

AI Powered Self-Checkout System

Final Year Project Report



GSN: Fall 22-13

Group Members

Muhammad Talha	19L-1271
Muhammad Abdullah	19L-1382
Muhammad Shehryar	19L-1397

Advisor

Mr. Mohsin Yousuf

Client

Mr. Mohsin Yousuf

3rd June, 2023

Department of Electrical Engineering
National University of Computer and Emerging Sciences, Lahore

CERTIFICATIONS

This document has been prepared by all of us together and we take joint ownership of its contents. We have provided references to the material consulted in preparing this document and, to the best of our knowledge, have not plagiarized anything.

Muhammad Talha, 19L-1271



Date: _____

Muhammad Abdullah, 19L-1382



Date: _____

Muhammad Shehryar, 19L-1397



Date: _____

I am the client of the product proposed in this document and the product specifications and other details are according to my requirements.

Client:

Mr. Mohsin Yousuf _____ Date: _____

The final year project proposal in this document is being submitted to the department of Electrical Engineering with my approval.

Advisor:

Mr. Mohsin Yousuf _____ Date: _____

Head of Department:

Dr. Saima Zafar _____ Date: _____

Abstract

In the fast-paced retail industry, long and inefficient checkout processes have emerged as a significant problem, impacting both customer satisfaction and sales performance. It's striking that 70% of customers name waiting in checkout lines as their least favored part of shopping in physical stores [1]. This clear problem statement prompted us to find an innovative solution, employing the power of Artificial Intelligence (AI) and computer vision to improve the retail checkout experience.

To address this issue, we set out to develop a system capable of quickly and accurately identifying and scanning products, thereby doing away with the need for traditional barcodes and RFID tags. The system leverages the power of the latest iteration of the You Only Look Once (YOLO) object detection system, YOLOv7. This tool offers significant improvements in accuracy and speed over previous versions and existing solutions in the market [5]. The system was purposefully designed to be integrated into a moving conveyor belt setting, enabling real-time product detection and quantity identification without interrupting the conveyor's operation. The system is also designed to be easily attachable to existing checkout lanes and is cost-effective

Following our thorough procedures, we managed to create a system that greatly reduces checkout times. It efficiently identifies the type and number of products, rapidly generates invoices, and displays this information on an easy-to-use interface. This technology outperforms traditional barcode scanning methods, which often required multiple scans, causing longer wait times and lowering customer satisfaction.

In conclusion, the broader impact of our project is significant. By providing a quicker, more efficient checkout process, we are improving the overall shopping experience, increasing customer satisfaction and boosting retail sales. Our findings and the developed system hold the potential to redefine industry standards, paving the way for more sophisticated, AI-enabled solutions.

Keywords: Retail Industry, Checkout Process, YOLOv7, Real-Time Tracking, Cost-Effective Solution, Customer Satisfaction, Sales Performance.

Table of Contents

Abstract	2
Table of Contents	3
List of Figures	5
List of Tables	7
Acknowledgements	8
Chapter 1: Introduction	9
Chapter 2: Problem Definition (Client Requirements)	10
2.1 Problem Formulation	10
2.2 Mapping to Sustainable Development Goals (SDG)	11
2.3 Record of Meetings with Client	11
2.4 Preliminary Product Specification	12
2.5 Expected Functionality of Product.....	13
Chapter 3: Problem Analysis	14
3.1 Engineering Problem Model	14
3.2 Recent Similar Projects.....	15
3.3 Distinguishing Features of this Project	15
3.4 Societal and Environmental Implications of the Project.....	16
Chapter 4: Design and Implementation	18
4.2 Preliminary Design	18
4.2.1 Hardware Block Diagram	20
4.2.2 Software Block Diagram.....	20
4.3 Detailed Hardware and Software Design:.....	21
4.3.1 Calculations.....	27
4.3.2 Hardware Design.....	29
4.3.3 Software Design	29
4.3.4 Software Implementation:.....	30

4.3.5 Hardware Implementation:	43
Chapter 5: Investigation and Testing	53
5.1 Camera and Object Recognition Test:	53
5.2 Performance between Local Processing and Client-Server Method:	57
5.3 User Interface and Touchscreen Test:	58
5.4 Complete System Test:	59
5.5 Payment Gateway Test:	60
Chapter 6: User Guide	62
Step 1: Setting Up the Self-Checkout System	62
Step 2: Scanning Items	62
Step 3: Reviewing Your Cart	64
Step 4: Making Payment	64
Step 5: Troubleshooting	64
Chapter 7: Deliverables and Cost	66
7.1 Deliverables	66
7.2 Project Plan	67
7.3 Project Cost (Projected)	71
Chapter 8: Conclusion	73
References	76
Appendices	79
Glossary	86

List of Figures

Figure 1 SDG 9.....	11
Figure 2 Major Components of the System	18
Figure 3 Top Level Block Diagram	19
Figure 4 Software Block Diagram.....	19
Figure 5 User Interface.....	20
Figure 6 Hardware Block Diagram	20
Figure 7 Software Block Diagram.....	21
Figure 8 Flask API	21
Figure 9 HTML, CSS AND JavaScript	22
Figure 10 Working of YOLO Algorithm.....	23
Figure 11 YOLOv7 Pre-trained Models.....	24
Figure 12 YOLOv7 Predicted Batch	24
Figure 13 Performance Metrics over Iterations.....	24
Figure 14 Predictions made by Demo Model	25
Figure 15 Product Multi Price Bracket Problem	25
Figure 16 Solution # 1 - Fetching Dimensions.....	25
Figure 17 Text Recognition from ABINET(MMOCR).....	26
Figure 18 Solution # 2 - Text Recognition	26
Figure 19 All Hardware Components.....	27
Figure 20 3D Schematic for the Conveyor Belt	29
Figure 21 Images for Training	31
Figure 22 Label File	31
Figure 23 YoloV7 (Regular) Model Metrics	32
Figure 24 Confusion Matrix for YoloV7 (Regular)	33
Figure 25 Test Results from YoloV7 (Regular)	33
Figure 26 UI of the Web-App	42
Figure 27 - Nvidia Jetson Nano Attached to a Screen.....	43
Figure 28 - Logitech C310 Webcam	44
Figure 29 7-inch touchscreen for Jetson Nano	44
Figure 30 10A 12-40V PWM Controller.....	45
Figure 31 Detachable Box	46
Figure 32 Base for the Conveyor Belt	46
Figure 33 Complete Structure.....	47
Figure 34 Painting Process of Conveyor Belt	47
Figure 35 Painting of Detachable Box.....	47
Figure 36 Testing of Conveyor Belt	48
Figure 37 Box Housing	49

Figure 38 Configuring Jetson Nano	49
Figure 39 Testing the System on Jetson Nano	50
Figure 40 - Camera Installation	50
Figure 41 Screen Installation & Testing	51
Figure 42 Testing the System	52
Figure 43 Testing on Choco Bliss.....	53
Figure 44 Choco Bliss displayed on Counter	54
Figure 45 Testing on Slanty	55
Figure 46 Testing on 2 products at once.....	55
Figure 47 Testing on Head & Shoulders from the Front	56
Figure 48 Testing on Head & Shoulders from the Back	56
Figure 49 Testing on Euthrix DF	56
Figure 50 Testing on Shangrila Chilli Sauce.....	57
Figure 51 Testing on Mountain Dew.....	57
Figure 52 Touch Screen Interface	59
Figure 53 All checkout items in the cart	60
Figure 54 Payment Processed for the Order.....	60
Figure 55 Entering Payment Information	61
Figure 56 Payment Successfully Completed	61
Figure 57 Payment History.....	61
Figure 58 Video Streaming Demonstration	63
Figure 59 Checkout Counter	63
Figure 60 Jetson Nano Pinout	79
Figure 61 Jetson Nano Pin Diagram	79

List of Tables

Table 1: Table of Meetings	11
Table 2 OCR Results	54
Table 3 High Level Work Breakdown Structure	67
Table 4 Complete Gantt Chart of the Project	67
Table 5 Projected vs. Actual Time	71
Table 6 Projected Cost	71
Table 6 Actual Cost.....	72

Acknowledgements

We would like to express our gratitude to our esteemed advisor, Mr. Mohsin Yousuf, for his invaluable guidance, unwavering support, and thoughtful mentorship throughout the duration of this project. His expert advice and insightful suggestions have been pivotal in shaping our work.

Our sincere appreciation extends to Ms. Khizra Farooq and Ms. Sara Kiran, members of the evaluation committee, whose constructive critique and valuable input have further refined our project, enhancing the strength and validity of our research findings.

We are also deeply grateful to the Pakistan Engineering Council (PEC) for their generous provision of funds. Their financial support has been crucial in bringing our research to fruition.

Our acknowledgement would be incomplete without mentioning the conducive academic environment provided by the Faculty of Electrical Engineering at FAST-NU Lahore. The resources and facilities provided have significantly facilitated our study, enabling the successful completion of this project.

Lastly, we express our heartfelt gratitude to our families. Their relentless support, encouragement, and belief in our abilities have been a constant source of motivation throughout this journey. They have shared our challenges and celebrated our successes, for which we are profoundly grateful.

We appreciate all who have directly or indirectly contributed to our work. The support and assistance received have been instrumental in the successful realization of this project.

Chapter 1: Introduction

The evolution of retail has consistently been driven by technological advancements, with AI and computer vision serving as the latest catalysts. The prevalent issue of slow and inefficient checkout processes in retail stores continues to persist, with 70% of customers marking their wait in checkout lines as the least enjoyable part of their in-store shopping experience [1]. Consequently, the retail industry is in dire need of an effective solution, which motivated the development of our system.

Several previous studies have investigated different ways to tackle this problem [2]. The use of barcodes and RFID tags has long been the standard practice, but these methods fall short in efficiency and customer satisfaction, primarily due to their reliance on manual scanning, which is time-consuming and prone to errors. Some solutions have suggested the use of AI, but most fail to realize the full potential of this technology. Our project takes a step further by incorporating an advanced object detection system.

Our solution, an automated checkout system, leverages AI and computer vision to redefine the checkout experience. It is powered by YOLOv7, the latest iteration of the You Only Look Once (YOLO) object detection system, known for its speed and accuracy [5]. The system eliminates the need for manual barcode scanning by automatically identifying products and their quantities as they pass on a conveyor belt. It generates invoices swiftly and accurately, displays them on a user-friendly interface, and keeps the conveyor belt in continuous operation for optimal efficiency.

Our design hypothesis centered around the potential of AI and computer vision to speed up the checkout process without sacrificing accuracy. The system aims to improve the customer shopping experience by reducing checkout time, making it more efficient and user-friendly. By doing so, it could potentially boost sales performance and customer satisfaction in the retail industry.

In conclusion, our project takes a comprehensive approach towards solving a pervasive problem in the retail industry. We connect the real-world issue of slow checkout processes with an innovative AI-based solution, establishing a direct link between the problem and our proposed solution. We believe that our project paves the way for the broader adoption of AI and computer vision in retail and beyond.

Chapter 2: Problem Definition (Client Requirements)

2.1 Problem Formulation

The retail industry has been grappling with the issue of slow checkout processes, primarily due to the prevalent method of barcode scanning. This conventional approach has become a bottleneck in streamlining retail operations, resulting in long queues at checkout lanes and leading to customer dissatisfaction [1], particularly during peak shopping periods like holidays or special occasions. This problem is compounded by the fact that customers are becoming more demanding, expecting quick, efficient, and seamless shopping experiences.

Upon consultation with our advisor, we recognized the need for a more effective solution that enhances the speed and accuracy of the checkout process while maintaining cost-effectiveness. The consensus was that the ideal solution would significantly minimize checkout times, thus improving the overall customer experience.

From the information gathered, we identified our problem as follows:

"How to develop a system that expedites the checkout process by surpassing the speed and efficiency of manual barcode scanning, while ensuring accuracy, cost-effectiveness, and improved customer satisfaction?"

Following this problem formulation, we established the following order of importance for the system features:

- 1. Speed:** The system should significantly reduce the checkout time compared to manual barcode scanning.
- 2. Accuracy:** It must correctly identify products and their quantities to avoid discrepancies in invoicing.
- 3. Cost-effectiveness:** The solution should be economically feasible for retail stores of various sizes.
- 4. Customer satisfaction:** By reducing queue times and streamlining the checkout process, the system should improve overall customer satisfaction.

2.2 Mapping to Sustainable Development Goals (SDG)



Figure 1 SDG 9

Our project resonates with Sustainable Development Goal (SDG) 9: Industry, Innovation, and Infrastructure. SDG 9 emphasizes the promotion of sustainable industrialization and the fostering of innovation [16], both central aspects of our project. By leveraging the cutting-edge advancements in AI and computer vision, we strive to revolutionize the retail industry, specifically focusing on improving efficiency at the checkout process. Our system aims to streamline this traditionally tedious procedure, reducing the turnaround time and enhancing the overall shopping experience. This innovation is expected to spur productivity in retail settings and is indicative of the sustainable and resilient infrastructure that SDG 9 seeks to promote. Therefore, our project's objectives align directly with the key elements of SDG 9, underlining our commitment to contributing to sustainable development on a global scale.

2.3 Record of Meetings with Client

Table 1: Table of Meetings

Meeting	Date	Discussion
1	(9-Sep-22)	Problem Statement Refinement and Discussing Solutions
2	(16-Sep-22)	Discussion of Final Product Description
3	(10-Oct-22)	Selection of Sustainable Development Goal
4	(17-Oct-22)	Progress in App Development

5	(24-Oct-22)	Selection of Hardware Components, Same Product Problem and Selection of Algorithm
6	(21-Nov-22)	Progress Before Final presentation
7	(25-Nov-22)	Feedback on Final Presentation and FYP-1 Report
8	(2-Feb-23)	Progress After Semester Break, Remaining Tasks, Plans
9	(9-Feb-23)	Hardware Integration and Conveyor Belt Mechanism Updates
10	(16-Feb-23)	Progress on Final Trained Model & Acquisition of Jetson Nano
11	(9-Mar-23)	Updates to Payment Gateway and Check-out Procedure
12	(16-Mar-23)	Improving User Interface based on Feedback
13	(22-Mar-23)	Final System Testing and Debugging
14	(4-Apr-23)	Comprehensive System Debugging
15	(19-Apr-23)	Future Work

2.4 Preliminary Product Specification

In response to the client's need for an efficient, accurate, and cost-effective product identification system that can seamlessly integrate with existing retail infrastructure, we have conceived an engineering solution that meets these demands.

Our proposed solution is a modular, attachable box that serves as a product identification and scanning system. It is designed to be the same size as a standard checkout counter, measuring

approximately 2 feet in height, 1 foot in length. The entry points of the box will remain open to accommodate the movement of products on the conveyor belt.

The system is powered by NVIDIA Jetson Nano, a low-cost, high-performing Single Board Computer (SBC) that will be housed inside the roof of the box. The SBC will be paired with a 1080p30 high-resolution camera that boasts a wide field of view, ensuring comprehensive coverage of the box's interior.

The system will employ a machine learning model, specifically the YOLOv7 object classification model, for product identification. The model will be trained using a dataset of retail products and subsequently deployed on the SBC. This enables the live processing of product images captured by the camera as they pass through the box on the conveyor belt.

To ensure usability, the system will also feature a user-friendly, 7-inch touch screen display on one side of the box. This display will provide real-time information about product names and quantities, enabling customers and cashiers to easily navigate and verify their purchases.

In summary, this system is designed to revolutionize the checkout process, combining speed, accuracy, and affordability in a package that fits seamlessly into the existing retail setup. As an engineering product, it encapsulates the translation of vague client requirements into a tangible, innovative solution.

2.5 Expected Functionality of Product

Designed to streamline the retail checkout process, our product leverages machine vision and AI to automate the product scanning and checkout process. Here is how it functions in a real-world retail environment:

1. The customer begins the checkout process by placing their grocery items on the conveyor belt.
2. As the items travel along the conveyor belt, they pass through our specially designed box, where the attached camera under the roof of the box captures images of each product.
3. The images are then immediately sent to the single board computer housed within the system.
4. Using the pre-trained YOLOv7 model, the computer rapidly identifies each product, determines their quantity, and fetches their respective prices.
5. This information, including the product names, quantities, and their total price, is displayed on the user-friendly, 7-inch touch screen interface attached to the box.
6. The customer can then easily proceed to payment, guided by the intuitive user interface.

Chapter 3: Problem Analysis

3.1 Engineering Problem Model

The backbone of this project rests on the integration of computer vision, convolutional neural networks (CNNs), deep learning, and Optical Character Recognition (OCR). These technologies combine to facilitate real-time object detection and classification, offering the potential to revolutionize the retail checkout process.

Computer vision is a field that enables computers to interpret and make sense of visual data, such as images and videos. Within this field, object detection is a technique that locates and identifies objects within an image or video stream. The technique is integral to various applications, including autonomous driving, facial recognition, and medical imaging analysis.

A vital aspect of this project is the use of deep learning, a subset of machine learning [3], to automate the learning of image features. It's the key driving force behind the object detection capabilities of our system. In particular, we leverage Convolutional Neural Networks (CNNs), a type of Artificial Neural Network (ANN) adept at recognizing patterns in multi-dimensional spaces, making them an excellent fit for image processing.

Several deep learning techniques exist for object detection [4] including Faster-RCNN, SSD, and YOLO. Our project opts for YOLO (You Only Look Once) due to its unique approach of analyzing an image in a single pass, providing both speed and efficiency. More specifically, we employ YOLOv7, the latest iteration in the YOLO family. YOLOv7 stands out due to its top-ranking performance in real-time object detection on the COCO dataset.

Moreover, our system supports multi-class classification, allowing each distinct product to be assigned to a specific class depending on the product type. To facilitate this, a labelled dataset of the grocery products we aim to detect is created and divided into training and testing subsets. The training subset is then used to train the YOLO algorithm.

Finally, we utilize OCR - a technique for extracting machine-readable text from images or video - to add another layer of accuracy to our system. This approach helps in differentiating the same product available in various price brackets. It serves as a secondary validation mechanism, ensuring the robustness and accuracy of our solution.

In summary, the combination of computer vision, CNNs, deep learning, and OCR provides a comprehensive and reliable solution to the problem at hand, marking a significant advancement in the retail checkout process.

