Scientific Journal of Engineering Research
PTTI PENELITI TEKNOLOGI TEKNIK INDONESIA
Date of publication xxx xx, 2024, date of current version xxx xx, 2024.
Digital Object Identifier ***10.59247/sjer.xxxx***

Article

A Low-Cost Vision-based Fruit Sorting System for Robotic Applications

Muhammad Afaq[1], Emanuele Lindo Secco [1,*]

[1] Robotics Lab, School of Computer Science and the Environment, Liverpool Hope University, Liverpool, L16 9JD, UK; 25005258@hope.ac.uk; seccoe@hope.ac.uk
* Correspondence – seccoe@hope.ac.uk

This paragraph of the first footnote will contain support information, including sponsor and financial support acknowledgment. For example, “This work was supported by the Universitas Islam Riau under Grant GR090XXXX.”

Abstract: Modern robotic systems address complex engineering challenges using artificial intelligence and machine learning techniques. In agricultural robotics, fruit identification and sorting remain challenging due to variations in size, shape, color, orientation, and lighting conditions. This study presents the design and implementation of a vision-based fruit sorting robotic system integrating YOLOv8-based object detection with robotic manipulation. A custom dataset consisting of images of 2 different fruits (namely banana and strawberry images), including single-fruit and multi-fruit scenarios, was used and manually annotated using bounding boxes in CVAT. The dataset was divided into training, validation, and test subsets to enable robust model development under realistic operational conditions. A lightweight YOLOv8 model was trained using CUDA acceleration and optimized for edge deployment by selecting YOLOv8n to balance inference speed and detection accuracy. The trained model was converted to ONNX format and deployed on a Raspberry Pi 5 for real-time inference using live camera input. Evaluation on an independent test dataset achieved a precision of 0.999, recall of 1.000, mAP@0.5 of 0.995, and mAP@0.5:0.95 of 0.963 under controlled experimental conditions with limited object classes. The modular architecture enables low-cost and scalable deployment and provides a foundation for future enhancements, including closed-loop robotic control, additional object categories, and operation in more dynamic environments.

Keywords: Agricultural robotics; Vision-based sorting; YouOnlyLookOnce YOLOv8; Object detection; Robotic manipulation; Edge deployment

Abbreviations

The following abbreviations are used in this manuscript:

Abbreviation	Full Term
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
CVAT	Computer Vision Annotation Tool
DH	Denavit–Hartenberg
DOF	Degree Of Freedom
FDM	Fused Deposition Modeling
GPU	Graphics Processing Unit
IoU	Intersection over Union
mAP	Mean Average Precision
NMS	Non-Maximum Suppression
ONNX	Open Neural Network Exchange
PAN	Path Aggregation Network
PLA	Polylactic Acid
PWM	Pulse-Width Modulation
RGB	Red, Green, Blue
YOLO	You Only Look Once

SJER Scientific Journal of Engineering Research
PTTI PENELITI TEKNOLOGI TEKNIK INDONESIA
Date of publication xxx xx, 2024, date of current version xxx xx, 2024.
Digital Object Identifier **10.59247/sjer.xxxx**

Copyright: © 2025 by the authors. This is an open-access article under the CC-BY-SA license.

1. Introduction

The demand for robotics and automation has increased significantly across industries due to high labor costs, workforce shortages, and the need for consistent quality and efficiency. Industries such as manufacturing, logistics, food processing, recycling, warehousing, and agriculture are gradually replacing traditional manual sorting practices with robotic manipulators integrated with artificial intelligence techniques. In agricultural automation, vision-based robotic systems have gained attention for fruit detection, localization, and harvesting applications [1], [2], [3]. Although such systems perform effectively in controlled environments, they often face challenges when environmental conditions vary, including changes in lighting, object appearance, size, and object density [4].

Recent advances in machine learning and deep learning have significantly improved object recognition and localization in real-world scenarios [5]. Object detection approaches can broadly be categorized into two-stage detectors, such as Faster R-CNN [6], which generate region proposals before classifying each candidate region, and single-stage detectors, which predict bounding boxes and class labels directly from the image in a single pass. Single-stage approaches are generally preferred in embedded and real-time robotic applications due to their significantly lower computational cost and faster inference speed. Vision-guided robotic manipulators are increasingly adopted in industrial and agricultural applications due to their adaptability and automation capabilities [7], [8]. Several machine vision-based fruit sorting and grading systems have been reported in the literature [9], [10], [11]. Furthermore, *YOLO-based* (YouOnlyLookOnce) object detection approaches have demonstrated effectiveness for real-time agricultural robotics applications [12], [13]. However, despite the availability of advanced manipulators and detection algorithms, many systems struggle to handle multiple objects within a single frame in real time while maintaining low computational cost and deployment feasibility on embedded platforms. Additionally, highly accurate industrial solutions are often application-specific and expensive, limiting scalability and accessibility.

Recent studies have explored vision-based robotic systems for fruit harvesting, detection, and grading. Montoya-Cavero et al. [1] and Yang et al. [3] investigated produce detection and localization strategies for harvesting robots; however, their focus was primarily on perception systems rather than complete low-cost embedded deployment. Tan et al. [2] provided a comprehensive review of deep learning-based picking robots, highlighting the importance of multi-sensor integration, yet without presenting a fully integrated real-time sorting framework on resource-constrained hardware. YOLO-based detection approaches have demonstrated high classification accuracy in agricultural applications [12], [13], but several implementations were either evaluated in simulated environments or did not present full robotic sorting integration with edge execution. Similarly, robotic sorting systems based on image processing and grading mechanisms have been reported [8], [9], [10], though many relied on conventional segmentation methods or exhibited limitations in processing speed and adaptability. CNN-based classification approaches have also been applied to related sorting and recognition tasks in different domains [14]. These observations indicate the need for an integrated, low-cost vision-based robotic sorting system that combines real-time object detection, embedded deployment, and autonomous manipulation within a unified framework.

This research focuses on the design and development of a vision-based fruit sorting system using machine learning and a *4-Degree-Of-Freedom* (4-DOF) Robotic Arm. The robotic manipulator design is supported by established robotic modeling principles [15]. A custom dataset consisting of banana and strawberry images was created and annotated using bounding boxes to enable robust detection under realistic conditions. A lightweight YOLOv8n model was trained and optimized for deployment on an embedded platform – namely a Raspberry Pi 5 -, leveraging the suitability of low-cost systems for edge-based computing applications [16]. The system integrates camera sensing, object detection, and robotic manipulation to perform autonomous pick-and-place operations in real time.

The contributions of this research are as follows:

1. Development of a custom annotated dataset containing single and multiple fruit instances for real-time sorting applications;
2. Training and optimization of a lightweight YOLOv8n detection model for accurate classification and localization;
3. Deployment of the trained deep learning model on an embedded device (Raspberry Pi 5) for edge inference;
4. Design and implementation of a modular vision-based 4-DOF robotic arm framework for automated pick-and-place operations;
5. Experimental validation of the integrated system under controlled laboratory conditions.

The scope of the study extends beyond fruit sorting, as similar vision-based robotic approaches are applicable to grading and handling tasks in agriculture and related domains [2], [8]. However, certain limitations were identified. The system was trained on only two object classes, and expansion to additional categories requires further data collection and retraining. The model was developed under specific laboratory conditions, and performance may vary under significantly different environmental constraints. Additionally, the robotic gripper design and the use of bounding box centres for grasp estimation may limit grasping precision in complex scenarios.

The review of existing literature reveals several key gaps that the present work aims to address. First, many YOLO-based detection systems reported in the literature were validated in simulated environments or without integration into a complete robotic sorting pipeline [11], [12]. Second, systems that demonstrate full robotic integration typically rely on high-cost industrial hardware, limiting reproducibility and accessibility in resource-constrained settings [8], [9]. Third, embedded deployment of deep learning models on low-cost single-board computers with real-time performance has not been widely demonstrated alongside autonomous manipulation within a unified framework [16]. This study addresses these gaps by presenting an end-to-end integrated system combining a custom-trained YOLOv8n detection model, embedded inference on a Raspberry Pi 5, and a 3D-printed 4-DOF robotic arm within a single low-cost platform.

