic mean of precision and recall, and it is used when both precision and recall are equally important.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

The **average precision** (*AP*) quantifies the area under the precision–recall curve generated by varying the confidence threshold. For a single class, *AP* is computed with the following formula:

$$AP = \int_0^1 P(R)dR \tag{7}$$

where *P(R)* is the precision at each recall value.

Mean average precision (*mAP*) extends the *AP* by averaging it across all object classes in a dataset, providing a single value for the evaluation of the overall performance of an object detection model.

$$mAP = \frac{1}{N} \sum_{I=1}^{N} AP_i \tag{8}$$

where *N* is the number of classes and *APi* is the average precision for the i-th class.

The following two variants of mAP are commonly used for model performance evaluation: mAP@0.5 (Mean Average Precision at IoU = 0.5) and mAP@0.5:0.95 (Mean Average Precision across IoU thresholds ranging from 0.5 to 0.95 in steps of 0.05). While mAP@0.5 can easily achieve high scores because it allows for detections with only a moderate overlap, mAP@0.5:0.95 provides a more robust assessment of model performance by considering varying levels of overlap strictness.

3.2. Training, Validation, and Testing of YOLO Models

The training, validation, and testing of YOLO models were performed using Python programs running on a DELL Precision 3551 laptop, manufactured by DELL Technologies, and sourced from Brasov, Romania, equipped with an Intel® Core™ i7-10750H processor, 32 GB RAM, 1TB SSD, and an Nvidia Quadro P620 GPU with 4GB graphics memory.

Four pre-trained YOLO models were imported and trained using the 725-image dataset, with validation performed on an additional 208 images. The key characteristics of each imported model are summarized in Table 6.

The parameters of each YOLO model refer to learnable weights and biases that the model adjusts during training. Gradients are the partial derivatives of the loss function with respect to the parameters. Some parameters may not require gradient updates due to architectural constraints. Computational complexity represents the number of floating-point operations (FLOPs) required to process a single input image.

Table 6. Imported YOLO models' characteristics.

MODEL Type	Number of Layers	Number of Parameters	Number of Gradients	Computational Complexity
YOLOv8n	225	3,012,603	3,012,587	8.2 GFLOPs
YOLOv8s	225	11,139,083	11,139,067	28.7 GFLOPs
YOLOv10n	385	2,710,550	2,710,534	8.4 GFLOPs
YOLOv10s	402	8,073,318	8,073,302	24.8 GFLOPs

The following parameters were configured for training the YOLO models [34]: number of epochs = 200; batch size = 16; patience=100 (default setting); device = 0 (GPU used for processing instead of CPU); optimizer = SGD (Stochastic Gradient Descent); and initial learning rate lr0 = 0.01. Further discussions of these parameters and their influences on model performance are presented in the next section.

A confusion matrix was utilized to assess the training effectiveness of each model. This matrix compared the model's predictions to the actual ground truth labels, providing insights into the errors made by the model and identifying areas where it performed well.

A program for model testing was written in Python to compute and display for each following class: the total number of tested images; TP—true positive; FP—false positive; FN—false negative; precision; recall; F1-score; and accuracy for certain values of thresholds (Table 7).

Table 7. Results obtained with YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s models tested on the VALIDATION dataset with a 0.25 confidence threshold.

Class Name	Total Images	TP	FP	FN	Precision	Recall	F1-Score	Accuracy
Door_Open_DI3	23	23	0	0	1.00	1.00	1.00	100.00%
Fire_extinguisher	16	16	0	0	1.00	1.00	1.00	100.00%
Inspection_1	16	16	0	0	1.00	1.00	1.00	100.00%
Inspection_2	16	16	0	0	1.00	1.00	1.00	100.00%
Inspection_3	16	16	0	0	1.00	1.00	1.00	100.00%
Inspection_4	16	16	0	0	1.00	1.00	1.00	100.00%
Stop_Sign	16	16	0	0	1.00	1.00	1.00	100.00%
Window_Open_DI3	40	40	0	0	1.00	1.00	1.00	100.00%
Window_Open_Hall	49	49	0	0	1.00	1.00	1.00	100.00%
TOTAL	208	208	0	0				100.00%

The program also generated a confusion matrix to analyze model performance across different classes further. As observed in Table 7, all four YOLO models correctly detected and classified all objects in the dataset.

A comparative analysis of model performance metrics for all four models (YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s) after 200 training epochs is presented in Figure 5.

A comparison of the loss function trends for the models after 200 training epochs is illustrated in Figure 6.

The next step involved evaluating these models on the 100-image test dataset. The tests were conducted using a confidence threshold of 0.25 and an IoU threshold of 0.7, which are the default values for Ultralytics YOLO [35]. Figure 7 presents the confusion matrices, raw and normalized.

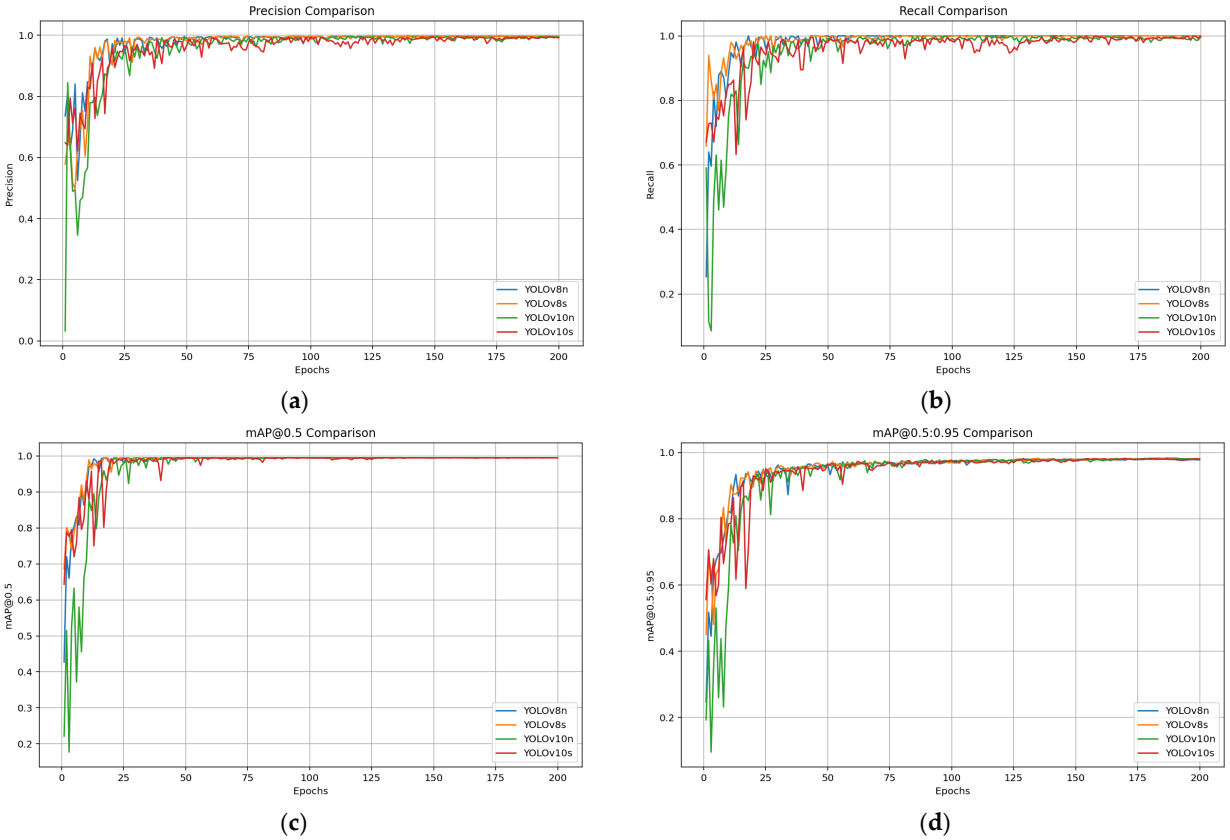


Figure 5. Metrics comparison for models YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s. (**a**) Precision; (**b**) recall; (**c**) mAP@0.5; and (**d**) mAP@0.5:0.95.

