Implementation of YOLOv8s Computer Vision for Conveyor Counting Using Raspberry Pi 5 and Hailo-8L Accelerator

Mochammad Dimas Purwanto[1], Soraya Norma Mustika[2], Achmad Hamdan[3], Muladi[4], Richard James Dixon[5]
Electronic Systems Engineering Technology, Vocational Faculty
Malang State University, Semarang Street
, Malang, East Java, Indonesia
E-mail: mochammad.dimas.2309356@students.um.ac.id, soraya.norma.ft@um.ac.id, achmad.hamdan.ft@um.ac.id, muladi@um.ac.id

***Abstract* — Industrial automation systems require automatic, real-time object counting on conveyors. Conventional sensors have limitations in handling variations in object shapes and conditions. A deep learning-based computer vision approach can be a solution, but its implementation on embedded devices such as the Raspberry Pi 5 faces computational limitations that result in slow inference speeds. This research implements the YOLOv8s model, accelerated using the Neural Processing Unit (NPU)-based Hailo-8L on the Raspberry Pi 5, for real-time object counting. A dataset of 1,343 bottle cap images was annotated using Roboflow and divided into 70% for training, 20% for validation, and 10% for testing, with a resolution of 640 x 640 pixels. The model was trained using Google Colab, exported to ONNX format, and compiled into Hailo Executable Format (.hef) using the Hailo Dataflow Compiler. Testing was conducted at conveyor speeds of 1.6–10.0 m/min with and without the Hailo-8L. Test results showed an increase in FPS from an average of 2.2 FPS to 28.9 FPS—approximately a 13.1-fold increase. In controlled counting tests, the system with Hailo-8L achieved up to 100% accuracy, while the system without Hailo-8L experienced a decline in counting accuracy at higher conveyor speeds. Validation in an industrial environment demonstrated a counting success rate of 97%. These results indicate that the Hailo-8L's acceleration capability can reduce the YOLOv8s inference computational bottleneck and improve the real-time performance of the counting system.**

Keywords – YOLOv8s, Object Counting, Raspberry Pi 5, Hailo-8L, Conveyor

I. INTRODUCTION

Industrial automation systems continue to evolve in response to increasing demands for efficiency, quality, and speed in production processes [1]. The adoption of digitalization and automation supports improvements in efficiency and competitiveness within the manufacturing industry and drives the implementation of smart manufacturing, which integrates computing systems, sensors, and communication technologies to support production processes in real time [2] , [3].

On conveyor-based production lines, the ability to detect and count objects in real time is critical because objects can move at high speeds [4], requiring sufficient computational power to ensure an efficient detection process [5]. Proximity sensor-based counting systems have limitations because detection success is influenced by the object's position, material, and distance from the sensor [6]. Additionally, variations in conveyor speed can affect system performance [7]. These limitations have driven the adoption of computer vision, which utilizes visual information about objects for detection and identification, thereby reducing reliance on the object's physical characteristics and position relative to the sensor [8].

Advances in computer vision and deep learning have enabled the development of automated, real-time object detection systems. One widely used algorithm is YOLO (You Only Look Once), as it can detect objects in a single processing step, making it suitable for real-time applications [9]. Mahardhika and Pakereng applied YOLOv8l combined with DeepSORT for real-time object detection and tracking [10]. However, the application of deep learning models on embedded devices with limited computational resources still faces a trade-off between inference speed and detection performance. This was demonstrated by Rahmat et al. in their implementation of a deep learning model on a Raspberry Pi [11].

Advances in edge devices enable detection to be performed directly near the data source without relying entirely on the cloud. Luo et al, demonstrated that YOLOv8n can be deployed on a Raspberry Pi 5 for real-time detection in a production environment [12]. However, computational limitations on edge devices can restrict inference speed, making it necessary to use an AI accelerator based on a Neural Processing Unit (NPU) to support neural network inference [13].

Hailo-8 is an NPU-based AI accelerator designed to speed up deep learning model inference on edge devices. Harisch demonstrated the application of Hailo-8L on edge devices for a computer vision-based vehicle speed estimation system [14]. Meanwhile, Christo et al. Implemented a YOLO-based detection system on a Raspberry Pi 5 equipped with a Hailo-8L accelerator to support real-time helmet detection in an intelligent parking system [15].

Based on previous research, the implementation of YOLO on edge devices, the use of the Raspberry Pi 5 for detection in production environments, and the use of the Hailo-8L accelerator for computer vision applications have been carried out [12], [15]. However, those studies have not specifically evaluated the performance of an object-counting system on a conveyor using YOLOv8s running on a Raspberry Pi 5 with and without Hailo-8L acceleration. In addition to inference speed, variations in conveyor speed must be considered as they can affect the system's ability to maintain the detection and counting process. Therefore, this study focuses on evaluating the performance of the object counting system based on inference speed and counting success rates at various conveyor speeds.