The remainder of this paper is organized as follows. Section 2 describes the methodology, including the system architecture, the mechanical design, the kinematic modeling, dataset preparation, model training, and embedded deployment. Section 3 presents the experimental results and discussion, covering the mechanical validation, training convergence, detection performance, and embedded deployment outcomes. Finally, Section 4 concludes the paper and outlines directions for future work.

2. Methodology

The adopted methodology follows an experimental and implementation-driven approach to design, validate, and deploy a vision-based robotic fruit sorting system. The workflow includes mechanical fabrication, dataset development, model training, embedded deployment, and real-time validation. The modular separation of mechanical design, perception system, and embedded implementation allows independent evaluation and improvement of each subsystem.

2.1. System Architecture

The proposed system follows a modular architecture integrating computer vision, machine learning, embedded computation, and robotic manipulation. The objective of the architecture is to translate object detection and localization results into autonomous pick-and-place sorting actions using a 4-DOF Robotic Arm.

As illustrated in Fig. 1, the system is divided into three principal layers: sensor architecture, hardware architecture, and software architecture. The complete processing pipeline consists of image acquisition, intelligent inference, decision-making, and robotic actuation. Each stage operates as an independent module, enabling flexibility in design modification and scalability.

In the sensor layer, the Raspberry Pi Camera Module 3 continuously captures live image frames from the workspace. These frames are passed to the hardware layer, where the Raspberry Pi 5 serves as the central computing unit, executing the ONNX-format YOLOv8n model via ONNX Runtime. The software layer encompasses the complete processing chain: frame preprocessing (resizing and color space conversion from RGB to BGR), object detection inference, confidence thresholding, *Non-Maximum Suppression* (NMS), and decision-making logic that maps detected class labels to predefined sorting bin targets.

Upon detection of a fruit object, the system extracts the bounding box center coordinates and translates these pixel-space values into joint-space commands for the 4-DOF robotic arm. The arm then executes a pick-and-place motion sequence to deposit the identified fruit into the corresponding sorting bin. This modular separation of sensing, inference, and actuation allows each subsystem to be independently modified or upgraded without redesigning the entire pipeline and provides a scalable foundation for future integration of additional object classes or closed-loop visual feedback.

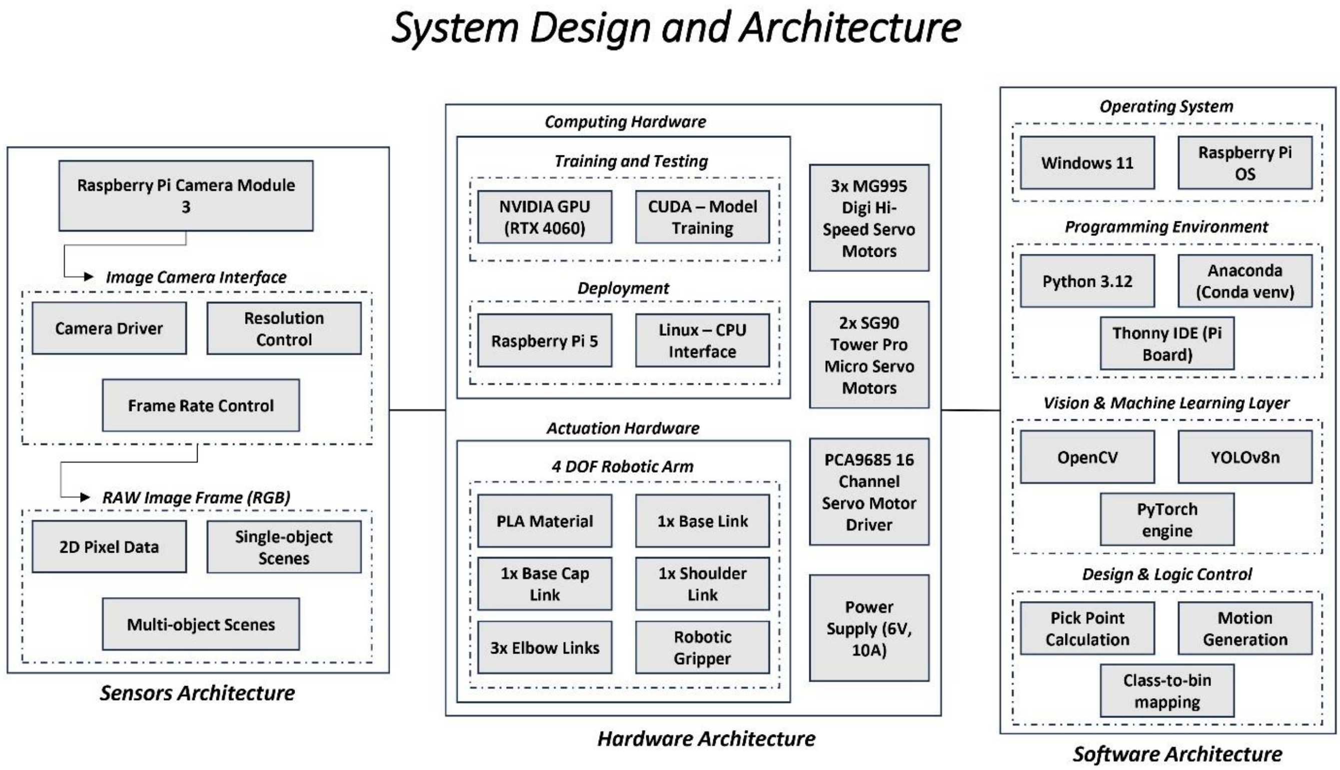

Figure 1: Overall system architecture of the vision-based fruit sorting robotic system, including sensing, computing, and actuation modules.

In the image acquisition stage, frames are captured using a Raspberry Pi Camera Module 3. The captured data are used both for dataset preparation and real-time deployment to maintain environmental consistency. The intelligent inference stage employs a deep learning-based object detection model (YOLOv8) to generate bounding boxes, class labels, and confidence scores. The decision-making module interprets the detection outputs and save the data on the board for computation of pick points, and mapping of object classes to predefined sorting bins.

Figure 2: 3D design of the Robotic Arm [17]

2.2. Mechanical Configuration

A 4-DOF revolute manipulator was selected to balance mechanical simplicity and functional capability. The robotic arm was not designed from scratch but was constructed from an open-source design available on Printables.com [17], which provides all the necessary STL files for the 3D printing manufacturing. The structure was fabricated by means of a Prusa 3D Printer with PLA material through *Fused Deposition Modeling* (FDM). Minimal modifications were made to the original design to accommodate the servo motors and wiring configuration used in this study. The assembly, electronics integration, servo calibration, and software control were carried out by the authors. A similar approach of designing and manufacturing low-cost manipulators using 3D printing technology has been demonstrated for robotic applications [18]. Fig. 2 displays the 3D overall design of the robot [17]. Fig. 3 shows the manufactured and assembled robotic arm.

The configuration consists of a base rotation joint (θ_1) about the vertical axis, followed by shoulder and elbow pitch joints (θ_2 and θ_3) that define planar motion in a vertical workspace. The fourth joint provides wrist roll motion (θ_4) to adjust end-effector orientation during grasping.

The base-to-shoulder vertical offset is defined as $L_1 = 45$ mm. The upper-arm and forearm link lengths are approximately $L_2 = 245$ mm and $L_3 = 245$ mm, respectively. Servo motors (MG995 and SG90) were mounted at designated joints to enable position control. The structure was fabricated using PLA material through *Fused Deposition Modeling* (FDM) for lightweight prototyping and experimental validation.