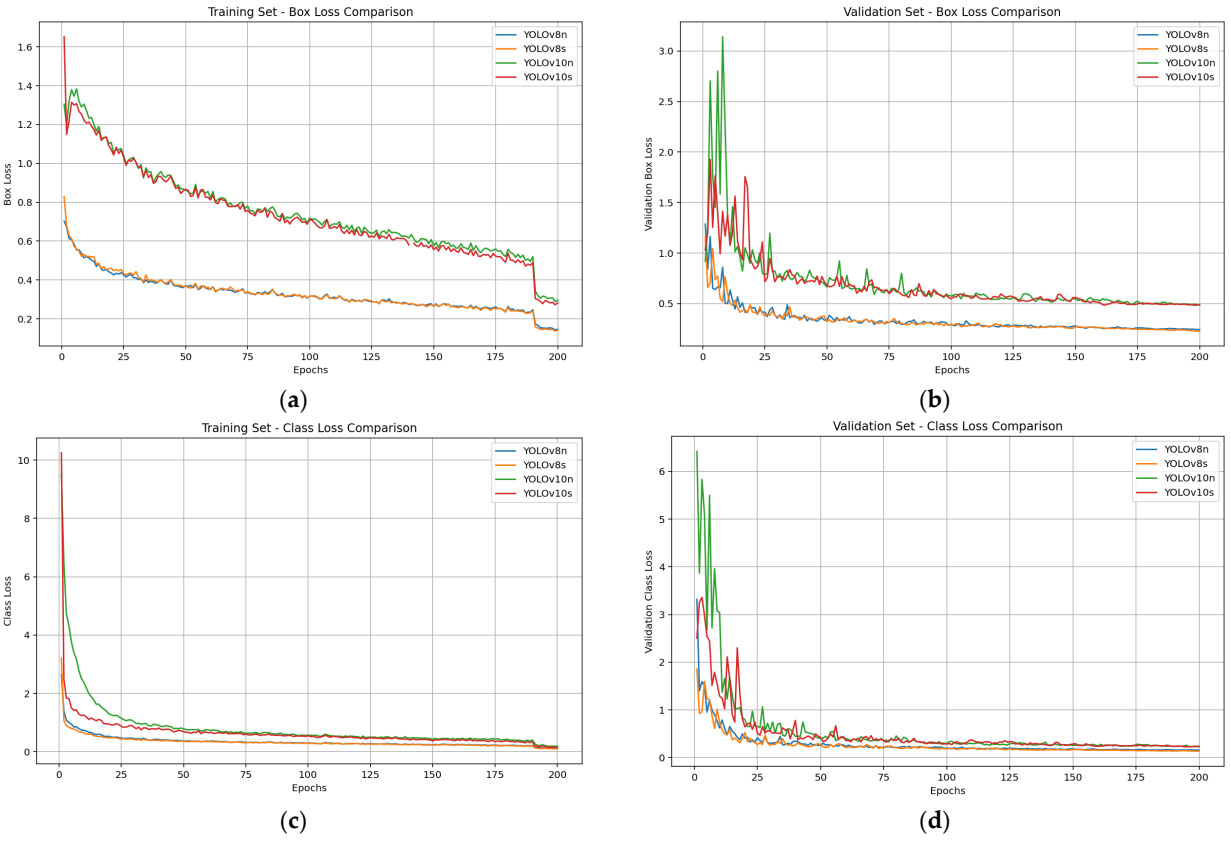
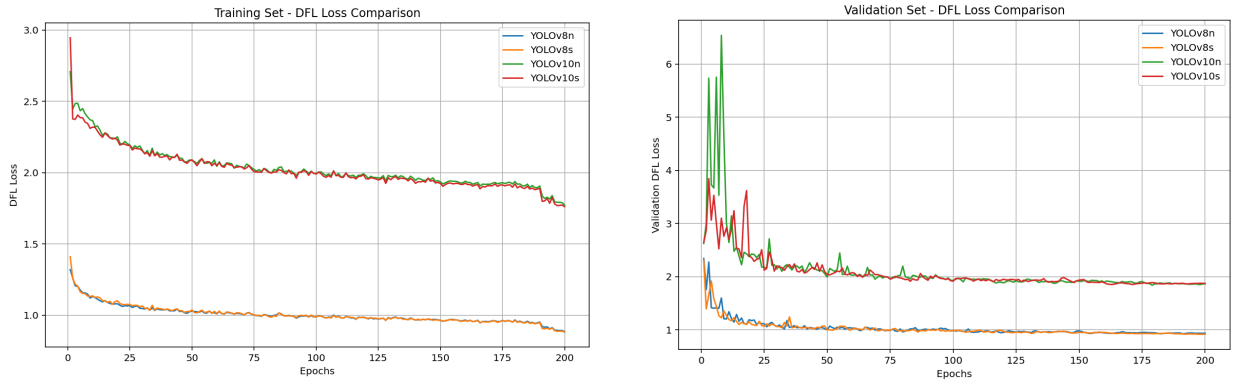
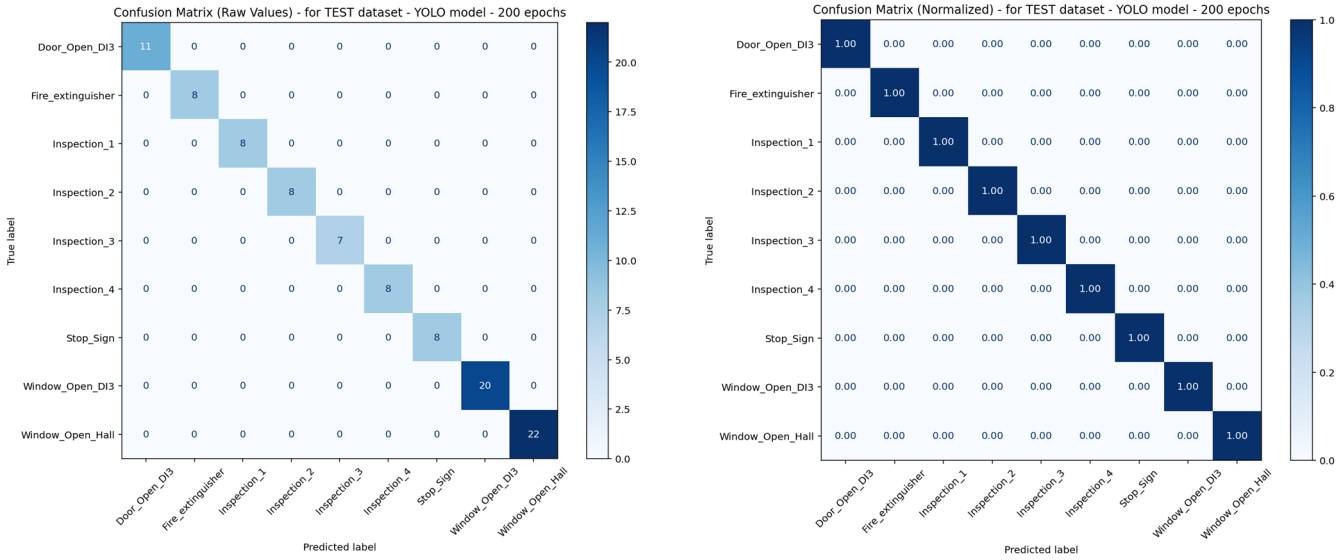


Figure 6. *Cont.*



(**e**) (**f**)

Figure 6. Loss functions comparison for models YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s. (**a**) Training set—box loss; (**b**) validation set—Box loss; (**c**) training set—class loss; (**d**) validation set—class loss; (**e**) training set—DFL loss; and (**f**) validation set—DFL loss.



(**a**) (**b**)

Figure 7. The confusion matrices for the TEST dataset obtained with models YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s. (**a**) Confusion matrix—raw values and (**b**) confusion matrix normalized.

With all four YOLO models trained for 200 epochs, detection and classification were perfect across all objects, as measured by precision, recall, F1-score, and accuracy.

As previously mentioned in Section 2.4.5, the dataset of 1033 images was split into 70% (725 images) for training, 20% (208 images) for validation, and 10% (100 images) for testing. Each image contained a single object, which was cropped from a larger image and manually annotated (Figure 8).

Table 8 presents the performance metrics for training, validation, and testing on the cropped images dataset, including the mAP@0.5 and mAP@0.5:0.95 for each YOLO model trained for 200 epochs.

All four trained YOLO models were tested on the entire set of 1033 original images (not cropped) to assess real-world performance. The corresponding confusion matrices are shown in Figure 9, which provides insights into how the models classified the objects in these images.

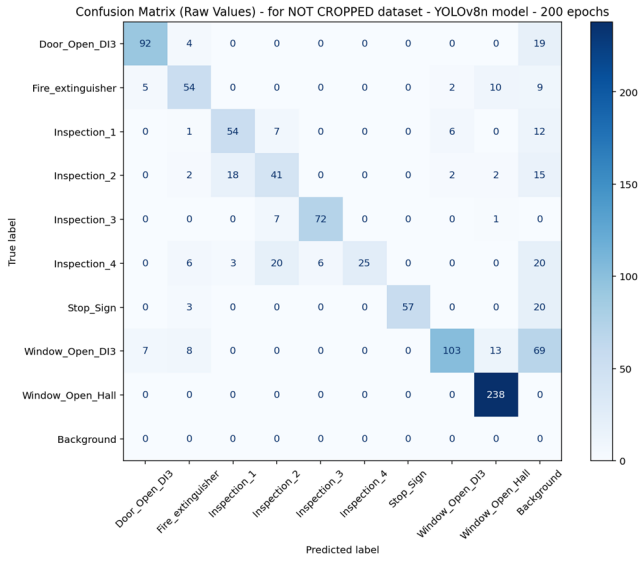


(**a**) (**b**) (**c**)

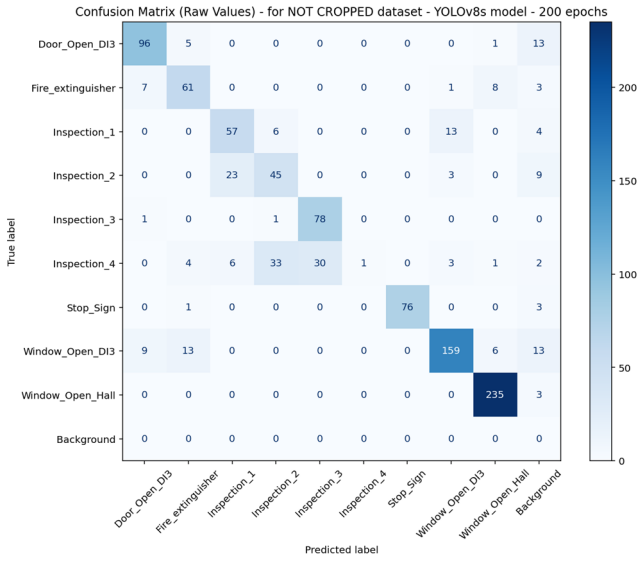
Figure 8. Image sample used for the training dataset. (**a**) Initial image; (**b**) cropped image containing the object that must be detected; and (**c**) annotated cropped image used in the training dataset.