3.2 Recent Similar Projects

A Hybrid Checkout System by Tycho Hartman [7]

This project outlined the design of a combined checkout system comprising both attended checkout and self-checkout lanes. The checkout process within this model is handled either by customers using a barcode scanner or through RFID tags on products. However, unlike our approach, Hartman's project does not incorporate machine learning or AI-based object detection capabilities.

AI-based machine vision for retail self-checkout system by Anton Rigner [8]

This Master Thesis presents a self-checkout system that uses the YOLOv3 object detection algorithm for retail products. The project discussed three prominent object detection algorithms (Mask R-CNN, RetinaNet, and YOLOv3) and evaluated their performance on a dataset of ten Swiss retail products. A functional prototype was also developed, which featured a shelf that detected and displayed an object once removed. However, the scope of this project is limited to static scenarios such as shelves, and the checkout screen is immediately adjacent to the shelf, which contrasts with our broader, more versatile solution.

ARC: A Vision-based Automatic Retail Checkout System [9]

This project also introduced a self-checkout system using computer vision and convolutional neural networks, trained on a dataset of a hundred local retail items across various categories. The design incorporated a conveyor belt with a hood and a LASER-LDR (Light Dependent Resistor) module to detect products. The conveyor belt would pause once a product is detected, allowing the computer to extract a frame before proceeding. However, this system relies on OpenCV for object detection and lacks real-time multiple object detection capabilities. This differs from our approach, where we aim to enable real-time detection and recognition of multiple objects simultaneously.

3.3 Distinguishing Features of this Project

Our project introduces several distinguishing features that significantly differentiate it from similar initiatives, driving notable enhancements in retail checkout experiences. The unique aspects of our project are as follows:

- **Implementation of YOLOv7:** Our system leverages the cutting-edge YOLOv7 algorithm, which provides improved performance over previous YOLO versions in terms of object detection. This choice of algorithm offers greater accuracy and efficiency in real-time identification and counting of products [5].

- **Continuous Movement:** Unlike some existing solutions, our system operates on a conveyor belt where objects remain in constant motion. This continuous flow negates the need to stop the belt for product detection, promoting operational efficiency.
- **Real-Time Multi-Object Detection:** Our solution is capable of identifying and quantifying multiple products in real-time, which considerably speeds up the checkout process and enhances user experience.
- **Modularity:** The system is designed to be an attachable module for existing checkout lanes, facilitating easy integration without disrupting the current retail setup.
- **Cost-Effective:** Compared to market solutions and similar projects, our system stands out as a cost-effective alternative. We strive to ensure its economic feasibility within the local Pakistani context.
- **User-Friendly Interface:** Our system incorporates a user-friendly interface designed to assist cashiers or customers in seamlessly navigating and utilizing the system.
- **Object Tracking:** The application of a tracking algorithm enhances the system's accuracy by continually tracking detected objects, ensuring accurate identification even in the rare instance of detector malfunction.
- **Diverse Product Identification:** Our system is capable of differentiating between the same product available in different sizes and price brackets. This is achieved through evaluating object dimensions and applying a text recognition algorithm to extract pertinent details from the product. This dual-check mechanism also serves as an additional safeguard against any potential misclassifications by YOLOv7.

3.4 Societal and Environmental Implications of the Project

This project aims to enhance the shopping experience in Pakistan by expediting the checkout process in retail stores, a sector currently lagging in terms of technological advancement [10]. The proposed system will not only improve the quality of life for customers but also encourage a shift back to in-person shopping from online platforms, especially for those items where physical verification is crucial before purchase. By reducing wait times, the system will elevate customer satisfaction and minimize the chances of disgruntlement or complaints. However, the cultural aspect of implementing a fully self-checkout system, particularly in Pakistan, has its own implications. Issues surrounding honesty and trust mean that complete autonomy of the system is not feasible. We propose a model that still involves human supervision to mitigate the risk of exploitation, ensuring that all items are correctly placed on the conveyor belt.

Environmentally, the system must be power-efficient. With Pakistan grappling with an energy crisis, it is paramount to deploy a system that optimizes power consumption. Utilizing a single-board computer offers an energy-efficient solution without compromising processing power, making the system sustainable for wider-scale implementation.

From a societal perspective, the implementation of self-checkout systems could lead to job losses in a country already battling high unemployment rates. Our proposed system, however, doesn't aim to render cashiers redundant. Instead, it seeks to alleviate their workload and enhance customer service efficiency. The cashiers would still be present, providing guidance to customers on system usage and monitoring both the system and customers for smooth operations. The primary objective of this system, therefore, is not to eliminate jobs, but to streamline the checkout process, making shopping experiences more efficient and enjoyable.

Chapter 4: Design and Implementation

Design Requirements:

- System should be able to identify products with high accuracy.
- System should be faster than manual scanning.
- System should reduce the wait time of checkout lines.
- UI of the system should be easy to navigate even for a new customer.
- System should be modular such that it is attachable to existing conveyor belts.
- System should be cost-effective than market available solutions.

Design Constraints:

- This project will focus on conveyor belt implementation only and not on smart carts and baskets.
- Initial Dataset would be kept small and have popular items from each category.
- The UI should be easy to use and navigate for any customer regardless of their educational level
- The system shouldn't have high power consumption due to already high demands and relatively less supply of electricity in Pakistan.
- The components of the hardware shouldn't be exposed to account for safety of the system and the customer.

4.2 Preliminary Design

The system will have 3 components

- An existing conveyor belt that will move the products around
- An identification model that detects and identifies the retail products via cameras.
- An app that displays the retail products, their prices and a checkout button to generate a receipt.

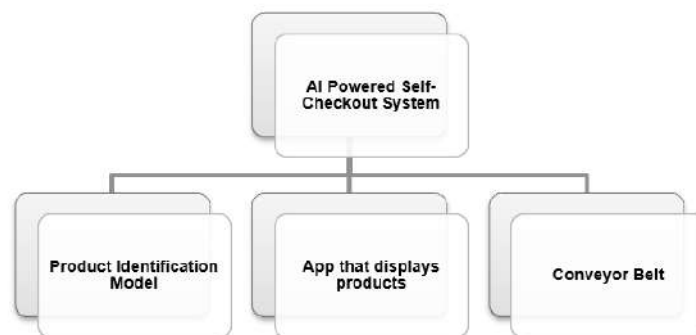


Figure 2 Major Components of the System

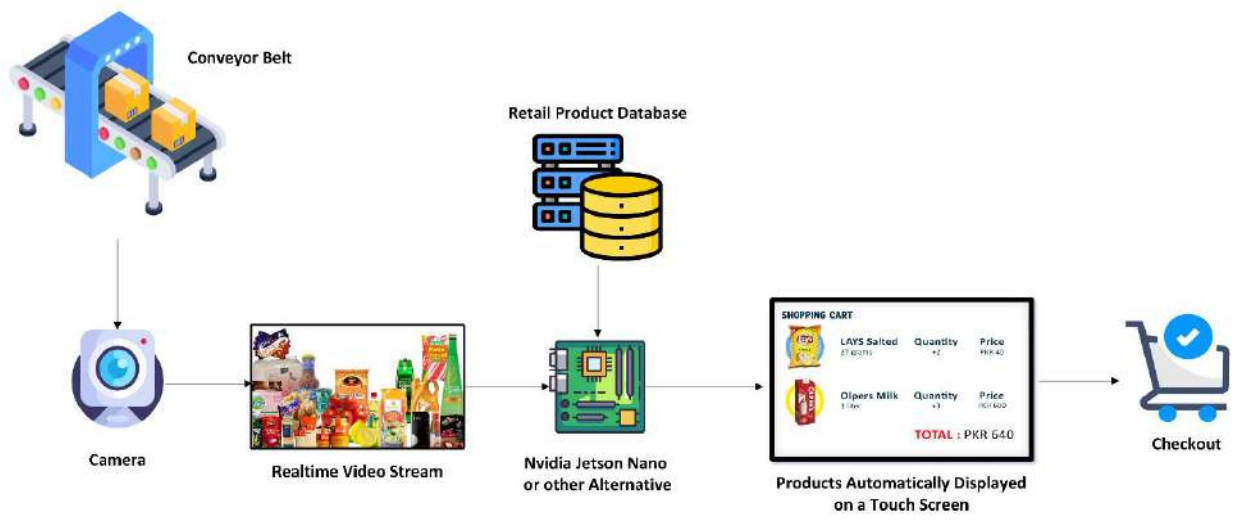


Figure 3 Top Level Block Diagram

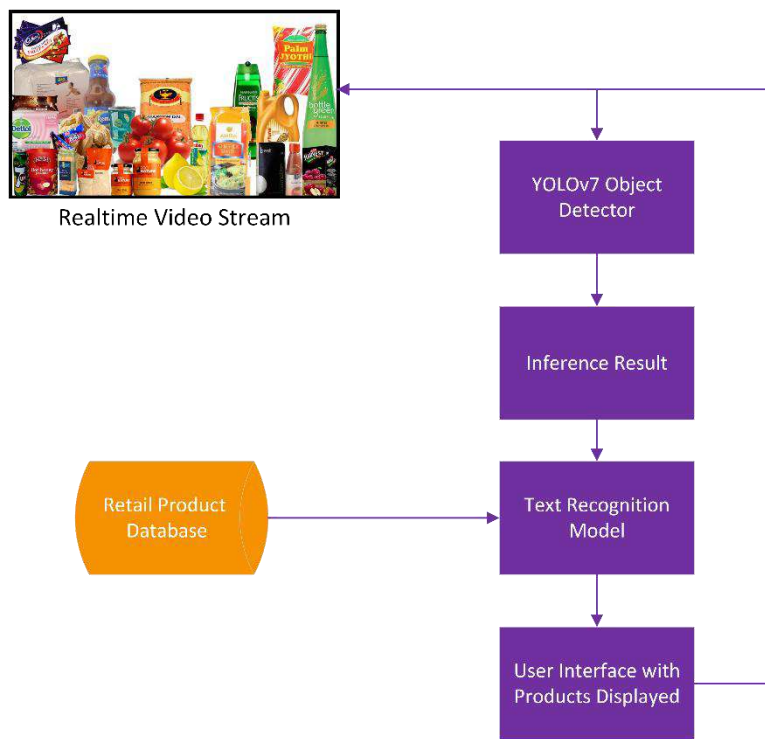


Figure 4 Software Block Diagram



Figure 5 User Interface

4.2.1 Hardware Block Diagram

The conveyor belt has a modular box attached to it. The box houses a Nvidia Jetson Nano or alternative SBC inside its roof. The camera is attached below the roof of the box which is also connected to the Nvidia Jetson Nano/SBC. A 7-inch touchscreen is attached on the side of the box visible to the customer. This touchscreen is also connected to Nvidia Jetson Nano/SBC

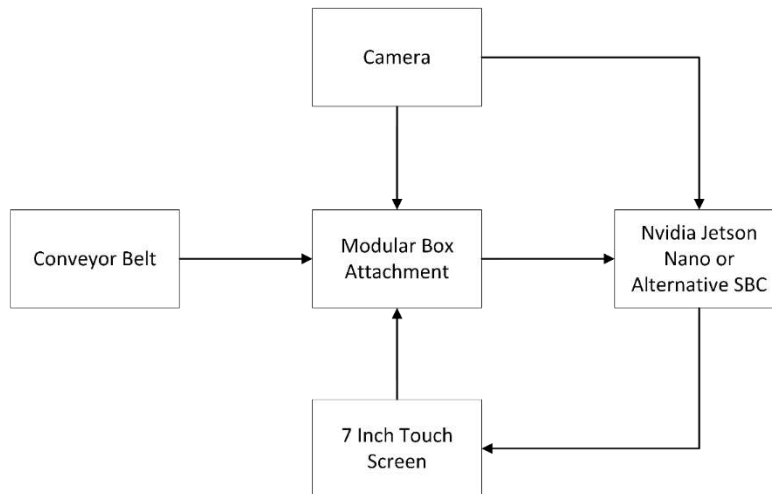


Figure 6 Hardware Block Diagram

4.2.2 Software Block Diagram

The front end of the web app is made using HTML, CSS and JavaScript. This front end communicates with Flask API on what products to show on the app. The Flask API communicates with the YoloV7 model to get the required product data for the front end to display.

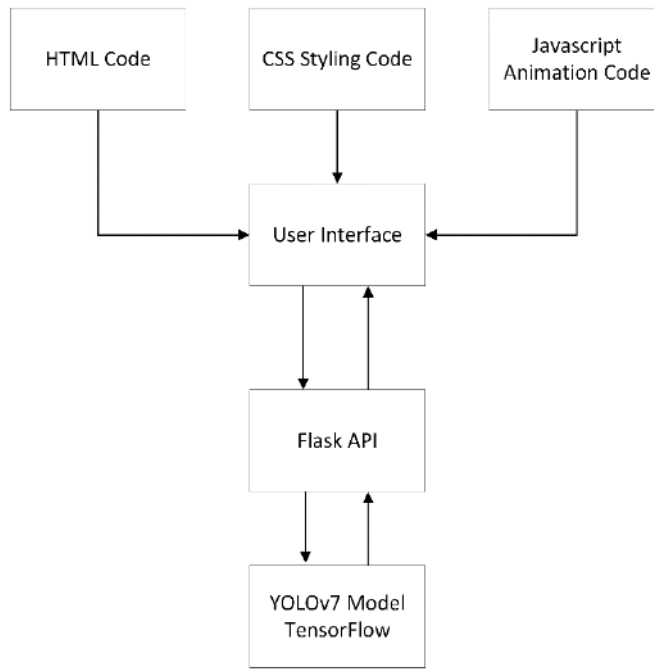


Figure 7 Software Block Diagram

4.3 Detailed Hardware and Software Design:

The interactive display of products on the touchscreen will be facilitated by a web application, a strategic choice aimed at ensuring scalability. This design allows for seamless deployment across multiple branches of a departmental store, if required. To construct the backbone of this web application, we will employ Flask, a well-regarded micro-framework renowned for its utility in application development [17]. Complementing the back-end, the front-end of the application will be crafted using a combination of HTML, CSS, and JavaScript, integrating a user-friendly interface with robust functionality.



Figure 8 Flask API



Figure 9 HTML, CSS AND JavaScript

The web application can also be efficiently converted into a cross-platform desktop application with the help of Flask-Desktop. This powerful utility, powered by PySide2 [18], enables the creation of standalone executables that can operate seamlessly on diverse platforms such as Windows, Mac, and Linux. This ensures a broader reach of our solution, accommodating various user preferences and providing an added layer of flexibility, if needed.

For the object detection algorithm several options are available which include but are not limited to:

- Faster R-CNN
- EfficientDet
- ConvNeXt
- SSDNet
- SPP-Net
- HOG
- YOLOv5
- PP-YOLO
- YOLOR
- YOLOv7 and many more

Among various viable choices, YOLOv7 was selected as the optimal solution for the object detection task. The reasons for this preference include its top-ranking status in Average Precision for Real-Time Object Detection in the COCO dataset [11]. This dataset is a crucial benchmark because our system necessitates both real-time processing and high average precision.

YOLOv7 operates via a single-stage object detection methodology. It initiates by segmenting the image into N distinct grids, each grid with a dimension of $S \times S$. Object detection and localization are performed within these grids. Following this, for each grid, bounding box predictions are made along with their corresponding probability scores. In order to prevent any overlap, Non-Maximal Suppression is applied, effectively eliminating all bounding boxes associated with low probabilities. This ensures a refined and precise object detection output [11]

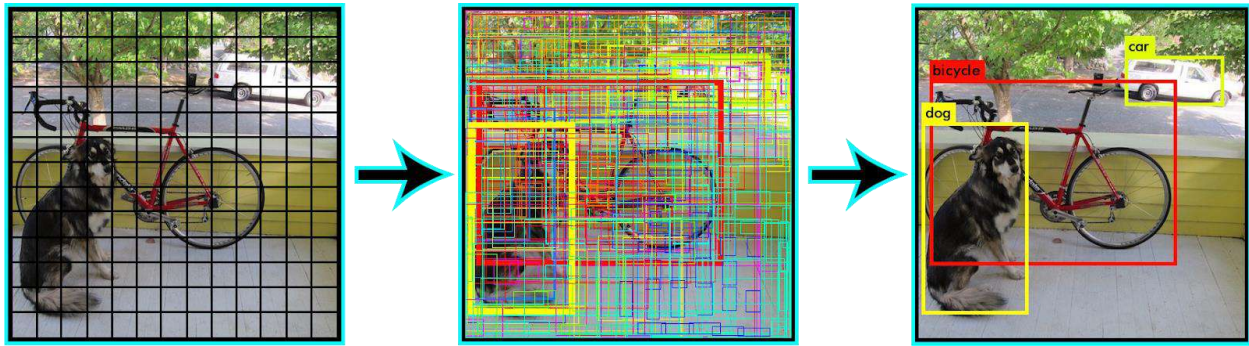


Figure 10 Working of YOLO Algorithm

To train this model, a dataset needs to be created. This dataset will contain a few products each from the following categories

- Beverages
- Dairy
- Cereals
- Frozen Foods
- Personal Care
- Household Supplies
- Pet Care
- Baby Items
- Snacks
- Canned Goods
- Condiments
- Baking
- Health Care

The process of collecting and annotating the dataset would be semi-automated to speed up the process and allow for a large number of labelled images for each product. Tools like autoAnnoter [12] and MakeSenseAI [13] can annotate images automatically using any pre-trained model. This can speed up labelling even if there is a slight cost of accuracy. Wrong labels can be corrected later. Though if annotating images without any model dependency is a priority, then DarkLabel [14] can be used which automatically annotates the videos with its built-in object tracking algorithms.

For the training part of the model, a pretrained model was selected (YOLOv7.pt for Jetson Nano)

Model	Test Size	AP ^{test}	AP ₅₀ ^{test}	AP ₇₅ ^{test}	batch 1 fps	batch 32 average time
YOLOv7	640	51.4%	69.7%	55.9%	161 fps	2.8 ms
YOLOv7-X	640	53.1%	71.2%	57.8%	114 fps	4.3 ms
YOLOv7-W6	1280	54.9%	72.6%	60.1%	84 fps	7.6 ms
YOLOv7-E6	1280	56.0%	73.5%	61.2%	56 fps	12.3 ms
YOLOv7-D6	1280	56.6%	74.0%	61.8%	44 fps	15.0 ms
YOLOv7-E6E	1280	56.8%	74.4%	62.1%	36 fps	18.7 ms

Figure 11 YOLOv7 Pre-trained Models

A demo model based on 4 products was then trained on our dataset. Below are some results and visualized metrics of the trained model.



