Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Evaluation and Comparison of Real-Time Robot Perception Capabilities in a 24/7 Supermarket

Hamsa Datta Perur

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Erwin Prassler
Prof. Dr. Sebastian Houben
Santosh Thoduka, M.Sc.

January 15 2023

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Hamsa Datta Perur

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Prof. Dr. Erwin Prassler, Prof. Dr. Sebastian Houben, and Santosh Thoduka for their invaluable guidance and support throughout this R&D project. I am particularly thankful to Santosh for his help at various stages of this project.

I would also like to thank Vamsi, Vivek, Kishan, Ragini, and Kevin for their assistance in setting up the experiment scenario. Additionally, I would like to acknowledge the constructive discussions and support from my friends at HBRS. Lastly, I would like to extend my appreciation to my family for their unwavering moral support during the project.

Abstract

The increasing use of robots in retail, particularly for logistics, has brought about significant investments in robot-based technologies in the grocery industry. However, automating logistical tasks in supermarkets remains a challenge. To study this issue, the deployment of robots in supermarkets during off-peak hours to assist customers with their shopping has been considered. This project focuses on evaluating and comparing the perception cycle time of modern object detectors in a supermarket setting and identifying factors that affect the cycle time. The project aims to investigate the use of real-time detectors like YOLOv7, YOLOv5, NanoDet-Plus, and DETR, in a supermarket scenario and evaluate the performance over different hardware configurations. Based on the above comparison results, the project also aims to analyze if shared task execution between humans and robots is feasible.

The first three experiments investigate cycle time using four selected object detectors over three different hardware like high-end GPU, embedded GPU, and Toyota HSR platform. In the final fourth experiment, the project aims to analyze the effect of various factors over cycle time using the YOLOv7 detector. The results found that the YOLOv5 detector performed the best in achieving low cycle time and maintaining good accuracy. If shared task execution is implemented in a supermarket scenario, it is best done with detectors like YOLOv5. The study also found that lightweight models like NanoDet-Plus performed well in cycle time while maintaining reasonable accuracy. Factors such as image size and blur were found to affect the cycle time directly.

Finally, the project emphasizes the importance of considering multiple factors when selecting a detector for use in a supermarket setting. These factors include the detector's ability to detect a wide range of objects with different orientations and with different scales.

Contents

List of Figures

List of Tables

1 Introduction

Robots are being utilized increasingly often in the retail sector, primarily for logistics, but it is still challenging to automate many supermarket-based tasks [1]. The grocery business is aggressively increasing its investments in robot-based technologies and clearly with much confidence [2]. According to Juniper Research [3], around \$3.6 billion of investment in artificial intelligence is expected by retailers globally, with an additional \$12 billion predicted by 2023. Deploying the robots during night time in a supermarket, and enabling the customers for buying groceries is one of the potential solutions to increase revenue by utilizing robotic technologies. However, robots face many challenges during deployment in supermarket scenarios. For example achieving low perception cycle time [1] still needs to be solved. In this project, we evaluate and compare modern object detectors and mainly focus on factors influencing the perception cycle time.

1.1 Motivation

In this project, we consider supermarkets as our target environment. Figure 1.1 shows the overview of the project and also describes the various tasks that are associated. In this project, we focus primarily on investigating the cycle time of the perception pipeline of the robot. Based on our cycle time analysis, we also explore if share task execution is feasible between humans and the robot (discussed more in section 4). After this project, we can accomplish the following:

- By decomposing and comparing different detection methods, we can make an informed decision about which part of the algorithm takes more time. We can use this information to optimize the total perception cycle time.

[1]Cycle time: the total time allotted for executing a task. We also use this interchangeably with total inference time in an object detection scenario.

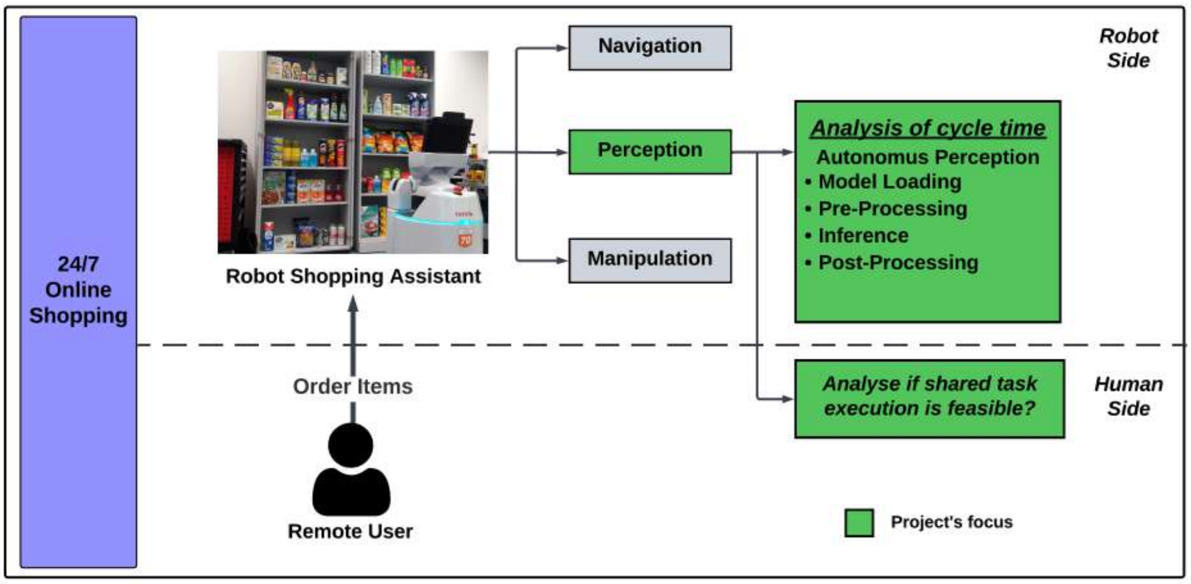


Figure 1.1: Project overview diagram: This project considers a 24/7 online shopping scenario in which a remote user tries to shop using a robot. The project focuses primarily on investigating the perception cycle time. Based on our cycle time analysis, we also explore if share task execution is feasible between humans and the robot.

- Identify the various factors that affect the perception cycle time.

1.2 Challenges and Difficulties

Some of the general problems in real-time object detection in supermarkets include:

- Large and varied product catalog [4]: Supermarkets often have various goods in various sizes, colors, and forms. Because of this, it may be challenging for an object detection model to recognize and categorize each product correctly.
- Lighting and viewpoint variations [5]: Depending on the different working hours and the location within the supermarket, the illumination might change. Additionally, depending the camera location and orientation, the robot's point of view might change while taking pictures [5]. The model may struggle to recognize and categorize items effectively due to these changes.
- Occlusion and clutter [4]: Products in a supermarket are often closely packed together on shelves, which can lead to occlusion, where one product partially or entirely

blocks the view of another product [4]. Because of this, it may be challenging for an object detection model to recognize and categorize certain items. Furthermore, other things, such as shoppers and shopping carts, might add clutter and make it challenging for the system to focus on the essential elements.

- High throughput [6]: An object identification model must be able to process a large volume of pictures quickly to keep up with the flow of consumers and the movement of goods inside the shop. This may need a robust hardware configuration and effective algorithms to guarantee that the model can manage the necessary workload [6].

- Limited training data: Supermarkets carry a wide variety of products, and it can be difficult to collect a large and diverse enough dataset to train an object detection model that can accurately identify and classify all of the different products [4].

- Limited resources [6]: Real-time object detection systems may have limited processing power and memory, which can make it challenging to perform object detection in real-time on resource-constrained devices.

- Label noise and incorrect labels [5]: It is crucial to check that the labels given to each image are correct and accurately reflect the details of an image when gathering training data. However, this can be hard to achieve, as in a real-world the data is being collected by different people. As a result, the training dataset could include inaccurate labels and label noise, that could degrade how well the object detection model performs [5].

- Adversarial examples [6]: A machine learning model's adversarial examples are inputs that are intended to deceive the model and end up causing it to make false predictions. In the context of object detection in supermarkets, an adversarial example could be a product that has been deliberately modified to look like a different product, with the aim of tricking the object detection model into making an incorrect classification.

- Privacy concerns: In a supermarket setting, object detection models are likely to be capturing images of customers and their personal belongings. This gives rise to privacy issues, thus it's critical to make sure the object detection system is built to preserve consumer privacy and adhere to all legal and regulatory requirements.

1.3 Project Relevance

Long perception cycle times can lead to slower performance, decreased efficiency for the robot, and inconvenience for customers waiting for the robot to complete its tasks. So investigating the cycle time is more relevant in the context of online supermarkets. Below are some benefits and market trends highlighting the project's relevance.

- Benefits
  - Allow customers to shop 24/7, even during the night times, which in turn brings additional revenue to supermarkets/retail.
  - By analyzing perception cycle time, developers can identify factors that may be contributing to longer cycle times and can address these issues and improve the shopping experience for customers.
- Market trends
  - According to new John Lewis Partnership Card data, one in every 15 purchases is now done between the hours of midnight and 6:00 a.m [7].
  - According to an interview conducted by BBC, many women say that "bedtime is the only period in the day when they have the time to go online to do their shopping, whether it is for essential or spur-of-the-moment purchases" [8] .
  - In a survey done by Eachnight over 1000 people, it is found that 19.1% of the average nocturnal spending goes towards food and groceries [9].
  - According to Korn Ferry's forecasts [10], by 2030, there may be a shortage of skilled workers to fill 85 million jobs globally.
  - Deloitte estimates that the global retail industry will need at least 1 million additional laborers (in general during the day) in addition to the 15.7 million existing employees in order to thrive [11].

2

Background

This chapter covers the topic of object detection, including real-time object detection and the general differences. It provides a classification of different object detection methods and briefly discusses several key meta architectures. The chapter also explains standard object detection metrics and other terminology related to the field. This section's structure is based on many survey papers [12], [5], [6], [13], [14] referred.

2.1 Object Detection

Object detection is a computer vision task that involves "identifying the presence, location, and type of objects in an image or video" [14]. It is a key technology underlying many applications, such as self-driving cars, security systems, and image or video search engines. The various stages of object detection (also shown in Figure 2.1) are as follows:

- Input: A video frame or an image serves as the input for an object detection system.
- Preprocessing: The input is typically preprocessed to remove noise, resize or crop the image, and perform other image processing operations.
- Feature extraction: After the image has been preprocessed, it moves on to the step of feature extraction where features like borders, corners, and texture sequences are retrieved and utilised to represent the image.
- Classification: The extracted features are then passed through a classifier, which is a machine learning model trained to recognize specific objects. The classifier outputs a list of detected objects, along with their locations and class labels (e.g. "car", "person", "dog").

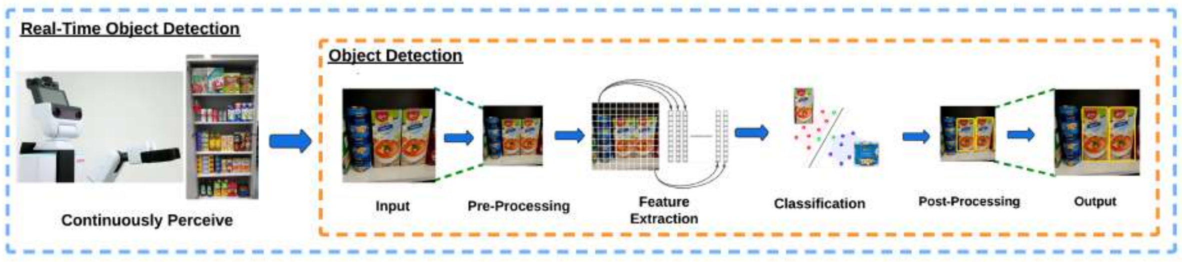


Figure 2.1: Object detection pipeline using traditional methods.
Note: Image adapted from A. C. Hernandez et al. [15]

- Post processing: The output of the classifier is typically post processed to filter out false positives, combine overlapping detections, and perform other operations to improve the detection of the object detections.

- Output: The final output of the object detection system is a list of detected objects, along with their locations, class labels, and other relevant information.

2.2 Real-Time Object Detection

Real-time object detection refers to the ability of a system to detect and locate objects in images or videos in real-time, as the images or videos are being captured (shown in Figure 2.1). It is an important capability in applications such as self driving vehicles, monitoring systems, and augmented reality, where the system needs to respond to the environment in real time.

Real-time object detection and object detection show differences in how quickly and thoroughly the system analyses the incoming photos or videos.

- In object detection, the system processes the input images or videos to detect and locate objects, but the processing speed is not necessarily a critical factor. The system may require some time to go through the pictures or videos, and the results could come out later than expected. This is suitable for applications where the processing speed is not a critical factor, such as image search engines or offline video analysis.

- To achieve real-time performance, the system must optimize various components such as the pre-processing and feature extraction steps, and use efficient algorithms and hardware architectures to enable fast object detection. This may involve trade-offs in terms of the accuracy or robustness of the object detection algorithm, as faster algorithms may not always be as accurate or robust as slower algorithms.

- In this project, we evaluate different object detectors (see section 5) and compare the performance over the cycle time. In order to do so in a real world, we need to set the cycle time deadline to decide whether the detector satisfies the real-time condition. Setting the proper deadline for the cycle time is based on the context of the application of the robot. In our case, the context is the online shopping scenario. So it must be the responsibility of the developer to set the deadline based on the online shopping context. For example, we need a shorter cycle time deadline when a customer is trying to shop for a product. In this project, we aim to provide an estimate based on experiments.

2.3 Taxonomy of Object Detectors

Over the years, various object detection methods have been developed, each with strengths and limitations. It is helpful to organize these methods into a taxonomy or a hierarchical classification system to understand better and compare them. In this taxonomy, object detectors can be grouped based on their underlying principles. Figure 2.2 shows the different methods of how object detection can be realized.

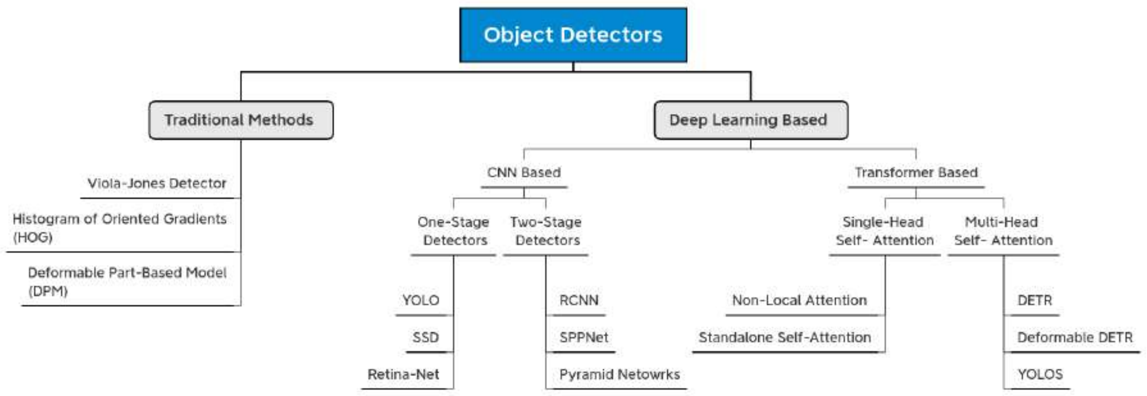


Figure 2.2: Taxonomy of Object Detectors.
Note: Image adapted from K. Salman, et al. [13], M. Payal, et al. [16]

2.3.1 Object Detection Using Traditional Methods

Classic object detection frequently uses sliding windows to search for specific objects in a picture. The classifier makes predictions for each window, and a post-processing step removes false positives and integrates overlapping detections. Below are some of the standard object detectors that use traditional methods.

- Viola-Jones Detector (by P. Viola et al. [17]): uses a simple method of detection called the sliding window approach, which involves going through all potential positions and scales in a picture to determine whether any of the windows include a human face. It appears to be a straightforward operation [12], the calculations required were well above the capabilities of the computer at the time.

- Histogram of Oriented Gradients (HOG) (by N. Dalal et al. [18]): is an important object detector and is used as a foundation for several other detectors. The HOG descriptor calculates on a dense matrix of uniformly spaced cells with overlapped contrast enhancement normalisation for increased accuracy [12] and also detects different classes of objects [12].

- Deformable Part-based Model (DPM) (by P. Felzenszwalb et al. [19]): extending the HOG detector the DPM approach is based on “divide and conquer”.The inference may be thought of as an ensemble of detections on distinct object pieces, and “the training can be thought of as the learning of a valid manner of object decomposition” [12].

2.3.2 Object Detection Using CNN-Based Methods

CNN-based object detection uses convolutional neural networks (CNNs) to categorize object proposals (or re. Popular techniques in this field, such as R-CNN and its derivatives (Fast R-CNN, Faster R-CNN), involve using a CNN to extract features from an image which are then used to train a different classifier to predict bounding boxes and corresponding labels for the objects. Some of the popular CNN-based methods are as follows:

- Regions with CNN features (RCNN) (by R. Girshick et al. [20]): Region proposals in object detection are a method for identifying potential regions or regions of interest

(ROIs) in an image that may contain objects. The working of RCNN is simple. The technique begins with a selective search to identify a set of object recommendations (object candidate boxes) [12]. The features are extracted using a CNN model trained on a deep neural network.

- You Only Look Once (YOLO) (by R. Joseph et al. [21]): works by using a single neural network to process the entire image. This network splits the picture into sections and parallelly predicts bounding boxes and probabilities for each region. This is an extremely fast algorithm [12]. YOLO makes predictions for multiple scales of the image, which allows the detection of objects of different sizes. Additionally, it uses anchor boxes to adjust the shape of the bounding boxes to fit the object better.

- Single Shot MultiBox Detector (SSD) (by W. Liu et al. [22]): The implementation of "multi-reference and multi-resolution detection techniques" [12], which considerably increasing the detection performance of a one-stage detector, "especially for some tiny objects, is the main key contribution" [12]. SSD has a good performance on accuracy.

2.3.3 Object Detection Using Transformer-Based Methods

Transformer-based object detection is a recent development that utilizes transformer networks, such as the famous Vision Transformer (ViT) architecture. These methods use a transformer network to directly process the entire image and predict object bounding boxes and class labels in a single step, bypassing the need for object proposals or region-based feature extraction.

- Detection Transformer(DETR) (by C. Nicolas et al. [23]): Unlike CNN-based methods that require region proposals and anchors, DETR uses a combination of a CNN network and a transformer encoder-decoder architecture to simultaneously predict all objects in an image [24]. It is trained in an "end-to-end manner" [23] using a loss function that matches the predicted and ground-truth objects [25]. With this strategy, DETR can reduce the requirement for region proposals and anchors, which may enhance the effectiveness and precision of object identification systems [24].

- Deformable DETR (by X. Zhu et al. [26]): Deformable DETR is an improved method over DETR for object detection. The DETR suffers from spatial resolution problems and performs poorly with small objects. As the DETR uses the basic transformer attention, the DETR model takes more time for convergence during the training process [26]. To overcome this problem, deformable DETR focus solely on a selected number of significant sample points around a reference [26]. The authors employ an iterative improvement method for the bounding boxes to process the feature maps, thus decreasing the training time up to ten times [26].

- You Only Look at One Sequence (YOLOS) (by Y. Fang et al. [27]): YOLOS is another object detection method that uses pure vision transformers, which improves over DETR. YOLOS uses the basic vision transformer architecture and works with minimum modifications [27]. The modifications in the architecture are based on updating the classification and detection tokens that the basic transformer uses in a natural language processing task [27]. The authors of YOLOS utilize the same loss functions as done in DETR. YOLOS demonstrates object detection tasks with less spatial information in a sequential manner [27].

2.4 Object Detection Metrics

Below are the popular metrics that are extensively used in object detection tasks.

- Precision and recall: Precision measures the proportion of correct detections, while recall measures the proportion of actual positive instances correctly detected.

- Intersection Over Union (IOU): As the name suggests, it is the ratio of the area of the intersection between the predicted bounding box and the ground truth bounding box over the area of the union of both. Values closer to one indicate a better match. Most cutting-edge object detection algorithms employ this measure [28].

- Average precision(AP): It sums the weighted mean of precisions for each threshold as recall increases, based on the precision-recall curve AP [28].

- Mean average precision(mAP): An extension version of average precision. Metric mAP offers the precision for the entire model. This metric measures the average precision of the model across all object classes, taking into account both the precision and recall of the detections. [28].

2.5 Additional Nomenclature

This section discusses some of the additional nomenclature that are relevant for subsequent chapters.

2.5.1 Anatomy of Object Detectors

Object detector architectures are ”divided into three stages: the backbone, the neck, and the head” [29]. We input the image into the backbone and get the output after processing it through the neck and head. This process is shown in Figure 2.3.

- Backbone: The backbone of a deep learning architecture is a feature extractor that is typically a classification model. Some common examples include VGG16, ResNets, MobileNet, and ShuffleNet [30].
- Neck: The neck collects features of different sizes from the backbone and fuses them. The neck's primary goal is to act as a feature aggregator [30] that the head can process.
- Head: The role of the head is to perform object detection and predict the label along with the bounding box. Depending on the architecture, the head can be one-stage or two-staged [30].

2.5.2 Non-Maximum Suppression (NMS)

The object detector may produce duplicate or overlapping bounding boxes, so non-maximum suppression (NMS) is a strategy to eliminate these. It operates by comparing with bounding boxes' confidence scores and choosing the one with the biggest score while eliminating any that overlap.

The goal of NMS is to reduce the number of false positive detections (see Figure 2.4) and improve the overall accuracy of the object detector. It is typically applied after the object detector has generated a group of bounding boxes. It is often used with other techniques, such as thresholding, to filter the results further.

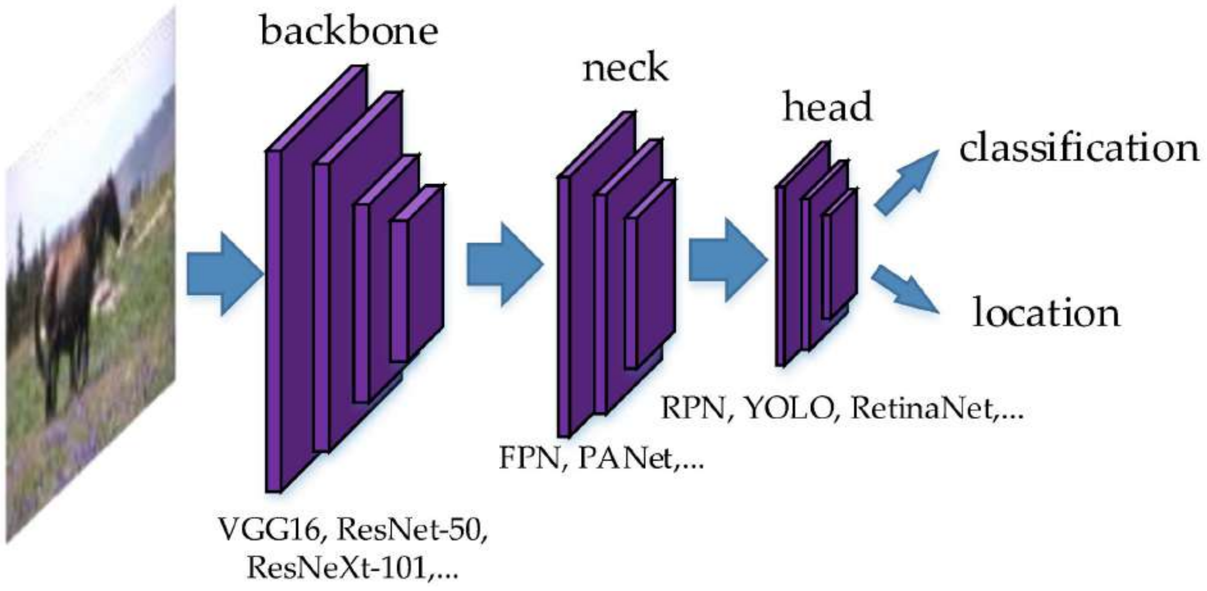


Figure 2.3: Anatomy of Object Detectors.
Note: Image taken from Dengshan et al. [31]

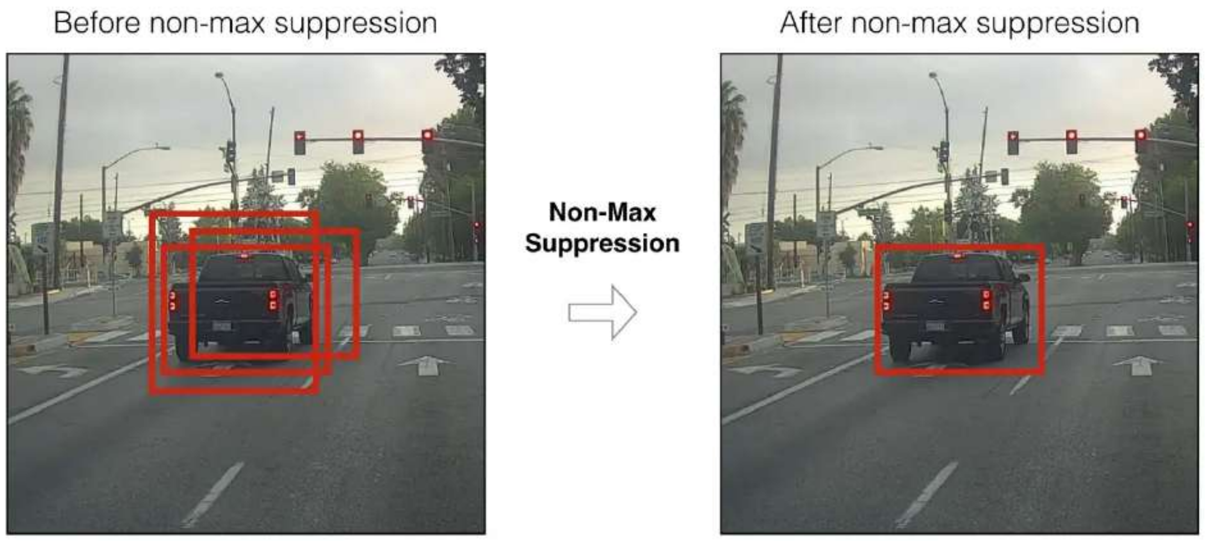


Figure 2.4: Image before and after NMS.
Note: Image taken from Sambasivarao. K [32]

2.5.3 End-to-End Learning

End-to-end object detection refers to using of a ML model that is trained to perform both object detection and classification tasks, without any non-differentiable components (such as non-maximum suppression, or NMS) in the pipeline [33]. The input to the model is an image, and the output is direct predictions of the object categories or background, as well as box regression [33]. The model is "trained in an end-to-end manner using back-propagation" [33]. Figure 2.5 shows this difference between the two approaches.

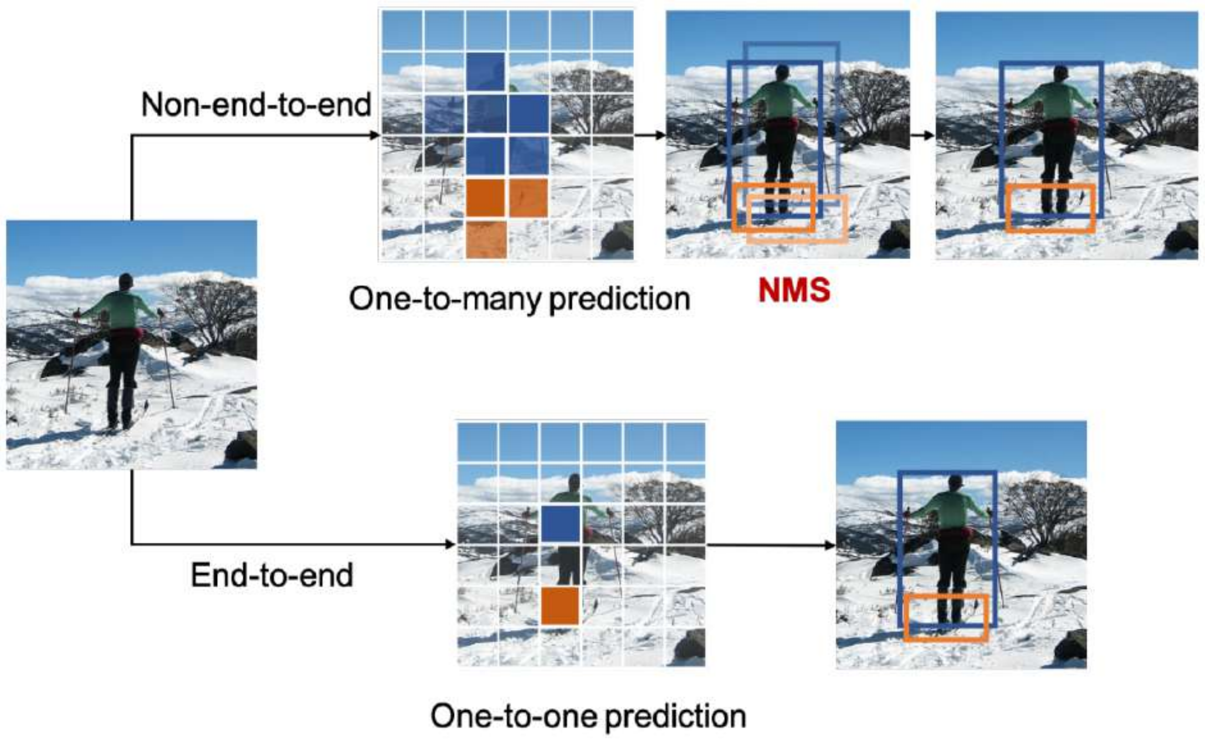


Figure 2.5: End-to-end detection vs Non-end-to-end detection.
Note: Image taken from S. Peize et.al [33]

3 Related Work

In this chapter, we will review the previous work. The chapter will be divided into three sections, each focusing on a specific aspect of the field. The first section will review works based on cycle time, and real-time considerations, the second will delve into object detection in retail, and the third will explore robots deployed in retail.

3.1 Works Based on Cycle Time, and Real-Time Considerations

The robot perception cycle time, or the amount of time it takes for a robot to observe its surroundings and act accordingly, is crucial in supermarkets since it directly impacts the robot's performance. Real-time detection is also vital in supermarkets because it allows the robot to respond to changes in its environment promptly. In this section, we review some of the works that consider cycle time and real-time processing as their primary goals on a robot or a standalone detection system.

3.1.1 Literature Review

Some of the early methods of robot detection systems relied on server-side inference. The work by N. Yamini et al. [34] proposes an approach for real-time object recognition and tracking using robot computation offloading. This process involves moving some of the calculations to servers to reduce the time needed on the robots, but this also requires additional communication time, which is affected by the data size and available wireless bandwidth [34]. By determining whether to perform jobs on robots or servers in order to fulfill real-time limitations, the results of these studies show to reduce the overall execution time. In contrast to the latter method the work by Dirk Holz et al. [35] uses a high computing power-based machine on the robot for the depalletizing task. In this study, a method for depalletizing items is presented, along with a pipeline for their detection, identification, and verification that they are the right things [35]. The system has been proven for depalletizing automobiles and other prefabricated components with excellent reliability and efficiency [35]. It is designed to work at high frame rates, resulting in reduced cycle times. The method employs multi-resolution surfel models to achieve high resilience and adaptability to various objects and lighting conditions [35].