The contribution of this research is the implementation of YOLOv8s integrated with a Raspberry Pi 5 and a Hailo-8L accelerator for a real-time object counting system on a conveyor. The evaluation was conducted by comparing the system's performance with and without the Hailo-8L based on frames per second (FPS), as well as by assessing counting accuracy at various conveyor speeds. The evaluation results were used to analyze the effect of the Hailo-8L accelerator on inference speed and counting accuracy as the conveyor speed increases.

Based on this background, this study aims to implement the YOLOv8s model to detect and count objects on a conveyor in real time, as well as to analyze the impact of using the Hailo-8L accelerator on inference speed and counting accuracy. Testing was conducted by comparing the system with and without the Hailo-8L based on the Frame per Second (FPS) value and the number of objects successfully counted at various conveyor speeds.

II. Materials and Methods

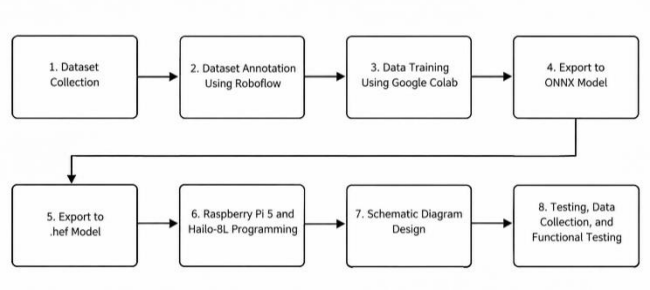


Fig. 1. Research Process

This study employs a Research and Development (R&D) methodology aimed at designing, implementing, and testing a real-time YOLOv8s-based object detection and counting system on a conveyor belt. The research stages as shown in Fig. 1 were carried out systematically through dataset collection, annotation using Roboflow, model training using Google Colaboratory, and model conversion from ONNX format to Hailo Executable Format (.hef). The compiled model was then implemented on a Raspberry Pi 5 with Hailo-8L accelerator support. The next phase involved device programming and the design of a system schematic integrating the camera, Raspberry Pi 5, Hailo-8L, and conveyor. The integrated system was then tested, data was collected, and functional tests were conducted at various conveyor speeds. The test results were used to evaluate system performance based on inference speed, FPS, and object counting accuracy, as well as to compare system performance with and without the Hailo-8L accelerator.

A. *Dataset Collection*

The dataset collection phase was conducted to obtain images of objects representing the conveyor system's operational conditions. Images were captured using a Logitech C920 camera positioned above the conveyor, 40 cm from the conveyor surface. Images were captured while the conveyor was moving at a speed of 1 m/min, thereby obtaining images of objects in different positions and orientations, as shown in Fig. 2.



Fig. 2. Sample Dataset

The dataset was collected while taking into account several variations in conditions, including the number of objects, object positions, object orientations, distances between objects, lighting conditions, and object positions on the conveyor. The dataset consists of 1,343 images containing a total of 8,224 object instances in a single class: bottle caps. The images were obtained from several capture sessions over the course of a single day using a camera positioned at a fixed distance of 40 cm from the conveyor.

B. *Dataset Annotation Using Roboflow*

The dataset annotation phase was conducted using the Roboflow platform to label each bottle cap object in the images. The dataset was uploaded to Roboflow and configured using the Object Detection project type. Each bottle cap object was annotated with a bounding box that individually encloses the object's area, as shown in Fig 3.

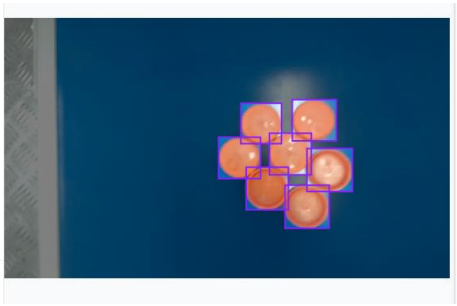


Fig. 3. Object Annotation using Roboflow

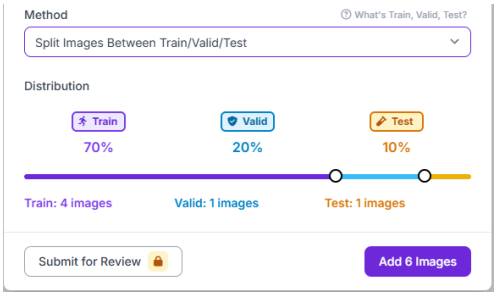


Fig. 4. Split Image

After the annotation process was complete, the dataset was divided into three parts: 70% for training, 20% for validation, and 10% for testing. The dataset splitting configuration in Roboflow is shown in Fig. 4. A random split was performed to distribute the data across each subset before it was used in the YOLOv8s model training and evaluation process. Next, the dataset was configured for use with the YOLOv8 model, and the dataset was exported. Roboflow then provided an API key used to access and retrieve the dataset directly within the Google Colaboratory environment.

C. YOLOv8s Training Data

Model training process was conducted using Google Colaboratory with the Ultralytics framework, version 8.3.241. The model used was a pre-trained YOLOv8s model, while the dataset—which had been annotated using Roboflow—was accessed via an API and used as input during the training process. Training was performed using an NVIDIA Tesla T4 GPU with CUDA support. The YOLOv8s model was trained for 100 epochs with an input image size of 640 × 640 pixels and a batch size of 16. The patience value was set to 80 epochs. The optimizer uses the "auto" configuration, with an initial learning rate (lr0) of 0.01, a final learning rate factor (lrf) of 0.01, momentum of 0.937, and weight decay of 0.0005.