Figure 12 YOLOv7 Predicted Batch

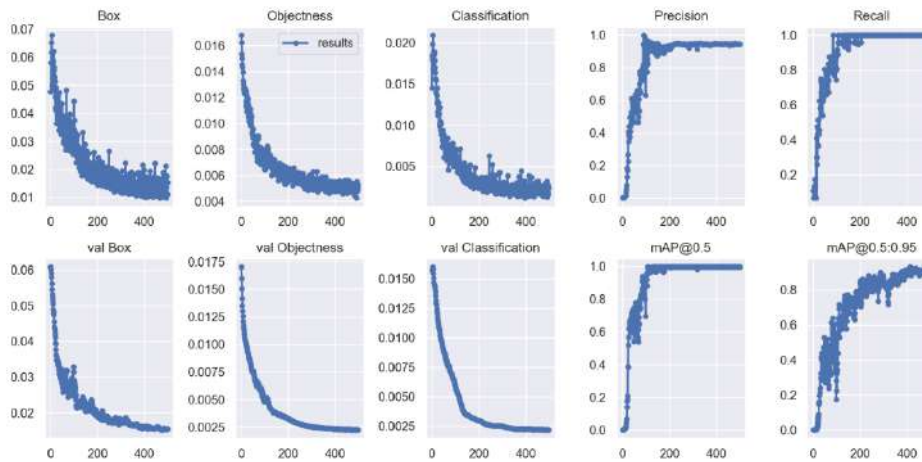


Figure 13 Performance Metrics over Iterations

Below are some predictions made by the model. It can be observed that the model can detect the front and back of an object. But the model is still unable to differentiate the same product of different sizes.



Figure 14 Predictions made by Demo Model

For that, there are two proposed solutions. The first is to use the height and width information from the bounding boxes of the product to classify the exact product. This is accurate for most scenarios but the distance of the camera needs to be fixed and the ranges of height and width need to be pre-defined for that fixed distance.



Figure 15 Product Multi Price Bracket Problem

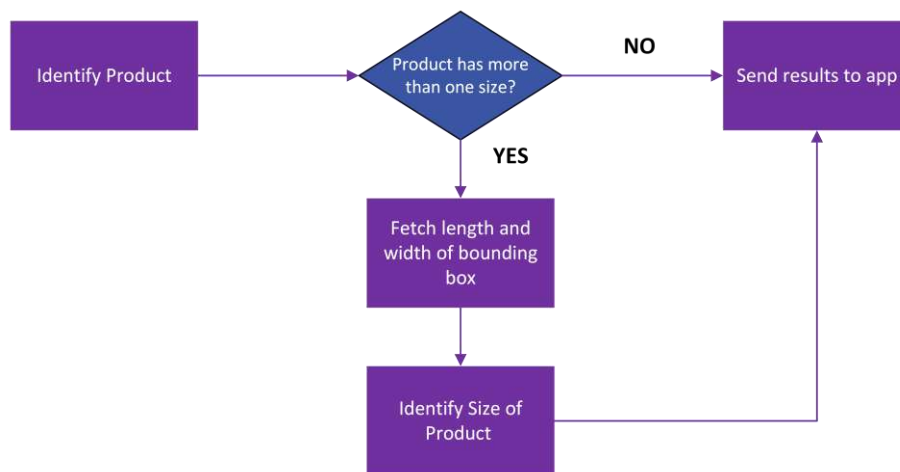


Figure 16 Solution # 1 - Fetching Dimensions

The second solution is to use a text-recognition algorithm in addition to the object detection algorithm. YOLOv7 is able to save the cropped images from its predicted results. These cropped images can be sent to a text-recognition model where it can extract information that can help the system identify the exact product. This text-recognition layer can also serve as another check even if the object detector wrongly classifies a product. The reason of applying text-recognition on a cropped image is to make sure that the text belongs to that product only and not mix-up text from multiple images.



Figure 17 Text Recognition from ABINET(MMOCR)

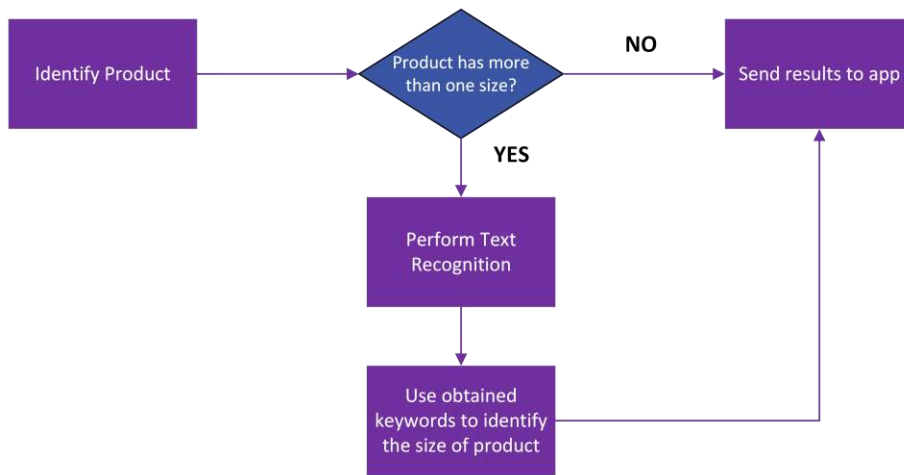


Figure 18 Solution # 2 - Text Recognition

The box will have the following components

The design of the conveyor belt is a secondary part of the project as the box will be attachable to any pre-existing conveyor belt to reduce cost.



Figure 19 All Hardware Components

A 775 100W DC motor (controlled by a 10A 12-40V PWM Controller) will be used for the conveyor belt to prevent unnecessary load on the Nvidia Jetson Nano and to keep the conveyor belt part independent from the box.

4.3.1 Calculations

Following are the significant design calculations needed for this project

- Power Consumption of Attachable Modular Box
- Belt Pull and Power Calculation (For the optional conveyor belt)

a) Power Consumption of Attachable Modular Box

Nvidia Jetson Nano 4GB consumes 10W of power in the default mode and up to 20W of power for peak performance

Assuming Jetson Nano is running at maximum power of 20W for the entire 24 hours. The maximum power consumption would be

$$E = P \times t \dots \text{Wh}$$

Power Consumed per Day

$$= 20\text{W} * 24 \text{ hours}$$

$$= 480 \text{ Wh / Day}$$

$$= 0.48 \text{ kWh / day}$$

Power Consumed per Month

$$= 20\text{W} * 24 \text{ hours} * 30 \text{ days}$$

$$= 14,400 \text{ Wh / Month}$$

$$= 14.4 \text{ kWh / Month}$$

Power Consumed per Year

$$= 20\text{W} * 24 \text{ hours} * 365 \text{ days}$$

$$= 175,200 \text{ Wh / Year}$$

$$= 175.2 \text{ kWh/Year}$$

b) Belt Pull and Power Calculation (For the optional conveyor belt)

For this portion, the calculations are done in metric units

Mass of a standard conveyor belt = 4.5 kg/m

Length of a standard retail conveyor belt = 900 mm or 0.9 m

Belt Speed = 0.25 m/s

Total Mass = 50 kg + 0.9 m * 4.5 kg/m

Total Mass = 50 kg + 4.05 kg

Total Mass = 54.05 kg

Total Weight = 54.04kg * 9.81 N/kg = 530.23 N

Belt Pull = Total Weight * Frictional Coefficient

Belt Pull = 530.23 N * 0.5 (Assuming Frictional Coefficient of 0.5)

Belt Pull = 265.115 N

Required Power = Belt Pull x Belt Speed

Required Power = 265.115 N * 0.25 m/s

Required Power = 66.28 Nm/s

Required Power = (66.28 Nm/s)/(1 Nm/s per Watt)

Required Power = 66.28W

4.3.2 Hardware Design

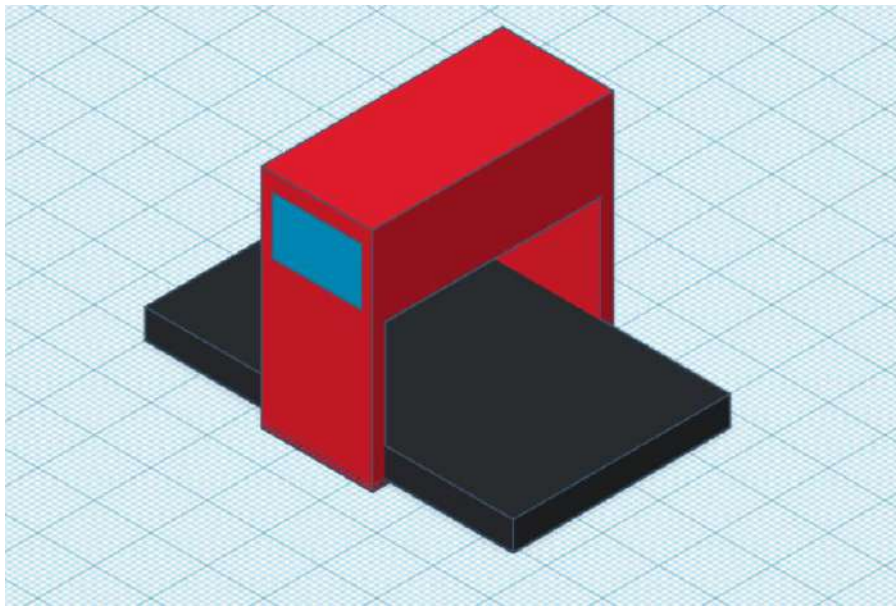


Figure 20 3D Schematic for the Conveyor Belt

4.3.3 Software Design

The following tools and libraries will be used for the software part of this project

- Anaconda
- MMOCR
- YOLOv7
- PyTorch
- FlaskAPI
- HTML
- CSS
- JavaScript
- TensorFlow
- TensorRT

For the model training, Anaconda will be used to set up a Python environment for training. For object detection, YOLOv7 algorithm will be used. For text recognition, MMOCR toolbox [15] will be used. For running the model on the cloud on a desktop, PyTorch can be used. Though the model will be exported as a TensorFlow model specifically a TensorRT model so that it can run

at maximum performance on the Nvidia Jetson Nano. The frontend of the web app will be designed using HTML, CSS, and JavaScript while the backend will be designed using Flask API

4.3.4 Software Implementation:

Training Object Detection Model (YoloV7):

For the training of the complete model of YoloV7, we first shortlisted the products that would be classified by the final model. Those products are:

- Nestle Milo
- National Chilli Garlic Sauce
- Nestle Everyday
- Shangrila Chilli Garlic
- Marie Biscuits
- Saltish Biscuits
- Nescafe 3 in 1 Coffee
- Head and Shoulders
- Euthrix DF
- Kurkure Red Chilli
- Lays Masala
- Slanty Masala
- Mountain Dew
- Pepsi
- Choco Bliss
- Shan Bombay Biryani Mix
- Shan Tandoori Mix
- Perk Chocolate
- Mortein Mosquito Spray
- Ponds Face Wash
- Todays Canned Mixed Fruit
- Scotch Brite
- Tapal Danedar Tea
- Crispo Spaghetti
- Tibet Soap
- Palmolive Soap
- King's Ice Cream Syrup
- Colgate Herbal
- Numberdaar Detergent
- Crispo Pasta

To facilitate the training process and ensure uniformity between the training, testing, and deployment environments, we recorded a video of each product on a black conveyor belt that mirrors the final setup. Capturing the product from various angles generated a robust dataset and mimicked the diverse orientations a product might adopt in real-life scenarios.

Every video was approximately 50 seconds long with a frame rate of 30 frames per second. This yielded approximately 1500 images for each product, producing a comprehensive image library for model training.

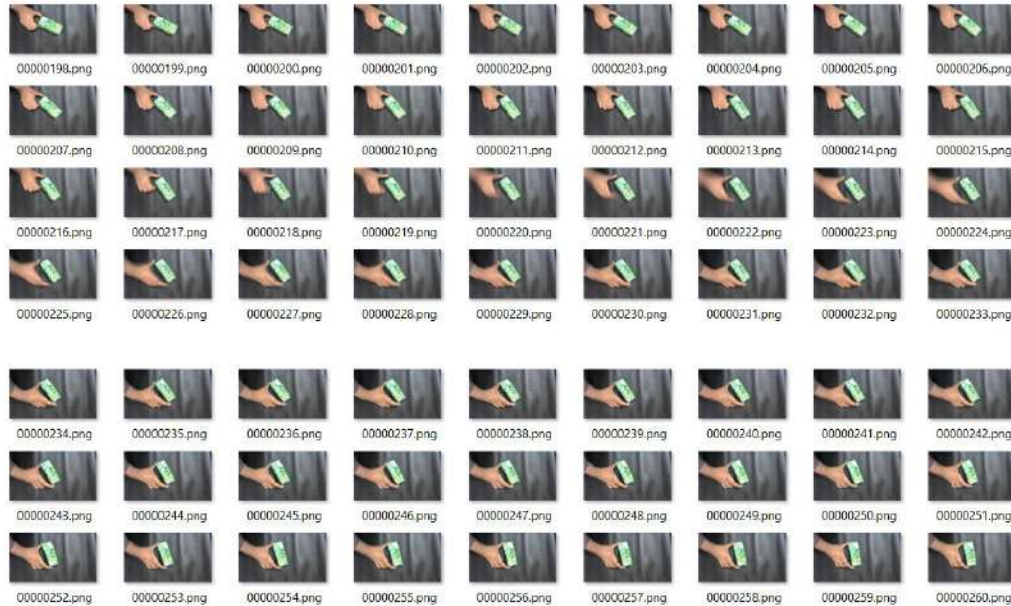


Figure 21 Images for Training

We saved the images using a simple numbering system, ending up with 47,188 images in total. These images took up about 46GB of storage space. Along with each image, we also created a label.txt file. This file had the product index and the details of where the product was located within the image.

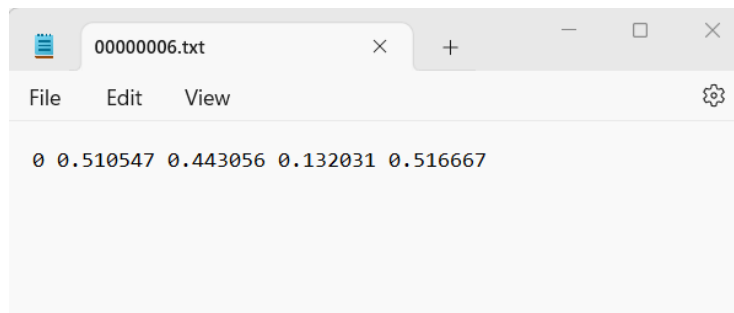


Figure 22 Label File

The training of the model was done using 6GB of GPU memory. We set a limit of 500 epochs, but we found that the model usually performed well after just 10 epochs. On average, each epoch took about an hour and a half to complete.

Once we had trained the model for 5 epochs, we checked its performance using a few different measures. The results of this check are provided below.

Precision = 96.66%

Recall = 98.99%

mAP@0.5 = 99%

The precision of 96.66% implies few false positives, and a high recall of 98.99% indicates that it correctly identifies the majority of object instances. The mAP score of 99% at a 0.5 IoU threshold demonstrates a high level of overlap between the predicted and actual bounding boxes. These numbers suggest that the model performs very well on the dataset used for evaluation. However, it's crucial to verify this performance in real-world environments as well, as results could vary based on the nature and complexity of the data encountered there.

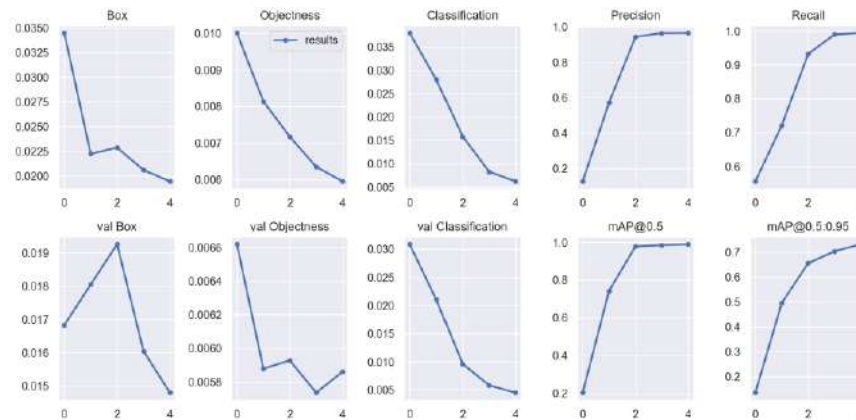


Figure 23 YoloV7 (Regular) Model Metrics

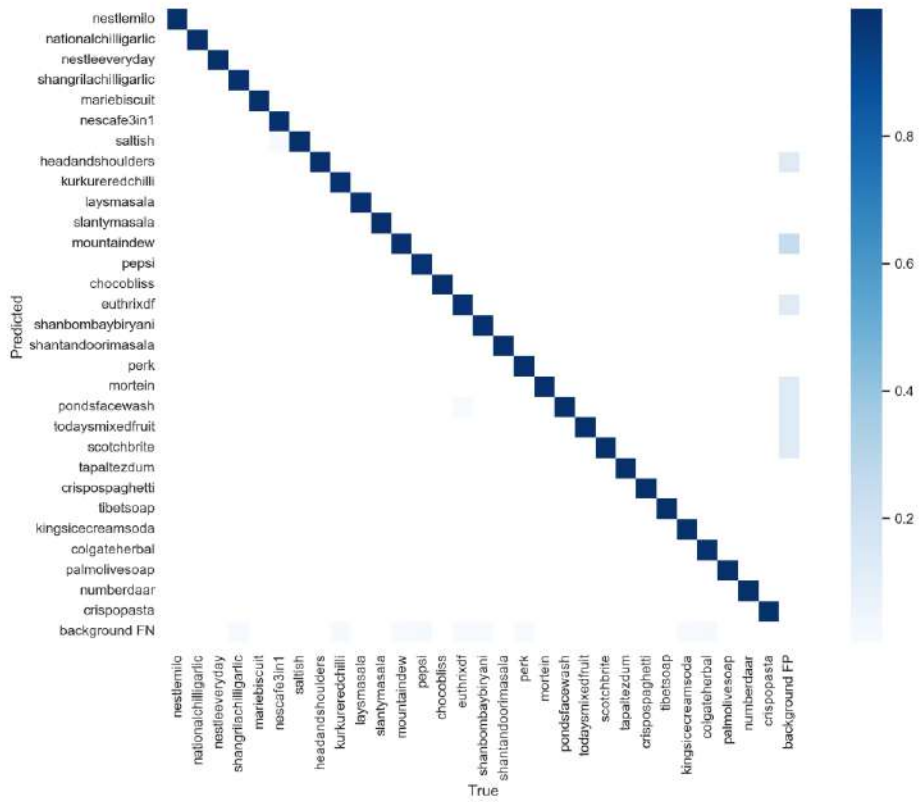


Figure 24 Confusion Matrix for YoloV7 (Regular)



Figure 25 Test Results from YoloV7 (Regular)

Text Recognition:

For text recognition we have used three different algorithms for the sake of comparison and testing:

- **Tesseract:** Developed by Google, Tesseract is a versatile optical character recognition (OCR) engine that supports a wide range of languages. It works exceptionally well with high-quality images and clear text layout. The performance of Tesseract can be further enhanced with preprocessing techniques such as noise removal, skew correction, and binarization [19].
- **EasyOCR:** EasyOCR is a powerful OCR tool that comes with support for over 80 languages, including complex ones like Chinese, Japanese, and Korean. It employs a deep learning-based approach and is relatively easy to use and implement. Despite being robust, it might be slower than traditional OCR tools due to its reliance on deep learning [20].
- **MMOCR:** MMOCR is an open-source OCR toolbox based on PyTorch and MMDetection. It is designed to address several major text detection and recognition scenarios. This tool provides support for a range of cutting-edge text detection and recognition algorithms, making it a versatile option for various OCR tasks [15].