After the rise of GPU computing and deep learning, traditional handcrafted computer vision methods have become a thing of the past. This is because neural architectures are more efficient in learning representations than manual handcrafted algorithms but come at the cost of the inference time. Canziani et al. [36]; investigated the effect of the parameters and number of operations on the inference time and power consumption. By considering the frequently constrained resources in actual deployments, the authors offer insights into design decisions that might result in effective neural networks for use in practical applications [36]. The research demonstrates the hyperbolic relationship [36] between accuracy and inference time, where a modest improvement in accuracy necessitates a significant increase in processing time [36]. The study also illustrates how a network model's number of operations may be used to predict inference time.

J. Huang et al. [37] explore how contemporary neural object detection algorithms might "trade accuracy for speed and memory use" [37]. The authors "examine the trade-off between speed and accuracy by employing several feature extractors and adjusting other factors like image size" [37]. They also show a unified implementation of numerous meta-detection algorithms. The resultant spectrum contains a detector that can be used on a mobile device and is enhanced for speed and memory [37]. Similar work by Huizi Mao et al. [38] investigates the right metric for video object detection, which is close to real-time object detection. The authors discuss the evaluation of object detection in videos using average precision (AP). The authors assert that AP falls short of capturing the temporal character of video object identification and provide the average delay (AD) as a new metric for determining and evaluating detection latency [38]. The paper demonstrates that many methods significantly increase detection delay, but the AP value is unaffected [38]. This work contradicts the work done by Canziani et al. [36], as they demonstrate the relation between delay and average precision.

In the future, we anticipate that more manufacturing will incorporate human-robot interaction and teleoperated robots. While taking into account technological advancements, it also protects workplaces for people. So it is essential to consider the effect of the cycle time under such collaborative scenarios. Some recent works [39], [40], [41], [42] demonstrate how cycle time is affected in such scenarios. Sarah L el al. [39] compares workers' perspectives on human-robot interaction with and without cycle time (i.e., workers working with a deadline and without) in three scenarios: cooperation, coexistence, and collaboration. The study indicated that the group with cycle time felt more time pressure, dissatisfaction, and perceived inferior performance than the group without cycle time through a two-group experiment including pre-and post-testing [39]. The findings also showed that a longer cycle duration resulted in the sense of being a less ideal time [39]. Using three different handover prediction methods, Tang et al. [40]investigated the cycle time, waiting time, and operators' arbitrary choices for a human-robot interactive assembly activity. For a task to be completed smoothly, the robot must be able to predict how long the human worker will need to do it and change its own cycle time accordingly.

3.1.2 Summary

The Table 3.1 summarizes the above-discussed literature by highlighting the main contributions and deficits. The table also includes some other works that are relevant under real-time scenarios.

3.2 Works Based on Object Detection in Retail Settings

Object detection is an vital task(considering the robots) in retail because it enables robots to identify and locate objects in their environment. This capability is essential for various applications, such as inventory management, shelf restocking, and customer assistance. Object detection can help retailers improve efficiency, reduce costs, and enhance the customer experience. It can also enable the use of robotics in new areas, such as personalized recommendations and interactive displays. By using object detection, retailers can harness the power of artificial intelligence and robotics to transform their operations and better serve their customers. This section reviews some of the works on object detection in retail stores.

References	Contributions	Deficits
Yamini N et al. [34]	Propose a computational offloading mechanism using server-side load balancing to reduce cycle time.	Assumes only one object in the frame during inference.The computational requirements are pre-estimated. Not suitable for dynamic environments.
Dirk Holz et al. [35]	Demonstrate that visual object geometry affects the cycle time during grasping.Propose a detection verification mechanism during grasping.	Considers fixed camera workspace during object detection.Not scalable for supermarket-based settings due to high on-board computation.
Canziani et al. [36]	Show how a network model's number of operations may be used to predict inference time."Identify relationship between accuracy and inference time" [14].	Only considers inference on high-end GPU and ignores total cycle time.
J. Huang et al. [37]	Illustrate the speed vs. accuracy trade-off curve for different detection systems	During the experiments, embedded platforms are not considered.
H .Mao et al. [38]	Propose a new metric, average delay (AD), to measure and compare detection delay.	The work is limited to video-based object detection.
Sarah L et al. [39]	Study the effect of cooperation, coexistence, and collaboration in robots and humans.	Considers only onsite human-robot scenarios and remote collaborative scenarios not explored.
Tang et al. [40]	Demonstrate the effect of waiting time in human-robot object handover.	Focus only on the human-to-robot waiting time but not vice versa.
Berglund et al. [41]	Demonstrate how knowledge-based robotic systems affect the total cycle time.	Considers only human-in-loop scenarios.
M. Zhang et al. [42]	Demonstrate human fatigue recovery while maintaining the cycle time.	Focus only on the case study than real-world implementation.
A. Kumar et al. [43]	Demonstrates that distributed inference can reduce the cycle time.	Architecture tailored only to one robot, missing generalization.

Table 3.1: Summary of related works based on cycle time, and real-time consideration.

3.2.1 Literature Review

The work by M. Merler et al. [44] in early 2007 demonstrated product recognition in real-world settings by training on in vitro datasets. The framework points to the necessity for more detailed and complex detection/recognition algorithms to address the stated issue. For this purpose, the authors develop the GroZi-120 [44] dataset with 120 product categories. The authors show how to use typical image search criteria on photos taken with a camera-equipped smartphone to identify related photos on the Internet or in other generic databases [44]. The study also covers the algorithms' achievements and shortcomings in classification performance and localization accuracy. The work by C. Melek et al. [45] surveys the initial attempts for product recognition in supermarkets. The survey summarizes the works based on Remote Sensing Technology and traditional computer vision-based detection systems [45]. The authors identify that compared to the technique of object recognition after object detection, the approach of object identification directly without object detection is more straightforward and quicker [45].

The work discussed before heavily relay on traditional computer vision-based object recognition. However, after the introduction of deep learning, many planogram stock checking-based recognition has spiked interest in the researchers. B. Santra et al. [5] presents a new taxonomy for the domain of machine vision-based retail product recognition. The paper thoroughly examines the features employed and the results of various approaches, as well as the challenges and inherent difficulties related to this problem from prior works [5]. The study also offers information on publicly accessible datasets and suggests future research topics in relevant areas [5]. The objective of autonomous product identification in retail outlets is to enhance customer satisfaction and assist businesses financially. Nevertheless, this task is more challenging for machine vision systems than other object recognition tasks [5]. The work by E. Goldman et al. [46] provides a deep learning-based approach for effective item detection in highly populated artificial environments with many close objects that are frequently indistinguishable. The proposed method incorporates a layer that computes the correlation index as a measurement of the detection capability and a brand-new expectation-maximization merging unit [46] that employs these quality scores to clear up overlap problems. The authors also contribute the

SKU110k dataset [46], considered a standard benchmarking dataset in product detection.

Identifying products effectively from image streams is challenging for computer vision tasks, but it has garnered interest because of the possible applications like online shopping. In order to perform tasks like stock monitoring, planogram regulation, automated checkout, and help those with various disabilities, deep learning methods for retail product recognition have been surveyed by Y. Wei et al. [4]. The authors demonstrate the advantages of automatic product recognition over the manual operation [4]. The review discusses real challenges, prospective deep-learning methods for this task, and information on free-to-access datasets. The work also considers the monetary and societal implications related to the product recognition challenge [4]. The paper's conclusion is a summary of the state of the field's research and recommendations for future research in related areas.

In retail applications where bounding boxes are difficult due to solid occlusions across clusters of samples of the same categories, this study by Y. Cai et al. [47] addresses the challenge of "simultaneous object localization and counting" [47]. Over 50,000 pictures and 1.9 million object occurrences in 140 categories comprise the authors' extensive object localization and counting dataset for retail stores [47]. The dataset is complemented by a novel assessment process and a training/testing split. The authors provide a "cascaded localization and counting network" [47] as a solid starting point for this challenge.

Understanding how various detectors perform can help design and implement automated systems in supermarkets, potentially resulting in increased accuracy and efficiency.In this study, E. Arani et al. [6] compare a variety of cutting-edge real-time object detectors using a uniform experimental setup and a variety of datasets. The authors emphasized comparing hardware resources, calibration, accuracy, and speed [6]. The study also investigates the effects of different hyperparameters and model backbones. The reliability of the detectors against actual adversarial attacks is also evaluated. The recent works after 2020 address some fundamental problems in retail object detection—(i) the work by F. Chen et al. [48] address the problem of detecting different sub-classes in a single class by incorporating optical character recognition (OCR) after the detection pipeline. (ii) M. Saqlain et al. [49] study the detection performance using modern detectors for stock-keeping tasks. (iii) Finally, the work by J. Jeon et al. [50] Proposes the new "view-aware."

object classification in scenarios with interclass similarity. The authors utilized multiple camera-based image-capturing systems during the training.

3.2.2 Summary

The Table 3.2 summarizes the literature discussed in the last section by highlighting the main contributions and deficits. The Table also includes some survey papers that are relevant under retail scenarios.

3.3 Works Based on Robot Deployment in Retail

There are several potential benefits to deploying robots in supermarkets, and we have motivated this in our introduction. Robots may be designed to be extremely exact and precise, lowering the possibility of mistakes in jobs like checkout and inventory management. Automating tasks with robots can reduce labor costs and increase profitability. This section reviews some of the works based on robot deployment in retail.

3.3.1 Literature Review

Some of the initial methods explore inventory management in supermarkets using the robots. Gopichand et al. [51] use a telepresence robot to patrol a retail store and capture images of the shelves. These images are then processed to identify stock outages and misplaced products, and an alert system is used to notify store associates of any issues [51]. The solution uses a hybrid methodology of computer vision and deep learning algorithms to automate these processes and reduce the manual effort required by store associates. In order to study the uncertainty in collaborative robots. F. Coelho et al. [52] look into the deployment of robots in the production of supermarkets, which supply flexible assembly lines in the manufacturing industry. A simulation tool was used to assess the effectiveness of deploying robots for order picking in a logistics supermarket. The model's findings demonstrated that adding humans to a system's loop improves performance [52]. However, when uncertainty is considered, collaborative robots are more adaptable and result in less variability in performance [52]. According to this study [52], using robots to build supermarkets may produce financial savings.

References	Contributions	Deficits
M. Merler et al. [44]	Demonstrate how in vitro training can also help in real-world product recognition. The authors also released the GroZi-120 dataset, now used as a benckmark.	Only rely on traditional computer vision-based object recognition.
C. Melek et al. [45]	Identify the approach of object identification directly without object detection is more straightforward and quicker.	The survey is limited to remote sensing and computer vision-based methods.
B Santra et al. [5]	Presents a comprehensive taxonomy for retail product recognition.	Focus only on early attempts of deep learning.
E Goldman et al. [46]	DL-based approach for effective item detection in densely Packed scenarios. Released the standard SKU110k dataset.	The dataset contains only one class. The bounding box overlap problem still needs to be fully addressed.
Y. Wei et al. [4]	Demonstrate the advantages of automatic product recognition over manual operation. Provide a comprehensive survey of problems.	Do not consider the challenges faced in human-robot collaborative tasks.
Y. Cai et al. [47]	Address the problem of duplicate bounding boxes by effective localization and counting techniques.	The solution needs more data with different variations per class.
E. Arani et al. [6]	Provide a comprehensive study of the state-of-the-art real-time object detection methods.	Do not consider the case study of object detection in supermarkets.
F. Chen et al. [48]	Address the problem of detecting different sub-classes in a single class.	The solution also needs character recognition people, which can affect the cycle time.
M. Saqlain et al. [49]	Study the detection performance using modern detectors for stock-keeping tasks.	The solution is focused on a stock-checking application.
J. Jeon et al. [50]	Proposes the new “view-aware.” object classification, in scenarios with interclass similarity.	The solution uses many cameras, which is not scalable in retail settings.

Table 3.2: Summary of related works based on object detection in retail settings.

Some of the other methods explore assisting customers in supermarkets using a robot. R. Alves et al. [53] came up with an autonomous shopping robot to assist customers during shopping. The work focus on helping people with disabilities to navigate around the supermarket. The authors use genetic algorithms for path scheduling [53]. The result shows that the genetic algorithms perform well for distances less than 65 meters within the facility.

Y. Aquilina et al. [54] developed a SCARA-based robot for an automatic supermarket checkout system. In this work, the authors found that the camera's resolution can affect the performance of the checkout system [54]. Another critical criterion is the field of view [54] during the checkout. By conducting pick-and-place experiments, the authors found that the average cycle time for the entire process took approximately 20 seconds for one object [54]. A similar work by W. Hu et al. [55] using a six-degree-of-freedom robot demonstrates that it would take approximately 24 seconds per object at a success rate of 98% for the entire operation [55]. The authors also consider the camera mounting position essential, which can affect the field of view. D. Ryumin developed et al. [56] a multi-modal user interface for a robot-assisted shopping cart. According to their experiments, the total cycle time is about 50 seconds, where selecting and scanning an item takes 30 seconds and the transfer task 20 seconds [56].

The works by R. Caccavale et al. [57] and P. Arpenti et al. [58] focus on robot depalletization in supermarkets. In the first work, the authors devised a "depalletizing system that can schedule, monitor, and adapt the depalletizing process considering both online perceptual information provided by non-invasive sensors." [57]. In the second work, the authors developed a new RGBD- recognition system for depalletizing tasks based on apriori knowledge [58]. The latter method required less training dataset due to the knowledge base. The work by X. Zhang et al. [59] demonstrates that grayscale images would be sufficient for object tracking in a supermarket scenario. Finally, the work by M. Costanzo et al. [60] provides a comprehensive summary problem in the supermarket pick and place task.

3.3.2 Summary

The Table 3.3 summarizes the literature discussed in the last section by highlighting the main contributions and deficits.

References	Contributions	Deficits
Gopichand et al. [51]	Studies the uncertainty in collaborative robots deployed in supermarket	Focused only on stock-checking robots and does not consider a active sensing scenario.
F. Coelho et al. [52]	Demonstrates that adding humans to a system's loop improves performance.	Demonstration on only simulation based scenarios
R. Alves et al. [53]	Develops autonomous shopping robots using genetic algorithms.	Focus only on genetic algorithms for path scheduling.
Y. Aquilina et al. [54]	Achieve a total cycle time of 20 seconds using a SCARA robot.	Densely packed environments during experiments were not considered.
W. Hu et al. [55]	Achieve a total cycle time of 24 seconds per object at a success rate of 98% using a 6 DoF robot.	The primary focus is only on the grasp configurations.
D. Ryumin et al. [56]	Demonstrates total cycle time takes about 50 seconds using a multimodal user interface.	Considers only two classes of objects and more focus on user interface design.
R. Caccavale et al. [57]	Develops a depalletizing process considering active perception.	The primary focus is only on logistical tasks rather than detecting grocery products.
P. Arpenti et al. [58]	Demonstrates that knowledge-base systems need fewer data during training.	The primary focus is only on large-scale grocery products.
X. Zhang et al. [59]	Demonstrates that grayscale images would be sufficient for object tracking in a supermarket scenario.	Demonstrates only object-tracking scenarios in supermarkets.
M. Costanzo et al. [60]	Provides a comprehensive summary problem in the supermarket pick and place task.	Problems related to perception and cycle time are not covered.

Table 3.3: Summary of related works based on robot deployment in retail.

4

Problem Formulation

This chapter aims to provide the formulation of the project. To do so, we will summarize the limitations of previous works discussed in the chapter 3. Following this, we will present our problem statement, outlining the specific goals and objectives that our approach aims to achieve.

4.1 Limitations of Previous Works

This section summarizes the limitations of the previous works discussed in chapter 3. We consider the following limitations in our project:

L1 The perception cycle time for a robot still needs to be fully investigated for an online shopping scenario.

L2 The current state-of-the-art object detection solutions are heavily GPU-dependent. This dependency is not feasible for real-time deployment for a cost-effective robotic solution.

L3 The real-time object detection algorithms must be better tested under various conditions (e.g., lighting condition, viewing angle) and parameters (e.g., image size, model size) in a supermarket scenario. Moreover, some works do not consider densely packed scenes.

L4 Most of the works related to object detection in the supermarket only compare the performance of one particular hardware. It is hard to decide how the model performs on embedded hardware based on results from powerful machine.

L5 In the context of object detection in supermarkets, lightweight deep learning models (e.g., ShuffleNet) and transformers-based models must be investigated.

4.2 Problem Statement

The project evaluates and compares a robot's real-time perception capabilities in a supermarket scenario. As shown in the Figure 1.1 in chapter 1 we assume that, the current project would take place in 24/7 supermarket settings and the robot will be responsible for shopping based on remote user inputs. Although the robot might have several tasks like navigation, perception, and manipulation, this project focuses on analyzing the perception cycle time. Based on the cycle time analysis results, we explore if shared task execution between humans and robots is feasible. Moreover, the project lies at the intersection of investigating the cycle time, using real-time detectors, and considering a robot deployed in a supermarket scenario.

By considering different combination of limitations we formulate our problem(P) as below(the considered limitation are given in the end):

P1 Select four different real-time object detector(refer chapter 5) , compare the cycle time and report which detector performs better. [**L1, L5**]

P2 Study the variations of cycle time on different hardware(refer chapter 6)using the selected detectors [**L1, L2, L4**]

P3 Investigate the effect of cycle time over different conditions(lighting condition, viewing angle, distance, blur) and parameters (image size, model size, number of classes) using the selected detectors [**L1, L3**]

P4 Based on the results of above problem statements analyse if shared task execution between human and robot is feasible? [**L1**]

5 Methodology

This chapter provides an overview of the datasets, object detectors, and code profilers used in this project. We begin by surveying a variety of supermarket datasets and discussing the criteria for selection. We also explain the creation and aspects of our custom dataset. Next, we describe the different object detectors employed. Finally, we discuss the various code profilers and describe our profiling strategy.

5.1 Datasets

We present a brief overview of the available supermarket datasets in this section that are relevant to this project (Refer to section A.4 to see samples of the datasets). Our search criteria are as follows:

- All the images and annotations should be openly available.
- The images in the dataset must include only real-world supermarket-based products.
- Datasets with multiple classes are encouraged.
- More preference is given to standard benchmarking datasets with densely packed grocery objects.

5.1.1 Freiburg Groceries Dataset

The Freiburg Groceries Dataset [61](2016) is a collection of images and annotated labels with a small set of grocery products. The Freiburg Groceries Dataset is mainly used for tasks such as product classification and object detection. It was created by researchers at the University of Freiburg in Germany and is intended for use in research on computer vision and retail object recognition algorithms. The authors "present a dataset consisting of 4947 images covering 25 different classes of groceries, with at least 97 images per class" [61]. They also provide a separate testing dataset with 74 images with cluttered scenes. The data is collected using four mobile phone cameras at various locations like supermarkets and apartments in Freiburg, Germany [61].

5.1.2 RPC Dataset

The Retail Product Checkout (RPC) [62](2019) dataset is one of the largest datasets by both product image number and product classes. The dataset has pictures of a single product taken in a lab setting and images of many products captured in supermarket settings [62]. The authors provide 83,739 images, where 30,000 images are checkout based and 53,739 single-product based. It provides different types of labels for the images, which means that it classifies and labels the objects in the pictures with varying degrees of detail. This allows for more granular analysis and understanding of the objects in the image and can be helpful in different applications and use cases [62].

5.1.3 SKU-110K Dataset

The SKU110k dataset [46](2019) is one of supermarket shelves' most enormous and standard benchmark data. The dataset was released as part of the Retail Detection Challenge, in Retail Vision Workshop 2020 [63]. The dataset consists of 8,233 training images, 588 validation images, and 2,941 test images [46]. The dataset is collected from various supermarkets from different parts of the world, namely Europe, Asia, and The United States, corresponding to a more significant set of product catalogs in different forms and shapes [46]. The images in the dataset consist of densely packed configurations of supermarket shelves. Due to this reason, the dataset has only one object class. The task was to detect only the correct bounding box. SKU-110k dataset is frequently used as benchmark for many retail related applications and many new datasets are also built re-using this datasets.

5.1.4 RP2K Dataset

The RP2K dataset [64](2021) is a collection of retail product images that has been gathered from real retail stores. It contains 350,000 images of over 2,000 different products, which have been manually captured under natural lighting conditions [64]. The dataset includes two components: shelf images, which are labeled with information such as the store ID and a list of labels for the objects, and individual object images, which represent products from the 2388 different SKUs in the dataset [64]. The shelf images were collected from 500 stores in 10 cities and have an average resolution of 3024 by 4032 pixels. The dataset includes a total of 10,385 shelf images and 384,311 individual object images, and has been split into a training set and a testing set in a ratio of 0.9 to 0.1 [64]. The RP2K dataset is designed to be utilised for retail item recognition research. This field has a variety of applications, such as stock checking [64].

5.1.5 Locount Dataset

The Locount dataset [47](2020) is a collection of "50,394 JPEG images" [47] with a resolution of 1920 x 1080 pixels. It was collected from twenty eight separate shops and apartments using a variety of lighting setups and camera positions, and contains 140 standard products divided into 9 main sub classes [47]. "The dataset includes more than 1,905,317 annotated object instances, with 34,022 images and 1,437,166 instances in the training subset, and 16,372 images and 468,151 instances in the testing subset" [47]. These two groups of pictures were taken in various places, yet they share a lot of the same settings and characteristics. Drinks, cooking utensils, food, baby products, apparel, sports equipment, daily chemicals, stationery and electrical items are just a few of the nine primary subgroups that make up the hierarchical structure that divides the object categories in the Locount dataset into such categories [47].

5.1.6 Small Groceries Dataset(Ours)

This project aims to experiment with real supermarket-like scenarios using a robot and other hardware configurations. For this reason, we surveyed the existing datasets to create a mock supermarket scenario, but procuring the exact objects (i.e., obejcts in other datasets) for the task was very challenging. In order to simplify our process, we came up with our custom dataset. This provided us the opportunity to decide how many classes to include in our dataset and to arrange the items in accordance with supermarket's standard planographic compliance.

The Small Groceries dataset consists of 317 images, 25 classes, and 6,361 annotations in total. All the images and annotations are published as open-sourced data by the name "Small Groceries" dataset in RoboFlow repository[2].The images were collected using three different cameras: (i) Intel RealSense D435 RGBD camera, (ii) Smartphone, and (iii) Laptop's integrated webcam, and saved in JPG format. We create two different sets as shown in Figure A.4. The first set consists of 150 images where the objects are arranged as per supermarket shelves. The second set consists of 167 images with single objects. The aspect ratio is also distributed as two categories, with 44 tall images and 273 wide images. The median image ratio in the dataset is 1920 by 1080. The Figure 5.1 show the class labels along with the number of occurrences in the total dataset and Figure 5.2 shows the histogram of count of all objects in the dataset. Some of the classes like pasta, cake_box, soap are underrepresented and this is also evident in the annotation heat map show in Figure 5.3. All the images were manually annotated using the LabelImg tool [65]. A sample of annotation instance is shown in appendix Figure A.15. The annotations are saved in YOLO format and using the Roboflow interface it can be exported to various other formats. Some of the example images in our dataset are shown in Figure A.4.

5.1.7 Datasets Summary and Selection

All the relevant supermarket datasets, along with our Small Groceries dataset, are summarized in the table 5.1. Finally, we selected SKU110k and Small Groceries datasets in our experiments. SKU110k was selected because it serves as the standard benchmark and contains densely packed objects with one object class, which would be sufficient for our experiments.

5.2 Object Detectors

In this section, we will discuss the various object detectors we considered for our project. We will explain the reasons behind our choices and provide a brief overview of how each

[2]Dataset available at: https://universe.roboflow.com/hamsa-datta/small_groceries

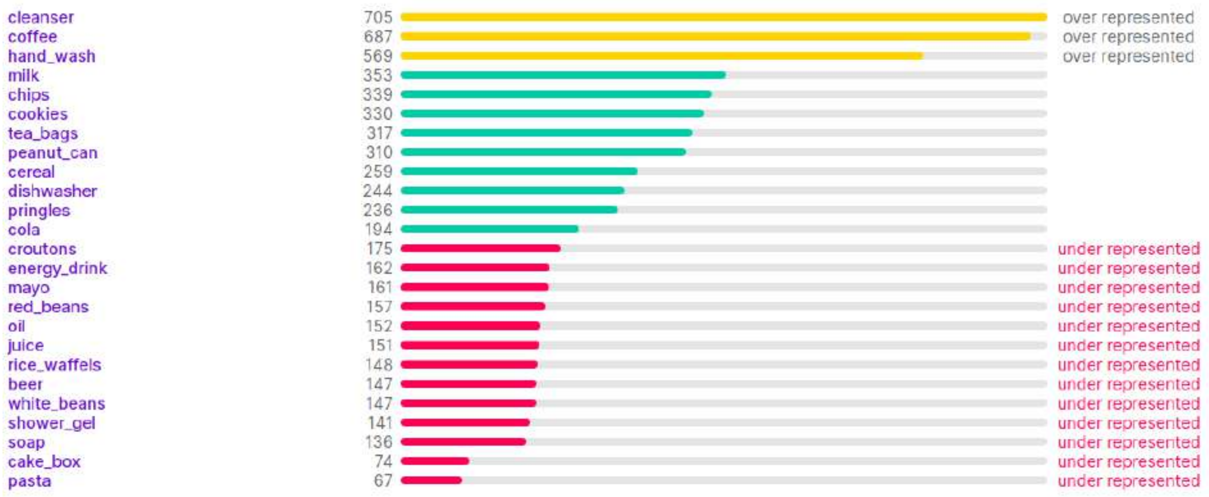


Figure 5.1: Small Groceries dataset class balance diagram

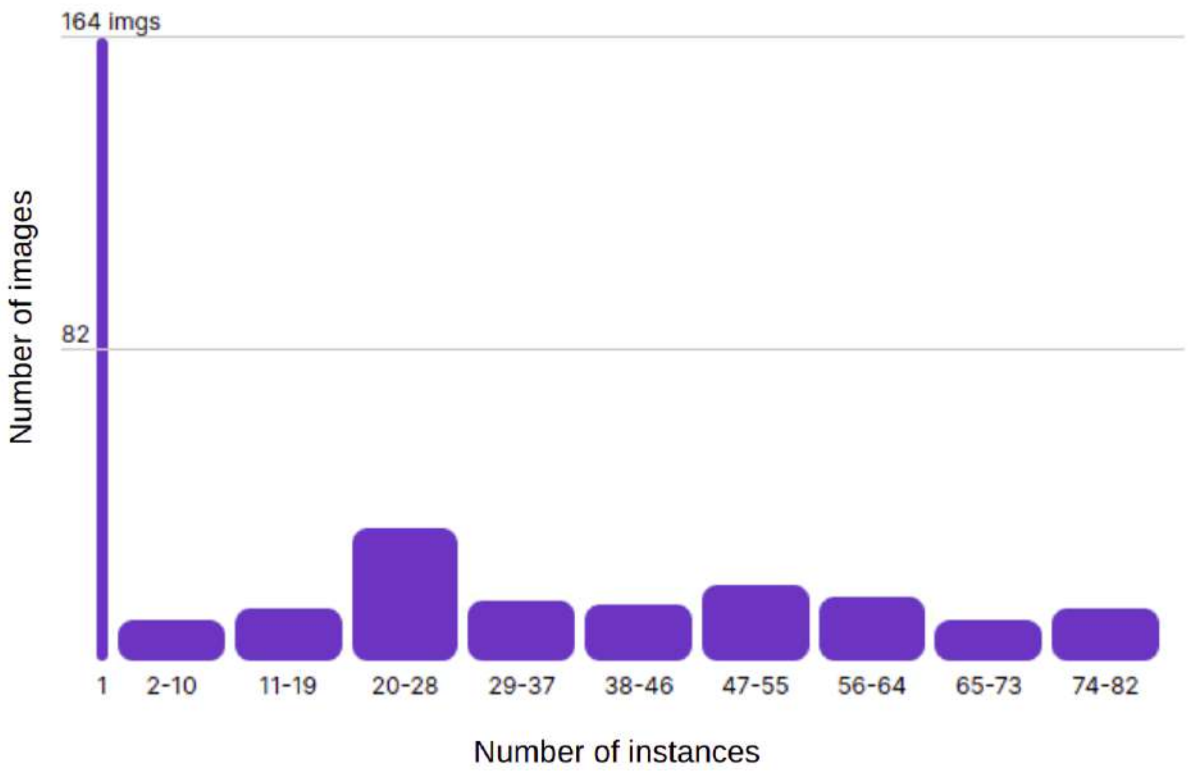
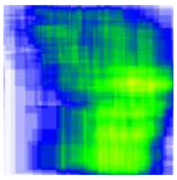
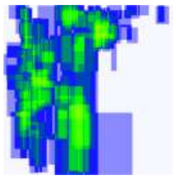
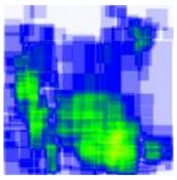
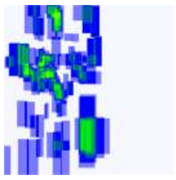


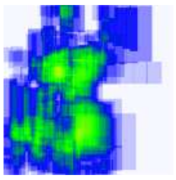
Figure 5.2: Small Groceries dataset objects histogram. The x-axis shows the number of instances, and the y-axis shows the number of images.

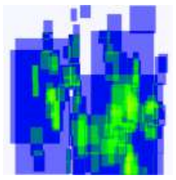

(1) chips

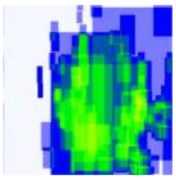

(2) oil

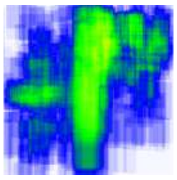

(3) cookies

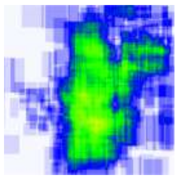

(4) energy_drink

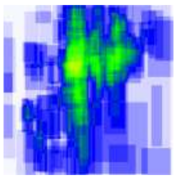

(5) milk

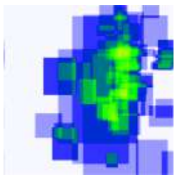

(6) white_beans

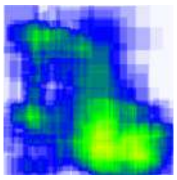

(7) red_beans

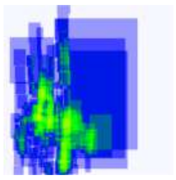

(8) cleanser

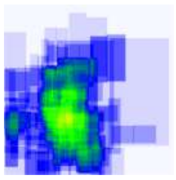

(9) hand_wash

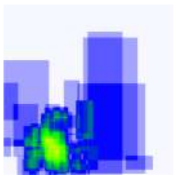

(10) dishwasher

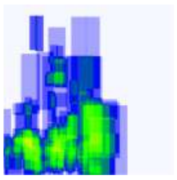

(11) cake_box

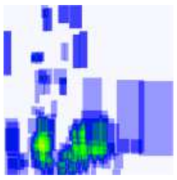

(12) cereal

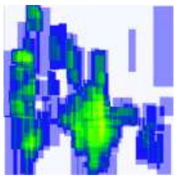

(13 peanut_can

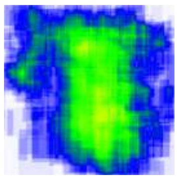

(14) croutons

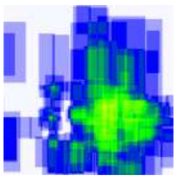

(15) pasta

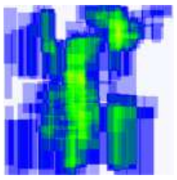

(16) rice_waffels

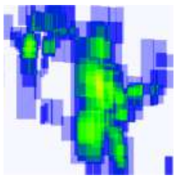

(17) beer

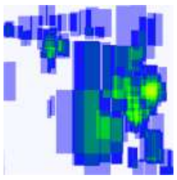

(18) cola

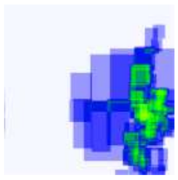

(19) coffee

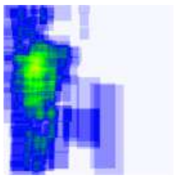

(20) juice


(21) pringles


(22) mayo


(23) shower_gel


(24) soap


(25) tea_bags

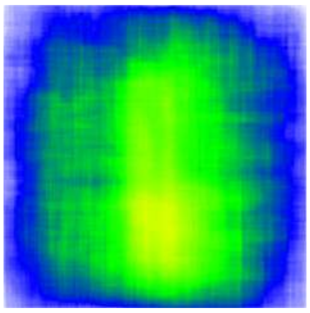

(26) All

Figure 5.3: Annotation heat map for all classes. Every annotation's placement in the dataset is shown so we can see where they are most frequently placed.

