Palestine Polytechnic University

College of Information Technology and Computer Engineering

Department of Computer System Engineering

Smart Products Sorting Line Using Computer Vision

Project Team

Yaseen Hazem Joba

Mohammad Khaled Halahlah

Muhammad Judeh Herbawi

Supervisor

Dr. Mousa Al-Refaiyeh

May - 2023

Abstract

Sorting plays a critical role in manufacturing processes as it ensures that products meet specific quality standards and specifications before being shipped to customers. However, traditional sorting methods often rely on manual labor, which can be time-consuming and prone to errors. To address this issue, we propose a real-time, fully automated sorting system that employs artificial intelligence and modern hardware techniques to handle the sorting process quickly and accurately.

In this project, we implemented a fruit sorting system that sorts and categorizes fruits into four categories and detects rotten fruits. Our ultimate goal is to deploy our sorting system as a standalone sorting line in real factories, helping these facilities save time and money by automating the sorting process in real time without human intervention. We believe that our system has the potential to revolutionize the way sorting is done in manufacturing and contribute to the development of a more efficient and sustainable supply chain.

The results of this project indicate that our proposed sorting system has been successfully developed and tested, providing fast and accurate sorting without the need for human intervention. The potential impact of this system on the manufacturing industry is significant, as it provides a more efficient and sustainable supply chain. We are confident that the implementation of this system in real factories has the potential to transform sorting methods in the manufacturing industry.

تلعب عملية التصنيف دوراً حرجاً في عمليات التصنيع، إذ يجب ان تضمن تلبية المنتجات معايير الجودة المحددة والمواصفات قبل شحنها للعملاء. ومع ذلك، تعتمد طرق التصنيف التقليدية عادة على العمل اليدوي، الذي يمكن أن يستغرق وقتاً طويلاً ويكون عرضة للأخطاء. ولحل هذه المشكلة، نقترح نظام تصنيف آلي في الوقت الحقيقي وتلقائي بالكامل يستخدم الذكاء الاصطناعي وتقنيات الأجهزة الحديثة للتعامل مع عملية التصنيف بطريقة سريعة ودقيقة.

في هذا المشروع، قمنا بتصميم و تنفيذ نظام تصنيف الفواكة المقترح الذي يمكنه الكشف عن الفواكه وفرزها إلى أربع فئات مختلفة من تفاح و موز و جزر و برتقال بالإضافة الى فرز الفواكه الفاسدة. خلال مرحلة التنفيذ، قمنا بتصميم الأجهزة المطلوبة ودرسنا احتياجات البرمجيات. هدفنا النهائي هو نشر نظامنا للتصنيف كخط تصنيف مستقل في المصانع الحقيقية، مما يساعد هذه المرافق على توفير الوقت والمال من خلال تحويل عملية التصنيف إلى عملية تلقائية و آلية وفي الوقت اللحظي دون تدخل الإنسان. نحن نعتقد أن نظامنا لديه القدرة على عمل ثورة في طريقة التصنيف في عمليات التصنيع والمساهمة في تطوير سلسلة إمدادٍ أكثر كفاءة واستدامة.

تشير نتائج هذا المشروع إلى أن نظام الفرز الذي اقترحناه تم تطويره واختباره بنجاح، ويوفر فرزًا سريعًا ودقيقًا دون الحاجة إلى تدخل الإنسان. إن الأثر المحتمل لهذا النظام على مجال التصنيع كبير، حيث يوفر سلسلة إمداد أكثر كفاءة واستدامة. نحن على ثقة من أن تنفيذ هذا النظام في المصانع له القدرة على تحويل طرق الفرز في الصناعة.

Acknowledgments

In the name of Allah, Most Gracious, Most Merciful, for giving us the strength and knowledge to complete this project. We would like to express our heartfelt thanks to our parents for their unconditional love and support, which has been invaluable to us throughout our lives. We know that words cannot fully convey our gratitude, but we hope that our actions and achievements will reflect the depth of our appreciation.

We also want to express our gratitude to our families and friends, who have provided us with endless encouragement and support, especially during the completion of this project. We could not have done it without their guidance and motivation.

Finally, we would like to thank our supervisor, Dr. Mousa Al-Refaiyeh, for his valuable help and advice throughout the project. His expertise and guidance were essential to our success. We would also like to thank Eng. Momen Herbawi for his expertise and assistance in CAD design, which has been essential to the success of this project. We are truly grateful for their support and encouragement, and we hope to continue learning from their wisdom and expertise in the future.

Table of Contents

List of Figures and Tables

Figures

Tables

Chapter 1

Introduction

1.1 Preface

This project aligns with the fourth industrial revolution, which is the integration of advanced technologies like artificial intelligence and automation into traditional manufacturing processes. Our aim is to develop a product sorting system that uses computer vision and deep learning algorithms to monitor and sort products in real time within factory settings. By detecting and segregating defective items, the system will enhance sorting efficiency.

The ultimate objective was to establish this intelligent sorting solution as a reliable and efficient tool for factories to increase productivity and quality control.

1.2 Aims and Objectives

In this project, we propose a system that can provide the following:

1. Develop a system that recognizes and sorts products or fruits (apples, oranges, bananas, carrots). Computer vision and deep learning will recognize the products and trigger the microcontroller to sort them.
2. Create a real-time system that simulates human sorting activities to help factories increase their production speed.
3. Drop the error percentage that humans can make by increasing and optimizing the system algorithms' accuracies.
4. Create a real-time detection for rotten fruits and drop them out during the sorting process by developing a special deep-learning model for this detection.

1.3 Problem Statement

Most of the sorting process in factories is currently performed by humans, resulting in a higher error rate and longer processing time compared to machines. Many factories strive for complete automation and intelligence in their production lines. However, one of the challenges is integrating a sorting line at the end of the production process. Consequently, the demand for efficient product sorting lines has increased, and there has been rapid progress in optimization techniques in this domain, intending to achieve automatic, innovative, and remarkably accurate sorting.

1.4 Project Requirements

Functional Requirements:

1. The system should be able to sort four different types of fruits with no human inference (Apple, Banana, Orange, Carrot)
2. The system should be able to recognize the rotten fruits.
3. The system should be able to take frames of the fruits using two cameras with a real-time response while the conveyor is moving.
4. The system should be able to recognize the fruits through an object detection model.
5. The system should be able to drive a rotating cylinder, so the right slot is exactly under the conveyor.

Non-functional Requirements:

1. The system must maintain an unbroken and continuous workflow. This necessitates the system's ability to execute tasks in real-time, without any occurrences of delays or interruptions, thereby optimizing its performance.
2. The system must be equipped with the necessary capabilities to sort a single fruit promptly, with a maximum allowable time of 8 seconds. This ensures that the system operates efficiently, with minimal delay or disruption to the overall process.
3. The system's accuracy and reliability are crucial aspects to consider. It should consistently sort 90 out of every 100 fruits without any mistakes or issues. This ensures that the system operates effectively and efficiently, reducing the chances of problems that may occur due to incorrect or inconsistent sorting.

1.5 System Description

Our system's primary objective was to utilize computer vision techniques to classify various types of fruits accurately. This involved continuously capturing frames from a conveyor belt using two cameras. These cameras provided feedback to an algorithm responsible for fruit identification and freshness assessment. The algorithm evaluated each fruit to determine if it was fresh or rotten. Based on the algorithm's results, the fruits were efficiently sorted into separate

slots using a rotating cylindrical container. This container was specifically designed to rotate based on the fruit's type and condition, enabling a highly efficient and precise classification process.

1.6 Project Limitations/constraints

- **Time constraint**

  The fruit sorting process on the conveyor belt, including fruit class detection, typically takes a maximum of 8 seconds. However, the actual time required for sorting may vary for individual fruits based on their respective slot positions due to the interdependency between the software and hardware components.

- **Scope constraint**

  The scope of this project is expansive and has potential applications in numerous fields. However, the focus of our project will be on fruits. Due to constraints on our time and budget, we will limit our work to specific fruit types and classes. The types of fruits that we will be focusing on include oranges, apples, carrots, and bananas. Furthermore, we will be developing a general class for rotten fruits that applies to all fruit types. The small size and mass of the fruit can be attributed to the limited dimensions of the conveyor belt. on the other hand, the cylinder can hold weight to 8 kilograms.

- **Cost constraint**

  The Deep Learning algorithms implemented in this project are known for their resource-intensive and complex nature. Consequently, the project necessitates specialized hardware components to guarantee the effective functioning of these algorithms. However, the acquisition of such components entails a considerable financial investment.

  Furthermore, the conveyor belt is designed with a black coloration to enable straightforward detection of objects placed atop it.

1.7 Report outline

This report is organized as follows: Chapter 2 introduces a quick overview of the main technologies and algorithms that will be used in the project with a literature review that compares our project to other related projects. Chapter 3 describes the design of the project in both hardware and software terms by showing the design options, software background conceptual design, and schematic diagram. Chapter 4 demonstrates the hardware and software implementation of the system. Chapter 5 shows the validation and testing process of the project and the final results that came out. Chapter 6 gives a summary of the project with some suggestions for future works.

Chapter 2

Theoretical Background

2.1 Preface

This chapter mainly displays a quick background of some of the topics on which building the project depends and the basic parts of it. First, we will talk about the theoretical background, and the most important thing we will address is deep learning and computer vision with its latest algorithms that work to detect objects.

2.2 Theoretical background

This section provides some information about some technologies and algorithms that will be used in our project.

2.2.1 Machine Learning and Deep Learning

Machine learning is a branch of AI that mimics the workings of the human brain in processing data for use in many areas such as detecting objects, recognizing speech, translating languages, and making decisions. Deep learning[1] is a branch of machine learning that extracts picture features without the need for human interference. So Deep Learning achieves great power and flexibility learning.

Most deep learning techniques use neural network architectures. The word “deep” usually refers to the number of hidden layers in the neural network. Deep learning models are trained by operating large sets of labeled data and neural network architectures that learn features straight from the data without the need for manual feature extraction[2][3].

2.2.2 Computer Vision

Computer vision[4] is a field of artificial intelligence that trains computers to analyze and understand the visual world. Using digital images from cameras, videos, and deep learning models, machines can accurately identify and categorize objects — and then respond to what they see. Computer vision allows sorting machines to make sense of its fruits type. Cameras grab video around the object and feed it to computer vision software, which then processes the images in real-time to detect fruits.

2.2.3 HOG as alternative

One approach used for object detection is the Histogram of Oriented Gradients (HOG)[5] combined with machine learning algorithms such as Support Vector Machines (SVM), as an alternative to deep learning. HOG is a feature descriptor commonly employed for extracting features from image data in computer vision applications. This method involves breaking down the entire image into smaller regions, calculating gradients and orientations for each localized part, and generating histograms based on these computed values. These histograms are constructed using pixel weights. However, this method faces challenges in multiple object detection. To address this, a bounding box object detection algorithm, such as SVM, is typically applied to identify potential objects. The HOG features are then used as input for the learning algorithm to detect relevant objects. In contrast, deep learning techniques like YOLO take the entire image as input and directly output the object's location and name[3]. Figure 2.1 illustrates one of the distinctions between machine learning and deep learning approaches.

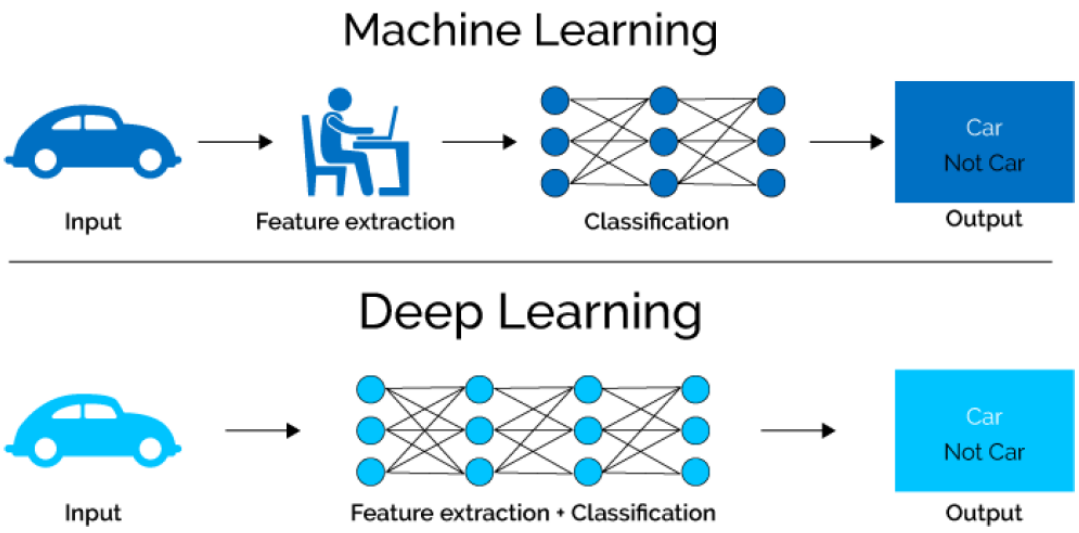

Figure 2.1: Machine Learning vs Deep Learning, towards data science[3]

2.3 Literature review

These studies helped us choose the technologies and models that will achieve speed and accuracy in detecting fruits and create a faster way of sorting. The following table shows the previous project that can be related.