Below are the implemented functions for all three:

Tesseract	EasyOCR	MMOCR
<pre>def perform_ocr_pytest(image): # Perform OCR on the image text = pytesseract.image_to_string(image, config="--psm 6") # Split the text into a list of lines lines = text.split('\n') # Remove any empty lines lines = [line.strip() for line in lines if line.strip()] return lines</pre>	<pre>def perform_ocr_easyocr(image): reader = easyocr.Reader(['en']) results = reader.readtext(image) temp = " for detection in results: text = detection[1] temp = temp + text.strip() # this will strip spaces at the end print(temp.upper(), end="") return temp</pre>	<pre>def perform_ocr_mmocr(image): # Load the OCR model results = ocr.readtext(image, print_result=True, imshow=False) # Extract the recognized text from the results return [result['text'] for result in results]</pre>

Flask Backend:

Method 1 (Local Processing):

This method performs local processing and requires the device to be capable of handling the processing at relatively fast speeds for proper real-time classification.

First let's talk about the imports and the initializations

- Various packages and modules are imported that are required to run the web application and process the images from the webcam.
- The MMOCR module is loaded with specific parameters for detection and recognition.
- The variables COLORS is defined for drawing bounding boxes with distinct colors.
- The intersect() and ccw() functions are defined for mathematical calculations on the coordinates.
- The get_output_fps_height_and_width() function is defined to get the height, width, and frames per second of the webcam feed.
- The _convert_detections_into_list_of_tuples_and_count_quantity_of_each_label() function is defined to convert the output of the object detection model into a list of tuples and count the quantity of each label detected.
- The perform_ocr_pytest(), perform_ocr_easyocr(), and perform_ocr_mmocr() functions are defined to perform OCR on an image using different OCR tools.
- The Flask web app and the Turbo-Flask extension are initialized.
- The names variable is set up to contain all the possible labels that the object detection model can detect.
- A list data is created, where each element is a list that contains the product ID, name, quantity, and price. This will later be displayed on the web page.
- The video capture object is created to capture video from the webcam.
- The PyTorch object detection model is loaded from a local file.
- The SORT tracker is initialized with specified parameters.
- The DETECTION_FRAME_THICKNESS, OBJECTS_ON_FRAME_COUNTER_FONT, OBJECTS_ON_FRAME_COUNTER_FONT_SIZE, LINE_COLOR, LINE_THICKNESS, LINE_COUNTER_FONT, LINE_COUNTER_FONT_SIZE, and LINE_COUNTER_POSITION variables are defined for drawing and displaying on the video frame.
- The extract_unit_price() function is defined to extract the price from a string.

Now let's talk about the Web Application Routes themselves:

The '/' route is defined to render the main page of the web application. It uses a template from Flask, passing in the headings and data for the table to be displayed on the page.

```

@app.route('/')
def index():
    return render_template('index.html',t_headings=headings,t_data=data)

def gen(video):

```

The gen(video) function is defined. It is responsible for processing the webcam feed frame by frame. Here's what it does in detail:

It captures a frame from the webcam and converts the color space from BGR to RGB.

```

    counter=0
    memory = {}
    dets_d = []
    while True:
        success, image = video.read()

        frame = image[:, :, [2,1,0]]
        frame = Image.fromarray(frame)
        frame = cv2.cvtColor(np.array(frame), cv2.COLOR_RGB2BGR)

```

The object detection model is run on the frame, resulting in a set of bounding boxes and class labels for detected objects.

```

    detections = model(frame,size=640)
    detv2 = detections.pandas().xyxy[0].to_dict(orient="records")

    #pass an empty array to sort
    dets_to_sort = np.empty((0,6))

    for det in detv2:
        x1 = int(det['xmin'])
        y1 = int(det['ymin'])
        x2 = int(det['xmax'])
        y2 = int(det['ymax'])
        conf = float(det['confidence'])
        cla = det['class']

        dets_to_sort = np.vstack((dets_to_sort, np.array([x1, y1, x2, y2,
conf, cla])))
        #print(dets_to_sort)

```

The detections are passed to the SORT tracker, which assigns each detection a unique ID and tracks it across frames. The Simple Online and Realtime Tracker (SORT) is a tracking method that uses Kalman filtering and Hungarian assignment algorithm. It is designed to track objects in a video as they move from frame to frame. This allows each object to be tracked individually and consistently.

```
# Run SORT
tracks = sort_tracker.update(dets_to_sort)

boxes = []
indexIDs = []
labels = []
previous = memory.copy()
memory = {}
```

For each tracked object, a bounding box is drawn on the frame, and the object is checked for intersection with a specified line (the "counter line").

```
if len(boxes) > 0:
    i = int(0)
    for box in boxes:
        (x, y) = (int(box[0]), int(box[1]))
        (w, h) = (int(box[2]), int(box[3]))

        color = [int(c) for c in COLORS[indexIDs[i] % len(COLORS)]]
        cv2.rectangle(frame, (x, y), (w, h), color,
DETECTION_FRAME_THICKNESS)

        if indexIDs[i] in previous:
            previous_box = previous[indexIDs[i]]
            (x2, y2) = (int(previous_box[0]), int(previous_box[1]))
            (w2, h2) = (int(previous_box[2]), int(previous_box[3]))
            p0 = (int(x + (w - x) / 2), int(y + (h - y) / 2))
            p1 = (int(x2 + (w2 - x2) / 2), int(y2 + (h2 - y2) / 2))
            cv2.line(frame, p0, p1, color, 3)
```

If an object intersects the counter line, the counter is incremented and the OCR functions are run on the cropped area of the frame that contains the object.

```
if intersect(p0, p1, (250,500), (250,0)):
    counter += 1
    cid = int(result['class'])

    # =====OCR
HERE===== #

    cropped_img = frame[y:y+h, x:x+w]
```

```

# Apply Gaussian blur to the cropped image
blur_img = cv2.GaussianBlur(cropped_img, (5, 5), 0)

# Apply Otsu thresholding to the blurred image
gray_img = cv2.cvtColor(blur_img, cv2.COLOR_BGR2GRAY)
thresh_img = cv2.threshold(gray_img, 0, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]

ocr_texts_pytest = perform_ocr_pytest(thresh_img)
ocr_texts_easyocr = perform_ocr_easyocr(cropped_img)
ocr_texts_mmocr = perform_ocr_mmocr(cropped_img)
print(ocr_texts_pytest)
print(ocr_texts_easyocr)
print(ocr_texts_mmocr)

ocr_texts = ocr_texts_mmocr

#Match OCR text with the hard-coded list
best_match = ""
highest_similarity = 0
best_match_index = -1

for index, ref_text in enumerate(ocr_results):
    #Calculate the average similarity across all detected
texts
    total_similarity = 0
    avg_similarity = 0
    for ocr_text in ocr_texts:
        similarity = SequenceMatcher(None, ocr_text,
ref_text).ratio()
        total_similarity += similarity

    if len(ocr_texts) != 0:
        avg_similarity = total_similarity /
len(ocr_texts)

    if avg_similarity > highest_similarity:
        highest_similarity = avg_similarity
        best_match = ref_text
        best_match_index = index

```

The OCR results are compared with a predefined list of possible results to find the best match. If a match is found, the corresponding counter in the data list is incremented.

```

data[cid][2] += 1
text2 = names[labels[i]]

```

The function yields the frame as a byte stream, which can be displayed on the web page.

Front-end Web App

```

ret, jpeg = cv2.imencode('.jpg', frame)

frame = jpeg.tobytes()

yield (b'--frame\r\n'
       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

```

Then a client token is generated which is required by the Braintree client SDK. This client token is then passed to the checkout.html page where it's used to setup the Braintree client instance and to tokenize payment information.

```

# Create a route for the checkout form
@app.route('/checkout', methods=['GET', 'POST'])
def checkout():
    # Generate a client token using the Braintree Python SDK
    client_token = braintree.ClientToken.generate()

    # Render the checkout form using Flask templating
    return render_template('checkout.html', client_token=client_token)

```

when a payment is initiated from the client (typically a form submission), the server receives a POST request at the /process_payment endpoint. The server then receives the payment method nonce, which is a one-time-use reference to payment information provided by the customer.

Using the Braintree Python SDK [21], a new transaction is made with the braintree.Transaction.sale function, which initiates the transaction. The amount to charge is calculated as total_price * 0.0035. The "submit_for_settlement": True option automatically submits the transaction for settlement, meaning the payment will be transferred to your account.

```

# Handle the payment request when the checkout form is submitted
@app.route('/process_payment', methods=['POST'])
def process_payment():
    global total_price
    # Get the payment method nonce from the request
    nonce = request.form['payment_method_nonce']

    # Create a transaction using the Braintree Python SDK
    result = braintree.Transaction.sale({
        "amount": total_price*0.0035,
        "payment_method_nonce": nonce,

```



```

        "options": {
            "submit_for_settlement": True
        }
    })

```

Once the transaction is attempted, the application checks the result of the transaction. If the transaction was successful (`result.is_success`), the user is redirected to a success page (`success.html`). If the transaction was not successful, the user is shown a message with the failure reason.

```

    # Handle the result of the payment request and provide feedback to the
    customer
    if result.is_success:
        return redirect(url_for('success'))
    else:
        return "Payment failed: %s" % result.message

```

Method 2 (Client-Server):

This method sends the webcam stream from the client to the server for processing and then the results are sent back to the client

First the webcam feed is captured. This could be done through various ways, but for web applications, the `MediaDevices` API is typically used. This API provides access to connected media input devices like cameras and microphones.

```

const video = document.getElementById("webcam");
const processed = document.getElementById("processed");

// Get the webcam feed
navigator.mediaDevices.getUserMedia({ video: true })
    .then(stream => {
        video.srcObject = stream;
    });

```

The client application then sends this feed to a server where the processing will occur.

```

@app.route('/process_frame', methods=['GET', 'POST'])
def handle_process_frame():
    frame_data = request.json["frame"]

    # Decode the base64 frame to a numpy array
    nparr = np.frombuffer(base64.b64decode(frame_data), np.uint8)

    # Convert the numpy array to an OpenCV image (BGR format)

```

```

frame = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

processed_image = process_frame(frame)

```

On the server-side, the received feed is processed (passed through the process(frame) function where it classifies the products in the frame and sends back the results)

```

def process_frame(frame_data):

    counter=0
    memory = {}
    dets_d = []

    frame = cv2.cvtColor(np.array(frame_data), cv2.COLOR_RGB2BGR)

```

Once the processing is complete, the server then needs to send the results back to the client. Depending on the application, this could be the processed images themselves, or some kind of data derived from the processing, like object detection results.

```

return frame

# Encode the processed frame back to base64
retval, buffer = cv2.imencode('.jpg', processed_image)
jpg_as_text = base64.b64encode(buffer).decode("utf-8")

return jsonify({"frame": jpg_as_text})

```

On the client side, these results are received and then rendered appropriately. If the results are images, they can be displayed directly. If the results are some form of data, the client application would then handle this data and display it appropriately, perhaps overlaying it on the original webcam feed or displaying it in another part of the UI.

```

// Send the frame to the server periodically
setInterval(async () => {
    const canvas = document.createElement("canvas");
    canvas.width = video.videoWidth;
    canvas.height = video.videoHeight;
    const ctx = canvas.getContext("2d");
    ctx.drawImage(video, 0, 0);
    const frameData = canvas.toDataURL("image/jpeg").split(",")[1];

    const response = await fetch("/process_frame", {
        method: "POST",
        headers: {

```

```

        "Content-Type": "application/json"
    },
    body: JSON.stringify({ frame: frameData })
  });
  const data = await response.json();
  processed.src = "data:image/jpeg;base64," + data.frame;
}, 100); // Adjust the interval duration as needed

```

This Client-Server method for processing webcam streams is useful when the client-side device is not powerful enough to do the necessary processing, or when the processing involves proprietary or confidential algorithms that you don't want to expose on the client side. However, it does require a stable and relatively fast internet connection, as the webcam feed needs to be sent to the server and the processed results returned in real time.

Front-End (UI):

The user interface of this web app is designed using html, CSS and JavaScript

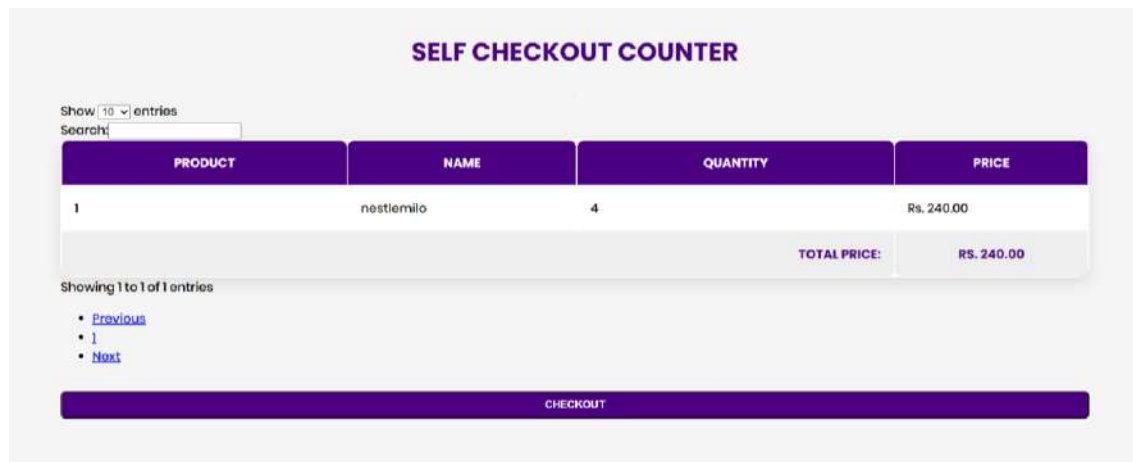


Figure 26 UI of the Web-App

Our webpage features a table with four columns: Product Number, Name, Quantity, and Price. Each row in the table represents a product in the user's shopping cart, with details about it. At the end of the table, we calculate and display the total price of all products in the cart.

In addition to the product table, the bottom of the page hosts a real-time webcam stream. This enables users to observe their surroundings while using the app.

Once the user is ready to complete their purchase, they can press the "Checkout" button located near the total price. Pressing this button will lead them to our secure payment gateway, finalizing the shopping process.

4.3.5 Hardware Implementation:

Conveyor Belt and Modular Box:

The system utilizes a 100cm long and 30cm wide conveyor belt, featuring an attached box with open entry and exit points. This box is the housing site for all operational components, ensuring their secure placement and easy maintenance. For seamless interaction, a 7-inch touchscreen is mounted on the side of the box, providing real-time system control and data display. This compact and efficient design ensures effortless integration into existing processes.



Nvidia Jetson Nano (Single Board Computer):

The Nvidia Jetson Nano serves as the central processing unit of the self-checkout system. This compact, yet powerful single board computer (SBC) is capable of running multiple neural networks in parallel, making it ideal for image recognition tasks.

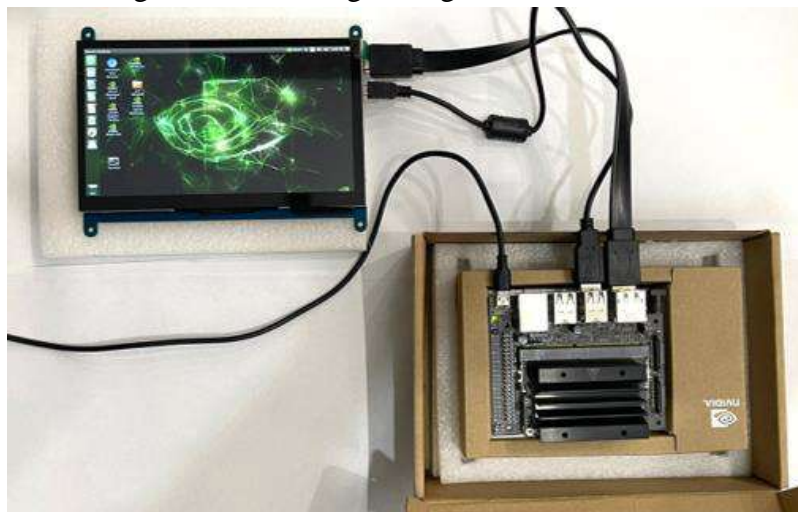


Figure 27 - Nvidia Jetson Nano Attached to a Screen

Camera:

The camera used in this system is the Logitech C310 Webcam. This device is well-regarded for its high-resolution image capturing capabilities, which is critical for the accurate functioning of the system. The Logitech C310 provides a clear, real-time video feed to the Nvidia Jetson Nano, which then identifies the products. Its ease of use and reliability make it a good fit for this application.