Table 8. The metrics computed for each YOLO model trained with 200 epochs on cropped images dataset.

Model Type	Total Images	TP	FP	FN	Precision	Recall	F1-Score	Accuracy	mAP@0.5	mAP@0.5:0.95
YOLOv8n	725	725	0	0	1.000	1.000	1.000	100.000%	0.9989	0.9868
	208	208	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9783
	100	100	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9850
TOTAL	**1033**	**1033**	**0**	**0**	**1.000**	**1.000**	**1.000**	**100.000%**	**0.9989**	**0.9843**
YOLOv8s	725	725	0	0	1.000	1.000	1.000	100.000%	0.9989	0.9874
	208	208	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9783
	100	100	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9830
TOTAL	**1033**	**1033**	**0**	**0**	**1.000**	**1.000**	**1.000**	**100.000%**	**0.9989**	**0.9846**
YOLOv10n	725	721	0	4	1.000	0.996	0.998	99.638%	0.9934	0.9812
	208	208	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9805
	100	100	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9854
TOTAL	**1033**	**1029**	**0**	**4**	**1.000**	**0.997**	**0.999**	**99.746%**	**0.9955**	**0.9810**
YOLOv10s	725	724	0	1	1.000	0.998	0.999	99.802%	0.9967	0.9815
	208	208	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9789
	100	100	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9815
TOTAL	**1033**	**1032**	**0**	**1**	**1.000**	**0.999**	**0.999**	**99.861%**	**0.9967**	**0.9797**

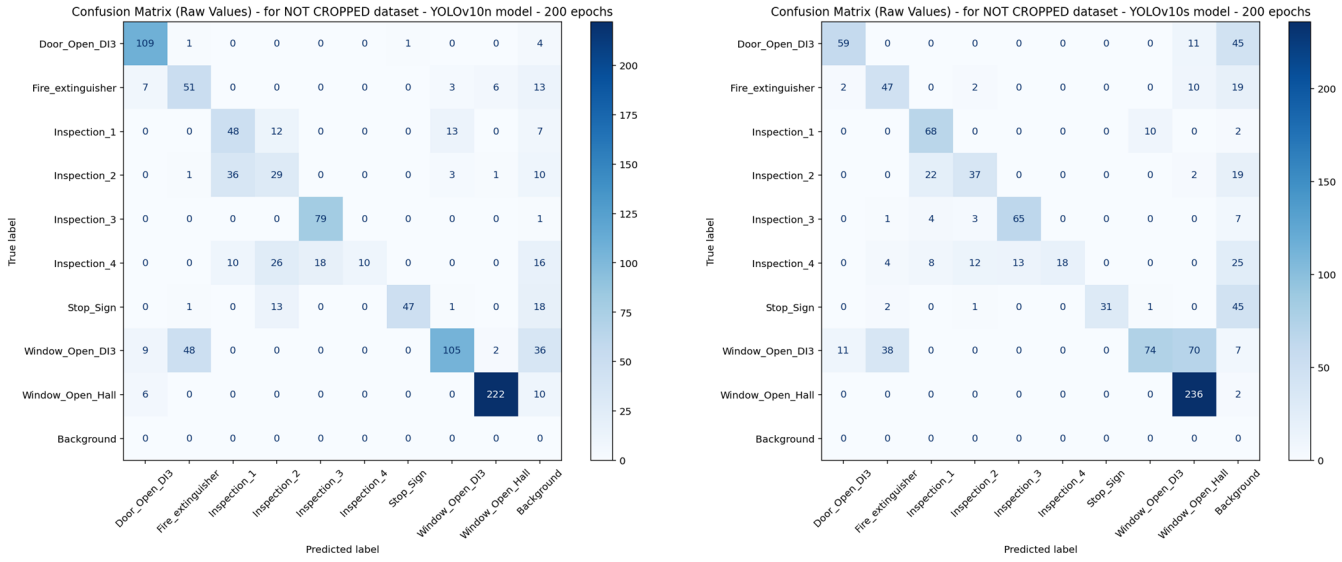


(**a**)



(**b**)

Figure 9. *Cont.*



(**c**) (**d**)

Figure 9. Confusion matrices obtained on the 1033 NOT CROPPED images dataset. (**a**) Model YOLOv8n; (**b**) model YOLOv8s; (**c**) model YOLOv10n; and (**d**) model YOLOv10s.

The computed metrics are summarized in Figure 10, where the tables were generated using Python-based evaluation scripts.

Upon analyzing their performances on the training, validation, and testing datasets (Table 8), the models exhibited an exceptional accuracy, raising concerns regarding overfitting. Overfitting in YOLO models occurs when the model learns the training data too precisely, capturing noise and specific details that reduce its generalization performance on new, unseen data. This overfitting leads to a high accuracy for the training set but a poorer performance for real-world images.

The *Patience* parameter was considered to mitigate overfitting. This parameter prevents excessive training by stopping the process when the validation performance plateaus.

Patience refers to the number of epochs to wait without improvement in validation metrics before stopping the training early [34]. In the 200-epoch training experiments, the *Patience* parameter was set to its default value of 100 epochs.

All four models were retrained with the *Patience* parameter set to 10 epochs to investigate the potential onset of overfitting before 200 epochs. Under these conditions, the training process automatically stopped at 50 epochs for the YOLOv8n and YOLOv10s models and 58 epochs for the YOLOv8s and YOLOv10n models, respectively.

All performance metrics were recomputed with these newly trained models, and the results are summarized in Table 9.

No significant differences were observed when comparing Table 9 with Table 8, suggesting that stopping early did not degrade performance. Consequently, the new models with fewer epochs were tested on the full dataset of 1033 original images to assess their generalization performance. The metrics can be observed in Table 10.

YOLOv8n model - trained with 200 epochs

Class	Total Images	TP	FP	FN	Precision	Recall	F1-Score	Accuracy
Door_Open_DI3	115	92	4	19	0.958	0.829	0.889	80.000%
Fire_extinguisher	80	54	17	9	0.761	0.857	0.806	67.500%
Inspection_1	80	54	14	12	0.794	0.818	0.806	67.500%
Inspection_2	80	41	24	15	0.631	0.732	0.678	51.250%
Inspection_3	80	72	8	0	0.900	1.000	0.947	90.000%
Inspection_4	80	25	35	20	0.417	0.556	0.476	31.250%
Stop_Sign	80	57	3	20	0.950	0.740	0.832	71.250%
Window_Open_DI3	200	103	28	69	0.786	0.599	0.680	51.500%
Window_Open_Hall	238	238	0	0	1.000	1.000	1.000	100.000%
Total	1033	736	133	164	0.800	0.792	0.790	67.806%

(**a**)

YOLOv8s model - trained with 200 epochs

Class	Total Images	TP	FP	FN	Precision	Recall	F1-Score	Accuracy
Door_Open_DI3	115	96	6	13	0.941	0.881	0.910	83.478%
Fire_extinguisher	80	61	16	3	0.792	0.953	0.865	76.250%
Inspection_1	80	57	19	4	0.750	0.934	0.832	71.250%
Inspection_2	80	45	26	9	0.634	0.833	0.720	56.250%
Inspection_3	80	78	2	0	0.975	1.000	0.987	97.500%
Inspection_4	80	1	77	2	0.013	0.333	0.025	1.250%
Stop_Sign	80	76	1	3	0.987	0.962	0.974	95.000%
Window_Open_DI3	200	159	28	13	0.850	0.924	0.886	79.500%
Window_Open_Hall	238	235	0	3	1.000	0.987	0.994	98.739%
Total	1033	808	175	50	0.771	0.868	0.799	73.246%

(**b**)