The 4-DOF structure provides sufficient mobility for planar positioning, vertical reach, and orientation adjustment while avoiding the complexity of higher-degree-of-freedom industrial manipulators. This configuration is suitable for sorting tasks within a constrained workspace where precise positioning is required rather than complex spatial manipulation.

The electronics platform is built around the Raspberry Pi 5 as the central controller, which handles both inference and servo control within a single board. Power is supplied via a regulated 5V DC source shared between the Raspberry Pi and the servo bus [19]. Servo motors are connected to the Raspberry Pi GPIO pins and driven using standard Pulse-Width Modulation (PWM) protocol. Two MG995 servos are positioned at the shoulder and elbow joints due to their higher torque rating, making them capable of driving the heavier upper-arm links. The lighter SG90 servos are used at the wrist and gripper joints where lower torque is sufficient. The camera module connects via the dedicated *Camera Serial Interface* (CSI) port on the Raspberry Pi 5 and is interfaced through the Picamera2 API. No additional microcontroller or motor driver board was used; all control logic runs directly on the Raspberry Pi OS. Table 1 presents the key hardware components of the system along with their technical specifications.

Table 1: Hardware Components and Technical Specifications of the 4-DOF Robotic Arm System.

Component	Model / Type	Key Specifications	Qty
Single-board Computer	Raspberry Pi 5	Quad-core Cortex-A76, 8 GB RAM, 64-bit OS	1
Camera Module	RPi Camera Module 3	12 MP Sony IMX708, autofocus, CSI interface, up to 1080p	1
Shoulder/Elbow Servo	MG995	Torque: 9.4 kg·cm (4.8V); Speed: 0.2 s/60°; Metal gear; PWM control	2
Wrist/Gripper Servo	SG90	Torque: 1.8 kg·cm (4.8V); Speed: 0.1 s/60°; Plastic gear; PWM control	2
Frame Material	PLA Filament	Density: 1.24 g/cm³; Print temp: 190–220°C; FDM process	—
3D Printer	Prusa MK Series	FDM; layer resolution 0.05–0.35 mm; open-source STL files	1
Inference Engine	ONNX Runtime (CPU)	CPU-based; compatible with RPi OS 64-bit; ONNX model format	—

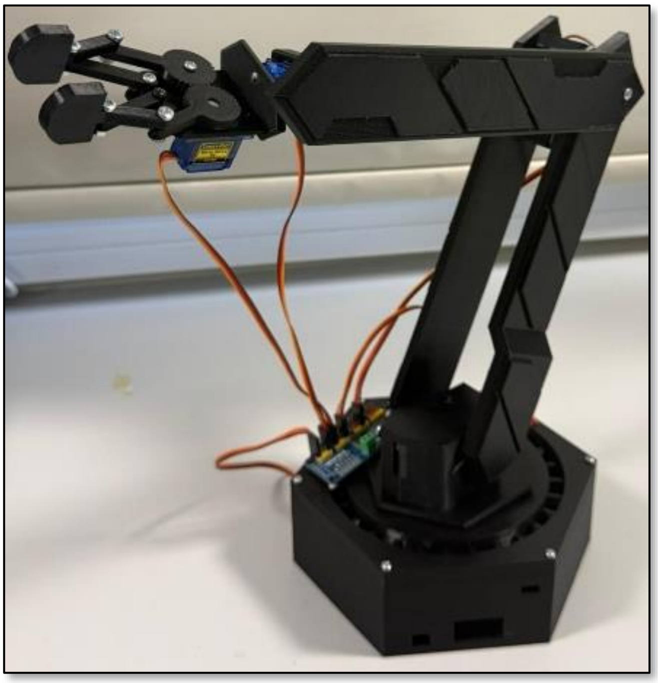

Figure 3: Fabricated 4-DOF robotic arm for fruit sorting experiments.

2.3. Denavit–Hartenberg Representation

The manipulator kinematics can be expressed using standard *Denavit–Hartenberg* (DH) parameters as summarized in Table 2. The DH parameters were assigned following the standard convention [15], in which a coordinate frame is attached to each link of the manipulator according to four rules: (i) the Z-axis of frame i is aligned with the axis of joint i+1; (ii) the X-axis of frame i points along the common normal between Z-axes of consecutive joints; (iii) the link length a_i is the distance between Z-axes measured along X_i; and (iv) the link twist α_i is the angle between consecutive Z-axes measured about X_i. For joint 1, the base rotation is about the vertical Z-axis with a vertical offset d_1 = 45 mm and no link length ($a_1 = 0$), while a 90° twist (α_1 = 90°) accounts for the transition to the horizontal plane of the upper arm. Joints 2 and 3 are parallel revolute joints operating in the same plane, with link lengths $a_2 = a_3$ = 245 mm, zero twist, and zero offset. Joint 4 provides wrist roll with all geometric parameters set to zero. The assigned coordinate frames are illustrated in Fig. 4, where the Z-axes (blue) represent the axes of rotation for each joint and the X-axes (red) represent the common normals between consecutive joint axes following the DH convention.

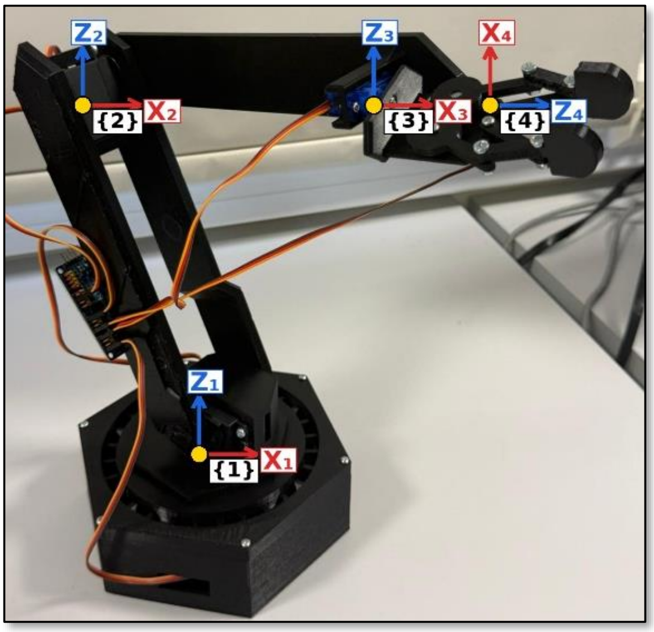

Figure 4: Coordinate frames assigned to each joint of the 4-DOF robotic manipulator following the Denavit–Hartenberg convention.

Table 2: DH parameters of the 4-DOF manipulator.

Link (i)	a_i (mm)	α_i	d_i (mm)	θ_i
1	0	90°	45	θ_1
2	245	0°	0	θ_2
3	245	0°	0	θ_3
4	0	0°	0	θ_4

Since the wrist joint primarily influences orientation, the end-effector position depends mainly on the first three joints.

2.4. Forward Kinematics

The Cartesian position of the end-effector (x, y, z) is derived by composing the homogeneous transformation matrices between consecutive frames using the DH convention. The general transformation matrix from frame i−1 to frame i is given by: ${}^{i-1}T_i$ = $Rot(Z,\theta_i)\cdot Trans(Z,d_i)\cdot Trans(X,a_i)\cdot Rot(X,\alpha_i)$. Applying the DH parameters from Table 2 and composing transformations ${}^0T_1\cdot{}^1T_2\cdot{}^2T_3\cdot{}^3T_4$, and noting that joint 4 (wrist roll) does not contribute to end-effector position, the Cartesian coordinates depend only on the first three joints and are expressed as (Eq. 1–3):

$$x = cos\theta_1(L_2cos\theta_2 + L_3cos(\theta_2 + \theta_3)) \quad (1)$$
$$y = sin\theta_1(L_2cos\theta_2 + L_3\cos(\theta_2 + \theta_3)) \quad (2)$$
$$z = L_1 + L_2sin\theta_2 + L_3sin(\theta_2 + \theta_3) \quad (3)$$

where $L_1 = 45$mm, $L_2 = 245$mm, and $L_3 = 245$mm.