The image augmentation techniques used include changes in hue, saturation, and value 0.015, 0.7, and 0.4, respectively; translation 0.1; scaling 0.5; horizontal flipping 0.5; and mosaicking 1.0. The rotation, shear, perspective, vertical flip, mixup, and copy-paste parameters are set to 0. During the inference stage, the confidence threshold is set to 0.50, so detections with a confidence value below 0.50 are not used in the tracking and counting process.

D. Export to ONNX Model

Next step, once the YOLOv8s training process is complete, is to export the best model—saved in the best.pt format—to the ONNX (Open Neural Network Exchange) format using the Ultralytics framework. The best.pt model consists of the trained weights that will serve as the basis for the conversion process. The export is performed using the model.export() function with an input size of 640x640 pixels. The corresponding code implementation in Google Colaboratory environment is shown in Fig. 5. This process generates a best.onnx file in the training directory.

The ONNX format is used because it is an interoperable model format and serves as an intermediate step in the process of compiling the model for the Hailo-8L accelerator. Therefore, conversion from `best.pt` to ONNX is required before the model is further processed using the Hailo Dataflow Compiler to generate the Hailo Executable Format (.hef).

```
# Get the path to the best trained model
best_model_path = os.path.join("runs", "detect", "train", "weights", "best.pt")

# Load the best trained model
model = YOLO(best_model_path)

# Export the model to ONNX format
model.export(format='onnx')
```

Fig. 5. ONNX Export Format

During the export process, several parameters must be considered to ensure the model meets the Hailo-8L compilation requirements. The `simplify=false` parameter is used to preserve the structure of the exported model without automatically simplifying the graph, while `dynamic=False` is used to ensure the model's input size remains static. A static input size is required because models compiled for Hailo use a predefined input format of 640 x 640 pixels. With this configuration, the `best.onnx` model is then used as input during the model compilation stage using the Hailo Dataflow Compiler.

E. Export to hef Model

After successfully exporting the YOLOv8s model to ONNX format, the next step is to compile the model into the Hailo Executable Format (.hef) so it can run on the Hailo-8L accelerator. The compilation process is performed using the Hailo Dataflow Compiler (DFC) within a WSL environment running Ubuntu 22.04. Since DFC is designed for x86-based Linux systems, using WSL allows the compilation process to be carried out on a Windows-based computer.

The first step is to download DFC version 3.33.1 from the Hailo Developer Zone and install it on an Ubuntu environment. Next, Hailo Model Zoo is installed on the same environment to support model parsing, optimization, and compilation. Version compatibility between DFC and Model Zoo must be ensured, as both are designed to be used in compatible version pairs. According to the release notes, Hailo Model Zoo version 2.18 uses Dataflow Compiler 3.33.1.

Best.onnx model is then used as input for the compilation process. In addition to the model, a set of images representing the actual conditions of the objects is prepared as calibration data. This data is used during the model optimization and quantization stages to ensure that the characteristics of the input data used during compilation align with the system's operational conditions. Once all configurations and calibration data were available, the compilation process was run using Hailo Model Zoo and DFC. This process generated the `best.hef` file, which was subsequently used during the implementation of YOLOv8s on a Raspberry Pi 5 with a Hailo-8L accelerator.

F. Programming the Raspberry Pi 5 and Hailo-8L

System was implemented on a Raspberry Pi 5 using the Python programming language. The program was designed to integrate a camera, the YOLOv8s model in .hef format, the Hailo-8L accelerator, and object tracking and counting mechanisms. In the image processing phase, OpenCV was used to handle camera input and frame processing, while the Hailo RPi 5 Examples component served as the foundation for managing the detection pipeline and processing the model's output on the Hailo-8L. The overall processing flow of the implemented system is illustrated in Fig. 6.

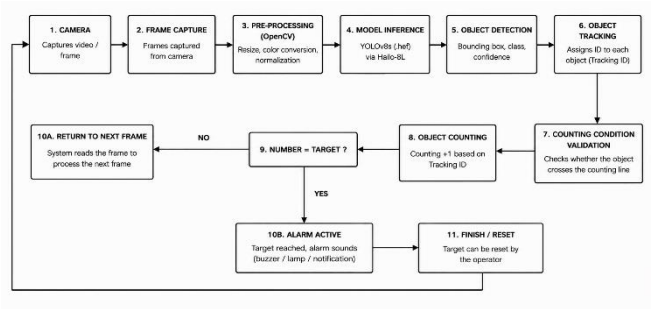


Fig. 6. Programming Flowchart

The program begins by capturing images from the conveyor. Each frame has a resolution of 640 x 640 pixels and is then fed into the YOLOv8s model on the Hailo-8L. The inference results consist of bounding boxes, class labels, and confidence scores. During the post-processing stage, Non-Maximum Suppression (NMS) is used to filter

out overlapping bounding boxes so that the detection results can be used for individual object identification.