YOLOv10n model - trained with 200 epochs

Class	Total Images	TP	FP	FN	Precision	Recall	F1-Score	Accuracy
Door_Open_DI3	115	109	2	4	0.982	0.965	0.973	94.783%
Fire_extinguisher	80	51	16	13	0.761	0.797	0.779	63.750%
Inspection_1	80	48	25	7	0.658	0.873	0.750	60.000%
Inspection_2	80	29	41	10	0.414	0.744	0.532	36.250%
Inspection_3	80	79	0	1	1.000	0.988	0.994	98.750%
Inspection_4	80	10	54	16	0.156	0.385	0.222	12.500%
Stop_Sign	80	47	15	18	0.758	0.723	0.740	58.750%
Window_Open_DI3	200	105	59	36	0.640	0.745	0.689	52.500%
Window_Open_Hall	238	222	6	10	0.974	0.957	0.965	93.277%
Total	1033	700	218	115	0.705	0.797	0.738	63.396%

(**c**)

YOLOv10s model - trained with 200 epochs

Class	Total Images	TP	FP	FN	Precision	Recall	F1-Score	Accuracy
Door_Open_DI3	115	59	11	45	0.843	0.567	0.678	51.304%
Fire_extinguisher	80	47	14	19	0.770	0.712	0.740	58.750%
Inspection_1	80	68	10	2	0.872	0.971	0.919	85.000%
Inspection_2	80	37	24	19	0.607	0.661	0.632	46.250%
Inspection_3	80	65	8	7	0.890	0.903	0.897	81.250%
Inspection_4	80	18	37	25	0.327	0.419	0.367	22.500%
Stop_Sign	80	31	4	45	0.886	0.408	0.559	38.750%
Window_Open_DI3	200	74	119	7	0.383	0.914	0.540	37.000%
Window_Open_Hall	238	236	0	2	1.000	0.992	0.996	99.160%
Total	1033	635	227	171	0.731	0.727	0.703	57.774%

(**d**)

Figure 10. Metrics obtained on the 1,033 NOT CROPPED images dataset. (**a**) Model YOLOv8n; (**b**) model YOLOv8s; (**c**) model YOLOv10n; and (**d**) model YOLOv10s.

Table 9. The metrics computed for each YOLO model trained with a limited number of epochs on cropped images dataset.

Model Type	Total Images	TP	FP	FN	Precision	Recall	F1-Score	Accuracy	mAP@0.5	mAP@0.5:0.95
YOLOv8n 50 epochs	725	725	0	0	1.000	1.000	1.000	100.000%	0.9989	0.9588
	208	208	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9496
	100	100	0	0	1.000	1.000	1.000	100.000%	1.0000	0.9644
TOTAL	**1033**	**1033**	**0**	**0**	**1.000**	**1.000**	**1.000**	**100.000%**	**0.9989**	**0.9561**
YOLOv8s 58 epochs	725	718	7	0	0.988	1.000	0.994	98.611%	0.9945	0.9645
	208	207	1	0	0.993	1.000	0.997	99.306%	1.0000	0.9590
	100	97	3	0	0.967	1.000	0.980	95.833%	0.9817	0.9519
TOTAL	**1033**	**1022**	**11**	**0**	**0.987**	**1.000**	**0.993**	**98.472%**	**0.9934**	**0.9610**
YOLOv10n 58 epochs	725	721	0	4	1.000	0.996	0.998	99.580%	0.9928	0.9564
	208	207	1	0	0.993	1.000	0.997	99.306%	0.9919	0.9386
	100	99	0	1	1.000	0.995	0.997	99.495%	0.9945	0.9583
TOTAL	**1033**	**1027**	**1**	**5**	**0.999**	**0.997**	**0.998**	**99.519%**	**0.9929**	**0.9520**
YOLOv10s 50 epochs	725	712	1	12	0.998	0.978	0.988	97.622%	0.9706	0.9356
	208	205	0	3	1.000	0.983	0.991	98.333%	0.9817	0.9451
	100	99	0	1	1.000	0.984	0.991	98.413%	0.9835	0.9503
TOTAL	**1033**	**1016**	**1**	**16**	**0.999**	**0.980**	**0.989**	**97.848%**	**0.9740**	**0.9374**

Table 10. New metrics obtained on the 1033 NOT CROPPED images dataset.

Model Type	Total Images	TP	FP	FN	Precision	Recall	F1-Score	Accuracy
YOLOv8n—50 epochs	1033	800	169	064	0.782	0.898	0.822	72.827%
YOLOv8s—58 epochs	1033	691	253	89	0.701	0.811	0.738	63.432%
YOLOv10n—58 epochs	1033	595	236	202	0.691	0.661	0.649	52.266%
YOLOv10s—50 epochs	1033	664	172	197	0.744	0.727	0.723	59.478%

3.3. Exporting YOLO Models to ONNX and Converting to HEF

The ultimate goal of training the YOLO models is their deployment on an autonomous mobile platform for the intelligent and flexible inspection of indoor environments. To achieve this, Ultralytics provides functionalities for exporting trained models into various formats, allowing them to be deployed across different devices and platforms [36]. The models are optimized for size and speed performance during the export process.

After comparing the results in Table 10 with those in Figure 9, no signs of overfitting were detected; therefore, the models trained for 200 epochs were selected for further use. These models were first exported in ONNX format and then converted into HEF to run on the Raspberry Pi AI Kit [37]. The conversion process from ONNX to HEF involves complex stages, including node remapping, optimization, calibration, and compilation, which adapt the model architecture and operations to Hailo's hardware. The following two different optimization strategies were employed: one using the 208-image validation dataset and another using COCO TRAIN2017, which contains 118,287 images, to evaluate whether the larger dataset could enhance performance compared to our dataset.

As a result, eight distinct HEF files were generated and deployed on the Raspberry Pi AI Kit for further testing and evaluation

3.4. Testing and Validation of the Artificial Vision System in Real-World Conditions Using the Autonomous Mobile Platform

For the final testing and validation phase, a real-world an inspection mission using the autonomous mobile platform was designed. The objects targeted for detection belonged to

the following five classes: two inspection points located by IP markers (*Inspection_1* and *Inspection_3*); one object that must exist in the inspected area (*Fire_Extinguisher*); one class for open door identification (*Door_Open_DI3*); and one class for open window detection in hall areas (*Window_Open_Hall*).

The images captured by the Raspberry Pi Camera of the artificial vision system during the inspection mission were stored to ensure a consistent comparison across models. From these, a subset of 260 images was selected for analysis, including images with challenging conditions such as shadows, partial views of the objects of interest, and images with no objects. These images were randomly renamed.

The initial analysis computed standard performance metrics—precision, recall, F1-score, and accuracy—for the following eight models: four models optimized during the ONNX-to-HEF conversion using the initial 208-image validation dataset (denoted as YOLOv8n_VAL_SET, YOLOv8s_VAL_SET, YOLOv10n_VAL_SET, and YOLOv10s_VAL_SET) and four models optimized using the COCO TRAIN2017 dataset (denoted as YOLOv8n_COCO_SET, YOLOv8s_COCO_SET, YOLOv10n_COCO_SET, and YOLOv10s_COCO_SET).

Object detection on the Raspberry Pi AI Kit was performed using the *deepview-validator* Python script (version 3.2.2-post14), which has been updated for HEF support [38]. For each image, prediction scores were computed and bounding boxes were drawn, with each bounding box annotated with the predicted class name and the corresponding score (Figure 11).