Figure 28 - Logitech C310 Webcam

Touchscreen:

A crucial part of the user interface is the 7-inch touchscreen. It serves as a point of interaction for the customers, allowing them to view their selected items, total cost, and complete the checkout process. The screen displays real-time information as the customer places items on the conveyor, offering a dynamic and intuitive self-checkout experience. Its compact size ensures it fits neatly into the system design, while still being large enough for comfortable use.



Figure 29 7-inch touchscreen for Jetson Nano

10A 12-40V PWM Controller:



Figure 30 10A 12-40V PWM Controller

The PWM (Pulse Width Modulation) Controller DC Motor Speed Controller is a vital component for controlling the speed of DC motors in a precise and efficient manner. This controller is designed to handle a working voltage between 12V and 40V DC and is capable of controlling power ranging from 0.01 to 400W, making it highly versatile for a variety of applications.

The static current of the controller is extremely low, at just 0.02A, which helps to reduce power consumption and improve overall efficiency. The speed control range is also wide, from 10% to 100%, offering significant flexibility in controlling the motor speed according to our needs.

The PWM frequency of this controller is 13 kHz. The higher frequency allows for more precise control over the motor's speed, and it also reduces the risk of the motor experiencing audible noise or vibration.

One key feature of this controller is its robust safety mechanisms. It is equipped with a fuse tube that protects the speed controller from burnout caused by short circuits. It also includes a reverse connection protection function that guards against damage from incorrectly wired connections.

The design of this speed controller includes convenient screw terminals that provide a straightforward means for connecting wires, enhancing its ease of use and reducing the time required for setup.

To integrate this into our self-checkout system, the DC Motor Speed Controller could be used to regulate the speed of the conveyor belt. By adjusting the PWM signals, we can control the conveyor belt to move at the desired pace, ensuring smooth operation of the system.

Hardware Assembly Process:

The assembly process of our self-checkout system involves several key steps, with each component carefully installed to ensure a cohesive, effective, and user-friendly solution.

1. Establishing the Base Framework:

Our assembly began with the construction of the base structure. This structure forms the backbone of our self-checkout system, providing essential support and positioning for the conveyor belt, box housing for system components, and the touchscreen interface.



Figure 31 Detachable Box



Figure 32 Base for the Conveyor Belt



Figure 33 Complete Structure



Figure 34 Painting Process of Conveyor Belt



Figure 35 Painting of Detachable Box

2. Conveyor Belt Installation:

Following this, we secured the conveyor belt onto the base framework. We paid close attention to ensuring the belt was taut and could run smoothly, an essential characteristic for effective product transportation. The conveyor belt's motion was powered by a DC motor, controlled by a PWM Controller DC Motor Speed Controller 12V-40V 10A.



Figure 36 Testing of Conveyor Belt

3. Box Housing Setup:

Next, we attached the box housing to the conveyor belt system. This housing was carefully designed to securely contain the critical components of our self-checkout system, while allowing for easy maintenance and potential future upgrades.



Figure 37 Box Housing

4. NVIDIA Jetson Nano Installation

With the box housing in place, we installed the NVIDIA Jetson Nano. This single-board computer forms the processing hub of our self-checkout system.



Figure 38 Configuring Jetson Nano

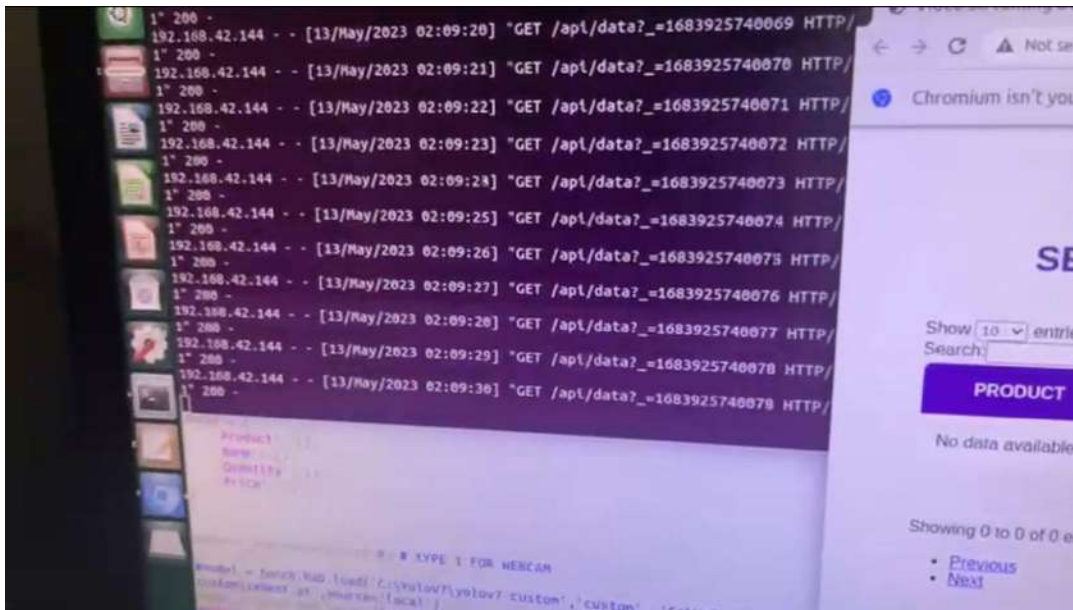


Figure 39 Testing the System on Jetson Nano

5. Camera Setup

We then mounted the Logitech C310 webcam, strategically positioning it for optimal product viewing and identification. The webcam, connected to the NVIDIA Jetson Nano, provides crucial visual data for the system to identify and price products correctly.



Figure 40 - Camera Installation

6. Touchscreen Installation

Next, we installed the 7-inch touchscreen onto the side of the box housing. This screen provides an intuitive interface for user interaction and was placed for comfortable, easy access. As with the webcam, this was connected to the NVIDIA Jetson Nano.



Figure 41 Screen Installation & Testing

7. Payment Gateway Setup

On the software side, we made sure the payment gateway was correctly integrated with the interface displayed on the touchscreen. This gateway provides the necessary infrastructure for processing user payments. (More on this in Chapter 5)

8. Software Installation

We installed all necessary software on the NVIDIA Jetson Nano. This included the image processing software and the software required to control the conveyor belt and process payments.

9. Testing

With all components installed and software in place, we conducted comprehensive testing to ensure the system functioned as intended. We tested the conveyor belt's smoothness, the camera's accuracy in identifying products, the touchscreen's response to user input, and the payment processing functionality. These tests are vital for ensuring system reliability and performance.



Figure 42 Testing the System

In conclusion, our assembly process was methodical and meticulous, ensuring each part of our self-checkout system worked harmoniously with the rest. This process, combined with rigorous testing, gave us confidence in the system's functionality and reliability.

Chapter 5: Investigation and Testing

5.1 Camera and Object Recognition Test:

Procedure: Set up a variety of products in front of the Logitech C310 Webcam and let the system identify the products. Use different lighting conditions and orientations to test the robustness of the model.

Expected Result: The system should accurately recognize and classify the products regardless of orientation.

Actual Result:

The system was largely successful in accurately recognizing and classifying the products under various orientations. Both the front and back sides of individual products were tested to ensure robust identification. The system showed an impressive ability to discern the unique features of each product, correctly classifying them even under different lighting conditions.

When testing multiple products simultaneously, the system managed to separate and identify individual items effectively. This affirms the capacity of our self-checkout system to handle real-world scenarios where customers may place several items onto the conveyor belt at once.

Despite the largely successful outcomes, it's important to note that the recognition model's accuracy may slightly vary depending on factors such as the similarity of product packaging and the clarity of the product labels. These minor challenges will be addressed as part of our future work to refine and improve the system.

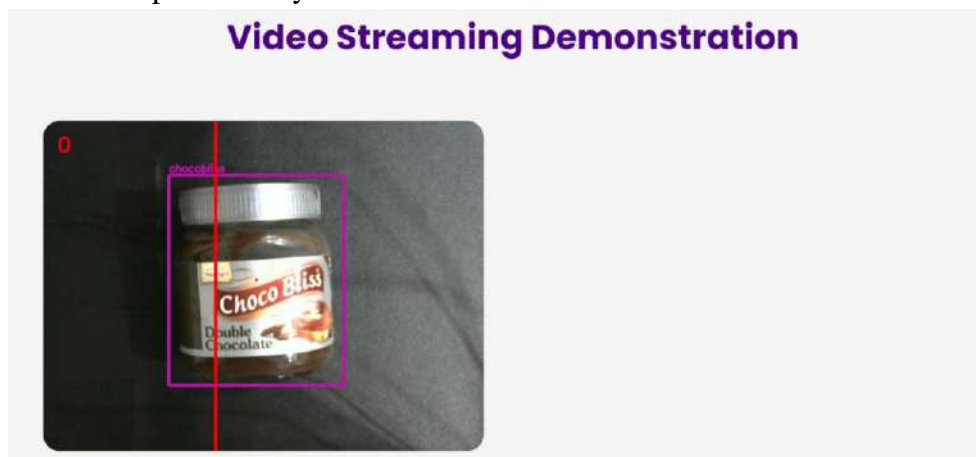


Figure 43 Testing on Choco Bliss

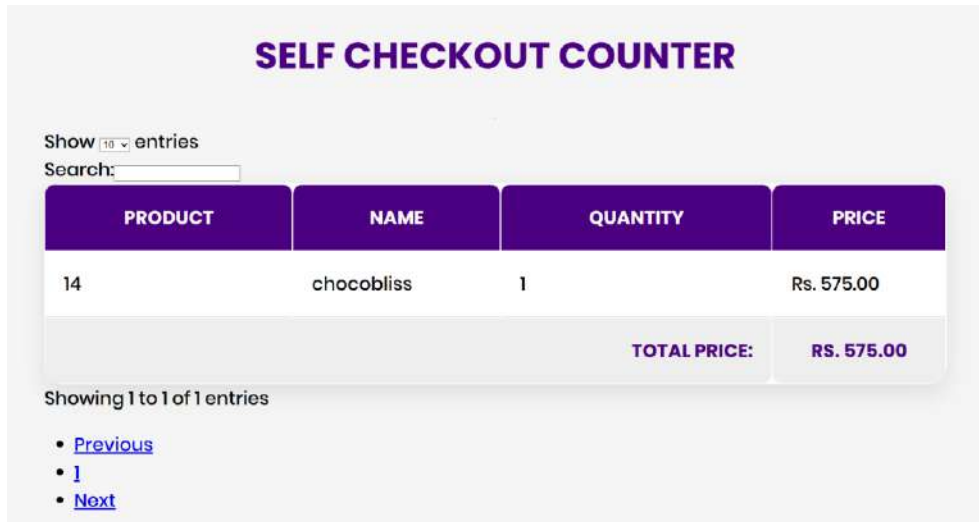


Figure 44 Choco Bliss displayed on Counter

Table 2 OCR Results

Results from PyTesseractOCR	['f a', 'ar 3']
Results from EasyOCR	DoubleChocolateBisChoco_
Results from MMOCR	[['chocolate', 'double', 'choco', 'blss']]

In the implementation of Optical Character Recognition (OCR) for our self-checkout system, we employed and tested three different OCR engines: PyTesseract, EasyOCR, and MMOCR. Our comparative analysis revealed distinct performance characteristics and levels of effectiveness for each.

PyTesseract, while recognized for its ease of use, unfortunately underperformed in our tests. It demonstrated less accuracy in character recognition compared to the other two OCR engines, making it less suitable for the precise requirements of our project.

Both EasyOCR and MMOCR showed significantly better performance, often running neck-and-neck in their capabilities. They successfully recognized and translated printed text in many scenarios, proving to be more reliable for our application. However, there were specific

situations where each showed strengths and weaknesses, and no clear winner emerged from these close ties.

Ultimately, we identified MMOCR as the superior option, despite it being the slowest among the three. Its slower speed is offset by its impressive accuracy and consistency in character recognition. It managed to outperform the other engines in complex recognition scenarios, solidifying its position as the most suitable OCR engine for our project.

Nonetheless, the slower operation speed of MMOCR is a trade-off that we acknowledge. In our future work, we aim to find ways to optimize its speed without compromising the quality of its OCR capabilities.

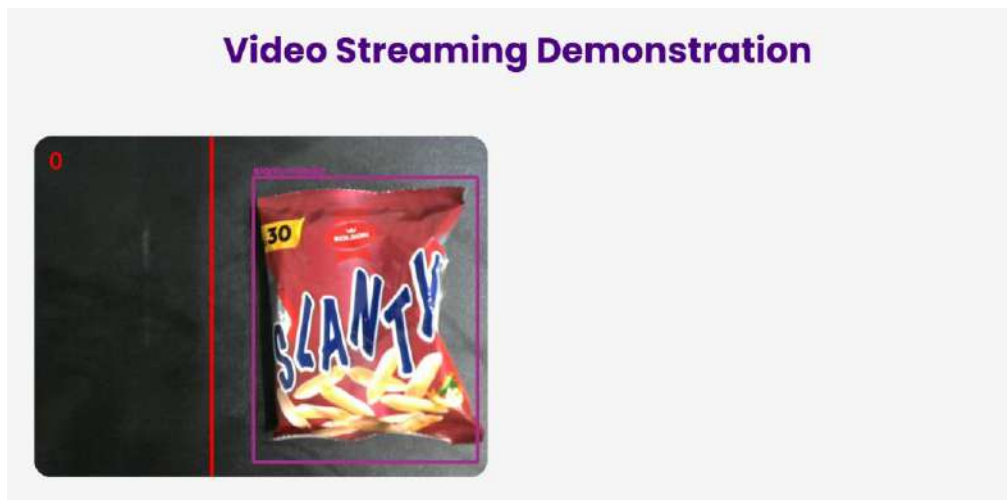


Figure 45 Testing on Slanty

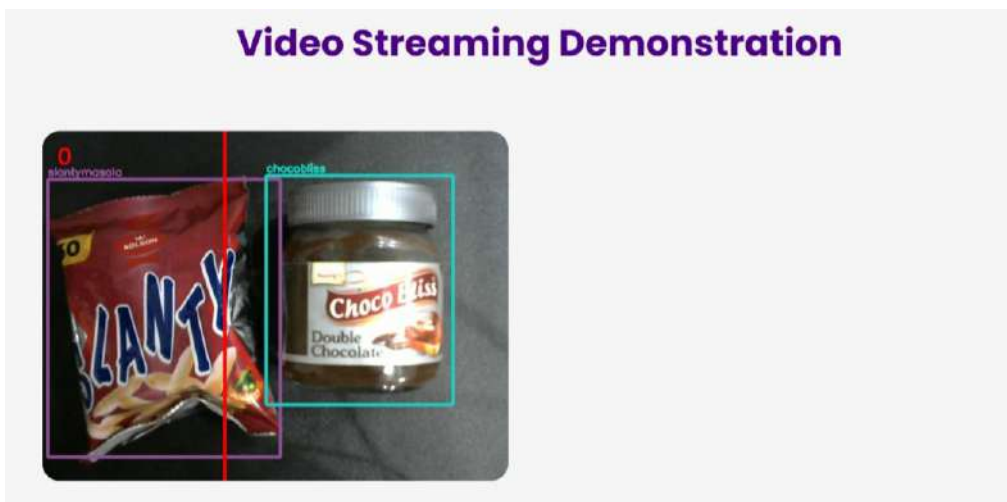


Figure 46 Testing on 2 products at once

Video Streaming Demonstration



Figure 47 Testing on Head & Shoulders from the Front

Video Streaming Demonstration



Figure 48 Testing on Head & Shoulders from the Back

Video Streaming Demonstration



Figure 49 Testing on Euthrix DF

Video Streaming Demonstration



Figure 50 Testing on Shangrila Chilli Sauce

Video Streaming Demonstration

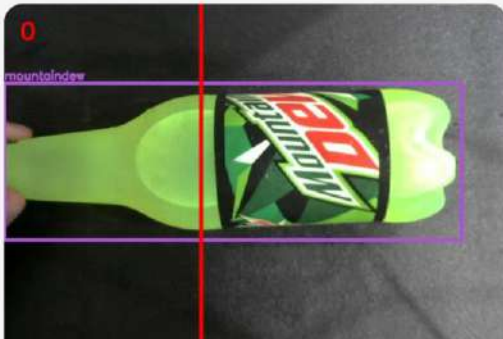


Figure 51 Testing on Mountain Dew

5.2 Performance between Local Processing and Client-Server Method:

Procedure:

The performance of Local Processing and Client-Server Method was measured using two primary factors: time taken for processing and accuracy of the results. Both methods were tested with the same set of data to ensure fairness in comparison.

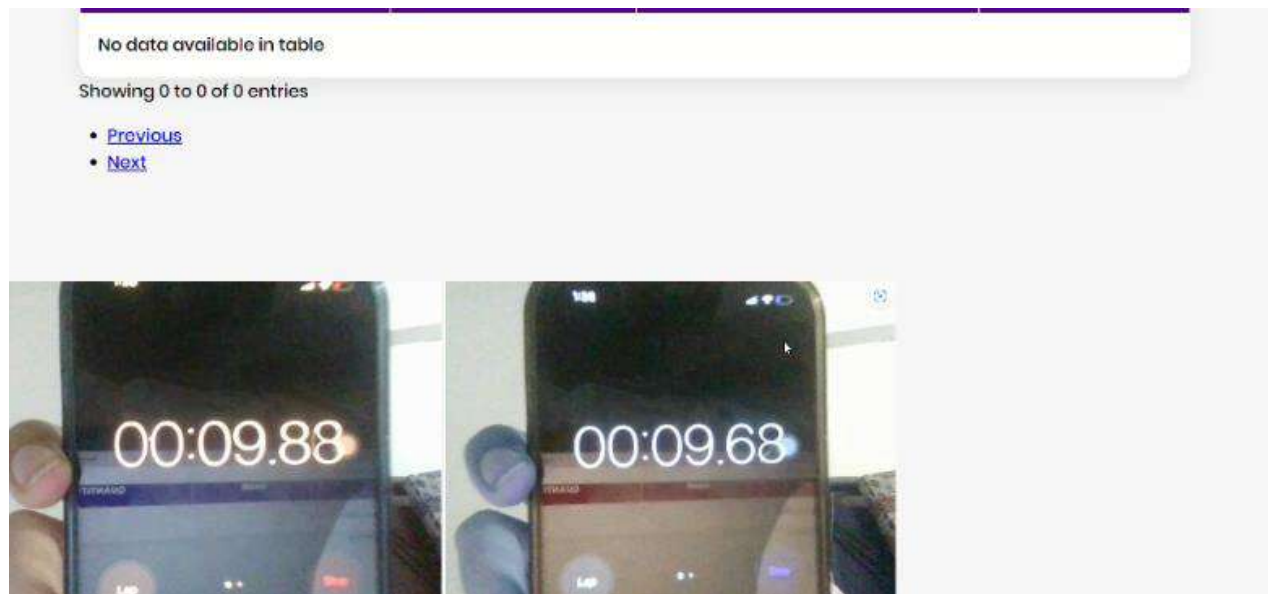
Expected Result:

The expected result was that Local Processing would be faster due to the absence of data transfer times as compared to the Client-Server method. However, we anticipated a possible trade-off in the accuracy of results, as the processing power of a local system might be lower compared to a dedicated server.