Tracking process uses custom centroid-based tracking based on Euclidean distance. Each object is assigned a sequential tracking ID and matched with the next frame's detection based on centroid distance. Objects are considered the same if they share the same class and their centroid distance is less than 80 pixels. Objects that remain undetected for more than 1.5 seconds are removed from the tracking list. This configuration is used to preserve object identity across frames.

Counting process uses a counting zone consisting of a horizontal line at the Y-coordinate of 300 pixels with a tolerance of 20 pixels. An object is counted when its centroid falls within the range of 280–320 pixels, the system is active, and the object has not been counted before. This mechanism ensures that objects appearing across multiple frames are treated as a single object and counted only once. The settings for the confidence threshold, NMS, and tracking parameters also play a role in ensuring that nearby objects are detected and processed individually.

G. Schematic Diagram Design

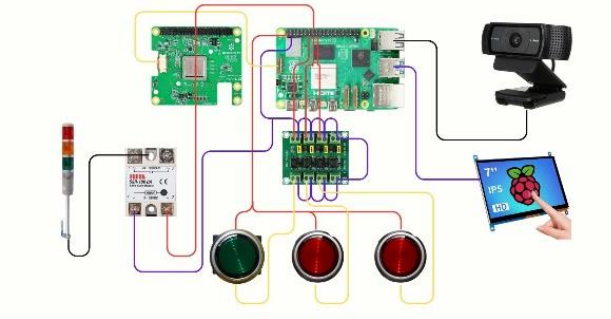


Fig. 7. Schematic Diagram

The schematic diagram illustrates the connections between the components used in the conveyor counting system, as shown in Fig. 7. The Raspberry Pi 5, integrated with the Hailo-8L, serves as the system's processing hub. A camera is used to capture images of objects on the conveyor; the image data is then processed by the YOLOv8s model via the Hailo-8L accelerator to perform object detection and counting.

System is operated using three physical buttons—START, STOP, and RESET—which are connected to the Raspberry Pi 5's GPIO pins via optocouplers, serving as an isolation interface between the button circuit and the Raspberry Pi's GPIO. The system also features a touchscreen LCD to display counting information and counting settings that can be accessed by the operator.

The Raspberry Pi also controls the SSR (solid-state relay) used to activate the alarm. When the number of objects counted reaches the specified threshold, the Raspberry Pi sends a command to the SSR, causing the alarm to sound as a notification to the operator. In this way, all components are integrated into a single system to support the automatic operation of this object detection device.

H. Testing and Data Collection

A trial phase was conducted to evaluate the performance of the object detection and counting system on the Raspberry Pi 5 by comparing the use of the Hailo-8L accelerator with and without the Hailo-8L. Testing was performed on a conveyor belt using the YOLOv8s model and the same dataset to determine the effect of the Hailo-8L on real-time inference performance. The data collected included conveyor speed, frames per second (FPS), the number of detected objects, and image resolution, with variations in conveyor speed and lighting conditions.

To ensure a fair comparison, both configurations used the YOLOv8s model, the same camera and image input, a resolution of 640x640 pixels, a confidence threshold of 0.50, the same NMS configuration, and the same custom centroid-based tracking. The conditions, number of objects, and conveyor speed were also kept consistent. The test results were used to compare the FPS, detection and counting capabilities, and system stability based on the use of the Hailo-8L.

III. Results and Discussion

This chapter presents the results of testing and analysis of a YOLOv8s-based object detection and counting system implemented on a Raspberry Pi 5 with and without the Hailo-8L accelerator. The discussion aims to determine the model's performance and the impact of the accelerator on the system's ability to detect and count objects in real time. The research results were analyzed based on several parameters, including the performance of the YOLOv8s model, the number of objects successfully detected at various conveyor speeds, the Frame per Second (FPS) value, and the effect of varying input resolutions on the system's processing capabilities. All test results were then compared and analyzed to determine the relationship between computational capability, object movement speed, and the success of the counting process in the developed system.

A. YOLOv8s Model Training Results

After training the model using YOLOv8s, an evaluation was conducted on the validation data to assess the model's ability to detect bottle caps. The validation data consisted of 268 images containing 1,650 objects. The evaluation was performed using precision, recall, mAP@50, and mAP@50–95. The evaluation results are shown in Table I.

TABLE I. Evaluation Metrics

NO	Matrik Evaluasi	Nilai
1	*Precision*	0,9865
2	*Recall*	0,9867
3	*mAP@50*	0,9915
4	*mAP@50-95*	0,9245

Based on Table I, the YOLOv8s model achieved a precision of 0.9865 and a recall of 0.9867. These values indicate that the model was able to successfully predict and detect the majority of bottle cap objects in the validation dataset. An mAP@50 value of 0.9915 was obtained at an IoU of 0.50, while mAP@50–95 of 0.9245 represents the average precision across IoU values from 0.50 to 0.95 in 0.05 intervals. The lower mAP@50–95 value compared to mAP@50 indicates that the evaluation becomes more stringent as the precision of the bounding box position is increased.