Datasets	**Images**	**Category**	**Instance**	**Resolution**	**Task**	**Year**
SOIL-47 [66]	987	47	NA	576 × 720	C	2002
Supermarket [67]	2,633	15	NA	640 × 480	C	2010
D2S [68]	21,000	60	72,447	1920 × 1440	M	2018
Grozi-3.2k [69]	9,030	80	11,585	640 × 450	M	2014
Grocery Shelves [70]	354	10	13,000	-	M	2015
Freiburg Groceries [61]	5,021	25	NA	1920 × 1080	C	2016
Retail-121 [71]	567	122	NA	-	M	2017
GameStop [71]	1,039	3,700	NA	1200 × 900	C	2017
TGFS [72]	38,027	24	38,027	480 × 640	M	2019
SKU110k [46]	11,762	1	1,733,711	1920 × 2560	S	2019
RPC [62]	83,739	200	421,674	1800 × 1800	M	2019
Locount [47]	50,394	140	1,905,317	1920 × 1080	M	2020
RP2K [64]	10,385	14	95,800	3024 × 4032	M	2021
Small Groceries (Ours)	**317**	**25**	**6,361**	**1920x1080**	**M**	**2022**

Table 5.1: Summary of datasets related to supermarkets."C: Classification,S:Single-class object detection , M:Multi-class object detection". Note: The table adapted from Y. Cai et al. [47]

detector works. In our project we select YOLOv5 [73], YOLOv7 [74], NanoDet-Plus [75], and detection transformer (DETR) [23] as our real-time object detector. The selection criteria for each detector is given below:

- We choose YOLOv5 and YOLOv7 object detectors because they provide the current state-of-the-art performance concerning speed and accuracy. Both the detectors provide excellent community support with respect to the issues handled in the code repository and provide comprehensive documentation along with many easy to use guides.

- As discussed in chapter 4, we aim to consider lightweight object detection models in our project. Based on the literature review discussed in chapter 3, one of the survey papers [6] pointed out that NanoDet detector showed excellent real-time performance in their study. In this project we consider NanoDet-Plus which is expected to perform at par with the current state-of-the-art object detectors in terms of speed.

- Object detection using transformers is a new paradigm in field of computer vision [6]. In this project we consider DETR as one of our object detectors for comparison. The detector provides good community support and documentation.

5.2.1 YOLOv5

YOLOv5 is a CNN-based object detector from the YOLO family. The architecture of the YOLOV5 is based on the combination of YOLOv3 and YOLOv4 architectures [76]. The head detector is reused from the YOLOv3 and a similar backbone from YOLOv4. The four important YOLOv5 models are small, medium, big, and extra large, each of which delivers progressively greater accuracy rates [76].

The architecture [73] consists of the below:

- Backbone: CSP-Darknet53
- Neck: Spatial Pyramid Pooling - Fast (SPPF)
- Head: YOLOv3 Head

key contributions:

- The focus layer was added in YOLOv5, thus bringing down the number of tuning parameters and GPU memory usage and increasing the detection speed. The resulting change did not affect the accuracy overall [77].
- The authors used Sigmoid Linear Unit SiLU compared to the sigmoid for the nano model, thus reducing the model's size [77].
- The entire YOLOv5 model is based on Python framework that increases the usability for the users [77].

5.2.2 YOLOv7

Based on an efficient layer aggregation network (ELAN), the YOLOv7 architecture was created to perform at real-time speeds [78]. For deeper networks to converge and train efficiently, ELAN considers constructing an efficient network by managing the smallest and biggest gradient paths [78]. YOLOv7 uses the extended-ELAN (E-ELAN). The connections and attribute values of the computing block are increased via group convolution in E-ELAN. “All of the computational blocks in a computing layer have the same channel multiplier and group parameter applied to them" [78]. The new architecture helps the neural network to learn better and faster [79].

Key contribution:

- Model scaling is a technique used to modify a machine learning model to match better the requirements or capabilities of a particular device or software platform. It involves adjusting variables like image resolution, the number of channels or stacks in the model, and the number of feature pyramids used. YOLOv7 scales the model's breadth and depth together to improve optimization [79].
- YOLOv7 has developed "Bag of Freebies" (BoF) [74] techniques to enhance model performance without raising training costs. Re-parameterization is one of these techniques; it entails assembling various models or modules to enhance inference output [79].
- The author's present label assignment [74], a system that allocates soft labels after considering the ground truth and network prediction outcomes. It is crucial to remember that the label assignment creates coarse and soft labels rather than rigid ones [79].

5.2.3 NanoDet-Plus

NanoDet-Plus is an improvement of NanoDet architecture. NanoDet-Plus is based on Fully Convolutional One-Stage Object Detection (FCOS) [80]. Based on object attributes, the detector automatically chooses positive and negative training data for the label-matching technique updated in this new approach [80]. Based on the mean and variance of IOU, the label-matching algorithm will automatically choose matching instances for each layer of the feature space. The latter method also helps to maintain the lightweight model more straightforward [81].

The Architecture consists of the below:

- Backbone: ShuffleNetV2
- Neck: Ghost-PAN based on YOLOv5
- Head: NanoDet head with depth-wise convolution

Key contributions:

- The authors improved the performance of the neck by introducing the Ghost-PAN [81] feature aggregator. The idea is based on using the depthwise 3x3 convolutions and the 1x1 convolutions before feeding the output to the head. The number of parameter calculations is significantly decreased [81].
- NanoDet-Plus uses AdamW as its optimizer compared to the usual gradient descent with momentum. The model convergence is increased after switching to AdamW due to more minor effects of other hyperparameters [81].
- The authors support some of the well-known deployment methods like OpenVINO. The latter method is used for deploying the model on CPU-based inference acceleration [81].

5.2.4 DETR

DETR stands for detection transformer, an object detection model from Facebook's research group [82]. Unlike the standard CNN-based object detectors, DETR uses a transformer in its detection pipeline. The transformer architecture consists of an encoder and a decoder, and the essential innovation in the transformer architecture is the use of attention mechanisms [83]. Attention allows the model to focus on specific parts of the input rather than using a fixed-length context. This allows the model to handle input sequences of varying lengths and to weigh different parts of the input differently. One vital point to note here is that DETR still uses CNN based backbone for obtaining features in small dimensions [82]. The role of the transformer is to act like the neck in the CNN-based methods. The features from the backbone are passed through the transformer's encoder-decoder blocks.The output from the decoder block is propagated to multiple detections feed-forward channels [23]. The prediction head is responsible for detecting the correct bounding box with the class label. The Figure 5.4 shows the DETR architecture.

Key contributions:

- The authors introduce a new paradigm in object detection using transformers. This work's results are comparable with the R-CNN family of detectors that perform better over accuracy in general [23].

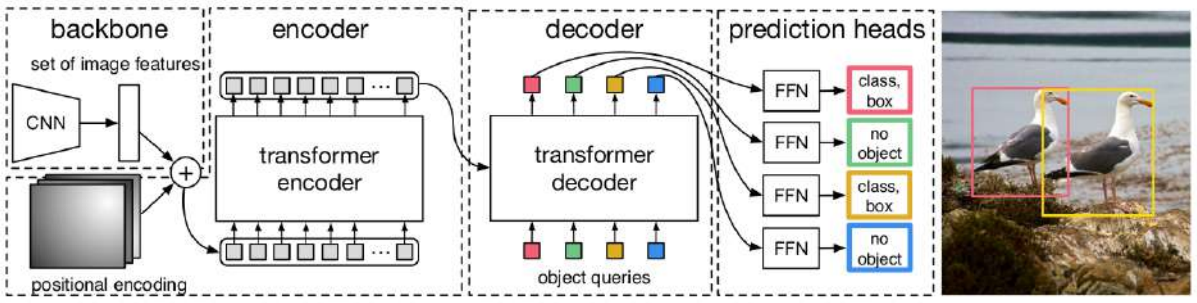


Figure 5.4: The DETR architecture
Note: Image taken from N. Carion [23]

- The DETR object detector is termed end-end because it does not use additional techniques like non-maximum suppression(NMS) after the detection is performed [82]. This reduces the computation needs to an extent.
- The authors developed the new "set prediction loss" [23] function that produces efficient bipartite matching with the class labels and the predictions.

5.3 Code Profilers

Code profiling is the process of analyzing and measuring the performance and efficiency of a computer program or system. It entails executing the programme while gathering information on its execution, including how long it takes to run, how many instructions are executed, and how much memory is used.
In this project, we focus on using time-based code profilers to analyze the performance of our code. There are two types of time-based profilers in Python: deterministic profilers and statistical profilers.

- Deterministic profilers: "Provide precise timings for intervals between function calls, function returns, and exception events" [84].Figure 5.5 shows the process for deterministic profiling.
- Statistical profilers: "Randomly sample the effective instruction pointer and provide relative indications of where time is being spent" [84]. Deterministic profilers typically involve more overhead as the code needs to be instrumented, while statistical profilers typically have lower overhead but provide less detailed information [84].Figure 5.6 shows the process for statistical profiling.

This project aims to analyze the cycle time for different object detectors. In order to find out how long each function call in our code takes, we must profile it using one of the profilers. From the above discussion, it is clear that the deterministic profiler provides precise timings for each function call execution, so we select the deterministic profiler as our measuring tool. Since we use Python as our programming language, we select Python's built-in library, "cProfile" [84], for measuring the cycle time. In the coming sections, we explain our profiling strategy on the object detectors and guide how to read cProfile's output trace(Refer sectionA.2).

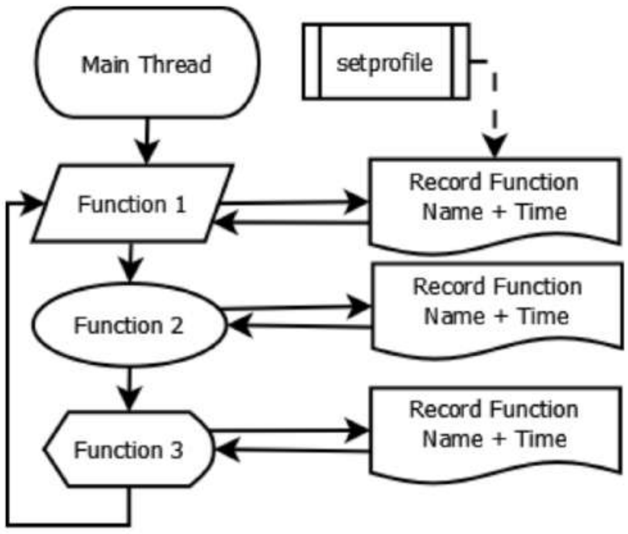

Main Thread
setprofile
Function 1
Function 2
Function 3
Record Function Name + Time
Record Function Name + Time
Record Function Name + Time

Figure 5.5: Deterministic code profiling
Note: The image taken from M. Emin [85]

5.3.1 Profiling Strategy

In this section, we will discuss the code profiling strategy that was adopted to analyse the cycle time in our project.

After going through the code repositories of each object detector, we identified that all four detectors follow a similar meta-structure irrespective of how they are implemented in the code (i.e., using object-oriented style or function-based style). In order to make the comparison uniform across all the detectors, we divide the various object detection processes into five main tasks, as given below.

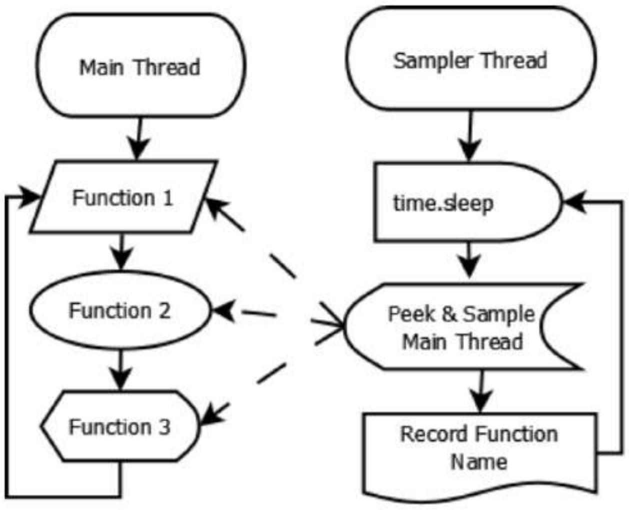


Figure 5.6: Statistical code profiling
Note: The image taken from M. Emin [85]

- Model loading task
- Image pre-processing task
- Inference task
- Image post-processing task
- Miscellaneous operations task

The Python pseudo code 5.1 provides the overview of our profiling strategy. The code execution starts at line 31 and ends at line 38. The profiler instance is created at line 33, and we start and end the profiling by wrapping the entire main function. The main function comprises model loading, inferencing, processing, and other miscellaneous tasks. The essential advantage of using the "cProfile" library is that we can precisely time the function calls, and the trace files are saved in comma separated values (CSV) file type with the function names as per the code (refer section A.2 for more details). So it is easy for us to identify which function call took how much time.

The pseudo-code clearly indicates that the main function's cumulative time gives us the object detector's total cycle time. To identify the time taken for each main task (i.e., model loading, inferencing), we refer to the trace file and manually add the time taken for each sub-task. For example, the pre-processing task consists of reading an image and converting the color format. In order to get the cycle time for the pre-processing task, we add the cumulative time of the function calls, read image, and convert color by looking at the trace files. To report the cycle time of the miscellaneous operations, we subtract model loading, pre-processing, inferencing, and post-processing time from the total cycle time.

```python
import cProfile #import profiling library

def load_model():
    #Build the neural network model, and load the trained model weights.
    return loaded_model

def pre_process_image():
    #Read the current frame from the camera/video.
    #Process the frame: set image size, convert from BGR to RGB.
    return pre_processed_image

def do_inference(model, frame):
    #Pass the frame to the model.
    #get results: predictions, confidence scores.
    return results

def post_process_image(results):
    #Read the results and get the bounding boxes, class labels.
    #Attach the bounding boxes, class labels to the frame.
    #Display the frame if required.
    return post_processed_image

def main():
    my_model=load_model() #Load the model.
    iterator=0 #initialize the iterator variable.
    while(iterator<10): #Run while loop for 10 iterations.
        get_frame=pre_process_image() #Perform pre-processing.
        get_results=do_inference(my_model,get_frame) #Do inference.
        post_process_image(get_results) #Perform post-processing.

if __name__ == "__main__":
        #Code execution starts here.
        my_profiler = cProfile.Profile() #Create a profiler instance.
        my_profiler.enable()  #Start Profiling.
        main()                #main function.
        my_profiler.disable() #End Profiling.
        save_results = my_profiler.results() #save the profiler results.
        #Code execution ends here.

```

Code Listing 5.1: profiling strategy pseudo code

6 Experiments

As discussed in previous chapters 4 ,5, we select four object detectors to perform a comparative analysis of their performance by means of code profiling. This chapter provides an explanation of the experimental design and description

6.1 Overview

Since the project's primary focus is to compare and evaluate the perception cycle time for a system deployed in a supermarket scenario, we propose four different sets of experiments (where each experiment has its sub-tasks) to evaluate the performance as effectively as possible. Although the project assumes a remote shopping scenario in supermarkets using a robot, we also consider different hardware configurations in our experiments. In the following sections, we explain each experiment as below:

- **Reason:** Motivation for experimenting.
- **Metric:** Metric used for analysis.
- **Description:** A complete description of how the experiment is conducted. What are the diffident hardware configurations , assumptions considered, and other relevant information.

Note: Unless stated in the respective experiment, the standard experimental setup is given in Figure 6.1,6.2.

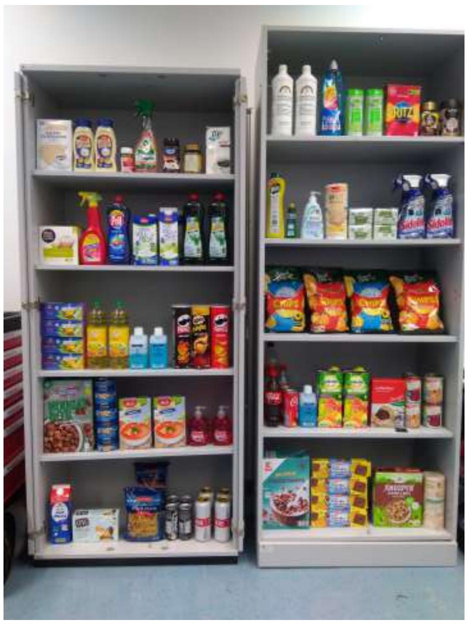


Figure 6.1: Grocery shelf for experiments

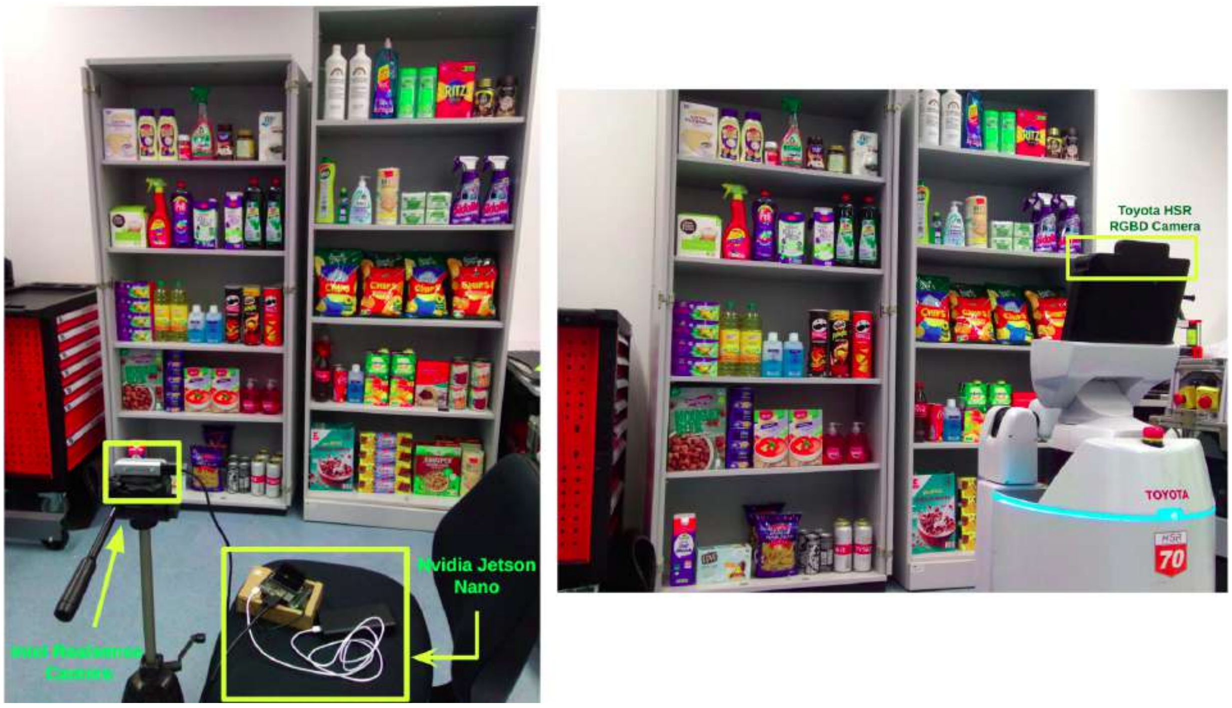


Figure 6.2: Hardware apparatus for experiments. Left image uses NVIDIA board with camera. Right image uses Toyota HSR.

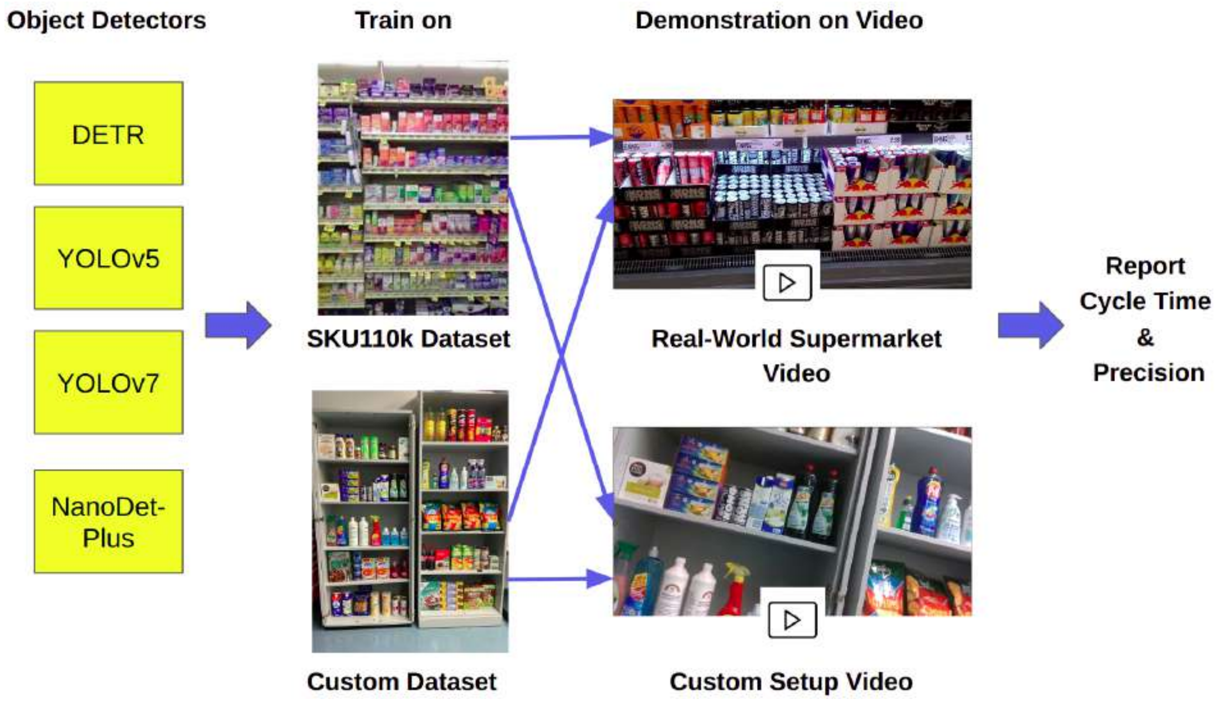


Figure 6.3: Experiment-1 overview diagram: The four selected object detectors in this experiment are trained on SKU110k and Small Groceries(custom) datasets. We create two custom videos, one from a real-world supermarket and the other from a custom setup, and cross-validate the models trained on one dataset with the other. Finally, we report cycle time and precision only on our custom video and datasets respectively.

6.2 Experiment-1: Object Detection Using High-Performance GPU

- **Reason:** In principle, object detection uses powerful machines with GPU and CPU configurations for reporting performance metrics. In real-world deployment scenarios, this is only possible sometimes. The central goal of this experiment is to consider the results as an ideal baseline for comparison. We also compare the results of the models trained on SKU110k and Small Groceries datasets.

- **Metric:** Precision, cycle time

- **Description:** The four selected object detectors in this experiment are trained on SKU110k and Small Groceries datasets using the workstation PC. The workstation has two NVIDIA GeForce RTX 3090 24GB GPU cards and AMD Ryzen 9 5900X

12-Core processor with 54GB RAM. All the trained models will be re-utilized in further experiments; please refer to section A.1 to get more details about training. For this experiment, we create two custom videos, one from a real-world supermarket and the other from a custom setup, shown in Figure 6.1. We also cross-validate the models trained on one dataset with the other, shown in Figure 6.3. The inference is done on the workstation PC, and the code profiling is done using Python's cProfile as discussed in section 5. We use the open-source test scripts each detector (in its repository) provides to calculate the precision. The vital things to note here are: (i)The cycle time is reported on the video with ten trials/detector, (ii) Precision is reported only using the test images from the Small Groceries dataset. The latter is because the video annotations are unavailable or created.

6.3 Experiment-2: Object Detection Using Embedded GPU

- **Reason:** Robots are expected to work with limited resources. Generally, it is only sometimes feasible to use high-end GPU and CPU computing resources in a deployment scenario due to cost/energy constraints. The same principle applies when robots get deployed in supermarket settings. In this experiment, we chose NVIDIA Jetson Nano as our embedded platform. We conducted object detection with two cameras to investigate the effect of different hardware configurations on the cycle time.
- **Metric:** Cycle time
- **Description:** We performed this experiment on the NVIDIA Jetson Nano 2GB version with the latest Jetpack 4.6.1 software installed. Unlike the previous experiment that used video during inference, we performed online object detection using two cameras (i.e., Intel Real-Sense D435 RGBD camera, Logitech C270 webcam) in this experiment. Figure 6.2 (left image) shows the basic experimental setup. The camera is connected to the Jetson board and faces toward the grocery shelves. We perform all the computations related to the object detector on the Jetson board and record the data in parallel using the code profiler. We install PyTorch, Intel Real-Sense camera drivers, from the source because the Jetson board is an embedded platform. The libraries cannot be installed directly as in a typical PC. We conduct ten trials for each detector(see Figure 6.4, and the detector infers ten times (i.e.,

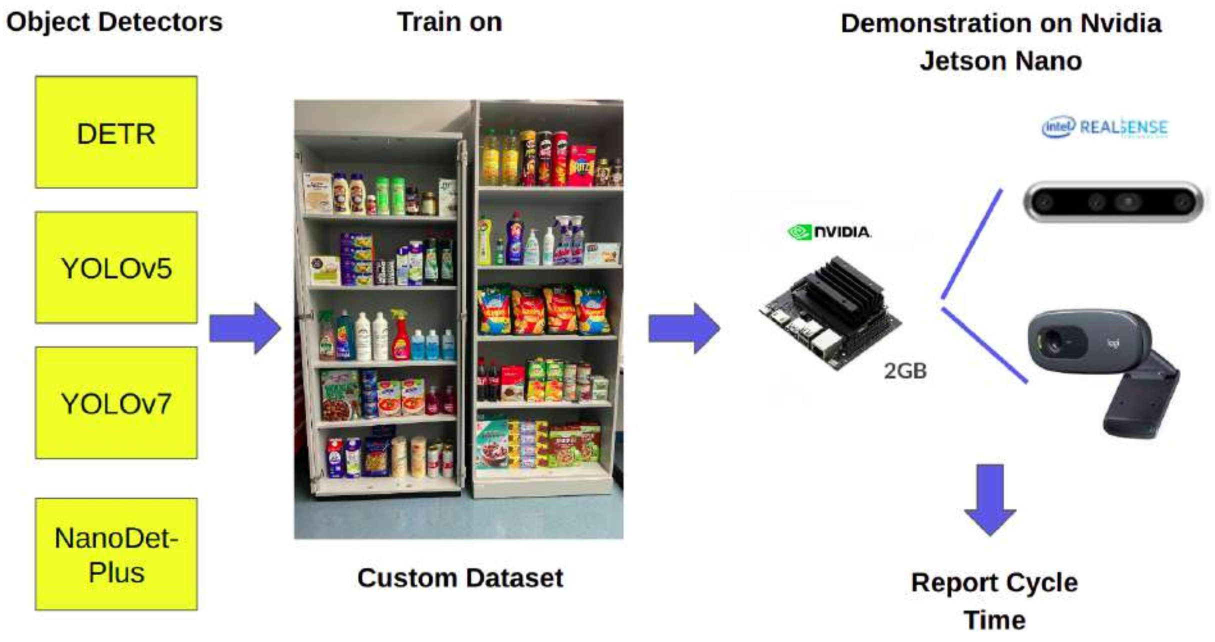


Figure 6.4: Experiment-2 overview diagram: The four selected object detectors in this experiment are trained on SKU110k and Small Groceries(custom) datasets (models reused from experiment 1).We demonstrate our object detection experiment on the NVIDIA Jetson Nano 2GB version. Unlike the experiment 1 that used video during inference, we performed object detection using two cameras (i.e., Intel Real-Sense D435 RGBD camera, Logitech C270 webcam) in this experiment and report the cycle time.

Detects bounding boxes for ten frames) per each trial. We do the latter process for both cameras.

6.4 Experiment-3: Object Detection Using Toyota Human Support Robot

- **Reason:** In order to verify how a robot's object detection capabilities perform when deployed in a supermarket scenario, one must not just rely on non-real-time experimentation, as discussed in experiment-1. We conduct experiments on the Toyota HSR to understand how the cycle time is affected when we deploy these object detectors on the robot.

- **Metric:** Cycle time

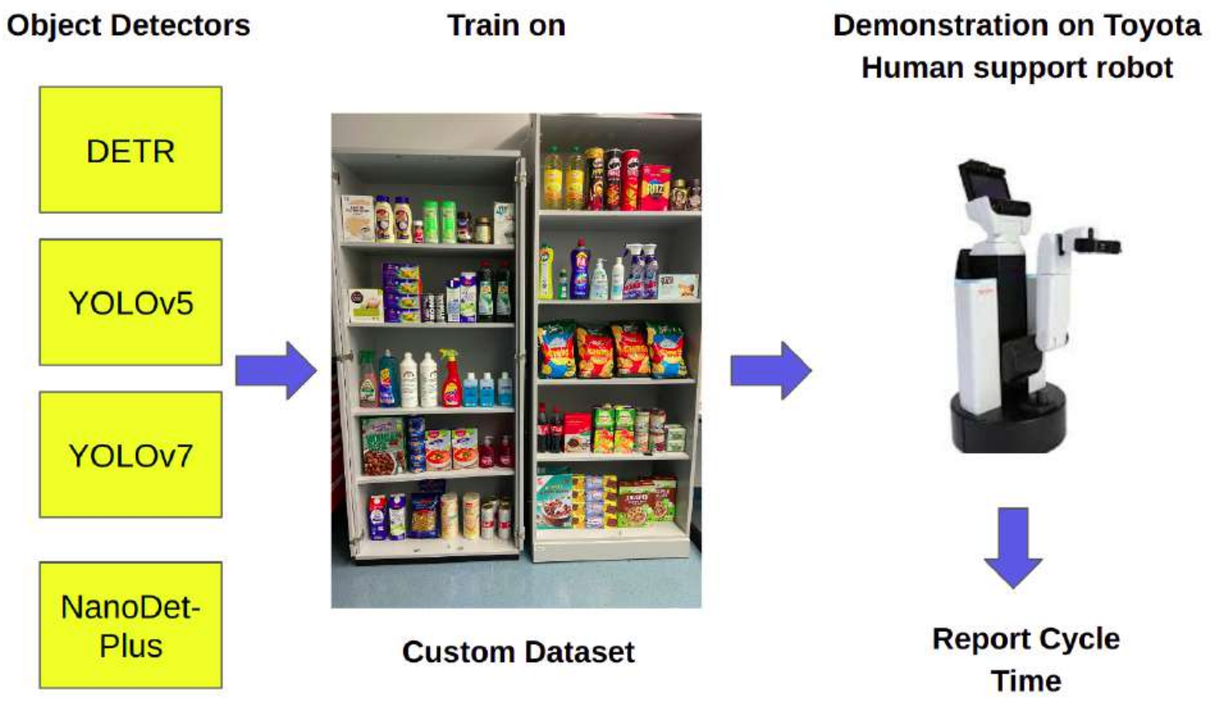


Figure 6.5: Experiment-3 overview diagram: Experiment-2 overview diagram: The four selected object detectors in this experiment are trained on SKU110k and Small Groceries(custom) datasets (models reused from experiment 1).We demonstrate our object detection experiment on the Toyota HSR platform. We use the robot's builtin RGBD camera for this experiment. Finally, we report the cycle time.