Table 3 summarizes all variables and parameters used in Equations (1) – (6).

Table 3: Definition of Variables and Parameters Used in the Kinematic Equations.

Symbol	Description	Value / Unit
x, y, z	Cartesian coordinates of the end-effector in the base frame	mm
θ_1, θ_2, θ_3, θ_4	Joint angles at base, shoulder, elbow, and wrist, respectively	degrees (°)
L_1	Vertical offset from base to shoulder joint	45 mm
L_2	Upper-arm link length (shoulder to elbow)	245 mm
L_3	Forearm link length (elbow to wrist)	245 mm
r	Projected horizontal reach: $r = \sqrt{(x^2 + y^2)}$	mm
z′	Adjusted vertical height: $z' = z - L_1$	mm
D	Cosine of elbow angle: $D = (r^2 + z'^2 - L_2^2 - L_3^2) / (2L_2L_3)$	dimensionless
atan2	Two-argument arctangent function, returning angle in correct quadrant	radians

2.5. Inverse Kinematics

For a target position (x, y, z), the inverse kinematics derivation proceeds in four steps. Step 1 – Base rotation (θ_1): The base joint rotates about the vertical axis. From Eq. 1 and Eq. 2, the ratio y/x eliminates the planar reach term, giving the base rotation angle (Eq. 4). Step 2 – Planar reach and height: The projected horizontal distance $r = \sqrt{(x^2+y^2)}$ and adjusted vertical height $z' = z - L_1$ decouple the 2-link planar problem. Step 3 – Elbow angle (θ_3): The cosine rule applied to the triangle formed by links L_2 and L_3 and the straight-line distance to the target gives $D = (r^2+z'^2-L_2^2-L_3^2)/(2L_2L_3)$, and the elbow angle as Equation (5), selecting the elbow-up configuration Step 4 – Shoulder angle (θ_2): Derived from the geometry of the planar two-link chain as Equation (6). The wrist roll angle θ_4 is selected based on the desired end-effector orientation. The base rotation angle is determined by:

$$\theta_1 = atan2(y, x) \quad (4)$$
$$\theta_3 = atan2(\sqrt{1 - D^2}, D) \quad (5)$$
$$\theta_2 = atan2(z', r) - atan2(L_3sin\theta_3, L_2 + L_3cos\theta_3) \quad (6)$$

The wrist roll angle θ_4 is selected based on the desired orientation during grasping. In this implementation, joint angles were commanded directly through servo position control, while the kinematic formulation provides geometric interpretation and supports future closed-loop extensions.

2.6. Dataset Preparation and Annotation

A custom dataset consisting of 914 images was set up to reflect realistic operating conditions for fruit sorting.

Two object classes were considered: banana and strawberry. Images were captured under controlled indoor lighting using a Raspberry Pi Camera Module 3.

The dataset includes single-object scenes as well as multi-object configurations in which bananas and strawberries appear simultaneously. Variations in object orientation, scale, and relative position were introduced to reduce overfitting to a fixed scene layout. Extreme background clutter and lighting variations were intentionally avoided to maintain a controlled experimental environment.

The dataset was divided into training (70%), validation (20%), and testing (10%) subsets, corresponding to 642, 181, and 91 images, respectively. Annotation was performed using bounding box labeling in CVAT. Each visible fruit instance was labeled independently, and occluded objects were annotated only when visually identifiable. The dataset was exported in YOLO-compatible format with normalized bounding box coordinates.

Representative samples from the dataset are illustrated in Fig. 5, showing single-fruit scenes for each class as well as multi-fruit configurations with annotated bounding boxes as generated in CVAT. The diversity in object orientation, scale, and relative positioning across samples reflects the controlled variation introduced during data collection to reduce overfitting.

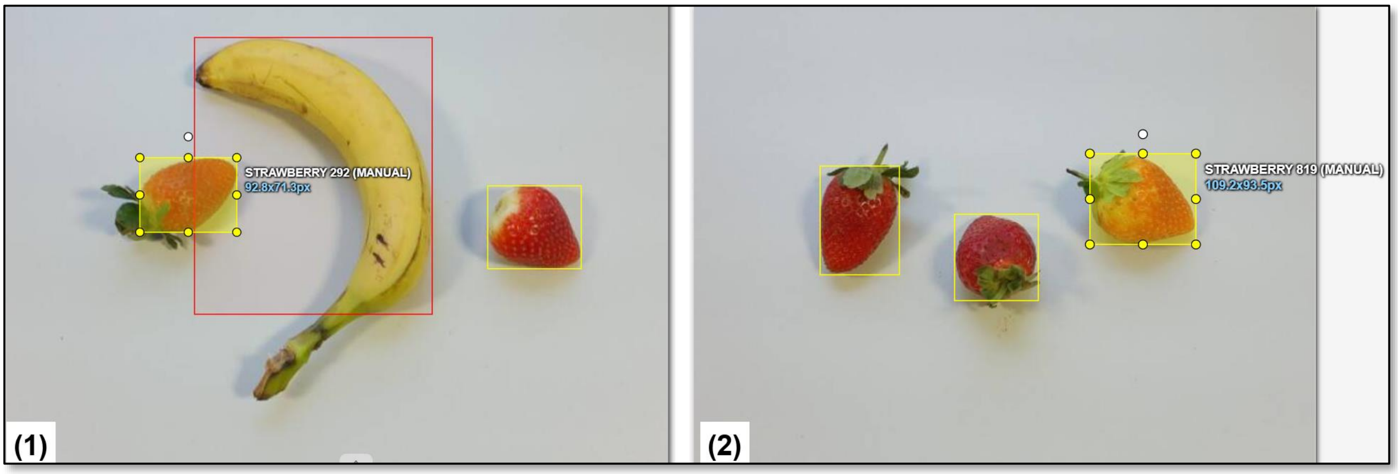

Figure 5: Manual dataset annotation using bounding boxes in CVAT for training the model.

2.7. Model Training and Learning Strategy

YOLOv8n was selected as the object detection architecture due to its lightweight design and suitability for embedded deployment. The *You Only Look Once* (YOLO) family of detectors was originally introduced by Redmon et al. [20] as a unified real-time object detection framework that processes the entire image in a single forward pass, eliminating the need for region proposal stages such as those used in Faster R-CNN [6]. Since its introduction, the YOLO architecture has undergone significant development across successive versions, culminating in YOLOv8 as developed by Ultralytics [21]. The YOLOv8 architecture consists of three principal components: a backbone, a neck, and a detection head [22]. The backbone, based on a modified *CSPDarknet* structure, performs hierarchical feature extraction from the input image tensor of dimensions 1×3×640×640 (batch × channels × height × width), producing multi-scale feature maps at different spatial resolutions. The neck employs a *Path Aggregation Network* (PAN) with *Feature Pyramid Network* (FPN) [23] connections to fuse features across scales, enabling detection of objects at varying sizes. The detection head applies decoupled convolution branches for simultaneous classification and bounding box regression, outputting class probabilities, bounding box coordinates, and confidence scores. Post-processing steps including confidence thresholding and *Non-Maximum Suppression* (NMS) are applied to produce the final detections. The YOLOv8n variant used is the smallest configuration in the family, with approximately 3.2 million parameters, making it suitable for CPU-based embedded deployment, unlike heavier architectures such as MobileNetV2 [24].