Further evaluation was conducted using a confusion matrix, a precision-recall curve, and an F1-confidence curve. Based on the confusion matrix, 1,630 bottle cap objects were successfully detected, 20 objects were not detected (false negatives), and 30 bottle cap predictions were made in the background area (false positives). The results of the normalized confusion matrix show that approximately 99% of bottle cap objects were successfully identified, while about 1% were not detected. In the visual sample of the prediction results, the bounding boxes appear to have been successfully placed on the bottle cap objects with high confidence scores.

The Precision-Recall Curve shows an mAP@50 value of 0.991, while the F1-Confidence Curve shows an F1 score of approximately 0.99 at a confidence level of approximately 0.454. In the system implementation, the confidence threshold is set at 0.50, so predictions with a confidence score below 0.50 are not used in the tracking and counting process.

Overall, the evaluation results show that the YOLOv8s model exhibits high performance in detecting bottle caps on the validation dataset. The model was then deployed on a Raspberry Pi 5 for a bottle caps-counting system, and comparisons were made between CPU-based processing and the Hailo-8L accelerator.

B. Counting Results With and Without an Accelerator

The tests were conducted to compare the system's ability to detect and count objects with and without an accelerator. Each experiment used 10 bottle caps and 640 x 640-pixel images, with several variations in conveyor speed. For each conveyor speed variation, 5 tests were conducted, yielding 5 count results, which were then averaged.

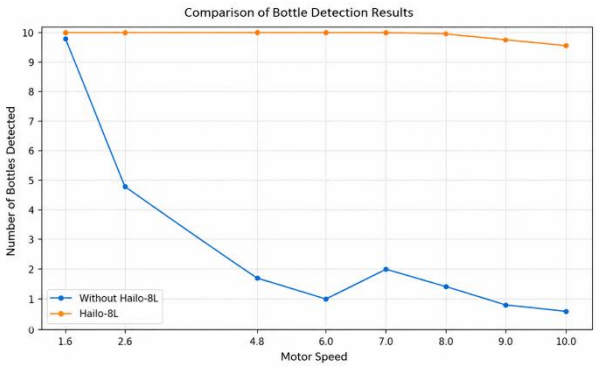


Fig. 8. Comparison of Bottle Cap Detection Results

Values in Fig. 8 represent the average number of bottle caps successfully counted from five tests at each speed variation. The system using the Hailo-8L was able to maintain relatively consistent counting results across all speed variations. The average number of objects successfully counted was 10 at speeds ranging from 1.6 to 8.0 m/min, then dropped to 9.8 at 9.0 m/min and 9.6 at 10 m/min.

In contrast, the system without the Hailo-8L showed a decrease in the average number of successfully counted objects as the conveyor speed increased. The average number of successfully counted objects was 9.8 at 1.6 m/min, then continued to decrease at speeds ranging from 2.6 to 10 m/min. The results indicate that the system without an accelerator experiences a decline in its ability to maintain the object detection and counting process as conveyor speed increases.



Fig. 9. Test Results with Hailo-8L

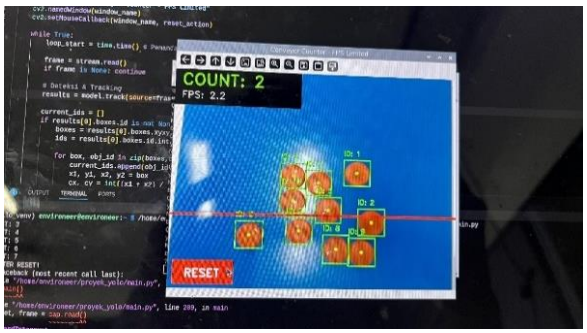


Fig. 10. Test Results without Hailo-8L

The frame processing performance of the system was also observed through the FPS displayed on the Raspberry Pi. The FPS documentation for the system using the Hailo-8L is shown in Fig. 9, while the corresponding FPS documentation for the system without the Hailo-8L is shown in Fig. 10. These observations provide a visual comparison of the frame processing performance under both configurations.

Based on the average of five test runs, the system with Hailo-8L produced more consistent object counts compared to the system without an accelerator. Hailo-8L acts as an inference accelerator on the Raspberry Pi 5, allowing frame processing to occur more quickly. This helps maintain continuous detection and tracking when objects move at higher conveyor speeds.