- **Description:** In experiments one and two, we have demonstrated how the object detectors perform concerning the cycle time on the workstation and the embedded platform. The current experiment demonstrates the same effect on a real robot. We choose the Toyota Human Support Robot with the following specifications: Intel Core i7 4th generation processor and 16GB RAM. The robot uses its built-in RGBD camera for object detection, as shown in Figure 6.2 (right image), facing the grocery shelves shown in Figure 6.1. A brief overview of the experiment is shown in Figure 6.5. In a real supermarket scenario, the robot's goal is not just to do the object detection process but also to do various tasks like localization, navigation, and manipulation. Keeping the latter point in mind, we allow various robot-related services to run in the background while conducting our experiment. With this, we also consider the computation load, which emulates the real-world use case as effectively as possible.

We use open-source Robot Operating System (ROS) based object detection nodes to run our experiment. We also follow the same code profiling strategy discussed in section 5. However, we came up with custom ROS-based inference nodes for DETR and NanoDet-Plus object detectors due to the unavailability of open-source implementations. Like experiment two, we conduct ten trials for each detector, and the detector infers ten times (i.e., Detects bounding boxes for ten frames) per each trial. We reuse the models trained on our Small Groceries dataset from our experiment, run the inference using the robot, and report the cycle time.

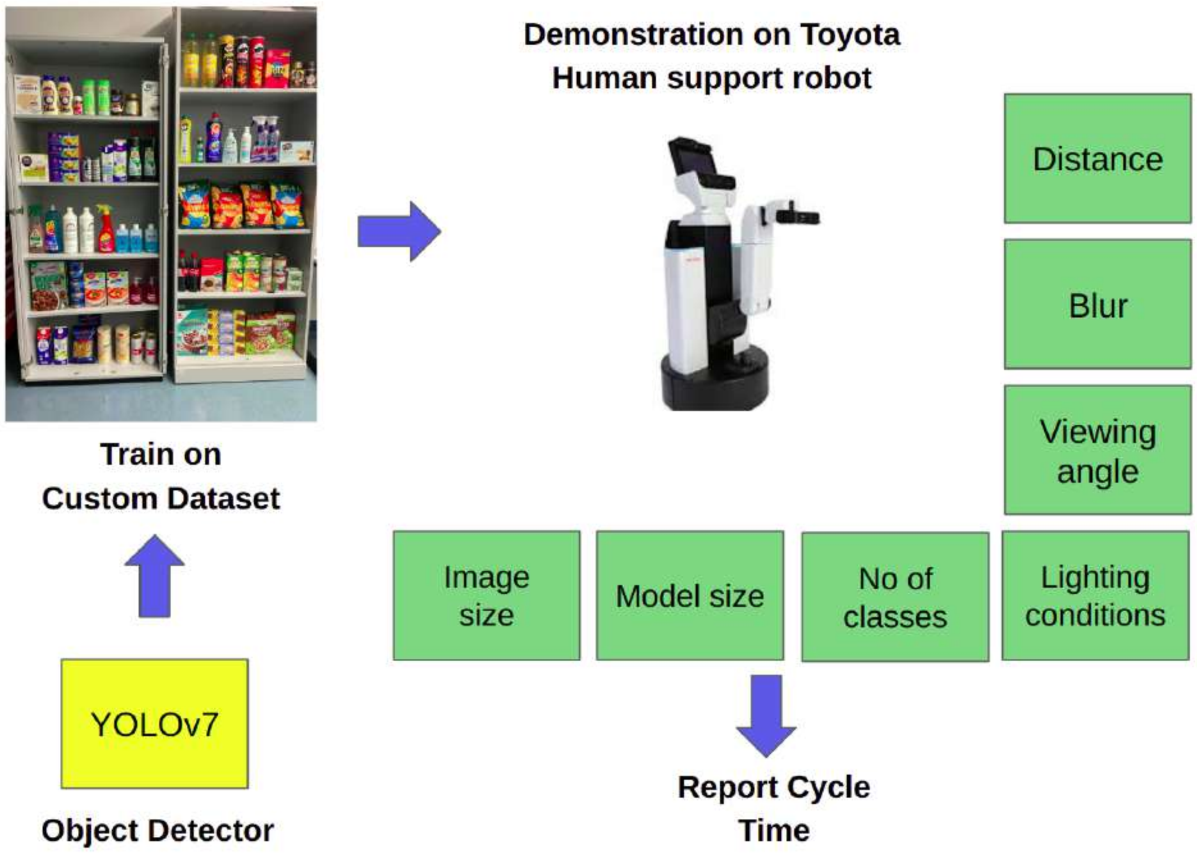


Figure 6.6: Experiment-4 overview diagram: In this experiment, we investigate the cycle time by considering seven perception-related challenges that can arise in a supermarket scenario: (1)Image size, (2) Model size, (3)Number of classes, (4) Lighting conditions, (5) Viewing angle, (6)Blur, and (7) Distance.We use the Toyota HSR platform for the object detection task similar to experiment 3 and report the cycle time.

6.5 Experiment-4: Factors Affecting Object Detection Cycle Time in Supermarkets

- **Reason:** The robot might face many challenges during deployment in a real supermarket scenario. In this experiment, we investigate various perception-related challenges like lighting conditions, viewing angles, distance from the object, and image size that affect cycle time in supermarkets.
- **Metric:** Cycle time
- **Description:** Experiments one, two, and three demonstrated how changing the hardware platforms affects the perception cycle time. In this experiment, we investigate the cycle time by considering seven perception-related challenges that can arise in a supermarket scenario. We use the Toyota HSR platform for the object detection task similar to experiment 3. We hypothesize that the seven factors do not affect the cycle time. However, we want to include them in the experiment for completeness because other related works do not necessarily include them explicitly. Unlike the previous experiments, we consider only YOLOv7 as our object detector and perform five trials per each sub-task. We are interested in evaluating the state-of-the-art object detector, so we are considering the latter. We consider the following seven factors as individual sub-tasks(see Figure 6.6), and the explanation is as follows:
  - Image size: In this task, we investigate the effect of cycle time for four different dimensions: 320x320, 480x480, 640x640, and 1280x1280.
  - Model size: Since the model architectures come in different sizes, we investigate the YOLOv7's tiny, small, and medium-sized models.
  - Number of classes: To examine the effect of the number of classes per given frame. We consider four, ten, fifteen, and twenty classes for our analysis.
  - Lighting conditions: We consider low, standard, and sunny illumination levels for this task because robots are generally expected to work in different lighting conditions. To create a low-light scenario, we close all the curtains in the room during a low-light day. To create a sunny environment, we move the experimental setup in Figure 6.1 to face the windows during a sunny day.
  - Viewing angle: We consider left, right, and straight viewing angles in this task.

- Effect of blur: To investigate the effect of the blur, we programmatically induce a Gaussian blur into the camera using the OpenCV library. We consider four different blur levels in increasing order.
- Distance from the object: study the effect of the distance, we consider four different distances from the shelves: 70cm, 110cm, 160cm, and 200cm.

7

Results

This chapter presents our results from the four experiments (refer chapter 6) conducted as part of this project. We discuss the results of each experiment in its section, with the data and results specific to that experiment presented. In the final section, we present a summary considering all four experiments.

7.1 Experiment-1: Object Detection Using High-Performance GPU

7.1.1 Overview

This section discusses and provides results concerning cycle time and precision. As discussed in chapter 6, we trained our four object detectors on SKU110k and our Small Groceries datasets. We use the trained models (for DETR, NanoDet-Plus, YOLOv5, and YOLOv7) of the SKU110k dataset for the visualization results, which we will discuss later in this section. We utilize the trained models of our dataset in all our experiments. We ran the object detection task (using the trained model on our dataset) on our Small Groceries test dataset to report the accuracy and on a sample video (created over the experimental setup shown in Figure 6.1) to report the cycle time.

In order to report the cycle time, we conduct ten trials for each detector. Each trial consists of ten iterations (also called runs or inferences) of detecting an object in a given frame (one frame for each iteration). For each trial, we use the profiler to record the trace data, as discussed in section 5.3.1. After going through the trace files, we categorize the main tasks discussed in same section 5.3.1. We consider the number of calls, cumulative time, and per-call time for our analysis from the trace files (refer to section A.3 for a sample trace file). We follow the latter approach for the rest of our experiments.

7.1.2 Accuracy Results

Before we discuss the cycle time, the accuracy results are as follows:

- We achieve 0.941 mAP@0.5 for the YOLOv5 model.
- The mAP@0.5 value for the YOLOv7 model is 0.966.
- For NanoDet-Plus mAP@0.5, the value is 0.595.
- Finally, for the DETR[3] model, we get 0.321 for mAP@0.5 metric.

7.1.3 Cycle Time Results

Table 7.1 shows each detector's total average time for various tasks. We take the average for the ten trials conducted. From the table, it is clear that YOLOv5 performs better regarding the total cycle time than the other detectors. A similar comparison is shown in Figure 7.1 (in log scale[4]) using the bar plots. From Table 7.1 and Figure 7.1, we can deduce that YOLOv5 performed top in total cycle time, model loading time (generally done only once per each trial), and inference time tasks. The cycle time performance(least value as better) is in this order with the corresponding average time value given in seconds: YOLOv5(3.805 sec), NanoDet-Plus(3.939 sec), YOLOv7(4.220 sec), and DETR(14.639 sec). The YOLOv7 detector took the least time in the post-processing and miscellaneous operations tasks. The NanoDet-Plus model performed well, falling behind YOLOv5 by a small margin, but took the least time for pre-processing tasks. The DETR model took the highest time in most tasks except the miscellaneous task, where YOLOv5 came last. Additionally, Figure 7.2 shows the total cycle time for ten trials, and Figure 7.3 shows the individual cycle time for each task, comparing the four detectors in the log scale. By

[3]We used more number of epoch while training the DETR model as the accuracy was too low with 50 epochs. See section A.1.

[4]We show most of our figures related to time in the log scale and its corresponding time (wherever required) in seconds unless we mention otherwise.

Task / Detector	**DETR**	**NanoDet-Plus**	**YOLOv5**	**YOLOv7**
Total cycle time	14.639±5.206	3.939±0.018	**3.805±0.017**	4.220±0.069
Model loading time	4.265±1.444	3.597±0.017	**3.290±0.016**	3.797±0.064
Pre-processing time	0.092±0.033	**0.048±0.003**	0.159±0.007	0.139±0.010
Inference time	10.130±3.737	0.185±0.003	**0.026±0.001**	0.190±0.009
Post-processing time	0.048±0.017	0.015±0.001	0.043±0.002	**0.008±0.001**
Miscellaneous time	0.104±0.039	0.094±0.008	0.286±0.006	**0.087±0.003**

Table 7.1: The table shows the total average time(seconds) along with the standard deviation for various tasks for each detector. The highlighted are the least time taken for that particular task and detector.

observing the inference task, we can see the difference between Figure 7.3 and Figure 7.4 as the time taken in Figure 7.4 is less due to single iterations, while Figure 7.3 takes more time due to ten iterations.

Figure A.5 shows the sample visualizations of all four detectors on the custom setup video using the model trained on the Small Groceries dataset. Similarly, Figure A.6 shows visualizations of all four detectors on the real-world supermarket video using the model trained on the SKU110k dataset. The rest of the visualizations are also available in the same section A.6.1.

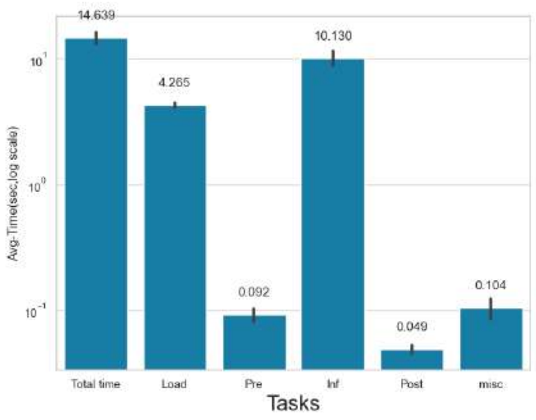
The main takeaway results from experiment-1 are:

- The YOLOv7 detector has the highest accuracy on our Small Groceries test dataset with a 0.966 mAP@0.5 score.

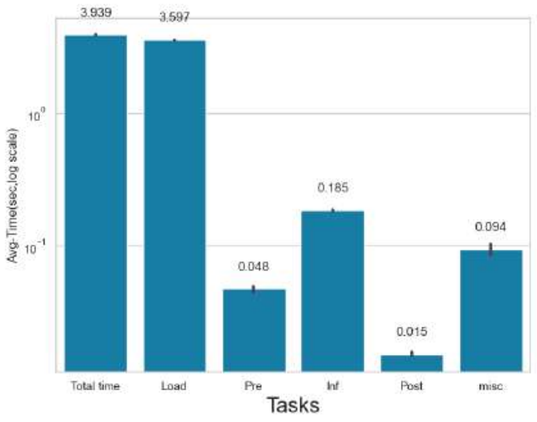
- Concerning the cycle time, the YOLOv5 detector performs top, and DETR performs last.

- Interestingly NanoDet-Plus performed consistently in most of the tasks and better in inference tasks than the YOLOv7 detector.

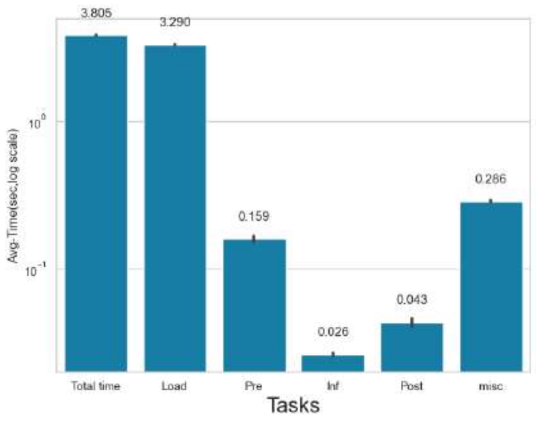
- Regarding accuracy and cycle time, The DETR detector's performance was not up to the mark compared to the other three detectors.



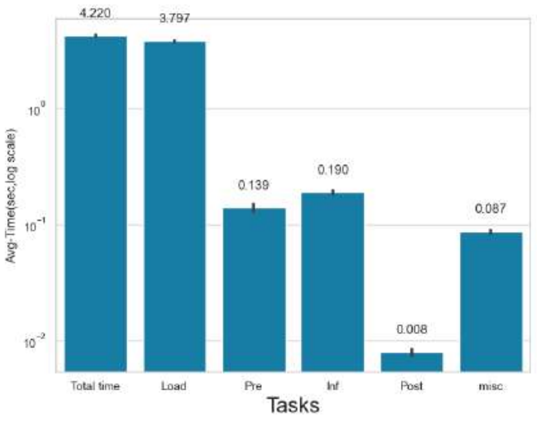
(a) DETR



(b) NanoDet-Plus

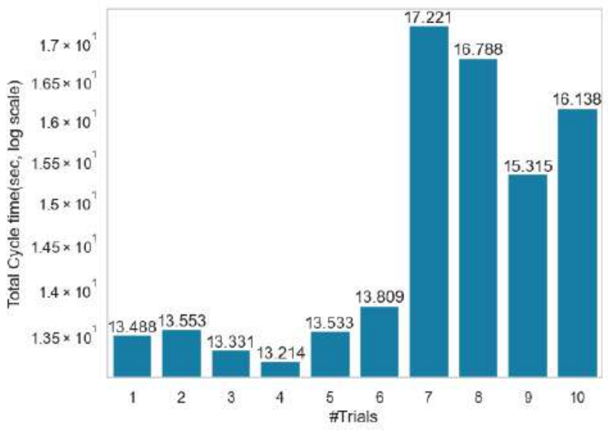


(c) YOLOv5

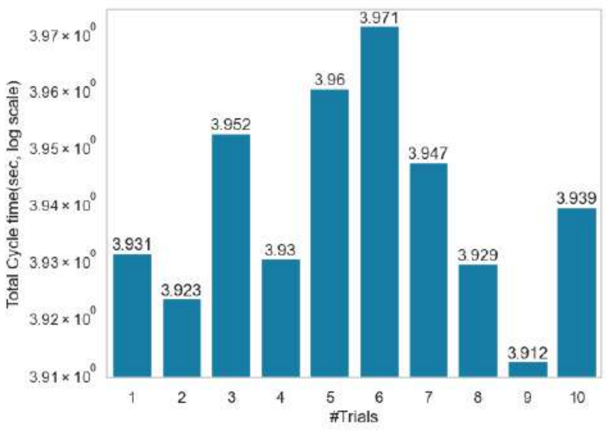


(d) YOLOv7

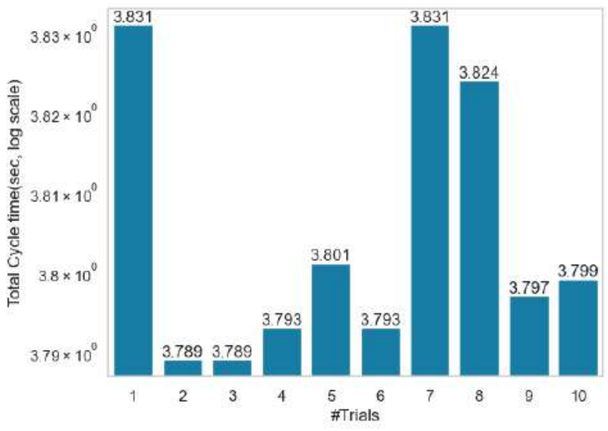
Figure 7.1: Bar plots showing the average time for each detector with different tasks. The x-axis represents the different tasks, and the y-axis represents the average time in the log scale with its corresponding value in seconds.



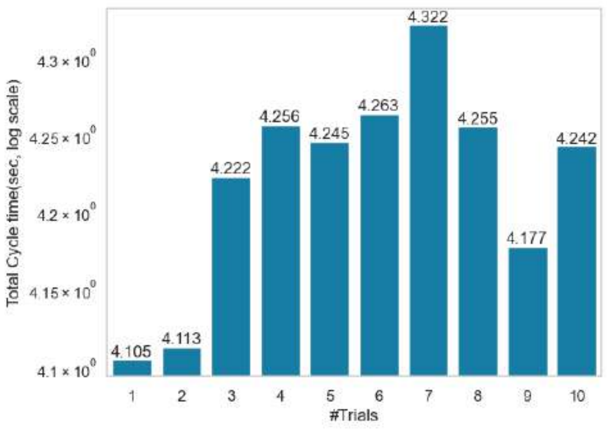
(a) DETR



(b) NanoDet-Plus

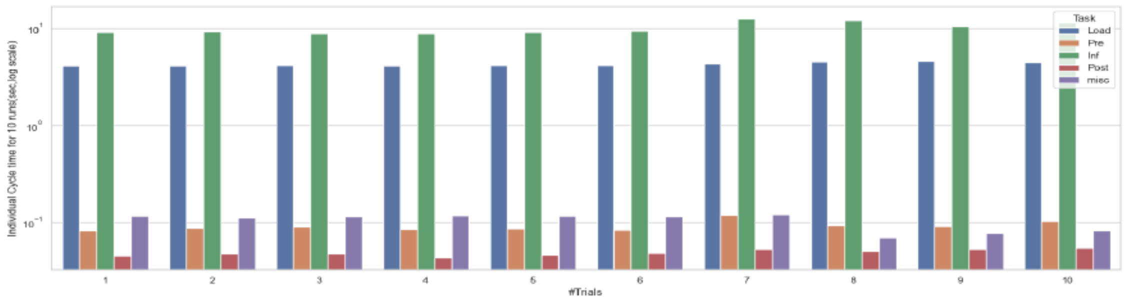


(c) YOLOv5

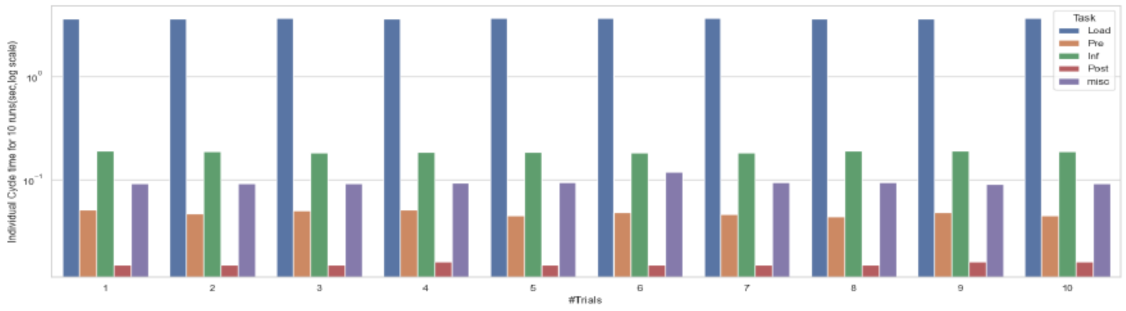


(d) YOLOv7

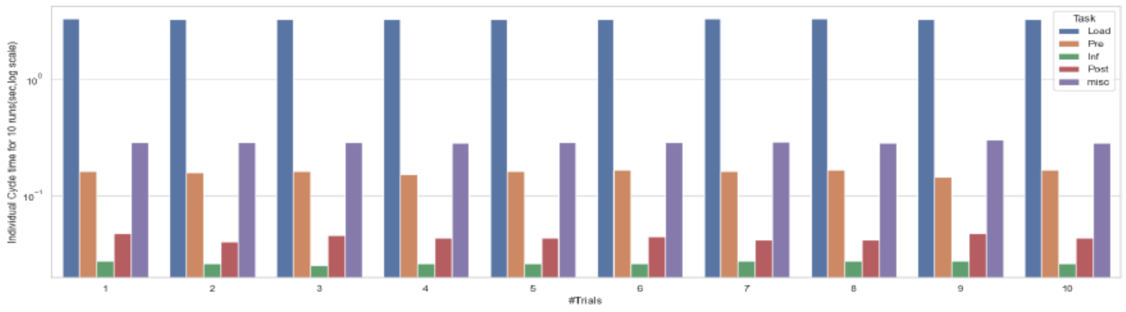
Figure 7.2: Bar plot showing the total cycle time for each trial.The x-axis represents the total number of trials, and the y-axis represents the total cycle time for each trial in the log scale with its corresponding value in seconds.



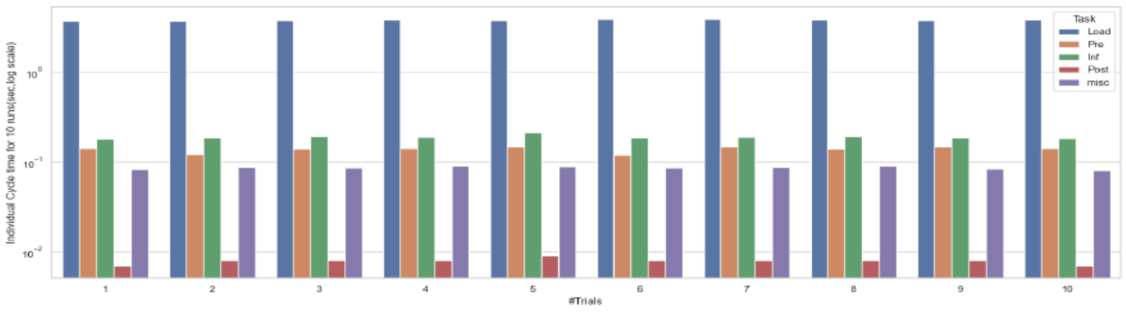
a)DETR



b)NanoDet-Plus

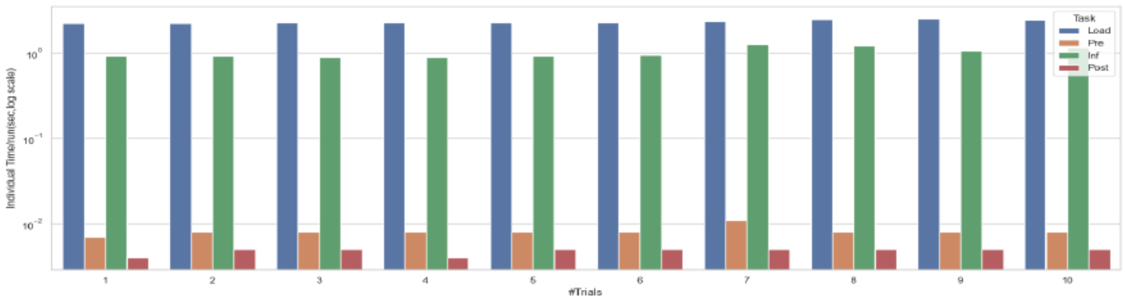


c)YOLOv5

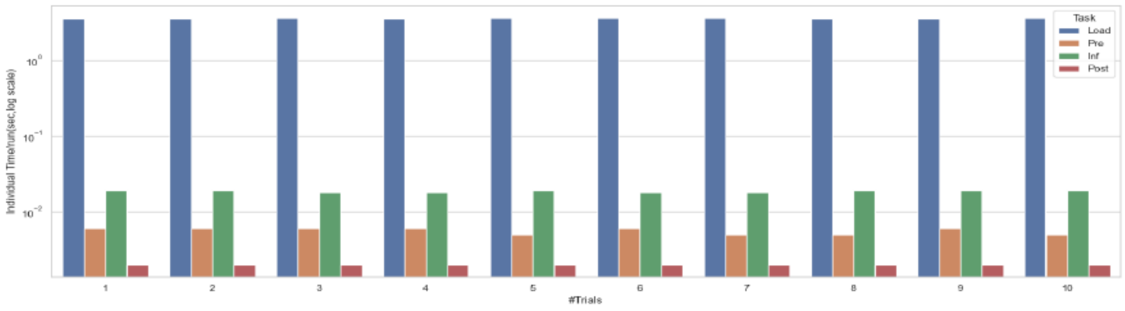


d)YOLOv7

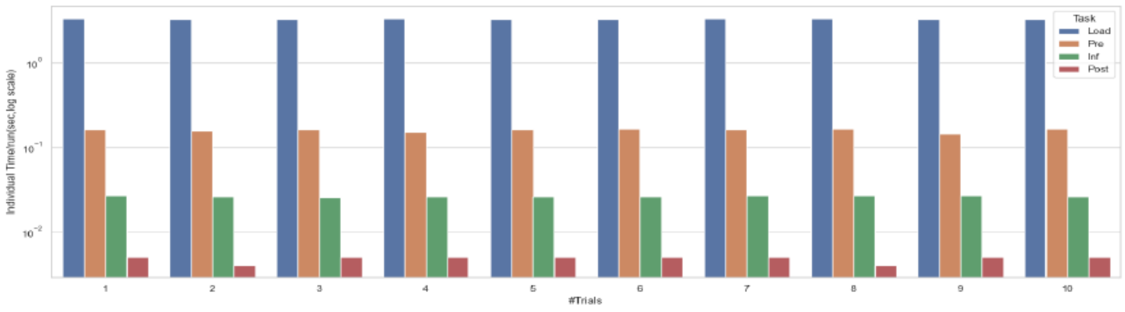
Figure 7.3: The bar plots shows the individual cycle time for each task, comparing the four detectors in the log scale.



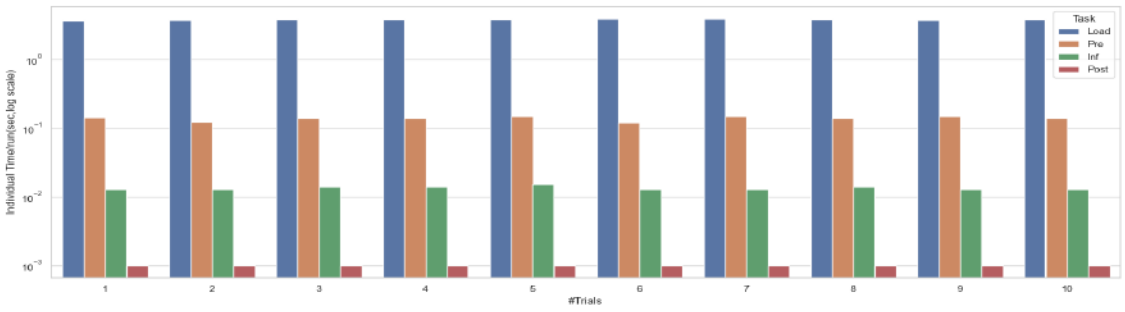
a)DETR



b)NanoDet-Plus



c)YOLOv5



d)YOLOv7

Figure 7.4: The bar plots shows the individual time per run for each task, comparing the four detectors in the log scale

7.2 Experiment-2: Object Detection Using Embedded GPU

7.2.1 Overview

In this experiment, we demonstrate the results of the object detection task on an embedded platform, the NVIDIA Jetson Nano board. In addition, we also consider presenting the results for two different cameras, Intel RealSense (RS) and webcam (WC). The approach used for obtaining the results is the same as in experiment 1. From this experiment onwards, we report the results related to cycle time alone.

7.2.2 Cycle Time Results

In this section, we first discuss the results for the RealSense camera, next for the webcam, and then compare the results considering both.

- **Results on the RealSense camera**
  - Table 7.3 shows that NanoDet-Plus performs best in total cycle time compared to other detectors surpassing the state-of-the-art object detectors. A similar comparison is shown in Figure 7.5, Figure 7.6 (in log scale) using the bar plots.
  - The YOLOv7 detector performs best during inference, but the cycle time is highest in the post-processing task.
  - We observed that the performance of YOLOv5 could be more consistent. It performs best in post-processing tasks but takes a longer average time for inference and pre-processing tasks.
  - The DETR detector takes the highest average time in model loading, inference, and miscellaneous tasks hence attaining the highest average total cycle time.
- **Results on the webcam**
  - The YOLOv5 detector performs best when it comes to the total cycle time. The latter is because of the lower average time in model loading and inference tasks(as shown in Table 7.2).

Detector	DETR	NanoDet-Plus	YOLOv5	YOLOv7
Task/Camera	**Webcam**	**Webcam**	**Webcam**	**Webcam**
Total cycle time	892.117±186.683	322.753±18.801	**222.558±20.526**	322.663±53.421
Model loading time	206.105±64.289	**34.727±2.896**	69.210±4.385	115.046±18.720
Pre-processing time	4.7133±0.183	**1.548±0.206**	11.285±2.071	4.770±0.154
Inference time	655.779±134.413	268.769±17.533	**134.037±18.299**	168.476±32.056
Post-processing time	4.660±0.268	8.118±3.892	**2.845±1.779**	26.539±9.554
Miscellaneous time	20.860±10.645	9.591±1.127	**5.180±0.566**	7.832±2.172

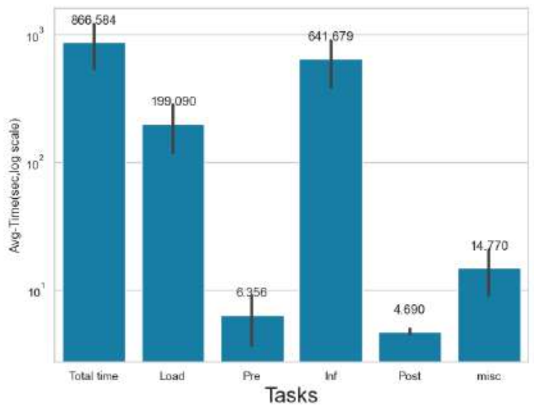
Table 7.2: The table shows the total average time(seconds) along with the standard deviation for various tasks for each detector considering webcam. The highlighted are the least time taken for that particular task, camera, and detector.