Table 2.1: A literature review table

Title	Team	Year	The main idea
Object Sorting System Using Robotic Arm.	Enas Ideas Haneen Qabajah Kifah Abdeen **Supervisor: Dr. Liana Tamimi**	2018	Designed and implemented a robot to sort the object based on their colors using Arm and image processing.
Potato Sorting Machine Based on LabVIEW with Image Processing Strategy	Anas AL-Hraine Diaa Msulam **Supervisor: Prof. Sameer Khader**	2019	Designed and implemented a robot to sort the potatoes based on their size using image processing and mechanical movements.
Design and Implementation of an Automatic Sorting Machine	Islam Abu Khalaf Murad Abu Arafeh **Supervisor: Eng. Zoheir Wazwaz**	2018	Designed and implemented a machine to sort metals without technology related to AI, and using mechanical movement.
Object Categorization for Industrial Applications	Sarah Abu Zeneh, Sondos Zaro, Sarah Alsharif **Supervisor: Dr. Alaa Halawani**	2022	Designed and implemented a machine to sort pipes using computer vision, and using mechanical movement.

To achieve high performance and ensure accurate tracking of products and fruits, sorting systems must prioritize speed. However, a previous project titled "Object Sorting System Using Robotic Arm" (2018)[6] utilized an arm for object manipulation, which resulted in slower system movement. Conversely, the project relied on simple image processing techniques to determine the color of the objects.

The variety of objects that can be detected is crucial; Potato Sorting Machine Based on LabVIEW with an Image Processing strategy (2019)[7] project used image processing techniques to sort the potatoes based on their size with Electrical and mechanical ways for sorting.

The primary objective of the study Object Categorization for Industrial Applications (2022)[8] is to classify diverse industrial materials. To achieve this, the system employs a distinct mechanism comprising servo motors and metallic arms. This mechanism effectively sorts pipes and directs them to their respective containers. Notably, the system operates at a pace that does not impose real-time constraints.

This project will introduce a new way for sorting on both hardware and software levels, the software level will handle the problem by using deep learning by YOLO, and the hardware will contain a cyclic cylinder working with rotations to move the products on specific slots.

2.4 Summary

The core component of our project is computer vision, which involves processing visual inputs such as photos and videos. These visuals are fed into an algorithm that recognizes the type and condition of fruits and determines their placement.

To enhance the performance and analysis time, we opted for YOLO as an alternative algorithm over conventional methods like HOG. YOLO demonstrates superior execution and analysis capabilities. While our project shares similarities with others at a basic level, there are notable differences in terms of the chosen algorithm and the mechanism used for fruit separation.

Chapter 3

System Design

3.1 Preface

This chapter serves as an introduction to the primary components of the project, containing both software and hardware aspects along with their respective alternatives. The chapter primarily emphasizes the design, and system diagrams, and outlines the overall process.

3.2 Design options for hardware components

This section describes the main hardware components needed to construct our project and illustrates each component's function.

3.2.1 Arduino Mega 2560

The ATmega2560-based Arduino Mega 2560[9] microcontroller board offers a wide range of impressive capabilities. Boasting 54 digital input/output pins, including 15 that support PWM output, as well as 16 analog inputs, the Mega 2560 is well-equipped to bring your projects to fruition. Additionally, it includes 4 UARTs for serial communication, a 16 MHz crystal oscillator, a USB connection, and a reset button, making it easy to get started by either plugging it into a computer via USB or powering it with an AC-to-DC adapter or battery. Moreover, the Mega 2560 is highly versatile and compatible with a multitude of shields originally designed for the Uno, Duemilanove, and Diecimila boards. The figure below displays the Arduino Mega 2560 along with a schematic showcasing the specifications of its pins.

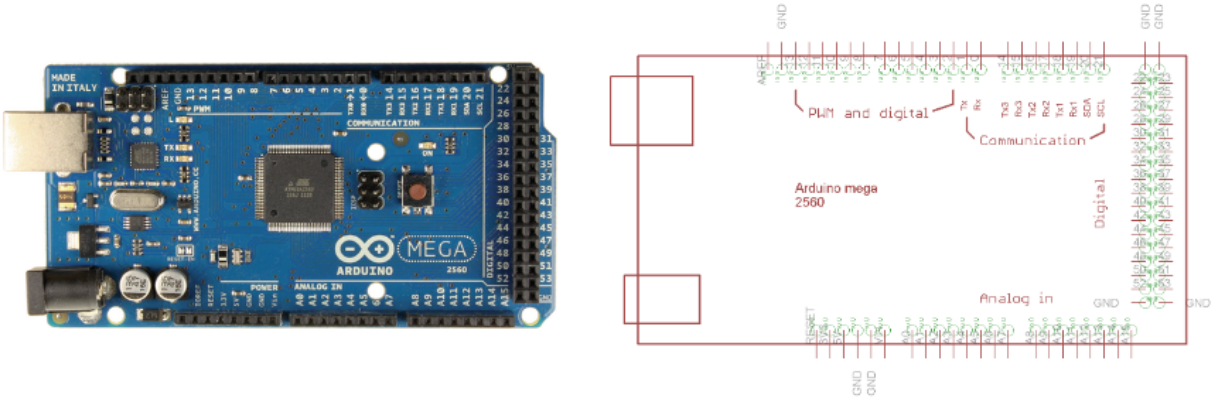

Figure 3.1: Arduino Mega 2560 and its schematic diagram

The following table provides a comprehensive overview of the specifications for the Arduino Mega 2560 microcontroller. It presents essential information about its various features, including its operating voltage, memory capacity, digital and analog input/output pins, etc.

Table 3.1: Arduino Mega 2560 key features

Microcontroller	Atmega2560
Operating Voltage	5V
Input Voltage	7V – 12V
USB Port	Yes
DC Power Jack	Yes
Current Rating Per I/O Pin	20mA
Current Drawn from Chip	50mA
Digital I/O Pins	54
PWM	15
Analog Pins (Can be used as Digital Pins)	16 (Out of Digital I/O Pins)
Flash Memory	256KB
SRAM	8KB
EEPROM	4KB
Crystal Oscillator	16 MHz
LED	Yes/Attached with Digital Pin 13
Wi-Fi	No
Shield Compatibility	Yes

3.2.2 CyberTrack H3-TAA

Figure 3.2 showcases the CyberTrack H3-TAA camera[10] with impressive features. It offers high-quality video with HD resolution, providing a wide viewing angle of 160 degrees without any blind spots. Equipped with a 1.3 MP CMOS color sensor, it can capture videos at a resolution of 1280*720P with a shooting speed of 30 frames per second. The camera supports video shooting formats such as YUY2 and MJPG. In terms of physical dimensions, it measures 3″ x 3.35″ x 1.26″ and weighs 100g, making it compact and lightweight.

Figure 3.2: CyberTrack H3-TAA

3.2.3 sj4000cam as alternative

Sj4000cam[11] shown in Figure 3.3, its Action Camera has full HD resolution recording at 1920×1080. Wide Dynamic Range (WDR) lets you Capture friendly and vivid Picturesque locations with the SJ4000. Most of its specifications are like an SD camera specification, as it has a 12-megapixel sensor for shooting crisp photos and recording 1080P videos, with a wide viewing angle of 170 degrees, and has maximum external storage of 64GB.

Figure 3.3: sj4000 cam

The CyberTrack H3-TAA camera was selected for our project due to its suitable capabilities. While the sj4000cam camera may have more features, its excessive capabilities surpass our requirements. The limited resolution of the CyberTrack H3-TAA camera aligns with our deep learning model needs. Furthermore, considering cost-effectiveness, the CyberTrack H3-TAA camera fits our budget while meeting the necessary specifications for project success.

3.2.4 Conveyor System

Typically[12], conveyor systems be made up of a belt stretched across two or more pulleys. The belt comprises a closed loop around the pulleys so it can always rotate. One pulley, known as the drive pulley, drives or tows the belt, carrying items or fruits from one location to another.

Figure 3.4: Conveyor System

3.2.5 Nema 23 23HS5628 Stepper Motor

The requirement for a stepper motor arises from the need for rotational movement in the Rotating Cylinder system. The stepper motor, a DC electric motor, enables a full rotation to be divided into precise and equal steps. It is worth noting that the maximum speed typically achieved by a stepper motor is 1000 rpms[13]. Figure 3.5 depicts the stepper motor, accompanied by its corresponding schematic diagram.

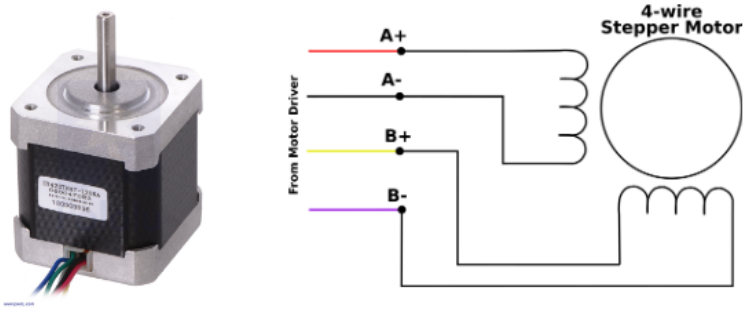

Figure 3.5: Stepper motor and its schematic diagram

Figure 3.6 highlights a table row that presents the specifications of the stepper motor, including crucial information such as the step length and holding torque.

Step angle accuracy: + - 5%(full step,not load)
Resistance accuracy: + - 10%
Inductance accuracy: + - 20%
Temperature rise: 80deg Max(rated current,2 phase on)
Ambient temperature ----------—20deg ~+50deg
Insulation resistance:100MΩ Min,500VDC
Insultion Strength--------500VAC for one minute
6.35mm OR 8mm diameter shaft

Model	Step Angle (deg)	Motor Length (mm)	Rated voltage (V)	Rated Current (A)	Phase Resistance (Ohm)	Phase Inductance (mH)	Holding Torque (Kg.cm)	Moment of inertia (g.cm2)	Lead Wire (No)	Motor Weight (Kg)
23HS4128	1.8	41	2	2.8	0.7	1.4	5.5	120	4	0.45
23HS5128	1.8	51	2.3	2.8	0.83	2.2	10.1	275	4	0.65
23HS5628	1.8	56	2.5	2.8	0.9	2.5	12.6	300	4	0.7
23HS7628	1.8	76	3.2	2.8	1.13	3.6	18.9	480	4	1
23HS8240	1.8	82	2.8	4	0.7	1.5	18	520	4	1.2
23HS11528	1.8	115	5.88	2.8	2.1	10	30	650	4	1.8

Red: A+
Green: A-
Yellow: B+
Blue: B-

Figure 3.6: Stepper motor specifications

3.2.6 TB6600 Stepper Driver

We used the TB6600 stepper driver for the stepper motor because:

- It provides precise control over the motor's rotation by converting digital signals into the appropriate current levels for each phase of the motor.
- It helps protect the motor and other components from damage by providing overcurrent and thermal shutdown protection.
- Figure 3.7 showcases the TB6600 stepper driver alongside its schematic diagram.

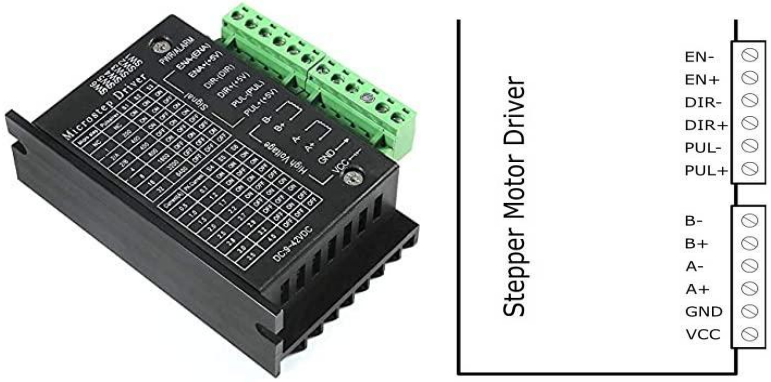

Figure 3.7: TB6600 stepper driver and its schematic diagram

3.2.7 Rotating Cylinder

This component is designed to sort fruits using slots of the same size and suitable depth. Each type of fruit has its dedicated slot, and wheels are used to enable the rotation while the sorting process, To ensure lightweight rotation, this component is constructed using cardboard. Figure 3.8 represents the main shape of our rotating cylinder.