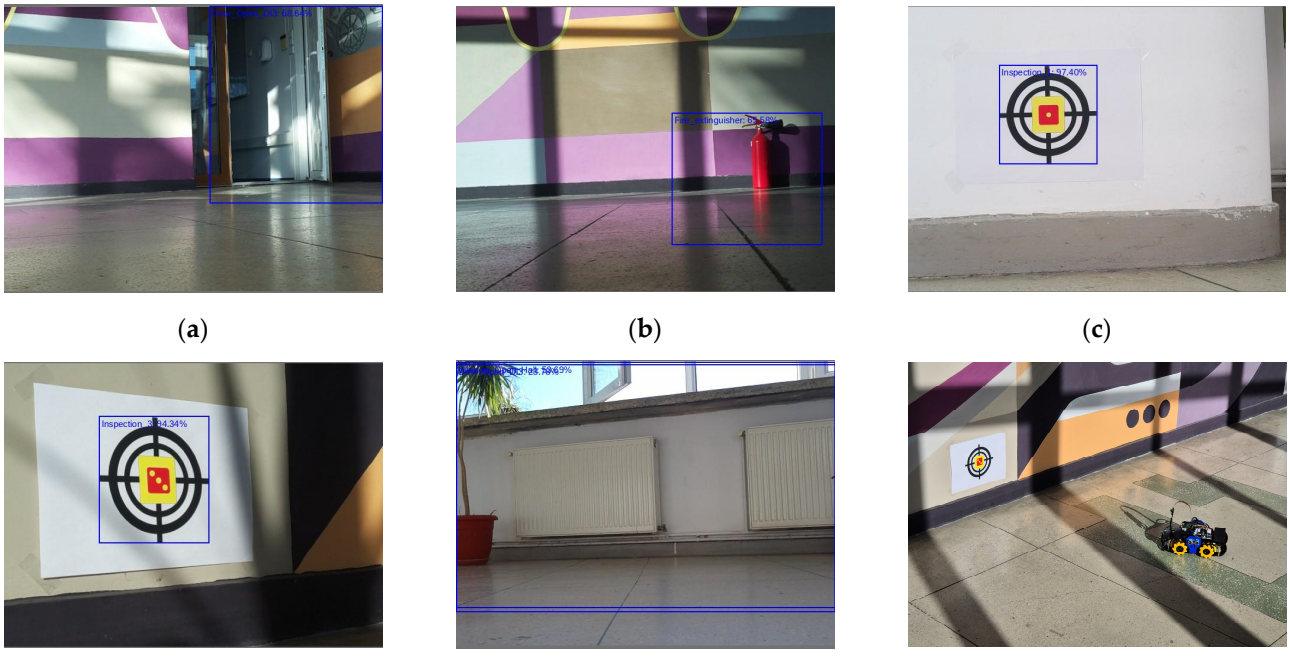
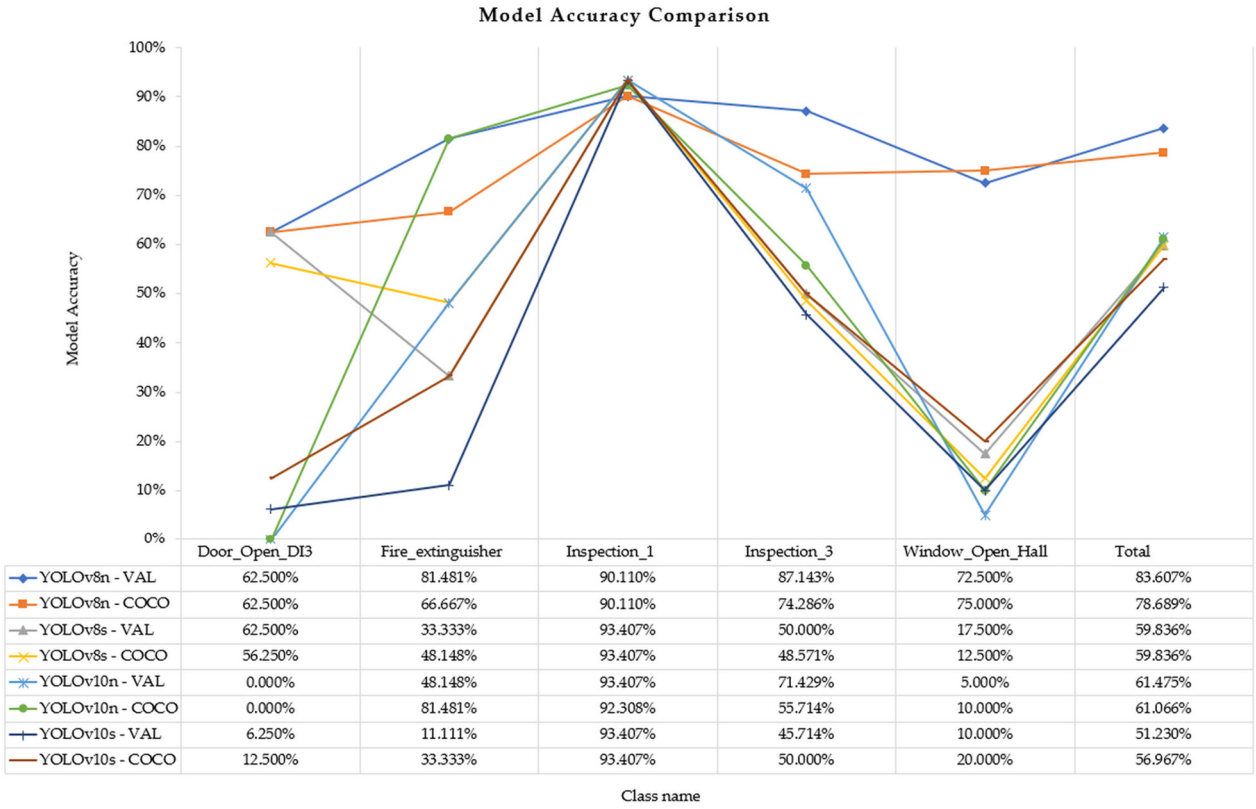


Figure 11. Image samples from the inspection mission. (**a**) Door_Open_DI3; (**b**) Fire_extinguisher; (**c**) Inspection_1; (**d**) Inspection_3; (**e**) Window_Open_Hall; and (**f**) autonomous mobile platform approaching the *Inspection_3* IP marker during the mission.

The prediction scores were stored in .txt files, where each row contained the image file number alongside scores for the nine possible object classes that the model could classify. These data were imported into Excel for processing, where the true positives (TPs), false positives (FPs), and false negatives (FNs) were computed. Subsequently, the following key performance metrics were computed: precision, recall, F1-score, and accuracy.

Figure 12 presents a comparative analysis of accuracy across the YOLO models.



	Door_Open_DI3	Fire_extinguisher	Inspection_1	Inspection_3	Window_Open_Hall	Total
YOLOv8n - VAL	62.500%	81.481%	90.110%	87.143%	72.500%	83.607%
YOLOv8n - COCO	62.500%	66.667%	90.110%	74.286%	75.000%	78.689%
YOLOv8s - VAL	62.500%	33.333%	93.407%	50.000%	17.500%	59.836%
YOLOv8s - COCO	56.250%	48.148%	93.407%	48.571%	12.500%	59.836%
YOLOv10n - VAL	0.000%	48.148%	93.407%	71.429%	5.000%	61.475%
YOLOv10n - COCO	0.000%	81.481%	92.308%	55.714%	10.000%	61.066%
YOLOv10s - VAL	6.250%	11.111%	93.407%	45.714%	10.000%	51.230%
YOLOv10s - COCO	12.500%	33.333%	93.407%	50.000%	20.000%	56.967%

Figure 12. Model accuracy comparison.

The results indicate that the YOLOv8n model optimized using the 208-image validation dataset during the HEF conversion achieved the highest accuracy, reaching 83.607%. The following section provides a more detailed discussion regarding class performance and model comparisons.

Object detection systems are critical to modern indoor inspection protocols, particularly in applications requiring high precision, such as energy efficiency monitoring, security, and compliance. However, these systems are susceptible to object detection and recognition errors.

These errors typically manifest in the following two forms: false negatives (FNs), where an object is present but not detected, and false positives (FPs), where an object is incorrectly identified in an image. FN and FP detections directly affect such systems' reliability, introducing operational inefficiencies and potential risks.

For an autonomous mobile platform designed for intelligent and flexible indoor environment inspection, FN and FP detections pose several challenges depending on the category of objects being identified. These categories include IP markers, final position markers, target objects for presence verification, and structural elements for anomaly detection.

The accurate detection of IP markers is crucial for ensuring the correct and complete execution of the inspection process. FN errors (failure to identify an IP) disrupt the inspection sequence by omitting IPs, making it impossible to verify specific areas. Conversely, FP errors (the misidentification of an IP marker or recognition of an incorrect marker) can lead to erroneous or incomplete inspections, resulting in additional time and energy consumption. Given these implications, achieving a high detection accuracy for IP markers is essential. The test results presented in Figure 11 highlight a superior performance for these classes compared to others. This improvement can be attributed to the consistent visual characteristics of IP markers, which exhibit minimal variability.

Regarding target objects for presence verification, minimizing FP and FN responses is fundamental to maintaining the reliability of object detection systems in automated indoor inspection applications. While FP detections primarily lead to inefficiencies and unnecessary interventions, FN detections can have more serious consequences, impacting safety, security, and compliance. Addressing these challenges requires the continuous refinement of detection algorithms and implementation of adaptive learning techniques to enhance model accuracy across diverse indoor environments. Factors commonly encountered in real-world scenarios, such as variations in lighting conditions, object occlusions, and clutter levels, must be carefully considered. These conditions should be adequately represented in the training, validation, and testing datasets to ensure robustness.

Concerning structural elements for anomaly detection, FN errors, such as failing to detect an open window or door, can have significant consequences. Undetected openings in secured spaces may lead to unauthorized access or environmental contamination, compromising safety and product integrity. Furthermore, failing to detect such anomalies can result in substantial energy losses as heating or cooling systems continue to operate under the assumption of a sealed environment. Meanwhile, FP detections may trigger false alarms, leading to unnecessary human interventions and operational inefficiencies.

The second analysis focused on the computational performance of the models, considering the number of images that the models could process per second, called FPS—frames per second (higher values are better)—and latency, representing the time (in milliseconds) it took the models to process a single image (lower values are better).

If image processing for object detection and recognition was performed on the server instead of the embedded system, the system's performance in terms of FPS and latency would also be influenced by additional factors, such as network latency (due to communication with the server), battery power, and potential interference from other electronic devices.

This study performed the object detection and recognition process entirely on an embedded system consisting of a Raspberry Pi 5 and the Raspberry Pi AI Kit accelerator. Performance metrics were evaluated by comparing object detection executed on the Raspberry Pi 5 CPU versus detection utilizing the Raspberry Pi AI Kit with the Hailo-8L accelerator. This analysis emphasizes the advantages of employing the Raspberry Pi AI Kit accelerator, introduced in 2024, in contrast to running the YOLO model solely on the Raspberry Pi 5 CPU.