The model was trained for 100 epochs with an input resolution of 640 × 640 pixels and a batch size of 16. Pre-trained YOLOv8 weights were used for transfer learning to improve convergence and generalization. Mixed precision training enabled to optimize computational efficiency. The trained YOLOv8 model was exported from *PyTorch* format to ONNX to facilitate embedded deployment. The input tensor was fixed to 1 × 3 × 640 × 640. Post-processing included bounding box decoding, confidence thresholding, and non-maximum suppression to ensure accurate detections during inference.

2.8. Embedded Deployment on Raspberry Pi 5

Deployment was performed on a Raspberry Pi 5 running *Raspberry Pi OS* (64-bit). ONNX Runtime was selected as the inference engine for CPU-based execution [25]. Real-time image acquisition was achieved using the Raspberry Pi Camera Module 3 interfaced through the Picamera2 API.

Captured frames at 1280 × 720 resolution were resized to 640 × 640 for inference. RGB-to-BGR conversion was applied to ensure compatibility with the ONNX model. Detection outputs were visualized with bounding boxes and confidence scores, and results were logged to CSV files for performance analysis. The deployed system maintained stable real-time performance under controlled laboratory conditions.

2.9. Integration of Detection and Robotic Control

The integration between the YOLOv8 detection pipeline and the robotic arm control operates as a sequential sensing–actuation loop. During operation, the *Raspberry Pi Camera Module 3* continuously captures frames at 1280×720 resolution. Each frame is preprocessed – resized to 640×640 pixels and converted from RGB to BGR – before being passed to the ONNX Runtime inference engine executing the YOLOv8n model. The model outputs bounding box coordinates (x_{min}, y_{min}, x_{max}, y_{max}), class labels, and confidence scores for each detected object. Detections below a confidence threshold of 0.5 are discarded, and *Non-Maximum Suppression* (NMS) is applied to eliminate overlapping detections of the same object.

For each valid detection, the pixel-space bounding box center (c_x, c_y) is computed and used as the estimated grasp point. This 2D coordinate is mapped to the physical workspace using a fixed camera-to-workspace homography derived from a prior calibration procedure. The resulting 3D target position (x, y, z) is passed to the inverse kinematics solver described in Section 2.5, which computes the required joint angles θ_1, θ_2, θ_3, and θ_4. These angles are converted to servo PWM commands and transmitted to the servo motors via the Raspberry Pi GPIO interface. The arm then executes a pick-and-place sequence: it moves to the grasp position, closes the gripper, lifts the object, rotates to the target bin location based on the class label, and releases the gripper. Detection results and confidence scores are logged simultaneously to a CSV file for offline analysis.

The overall operational sequence of the system is summarised in the following pseudocode (Algorithm 1):

```
Algorithm 1: Vision-based Fruit Sorting Pipeline
BEGIN
     Load YOLOv8n ONNX model
     Initialise camera (1280x720) and servo controller
     WHILE system is running DO
          frame ← capture_frame()
          input ← preprocess(frame)    // resize 640x640, RGB→BGR
          detections ← model.infer(input)
          detections ← NMS(detections, threshold=0.5)
          FOR each detection IN detections DO
               class_label ← detection.class
               (cx, cy) ← bounding_box_centre(detection)
               (x, y, z) ← map_to_workspace(cx, cy)
               (θ1,θ2,θ3,θ4) ← inverse_kinematics(x, y, z)
               move_arm_to(θ1,θ2,θ3,θ4); close_gripper()
               rotate_to_bin(class_label); open_gripper()
               log_result(class_label, confidence, cx, cy)
          END FOR
     END WHILE
END
```

3. Results and Discussion

This section presents the experimental evaluation of the proposed vision-based fruit detection system and the fabricated 4-DOF robotic platform. The results include mechanical validation of the manipulator, offline detection performance, embedded deployment outcomes, and a critical discussion of system capabilities and limitations.

3.1. Mechanical Validation of the 4-DOF Robotic Arm

The fabricated robotic arm components were successfully printed using fused deposition modeling and assembled according to the CAD design. Visual inspection and dimensional verification confirmed acceptable fabrication accuracy, with no observable deformation or structural instability.

Each joint was tested individually to evaluate its practical range of motion. Repeated actuation did not introduce noticeable misalignment or mechanical drift. The servo-driven joints achieved stable and repeatable angular motion within the mechanical constraints of the design. The practical operating ranges of each joint are summarized in Table 4.

Table 4: Range of Motion and Workspace Configuration of 4-DOF Robotic Arm.

Joint	Function	Practical Operating Range	Role in Sorting Task
Base (J1)	Horizontal rotation	~30°–150°	Positions arm across workspace
Shoulder (J2)	Vertical lift	~20°–140°	Controls height positioning
Elbow (J3)	Reach extension	~30°–150°	Adjusts forward reach
Wrist (J4)	End-effector orientation	~45°–135°	Adjusts grasp orientation

The mechanical platform demonstrated sufficient workspace coverage for small-scale sorting tasks. However, the manipulator was not operated under closed-loop visual feedback during this study.

3.2. Training Convergence

The training curves demonstrate stable convergence across bounding box regression loss, classification loss, and distribution focal loss. Both training and validation losses decreased progressively over successive epochs, indicating effective learning of localization and class discrimination features. Minor fluctuations were observed during early epochs as the model adapted to the dataset; however, performance stabilized during later training stages without divergence or instability. The convergence behavior suggests that the selected hyperparameters and training duration were appropriate for the dataset size and task complexity. The training and validation loss curves are shown in Figure 7.

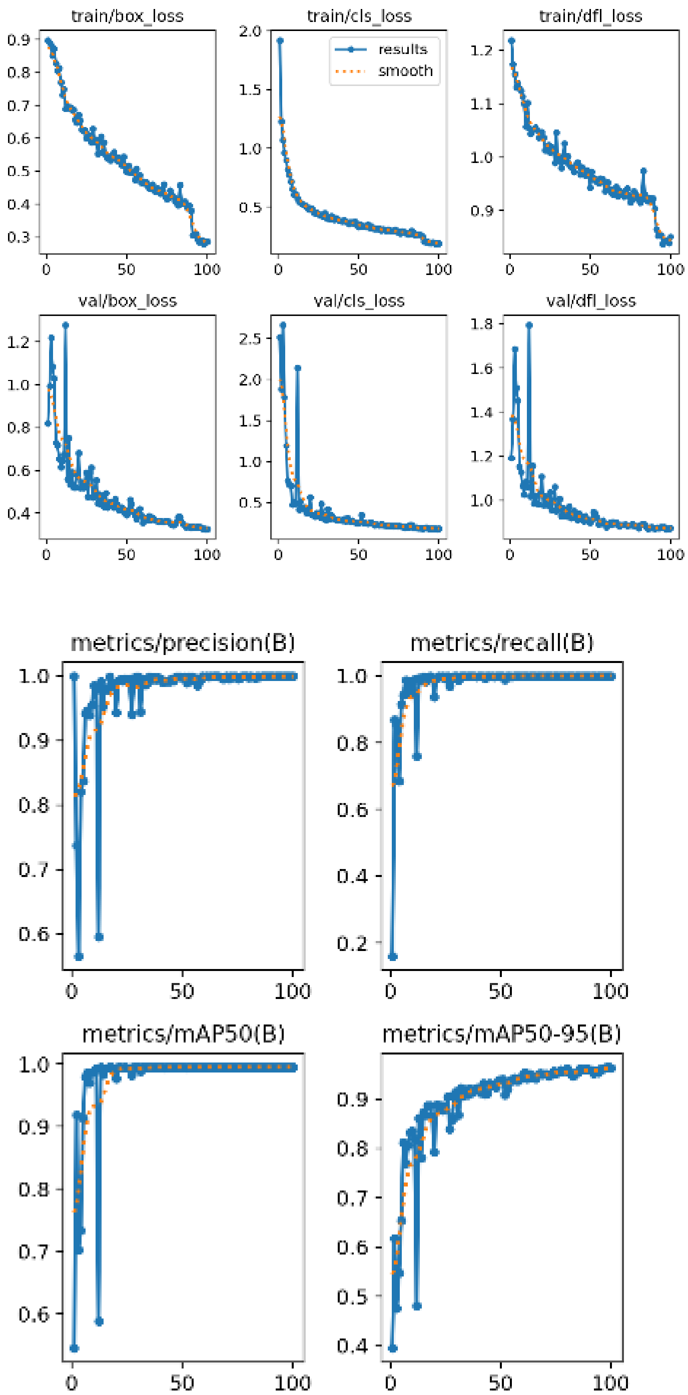

Figure 6: Training and validation loss curves for bounding box regression, classification loss, distribution focal loss, and evaluation metrics over 100 epochs.