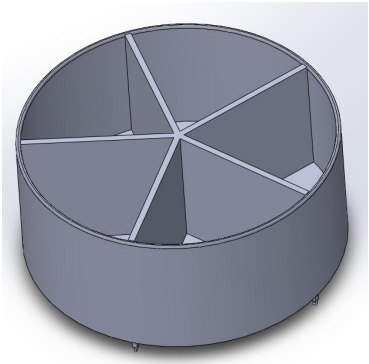

Figure 3.8: Rotating Cylinder

3.2.8 Raspberry Pi as an alternative

The Raspberry Pi4[14] Model B (shown in Figure 3.9) is the latest product in the Raspberry Pi range of computers. This product's key features include a high-performance 64-bit quad-core processor, a dual-display asset at resolutions up to 4K via a pair of micro-HDMI ports, hardware tape decodes at up to 4Kp60, up to 4GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0.

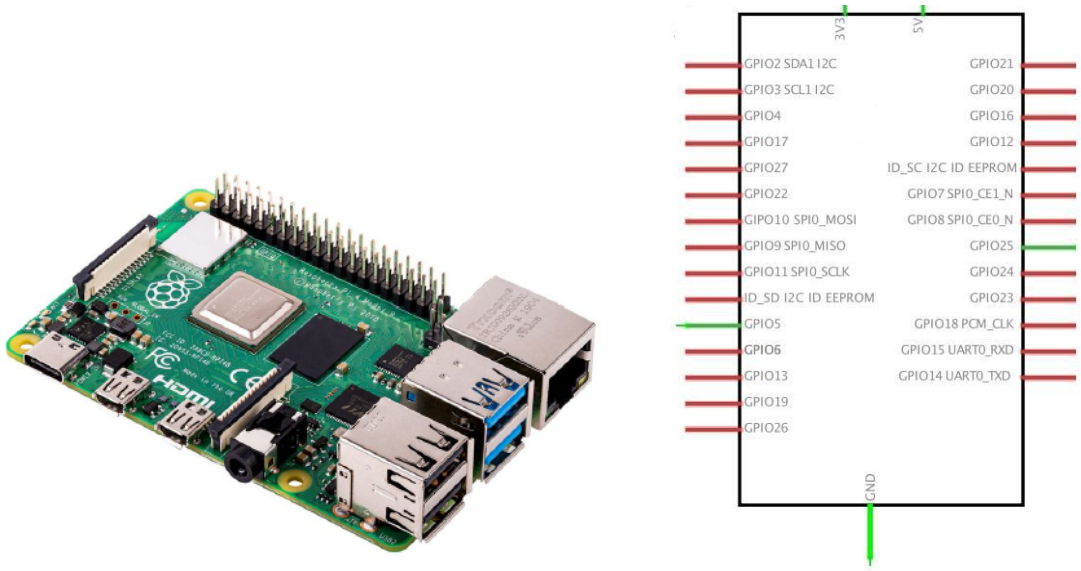

Figure 3.9: The Raspberry Pi Model B and its Schematic diagram, conrad[15]

The following table provides a comprehensive comparison between Arduino Mega and Raspberry Pi microcontrollers. It highlights the key differences and features of both components.

Table 3.2: Comparison between Raspberry Pi 4B and Arduino mega[15]

SBC	**Raspberry Pi 4B**	**Arduino Mega 2560**
CPU	Quad-core ARM Cortex-A72 64-bit @ 1.5 GHz	Atmega2560
GPU	Broadcom VideoCore VI (32-bit)	-

Networking	Gigabit Ethernet / Wifi 802.11ac	Serial communication / Ethernet
USB	2x USB 3.0, 2x USB 2.0	1x USB 2.0
Video Encoder	H264(1080p30)	-
Video Decoder	H.265(4Kp60) H.264(1080p60)	-
GPIO	40-pin GPIO	16-pin GPIO
Price	230$	30$

We opted for the Arduino Mega 2560 in this project since the controller's primary function is to manage system movements. All other processing, including the implementation of deep learning algorithms, is performed on a separate standalone computer. Additionally, the Arduino proves to be a cost-effective alternative compared to the expensive Raspberry Pi. With its abundance of input/output pins, the Arduino Mega 2560 facilitates seamless integration and control of various sensors and actuators. Moreover, the Arduino Mega 2560 is widely acknowledged for its reliability and user-friendly interface, making it the ideal choice for controlling system movements in this project.

Additionally, the fact that the majority of the processing will be done on a separate computer allows us to make use of the Arduino's capabilities without overloading it, ensuring the smooth operation of the system. Overall, the combination of the Arduino Mega 2560 and a separate computer provides the perfect balance of cost-effectiveness and performance for this project.

3.2.9 Stand-alone Processor (Nvidia GTX 1050) VS Raspberry Pi

Table 3.3 presents a clear and concise comparison between the Stand-alone Processor (Nvidia GTX 1050) and Raspberry Pi microcontrollers. It outlines the key distinctions and characteristics of both platforms, allowing for easy evaluation and selection based on individual project needs.

Table 3.3: Stand-alone Processor (Nvidia GTX 1050) VS Raspberry Pi

Specification	Nvidia GTX 1050	Raspberry Pi 4 Model B
Cost	Already Exists	$35 - $75 [16]
Number of Frames	60+ frames per second	30+ frames per second [17]
Computational Power	High performance	Low to Medium performance [18]
Fan Mechanisms	Already Fanned	Requires external fan [19]

As shown in the table, the Nvidia GTX 1050 already exists, and therefore the cost is not a factor to be considered. The Nvidia GTX 1050 can handle 60+ frames per second, making it a more suitable choice for real-time systems that require fast frame processing. In contrast, the Raspberry Pi 4 Model B can handle 30 frames per second in the best cases (without hard computations and deep learning models). Our deep learning model, considering its size, indicates that the Raspberry Pi can achieve a frame rate of at least 5 frames per second. Additionally, we have observed a frame rate exceeding 20 frames per second when utilizing the Nvidia GTX 1050 graphics card.

Regarding computational power, the Nvidia GTX 1050 is much more powerful than the Raspberry Pi 4 Model B, making it an ideal choice for deep learning applications and real-time processing tasks that require significant computational resources.

Finally, the Nvidia GTX 1050 is already fanned using the laptop fan, making it easier to integrate into a project, as it does not require additional hardware to maintain a safe operating temperature. In contrast, the Raspberry Pi 4 Model B requires an external fan to prevent overheating.

Overall, the Nvidia GTX 1050 is the preferred choice for real-time systems that require fast frame processing and deep learning applications that require high computational power. The Raspberry Pi 4 Model B is a more affordable option, but it has lower computational power and requires additional hardware to maintain safe operating temperatures, and is not guaranteed to give enough frames in the case of deep learning models.

3.2.10 Limit Switch

A limit switch is a device that detects the presence or absence of an object or limits the movement of a machine to a predetermined position. It consists of an actuator and a contact block that contains electrical contacts. It's commonly used in industrial and commercial applications to control the movement of equipment and in household appliances to detect the position of a door or a lid. In this project, the limit switch used to detect if the rotational cylinder reached the required slot or not.

Figure 3.10: Limit Switch

3.2.11 Ultrasonic sensor

An ultrasonic sensor is a device that uses sound waves with a frequency higher than the upper audible limit of human hearing (20 kHz) to detect the distance or presence of objects. The sensor emits ultrasonic waves, which then bounce back from objects in their path and are detected by the sensor. By measuring the time it takes for the sound waves to bounce back, the sensor can determine the distance to the object.

Ultrasonic sensors are commonly used in automation, robotics, and security systems. For example, they can be used in parking sensors to detect the distance between a car and an obstacle, or in robotics to detect the presence of objects or obstacles in the robot's path.

This project uses ultrasonic sensors to detect the presence of the fruit on the end of the conveyor, and the LED at the beginning of the conveyor will turn on after it is turned off, indicating that the next fruit can be put on the conveyor. Figure 3.11 presents the main pins of the Ultrasonic sensor.

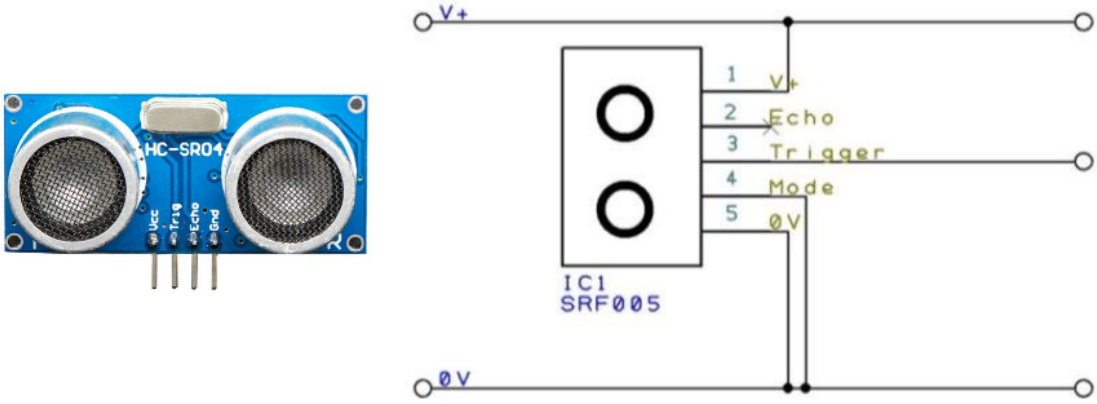

Figure 3.11: Ultrasonic sensor and its schematic diagram

3.3 Software background.

3.3.1 Programming languages.

We use Python[20] as the main programming language because it is simple and easy to handle complex algorithms in machine learning and deep learning.

Python has a wide variety of libraries and frameworks, it's well-structured and well-tested, and it will reduce development and testing time, for example (Keras, TensorFlow, and Scikit-learn).

3.3.2 Framework and Libraries.

3.3.2.1 PyTorch

PyTorch[21] is a Python package that provides Tensors for computation on either the CPU or GPU, resulting in significant acceleration for deep neural networks. It seamlessly integrates with Python, offering a minimal overhead framework. PyTorch is designed to maximize speed by utilizing acceleration libraries like Intel MKL and NVIDIA (cuDNN, NCCL). This makes it a fast solution, regardless of network size. Additionally, PyTorch is renowned for its intuitive and user-friendly design, facilitating linear and effortless usage.

3.3.3 Algorithms

3.3.3.1 YOLO

Yolo[22] is an algorithm that detects and identifies various objects in a picture in real time, Yolo algorithms employ convolutional neural networks (CNN) to detect objects in real-time, the algorithm requires single forward projection through a neural network to detect objects. That means the prediction in the entire picture is made in a single algorithm run. CNN is used to predict various class probabilities and bounding boxes at the same time.

Yolo improves the speed of detection because it can predict objects in real-time, and it provides accurate results with minimal background errors. the most important thing about Yolo is that it has high learning capabilities that help to learn representations of objects and apply them in object detection.

How Yolo works is that we can take an image and divide it into various grids (S*S) in Yolov5 the grid dimensions are (20*20), every grid cell casts B bounding boxes and provides their confidence score, and the cells predict the class probabilities to establish the class of each object. Figure 3.12 briefly shows how Yolov5 works.

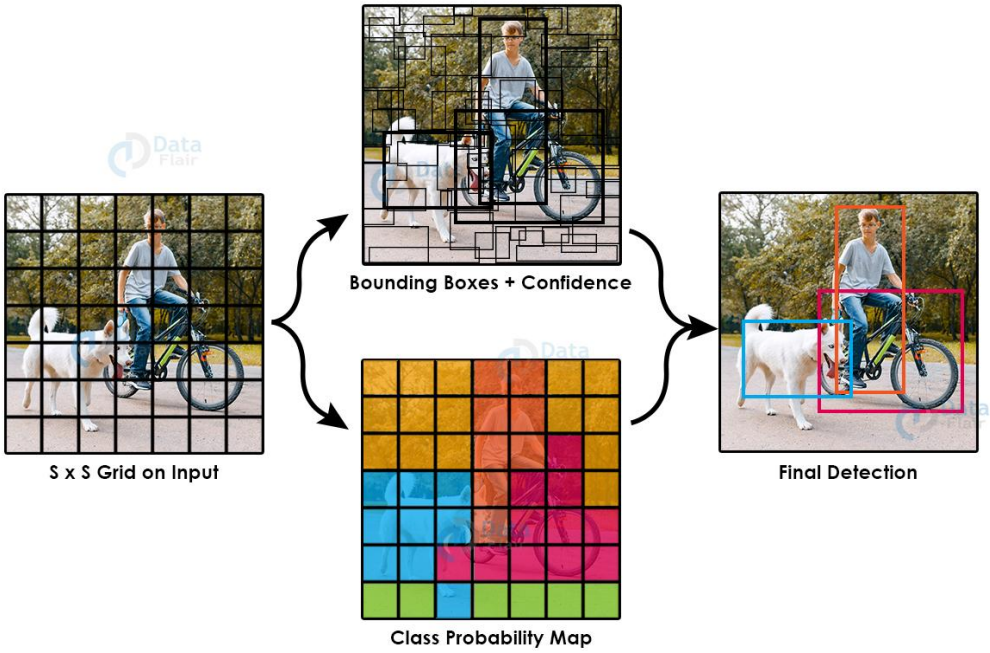

Figure 3.12: YOLO model with SxS grid cell was applied on the input image[23]

3.3.3.2 Yolov5 Model Structure

The image in Figure 3.13 shows the three parts of YOLO.