The following two model formats were compared to analyze speed: ONNX (executed on Raspberry Pi CPU) vs. HEF (executed on Raspberry Pi AI Kit with Hailo-8L accelerator). The tests were performed on Raspberry Pi 5 Model B Rev 1.0. For the ONNX format, a custom Python script based on the OpenCV and Ultralytics libraries was used, and for HEF, the *hailortcli* command line utility was used (version 4.18.0) [39,40].

Plots for FPS and latency are presented in Figure 13.

Based on the above data, we conclude that the Raspberry Pi AI Kit significantly enhanced processing speed (FPS) performance and latency reduction. Notably, the maximum FPS speedup factor achieved was approximately 15× compared to execution on the Raspberry Pi CPU.

Considering all the evaluated metrics, the YOLOv8n model was selected as the optimal choice for deployment on the autonomous mobile platform, and the artificial vision system was validated.

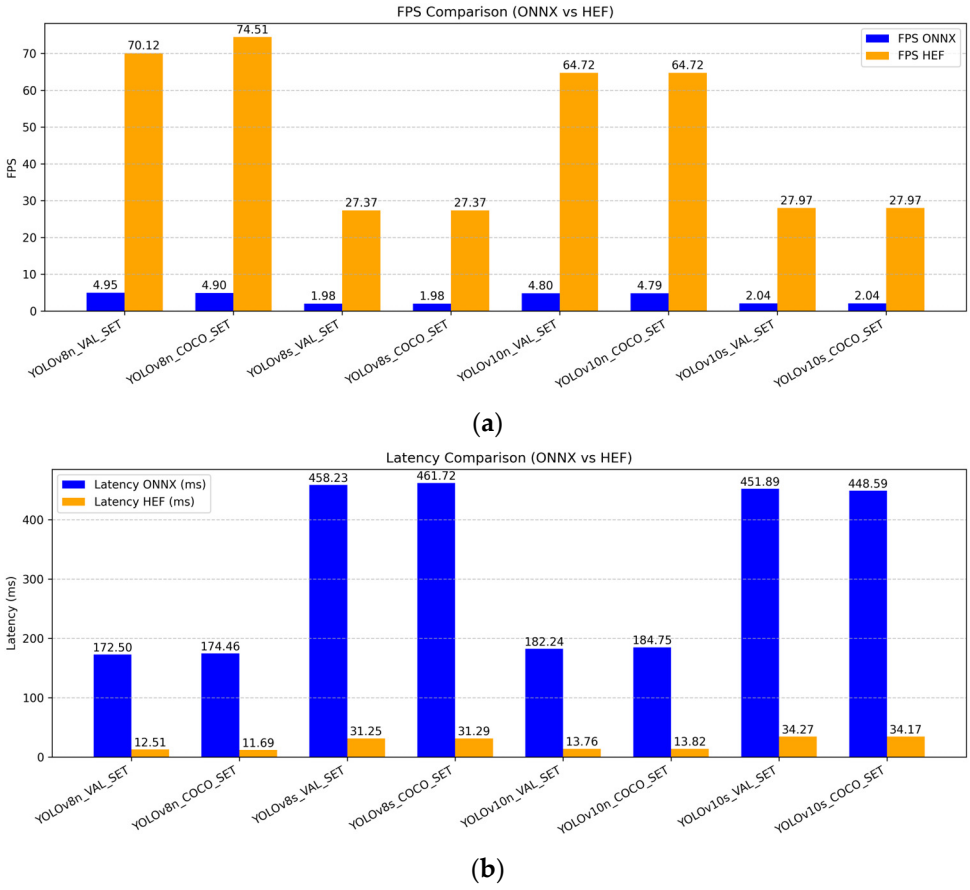


Figure 13. Object detection speed comparison ONNX vs HEF. (**a**) Frames per second (FPS) and (**b**) latency.

4. Discussion

4.1. AI Model Selection

The first challenge was selecting an appropriate object detection and classification model for the artificial vision system.

The choice of the Convolutional Neural Network model was based on a comprehensive analysis of the most high-performance solutions currently available, considering the specific requirements of the inspection application. These requirements included the following: a low computational cost, as the model would be deployed on an embedded system; a high processing speed, ensuring real-time object detection and classification; a high accuracy, to reliably identify objects in various conditions; and ease of training and implementation to facilitate model adaptation and optimization.

Given these constraints, a one-stage detection model was preferred, leading to the selection of the YOLO family. Several versions of YOLO were considered. Initially, YOLOv11, the latest version, was evaluated alongside YOLOv8, a state-of-the-art model, to compare their performances in the inspection task. However, at the time of the study, YOLOv11 lacked support for direct conversion to HEF, which was necessary for deployment on the Raspberry Pi AI Kit. Consequently, YOLOv10 was chosen for further analysis instead.

Due to hardware constraints on the available DELL Precision 3551 laptop, the experiments were conducted using only the nano (n) and small (s) variants of YOLO models. Larger versions, including medium (m), large (l), and extra-large (x), were excluded, as their size and complexity significantly impact training time, speed, accuracy, and resource requirements.

The training times for 200 epochs were recorded as follows: YOLOv8n took 9741.42 s (≈2 h, 42 min, 24 s); YOLOv8s took 22,450.20 s (≈6 h, 14 min, 10 s); YOLOv10n took 12,340.80 s (≈3 h, 25 min, 40 s); and YOLOv10s took 111,172.00 s (≈30 h, 52 min, 52 s).

4.2. Image Dataset Preparation

For the selection of images used for training, validation, and testing, smaller regions containing the target objects were cropped from larger images. This approach facilitated a more efficient and effective model training process, as evidenced by the test results on 100 newly introduced images that were not used during training or validation. However, a key objective was to evaluate model performance on the 1,033 original images. The results further confirm the effectiveness of the YOLO models [41,42].

4.3. Model Performance After Training and Validation

It is well established that Mean Average Precision (mAP) is the standard metric for evaluating object detection models such as YOLOv8 and YOLOv10. In this study, mAP was computed using the cropped image dataset, where annotations included ground truth bounding boxes and class labels. However, for the original (non-cropped) image set, mAP was not calculated, as this would require the comprehensive annotation of all objects within the images. Instead, alternative performance metrics were evaluated, including precision, recall, F1-score, and accuracy. It is also important to note that accuracy in object detection may be less informative due to class imbalances and the complexity of detecting multiple objects within an image. Therefore, as a future research direction, a more comprehensive analysis should be conducted using images containing multiple objects which are fully annotated to enable the calculation of mAP.

Object localization accuracy for IP markers is significantly more critical for managing the autonomous mobile system's orientation algorithm than the localization accuracy of objects that must be present within the inspected area. However, classification accuracy remains equally important in both cases.

As can be observed in Figure 4, the differences in the box loss and DFL (Distribution Focal Loss) trends between the nano and small model types are relatively minor. However, the small model achieves slightly better results, which aligns with its more advanced architecture. The nano model shows more fluctuations in its loss curve across epochs, likely due to its lower capacity and greater sensitivity to optimization noise. In contrast, the small model demonstrates a smoother and more stable convergence.

The differences in box loss and DFL loss trends are more pronounced between YOLOv8 and YOLOv10. These differences can be attributed to the two models' architectural modifications and training process variations. YOLOv10 has architectural changes compared to YOLOv8 regarding the backbone network, feature extraction, and loss function computation. These changes may introduce variations in how the models learn bounding box regression. YOLOv8n, for example, starts with a relatively low box loss value (0.70299 in epoch 1), decreases steadily, and reaches very low values in later epochs, with a final loss of 0.14613 at epoch 200. This trend indicates fast and stable convergence. YOLOv10n starts with a higher box loss (1.30431 in epoch 1), indicating that the initial predictions are worse than those of YOLOv8n. The box loss decreases slower in the early epochs but eventually catches up, achieving a final loss of 0.29594 at epoch 200. This case suggests slower convergence and a more complex loss function or learning process.

All four models demonstrate a good performance regarding precision, recall, mAP@0.5, and mAP@0.5:0.95 (Figure 5). Observing the saturation points, YOLOv8s and YOLOv10s reach peak performance earlier than their nano counterparts. YOLOv10n displays more fluctuations in its learning curve, suggesting a need for more refined hyperparameter

tuning. The marginal differences between the models in their final epochs suggest that all models achieve strong generalization, with the choice between them being dependent on other factors such as computational efficiency and model size.

A comparative analysis of YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s conducted on the original, uncropped images demonstrates that YOLOv8s provides the best balance of precision, recall, and F1-score (Figure 9), making it the preferred model for object detection tasks requiring a high sensitivity and accuracy. YOLOv8n also performs well in precision, while YOLOv10 models exhibit a relatively lower performance across all metrics. Also, a good performance is obtained by the YOLOv8n model trained with 50 epochs (Table 10).

4.4. Model Performance in Real-World Tests

We evaluated the accuracy of four YOLO models—YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s—optimized on the following two distinct datasets: the VAL dataset (208 images used for validation) and the COCO Train2017.