3.3. Validation and Test Metrics

Detection performance was quantified using the following standard metrics. Let TP, FP, and FN denote *True Positives* (TP), *False Positives* (FP), and *False Negatives* (FN), respectively. *Precision* measures the proportion of correct positive detections (Eq. 7), *Recall* measures the proportion of actual positives correctly identified (Eq. 8), and the *F1-score* is their harmonic mean (Eq. 9):

$$Precision = TP / (TP + FP) \quad (7)$$
$$Recall = TP / (TP + FN) \quad (8)$$
$$F1 = 2 \cdot (Precision \cdot Recall) / (Precision + Recall) \quad (9)$$

Intersection over *Union* (IoU) between a predicted bounding box B_{pred} and ground-truth box B_{gt} is defined as Equation 10:

$$IoU = |B\_pred \cap B\_gt| \; / \; |B\_pred \cup B\_gt| \quad (10)$$

Average Precision (AP) is computed as the area under the *Precision–Recall* curve for a single class (Eq. 11), and *Mean Average Precision* (mAP) averages AP across all object classes C (Eq. 12):

$$AP = \int_0^1 P(R)\, dR \quad (11)$$
$$mAP = (1/C) \sum AP_i \quad (12)$$

Two IoU threshold variants are reported. mAP@0.5 evaluates detection at a fixed IoU threshold of 0.50, while mAP@0.5:0.95 averages mAP over IoU thresholds from 0.50 to 0.95 in steps of 0.05, providing a stricter measure of localization accuracy.

Detection performance was evaluated using precision, recall, mean Average Precision at IoU 0.5 (mAP@50), and mean Average Precision across IoU thresholds 0.5–0.95 (mAP@50–95). On the validation dataset, the model achieved an overall precision of 0.99935 and recall of 1.000. The mAP@50 value reached 0.995, while mAP@50–95 was 0.96321. These results indicate highly reliable classification and accurate bounding box localization under controlled conditions.

Evaluation on the unseen test dataset produced similar results, with overall precision and recall values of approximately 1.000 and mAP@50 remaining at 0.995. The mAP@50–95 value was 0.958, demonstrating that localization accuracy was preserved under stricter IoU thresholds. The close alignment between validation and test performance suggests that the model generalized well and did not exhibit significant overfitting.

In Table 5, class-wise analysis revealed consistent performance across bananas and strawberries, indicating balanced representation in the dataset and absence of class-specific bias.

Table 5: Class-wise detection performance on validation and unseen test datasets.

Dataset (Split)	Class	Precision	Recall	mAP@50	mAP@50–95
Validation	Banana	1.000	1.000	0.995	0.954
Validation	Strawberry	0.999	1.000	0.995	0.973
Validation (Overall)	All	0.99935	1.000	0.995	0.96321
Test (Unseen)	Banana	0.999	1.000	0.995	0.941
Test (Unseen)	Strawberry	1.000	1.000	0.995	0.975
Test (Overall)	All	1.000	1.000	0.995	0.958

3.4. Precision–Recall Characteristics

The Precision–Recall curves exhibit a smooth and near-ideal profile, maintaining high precision across increasing recall levels. The reported mAP@50 value of approximately 0.995 reflects strong detection reliability at the standard IoU threshold. Such characteristics are beneficial for robotic systems, where confidence thresholds may need to be adjusted dynamically depending on task sensitivity. The Precision–Recall characteristics are illustrated in Figure 8.

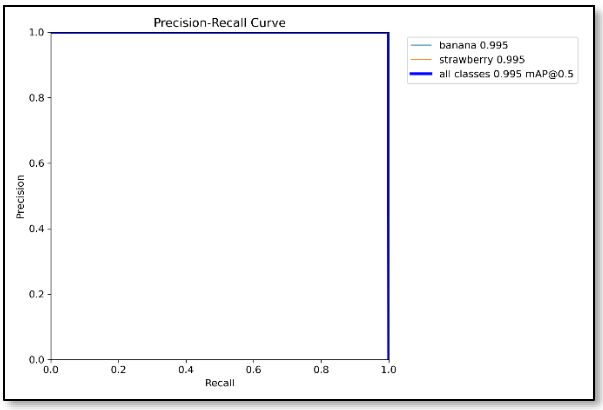

Figure 7: Precision–Recall curves for banana and strawberry detection with overall mAP@50 of 0.995.

3.5. Embedded Deployment Performance

The trained YOLOv8n model was successfully deployed on a Raspberry Pi 5 using ONNX Runtime for CPU-based inference. Live experiments using real-time camera input confirmed stable detection of bananas and strawberries. During testing, the system operated at approximately 3.1 to 3.6 frames per second. Although CPU-based execution limits inference speed compared to GPU acceleration, the achieved frame rate was sufficient for low-speed or prototype-scale sorting applications.

Bounding box placement remained stable across consecutive frames, and confidence scores showed minimal fluctuation when objects were stationary or moving slowly. The correction of color space alignment between the camera input and the trained model significantly improved detection consistency, highlighting the importance of maintaining preprocessing compatibility between training and deployment environments.

Runtime logging of detection confidence values further confirmed stable inference behavior. The average confidence values were approximately 0.897 for bananas and 0.910 for strawberries, with maximum values exceeding 0.97 for both classes. Low-confidence detections were effectively suppressed using thresholding and non-maximum suppression, reducing false positives during live operation. Sample live detection results are presented in Figure 9. Figure 10 illustrates the complete experimental setup, showing the 4-DOF robotic arm, Raspberry Pi 5, camera module, and fruit objects positioned within the sorting workspace during operation.