1. **Backbone:**

   A CNN that aggregates and forms image features at different granularities.

2. **Neck:**

   A series of layers to mix and combine image features to pass them forward to the prediction step in the head. This step will help the model to generalize well on object scaling, Which helps to identify the same object with different sizes and scales to perform better with unseen data.

3. **Head:**

   Consume features from the neck that generate final output vectors with class probability and bounding boxes.

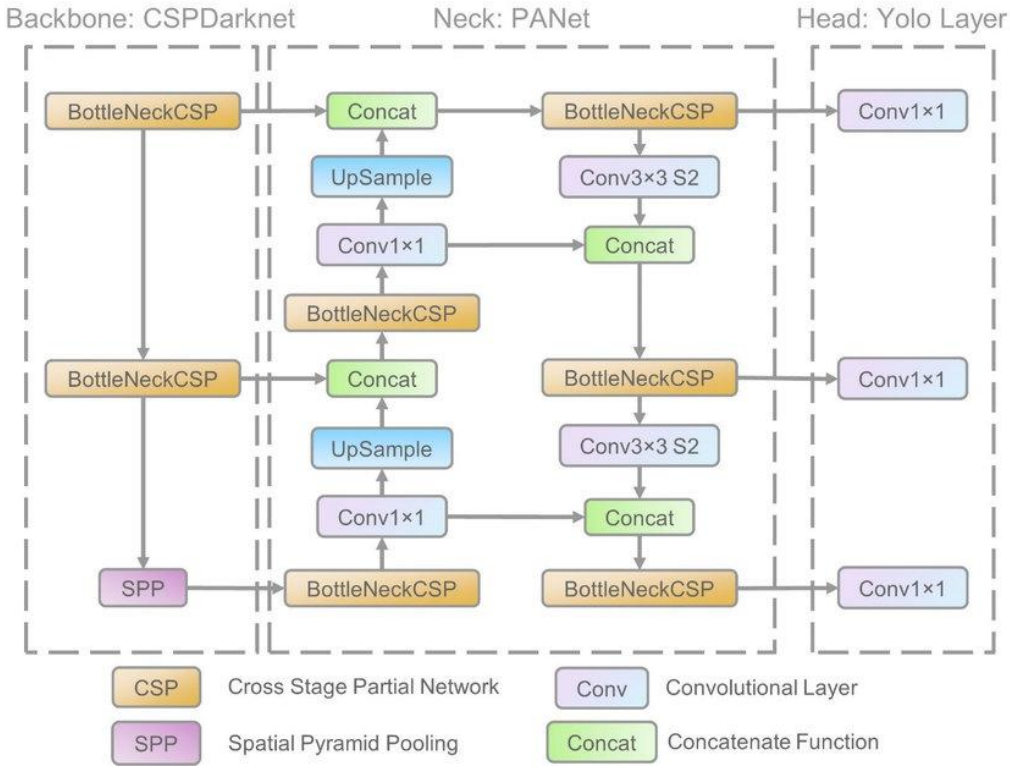

Figure 3.13: Yolov5 network parts[25]

3.3.3.3 Yolo dependencies libraries

These are all the requirements that needed to be installed to run the Yolov5 model, these requirements are compatible with each other.

Table 3.4: YOLO dependencies libraries with versions

Library	Version
Matplotlib	>=3.2.2
NumPy	>=1.18.5
OpenCV-python	>=4.1.2
Pillow	-
PyYAML	>=5.3.1
Scipy	>=1.4.1
Torch	>=1.7.0
Torchvision	>=0.8.1
Tqdm	>=4.41.0
Tensorboard	>=2.4.1
Seaborn	>=0.11.0
Pandas	-
Pycocotools	>=2.0

3.3.3.4 Yolo vs R-Cnn

There is another well-known algorithm used for object detection which is R-CNN[25] (Regional Convolutional Neural Network) that is based on select regions from the image called region proposals by using a selective search approach and then feeding them into two CNNs to predict the presence of an object within these region proposals. Figure 3.14 briefly shows how R-CNN works.

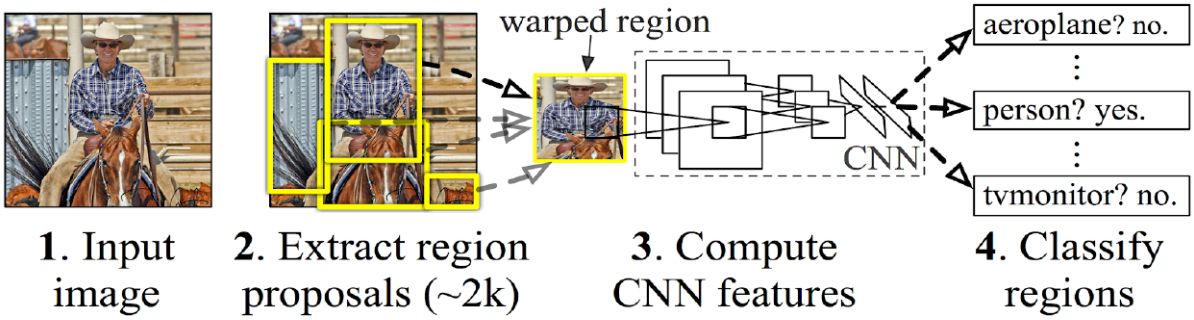

Figure 3.14: R-CNN, towards data science[25]

Comparison based on some experiments[26] between the two algorithms/models shows that:

1. YOLO v5 maintains an obvious advantage in terms of run speed. The small YOLO v5 model runs about two and a half times faster.
2. YOLO v5 works better performance in detecting smaller objects.
3. The detection results are also cleaner with little to no overlapping boxes when using YOLO v5.

The comparison is summarized in the following table (Table 3.5)

Table 3.5: YOLO v5 and Faster RCNN comparison, towardsdatascience[27]

	YOLO v5	Faster RCNN
Inference Speed	✓	
Detection of small or far away objects	✓	
Little to no overlapping boxes	✓	
Missed Objects	✗	✗
Detection of Crowded objects	✓	✓

3.4 Conceptual system description

3.4.1 Detailed design

The general system can be divided into two subsystems, the first one is the monitoring subsystem, the second one is the movement subsystem.

The monitoring subsystem consists of the cameras, which are responsible for transferring live frames to our stand-alone processor. Nvidia GTX 1050 will run the pre-trained deep learning model and classify the fruit into one of the 4 types, the result will be sent to the movement subsystem starting with the Arduino to generate the appropriate commands for the Rotating Cylinder.

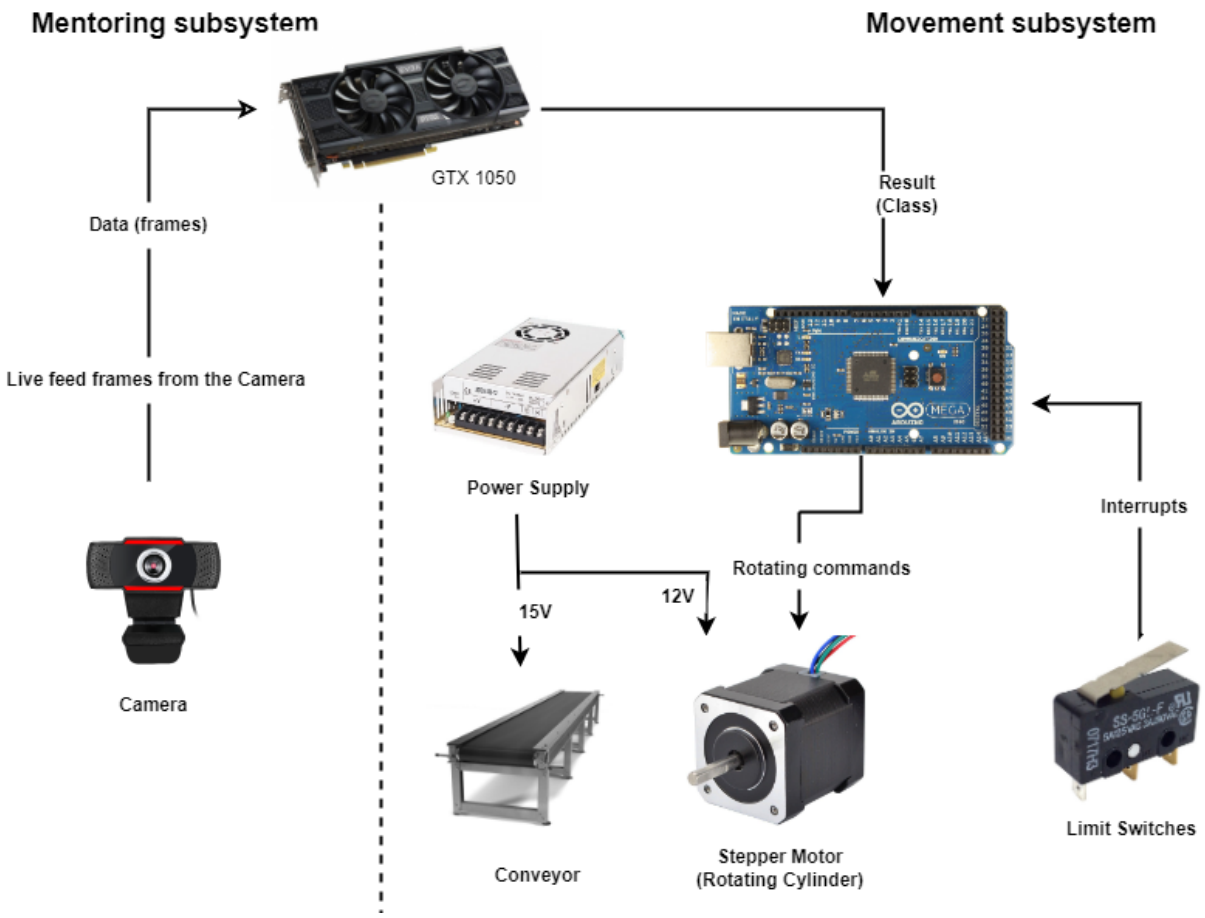

Figure 3.15: System Design

3.4.2 General block diagram.

The figure illustrates how the system components are connected and interact as blocks. The dashed bounding box represents the movement system, including internal components like Arduino, Conveyor, and limit switches. Arduino acts as the interface for this movement system, meaning all connections must pass through it. On the other side of the diagram, the camera is a crucial part of our mentoring system, transmitting frames to the Nvidia GTX 1050. It plays a central role in capturing visuals and forwarding them for processing.