Actual Result:

Local Processing: The local processing method provided faster results as there was no delay caused by data transfer between the client and the server. However, it was observed that the accuracy of results was slightly less consistent due to the varying processing capabilities of local systems (i.e. very poor frame rates on Jetson Nano but higher frames on a RTX 3060 GPU)

Client-Server Method: This method was slightly slower (0.2 seconds slower) due to the data transfer times, but it was observed that it consistently produced more accurate results because of the higher processing capabilities of the server.



In conclusion, the Local Processing method proved to be faster, as expected, but the Client-Server method offered more reliable results. The choice between the two methods would therefore depend on whether speed or accuracy is the higher priority for the specific use case of the self-checkout system.

5.3 User Interface and Touchscreen Test:

Procedure: Interact with the 7-inch touchscreen interface to test its responsiveness and functionality. Try completing the checkout process.

Expected Result: The touchscreen should respond accurately to input, and the user interface should update correctly based on user interactions.

Actual Result: During the testing phase, the 7-inch touchscreen interface demonstrated a high degree of responsiveness and seamless functionality. The checkout process was executed successfully using the touch input, further emphasizing the efficacy of the interface. The system accurately registered touches and the user interface updated accordingly in real-time, facilitating an intuitive and user-friendly shopping experience.

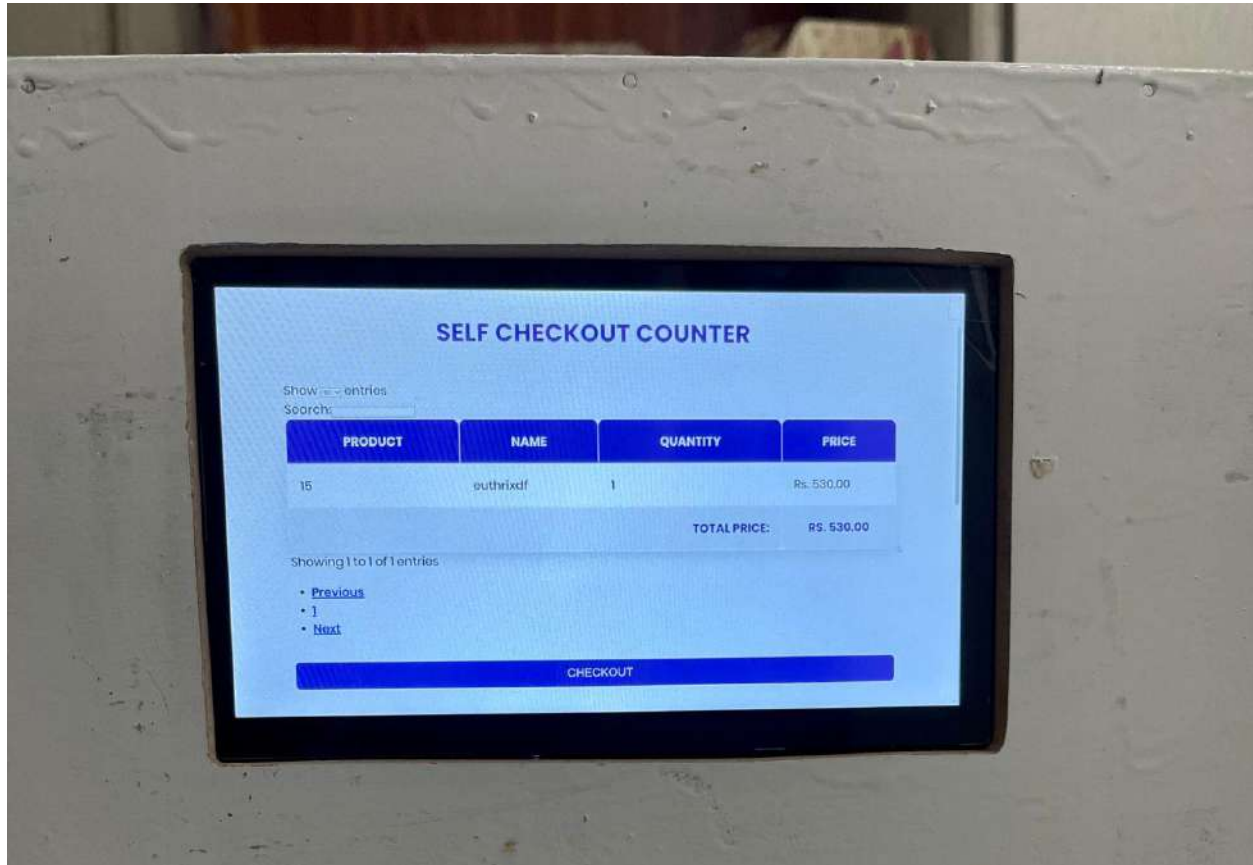


Figure 52 Touch Screen Interface

5.4 Complete System Test:

Procedure: Simulate a full shopping experience. Place multiple items on the conveyor, let the system recognize them, add them to the cart, and complete a transaction.

Expected Result: The complete system should function as expected, recognizing and adding items to the cart, displaying correct totals, and processing the payment successfully.

Actual Result:

Upon simulating a full shopping experience, the complete system functioned seamlessly and demonstrated its effectiveness. A variety of items were placed on the conveyor and were

successfully recognized by the system. These items were then automatically added to the shopping cart. The system effectively maintained an accurate running total of the cost of the items. The total calculated for the items, which amounted to Rs. 2910 or \$10.21, was accurately displayed. The checkout process was then completed with the system processing the payment transaction successfully. This comprehensive test further attests to the efficiency and reliability of the system in a real-world shopping scenario.

PRODUCT	NAME	QUANTITY	PRICE
4	shangrilachilligarlic	1	Rs. 210.00
8	headandshoulders	1	Rs. 1500.00
11	slantymasala	1	Rs. 30.00
12	mountaindew	1	Rs. 65.00
14	chocobliss	1	Rs. 575.00
15	outhrixdf	1	Rs. 530.00
TOTAL PRICE:			RS. 2910.00

Figure 53 All checkout items in the cart

Amount	Type	Status
\$10.21 USD	Sale	✓ Submitted For Settlement

Figure 54 Payment Processed for the Order

5.5 Payment Gateway Test:

Procedure: Simulate transactions using the payment gateway

Expected Result: The payment gateway should successfully process transactions of varying amounts.

Actual Result:

The system was subjected to transactions of varying amounts, each processed smoothly without errors. Furthermore, the transaction details were accurately reflected. This thorough test affirmed the system's proficiency in providing secure and seamless transactions for its users, hence enhancing the shopping experience.

Figure 55 Entering Payment Information

Figure 56 Payment Successfully Completed

Amount	Type	Status	ID	Customer	Payment Method
\$5.25 USD	Sale	Submitted For Settlement	7qbzbf3s		Mastercard 555555*****4444
\$10.21 USD	Sale	Submitted For Settlement	mtdyt5hm		Mastercard 555555*****4444
\$530.00 USD	Sale	Submitted For Settlement	qndzj5fz		Mastercard 555555*****4444
\$530.00 USD	Sale	Submitted For Settlement	pmbzzmyq		Mastercard 555555*****4444
\$1.85 USD	Sale	Submitted For Settlement	3x3zm162		VISA 411111*****1111
\$1.85 USD	Sale	Settling	cr81rsbm		Mastercard 555555*****4444
\$1.85 USD	Sale	Settling	jmpjjatr		Mastercard 555555*****4444
\$3.71 USD	Sale	Settling	037pxyra		Mastercard 555555*****4444
\$1.85 USD	Sale	Submitted For Settlement	4vpm1597		Mastercard 555555*****4444

Figure 57 Payment History

Chapter 6: User Guide

Safety Precautions:

To ensure safe usage of our Self-Checkout System, it's important to remember the following precautions:

1. **Stay Clear of Moving Parts:** The conveyor belt is a moving part. Make sure to keep hands, clothing, or any foreign objects away from it while it's in operation.
2. **System Stability:** Place the Self-Checkout System on a flat, stable surface to prevent it from tipping or falling.
3. **Avoid Liquids:** The system contains electrical components that can be damaged by liquid. Keep drinks and other liquids away, especially near the touchscreen and other system internals.

Getting Started:

Here's a step-by-step guide to using your Self-Checkout System:

Step 1: Setting Up the Self-Checkout System

- Plug the system into an electrical outlet using the provided power cord.
- Allow a few moments for the system to boot up. The user interface will display on the 7-inch touchscreen.

Step 2: Scanning Items

- Place the item to be scanned on the conveyor belt at the designated entry point. You can check the live video feed by scrolling down on the app under “Video Streaming Demonstration”.

Video Streaming Demonstration



Figure 58 Video Streaming Demonstration

- The Logitech C310 Webcam, pointed at the conveyor belt, will capture the image of the product. The system's AI will then use image recognition to identify the product and pull the corresponding price from the database.
- The item details, including its name and price, will be displayed on the touchscreen.

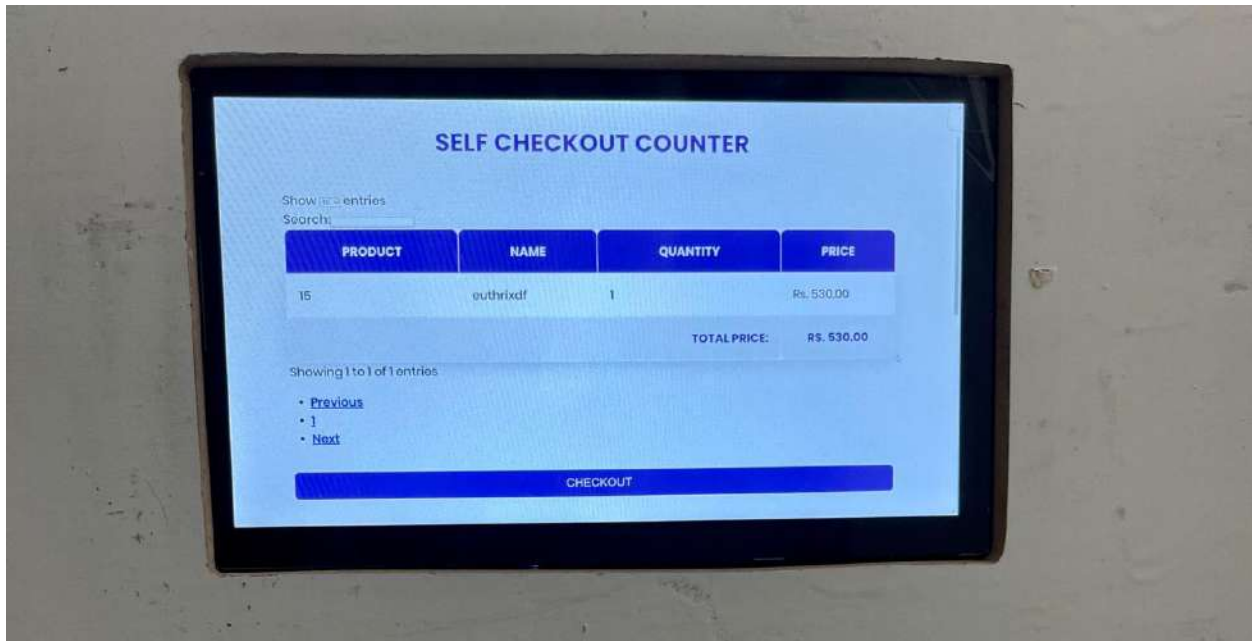


Figure 59 Checkout Counter

Step 3: Reviewing Your Cart

- The touchscreen display provides an easy-to-read list of all scanned items, including the quantity of each item and their respective prices.
- Once you've verified all items in your cart, tap 'Proceed to Checkout' on the touchscreen.

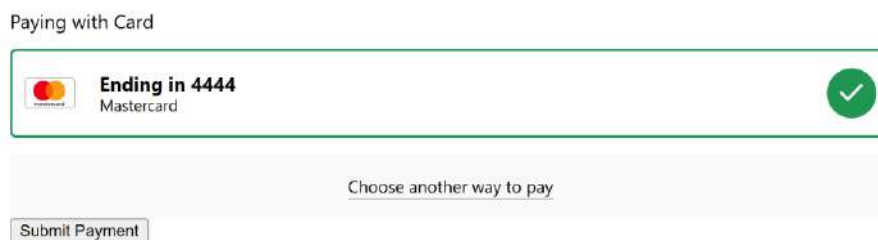
Step 4: Making Payment

- After you've confirmed your cart and tapped 'Proceed to Checkout', the system will display the payment gateway.
- Here, you can choose your preferred payment method and enter the necessary details.



The screenshot shows a 'Pay with card' interface. At the top, there are logos for VISA, Mastercard, AMEX, JCB, and DISCOVER. Below these, there are two input fields: 'Card Number' containing '5555 5555 5555 4444' and 'Expiration Date (MM/YY)' containing '03 / 39'. A 'Submit Payment' button is located at the bottom left of the form.

- The system uses a secure, encrypted process to ensure the safety of your information.
- Once the payment goes through, a This message will appear on the screen.



The screenshot shows a confirmation screen titled 'Paying with Card'. It features a green-bordered box containing the Mastercard logo, the text 'Ending in 4444 Mastercard', and a green checkmark icon. Below this box is a link that says 'Choose another way to pay' and a 'Submit Payment' button at the bottom left.

Step 5: Troubleshooting

- If an item isn't recognized when placed on the conveyor belt, try repositioning or replacing the item. If the issue persists, try manually entering the item's details.
- If you encounter a system error or if the conveyor belt malfunctions, turn off the system immediately. Contact the technical support number provided in the package for further assistance.

- In case of a malfunctioning display, ensure that all the associated cables are properly connected. Specifically, pay close attention to the micro-USB and HDMI cables, verifying that they are securely inserted and that their connections are intact.

Step 6: System Shutdown

- To turn off the system, tap the 'Shutdown' option on the touchscreen interface.
- Allow the system to fully power down before unplugging it from the electrical outlet. This ensures that all system processes have been correctly halted and helps extend the life of the system.

Chapter 7: Deliverables and Cost

7.1 Deliverables

The major deliverables are:

- 1.1. Product Identification Model
- 1.2. Web Application
- 1.3. Modular Self-Checkout Lane (Hardware)

The sub-deliverables are:

- 1.1.1 Selection of Algorithm
- 1.1.2 Data Pre-processing
- 1.1.3 Training the Model
- 1.1.4 Testing the Model
- 1.1.5 Tuning the Model

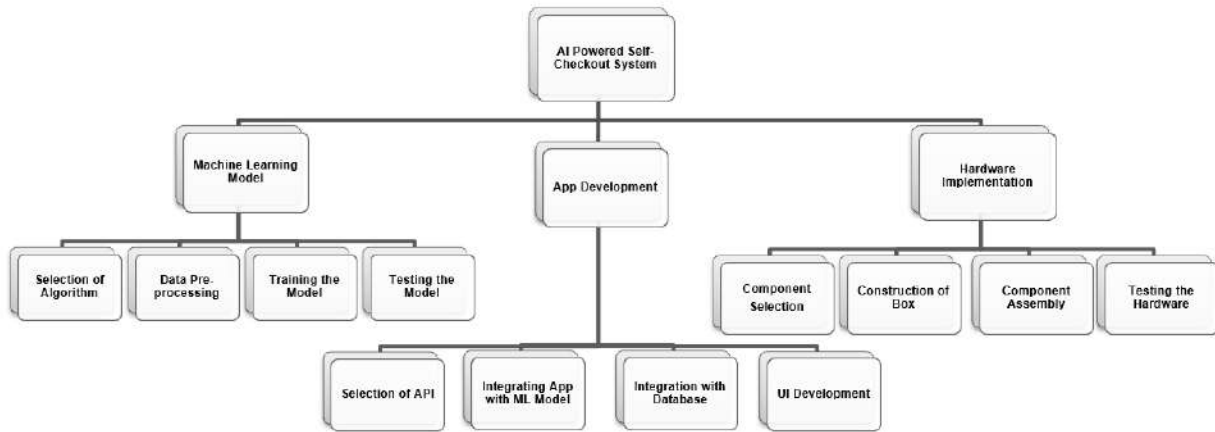
- 1.2.1 Application Programming Interface (API)
- 1.2.2 User Interface (UI)

- 1.3.1 Component Selection
- 1.3.2 Box Construction
- 1.3.3 Component Assembly
- 1.3.4 Set-up
- 1.3.5 Functionality Check