Detector	DETR	NanoDet-Plus	YOLOv5	YOLOv7
Task/Camera	**RealSense camera**	**RealSense camera**	**RealSense camera**	**RealSense camera**
Total cycle time	866.584±331.870	**246.768±22.821**	356.827±21.209	308.810±40.252
Model loading time	199.090±80.420	**34.024±3.932**	69.811±4.201	110.646±14.445
Pre-processing time	6.356±2.720	**1.675±0.213**	11.199±4.605	3.404±0.272
Inference time	641.679±257.160	194.878±21.635	265.905±24.057	**162.625±29.019**
Post-processing time	4.690±0.220	8.001±5.791	**4.229±3.717**	23.626±3.945
Miscellaneous time	14.770±5.740	8.190±1.613	**5.684±0.769**	8.509±2.057

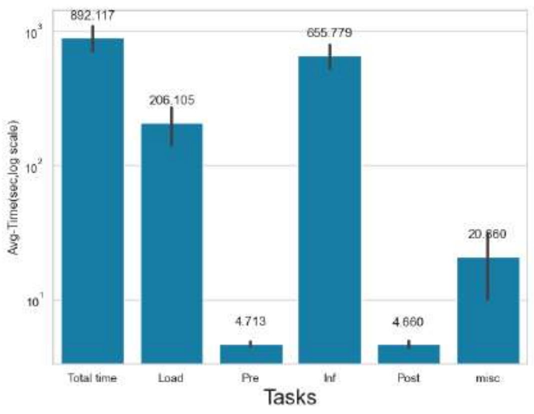
Table 7.3: The table shows the total average time(seconds) along with the standard deviation for various tasks for each detector considering RealSense camera. The highlighted are the least time taken for that particular task, camera, and detector.

- The YOLOv7 detector took a higher cycle time in the loading and pre-processing tasks but performed second in the inference task after the YOLOv5 detector.
- The NanoDet-Plus performs best in model loading and pre-processing tasks but sub-optimally in inference and post-processing tasks.
- The DETR detector takes the highest average time in model loading, inference, and miscellaneous tasks hence attaining the highest average total cycle time.

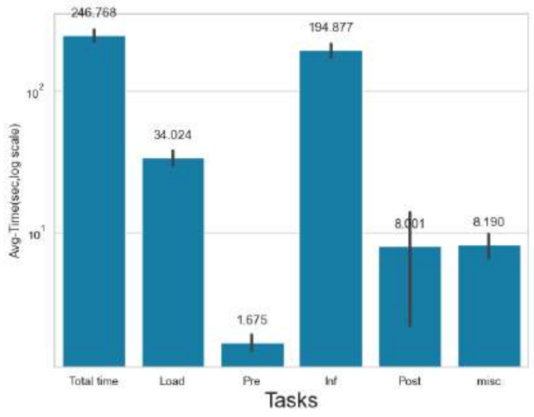
- **Common comparison**
  - Comparing the results of both cameras, we observe that the NanoDet-Plus detector performs the best with the RealSense camera but attains a higher average cycle time with the webcam. The exact latter behavior is observed for the YOLOv5 detector by requiring a low average cycle time for the webcam and higher with the RealSense camera.



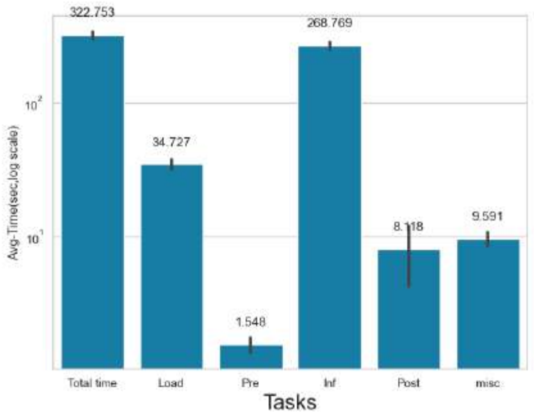
(a) DETR(RS)



(b) DETR(WC)

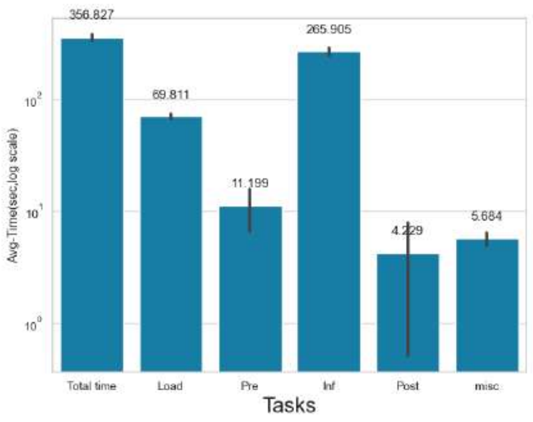


(c) NanoDet-Plus(RS)

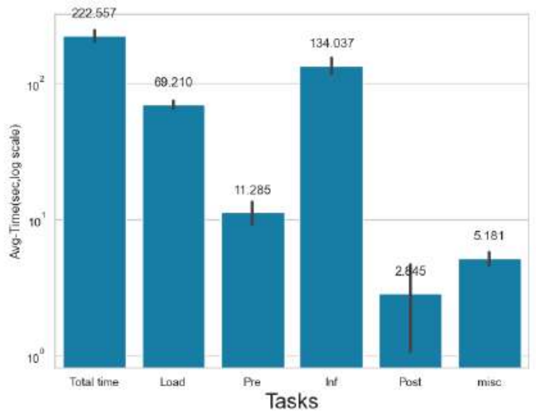


(d) NanoDet-Plus(WC)

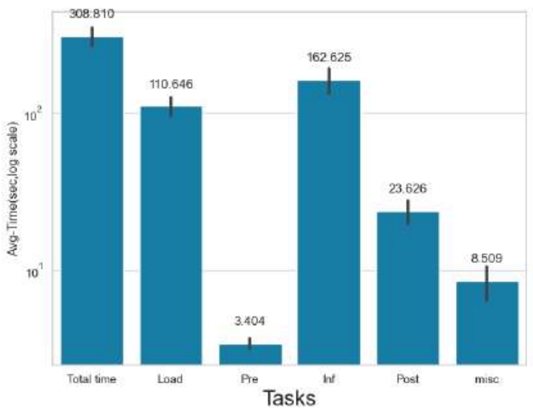
Figure 7.5: Bar plots showing the average time for DETR and NanoDet-Plus detectors with different tasks. The x-axis represents different tasks, and the y-axis represents the average time in the log scale with its corresponding value in seconds.



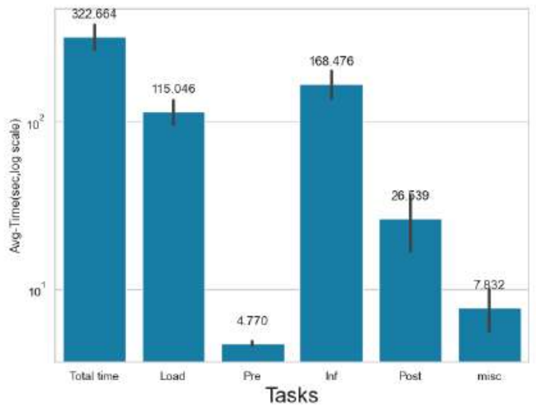
(a) YOLOv5(RS)



(b) YOLOv5(WC)



(c) YOLOv7(RS)



(d) YOLOv7(WC)

Figure 7.6: Bar plots showing the average time for YOLOv5 and YOLOv7 detectors with different tasks. The x-axis represents different tasks, and the y-axis represents the average time in the log scale with its corresponding value in seconds.

– The YOLOv7 detector performs consistently using both cameras but requires slightly more cycle time while using the webcam.

– The total average cycle time required for the DETR, NanoDet-Plus, and YOLOv5 is more while using the webcam as compared to when using the RealSense camera. However, the YOLOv5 has the opposite behavior with respect to the latter discussion.

– We can say that the camera selection criteria will affect the object detector's cycle time depending upon which object detection method is used, based on the above points.

– Interested readers can go through section A.6.2 to compare individual cycle times based on task, camera, and detector.

The main takeaway results from experiment-2 are:

– Comparing the four detectors, we find that the NanoDet-Plus detector performs best using the RealSense camera and YOLOv5 using the webcam. But none of the detectors met our real-world use cases due to high cycle time.

– The camera selection criteria will affect the perception cycle time. we expect the latter behaviour due the different processing speeds of the camera.

7.3 Experiment-3: Object Detection Using Toyota Human Support Robot

7.3.1 Overview

In this experiment, we demonstrate the results of the object detection task on the Toyota HSR platform. Unlike experiments one and two, where we report the cycle time for the individual tasks, in this experiment, we only report the total cycle time for the object detector. The latter case is because using our profiler on the Robot Operating System (ROS) based nodes did not provide the entire trace as the ROS-based code was blocking the profiler's execution.

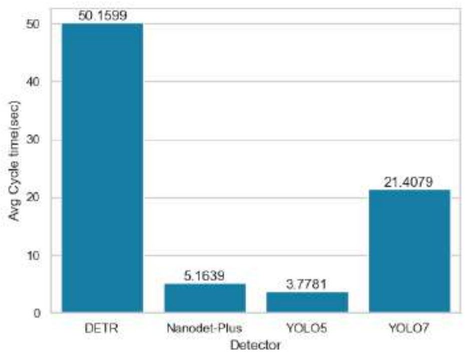
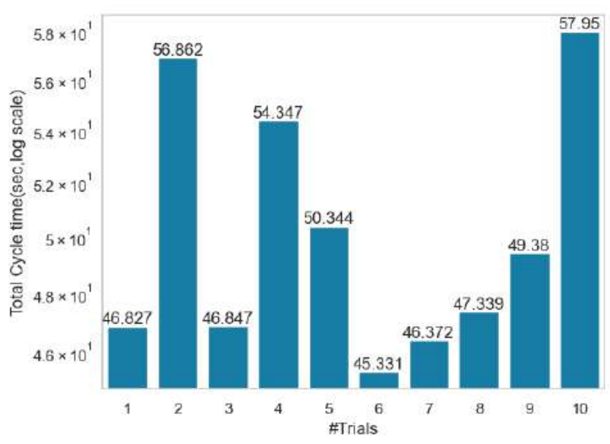


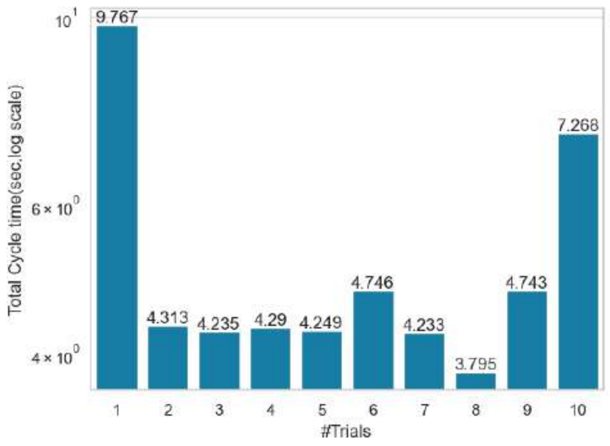
Figure 7.7: Bar plot showing the average time for each detector.

7.3.2 Cycle Time Results

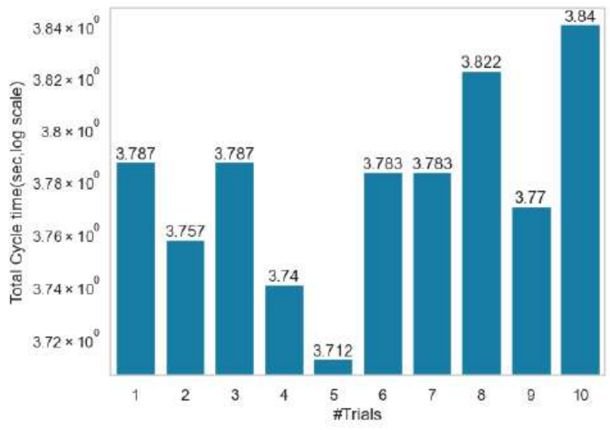
We conduct our experiments on the robot with ten trials for each detector, as shown in Figure 7.8. Figure 7.9 shows sample visualizations of each detector. The average cycle time for each detector (see Figure 7.7) in increasing order are YOLOv5 at 3.778(±0.037) seconds, NanoDet-Plus at 5.163(±1.883) seconds, YOLOv7 at 21.408(±0.274) seconds, and DETR at 50.160(±4.613) seconds. So YOLOv5 performs best on the Toyota HSR platform compared to other detectors. Additionally, based on the manual tracing, the model loading time for each detector is as follows: NanoDet-Plus at 0.616 seconds, YOLOv5 at 1.221 seconds, DETR at 2.186 seconds, and YOLOv7 at 2.293 seconds. In this experiment, NanoDet-Plus takes the least average model loading time.



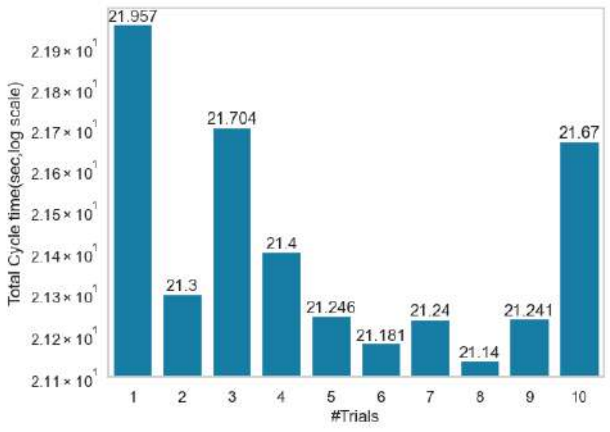
(a) DETR



(b) NanoDet-Plus

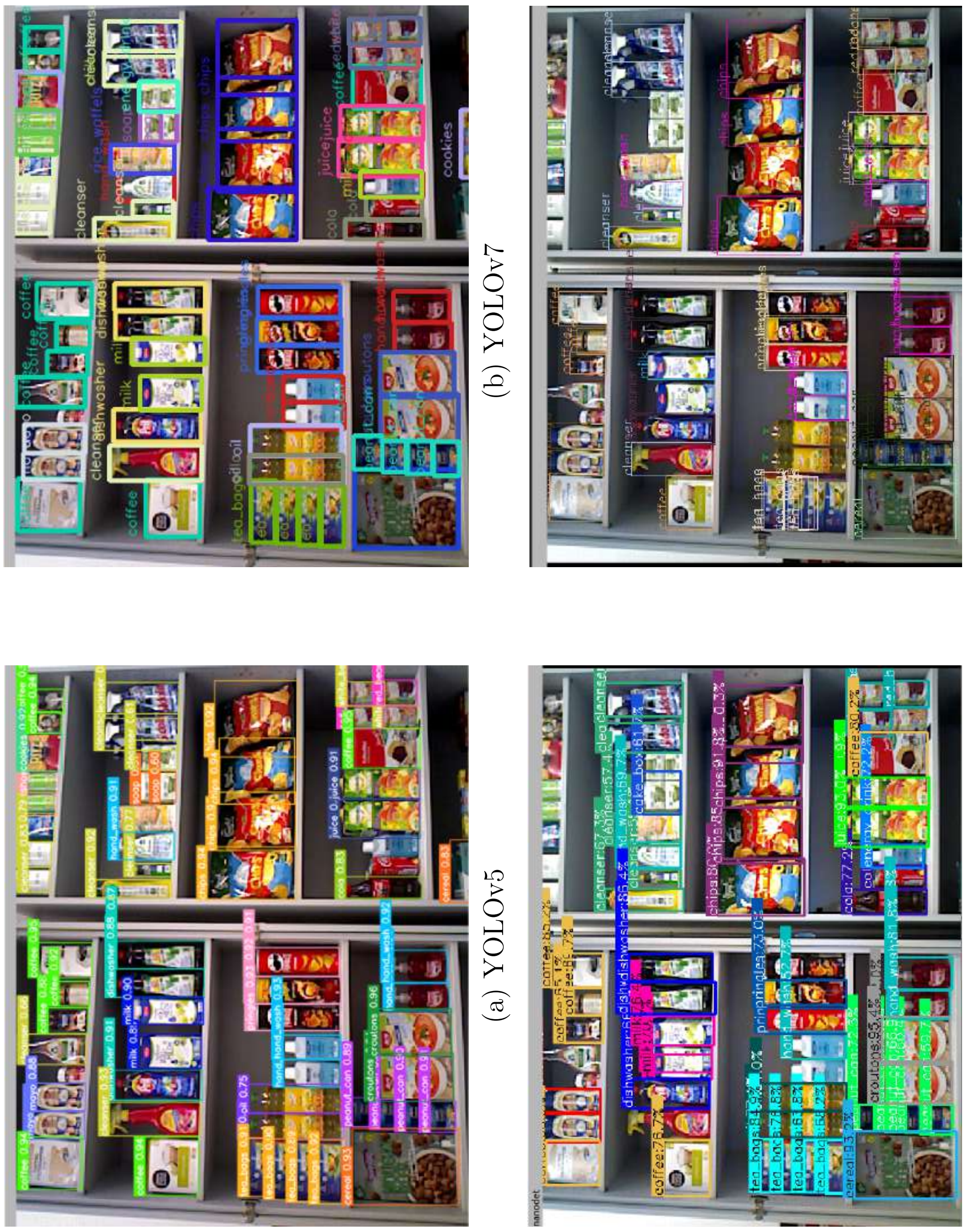


(c) YOLOv5



(d) YOLOv7

Figure 7.8: Bar plots showing the total cycle time for each trial.The x-axis represents the total number of trials, and the y-axis represents the total cycle time for each trial in the log scale with its corresponding value in seconds.



(a) YOLOv5

(b) YOLOv7

(c) NanoDet-Plus

(d) DETR

Figure 7.9: Sample detections for the four detectors on Toyota HSR.

7.4 Experiment-4: Factors Affecting Object Detection Cycle Time in Supermarkets

7.4.1 Overview

In this experiment, we demonstrate the results of the object detection task on the Toyota HSR platform, similar to experiment 3. In this experiment, we discuss the results related to various factors that affect the object detection task in supermarket scenarios. Unlike the previous experiments, we only consider the YOLOv7 detector and five trials per task. Table 7.4 shows the summary of all results in this experiment.

7.4.2 Cycle Time Results

Below are the results for the seven factors that may affect the cycle time:

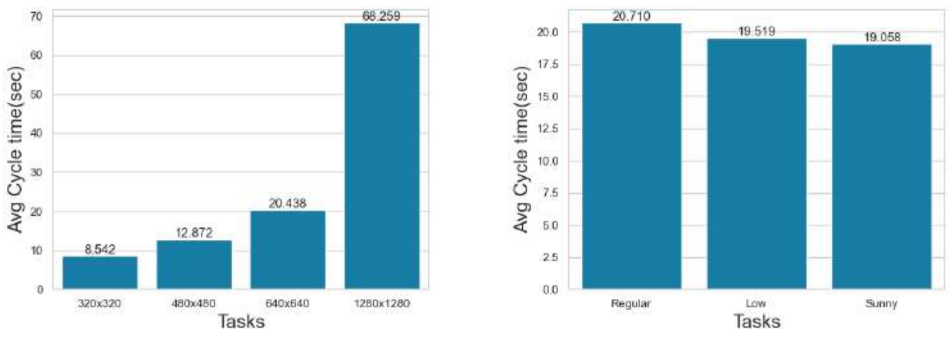
- Image size: From Figure 7.10(a), we observe that as the image dimensions increase, the average cycle time for each image dimension increases. From this task, image size directly relates to the cycle time proportionally (i.e., cycle time increases with increase in image size).
- Model size: This task investigates the YOLOv7's tiny, small, and medium-sized models. From Figure 7.11(e), we observe that as the model size increases, the average cycle time decreases and then increases because the tiny model takes more time

  than the small and medium-sized models. The latter behavior is different from what someone would expect (i.e., cycle time increases with the increase in model size). We suspect that while experimenting with the tiny model, the robot might have received an additional compute load from a different process as compared with experimenting with the small, medium-sized models. The latter reason is why the cycle time is more for the tiny model. So based on our experiments, varying model sizes had no significant effect on the cycle time.
- Number of classes: For this task, we considered four different class levels in increasing order. Figure 7.11(c) shows that the parameter number classes do not directly affect the cycle time, as the plot does not show consistent patterns or behaviors. We also

verified the live detections from the robot, as shown in Figure 7.12(a). The missed detection of few objects is causing the cycle time to decrease as we increase the number of classes, but almost all the objects were detected.

- Lighting conditions: Figure 7.10(b) depicts that the cycle time decreases as we move from regular lighting conditions to low-light and sunny scenarios. The latter behavior is because the number of detections during the sunny day decreased (see Figure 7.13(b)), and this caused the cycle time to decrease. The model's accuracy attributes to the cycle time. If the model had good performance (over predictions) in sunny environments, the detections would increase, causing the cycle time to increase. On the other hand, if the model had poor performance in sunny environments, the detections would decrease, leading to a decrease in the cycle time. Our experiment shows that the cycle time increases or decreases depending on the model's accuracy, but the cycle time has no direct effect when considered alone in different lighting conditions.

- Viewing angle: In this task, we consider a straight viewing angle as our baseline for comparison. So from Figure 7.11(a), the left view angle took slightly more time than the straight viewing angle. However, the right viewing angle took slightly less time than the straight one. Based on the latter observations, the viewing does not show consistent patterns or behaviors. Hence the viewing angle does not affect the total

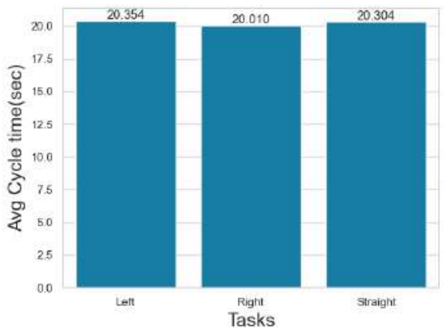


(a) Image size (b) Lighting conditions

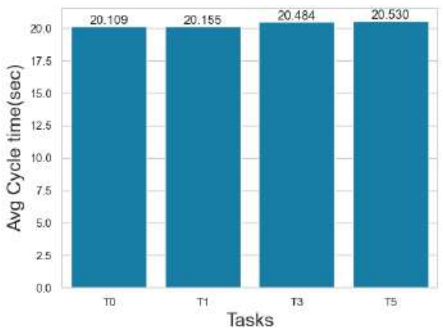
Figure 7.10: Bar plots show the total average cycle time for each task. The two plots show how the cycle time varies for different image sizes and lighting conditions.

cycle time. Figure 7.13(a) shows the sample detections in three different views. The detector performs well with most of the detections but misses a few due to occlusions caused by the viewing angles.

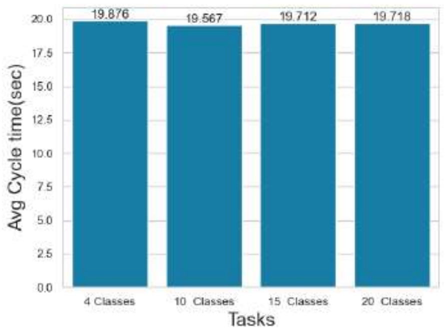
- Effect of blur: From Figure 7.11(b), as we increase the blur threshold(T) from T0 to T5, the cycle time increases slightly and steadily. Figure 7.13(b) shows how the object detector performs on different blur levels. Hence the blur directly relates to the cycle time proportionally.
- Distance from the object: Correlating our observations from Figure 7.11(d) and Figure 7.14, we see that as we move away from the grocery shelf, the average cycle time increases to 110cm and decreases from 110cm to 200cm, and the number of objects in the frame also increases based on the field of view. From the latter observations, as the plot does not show consistent patterns or behaviors, we deduce that varying distance from the grocery shelf does not affect the cycle time. Additionally, from Figure 7.14, we observe that, at a 200cm distance, we see many incorrect detections in the frame.



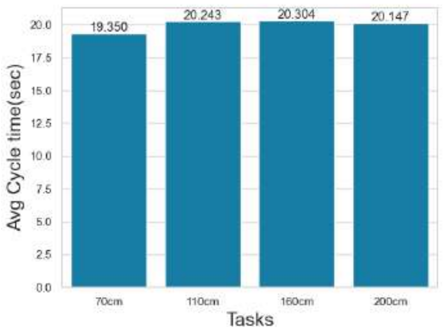
(a) Viewing angle



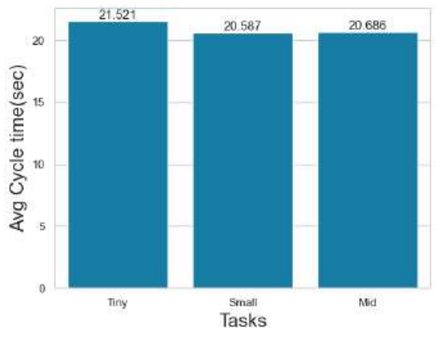
(b) Blur



(c) Number of classes



(d) Distance

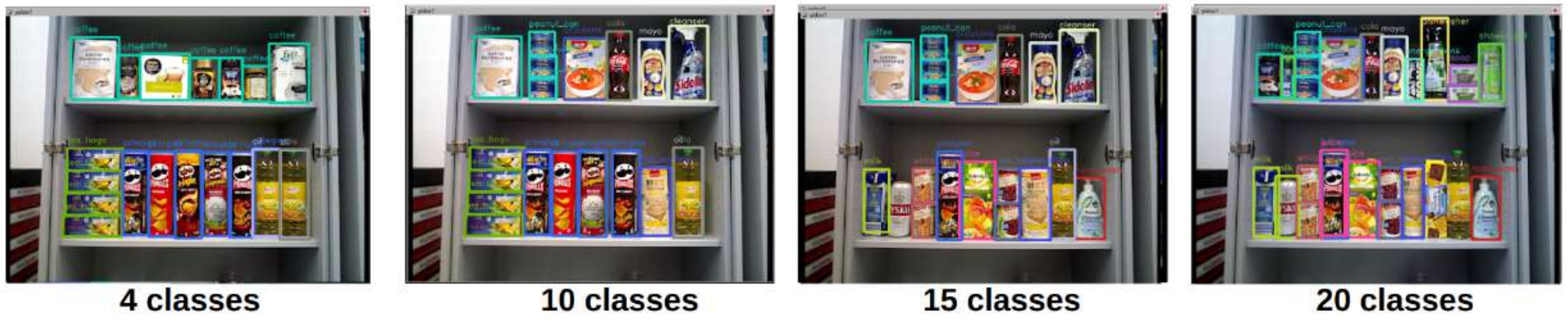


(e) Model size

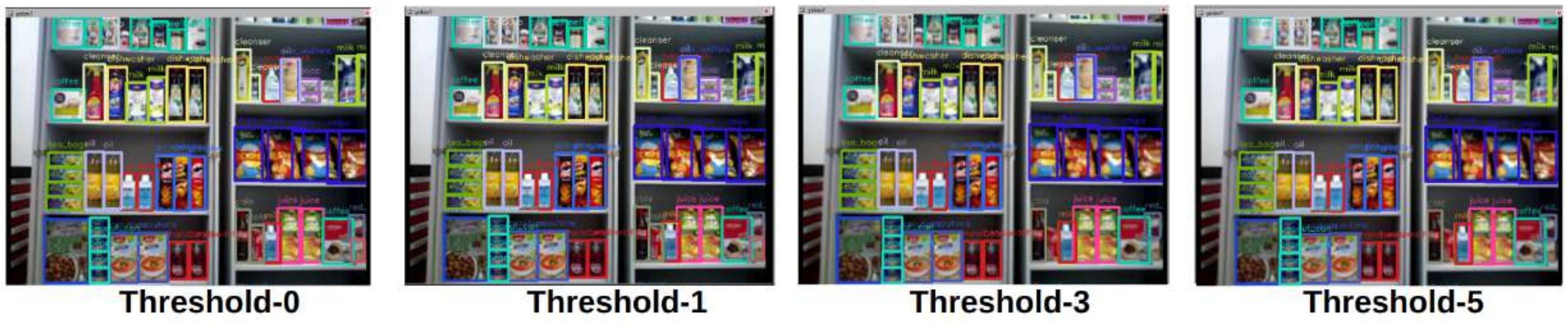
Figure 7.11: Bar plots show the total average cycle time for each task. The five plots show how the cycle time varies for different viewing angle, blur, number of classes distance, and the model size.

Criteria	Task	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average cycle time	Standard deviation (±)
Viewing angle	**Left**	20.249	20.303	20.180	20.730	20.310	20.354	0.216
	Right	19.681	20.227	20.655	19.636	19.851	20.010	0.429
	Straight	20.673	21.125	19.659	20.274	19.791	20.304	0.610
Blur	**T0**	20.102	20.015	20.644	20.146	19.637	20.109	0.360
	T1	19.685	20.546	20.606	19.762	20.178	20.155	0.428
	T3	20.811	20.090	20.659	20.731	20.127	20.484	0.347
	T5	20.219	21.165	19.939	21.095	20.234	20.530	0.560
Lighting conditions	**Regular**	21.266	19.713	21.310	21.179	20.084	20.710	0.754
	Low	19.740	19.678	20.151	19.023	19.001	19.519	0.497
	Sunny	18.939	19.097	19.023	19.074	19.158	19.058	0.082
Number of classes	**4 classes**	19.773	20.115	19.185	20.097	20.208	19.876	0.420
	10 classes	19.423	19.618	19.637	19.599	19.556	19.567	0.086
	15 classes	20.119	19.460	19.157	20.109	19.713	19.712	0.417
	20 classes	20.109	19.668	19.554	19.606	19.653	19.718	0.223
Distance	**70 cm**	19.072	19.616	19.773	19.182	19.107	19.350	0.322
	110 cm	20.755	20.238	20.399	19.749	20.075	20.243	0.374
	160 cm	20.673	21.125	19.659	20.274	19.791	20.304	0.610
	200 cm	20.170	19.639	20.559	20.035	20.331	20.147	0.345
Image size	**320x320**	8.637	8.465	8.809	8.645	8.155	8.542	0.248
	480x480	12.769	12.634	13.724	12.583	12.649	12.872	0.481
	640x640	20.058	20.193	20.159	21.082	20.700	20.438	0.438
	1280x1280	68.187	69.268	67.717	67.846	68.275	68.259	0.610
Model size	**Tiny**	21.957	21.300	21.704	21.400	21.246	21.521	0.301
	Small	20.168	20.260	20.657	21.138	20.714	20.587	0.390
	Mid	20.276	20.657	20.608	20.754	21.136	20.686	0.309

Table 7.4: Table shows the summary of all trials for each criteria by providing the average cycle time along with the standard deviation. All the values in the table are in seconds.