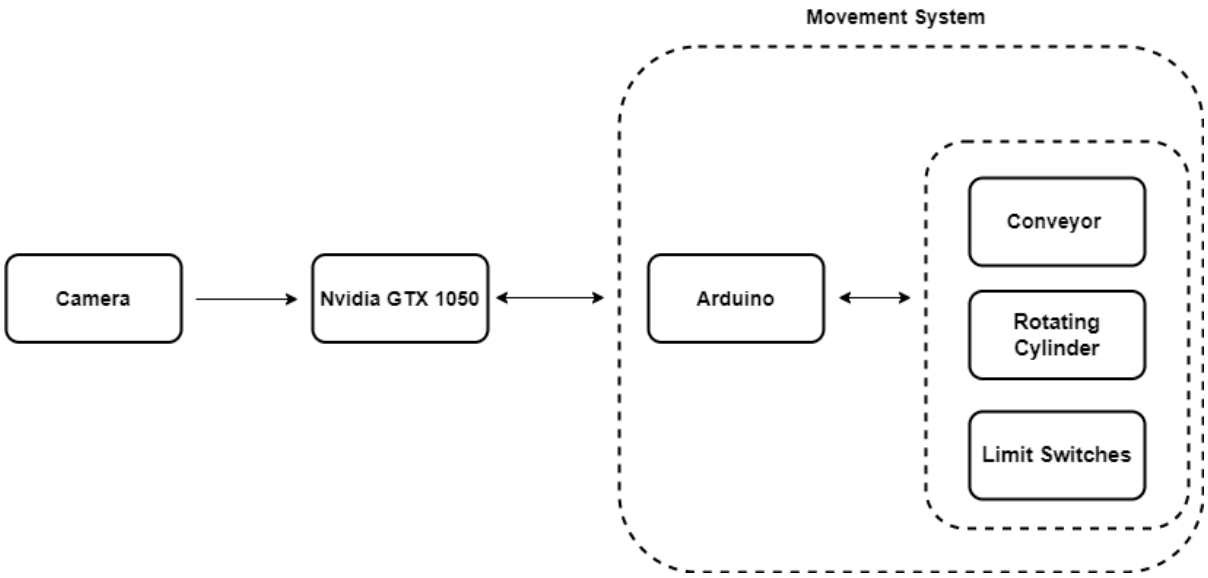

Figure 3.16: General block diagram

3.5 General flowchart

This flowchart describes how the system acts in every frame:

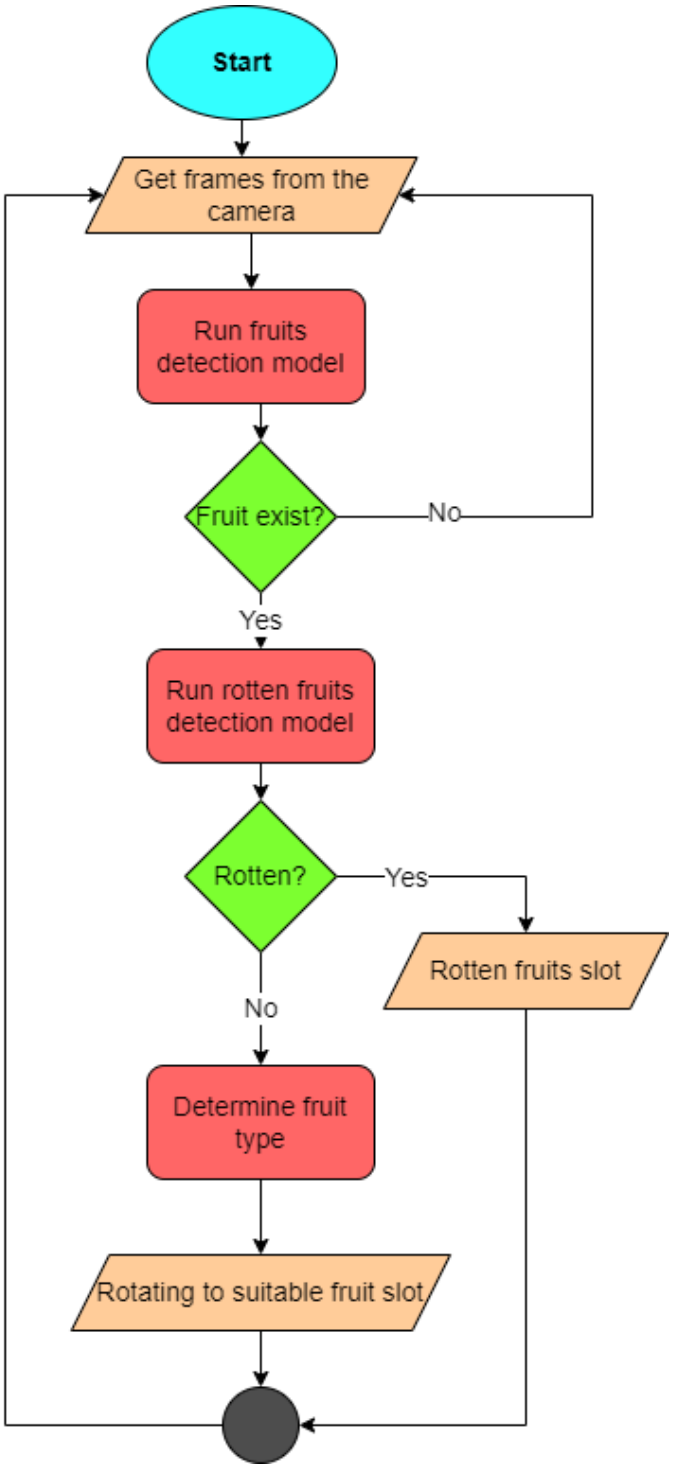

Figure 3.17: General flowchart

3.6 Schematic diagram

The following diagram shows the connection between the Arduino and the necessary components (limit swiches and stepper motor).

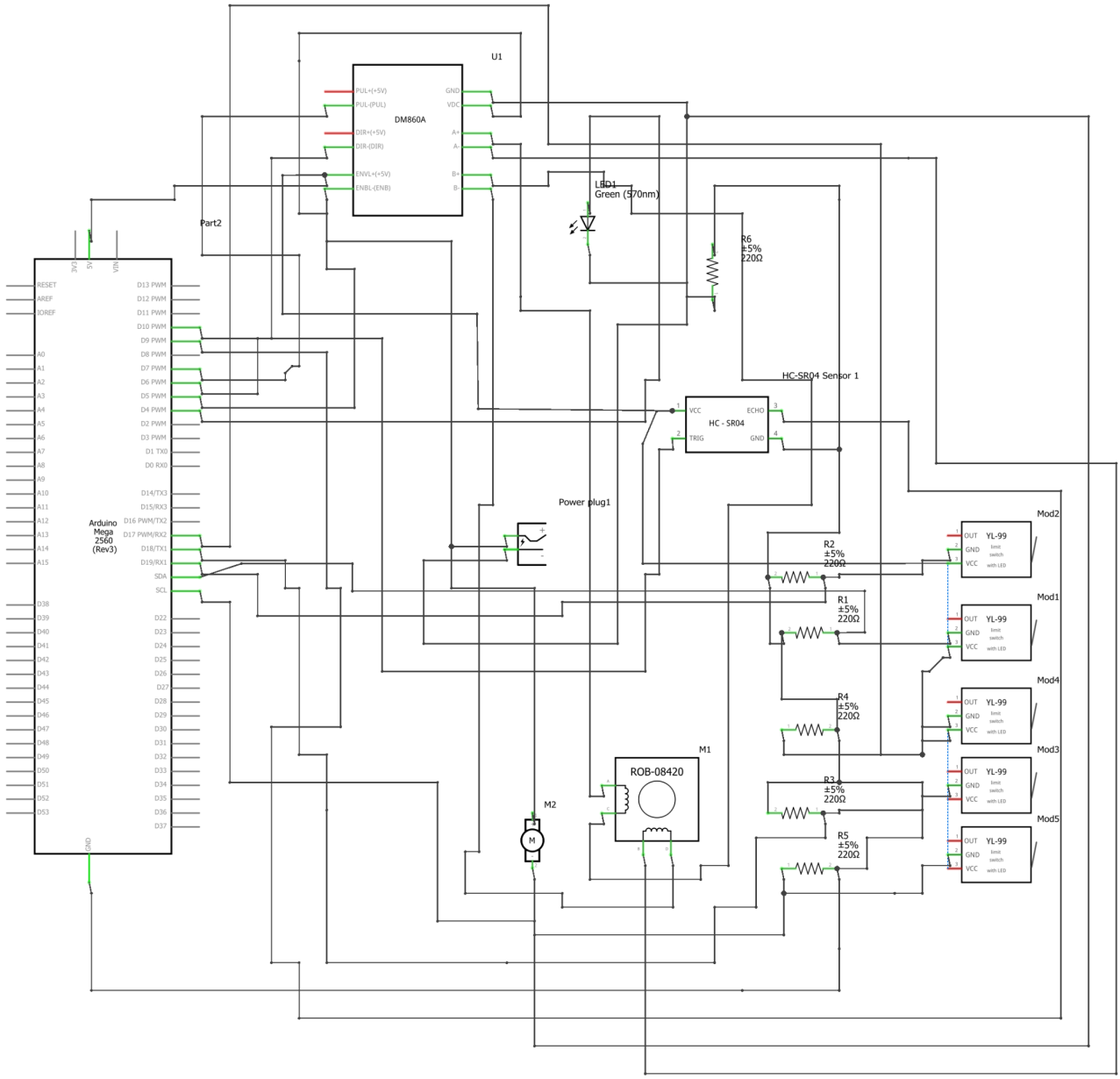

Figure 3.18: Schematic diagram

3.7 CAD Design

In this section, we present a 3D model of the envisioned system and its constituent parts. Figure 3.19 depicts the stand on which the system will be mounted, along with a static arm designed to support the camera.

Figure 3.19: System stand

As illustrated in Figure 3.20, the conveyor belt is where the fruits will be positioned for processing.

Figure 3.20: System conveyor

The rotating cylinder is depicted in Figure 3.21, situated on its stand and equipped with wheels that allow bidirectional movement.

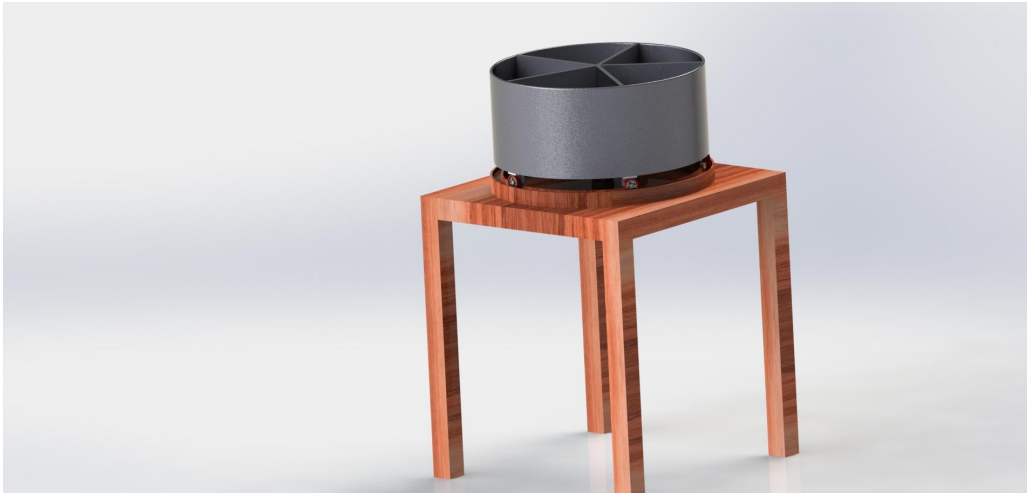

Figure 3.21: Rotating cylinder

The complete system, comprising all of its key components, is depicted in Figure 3.22. These include the stand, which holds the conveyor belt that transports the fruits; the camera, which captures images of the fruits for analysis; and the rotating cylinder, which sorts the fruits into appropriate compartments.

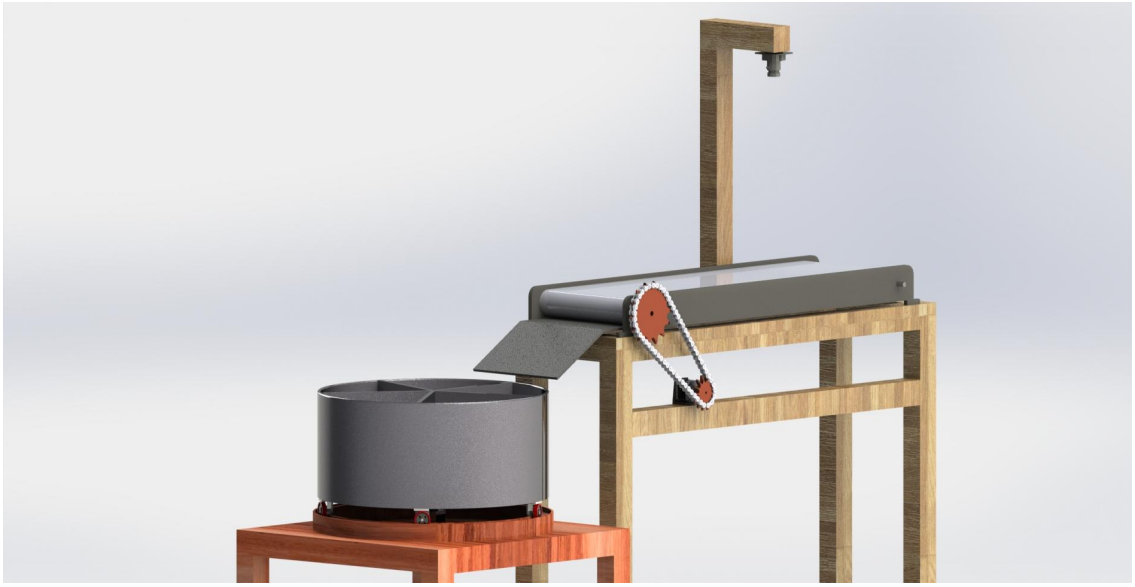

Figure 3.22: 3D model of the system

3.8 Summary

This chapter described the primary hardware components needed to build the system (Arduino Mega 2560, CyberTrack H3-TAA, sj4000cam as an alternative, Conveyor SystemRaspberry Pi as an alternative) and why we should use each one of them.