7.2 Project Plan

7.2.1 Work Breakdown Structure:

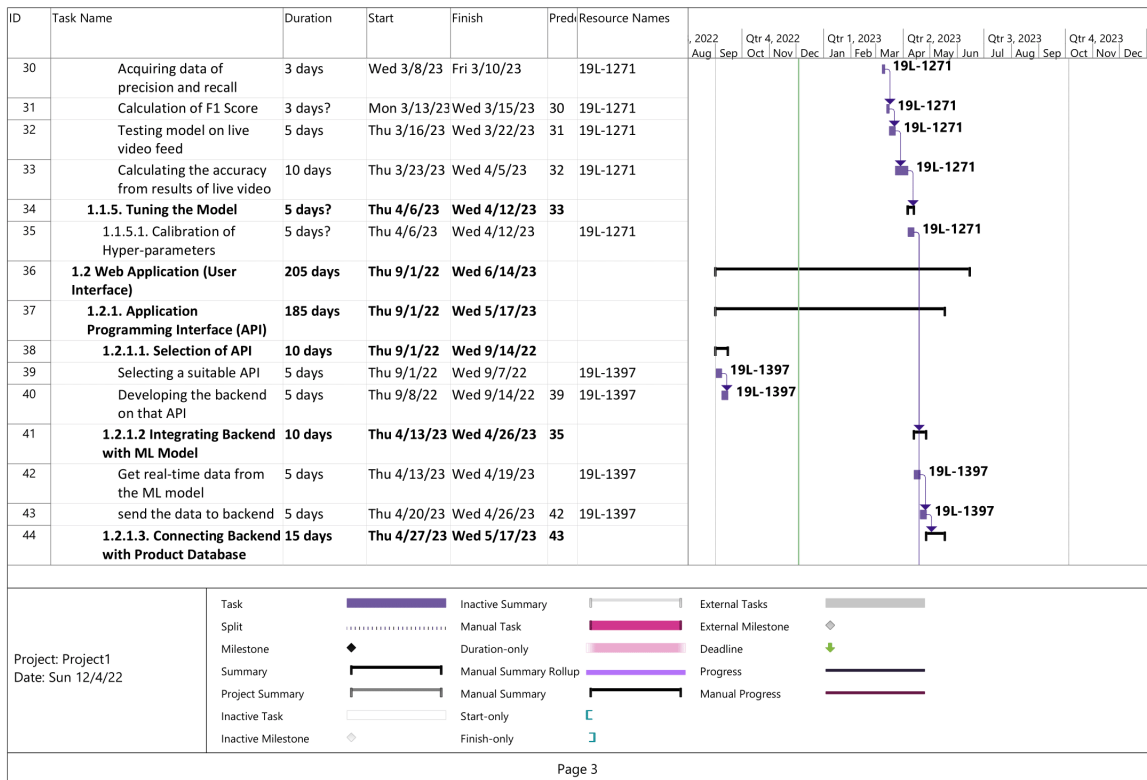
Table 3 High Level Work Breakdown Structure

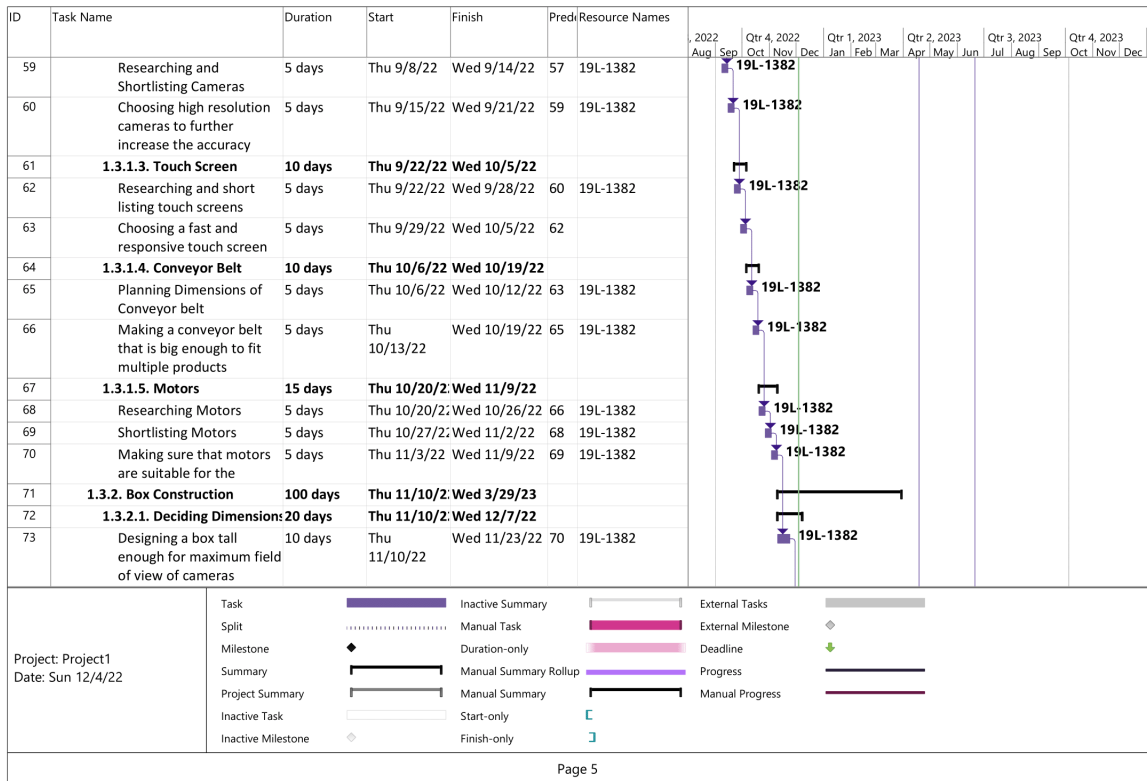
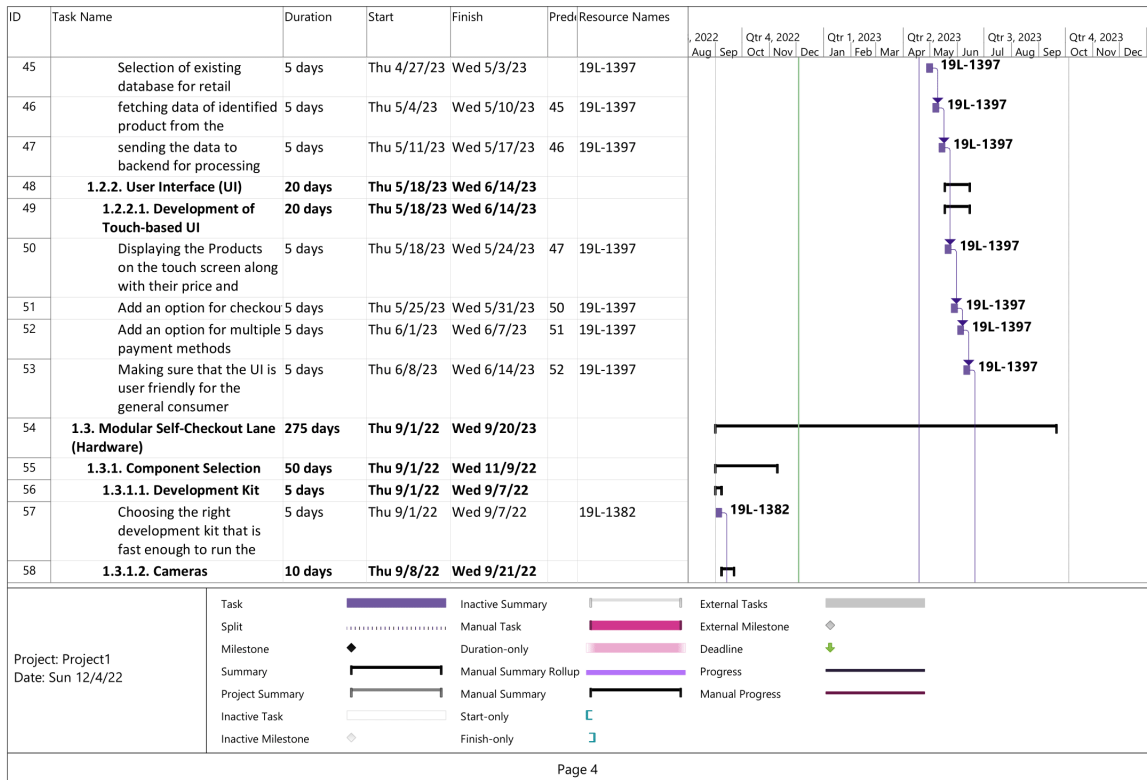


7.2.2 Gantt Chart:

Table 4 Complete Gantt Chart of the Project







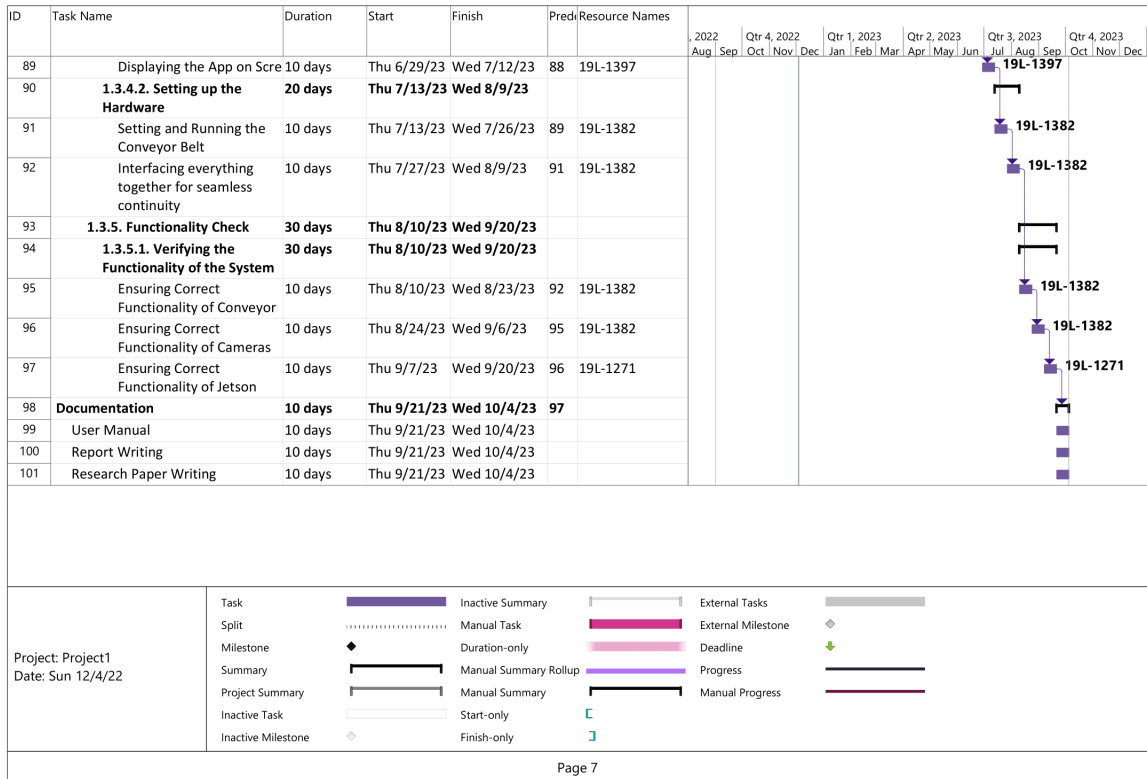
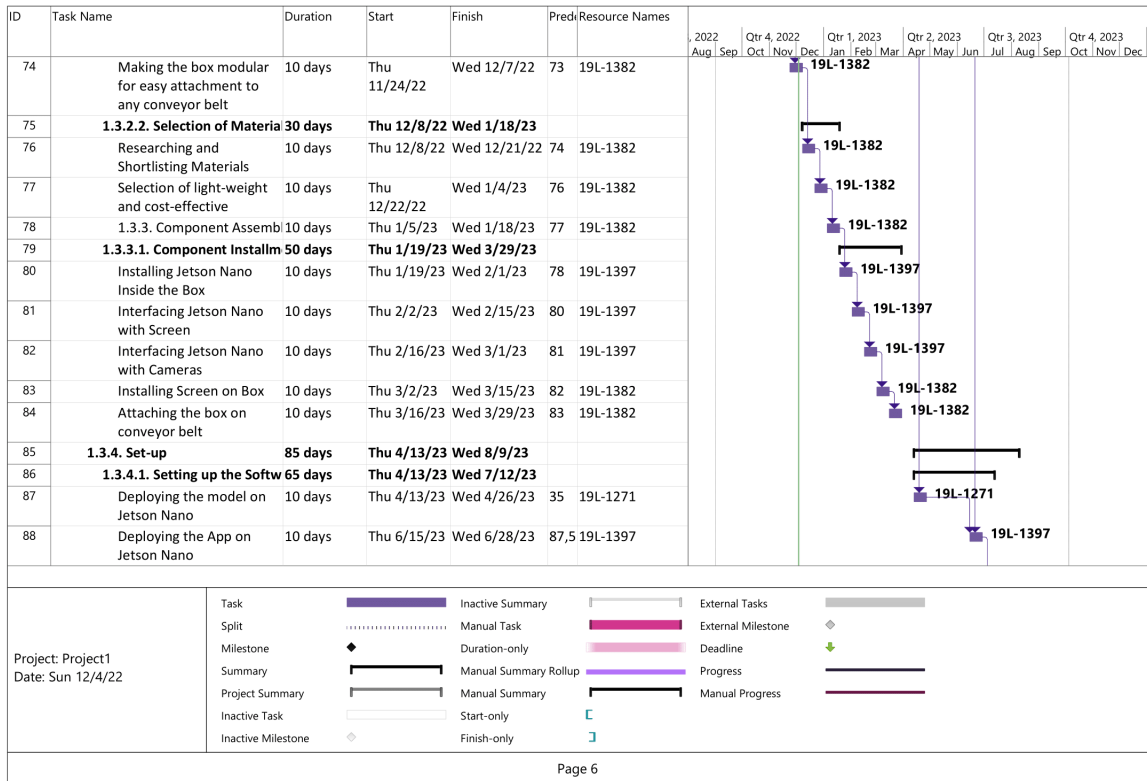


Table 5 Projected vs. Actual Time

Start Time	1 st August 2022
Projected Time to Completion	9 th August 2023
Actual Time of Completion	19 th April 2023
Difference	112 days

7.3 Project Cost (Projected)

Table 6 Projected Cost

Component List	Price
1080p30 CSI Cam or Webcam Equivalent	~ Rs. 8,000
Nvidia Jetson Nano 4GB	Rs. 40,000
7 Inch HDMI Capacitive LCD Touch Screen	Rs. 11,000
NEMA 17/24 Stepper Motors	Rs. 300 to 1500 (New & Used)
A4988 Stepper Motor Driver	Rs. 250
TB6560 3A Stepper Motor Driver	Rs. 950
ESP8266 CH340 NodeMCU V3 IOT Development Board	Rs. 550
Material & Construction Cost	Rs. 10,000
Estimated Total*	~ Rs. 74,750
Estimated Total with Overhead (30%)*	~ Rs. 97,175

7.4 Project Cost (Actual)

Table 7 Actual Cost

Component List	Price
Logitech C310 720p30 Webcam	Rs. 5000
Nvidia Jetson Nano 4GB	Rs. 46,000
7 Inch HDMI Capacitive LCD Touch Screen	Rs. 14,000
100w 775 Long Shaft DC Motor	Rs. 700
10A 12-40V PWM Controller	Rs. 600
Material & Construction Cost	Rs. 7,000
Total	Rs. 73,300

For our self-checkout system project, we had initially projected a budget of 74,750. This budget was carefully calculated considering all the components required, including the Nvidia Jetson Nano, Logitech C310 Webcam, 7-inch touchscreen, various electrical and mechanical parts for the conveyor belt system, and overhead costs. Overhead costs are those costs that aren't directly tied to a specific activity and include things such as administrative and operational expenses. With these overhead costs included, our total projected budget rose to Rs. 97,175.

However, we were able to strategically source materials and manage our resources effectively throughout the project. This resulted in our actual total expenditure coming to Rs. 73,300, which is below our initial projected budget excluding overheads. This demonstrates effective budget management and resource allocation throughout the project's duration.

In undertaking this project, we were fortunate enough to receive financial backing from the PEC (Punjab Engineering College). The funding was allocated to us as part of PEC's commitment to supporting innovative and impactful student-led projects that align with their vision of advancing technology and engineering solutions for societal benefits.

As a leading engineering institution, PEC is deeply invested in nurturing students' inventive thinking and practical application of their learning. By financing projects like ours, they create an environment that encourages technical skills development and fosters a culture of innovation and entrepreneurship.

We are grateful for PEC's financial support, and we believe that our successful completion of the project under the projected budget demonstrates responsible and effective use of these funds. It's our hope that this project will contribute positively to the broader mission of PEC to drive technological innovation and solution-driven engineering.

Chapter 8: Conclusion

The primary objective of this project was to create a self-checkout system powered by artificial intelligence to streamline the checkout process in retail environments. This goal was realized through the successful development and testing of a compact, automated system leveraging advanced technologies such as the NVIDIA Jetson Nano and image recognition algorithms.

The system developed achieved the desired functionality, reliably recognizing products placed on a conveyor belt, accurately calculating total prices, and facilitating a secure, seamless payment process. The integration of a user-friendly touchscreen interface also ensured that the system was accessible to users of varying levels of technical proficiency.

Throughout the course of this project, numerous tests were performed to ensure the system's accuracy and reliability. These tests included product recognition accuracy tests, stress tests on the conveyor system, and security tests for the payment gateway. All results met or exceeded initial expectations, providing confidence in the system's readiness for practical use.

Despite the successful completion of the project, there is always room for further refinement and expansion.

- **Refinement of Image Recognition Algorithm:** Further optimization of the image recognition algorithm could improve the accuracy of the product recognition, reducing the possibility of errors. This could involve implementing more advanced machine learning models or increasing the breadth of the training data set to encompass a wider variety of products.
- **Expanding Product Database:** As the system is implemented in different retail settings, it would be beneficial to expand the product database to cater to a more diverse range of items. This could involve a system for easy addition and removal of product details in the database by the store manager.