YOLOv8n consistently demonstrated the highest accuracy across both datasets, achieving 83.607% on the VAL dataset and 78.689% on COCO. It showed a strong performance in all classes, especially *Inspection_1* and *Inspection_3*. **YOLOv10n** had a moderate performance with a 61.475% accuracy on VAL and 61.066% on COCO, but it struggled with some classes, particularly *Window_Open_Hall* and *Door_Open_DI3*. **YOLOv8s** and **YOLOv10s** showed a significant drop in accuracy on both datasets, with a 59.836% accuracy for YOLOv8s on both datasets and 51.230% on VAL and 56.967% on COCO for YOLOv10s. Both small models had difficulties in detecting more complex objects like *Fire_extinguisher* and *Window_Open_Hall.*

The observed results can be explained and confirmed by the key features and particularities of the YOLOv8 and YOLOv10 architectures.

YOLOv8 is known for having a better computational efficiency, focusing on improving small object detection, and a more refined architecture for general detection tasks. YOLOv8n, the lightest version, performs well due to its better backbone and efficient feature extraction, even though it is computationally more affordable. It achieves a relatively high accuracy, especially in classes like *Inspection_1* and *Inspection_3*, where clear object distinction helps the model to generalize well.

YOLOv10 introduces further optimizations focusing on a higher accuracy and robustness for complex tasks. However, its complexity and some trade-offs between performance and real-time speed may negatively affect the detection of small or occluded objects, as observed for models like YOLOv10n and YOLOv10s. YOLOv10n's architecture struggles with objects such as *Door_Open_DI3* and *Window_Open_Hall*.

Classes like *Inspection_1* and *Inspection_3* include relatively simple and easily distinguishable objects, which explains the strong performance observed across models.

Window_Open_Hall and *Door_Open_DI3* involve more complex objects, because various other items can appear behind glass windows or door frames, making their classification particularly challenging. The lower accuracy obtained for these classes can also be attributed to the fact that these objects are only partially visible in the images captured by the camera at the current height of the autonomous mobile platform. Repositioning the camera to capture the entire object within the image would improve the detection results. Therefore, future development will focus on either positioning the camera on a higher support to ensure that objects of interest are fully captured within the frame or introducing a scanning mode where the camera is oriented horizontally and vertically. However, the latter approach would complicate the platform's navigation algorithm.

In conclusion, YOLOv8n is the top performer, achieving the highest accuracy across both optimization datasets and demonstrating a strong performance across individual classes. As a result, this model has been selected for implementation on the autonomous mobile platform for intelligent and flexible indoor environment inspection.

It is well established that the Ultralytics YOLOv8 and YOLOv10 models are pre-trained on the COCO 2017 dataset, which comprises 118,287 images in its training set and 5000 images in its validation set, spanning 80 object classes. When applying transfer learning using these pre-trained models, the number of training images per class required for effective performance depends on several key factors, including object complexity and variability, background diversity, and the intended performance objectives, ranging from proof-of-concept validation to robust generalization across various real-world conditions.

As a general guideline, from 50 to 100 images per class may be sufficient for proof-of-concept implementations. However, to ensure strong generalization across diverse indoor environments, including varying lighting conditions, occlusions, and viewing angles, a substantially larger dataset between 500 and 2000 images per class is typically recommended.

In the present study, the dataset used for training consisted of 1033 images selected to validate the concept and evaluate the performance of different YOLO models. While the models demonstrated a high accuracy on this dataset, these results may not fully reflect their robustness in more complex and heterogeneous real-world inspection scenarios.

Future work will focus on expanding the training dataset by incorporating additional images collected during inspection missions. Furthermore, additional training sessions will be conducted to meet the performance requirements specific to the targeted indoor inspection processes.

5. Conclusions

This paper presents the implementation, testing, and validation of an autonomous mobile platform designed for the intelligent and flexible inspection of indoor environments.

The novelty of the research stems from the integration of a new type of AI accelerator into an embedded computer vision system, conferring the platform with the attribute of intelligent inspection based on an artificial vision system. The Raspberry Pi AI Kit, launched in 2024, based on the Hailo-8L chip, represents one of the latest advancements in edge computing for embedded artificial vision applications. This system runs a convolutional neural network model (YOLO) to enable real-time object detection and classification. Furthermore, the platform incorporates a decision-making algorithm to adjust its movement trajectory upon detecting predefined IP markers.

Another key contribution is the flexible system architecture, which allows for seamless adaptation to different mobile inspection platforms. The selection of the AI accelerator is driven by the necessity of implementing the artificial vision system on an embedded system capable of being integrated into an autonomous mobile platform. The system's flexibility is ensured by its high mobility, allowing the same equipment to be deployed across various locations while maintaining operational efficiency, enabling rapid, easily configurable, and customizable inspection processes tailored to the specific characteristics of different indoor environments.

Another significant aspect of this study concerns the training procedure employed for YOLO object detection models. Instead of training on full-frame images, smaller regions containing only objects of interest are extracted from the original images. These cropped images are used for training to enable the neural network to learn object-specific features more effectively. Following the training phase, the model is evaluated on the full original images to assess its ability to correctly detect and localize objects in complex environments.

This approach enhances the model's generalization capability, ensuring accurate detection across diverse scenarios.

The main advantages of this method include improved feature learning, while the model focuses exclusively on objects of interest without being influenced by background noise; reduced dataset requirements, considering that multiple training images can be generated from a single original image, minimizing the need for extensive labeled datasets; and a faster training process, because using smaller, relevant regions speeds up convergence and optimizes computational resources.

However, it is important to consider that background variations and lighting conditions can significantly impact real-world detection performance. These factors are present in the images used for the testing and validation phases to ensure a robust performance in real-world scenarios.

Images captured by the autonomous mobile platform during an inspection mission are stored to evaluate the YOLO models under real-world conditions. These images are used to test four YOLO models (YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s), each with two optimized variants resulting from the conversion process to HEF, which can be run on the Hailo-8L AI accelerator.

The test results highlight YOLOv8n as the best-performing model, achieving an accuracy of 86.607% when optimized with the VAL dataset and 78.689% with the COCO TRAIN2017 dataset.

From a computational performance perspective, the significant contribution of the AI accelerator is evident, leading to an approximately 15-fold increase in FPS compared to running the model on the Raspberry Pi CPU.

Considering the results obtained, the hardware requirements, and the specificities of the inspection process, the YOLOv8n model is validated for implementation in the artificial vision system.

After the tests, it is determined that the system can be improved in the following areas: **the hardware structure of the platform**, including the repositioning of the camera at a higher height or adding the capability to orient it vertically; **the navigation algorithm**, as currently, the system does not return automatically to its initial starting location, but different variants for returning are being tested; route recording, navigation based on SLAM (Simultaneous Localization and Mapping), and navigation based on visual landmarks (arrow markers for the direction of movement of the system and the allocation of a marker for the START point); **enhancing the performance of the object detection and classification model** by *expanding the training set* with additional relevant images, *training the network using augmentation techniques*, which are crucial for improving the robustness and performance of YOLO models by introducing variability into the training data, thus helping the model generalize better to unseen data, and *testing various optimization algorithms* for training (SGD, Adam, AdamW, NAdam, RAdam, and RMSProp) [35]; and **optimizing the inspection platform for other scenarios**, such as those in industrial environments. All of these represent future research directions that the development team is considering.

The proposed solution advances current inspection practices through a flexible, modular architecture that allows for quick deployment in diverse indoor settings without major infrastructure changes. Its compact, embedded design enables local image processing, reducing latency and increasing reliability, even in low-connectivity environments.

Despite these advancements, several practical limitations must be acknowledged. The detection performance is influenced by real-world factors such as lighting variability, object occlusion, background clutter, and the physical constraints of the platform (e.g., camera angle and movement stability). Additionally, the training dataset used in this initial phase is relatively small and designed primarily for proof-of-concept validation. Expanding the

dataset and refining the model are necessary for full operational deployment to ensure robust generalization in diverse inspection scenarios.

In conclusion, the proposed system provides a viable and scalable approach to indoor autonomous inspection. With continued dataset enrichment and algorithmic optimization, the platform holds significant potential for real-world applications in security, facility monitoring, energy efficiency assessment, and compliance verification.