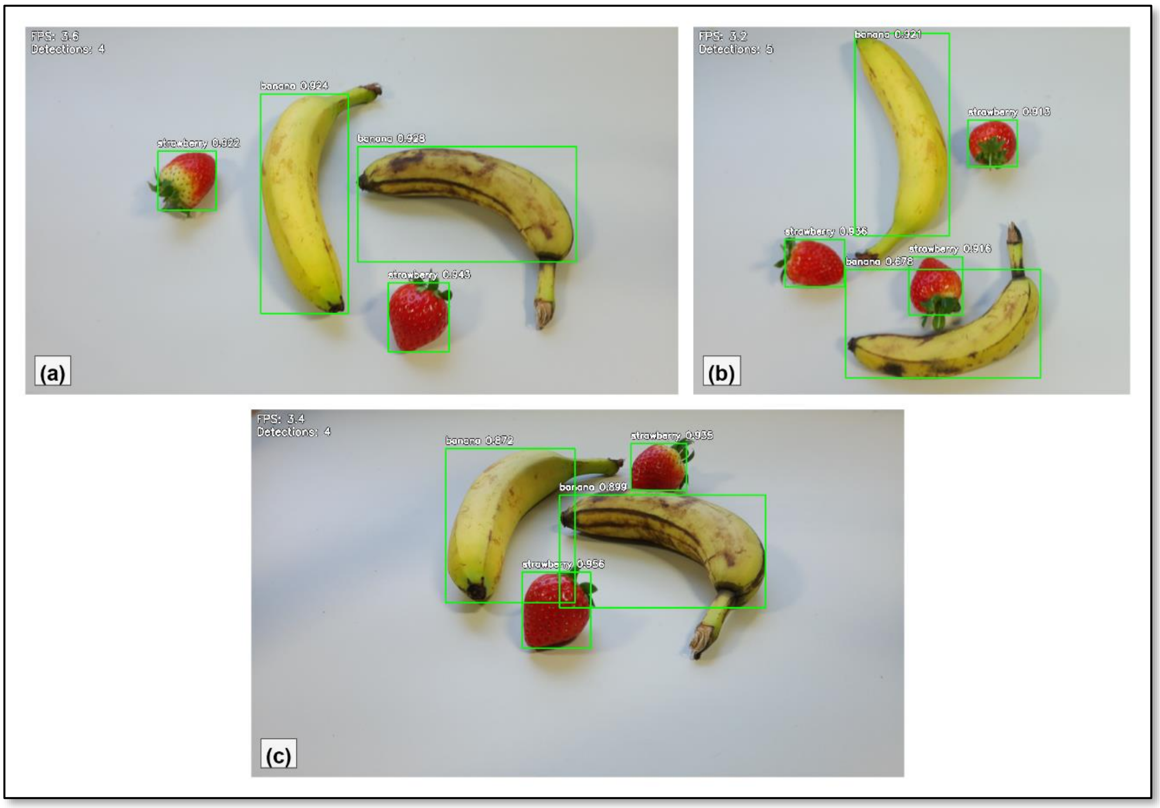

Figure 8: Representative real-time detection results on Raspberry Pi 5: (a) multi-object scene, (b) orientation variation, (c) different object positioning.

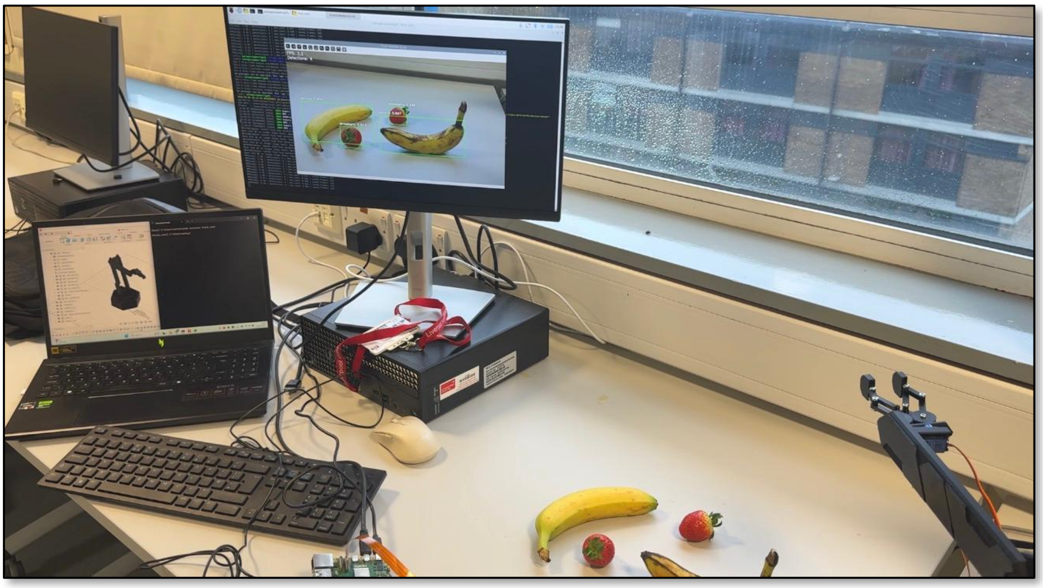

Figure 9: The overall system showing the 4-DOF robotic arm, Raspberry Pi 5, camera module, and fruit objects used during the sorting experiments.

3.6. Discussion

The experimental results demonstrate that a lightweight YOLOv8n model can achieve high detection accuracy on a small, task-specific dataset under controlled laboratory conditions [26] [27] [28]. The strong agreement between validation and test metrics indicates reliable generalization within the defined environment. Furthermore, successful deployment on Raspberry Pi 5 confirms the feasibility of executing deep learning-based object detection models on embedded platforms without GPU acceleration.

The mechanical platform demonstrated sufficient structural integrity and workspace coverage for lightweight fruit handling. Minor torque limitations were observed when handling heavier banana samples, suggesting that further mechanical reinforcement or design and servo upgrades may improve robustness [29].

Overall, the results confirm the feasibility of integrating vision-based detection with embedded deployment for small-scale sorting applications [27] [30], while also identifying areas for improvement in dataset diversity, hardware optimization, and closed-loop manipulation control.

Table 6 provides a comparison of the proposed system against selected related studies. The comparison highlights that the proposed system is one of the few combining real-time YOLO-based detection with full robotic pick-and-place integration on a low-cost CPU-only embedded platform. While the number of object classes is limited compared to larger industrial systems, the framework demonstrates feasibility for resource-constrained settings and provides a foundation for future expansion.

Table 6: Comparison of the Proposed System with Related Studies.

Study	Detection Model	Deployment Platform	Classes	Real-Time	Full Robotic Integration	Low-Cost HW
Montoya-Cavero et al. [1]	Custom CNN	Workstation/ GPU	Multi	Yes	Partial (perception only)	No
Dairath et al. [7]	Image processing	PC	Multi	Limited	Yes	Partial
Nikam et al. [12]	YOLOv8n	Not embedded	1	Yes	No	Partial
Proposed System	**YOLOv8n (ONNX)**	**Raspberry Pi 5 (CPU)**	**2**	**Yes (~3.4 fps)**	**Yes (full pipe line)**	**Yes**

4. Conclusion

This study presented the design and implementation of a vision-based robotic sorting system integrating a 4-DOF manipulator with a YOLOv8-based object detection model. A robotic arm was integrated using 3D printing and demonstrated sufficient structural stability and workspace coverage for lightweight sorting tasks under controlled laboratory conditions. Although occasional grasping instability was observed when handling heavier banana samples due to mechanical and material limitations, the platform validated the feasibility of low-cost prototype development for vision-guided manipulation.

The proposed detection model, trained on a custom dataset of bananas and strawberries, achieved high precision, recall, and mean average precision during validation and testing. The close agreement between validation and unseen test results indicates reliable generalization within the defined experimental environment. Furthermore, the trained model was successfully converted to ONNX format and deployed on a Raspberry Pi 5 for CPU-based inference. Despite hardware constraints, the embedded system demonstrated stable real-time detection using live camera input, with consistent confidence scores and reliable bounding box localization.

Overall, the results confirm the feasibility of integrating deep learning-based object detection with embedded computing for small-scale robotic sorting applications. While the current implementation validates the perception and deployment components, closed-loop visual feedback between detection and robotic actuation was not experimentally evaluated and remains an important extension of this work. Future improvements may include expansion of the dataset to more object categories and environmental variations, mechanical reinforcement of the manipulator, integration of feedback sensors for improved positioning accuracy, and performance optimization through model quantization or hardware acceleration. With such enhancements, the proposed framework can be extended beyond fruit sorting to broader automation and inspection tasks.

5. Conflicts of Interest

The authors declare no conflicts of interest

Acknowledgements

This work was presented in Dissertation form under the supervision of Emanuele Lindo Secco in fulfilment of the requirements for the MSc Robotics Engineering for the student Muhammad Afaq from the School of Computer Science and the Environment, Liverpool Hope University. We would like to express our thanks to all the Team of the School of Computer Science and the Environment for their support.

AI tool was used for coding assistance and structure. All AI-generated content was reviewed and approved by the author. The intellectual content, design decisions, and analysis within this coursework remain solely the author's own work.