C. FPS Analysis of Conveyor Speed

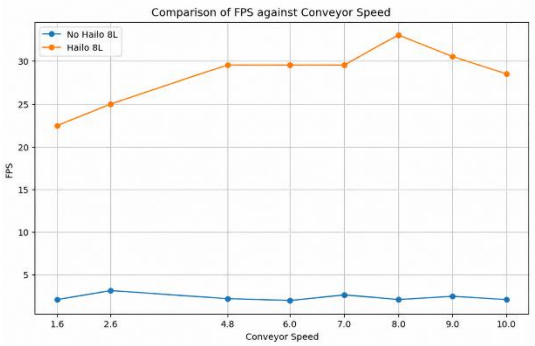


Fig. 11. Comparison FPS Results

FPS testing was conducted to determine the effect of using the Hailo-8L accelerator on the inference performance of YOLOv8s on a Raspberry Pi 5 with an image resolution of 640x640 pixels. As shown in Fig 11, the system without the Hailo-8L produced an FPS in the range of 2.2–3.0 FPS, while the system with the Hailo-8L produced 22.5–33 FPS across all conveyor speed variations. The highest FPS value for the system with the Hailo-8L was obtained at a conveyor speed of 8.0 m/min. The performance improvement was calculated using the following equation:

$$[Speedup = \frac{FPS_{Hailo}}{FPS_{CPU}}] \quad (1)$$

$$[Speedup = \frac{28,9}{2,2} = 13,14 \times \approx 13,1 \times] \quad (2)$$

The equivalent processing time per frame is calculated based on the relationship between FPS and processing time:

$$[T_{frame} = \frac{1000}{FPS}] \quad (3)$$

Therefore, we obtain:

$$[T_{CPU} = \frac{1000}{2,2} \approx 454,5 \text{ ms/frame}] \quad (4)$$

$$[T_{Hailo} = \frac{1000}{28,9} \approx 34,6 \text{ ms/frame}] \quad (5)$$

Results show that using the Hailo-8L increases inference throughput by approximately 13.1 times and results in a lower equivalent processing time per frame. This improvement in inference performance enables more responsive object detection and counting on the conveyor.

D. Effect of Input Resolution on FPS

Testing was conducted under two scenarios to evaluate the impact of system configuration on computer vision performance: the effect of resolution on FPS in Test A and the effect of camera height on counting accuracy in Test B.

Test A was conducted to determine the effect of changes in input resolution on the system's processing speed. In this test, the camera was kept at a fixed height of 40 cm above the conveyor surface, while the input resolution was varied. The test was conducted on a system using a Raspberry Pi 5, both with and without the Hailo-8L.

TABLE II. EFFECT OF RESOLUTION ON FPS

NO	Test Table A			
	Resolution	***Max Camera Height***	***FPS Without Hailo-8L***	***FPS Hailo-8L***
1	320x320 Piksel	40 cm	5,5	35
2	480x480 Piksel	40 cm	3,2	32
3	640x640 Piksel	40 cm	2.1	33

Based on Table II above, increasing the input resolution causes a decrease in FPS on systems without the Hailo-8L; FPS decreases as the input load increases. This decrease indicates that increasing the input resolution affects processing on the Raspberry Pi 5 without an accelerator. In contrast, systems using the Hailo-8L maintain relatively stable FPS at every given input resolution. Test A results show that the Hailo-8L is capable of maintaining inference speed when the input resolution is increased

Test B was conducted at a fixed resolution of 640x640 pixels with a conveyor speed of 4.8 m/min, using 10 bottle caps as test objects. Each camera height variation was tested three times, and the results from the three tests were averaged to obtain the calculated values and accuracy.

TABLE III. HIGH-RESOLUTION CAMERA TEST RESULTS FOR COUNTING

NO	Table Test B				
	Camera Height	***Hailo-8L Results***	***Accuracy Hailo-8L***	***Results Without Hailo-8L***	***Accuracy Without Hailo-8L***
1	23 cm	10,0	100%	2,0	20%
2	25 cm	10,0	100%	2,0	20%
3	30 cm	10,0	100%	1,33	13,33%
4	40 cm	9,67	96,67%	1,67	16,67%

Based on Table III above, the system with Hailo-8L achieved a calculation accuracy of 100% at camera heights of 23, 25, and 30 cm, and 96.67% at a height of 40 cm. Meanwhile, the system without the Hailo-8L achieved accuracies of 20%, 20%, 13.33%, and 16.67%, respectively. These results indicate that at a resolution of 640x640 pixels and a conveyor speed of 4.8 m/min, the use of the Hailo-8L is capable of maintaining high counting accuracy across the range of camera heights tested.

E. Implementation of the System on the Production Line

After undergoing testing, the system was implemented directly on the production line of a company in Surabaya that specializes in plastic bottle packaging. The system is used to automatically detect and count bottle caps according to the quantity specified by the operator. The system is equipped with start, stop, and reset buttons, as well as a touchscreen used to monitor the number of objects and set the counting target for each carton. The implementation and operation of the system on the production line are shown in Fig. 12.



Fig. 12. Equipment Used on the Production Line

Validation was conducted over 3 days during the production process at a capacity of 4 cartons. Each carton contained 1,500 bottle caps as ground truth, resulting in

approximately 12 test cycles with a total of 18,000 objects at a conveyor speed of 5 m/minute. Based on the evaluation results, the system achieved a 97% success rate. Equivalently, this figure indicates that approximately 17,460 objects were successfully counted in accordance with the actual count, while approximately 540 objects represented a counting discrepancy, consisting of both potential undercounts and overcounts.