For the software side of the project, we use Python with PyTorch regarding the simplicity and the number of computations that they provide when it comes to machine learning and deep learning.

The algorithm plays a vital role in the system, making the selection of the YOLO algorithm a pivotal choice. YOLO operates by partitioning images into grids, with each grid responsible for object detection within its boundaries. When compared to other popular algorithms such as R-Cnn, YOLO excels on various fronts, including speed, handling crowded photos, and detecting smaller objects. This superiority stems from the distinctive approach of the R-Cnn algorithm.

Finally, we provided a detailed design for the system which contains two main parts: the monitoring part and the movement part, also a few charts and diagrams were provided to visually show a general view of the flow of the system and the logical connection between its components.

Chapter 4

Implementation

4.1 Preface

The implementation chapter is a detailed description of the technical aspects of this project. It covers the hardware and software implementation, including the model training process. The chapter provides a comprehensive overview of the challenges encountered during the implementation and the solutions developed to address them. The aim is to provide guidance and insight to readers who may be working on similar projects. This chapter showcases technical expertise and highlights the ability to apply knowledge and skills to real-world problems.

4.2 Hardware implementation

In the previous chapter, we showed how the components interconnect with each other. This section will present how we assembled these components and how we connected them with the Arduino Mega 2560.

The system consists mainly of a conveyor belt shown in Figure 4.1, on which all of the other components will be assembled.

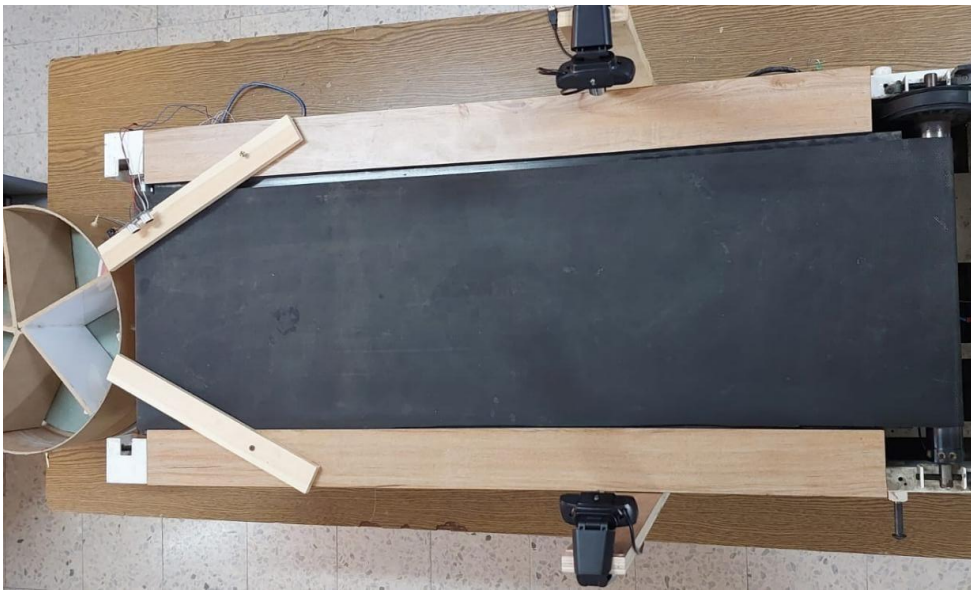

Figure 4.1: Conveyor belt system

To commence the implementation of the fruit sorting system, the initial phase involved installing two cameras positioned above the conveyor belt. The placement of the cameras was crucial, with a 35cm distance from the conveyor being determined as optimal. This distance allowed for complete coverage of the fruit within the camera's field of view, enabling the detection of any rotten areas on both sides of the fruit, while also ensuring that only the fruit is captured and not the surrounding objects.

To avoid any potential overlap or collision between fruits, an ultrasonic sensor was installed at the end of the conveyor system. This sensor was set up to enforce a minimum time gap of 8 seconds between the passage of two consecutive fruits. This time interval represents the worst-case scenario where a new fruit can only be introduced after the previous one has completely cleared the conveyor. By implementing this system, the sorting process can proceed smoothly and with minimal interruption.

After successfully establishing the connection between the stand-alone processor and the two cameras, the next step involved connecting the remaining components. The following sequence was adopted to connect these components.

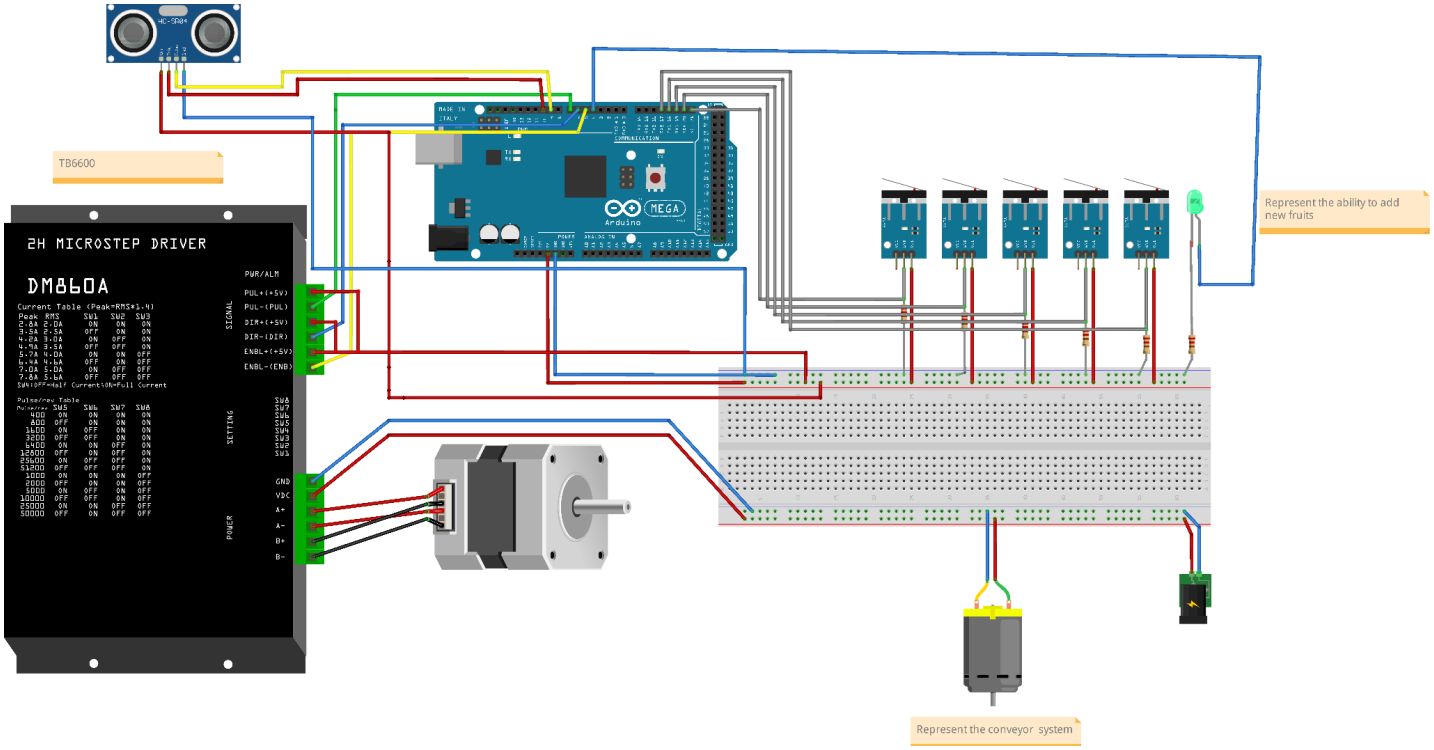

Figure 4.2: Detailed connections

Figure 4.2 exhibits the connections within the movement subsystem, where the DC motor represents the conveyor system, and the LED signifies the ability to add new fruits or products. In practical application, the subsequent image showcases the coordinated operation of the limit switches, stepper motor, and rotational cylinder.

The subsequent image illustrates the implementation of limit switches to enable the cylinder to stop accurately on the required slot. The interrupt principle was utilized to achieve immediate stoppage.

In addition, this image provides an overview of the types of fruits being sorted in this project, namely apple, carrot, banana, and orange. It is also evident that a dedicated slot has been allocated for the detection and segregation of any rotten fruits. Hence, this image offers a comprehensive view of the sorted fruits and the sorting process.

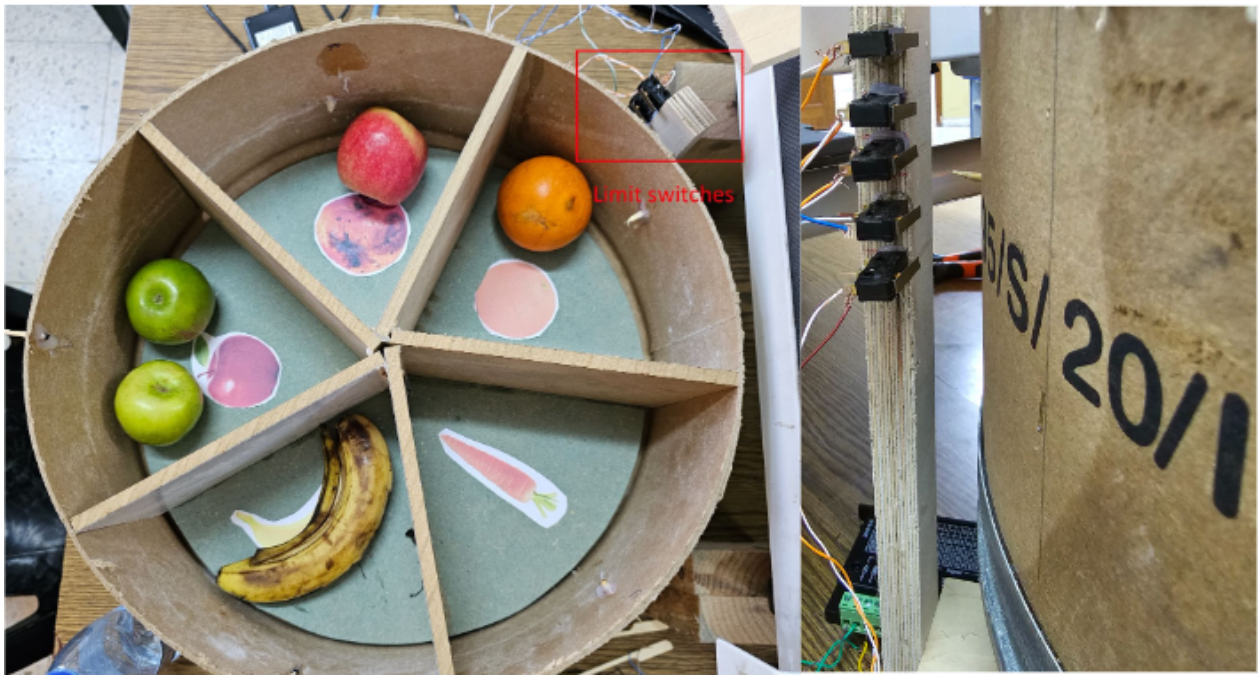

Figure 4.3: Rotational cylinder with limit switches

Figure 4.4 shows how all the components work together, including the conveyor subsystem, the rotational cylinder, and the cameras. By working seamlessly together, these elements improve the efficiency, speed, and accuracy of the system, resulting in increased productivity and higher-quality output.