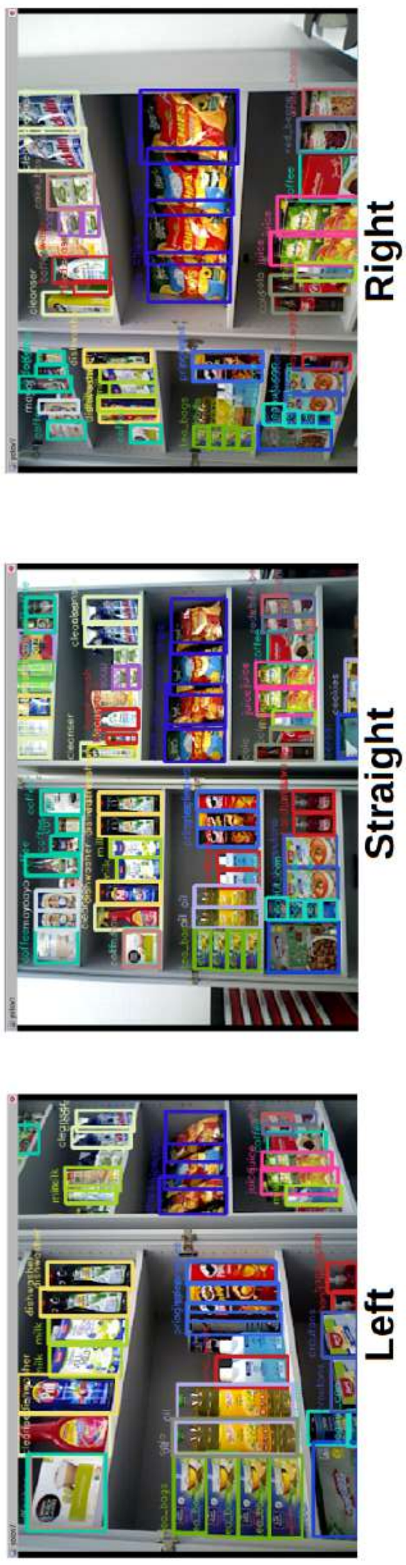


a) Number of classes

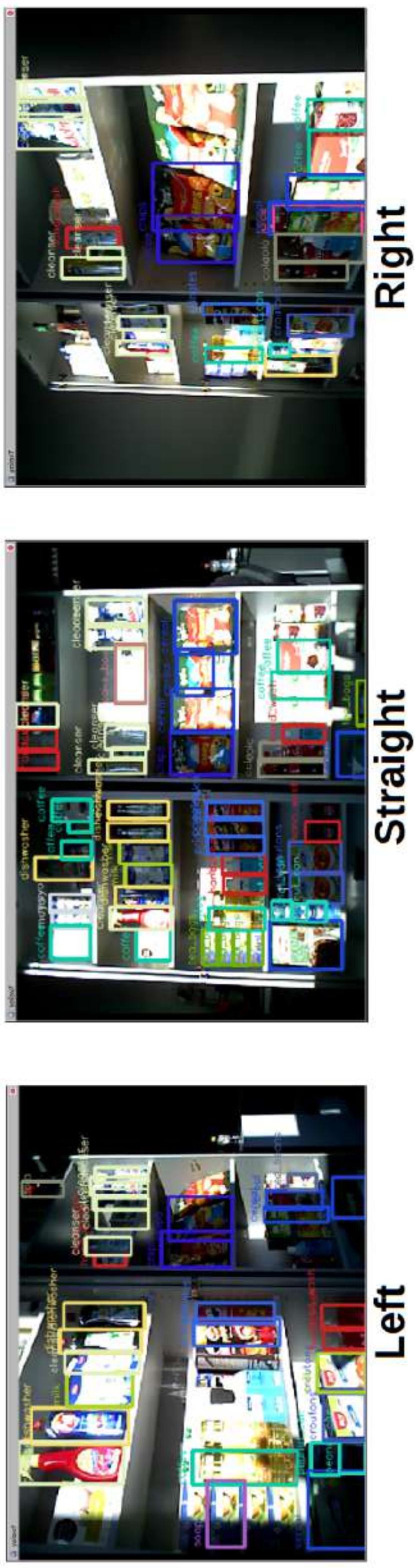


b) Blur

Figure 7.12: Sample detections using YOLOv7 for investigating the effect of the parameters: Number of classes and blur.

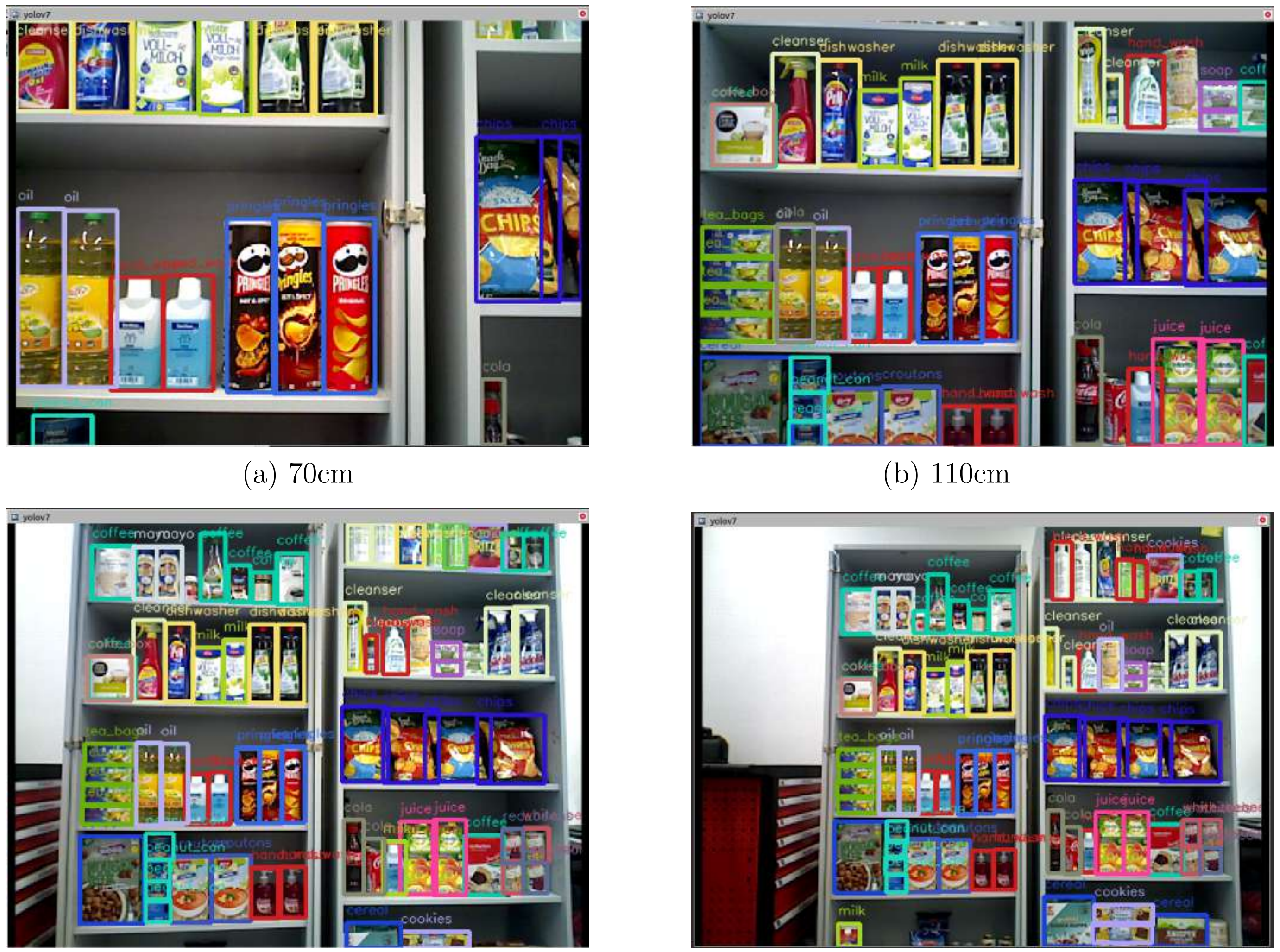


a) Viewing angle



b) Lighting conditions (sunny day)

Figure 7.13: Sample detections using YOLOv7 for investigating the effect of the parameters: Viewing angle and lighting conditions.



(a) 70cm

(b) 110cm

(c) 160cm

(d) 200cm

Figure 7.14: Sample detections using YOLOv7 for investigating the effect of distance. The distance is measured from grocery shelf to the robot's camera.

7.5 Overall Summary

The section provides the overall summary considering all four experiments. We also provide some observations and discuss other relevant points.

- Considering the first three experiments, YOLOv5 detectors performed the best in achieving low cycle time and maintaining good accuracy. The average cycle time for each experiment is as follows: 3.8 seconds (experiment-1), 222.5 seconds (experiment-2, webcam), and 3.7 seconds (experiment-3). The accuracy value is 0.941 mAP@0.5.
- The NanoDet-Plus detector performed best in experiment 2 compared to the other detectors, with 246.76 seconds as the average total cycle time using the RealSense camera.
- The YOLOv7 model performed consistently over all four experiments regarding cycle time and accuracy.
- The DETR detector performed the least in the first three experiments in cycle time and accuracy compared to the other detectors.
- If we compare the average cycle time results from the first three experiments of any detector, we observe that the results in experiment 2 take way longer cycle time and do not even come close to the real-world use case. The latter case is because NVIDIA board's low memory availability and compute power. Here we highlight that even state-of-the-art detectors are not readily usable on such embedded platforms and must be tailored according to the use case.
- Experiments 1 and 2 show that each detector performs pre-processing, post-processing, and miscellaneous tasks. Some detectors performed well in some tasks, while some took more cycle time. One way to address the latter problem is to discard the unnecessary operations under each task and develop a unified implementation in terms of code. The latter suggestion can improve performance by reducing the total cycle time.
- From experiment 4's evaluation, we found that factors like image size and blur directly related to the cycle time, and the number of classes in a frame, viewing

angle, and distance showed no direct effect. The factors like model size and lighting conditions affected the cycle time when we attribute cycle time along with the measures like accuracy, computation load, and the number of detections.

- In experiment 4, we observed a different behavior when evaluating the parameter model size. One can expect that the cycle time increases with an increase in model size, but in our case, the tiny model had more cycle time than the small and medium. The point to highlight here is that other factors (computation load in our case) might affect the cycle time during the execution.
- From experiments 1 and 3, we observe that our trained models on our Small Groceries dataset perform well in terms of detection in the experimental scenario but fail to scale in the real world when presented with the same/similar object in a supermarket. In contrast, models trained on the SKU110k perform well under supermarket scenarios but with only one class. The latter discussion highlights the challenges of supervised learning using labeled data and the need of real world data.
- By considering experiment 3, we observe that detectors like YOLOv5, YOLOv7, and NanoDet-Plus perform well in terms of total cycle time on the robot. However, in scenarios, like sunny lighting conditions discussed in experiment 4, the detections are missed due to the high glare on the objects. In the latter scenario, bringing the human into the loop and considering a shared task execution model is helpful. The triggering of a request to the human can be set based on the perception cycle time deadline. If the cycle time crosses the deadline, the human can take over; else, automatic detection should be sufficient.

8 Conclusions

In this project, we focused on evaluating and comparing the real-time perception capabilities of robots in a supermarket scenario. The main objective was to analyze the perception cycle time and explore the feasibility of shared task execution between humans and robots. We selected YOLOv7, YOLOv5, NanoDet-Plus, and DETR, four different object detectors, for comparative analysis. The results of this project found that the YOLOv5 detector performed the best in achieving low cycle time and maintaining good accuracy. In a supermarket setting, a shorter cycle time can enable the robot to more quickly and accurately identify and locate products on shelves. Based on this, it is suggested that if shared task execution is implemented in a supermarket scenario, it is best done with the detectors like YOLOv5. In this project, we also demonstrated that lightweight models like NanoDet-Plus performed well in cycle time while maintaining reasonable accuracy. The lightweight models can be used when accuracy is not critical (e.g., perception manipulation loops) and later switch back to a well-performing model while detecting the objects on the shelves. In experiment 4, we show that image size and blur directly affect the cycle time. So it is the responsibility of the developer to choose the right image size while maintaining minimum blur.

In conclusion, the performance of a detector in a laboratory setting may not necessarily translate to the same performance in a real-world environment. In the context of the supermarket, more factors should be considered when selecting a detector, such as the ability of the detector to detect a wide range of objects, the ability to detect objects with different orientations, and the ability to detect objects with different scales.

8.1 Contributions

The contributions of this project are as follows:

- A comparison of modern object detectors in a supermarket scenario, in terms of their impact on the perception cycle time.
- We evaluated the effect of various conditions (such as lighting, viewing angle, distance, and blur) and parameters (such as image size, model size, and the number of classes) on the perception cycle time using the state-of-the-art YOLOv7 object detector.
- The creation and release of a custom dataset, called the "Small Groceries" dataset, specifically designed to mimic real-world supermarket scenarios. This dataset consisted of 317 images, 25 classes, and 6,361 annotations and was collected using three different cameras. We open-sourced and made the dataset publicly available..
- Two custom inference scripts that utilize the Robot Operating System (ROS) framework to interface with the DETR and NanoDet-Plus object detectors were not previously available as ROS nodes. Creating the custom script allowed more seamless experimentation of these detectors using the Toyota HSR platform.

8.2 Lessons Learned

- One of the important lessons learned while creating the dataset was the class imbalance problem and its impact on the model performance. Some classes, like pasta, and cake_box, need to be more represented in our dataset. We observed that categories like cleansers and coffee received more preference during localization than pasta and cake_box. The latter problem is termed a foreground-foreground class imbalance problem, and a more comprehensive study can be seen in recent a review by K. Oksuz et al. [86].
- Although our dataset had fewer images and class imbalances, we relied on data augmentation during the training process to overcome overfitting over similar representations. We utilized brightness, flip, and random crop augmentations during the training. We observe that data augmentation helped in model detection performance. For example, the YOLOv5 model could detect a reasonable amount of objects during high lighting conditions, as seen in experiment 4. This highlights the importance of

data augmentation, and more details can be found in a recent survey by S. Connor et al. [87].

- In our initial hypothesis, we believed that the NVIDIA Jetson board would perform better in detection speed. In our experiment 2, we found that even lightweight models like NanoDet-Plus had high model loading time and inference time. We expect the latter case to be due to using PyTorch-based models directly. Here we highlight that GPU-based inference is not directly transferable to the embedded boards, even using lightweight detectors like the NanoDet-plus.

- Experiment 4 showed that even using a CPU-only system like the one used on the Toyota HSR platform can achieve good performance in perception cycle time.

8.3 Future Work

- One possible extension of this project could be to compare the perception cycle time on different deployment methods, such as TensorRT for NVIDIA-based GPU and OpenVINO for CPU-based deployment. This would provide insight into how these methods impact the efficiency of the robot's perception pipeline.

- Another extension could be to consider additional tasks beyond perception, such as navigation and manipulation, which are also important in a real-world scenario. This could involve exploring the cycle times for these tasks and how they may impact the overall performance of the robot.

- Conducting the same project on a real industrial robot would also be a valuable extension, as it may introduce additional parameters to consider and could potentially affect the perception cycle time. This would provide a more realistic representation of how the models would perform in a real-world setting.

- Finally, exploring other paradigms beyond supervised learning could also be a valuable extension of this project. This could include unsupervised, semi-supervised, or reinforcement learning approaches, which could potentially offer additional benefits or challenges in the context of a supermarket scenario.

A

A.1 Training Details

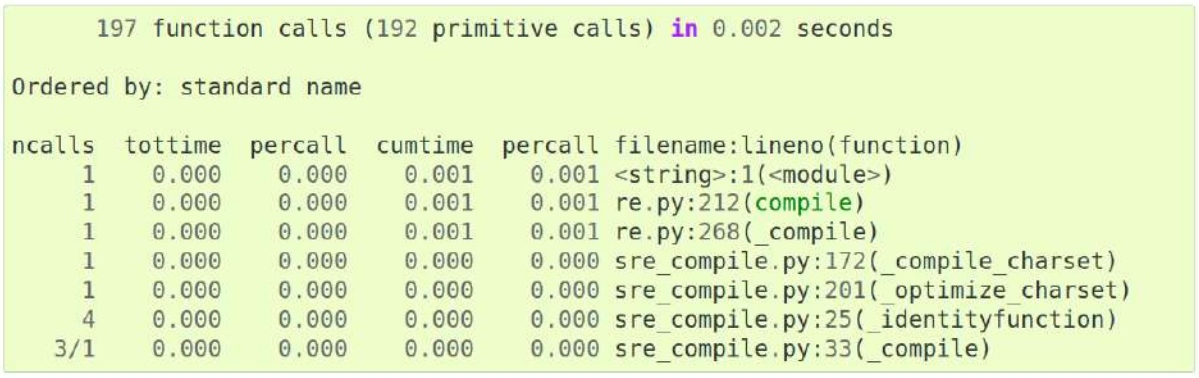
The training details are given in Table A.1. As a common note for all the detectors, we utilize flip, random crop, and brightness-based data augmentations during the training process.

Detector	YOLOv5	YOLOv7	NanoDet-Plus	DETR
Datasets	SKU110k, Small Groceries	SKU110k, Small Groceries	SKU110k, Small Groceries	SKU110k, Small Groceries
Pre-Trained model	YOLOv5 nano	YOLOv7-tiny YOLOv7 YOLOv7-X	NanoDet-Plus-m	DETR- basic
Backbone	CSP-Darknet53	E-ELAN	ShuffleNetV2	ResNet50
Neck	SPP, PANet	CSPSPP+(ELAN, E-ELAN)PAN	GhostPAN	Transformer
Head	YOLOv3 Head	YOLOR Head	NanoDetPlusHead	FFN
Epochs	50	50	50	100*
Batch Size	32	32	32	4
Activation Function	SiLU	SiLU	LeakyReLU	ReLU
Optimizer	SGD	SGD	AdamW	AdamW
Learning Rate	0.01	0.01	0.001	0.0001
Learning Rate Scheduler	CosineAnnealingLR	LambdaLR	CosineAnnealingLR	StepLR
Weight Decay	0.0005	0.0005	0.05	0.0001
Momentum	0.937	0.937	NA	NA
Loss Function	GIoU	CIoU	Generalized Focal Loss	Bipartite Matching Loss

Table A.1: The table shows the summary of training details. Note[*]: We used more number of epoch while training the DETR model as the accuracy was too low with 50 epochs.

A.2 Understanding cProfile Trace

This section helps the readers to understand cProfile trace. All the content in this section is taken from Python documentation [84]; none of it is the author's work. The sample output from the cProflile is given in Figure A.1. All the values are given in seconds.


```
      197 function calls (192 primitive calls) in 0.002 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    0.001    0.001 <string>:1(<module>)
     1    0.000    0.000    0.001    0.001 re.py:212(compile)
     1    0.000    0.000    0.001    0.001 re.py:268(_compile)
     1    0.000    0.000    0.000    0.000 sre_compile.py:172(_compile_charset)
     1    0.000    0.000    0.000    0.000 sre_compile.py:201(_optimize_charset)
     4    0.000    0.000    0.000    0.000 sre_compile.py:25(_identityfunction)
   3/1    0.000    0.000    0.000    0.000 sre_compile.py:33(_compile)
```

Figure A.1: cProfile's sample output
Note: Image taken from Python documentation [84]

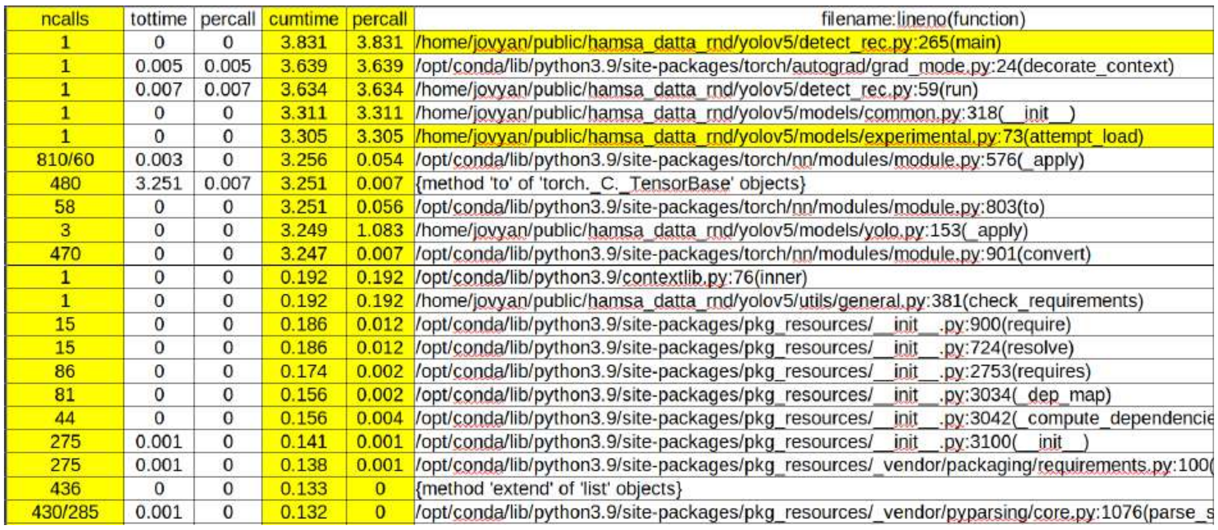
"The first line indicates that 197 calls were monitored. Of those calls, 192 were primitive, meaning that the call was not induced via recursion. The next line: **Ordered by: standard name**, indicates that the text string in the far right column was used to sort the output. The column headings include:

- ncalls : for the number of calls.
- tottime: for the total time spent in the given function (and excluding time made in calls to sub-functions).
- percall: is the quotient of tottime divided by ncalls
- cumtime: is the cumulative time spent in this and all subfunctions (from invocation till exit). This figure is accurate even for recursive functions.
- percall is the quotient of cumtime divided by primitive calls
- filename:lineno(function): provides the respective data of each function.

When there are two numbers in the first column (for example **3/1**), it means that the function recursed. The second value is the number of primitive calls and the former is the total number of calls. Note that when the function does not recurse, these two values are the same, and only the single figure is printed." [84]

A.3 Sample Trace Files

Refer Figure A.2



ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0	0	3.831	3.831	/home/jovyan/public/hamsa_datta_rnd/yolov5/detect_rec.py:265(main)
1	0.005	0.005	3.639	3.639	/opt/conda/lib/python3.9/site-packages/torch/autograd/grad_mode.py:24(decorate_context)
1	0.007	0.007	3.634	3.634	/home/jovyan/public/hamsa_datta_rnd/yolov5/detect_rec.py:59(run)
1	0	0	3.311	3.311	/home/jovyan/public/hamsa_datta_rnd/yolov5/models/common.py:318(__init__)
1	0	0	3.305	3.305	/home/jovyan/public/hamsa_datta_rnd/yolov5/models/experimental.py:73(attempt_load)
810/60	0.003	0	3.256	0.054	/opt/conda/lib/python3.9/site-packages/torch/nn/modules/module.py:576(_apply)
480	3.251	0.007	3.251	0.007	{method 'to' of 'torch._C._TensorBase' objects}
58	0	0	3.251	0.056	/opt/conda/lib/python3.9/site-packages/torch/nn/modules/module.py:803(to)
3	0	0	3.249	1.083	/home/jovyan/public/hamsa_datta_rnd/yolov5/models/yolo.py:153(_apply)
470	0	0	3.247	0.007	/opt/conda/lib/python3.9/site-packages/torch/nn/modules/module.py:901(convert)
1	0	0	0.192	0.192	/opt/conda/lib/python3.9/contextlib.py:76(inner)
1	0	0	0.192	0.192	/home/jovyan/public/hamsa_datta_rnd/yolov5/utils/general.py:381(check_requirements)
15	0	0	0.186	0.012	/opt/conda/lib/python3.9/site-packages/pkg_resources/__init__.py:900(require)
15	0	0	0.186	0.012	/opt/conda/lib/python3.9/site-packages/pkg_resources/__init__.py:724(resolve)
86	0	0	0.174	0.002	/opt/conda/lib/python3.9/site-packages/pkg_resources/__init__.py:2753(requires)
81	0	0	0.156	0.002	/opt/conda/lib/python3.9/site-packages/pkg_resources/__init__.py:3034(_dep_map)
44	0	0	0.156	0.004	/opt/conda/lib/python3.9/site-packages/pkg_resources/__init__.py:3042(_compute_dependencie
275	0.001	0	0.141	0.001	/opt/conda/lib/python3.9/site-packages/pkg_resources/__init__.py:3100(__init__)
275	0.001	0	0.138	0.001	/opt/conda/lib/python3.9/site-packages/pkg_resources/_vendor/packaging/requirements.py:100(
436	0	0	0.133	0	{method 'extend' of 'list' objects}
430/285	0.001	0	0.132	0	/opt/conda/lib/python3.9/site-packages/pkg_resources/_vendor/pyparsing/core.py:1076(parse_s

Figure A.2: Sample trace file.

A.4 Other Datasets Image

Refer Figure A.3

A.5 Custom Dataset Image

Refer Figure A.4

A.6 Additional Results from Experiments

A.6.1 Experiment-1

Refer from Figure A.5 to Figure A.8

A.6.2 Experiment-2

Refer from Figure A.9 to Figure A.14

A.7 Annotation Tool

Refer Figure A.15

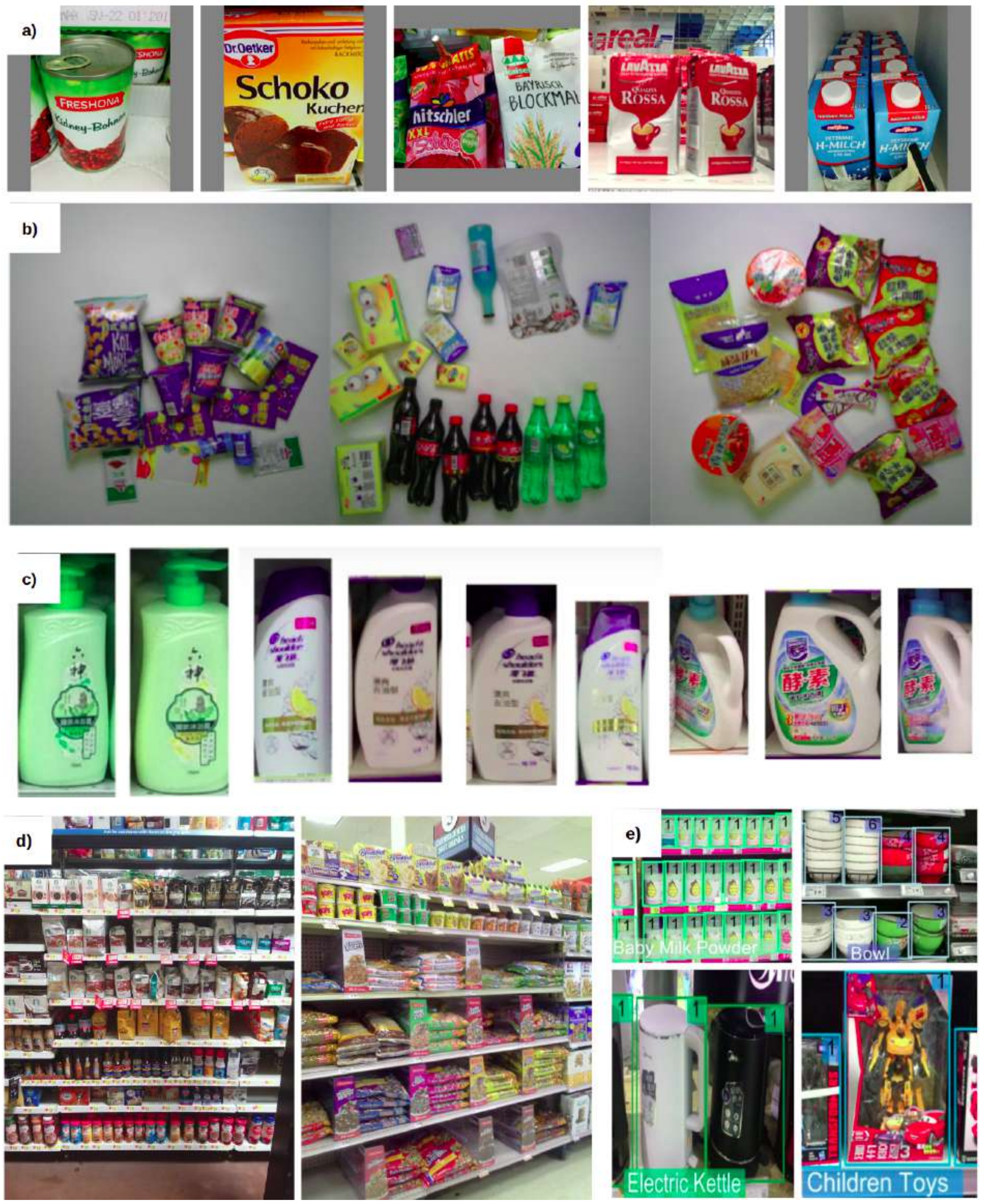


Figure A.3: Sample images from the datasets a)Freiburg groceries, b)RPC, c)RP2K, d)SKU110k, and e)Locount. Note: All the images taken from [61], [62], [64], [46], [47] respectively.

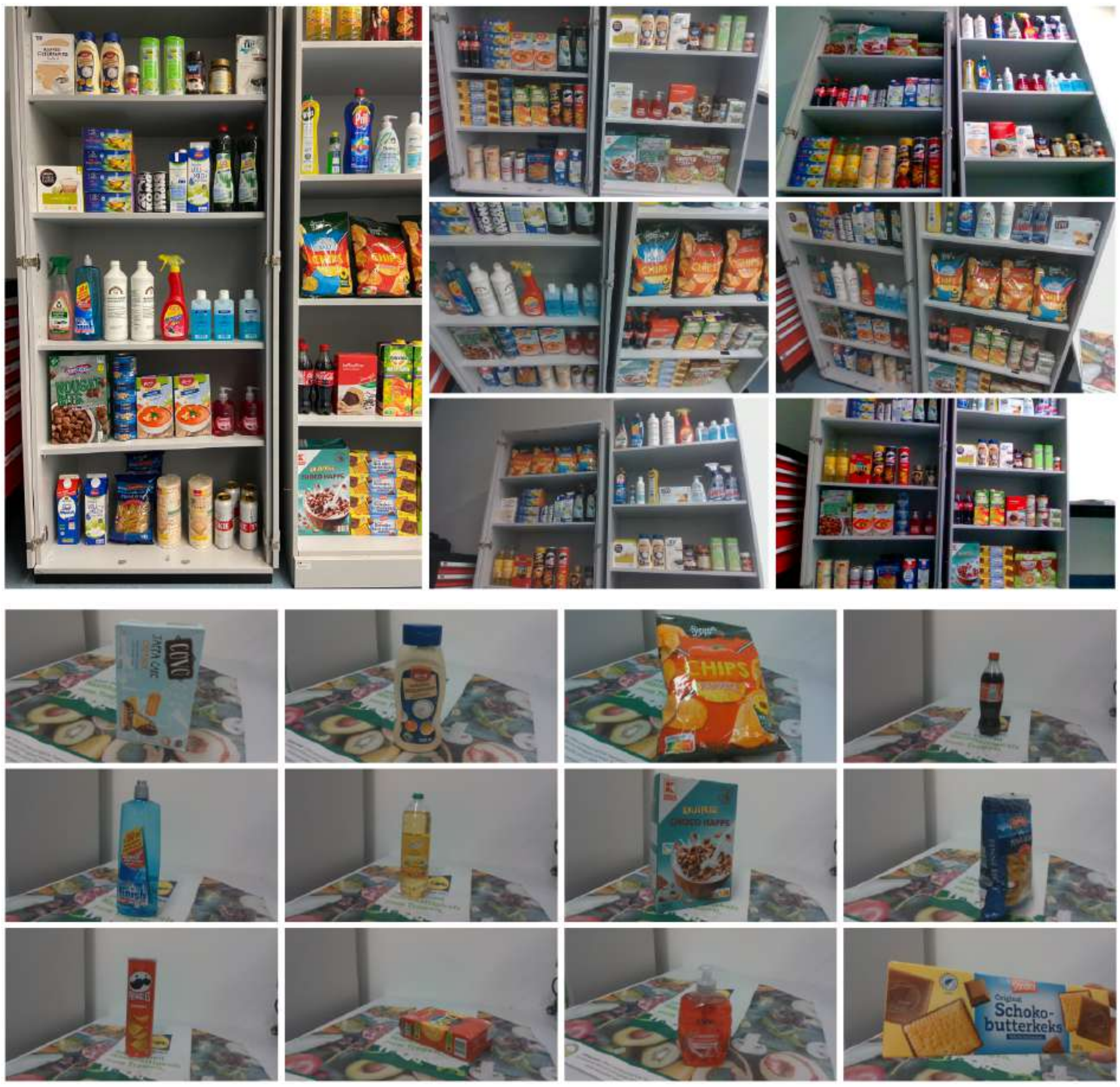


Figure A.4: Sample images from Small Groceries dataset. Top images corresponds to set1 with objects placed in shelves, and bottom images corresponds to set2 with single object in an image.