F. Comparison with Previos Research

A comparison of the system with previous research is shown in Table IV to examine the performance of the Raspberry Pi, YOLO, and the accelerator in computer vision applications. The comparison includes FPS and detection counting results.

TABLE IV. COMPARISON WITH PREVIOUS RESEARCH

NO	Comparison Table				
	Research	***Equipment and Methods***	***FPS***	***Results***	***Accel erator***
1	Luo *et al.* [12]	Raspberry Pi 5 + YOLOv8n, Product detection on a conveyor belt	2,7	Accuracy 99,3 %	No
2	Pasache *et al.* [16]	Raspberry Pi 5 + custom lightweight CNN	-	Accuracy 99,6%, processing time 12.4 ms/seed	No
3	Veena Devi *et al.* [17]	Raspberry Pi 4 + YOLOv5s, Conveyor belt classification	5-6	Sorting 94-96%	No
4	Arianta *et al.* [18]	Raspberry Pi 5 + YOLOv8s, deteksi kerusakan jalan	27	Accuracy 92,96%	Hailo -8L
5	This Research	Raspberry Pi 5 +YOLOv8s, *counting conveyor*	28,9	P 98,65%; R 98,67%; mAP@50 99,15%; 100 % Counting at 1,6-8,0 m/min	Hailo -8L

According to Table IV, the system without an accelerator on the Raspberry Pi produced 0.5–6 FPS, whereas Arianta et al. [18]. Used a Raspberry Pi 5, YOLOv8s, and Hailo-8L to achieve an average of 27 FPS. This study achieved an average of 28.9 FPS with the Hailo-8L and 2.2–3.0 FPS without an accelerator, representing a 13.1-fold increase. The difference in FPS is influenced by the model, resolution, preprocessing, postprocessing, and the tracking-counting stages in each pipeline.

YOLOv8s model used in this study achieved a precision of 98.65%, a recall of 98.67%, an mAP@50 of 99.15%, and an mAP@50-95 of 92.45%. These metrics reflect detection performance, whereas counting is also influenced by tracking and object movement. With the Hailo-8L, counting averaged 10 objects at 1.6–8.0 m/min, then 9.8 at 9 m/min, and 9.6 at 10 m/min. This decrease demonstrates a trade-off between conveyor speed and counting accuracy. Missed detections can be caused by faster object movement, position changes, overlap/occlusion, and inconsistent bounding boxes. Variations in camera height also have an effect.

Overall, the Hailo-8L improves throughput and helps maintain counting consistency. In an industrial setting, the system was tested on 18,000 bottle caps and achieved a success rate of approximately 97%. The system's limitations include reduced counting accuracy due to sensitivity to camera position and distance, as well as missed detections caused by overlap.

IV. CONCLUSION

Based on the research results, YOLOv8s was successfully implemented for the detection and counting of bottle caps on a Raspberry Pi 5-based conveyor, achieving a precision of 0.986, a recall of 0.986, an mAP@50 of 0.991, and an mAP@50–95 of 0.924. The use of the Hailo-8L accelerator increased the system's throughput from an average of 2.2 FPS to 28.9 FPS—approximately 13.1 times higher than processing without an accelerator. In counting tests, the system with the Hailo-8L was able to count an average of 10 objects at conveyor speeds of 1.6–8.0 m/min, while at a speed of 9 m/min the average number of counted objects decreased to 9.8, and at 10 m/min it dropped to 9.6. Validation in an industrial environment, conducted over three days with 12 cycles and a total of 18,000 bottle caps at a conveyor speed of 5 m/min, showed a system success rate of 97%. However, the system still has limitations, such as a decrease in counting accuracy influenced by the camera's position and distance from the objects, as well as the possibility of missed detections due to changes in position and overlap between objects. Therefore, future research should improve the robustness of the tracking and counting processes against variations in conveyor speed, camera position, and object density, as well as conduct testing under a wider range of operational conditions.

REFERENCES

[1] A. F. Dewi, J. Jamaluddin, M. Basyir, and M. Farhan Ali, "Simulasi Sistem Otomasi Penyortiran dan Pengemasan Barang Berbasis Warna," *Jurnal Litek : Jurnal Listrik Telekomunikasi Elektronika*, vol. 22, no. 2, pp. 157–162, Oct. 2025, doi: 10.30811/litek.v22i2.85.

[2] A. M. Maulan, G. L. D. Saputra, R. B. Syahputra, S. M. R. Hidayat, and Y. Prastyo, "Pengaruh Sistem Otomasi Robotik Terhadap Proses Produksi Mencakup Efisiensi Biaya, Waktu, Dan Kualitas Produk Menggunakan Metode Analisis Data Statistik," *GLOBAL: Jurnal Lentera BITEP*, vol. 3, no. 1, pp. 8–16, 2025, doi: 10.59422/global.v3i01.720.

[3] C. R. H. Barbosa, M. C. Sousa, M. F. L. Almeida, and R. F. Calili, "Smart Manufacturing and Digitalization of Metrology: A Systematic Literature Review and a Research Agenda," *Sensors*, vol. 22, no. 16, p. 6114, 2022, doi: 10.3390/s22166114.