Author Contributions: Conceptualization, M.C.L., L.C. and A.L.B.; methodology, M.C.L., L.C. and A.L.B.; software, M.C.L. and A.L.B.; validation, M.C.L., L.C. and A.L.B.; formal analysis, M.C.L., L.C. and A.L.B.; investigation, M.C.L., L.C. and A.L.B.; resources, M.C.L. and L.C.; data curation, M.C.L.; writing—original draft preparation, M.C.L. and L.C.; writing—review and editing, M.C.L. and L.C.; visualization, M.C.L.; supervision, L.C.; project administration, M.C.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Sanchez-Cubillo, J.; Del Ser, J.; Martin, J.L. Toward Fully Automated Inspection of Critical Assets Supported by Autonomous Mobile Robots, Vision Sensors, and Artificial Intelligence. *Sensors* **2024**, *24*, 3721. [CrossRef]
2. de M. Santos, R.C.C.; Silva, M.C.; Santos, R.L.; Klippel, E.; Oliveira, R.A.R. Towards Autonomous Mobile Inspection Robots Using Edge AI. In Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023), Prague, Czech Republic, 24–26 April 2023; SCITEPRESS–Science and Technology Publications, Lda.: Setúbal, Portugal, 2023; Volume 1, pp. 555–562.
3. Pearson, E.; Szenher, P.; Huang, C.; Englot, B. Mobile Manipulation Platform for Autonomous Indoor Inspections in Low-Clearance Areas. In Proceedings of the ASME 2023 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE2023, Boston, MA, USA, 20–23 August 2023.
4. Halder, S.; Afsari, K. Robots in Inspection and Monitoring of Buildings and Infrastructure: A Systematic Review. *Appl. Sci.* **2023**, *13*, 2304. [CrossRef]
5. Macaulay, M.O.; Shafiee, M. Machine learning techniques for robotic and autonomous inspection of mechanical systems and civil infrastructure. *Auton. Intell. Syst.* **2022**, *2*, 8. [CrossRef]
6. Dai, Y.; Kim, D.; Lee, K. An Advanced Approach to Object Detection and Tracking in Robotics and Autonomous Vehicles Using YOLOv8 and LiDAR Data Fusion. *Electronics* **2024**, *13*, 2250. [CrossRef]
7. Hütten, N.; Alves Gomes, M.; Hölken, F.; Andricevic, K.; Meyes, R.; Meisen, T. Deep Learning for Automated Visual Inspection in Manufacturing and Maintenance: A Survey of Open-Access Papers. *Appl. Syst. Innov.* **2024**, *7*, 11. [CrossRef]
8. Rane, N. YOLO and Faster R-CNN Object Detection for Smart Industry 4.0 and Industry 5.0: Applications, Challenges, and Opportunities. 25 October 2023. Available online: https://ssrn.com/abstract=4624206 (accessed on 17 January 2025). [CrossRef]
9. Toman, R.; Rogala, T.; Synaszko, P.; Katunin, A. Robotized Mobile Platform for Non-Destructive Inspection of Aircraft Structures. *Appl. Sci.* **2024**, *14*, 10148. [CrossRef]
10. Rea, P.; Ottaviano, E. Hybrid Inspection Robot for Indoor and Outdoor Surveys. *Actuators* **2023**, *12*, 108. [CrossRef]
11. Bai, C.; Bai, X.; Wu, K. A Review: Remote Sensing Image Object Detection Algorithm Based on Deep Learning. *Electronics* **2023**, *12*, 4902. [CrossRef]
12. Song, Q.; Zhou, Z.; Ji, S.; Cui, T.; Yao, B.; Liu, Z. A Multiscale Parallel Pedestrian Recognition Algorithm Based on YOLOv5. *Electronics* **2024**, *13*, 1989. [CrossRef]
13. TSCINBUNY. Available online: https://tscinbuny.com/products/tscinbuny-esp32-robot-for-arduino-uno-starter-kit-programmable-robot-educational-kit-4wd-60mm-omni-directional-wheel-chassis-with-wifi-app-obstacle-avoidance-line-tracking-smart-car-set (accessed on 17 January 2025).
14. Raspberry Pi 5. Available online: https://www.raspberrypi.com/products/raspberry-pi-5/ (accessed on 17 January 2025).

15. Raspberry Pi Camera Module 3. Available online: https://www.raspberrypi.com/products/camera-module-3/ (accessed on 17 January 2025).
16. Raspberry Pi AI Kit(Hailo-8L) vs. Coral USB Accelerator vs. Coral M.2 Accelerator with Dual Edge TPU. Available online: https://www.seeedstudio.com/blog/2024/07/16/raspberry-pi-ai-kit-vs-coral-usb-accelerator-vs-coral-m-2-accelerator-with-dual-edge-tpu/ (accessed on 17 January 2025).
17. Raspberry Pi AI Kit. Available online: https://www.raspberrypi.com/products/ai-kit/ (accessed on 17 January 2025).
18. Lin, L.; Guo, J.; Liu, L. Multi-scene application of intelligent inspection robot based on computer vision in power plant. *Sci. Rep.* **2024**, *14*, 10657. [CrossRef]
19. Serey, J.; Alfaro, M.; Fuertes, G.; Vargas, M.; Durán, C.; Ternero, R.; Rivera, R.; Sabattin, J. Pattern Recognition and Deep Learning Technologies, Enablers of Industry 4.0, and Their Role in Engineering Research. *Symmetry* **2023**, *15*, 535. [CrossRef]
20. Pavel, M.I.; Tan, S.Y.; Abdullah, A. Vision-Based Autonomous Vehicle Systems Based on Deep Learning: A Systematic Literature Review. *Appl. Sci.* **2022**, *12*, 6831. [CrossRef]
21. Li, Z.; Wang, Y.; Zhang, N.; Zhang, Y.; Zhao, Z.; Xu, D.; Ben, G.; Gao, Y. Deep Learning-Based Object Detection Techniques for Remote Sensing Images: A Survey. *Remote Sens.* **2022**, *14*, 2385. [CrossRef]
22. Trigka, M.; Dritsas, E. A Comprehensive Survey of Machine Learning Techniques and Models for Object Detection. *Sensors* **2025**, *25*, 214. [CrossRef]
23. Carranza-García, M.; Torres-Mateo, J.; Lara-Benítez, P.; García-Gutiérrez, J. On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data. *Remote Sens.* **2021**, *13*, 89. [CrossRef]
24. Ramalingam, B.; Hayat, A.A.; Elara, M.R.; Gómez, B.F.; Yi, L.; Pathmakumar, T.; Rayguru, M.M.; Subramanian, S. Deep Learning Based Pavement Inspection Using Self-Reconfigurable Robot. *Sensors* **2021**, *21*, 2595. [CrossRef]
25. Alotaibi, A.; Alatawi, H.; Binnouh, A.; Duwayriat, L.; Alhmiedat, T.; Alia, O.M. Deep Learning-Based Vision Systems for Robot Semantic Navigation: An Experimental Study. *Technologies* **2024**, *12*, 157. [CrossRef]
26. Rane, L.N.; Choudhary, P.S.; Rane, J. YOLO and Faster R-CNN Object Detection in Architecture, Engineering and Construction (AEC): Applications, Challenges, and Future Prospects. *SSRN Electron. J.* **2023**. [CrossRef]
27. Zi, X.; Chaturvedi, K.; Braytee, A.; Li, J.; Prasad, M. Detecting Human Falls in Poor Lighting: Object Detection and Tracking Approach for Indoor Safety. *Electronics* **2023**, *12*, 1259. [CrossRef]
28. Wang, K.; Zhou, H.; Wu, H.; Yuan, G. RN-YOLO: A Small Target Detection Model for Aerial Remote-Sensing Images. *Electronics* **2024**, *13*, 2383. [CrossRef]
29. What is YOLO? The Ultimate Guide. 2025. Available online: https://blog.roboflow.com/guide-to-yolo-models (accessed on 17 January 2025).
30. Zhu, P.; Chen, B.; Liu, B.; Qi, Z.; Wang, S.; Wang, L. Object Detection for Hazardous Material Vehicles Based on Improved YOLOv5 Algorithm. *Electronics* **2023**, *12*, 1257. [CrossRef]
31. Terven, J.; Córdova-Esparza, D.-M.; Romero-González, J.-A. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Mach. Learn. Knowl. Extr.* **2023**, *5*, 1680–1716. [CrossRef]
32. Yunusov, N.; Bappy, S.; Abdusalomov, A.; Kim, W. Robust Forest Fire Detection Method for Surveillance Systems Based on You Only Look Once Version 8 and Transfer Learning Approaches. *Processes* **2024**, *12*, 1039. [CrossRef]
33. Raspberry Pi AI Kit: Custom Object Detection with Hailo8L. Available online: https://www.cytron.io/tutorial/raspberry-pi-ai-kit-custom-object-detection-with-h (accessed on 11 October 2024).
34. Model Training with Ultralytics YOLO. Available online: https://docs.ultralytics.com/modes/train/ (accessed on 17 January 2025).
35. Model Prediction with Ultralytics YOLO. Available online: https://docs.ultralytics.com/modes/predict/ (accessed on 17 January 2025).
36. Model Export with Ultralytics YOLO. Available online: https://docs.ultralytics.com/modes/export/ (accessed on 17 January 2025).
37. Raspberry Pi AI Kit: ONNX to HEF Conversion. Available online: https://www.cytron.io/tutorial/raspberry-pi-ai-kit-onnx-to-hef-conversion (accessed on 11 October 2024).
38. Deepview-Validator 3.3.1. Available online: https://pypi.org/project/deepview-validator/ (accessed on 4 February 2025).
39. Hailort. Available online: https://github.com/hailo-ai/hailort (accessed on 4 February 2025).
40. Hailo Application Code Examples. Available online: https://github.com/hailo-ai/Hailo-Application-Code-Examples (accessed on 4 February 2025).

41. Aktouf, L.; Shivanna, Y.; Dhimish, M. High-Precision Defect Detection in Solar Cells Using YOLOv10 Deep Learning Model. *Solar* **2024**, *4*, 639–659. [CrossRef]
42. Computer Vision Model Leaderboard. Available online: https://leaderboard.roboflow.com/ (accessed on 4 February 2025).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.