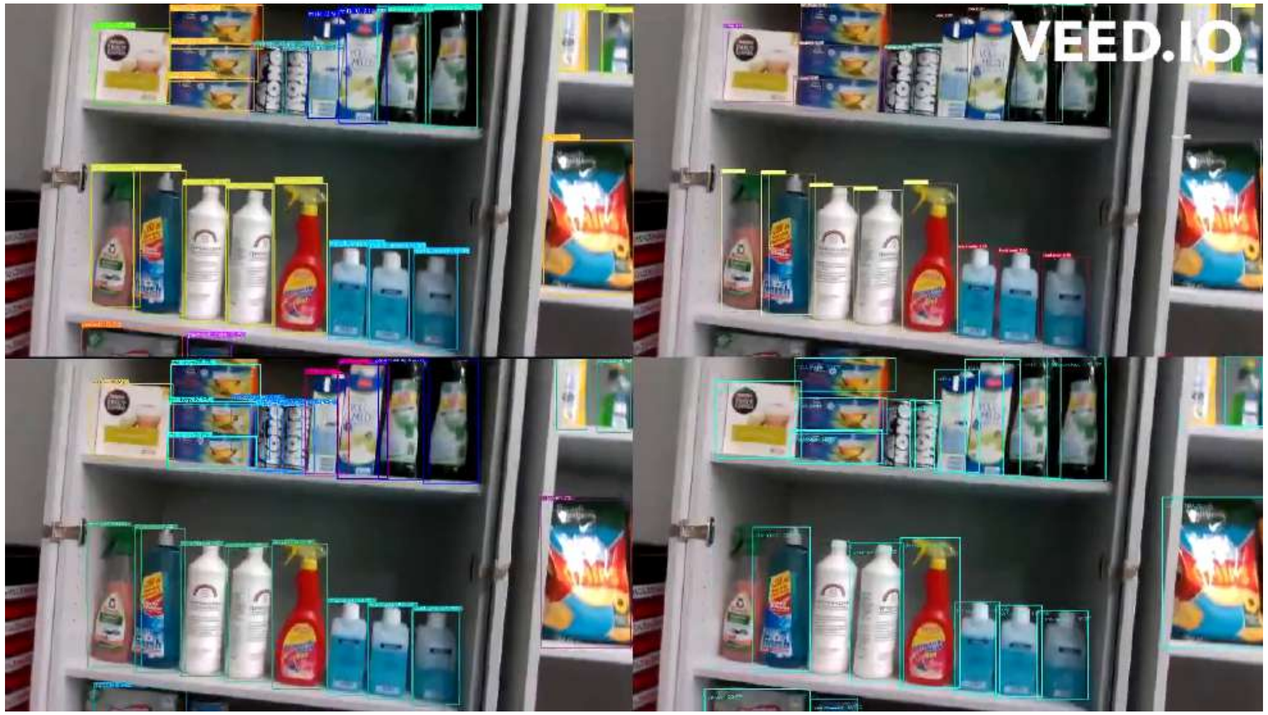


Figure A.5: Sample visualizations of all four detectors on the custom setup video using the model trained on the custom dataset.

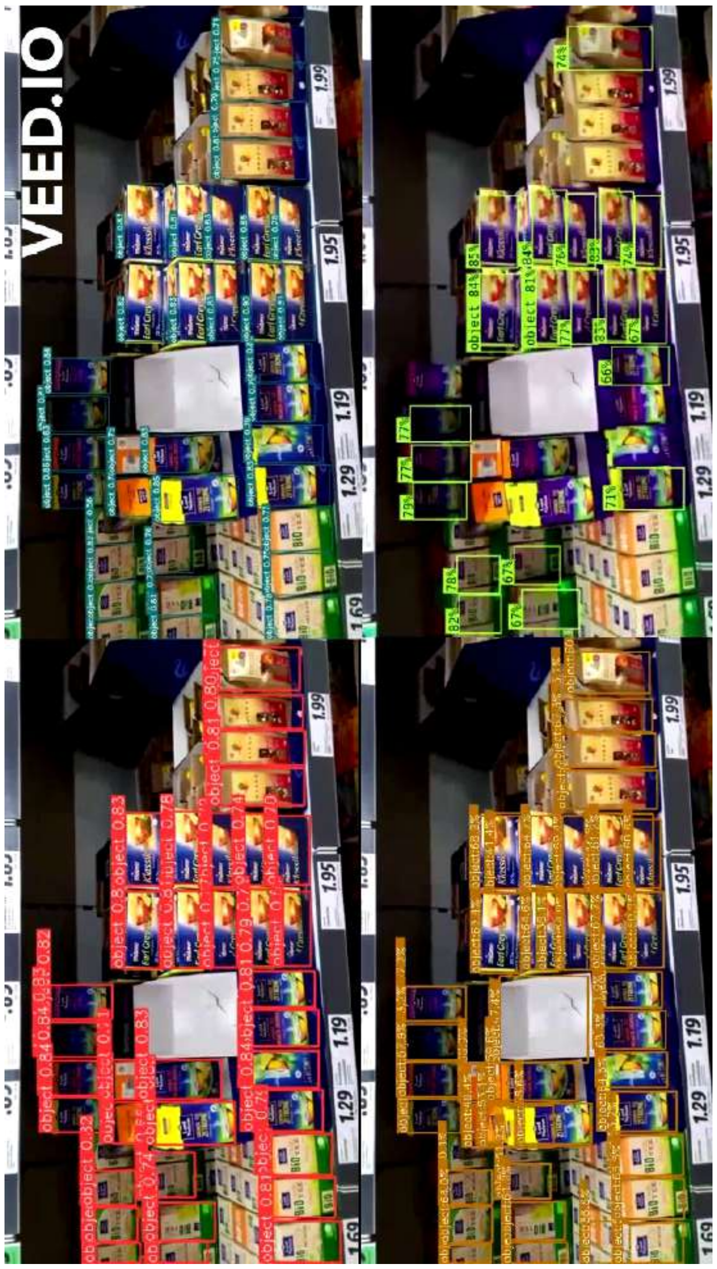


Figure A.6: Sample visualizations of all four detectors on the real-world supermarket video using the model trained on the SKU110k dataset.

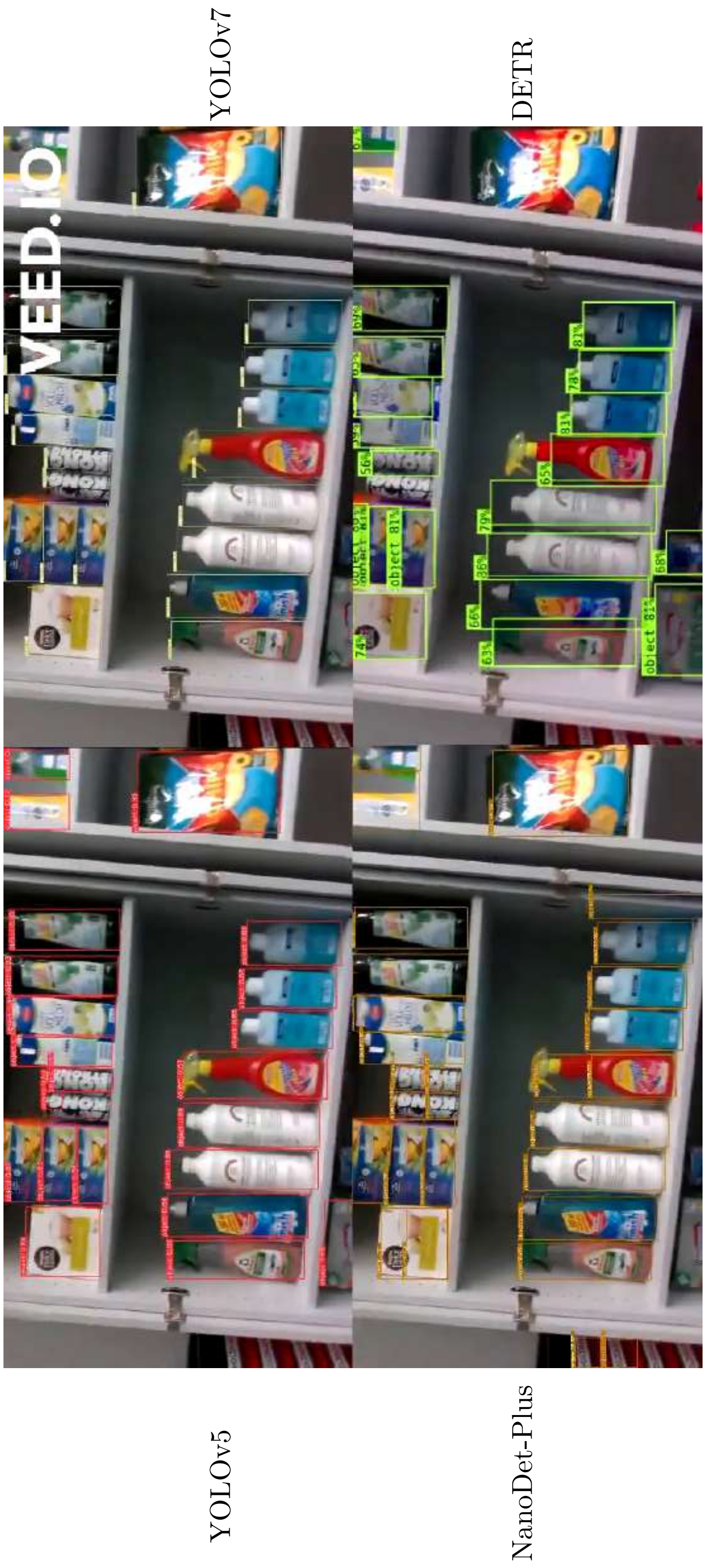


Figure A.7: Sample visualizations of all four detectors on the custom setup video using the model trained on the SKU110k dataset.

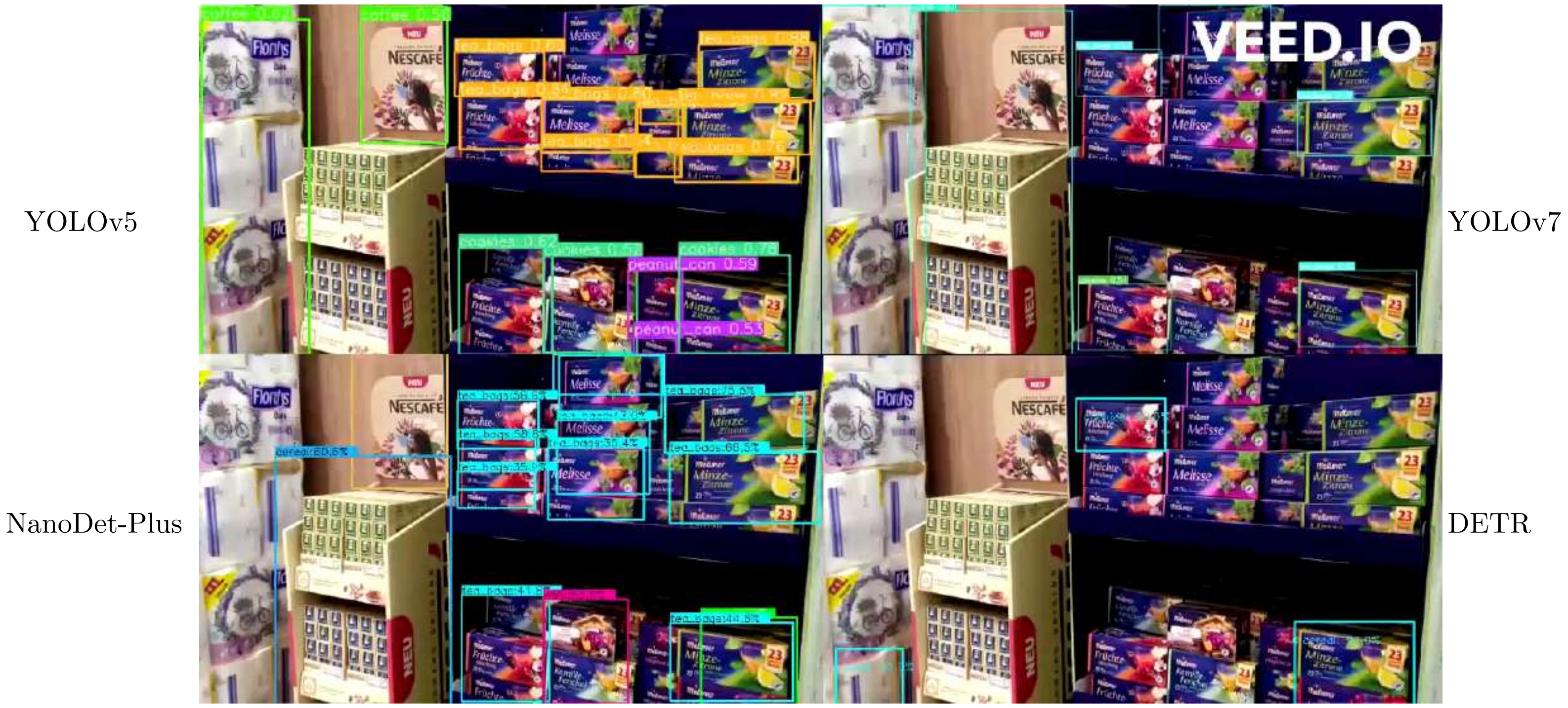
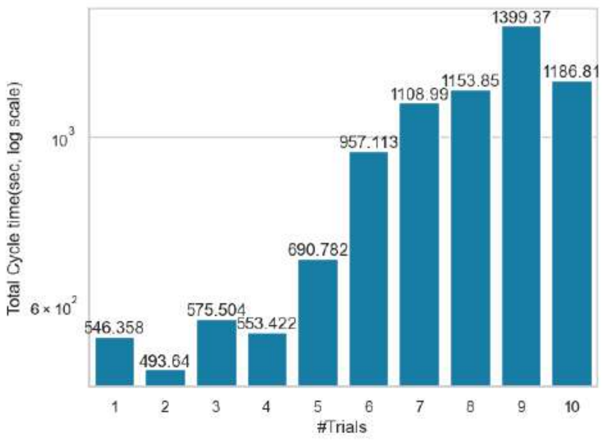
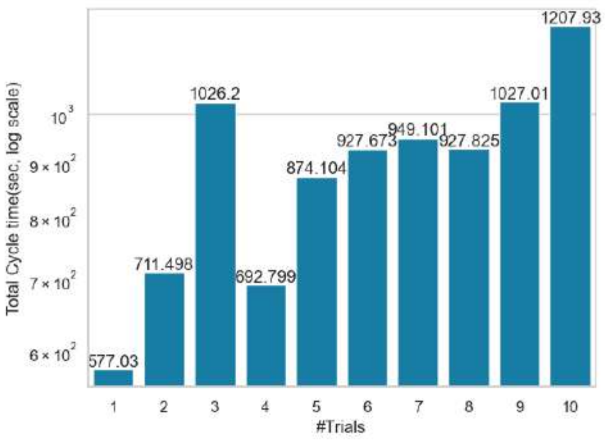


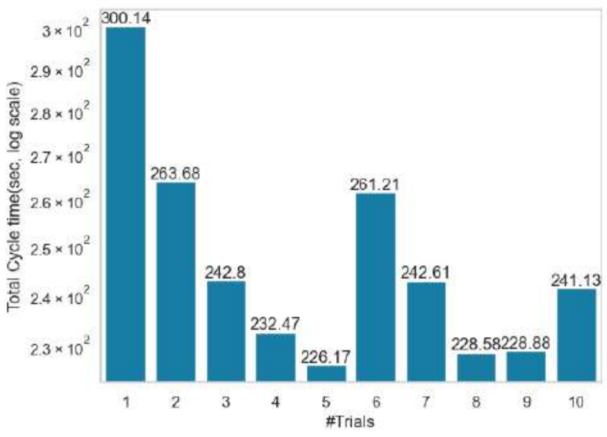
Figure A.8: Sample visualizations of all four detectors on the real-world supermarket video using the model trained on the custom dataset.



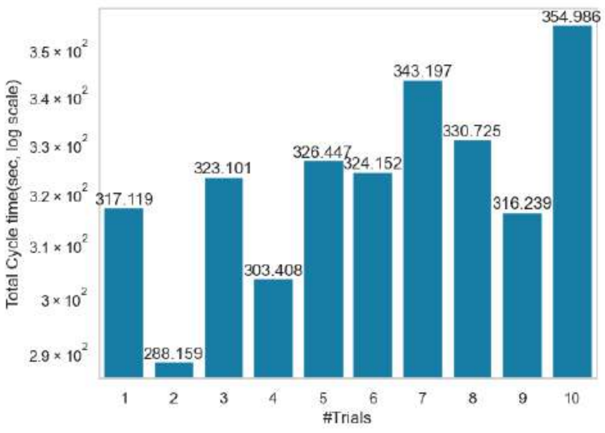
(a) DETR(RS)



(b) DETR(WC)

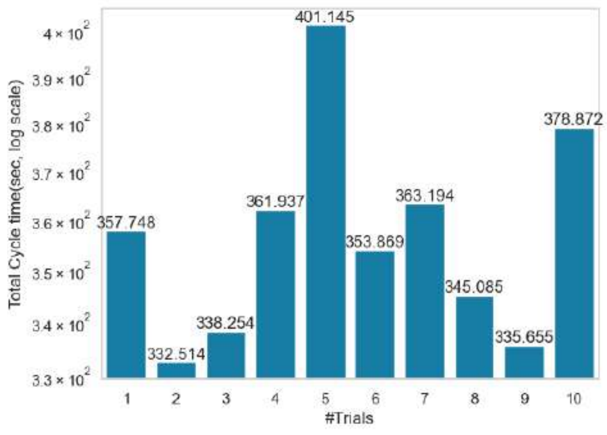


(c) NanoDet-Plus(RS)

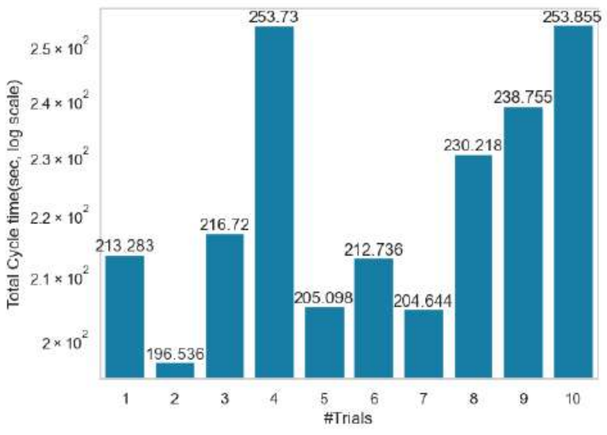


(d) NanoDet-Plus(WC)

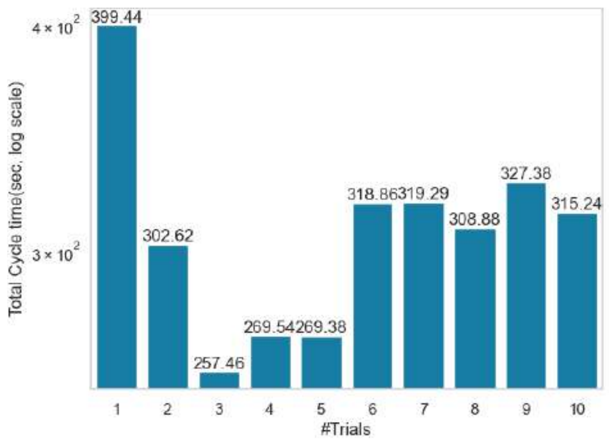
Figure A.9: Bar plot showing the total cycle time for each trial considering both cameras for DETR and NanoDet-Plus.The x-axis represents the total number of trials, and the y-axis represents the total cycle time for each trial in the log scale with its corresponding value in seconds.



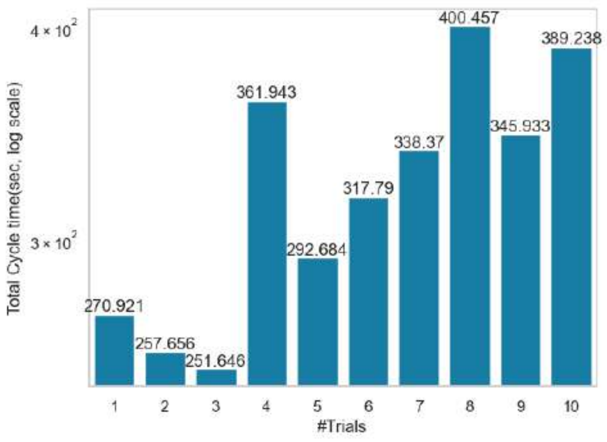
(a) YOLOv5(RS)



(b) YOLOv5(WC)

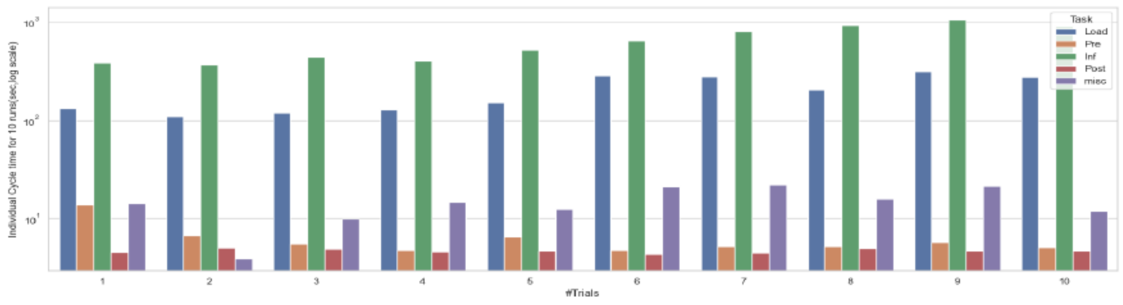


(c) YOLOv7(RS)

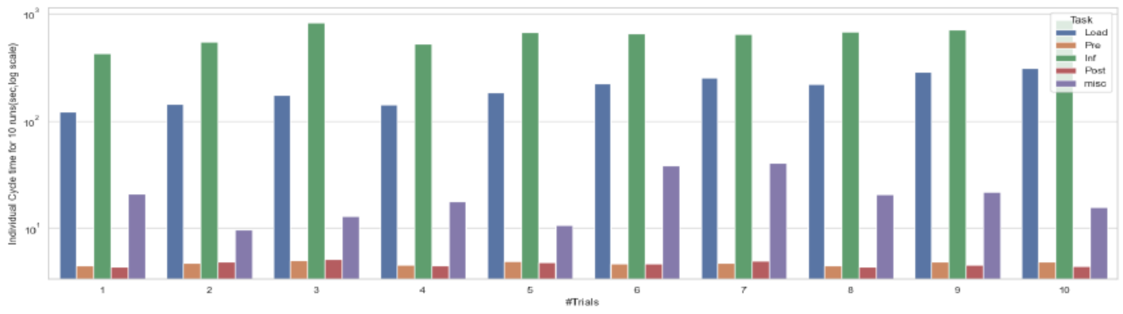


(d) YOLOv7(WC)

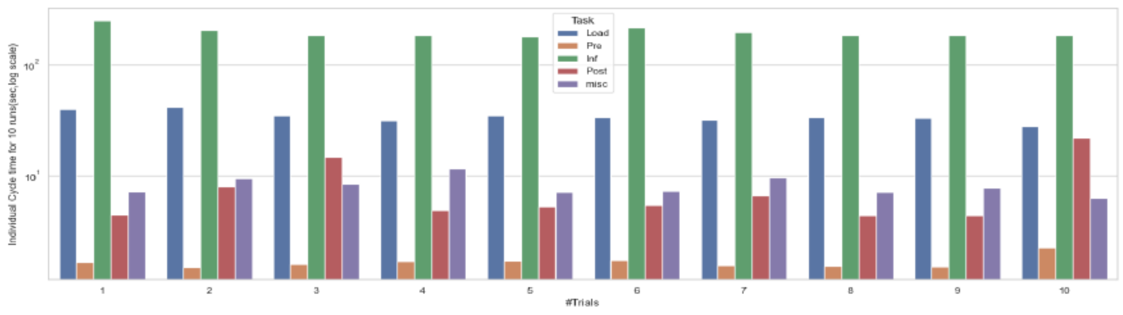
Figure A.10: Bar plot showing the total cycle time for each trial considering both cameras for YOLOv5 and YOLOv7.The x-axis represents the total number of trials, and the y-axis represents the total cycle time for each trial in the log scale with its corresponding value in seconds.



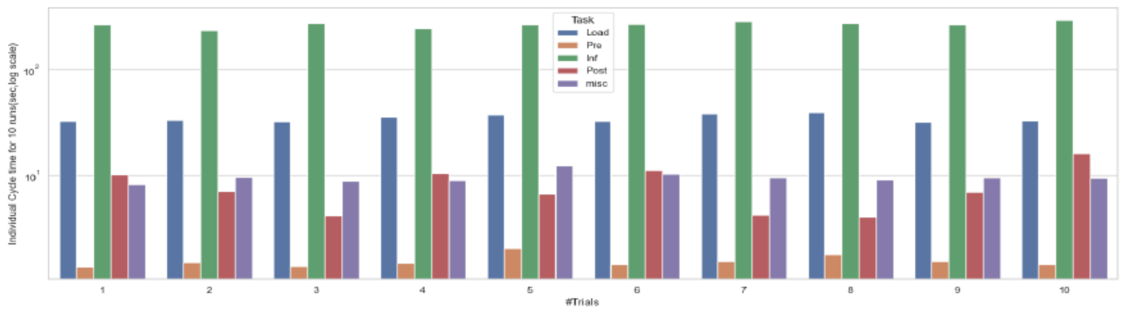
a)DETR(RS)



b)DETR(WC)

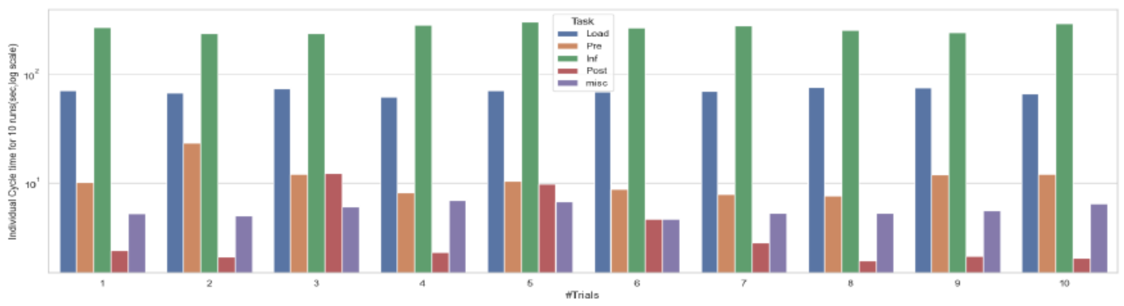


c)NanoDet-Plus(RS)

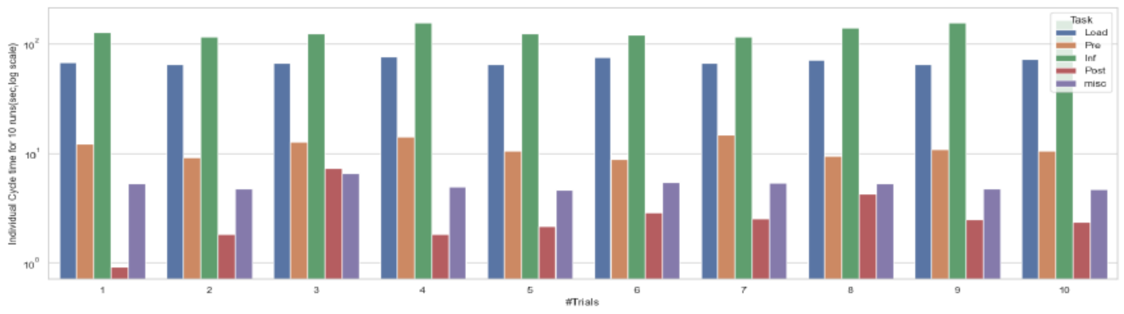


d)NanoDet-Plus(WC)

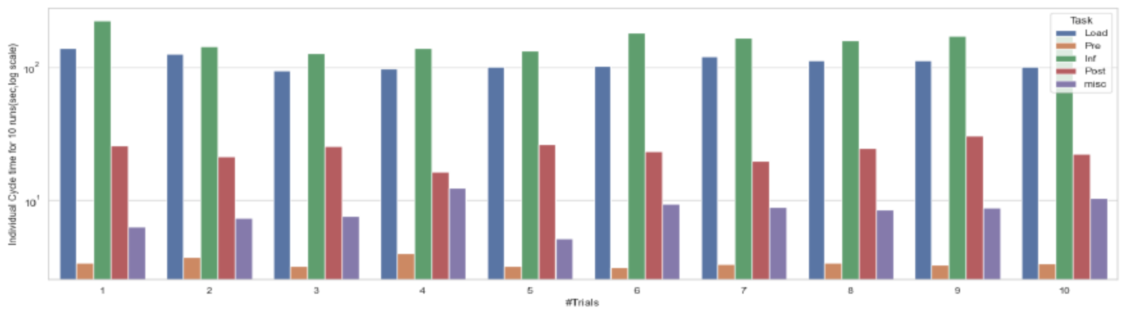
Figure A.11: The bar plots shows the individual cycle time for each task considering both cameras for DETR and NanoDet-Plus in the log scale.