6. References

[1] L. E. Montoya-Cavero, R. Díaz de León Torres, A. Gómez-Espinosa, and J. A. Escobedo Cabello, "Vision systems for harvesting robots: Produce detection and localization," *Comput. Electron. Agric.*, vol. 192, p. 106562, Jan. 2022, doi: 10.1016/J.COMPAG.2021.106562.

[2] Y. Tan, X. Liu, J. Zhang, Y. Wang, and Y. Hu, "A Review of Research on Fruit and Vegetable Picking Robots Based on Deep Learning," *Sensors 2025, Vol. 25, Page 3677*, vol. 25, no. 12, p. 3677, Jun. 2025, doi: 10.3390/S25123677.

[3] Y. Yang, Y. Han, S. Li, Y. Yang, M. Zhang, and H. Li, "Vision based fruit recognition and positioning technology for harvesting robots," *Comput. Electron. Agric.*, vol. 213, p. 108258, Oct. 2023, doi: 10.1016/J.COMPAG.2023.108258.

[4] T. Yoshida, T. Kawahara, and T. Fukao, "Fruit recognition method for a harvesting robot with RGB-D cameras," *ROBOMECH Journal 2022 9:1*, vol. 9, no. 1, pp. 15-, May 2022, doi: 10.1186/S40648-022-00230-Y.

[5] G. Ian, C. Aaron, and B. Yoshua, "Deep Learning," *Request exam copy View preview Adaptive Computation and Machine Learning series*, pp. 1–23, 2016, Accessed: Feb. 03, 2026. [Online]. Available: http://www.deeplearningbook.org

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", Accessed: May 10, 2026. [Online]. Available: https://github.com/

[7] M. H. Ali, K. Aizat, K. Yerkhan, T. Zhandos, and O. Anuar, "Vision-based Robot Manipulator for Industrial Applications," *Procedia Comput. Sci.*, vol. 133, pp. 205–212, Jan. 2018, doi: 10.1016/J.PROCS.2018.07.025.

[8] M. H. Dairath *et al.*, "Computer vision-based prototype robotic picking cum grading system for fruits," *Smart Agricultural Technology*, vol. 4, p. 100210, Aug. 2023, doi: 10.1016/J.ATECH.2023.100210.

[9] C. S. Nandi, B. Tudu, and C. Koley, "An automated machine vision based system for fruit sorting and grading," *Proceedings of the International Conference on Sensing Technology, ICST*, pp. 195–200, 2012, doi: 10.1109/ICSENST.2012.6461669.

[10] J. Yu, W. Miao, G. Zhang, K. Li, Y. Shi, and L. Liu, "Target Positioning and Sorting Strategy of Fruit Sorting Robot Based on Image Processing," *International Information and Engineering Technology Association*, vol. 38, no. 3, pp. 797–805, Jun. 2021, doi: 10.18280/ts.380326.

[11] B. Purwantana *et al.*, "Performance of an Automatic Sorting System Based on Combination of Robotic Arm and Vision Machine for Agricultural Products," *Social Science Research Network*, pp. 1–13, May 2025, doi: 10.2139/SSRN.5253197.

[12] S. B. Nikam, S. B. Lande, G. C. Wakchaure, V. J. Nagalkar, and S. M. Mukane, "ADQS-YOLO: Automatic Dragon Fruit Quality Classification and Sorting Mechanism Using YOLOv8n Model," Sep. 2025, doi: 10.21203/RS.3.RS-7171300/V1.

[13] H. Zhao *et al.*, "Real-Time Object Detection and Robotic Manipulation for Agriculture Using a YOLO-Based Learning Approach," *Proceedings of the IEEE International Conference on Industrial Technology*, 2024, doi: 10.1109/ICIT58233.2024.10540740.

[14] M. Fahmi, A. Yudhana, Sunardi, A.-N. Sharkawy, and Furizal, "Classification for Waste Image in Convolutional Neural Network Using Morph-HSV Color Model," *Scientific Journal of Engineering Research*, vol. 1, no. 1, pp. 18–25, Jan. 2025, doi: 10.64539/SJER.V1I1.2025.12.

[15] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2008. Accessed: Feb. 01, 2026. [Online]. Available: https://people.disim.univaq.it/~costanzo.manes/EDU_stuff/Robotics_Modelling,%20Planning%20and%20Control_Sciavicco_extract.pdf

[16] S. E. Mathe, H. K. Kondaveeti, S. Vappangi, S. D. Vanambathina, and N. K. Kumaravelu, "A comprehensive review on applications of Raspberry Pi," *Comput. Sci. Rev.*, vol. 52, May 2024, doi: 10.1016/J.COSREV.2024.100636.

[17] B. Sebastian, "servo Robot Arm 4DOF by sebastian barrio | Download free STL model | Printables.com," Printables.com. Accessed: Mar. 25, 2026. [Online]. Available: https://www.printables.com/model/823944-servo-robot-arm-4dof

[18] A. N. Sharkawy and J. M. Nazzal, "Design and Manufacturing Using 3D Printing Technology of A 5-DOF Manipulator for Industrial Tasks," *International Journal of Robotics and Control Systems*, vol. 4, no. 2, pp. 893–909, 2024, doi: 10.31763/IJRCS.V4I2.1456.

[19] Raspberry Pi, "Buy a Raspberry Pi 5 – Raspberry Pi," Raspberry Pi Ltd. Accessed: May 10, 2026. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/

[20] J. S. D. R. G. A. F. Redmon, "(YOLO) You Only Look Once," *Cvpr*, vol. 2016-December, pp. 779–788, Dec. 2016, doi: 10.1109/CVPR.2016.91.

[21] "Jocher, G., Chaurasia, A. and Qiu, J. (2023) YOLO by Ultralytics. - References - Scientific Research Publishing." Accessed: May 10, 2026. [Online]. Available: https://www.scirp.org/reference/referencespapers?referenceid=3532980

[22] OpenMMLab, "Algorithm principles and implementation with YOLOv8 — MMYOLO 0.6.0 documentation," MMYOLO Documentation. Accessed: May 10, 2026. [Online]. Available: https://mmyolo.readthedocs.io/en/latest/recommended_topics/algorithm_descriptions/yolov8_description.html

[23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," Dec. 2016, Accessed: May 10, 2026. [Online]. Available: https://arxiv.org/pdf/1612.03144

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, Jan. 2018, doi: 10.1109/CVPR.2018.00474.

[25] "ONNX Runtime | Home." Accessed: May 10, 2026. [Online]. Available: https://onnxruntime.ai/

[26] D. McHugh, N. Buckley, and E. L. Secco, "A low cost visual sensor for gesture recognition via AI CNNS," *Intelligent Systems Conference (IntelliSys)*, 2020, Accessed: May 10, 2026. [Online]. Available: https://www.researchgate.net/publication/339101870_A_low_cost_visual_sensor_for_gesture_recognition_via_AI_CNNS

[27] N. Buckley, L. Sherrett, and E. L. Secco, "A CNN sign language recognition system with single & double-handed gestures," *Proceedings - 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC 2021*, pp. 1250–1253, Jul. 2021, doi: 10.1109/COMPSAC51774.2021.00173.

[28] E. L. Secco, D. D. McHugh, and N. Buckley, "A CNN-Based Computer Vision Interface for Prosthetics' Control," *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 440 LNICST, pp. 41–59, 2022, doi: 10.1007/978-3-031-06368-8_3.

[29] S. Procter and E. L. Secco, "Design of a Biomimetic BLDC Driven Robotic Arm for Teleoperation & Biomedical Applications," *Journal of Human, Earth, and Future*, vol. 2, no. 4, pp. 345–354, Dec. 2021, doi: 10.28991/HEF-2021-02-04-03.

[30] Z. Isherwood and E. L. Secco, "A Raspberry Pi Computer Vision System for Self-driving Cars," *Lecture Notes in Networks and Systems*, vol. 507 LNNS, pp. 910–924, 2022, doi: 10.1007/978-3-031-10464-0_63.