[4] P. Buranajarukorn, S. Thongtab, J. Virunbut, I. Luanghan, and P. Apichayakul, "Benchmarking

ROI Strategies for Real-Time Chicken Counting Using YOLOv8 and LLM-Assisted Development in Industrial Slaughterhouses," *Naresuan University Engineering Journal*, vol. 20, no. 2, pp. 20–30, 2025.

[5] D. Varna and V. Abromavičius, "A System for a Real-Time Electronic Component Detection and Classification on a Conveyor Belt," *Applied Sciences (Switzerland)*, vol. 12, no. 11, Jun. 2022, doi: 10.3390/app12115608.

[6] A. Djafar, R. Gunawan, H. D. Haryono, and D. Suanggana, "Efektifitas Respon Sensor Proximity Induktif dalam Menyortir Pecahan Logam pada Model Conveyor," *Jurnal Serambi Engineering*, vol. 8, no. 1, pp. 4492–4499, 2023, doi: 10.32672/jse.v8i1.5126.

[7] S. C. Putra, "Pengaruh Variasi Kecepatan Konveyor Terhadap Waktu Operasional Dan Pemilah Logam Dan Non Logam," *Jurnal Informatika dan Teknik Elektro Terapan*, vol. 13, no. 3S1, Oct. 2025, doi: 10.23960/jitet.v13i3S1.7503.

[8] L. Dai *et al.*, "A new machine vision detection method for identifying and screening out various large foreign objects on coal belt conveyor lines," *Complex and Intelligent Systems*, vol. 9, no. 5, pp. 5221–5234, Oct. 2023, doi: 10.1007/s40747-023-01011-9.

[9] T. Diwan, G. Anirudh, and J. V. Tembhurne, "Object detection using YOLO: challenges, architectural successors, datasets and applications," *Multimed. Tools Appl.*, vol. 82, no. 6, pp. 9243–9275, Mar. 2023, doi: 10.1007/s11042-022-13644-y.

[10] D. S. Mahardhika and M. A. I. Pakereng, "Deteksi Objek Secara Real-Time Berbasis YOLOv8 dan Algoritma DeepSORT," *Jurnal Media Computer Science*, vol. 5, no. 1, pp. 671–686, 2026, doi: 10.37676/jmcs.v5i1.10038.

[11] H. Rahmat, S. Wahjuni, and H. Rahmawan, "Performance Analysis of Deep Learning-based Object Detectors on Raspberry Pi for Detecting Melon Leaf Abnormality," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 12, no. 2, pp. 572–579, Apr. 2022, doi: 10.18517/ijaseit.12.2.13801

[12] K. Luo *et al.*, "An optimized YOLOv8n based model for real time defect detection in taro strip production," *Sci. Rep.*, vol. 15, no. 1, Dec. 2025, doi: 10.1038/s41598-025-28216-x.

[13] M. Knocker, M. Sontheimer, G. G. Machado, and S. Y. Chou, "Democratizing UAV-Based Search and Rescue: Real-Time Human Detection on Affordable Edge AI Hardware," in *Advances in Transdisciplinary Engineering*, IOS Press BV, Oct. 2025, pp. 72–81. doi: 10.3233/ATDE251079.

[14] G. Bueno, L. Sanchez-Vargas, A. Diaz-Maroto, J. Ruiz-Santaquiteria, M. Blanco, J. Salido, and G. Cristobal, "Real-Time Edge Computing vs. GPU-Accelerated Pipelines for Low-Cost Microscopy Applications," *Electronics*, vol. 14, no. 5, Art. no. 930, 2025, doi: 10.3390/electronics14050930.

[15] A. A. Christo, M. B. P. Utama, Y. A. Camme, D. Eridani, and P. E. Mountaines, "YOLO-Based Helmet Detection and Intelligent Parking System: A Case Study at Diponegoro University," *International Journal of Safety and Security Engineering*, vol. 15, no. 5, pp. 1075–1088, May 2025, doi: 10.18280/ijsse.150520.

[16] H. Pasache, C. Tuesta, and C. Inga, "Design of an Automated System for Classifying Maturation Stages of *Erythrina edulis* Beans Using Computer Vision and Convolutional Neural Networks," *AgriEngineering*, vol. 7, no. 9, p. 277, 2025, doi: 10.3390/agriengineering7090277.

[17] S. V. Veena Devi, K. Sowmya Nag, Ravikant, S. Nayak, H. R. Kiran, B. V. Manoj Kumar, and A. S. Yogeesh, "Classification of cashew kernel using deep learning," *International Journal of Food Properties*, vol. 29, no. 1, Art. no. 2699529, 2026, doi: 10.1080/10942912.2026.2699529.

[18] I. W. O. Arianta, S. T. P. E. Sitompul, G. C. M. Dharmawan, N. I. ER, L. Jasa, W. Setiawan, and I. P. E. D. Nugraha, "Perancangan Purwarupa Alat Deteksi Kerusakan Jalan Berbasis Machine Learning," *Jurnal SPEKTRUM*, vol. 12, no. 2, pp. 9–21, 2025.