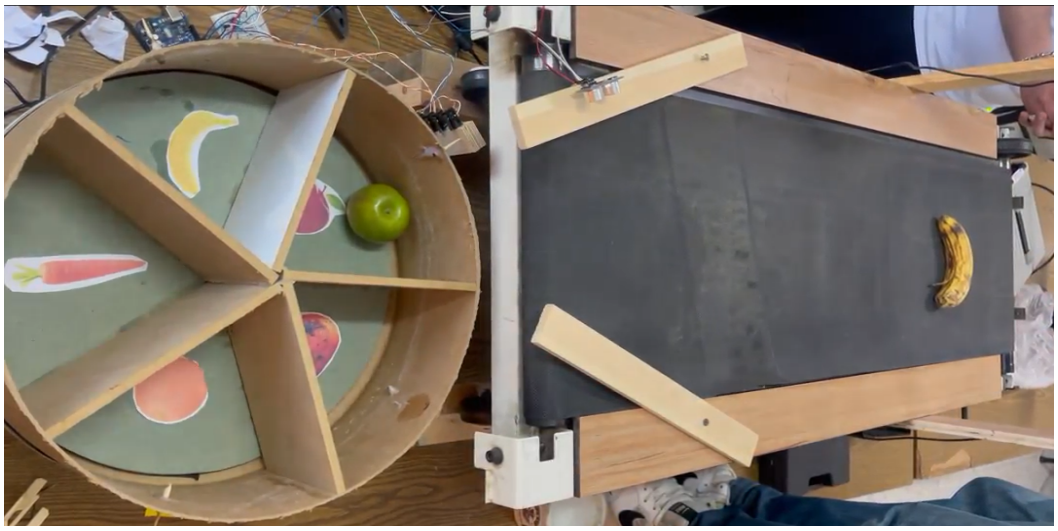

Figure 4.4: Complete System Setup

4.3 Software implementation

This section provides an in-depth account of the software components involved in the system implementation, along with a comprehensive overview of the various phases that were undertaken to realize the project.

4.3.1 Training YOLOv5 object detection models

The YOLOv5 algorithm comes with a collection of pre-trained models that utilize the MS COCO dataset to classify objects across 80 different classes (such as "person", "car", "bicycle", "boat", "bird", etc.). However, these classes may not be suitable for our specific project requirements. Therefore, it was necessary to retrain the algorithm using a customized set of classes and images that would cater to our needs.

Dataset

To train YOLOV5 for object detection in this project, the first step was to gather a suitable dataset that includes annotations and bounding boxes. The dataset was selected based on its compatibility with the hardware specifications and slots. The dataset used in this project consists of four different classes, namely orange, apple, banana, and carrot.

In this project, a dataset of 1604 images was collected for four classes, including 482 banana images, 438 apple images, 293 carrot images, and 397 orange images. These images were uploaded to Roboflow for annotation, and each image was annotated with a bounding box. The resulting annotations were saved in a .txt file, where each line described a bounding box for an object in the image, as shown in Figure 4.5.

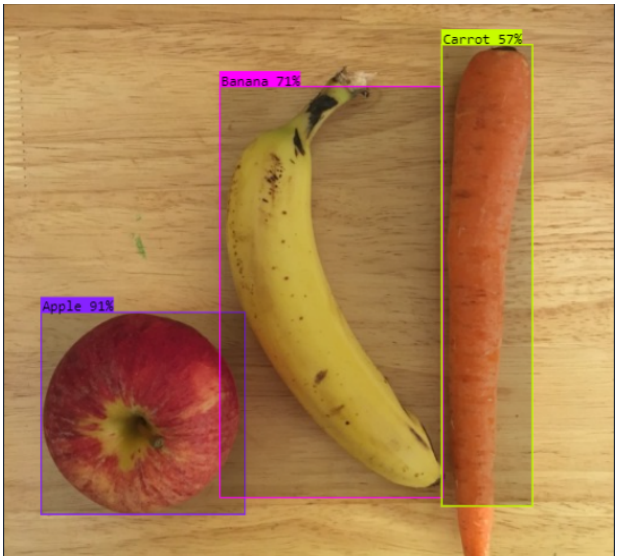

Figure 4.5: Roboflow annotation example

As shown in figure 4.5, there are three objects in the image, namely one apple, one banana, and one carrot. The bounding box annotations for each object are described in a single line. The format of each line is as follows: (class, x center, y center, width, height). Each row represents a single object. The values for x center, y center, width, and height are normalized by the image's dimensions, which means that they have values between 0 and 1.

Figure 4.6 displays the results obtained from Roboflow for the previous image. It shows the predicted objects, and their respective bounding boxes, as generated by Roboflow.


```
"predictions": [
  {
    "x": 116.5,
    "y": 342.5,
    "width": 171,
    "height": 169,
    "confidence": 0.907,
    "class": "Apple"
  },
  {
    "x": 274.5,
    "y": 241,
    "width": 187,
```

```
    "height": 344,
    "confidence": 0.707,
    "class": "Banana"
  },
  {
    "x": 405,
    "y": 227,
    "width": 76,
    "height": 386,
    "confidence": 0.567,
    "class": "Carrot"
  }
]
```


Figure 4.6: Object Detection Results: Predicted Objects and Bounding Boxes

The objective of this project was to detect rotten fruits in apples and oranges. To achieve this, we used two separate datasets and built a custom model for each fruit type. The datasets were selected based on the number of images available and the clarity of the background.

The dataset for rotten apples consisted of 978 images, with 520 images of fresh apples and 551 images of rotten apples. Similarly, the dataset for rotten oranges comprised 353 images, with 161 images of fresh oranges and 187 images of rotten oranges. An example of how the models detected the rotten apples is shown in the following figure.

Figure 4.7: Example of Rotten Fruit Detection in Apples

Dataset Partitioning

In this phase of the project, the fruits dataset was divided into three sets: training, validation, and testing. The training set contained 80% of the data, while the validation and testing sets contained 10% each. To accomplish this, we used Roboflow, which split the dataset and generated the corresponding folders for each set. Figure 4.8 illustrates the number of images in each set after the dataset was split.

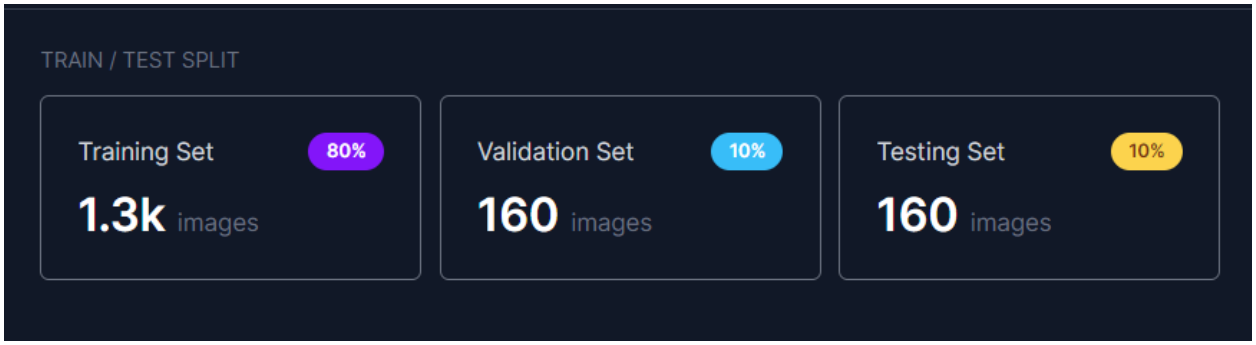

Figure 4.8: Dataset Split: Training, Validation, and Testing Sets

To train the remaining two models, the images were split based on the following percentage for the rotten apples model:

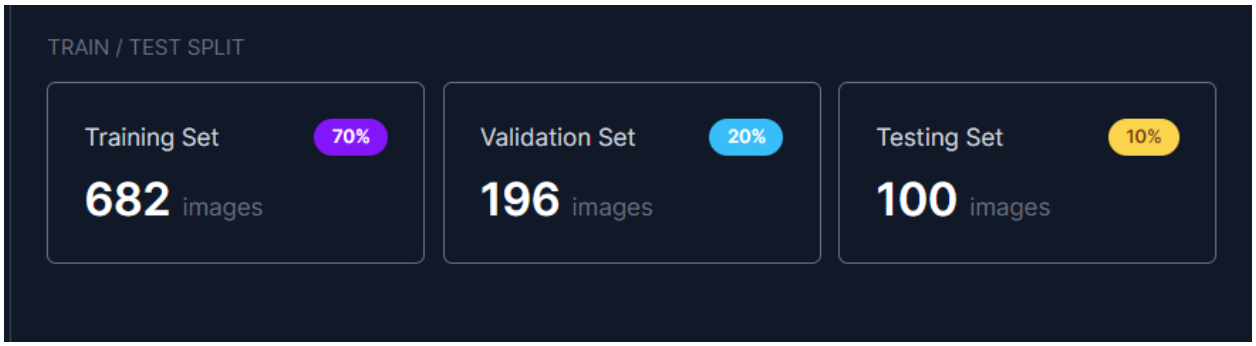

Figure 4.9: Rotten Apple Dataset Split: Training, Validation, and Testing Sets

On the other hand, the images were split based on the following percentage for the rotten oranges model:

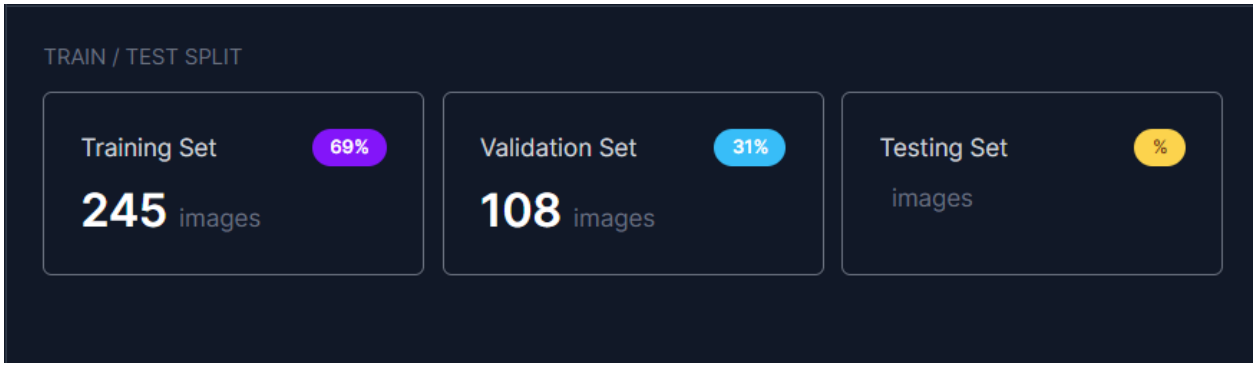

Figure 4.10: Rotten Oranges Dataset Split: Training, Validation, and Testing Sets

The lack of images in the dataset forced us to exclude the training set from the training process. Instead, we utilized the images from the training and validation sets.

Configuration File

The data configuration YAML file comprises important details for the dataset that we used to train our model. These details include the locations for the train, test, and validation sets, the number of classes in the dataset (denoted by "nc"), and the names of the classes in the dataset. In the code, the index of a class in the list corresponds to its class number. The contents of the data.yaml file in the first model is shown in Figure 4.11.

```
train: ../train/images
val: ../valid/images
test: ../test/images
nc: 4
names: ['Apple', 'Banana', 'Carrot', 'Orange']
```

Figure 4.11: Data YAML file content

Training Options

To ensure the model's network is properly configured for training, there are various settings that must be specified to optimize its performance. These settings include:

- **img**: the size of the image, which needed to be resized to a square shape. For this project, an image size of 640 was used.
- **batch**: the batch size of the network, set to 16 for this project.
- **epochs**: the number of epochs to train for, set to 100 for this project.
- **data**: the path for the data.yaml file that contains the dataset details.
- **weights**: the pre-trained weights to use for training, set to yolov5l.pt weights that come with Yolov5.
- **cfg**: the model architecture. There were four options available: yolov5s.yaml, yolov5m.yaml, yolov5l.yaml, and yolov5x.yaml. yolov5s.yaml was selected for this project due to its small size and simplicity, making it suitable for deployment on Raspberry Pi.

Train the Model

We chose Google Colab as our platform for training the model due to its high computational power and memory usage requirements. Colab provides free GPUs and allows working on the notebook for up to 12 hours. The first step in the training process was to clone the YOLOv5 GitHub repository and install the required dependencies from the requirements.txt file that comes with the YOLOv5.

After that, we uploaded our dataset, which in our case was uploaded directly from Roboflow. We then specified the training options we mentioned above and started the training process, which

ran for six hours for all models (Rotten and fruits detection). Finally, we evaluated the model's performance resulting from the training process (see Chapter 5), and we used it for our system.

Using Colab for training our model was highly beneficial as it provided us with the necessary resources to perform this computationally intensive task. The process of cloning the YOLOv5 GitHub repository and installing the required dependencies was straightforward and seamless. Furthermore, uploading our dataset directly from Roboflow was convenient and saved us time. Overall, the training process was smooth, and we were able to successfully obtain a trained model that we could use for our system.

4.3.2 Coding

This section will cover the installation and setup of libraries, as well as hardware functions, on the standalone processor and Arduino Mega 2560.