- **Additional Payment Methods:** While the system currently supports card payments via Braintree, there is potential to integrate other payment methods, such as digital wallets or QR code payments. This could make the system more versatile and convenient for users with different payment preferences.
- **Multilingual Support:** To cater to a more diverse user base, the system could be updated to offer multilingual support. This would allow non-English speaking users to use the system with ease and increase the system's accessibility.
- **Personalized Recommendations:** Leveraging data analytics, the system could offer personalized recommendations to customers based on their past purchases or scanned items. This could enhance the user experience and also drive additional sales for the retailer.
- **Integration with Loyalty Programs:** To provide a more comprehensive retail experience, the system could be integrated with store loyalty programs. This would allow users to earn and redeem points when they shop, encouraging repeat business.
- **Shopping Cart Tracking:** A future enhancement could involve integrating a system for tracking and managing shopping carts. This would enable the system to keep track of the items added to the cart, potentially reducing theft and improving inventory management. Furthermore, it could also provide a running total to the customer as they shop, enhancing the shopping experience.
- **Smart Shelf Implementation:** The development and integration of smart shelves could significantly enhance the functionality of the system. Smart shelves, equipped with weight sensors and RFID technology, can monitor stock levels in real-time and alert staff when restocking is needed. Additionally, these shelves could be linked to the self-checkout system, providing real-time updates of what items are being picked up and

placed in the cart, allowing for a seamless and potentially "checkout-free" shopping experience.

- **Robustness Against Varied Lighting Conditions:** Currently, the system performs optimally under certain lighting conditions. Further development could focus on enhancing the image recognition algorithm's performance under a wider range of lighting conditions, making the system more versatile for different store environments.
- **Customer Profile Creation:** For regular customers, the system could provide an option to create a customer profile. This profile could track purchase history, provide personalized discounts, and even predict shopping needs based on past purchases. This would not only increase customer satisfaction but also provide valuable data for inventory management and sales forecasting.

In conclusion, this project has achieved its set objectives and has substantial potential for future development and refinement. It is an encouraging step towards a more efficient, user-friendly, and technologically advanced retail landscape

References

- [1] J. Loechner, "Media Post," 27 May 2013. [Online]. Available: <https://www.mediapost.com/publications/article/201157/shoppers-prefer-personalized-brick-mortar-vs-on.html?print>. [Accessed 22 9 2022].

- [2] PYMNTS, "Amazon Expands 'Just Walk Out' as Shoppers' Checkout Expectations Rise," [Online]. Available: <https://www.pymnts.com/amazon-commerce/2022/amazon-expands-just-walk-out-as-shoppers-checkout-expectations-rise/>.

- [3] Mathworks, "What is Deep Learning?," [Online]. Available: <https://www.mathworks.com/discovery/deep-learning.html#:~:text=Deep%20learning%20is%20a%20machine,a%20pedestrian%20from%20a%20lampost..>

- [4] MathWorks, "Getting Started with Object Detection Using Deep Learning," [Online]. Available: <https://www.mathworks.com/help/vision/ug/getting-started-with-object-detection-using-deep-learning.html>.

- [5] 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*.

- [6] G. Bais, "Building Deep Learning-Based OCR Model: Lessons Learned," 14 November 2022. [Online]. Available: <https://neptune.ai/blog/building-deep-learning-based-ocr-model>. [Accessed 2 December 2022].

- [7] T. Hartman, "A Hybrid Checkout System," 24 June 2008. [Online]. Available: http://essay.utwente.nl/59064/1/Bsc._Thesis_IO%2C_Hartman_Tycho%2C_A_Hybrid_Checkout_System.pdf. [Accessed 2 December 2022].

- [8] A. Rigner, "AI-based machine vision for retail self-checkout system," 2019. [Online]. Available: <https://lup.lub.lu.se/student-papers/search/publication/8985308>. [Accessed 2 December 2022].

- [9] S. T. Bukhari, "ARC: A Vision-based Automatic Retail Checkout System," 17 May 2021. [Online]. Available: <https://arxiv.org/abs/2104.02832>. [Accessed 2 December 2022].
- [10] S. Ahmed, "Retail in Pakistan - An Overview," 27 February 2016. [Online]. Available: <https://www.linkedin.com/pulse/retail-pakistan-overview-sohail-ahmed#:~:text=There%20are%20approximately%20%20million,general%20stores%20and%20the%20like..> [Accessed 2 December 2022].
- [11] H. Bandyopadhyay, "YOLO: Real-Time Object Detection Explained," 7 October 2022. [Online]. Available: <https://www.v7labs.com/blog/yolo-object-detection>. [Accessed 2 December 2022].
- [12] N. AP, "autoAnnoter," [Online]. Available: <https://github.com/naseemap47/autoAnnoter>.
- [13] MakeSense. [Online]. Available: <https://www.makesense.ai/>.
- [14] darkpgmr, "DarkLabel," 4 September 2021. [Online]. Available: <https://github.com/darkpgmr/DarkLabel>. [Accessed 2 December 2022].
- [15] Z. a. S. H. a. L. Z. a. Y. X. a. L. T. H. a. C. J. a. W. H. a. Z. Y. a. G. T. a. Z. W. a. C. K. a. Z. W. a. L. D. Kuang, "MMOCR: A Comprehensive Toolbox for Text Detection, Recognition and Understanding," 14 August 2021. [Online]. Available: <https://arxiv.org/abs/2108.06543>. [Accessed 2 December 2022].
- [16] "Infrastructure, Industrialization," United Nations Sustainable Development, 2023. [Online]. Available: <https://www.un.org/sustainabledevelopment/infrastructure-industrialization/>
- [17] "Welcome | Flask (A Python Microframework)," Flask, 2023. [Online]. Available: <https://flask.palletsprojects.com/en/2.3.x/>
- [18] Widdershin, "Flask-Desktop," GitHub, 2023. [Online]. Available: <https://github.com/Widdershin/flask-desktop>

[19] "Tesseract-ocr/tesseract," GitHub, 2023. [Online]. Available: <https://github.com/tesseract-ocr/tesseract>

[20] JaidedAI, "EasyOCR," GitHub, 2023. [Online]. Available: <https://github.com/JaidedAI/EasyOCR>

[21] Braintree, "Braintree/braintree_python," GitHub, 2023. [Online]. Available: https://github.com/braintree/braintree_python

Appendices

Nvidia Jetson Nano Pin Diagram:

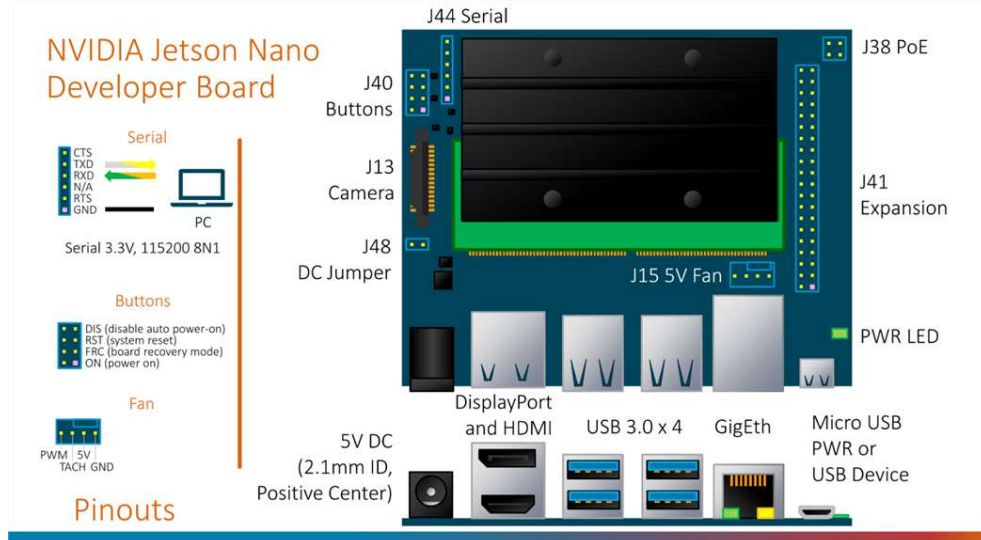


Figure 60 Jetson Nano Pinout

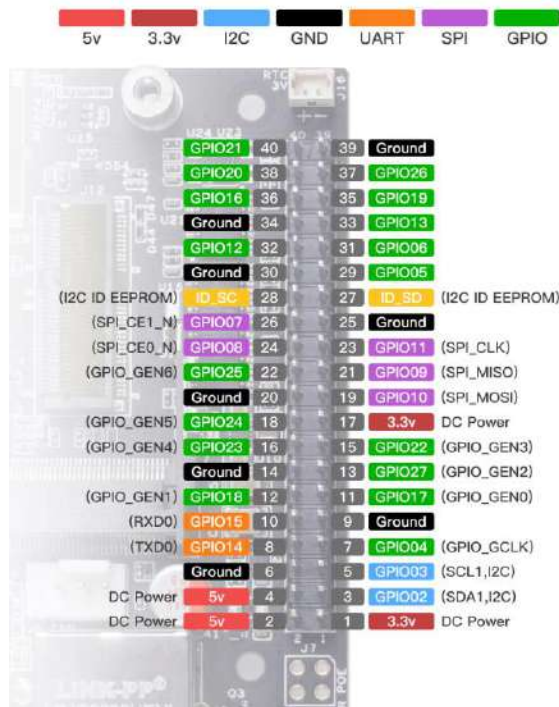


Figure 61 Jetson Nano Pin Diagram

HTML & CSS CODE:

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <title>Video Streaming Demonstration</title>
    {{ turbo() }}

  </head>
  <body>

    <link rel= "stylesheet" type= "text/css" href= "{{
url_for('static',filename='styles/style.css') }}">
    <div class="container">
      <h1>SELF CHECKOUT COUNTER</h1>
      <hr>
      <table id="data" class="table table-striped">
        <thead>
          <tr>
            <th>Product</th>
            <th>Name</th>
            <th>Quantity</th>
            <th>Price</th>
          </tr>
        </thead>
        <tbody>
          <tfoot>
            <tr>
              <th colspan="3" style="text-align:right">Total Price:</th>
              <th id="total-price"></th>
            </tr>
          </tfoot>
        </tbody>
      </table>

      <button id="checkout-btn" class="btn btn-success btn-lg">CHECKOUT</button>

    </div>
    <h1>Video Streaming Demonstration</h1>
    

    <div id="container">
      <div>
```

```

    <video autoplay="true" id="videoElement"></video>
    <canvas id="canvasElement"></canvas>
    <img id="photo">
  </div>
</div>

<script type="text/javascript" charset="utf8"
src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script type="text/javascript" charset="utf8"
src="https://cdn.datatables.net/1.10.25/js/jquery.dataTables.js"></script>
<script type="text/javascript" charset="utf8"
src="https://cdn.datatables.net/1.10.25/js/dataTables.bootstrap5.js"></script>

<script>

  var mytable;

  function initDataTable() {
    if (!$.fn.DataTable.isDataTable('#data')) {
      mytable = $('#data').DataTable({
        ajax: {
          url: '/api/data',
        },
        columns: [
          { data: 'Product' },
          { data: 'Name' },
          { data: 'Quantity' },
          { data: 'Price' },
        ],
      });
    } else {
      mytable = $('#data').DataTable();
    }
  }

  initDataTable();

  setInterval(function () {
    mytable.ajax.reload(null, false);

    // Calculate the total price
    let totalPrice = 0;
    mytable.rows().every(function () {
      const rowData = this.data();

```

```

        const priceStr = rowData.Price.replace('Rs. ', '');
        totalPrice += parseFloat(priceStr);
    });

    // Update the total price in the table footer
    document.getElementById('total-price').innerHTML = `Rs.
    ${totalPrice.toFixed(2)}`;
    }, 1000);

    function printReceipt() {
    const receiptWindow = window.open('', '_blank');
    receiptWindow.document.write('<html><head><title>Receipt</title></head><body>');
    receiptWindow.document.write('<h1>Receipt</h1>');
    receiptWindow.document.write(document.getElementById('data').outerHTML);
    receiptWindow.document.write('</body></html>');
    receiptWindow.document.close();
    receiptWindow.print();
    }

    document.getElementById('checkout-btn').addEventListener('click',
function () {
    window.open('/checkout', '_blank');
    });

</script>

</body>
</html>

/* Global Styles */
html, body {
    height: 100%;
    margin: 0;
    font-family: 'Poppins', sans-serif;
    background-color: #f5f5f5;
    font-size: 25px;
}

.container {

```

```

    display: flex;
    flex-direction: column;
    margin: 0;
    padding: 5rem;
    align-items: stretch; /* Add align-items: stretch */
  }
  /* Table Styles */
  #data {
    background-color: #ffffff;
    box-shadow: 0 0.4rem 1.2rem rgba(0, 0, 0, 0.1);
    border-radius: 1rem;
  }

  h1 {
    color: #4b0082;
    font-weight: 600;
    text-align: center;
  }

  #data th {
    background-color: #4b0082;
    color: #fff;
    padding: 1rem;
    text-transform: uppercase;
    border-radius: 0.5rem 0 0;
  }

  #data td {
    padding: 1rem;
    border-bottom: 1px solid #ddd;
  }

  #data tfoot th {
    background-color: #eee;
    color: #4b0082;
    font-weight: 600;
    border-radius: 0 0 0.5rem;
  }
  /* Checkout Button */
  #checkout-btn {
    background-color: #4b0082;
    color: #fff;
    font-size: 25px;
    padding: 0.5rem 1rem;
  }

```

```

    margin-top: 1rem;
    border-radius: 0.4rem;
    cursor: pointer;
    transition: background-color 0.3s;
}

#checkout-btn:hover {
    background-color: #6a1b9a;
}

/* Checkout Button */
#capture-btn {
    background-color: #4b0082;
    color: #fff;
    font-weight: 600;
    padding: 3rem 2rem;
    margin-top: 1rem;
    border-radius: 0.4rem;
    cursor: pointer;
    transition: background-color 0.3s;
}

#capture-btn:hover {
    background-color: #6a1b9a;
}

/* Video Streaming */
#bg {
    padding: 2rem; /* Added 'auto' for center alignment */
    border-radius: 3rem;
    overflow: hidden;
}

#videoElement {
    border-radius: 1rem;
}

#canvasElement {
    display: none;
}

#photo {
    position: absolute;
    top: 0;

```

```
        left: 0;
        border-radius: 1rem;
    }

/* Custom styles for DataTables */
.dataTables_wrapper {
    width: 100%;
    padding: 0;
}

.dataTables_wrapper div {
    box-sizing: border-box;
}

table.dataTable {
    width: 100% !important;
}
```

Glossary

List all acronyms and technical terms in alphabetical order along with their brief description, as shown below:

SBC	Single Board Computer
AI	Artificial Intelligence
RFID	Radio-frequency Identification
YOLO	You Only Look Once
DC	Direct Current
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
LDR	Light-dependent resistor
CV	Computer Vision
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
API	Application Programming Interface
COCO	Common Objects in Context
Flask	A micro web framework written in Python. It does not require particular tools or libraries, and it's used to make web applications.
TensorFlow	A micro web framework written in Python. It does not require particular tools or libraries, and it's used to make web applications.

Similarity (Plagiarism) Report

6/12/2023

Turnitin Originality Report

Turnitin Originality Report

AI Powered Self-Checkout System by Muhammad Talha

From FASTBachlor (FASTBachlor)



- Processed on 12-Jun-2023 10:41 PKT
- ID: 2114244322
- Word Count: 12540

Similarity Index

8%

Similarity by Source

Internet Sources:

7%

Publications:

2%

Student Papers:

4%

sources:

- 1 1% match (Internet from 16-Apr-2023)
<https://WWW.coursehero.com/file/126044950/Smart-Doorbell-System-11744281174405L174449docx/>
- 2 1% match (Internet from 11-Jun-2020)
<https://www.coursehero.com/file/42294577/report-fyp-2docx/>
- 3 < 1% match (Internet from 20-Jun-2020)
<https://www.coursehero.com/file/32709350/FYP-II-Reportdocx/>
- 4 < 1% match (Internet from 12-Mar-2023)
<https://WWW.coursehero.com/file/90767771/19BCE2191-Review1-AI-1pdf/>
- 5 < 1% match (Internet from 02-Sep-2022)
<https://programtalk.com/python-more-examples/tensorflow.lite.Interpreter/>
- 6 < 1% match (Internet from 29-May-2023)
https://github.com/RizwanMunawar/yolov7-object-tracking/blob/main/detect_and_track.py
- 7 < 1% match (Internet from 09-Jan-2023)
<https://github.com/rahafalhejaily/soccerproject/blob/main/main.py>
- 8 < 1% match (Internet from 18-Mar-2022)
<https://00w5.com/article/1ec7763d43fcf5d303c428a06c0fbfab.html>

file:///C:/Users/asad.khan/Downloads/Turnitin_Originality_Report_2114244322.html

1/34

- 9 < 1% match (Internet from 04-Feb-2023)
<https://blog.miguelgrinberg.com/post/video-streaming-with-flask/page/17>
-
- 10 < 1% match (Internet from 14-Jan-2023)
<https://python.tutorialink.com/no-data-available-in-table-flask-python-sqlite/>
-
- 11 < 1% match (student papers from 16-Aug-2020)
[Submitted to University College Technology Sarawak on 2020-08-16](#)
-
- 12 < 1% match (student papers from 08-Jun-2023)
[Submitted to Sreenidhi International School on 2023-06-08](#)
-
- 13 < 1% match (student papers from 30-May-2023)
[Submitted to BPP College of Professional Studies Limited on 2023-05-30](#)
-
- 14 < 1% match (student papers from 20-May-2023)
[Submitted to CSU, San Jose State University on 2023-05-20](#)
-
- 15 < 1% match (Internet from 01-Nov-2022)
<https://forum.zkoss.org/question/113726/how-to-select-a-css-classid-with-js-in-zk/>
-
- 16 < 1% match (Internet from 17-May-2023)
<https://stackoverflow.com/questions/64710313/how-to-capture-the-first-frame-of-a-video-in-react-konva>
-
- 17 < 1% match (Internet from 23-Nov-2020)
<https://www.namely.com/trust/>
-
- 18 < 1% match (Internet from 16-Aug-2022)
<http://www.iotsharing.com/2021/08/demo-52-computer-vision-with-esp32.html>
-
- 19 < 1% match (Internet from 11-Nov-2020)
<https://www.rent-a-tent.nl/kroatie/istrie/camping/lanterna>
-
- 20 < 1% match (Agus Kurniawan. "IoT Projects with NVIDIA Jetson Nano", Springer Science and Business Media LLC, 2021)
[Agus Kurniawan. "IoT Projects with NVIDIA Jetson Nano", Springer Science and Business Media LLC, 2021](#)
-
- 21 < 1% match (Internet from 26-May-2023)
<https://mail.datatables.net/forums/discussion/68512/show-search-pagination-disappears-when-column-is-deleted>
-