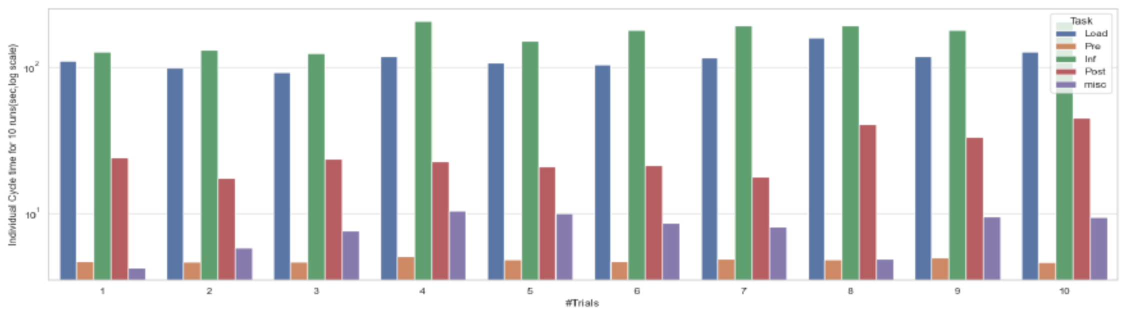
a)YOLOv5(RS)



b)YOLOv5(WC)

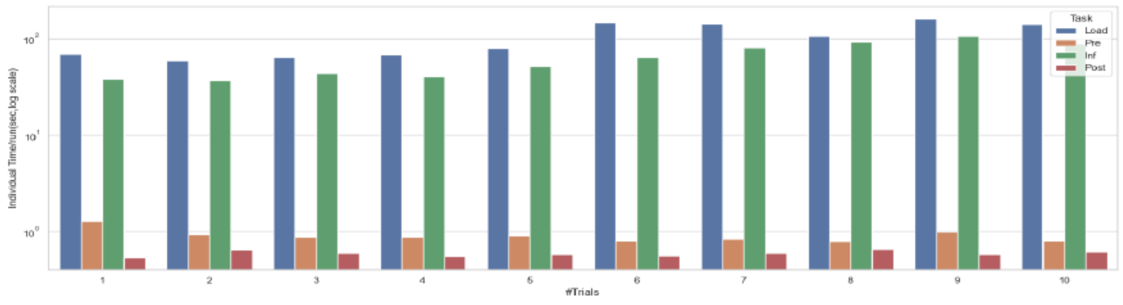


c)YOLOv7(RS)

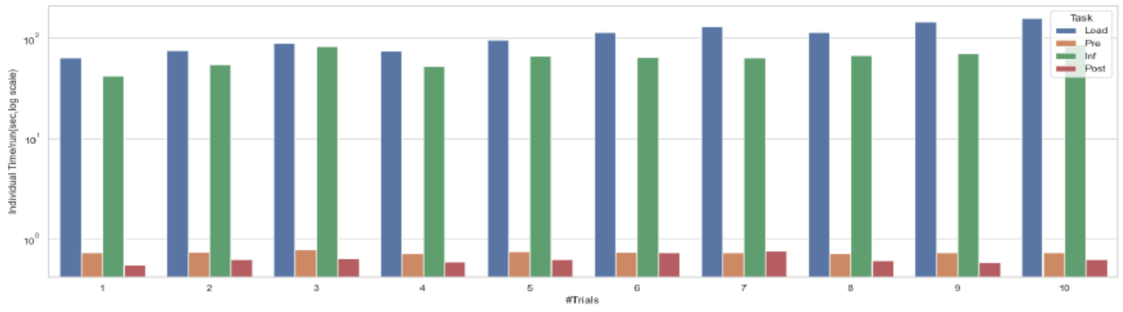


d)YOLOv7(WC)

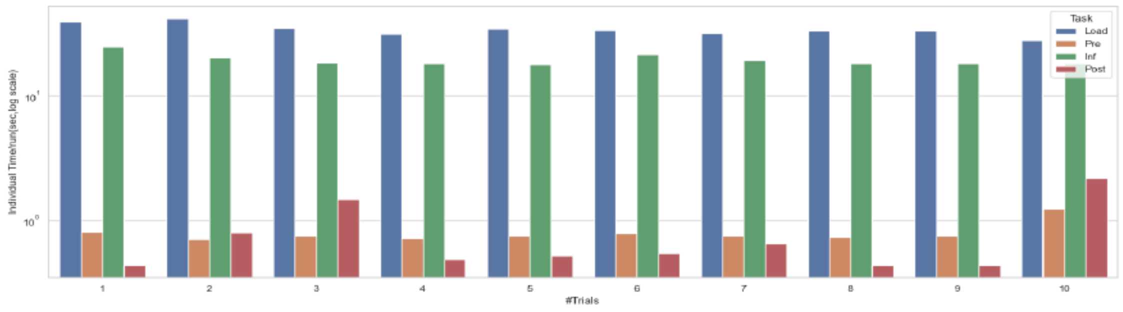
Figure A.12: The bar plots shows the individual cycle time for each task considering both cameras for YOLOv5 and YOLOv7 in the log scale.



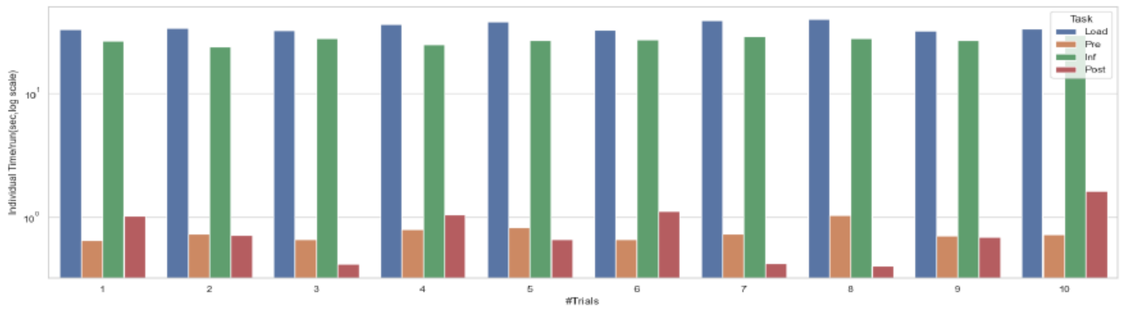
a)DETR(RS)



b)DETR(WC)

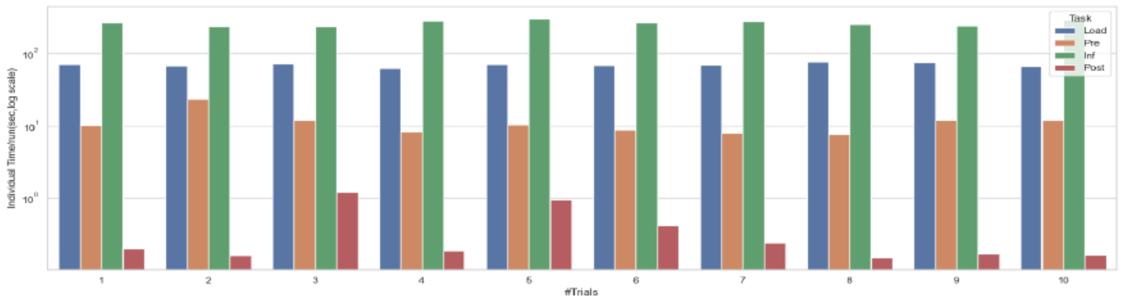


c)NanoDet-Plus(RS)

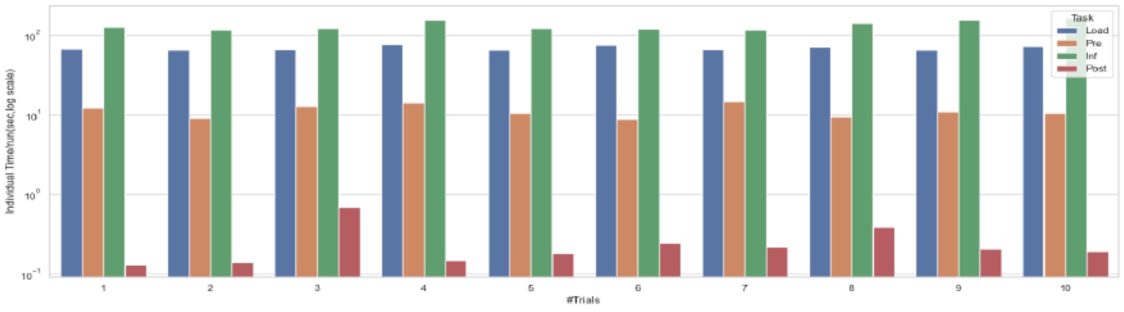


d)NanoDet-Plus(WC)

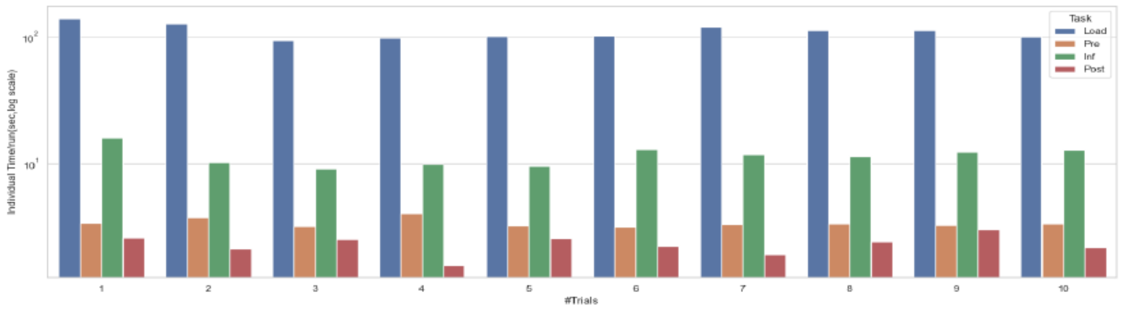
Figure A.13: The bar plots shows the individual time per run for each task considering both cameras for DETR and NanoDet-Plus in the log scale.



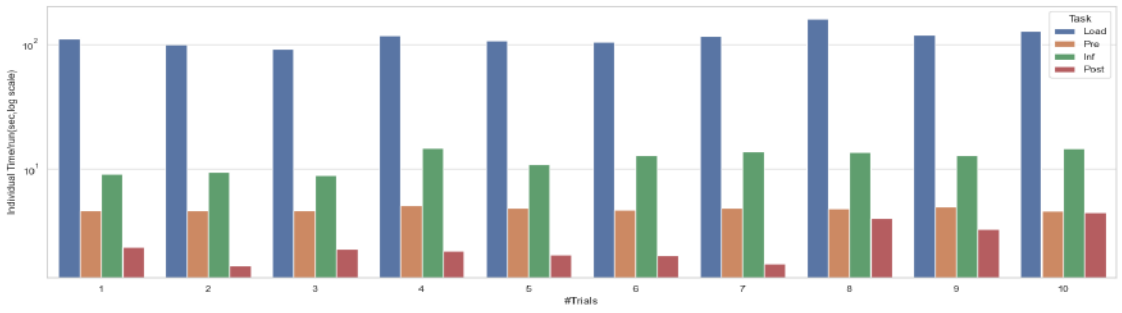
a)YOLOv5(RS)



b)YOLOv5(WC)

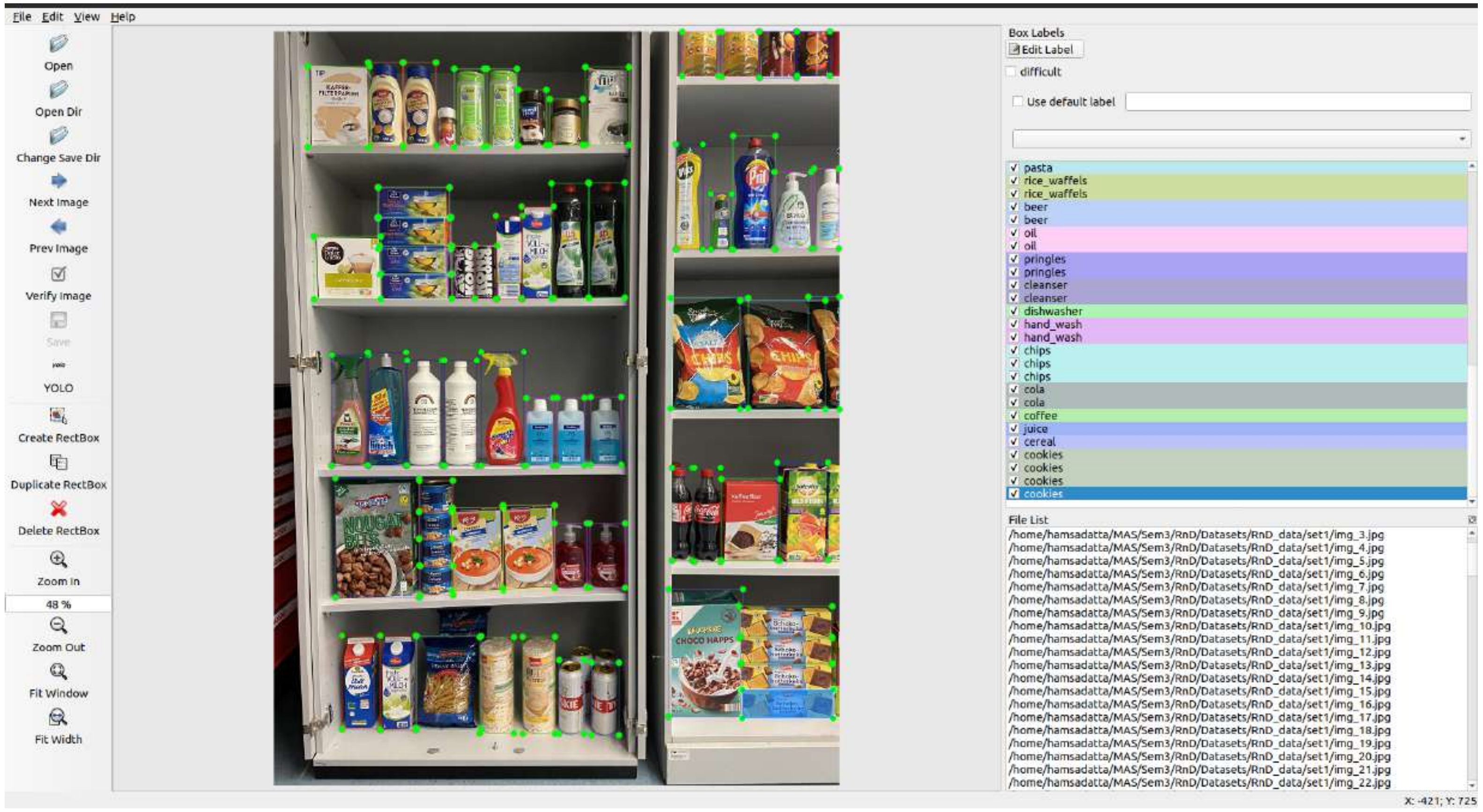


c)YOLOv7(RS)



d)YOLOv7(WC)

Figure A.14: The bar plots shows the individual time per run for each task considering both cameras for YOLOv5 and YOLOv7 in the log scale.



```
✓ pasta
✓ rice_waffels
✓ rice_waffels
✓ beer
✓ beer
✓ oil
✓ oil
✓ pringles
✓ pringles
✓ cleanser
✓ cleanser
✓ dishwasher
✓ hand_wash
✓ hand_wash
✓ chips
✓ chips
✓ chips
✓ cola
✓ cola
✓ coffee
✓ juice
✓ cereal
✓ cookies
✓ cookies
✓ cookies
✓ cookies
```

```
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_3.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_4.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_5.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_6.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_7.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_8.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_9.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_10.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_11.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_12.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_13.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_14.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_15.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_16.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_17.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_18.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_19.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_20.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_21.jpg
/home/hamsadatta/MAS/Sem3/RnD/Datasets/RnD_data/set1/img_22.jpg
```

```
X: -421; Y: 725
```

Figure A.15: labelImg annotation canvas

References

[1] M. Costanzo, G. De Maria, G. Lettera, and C. Natale, “Can robots refill a supermarket shelf?: Motion planning and grasp control,” *IEEE Robotics & Automation Magazine*, vol. 28, no. 2, pp. 61–73, 2021.

[2] B. Pearson, “Robots Are Taking The Store At Kroger, Walmart And Whole Foods. What Could Go Wrong?” https://www.forbes.com/sites/bryanpearson/2019/07/29/robots-are-taking-the-store-at-kroger-walmart-and-whole-foods\-what-could-go-wrong/?sh=2bd88a1c1ca9, accessed: 2022-05-10.

[3] S. Smith, “AI Spending by Retailers to Reach $12 Billion by 2023, Driven by the Promise of Improved Margins,” https://www.juniperresearch.com/press/ai-spending-by-retailers-reach-12-billion-2023, accessed: 2022-05-10.

[4] Y. Wei, S. Tran, S. Xu, B. Kang, and M. Springer, “Deep learning for retail product recognition: Challenges and techniques,” *Computational Intelligence and Neuroscience*, vol. 2020, 2020.

[5] B. Santra and D. P. Mukherjee, “A comprehensive survey on computer vision based approaches for automatic identification of products in retail store,” *Image and Vision Computing*, vol. 86, pp. 45–63, 2019.

[6] E. Arani, S. Gowda, R. Mukherjee, O. Magdy, S. Kathiresan, and B. Zonooz, “A comprehensive study of real-time object detection networks across multiple domains: A survey,” *arXiv preprint arXiv:2208.10895*, 2022.

[7] S. Halliday, “John Lewis says bedtime shopping habit surges,” https://ww.fashionnetwork.com/news/John-lewis-says-bedtime-shopping-habit-surges,1135304.html#fashion-week-milan-msgm, accessed: 2022-05-11.

[8] M.-A. Russon, “Why are people shopping online late at night?” https://www.bbc.com/news/business-49633006, accessed: 2022-05-11.

[9] A. Strand, “Sleepy Shoppers: What 1,000 People Reveal About Their Late-Night Purchase Habits,” https://eachnight.com/sleep/sleepy-shoppers-what-1000-people-reveal-about-their-late-night-purchase-habits/#, accessed: 2022-06-18.

[10] M. Franzino, “The $8.5 Trillion Talent Shortage,” https://www.toolsgroup.com/blog/retail-labor-shortage/#:~:text=Forecasts%20by%20Korn%20Ferry%20cited,for%20the%20industry%20to%20thrive., accessed: 2022-07-21.

[11] Deloitte, “2022 retail industry outlook,” https://www2.deloitte.com/content/dam/Deloitte/us/Documents/consumer-business/2022-retail-industry-outlook.pdf, accessed: 2022-07-21.

[12] Z. Zou, Z. Shi, Y. Guo, and J. Ye, “Object detection in 20 years: A survey,” *arXiv preprint arXiv:1905.05055*, 2019.

[13] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, “Transformers in vision: A survey,” *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[14] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A survey of modern deep learning based object detection models,” *Digital Signal Processing*, p. 103514, 2022.

[15] A. C. Hernández, C. Gómez, J. Crespo, and R. Barber, “Object detection applied to indoor environments for mobile robot navigation,” *Sensors*, vol. 16, no. 8, p. 1180, 2016.

[16] P. Mittal, R. Singh, and A. Sharma, “Deep learning-based object detection in low-altitude uav datasets: A survey,” *Image and Vision Computing*, vol. 104, p. 104046, 2020.

[17] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the IEEE computer society Conference on Computer Vision and Pattern Recognition. CVPR*, vol. 1. IEEE, 2001, pp. I–I.

[18] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 1. IEEE, 2005, pp. 886–893.

[19] P. Felzenszwalb, D. McAllester, and D. Ramanan, “A discriminatively trained, multiscale, deformable part model,” in *Conference on Computer Vision and Pattern Recognition.* IEEE, 2008, pp. 1–8.

[20] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *European Conference on Computer Vision.* Springer, 2016, pp. 21–37.

[23] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-end object detection with transformers,” in *European Conference on Computer Vision.* Springer, 2020, pp. 213–229.

[24] S. H. Tsang, “Review-DETR: End-to-End Object Detection with Transformers,” https://sh-tsang.medium.com/review-detr-end-to-end-object-detection-with-transformers-c64977be4b8e, accessed: 2022-012-23.

[25] D. Shah, “Revolutionary Object Detection Algorithm from Facebook AI,” https://medium.com/visionwizard/detr-b677c7016a47, accessed: 2022-012-23.

[26] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, “Deformable detr: Deformable transformers for end-to-end object detection,” *arXiv preprint arXiv:2010.04159*, 2020.

[27] Y. Fang, B. Liao, X. Wang, J. Fang, J. Qi, R. Wu, J. Niu, and W. Liu, “You only look at one sequence: Rethinking transformer in vision through object detection,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 26 183–26 197, 2021.

[28] P. varma, “5 Object Detection Evaluation Metrics That Data Scientists Should Know,” https://analyticsindiamag.com/5-object-detection-evaluation-metri\cs-that-data-scientists-should-know/, accessed: 2022-05-6.

[29] I. Majil, M.-T. Yang, and S. Yang, “Augmented reality based interactive cooking guide,” *Sensors*, vol. 22, no. 21, p. 8290, 2022.

[30] V. Rajput, “YOLO v4 explained in full detail,” https://medium.com/aiguys/yolo-v4-explained-in-full-detail-5200b77aa825, accessed: 2022-012-21.

[31] D. Li, R. Wang, P. Chen, C. Xie, Q. Zhou, and X. Jia, “Visual feature learning on video object and human action detection: a systematic review,” *Micromachines*, vol. 13, no. 1, p. 72, 2021.

[32] K. Sambasivarao, “Non-maximum Suppression (NMS),” https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c, accessed: 2022-12-21.

[33] P. Sun, Y. Jiang, E. Xie, W. Shao, Z. Yuan, C. Wang, and P. Luo, “What makes for end-to-end object detection?” in *International Conference on Machine Learning.* PMLR, 2021, pp. 9934–9944.

[34] Y. Nimmagadda, K. Kumar, Y.-H. Lu, and C. G. Lee, “Real-time moving object recognition and tracking using computation offloading,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2010, pp. 2449–2455.

[35] D. Holz, A. Topalidou-Kyniazopoulou, J. Stückler, and S. Behnke, “Real-time object detection, localization and verification for fast robotic depalletizing,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2015, pp. 1459–1466.

[36] A. Canziani, A. Paszke, and E. Culurciello, “An analysis of deep neural network models for practical applications,” *arXiv preprint arXiv:1605.07678*, 2016.

[37] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, and S. Guadarrama, “Speed/accuracy trade-offs for modern convolutional object detectors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7310–7311.

[38] H. Mao, X. Yang, and W. J. Dally, “A delay metric for video object detection: What average precision fails to tell,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 573–582.

[39] S. L. Mïller-Abdelrazeq, S. Stiehm, M. Haberstroh, and F. Hees, “Perceived effects of cycle time in human-robot-interaction,” in *2018 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO).* IEEE, 2018, pp. 25–30.

[40] K.-H. Tang, C.-F. Ho, J. Mehlich, and S.-T. Chen, “Assessment of handover prediction models in estimation of cycle times for manual assembly tasks in a human–robot collaborative environment,” *Applied Sciences*, vol. 10, no. 2, p. 556, 2020.

[41] Fast-Berglund and P. Thorvald, “Variations in cycle-time when using knowledge-based tasks for humans and robots,” *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 152–157, 2021.

[42] M. Zhang, C. Li, Y. Shang, and Z. Liu, “Cycle Time and Human Fatigue Minimization for Human-Robot Collaborative Assembly Cell,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6147–6154, 2022.

[43] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, “A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering,” in *International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 4559–4566.

[44] M. Merler, C. Galleguillos, and S. Belongie, “Recognizing groceries in situ using in vitro training data,” in *2007 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, 2007, pp. 1–8.

[45] C. G. Melek, E. B. Sonmez, and S. Albayrak, “A survey of product recognition in shelf images,” in *2017 International Conference on Computer Science and Engineering (UBMK).* IEEE, 2017, pp. 145–150.

[46] E. Goldman, R. Herzig, A. Eisenschtat, J. Goldberger, and T. Hassner, “Precise detection in densely packed scenes,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5227–5236.

[47] Y. Cai, L. Wen, L. Zhang, D. Du, and W. Wang, "Rethinking object detection in retail stores," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, 2021, pp. 947–954.

[48] F. Chen, H. Zhang, Z. Li, J. Dou, S. Mo, H. Chen, Y. Zhang, U. Ahmed, C. Zhu, and M. Savvides, "Unitail: Detecting, reading, and matching in retail scene," *arXiv preprint arXiv:2204.00298*, 2022.

[49] M. Saqlain, S. Rubab, M. M. Khan, N. Ali, and S. Ali, "Hybrid approach for shelf monitoring and planogram compliance (hyb-smpc) in retails using deep learning and computer vision," *Mathematical Problems in Engineering*, vol. 2022, 2022.

[50] J.-Y. Jeon, S.-W. Kang, H.-J. Lee, and J.-S. Kim, "A retail object classification method using multiple cameras for vision-based unmanned kiosks," *IEEE Sensors Journal*, vol. 22, no. 22, pp. 22 200–22 209, 2022.

[51] G. Agnihotram, N. Vepakomma, S. Trivedi, S. Laha, N. Isaacs, S. Khatravath, P. Naik, and R. Kumar, "Combination of Advanced Robotics and Computer Vision for Shelf Analytics in a Retail Store," in *2017 International Conference on Information Technology (ICIT)*, Dec. 2017, pp. 119–124.

[52] F. Coelho, S. Relvas, and A. P. F. Barbosa-Póvoa, "Simulation Of An Order Picking System In A Manufacturing Supermarket Using Collaborative Robots." in *ECMS*, 2018, pp. 83–88.

[53] R. Alves, B. A. Linhares, and J. R. Souza, "Autonomous shopping cart: A new concept of service robot for assisting customers," in *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE).* IEEE, 2018, pp. 451–456.

[54] Y. Aquilina and M. A. Saliba, "An automated supermarket checkout system utilizing a SCARA robot: preliminary prototype development," *Procedia Manufacturing*, vol. 38, pp. 1558–1565, 2019.

[55] W. Hu, J. Qiu, F. Zhang, Q. Yang, P. Wang, and C. Geng, "Control and Fetching Strategy of Goods-Picking Robot in the Self-service Supermarket," in *2019 IEEE*

9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER). IEEE, 2019, pp. 1285–1288.

[56] D. Ryumin, I. Kagirov, A. Axyonov, N. Pavlyuk, A. Saveliev, I. Kipyatkova, M. Zelezny, I. Mporas, and A. Karpov, “A multimodal user interface for an assistive robotic shopping cart,” *Electronics*, vol. 9, no. 12, p. 2093, 2020.

[57] R. Caccavale, P. Arpenti, G. Paduano, A. Fontanellli, V. Lippiello, L. Villani, and B. Siciliano, “A Flexible Robotic Depalletizing System for Supermarket Logistics,” *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4471–4476, Jul. 2020.

[58] P. Arpenti, R. Caccavale, G. Paduano, G. Andrea Fontanelli, V. Lippiello, L. Villani, and B. Siciliano, “RGB-D Recognition and Localization of Cases for Robotic Depalletizing in Supermarkets,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6233–6238, Oct. 2020.

[59] X. Zhang, Q. Xu, B. Li, and L. Liu, “Design and Implementation of Supermarket Shopping Robot,” in *2021 3rd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM).* IEEE, 2021, pp. 233–236.

[60] M. Costanzo, G. De Maria, G. Lettera, and C. Natale, “Can Robots Refill a Supermarket Shelf?: Motion Planning and Grasp Control,” *IEEE Robotics & Automation Magazine*, vol. 28, no. 2, pp. 61–73, Jun. 2021.

[61] P. Jund, N. Abdo, A. Eitel, and W. Burgard, “The freiburg groceries dataset,” *arXiv preprint arXiv:1611.05799*, 2016.

[62] X.-S. Wei, Q. Cui, L. Yang, P. Wang, and L. Liu, “Rpc: A large-scale retail product checkout dataset,” *arXiv preprint arXiv:1901.07249*, 2019.

[63] RetailVision, “Product Detection in Densely Packed Scenes Challenge,” https://retailvisionworkshop.github.io/detection_challenge_2020/, accessed: 2022-12-29.

[64] J. Peng, C. Xiao, and Y. Li, “Rp2k: A large-scale retail product dataset for fine-grained image classification,” *arXiv preprint arXiv:2006.12634*, 2020.

[65] Heartex, “labelImg,” https://github.com/heartexlabs/labelImg, accessed: 2022-12-29.

[66] D. Koubaroulis, J. Matas, and J. Kittler, "Evaluating colour-based object recognition algorithms using the soil-47 database," 2002.

[67] A. Rocha, D. C. Hauagge, J. Wainer, and S. Goldenstein, "Automatic fruit and vegetable classification from images," *Computers and Electronics in Agriculture*, vol. 70, no. 1, pp. 96–104, 2010.

[68] P. Follmann, T. Bottger, P. Hartinger, R. Konig, and M. Ulrich, "Mvtec d2s: densely segmented supermarket dataset," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 569–585.

[69] M. George and C. Floerkemeier, "Recognizing products: A per-exemplar multi-label image classification approach," in *European Conference on Computer Vision*. Springer, 2014, pp. 440–455.

[70] G. Varol and R. S. Kuzu, "Toward retail product recognition on grocery shelves," in *Sixth International Conference on Graphic and Image Processing (ICGIP 2014)*, vol. 9443. SPIE, 2015, pp. 46–52.

[71] L. Karlinsky, J. Shtok, Y. Tzur, and A. Tzadok, "Fine-grained recognition of thousands of object categories with single-example training," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4113–4122.

[72] Y. Hao, Y. Fu, and Y.-G. Jiang, "Take goods from shelves: a dataset for class-incremental object detection," in *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, 2019, pp. 271–278.

[73] "ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation," Nov. 2022. [Online]. Available: https://zenodo.org/record/7347926

[74] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[75] RangiLyu, "NanoDet-Plus," Jan. 2023, original-date: 2020-10-19T08:36:25Z. [Online]. Available: https://github.com/RangiLyu/nanodet

[76] S. Rath, “YOLOv5 – Custom Object Detection Training,” https://learnopencv.com/custom-object-detection-training-using-yolov5/, accessed: 2022-12-29.

[77] C. Imane, “YOLOv5 model architecture [Explained],” https://iq.opengenus.org/yolov5/, accessed: 2022-12-29.

[78] M. Rizwan, “YOLOv7 Architecture Explanation,” https://www.cameralyze.co/blog/yolov7-architecture-explanation, accessed: 2022-12-28.

[79] S. Rath, “YOLOv7 Object Detection Paper Explanation Inference,” https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/, accessed: 2022-12-28.

[80] P. Marimuthu, “High Accuracy Lightweight and Superfast Model Nanodet Plus,” https://www.analyticsvidhya.com/blog/2022/07/high-accuracy-lightweight-and-superfast-model-nanodet-plus/, accessed: 2022-12-29.

[81] RangiLyu, “NanoDet-Plus,” https://zhuanlan.zhihu.com/p/449912627, accessed: 2022-12-29.

[82] J. Briones, “Object Detection with Transformers,” https://medium.com/swlh/object-detection-with-transformers-437217a3d62e, accessed: 2022-12-29.

[83] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[84] Python-3.11.1 Documentation, “The Python Profilers,” https://docs.python.org/3/library/profile.html, accessed: 2022-012-13.

[85] E. Martinian, “Statistical Profiling (and other fun with the sys module),” https://speakerdeck.com/pycon2019/emin-martinian-statistical-profiling-and-other-fun-with-the-sys-module, accessed: 2022-12-29.

[86] K. Oksuz, B. C. Cam, S. Kalkan, and E. Akbas, "Imbalance problems in object detection: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 10, pp. 3388–3415, 2020.

[87] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.