Libraries Installation

To load a custom YOLOv5 model using PyTorch Hub in Python, the "torch" library was installed. All the necessary libraries specified in the "requirements.txt" file were then loaded to enable running the models locally. The Serial library was utilized to establish communication between the standalone processor (Nvidia GTX 1050) and the Arduino Mega 2560.

Arduino Setup Function

The following actions were taken to set up the hardware:

- The stepper motor, LED, serial communication, and ultrasonic sensor pins were initialized.
- The ultrasonic pins trig and echo were initialized to enable distance calculations.
- The limit switches pins were initialized as INPUT_PULLUP pins to enable interpretation.
- An interrupt service routine was assigned for all pins.

Interrupt Service Routine Function

- An interrupt service routine is established for each INPUT_PULLUP pin to respond to changes in the limit switches.
- Each function implemented will return the current slot position of the rotational cylinder.

- Each function implemented will also immediately trigger the stepper motor to stop if the rotational cylinder reaches the desired slot.

FruitDistance Function

- The function accepts the trigger and the echo of the ultrasonic sensor.
- The function returns the distance reading from the sensor.

Arduino Loop Function

- The Loop function will begin by waiting for a message from the series communication.
- The received message will contain a number indicating the required slot.
- The Loop function will then initiate the stepper motor movement and turn off the LED to indicate that no new fruit can be added.
- The Loop function will be interrupted by the limit switches when they are activated.
- The FruitDistance function will be called by the Loop function to determine if the fruit has reached the end of the conveyor.
- Once the fruit has been dropped, the LED will turn back on and the Loop function will return to waiting for a message from the series communication.

Arduino Loop Pseudo Steps

```
1. Wait for a message from the serial communication.
2. Read the message to determine the required slot.
3. Initialize stepper motor movement.
4. Turn off LED.
5. Wait for interrupt from limit switches.
6. If interrupt was received:
     6.1 Determine the current slot position using interrupt service
routine functions.
     6.2 If current slot position equals the required slot:
          6.2.1 Immediately trigger the stepper motor to stop.
          6.2.2 Turn on the LED to indicate that a new fruit can be added.
7. Call the FruitDistance function to determine if the fruit has reached
the end of the conveyor.
8. If the fruit has been dropped:
     8.1 Turn on the LED to indicate that a new fruit can be added.
9. Return to waiting for a message from the serial communication.
```

Object detection Pseudo code

This pseudocode captures frames from the two cameras, tracks the number of frames that have detected fruits, performs a rotten check on 45 frames containing fruits (specifically for Apple or Orange), makes a decision based on the check, and sends the decision using a serial connection.

```
Initialize variables: fruit_counter = 0, frames = []

Start capturing frames from the camera

while True:
    Capture frame from the camera

    Process frame (fruit detection, rotten check for Apple or Orange, etc.)

    If frame contains a fruit:
        Increment the fruit_counter

    Save the frame in the frames list

    If fruit_counter equals 45:
        Send frames to rotten check (for Apple or Orange)

        Determine decision based on the result of the rotten check

        Send the decision via serial
        Clear the frames list
        Reset the fruit_counter

Release resources (camera, serial connection)
```

4.4 Summary

In this chapter, a detailed overview was provided regarding the hardware components utilized in the project, which included a conveyor belt, stepper motor, ultrasonic sensor, and integrated cameras. The chapter then went on to explain the process of training the YOLOV5 model. Moreover, the chapter discussed how the trained model was integrated into a standalone processor as well as an Arduino board. The integration of the model with these hardware components was explained in detail with the serial communication.

Chapter 5

Validation and Results

5.1 Preface

This chapter is dedicated to discussing the testing of all system components, along with presenting the results obtained from this testing. The primary objective of this testing is to verify that all functions operate as intended and without errors. We conducted testing on each individual component to ensure that it performed its designated tasks as expected.

5.2 Hardware Testing

5.2.1 Testing Arduino Mega 2560

We performed thorough testing on the Arduino Mega 2560 microcontroller board, which serves as a controller for our movement system. This testing involved verifying the proper functioning of all inputs and outputs, as well as ensuring the compatibility of the board with the other system components. We also tested the code uploaded to the board to confirm that it executes correctly and without errors. The Arduino Mega 2560 board passed all tests successfully, and we are confident in its ability to effectively control the system's functions.

5.2.2 Testing Stepper Motor

We tested the stepper motor that drives the rotational cylinder to make sure it is reliable. We checked the wiring and connections, and the accuracy and consistency of movements at 200 RPM. We also ensured it could respond to signals from the Arduino board without any problems or vibrations. To check its strength, we did a stall test to see if it could hold a weight of over 8 kilograms. The stepper motor passed all of these tests and worked smoothly and accurately at the specified speed. Overall, we are confident that it meets all the requirements for our project and can drive the rotating cylinder precisely and reliably.

5.2.3 Testing Ultrasonic Sensor

We handled two main tests with the Ultrasonic Sensor and Arduino Mega 2560. The first test focused on distance testing when the fruit reached the end of the line. During this test, we encountered some issues as the height of the carrot was different from that of the apple.

To overcome this problem, we adjusted the sensor's height to make it capable of detecting all types of fruits accurately.

The second test focused on timing, and fortunately, it was not a major issue for us. We simply needed to ensure that the sensor triggered the Arduino at the proper time. The sensor was used to indicate that a fruit had just reached the end of the conveyor, and we were able to verify that the timing was correct. Overall, by conducting these tests, we ensured that the ultrasonic sensor was working properly and providing accurate information for our application.

5.2.4 Testing Webcams

We conducted thorough testing on our two webcams to ensure that they were properly set up and aligned to provide full coverage of the fruits from all angles except the bottom. We tested the cameras by capturing images and video footage and carefully reviewing the output to make sure that the fruits were clearly visible and well-defined in the captured media.

In addition to ensuring full coverage of the fruits, we also verified that the resolution of the cameras was suitable for use with our deep learning model. This was a critical step to ensure accurate object recognition and tracking speed, as well as the ability to capture a sufficient number of frames per second. Through testing, we were able to achieve a consistent frame rate of 50 frames per second with these cameras, providing us with the high-quality data needed for our models to perform at their best.

5.2.5 Testing The Conveyor System

We conducted a test on the conveyor system that is powered by a 15-volt motor. During the test, we observed that it took 8 seconds for the fruit to travel from the beginning of the conveyor to the end. This information allowed us to adjust the timing and positioning of the sensors and cameras accordingly, ensuring that our AI model could accurately detect and track the fruits as they moved along the conveyor.

5.3 Software Testing and Evaluation

The majority of our system's software testing pertains to object detection models. We will detail the stages we underwent in training the model until we achieved the intended outcomes.

It's important to note that while we evaluate our model using metrics like mAP 0.5, mAP 0.5:0.95, precision, and loss, we also test our model on various objects in different settings. This helps us gain a better understanding of how well the model performs in real-world scenarios. By doing so, we are able to obtain a confidence value that indicates the level of accuracy and reliability of our model.

mAP 0.5 is a metric that stands for the mean Average Precision (mAP) at the IoU (Intersection over Union) threshold of 0.5. On the other hand, mAP 0.5:0.95 is an average mAP calculated over different IoU thresholds, ranging from 0.5 to 0.95. In order to calculate the mAP metric, the following steps are taken: first, the confusion matrix is calculated, which includes TP (True Positive), FP (False Positive), TN (True Negative), and FN (False Negative). From there, the precision and recall metrics are calculated, followed by the area under the precision-recall curve. Finally, the average precision is measured.

To calculate the mAP metric, the Average Precision (AP) for each class is determined and then averaged over a certain number of classes. This process is represented mathematically using the formula [28]:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i$$

Here, N represents the total number of classes, while APi denotes the Average Precision for the ith class. By computing the mAP in this manner, we obtain a more comprehensive understanding of the model's overall performance.

Precision is a metric that measures the proportion of bounding box predictions that are correctly classified as true positives (TP) over the total number of bounding box predictions (TP + FP).
On the other hand, Recall measures the proportion of true bounding boxes that were correctly predicted as true positives (rTP) over the total number of true bounding boxes (TP + FN) [29]. The recall is a measure of a model's ability to correctly identify all positive instances. A high recall value indicates that the model is able to correctly identify a large percentage of positive instances, while a low recall value indicates that the model is missing a significant number of positive instances.

Box loss, also known as bounding box regression loss, is a type of loss function that calculates the Mean Squared Error (MSE) between the predicted bounding box and the true bounding box for a given object. This helps assess how well the algorithm is able to locate the center of an object and how accurately it can predict the bounding box that encompasses the object [29].

Objectness loss, commonly referred to as Obj loss, is a metric that measures the probability of an object existing in a particular region of interest. This helps determine whether an object is present in a specific area of the image or not [29].

Classification loss, or cls loss, is a metric that measures how well the algorithm is able to classify a given object into the correct category. This helps determine the accuracy of the algorithm in predicting the correct class of a given object [29].

5.4 Training YOLOv5 Object Detection Models

This section presents the training process and outcomes of our three models: Fruits, Rotten Apples, and Rotten Oranges. We will assess the improvements and perform an experimental analysis.

5.4.1 Fruits Detection Model

We utilized a varied dataset consisting of 1604 images, gathered for four distinct categories: 482 banana images, 438 apple images, 293 carrot images, and 397 orange images. During training, the following numbers of images were used for each category: 356 for apples, 385 for bananas, 230 for carrots, and 318 for oranges.

5.4.1.1 Training our First Fruits Detection Model

We conducted our training on the provided dataset, training our model from scratch without using any pre-trained models. As our dataset is moderately sized, the model would benefit most from training from scratch. Due to the limitations of the hardware provided by Colab, with only one GPU allowed, we used a batch size of 16, which is the maximum size allowed. For image size, we used 416x416 as the minimum, as using larger sizes would slow down the training process. We started with the default image size to test the results. The training was conducted for 100 epochs, taking 2 hours, 29 minutes, and 21 seconds to complete. The results of the training are presented in Figure 5.1. We noticed a rapid improvement in the model's precision, recall, and mAP metrics, but this progress plateaued around 80 epochs, which aligns with our desired behavior. Based on the figure, the optimal values for precision, recall, and mAP 0.5 were determined to be 0.90, 0.91, and 0.95, respectively.

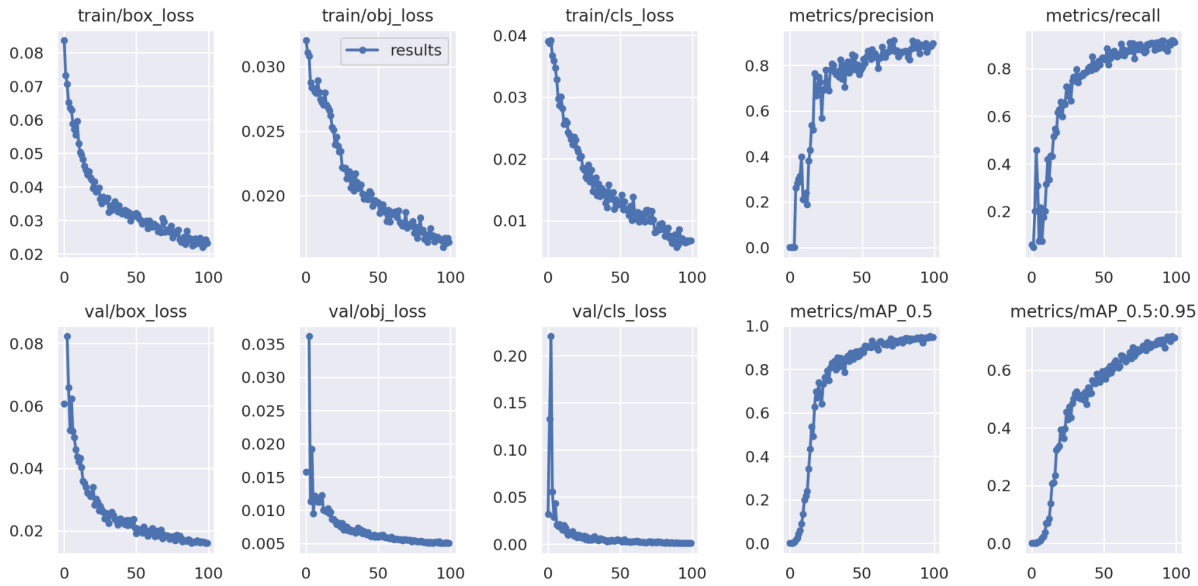

Figure 5.1: Metrics for our first model

5.4.1.2 Experimental Analysis of The First Fruits Detection Model

Upon completing the previous model, we tested it using our webcam and identified misclassifications, particularly for the banana class. To investigate the issue, we employed YOLOv5, which generated a confusion matrix displayed in Figure 5.2.

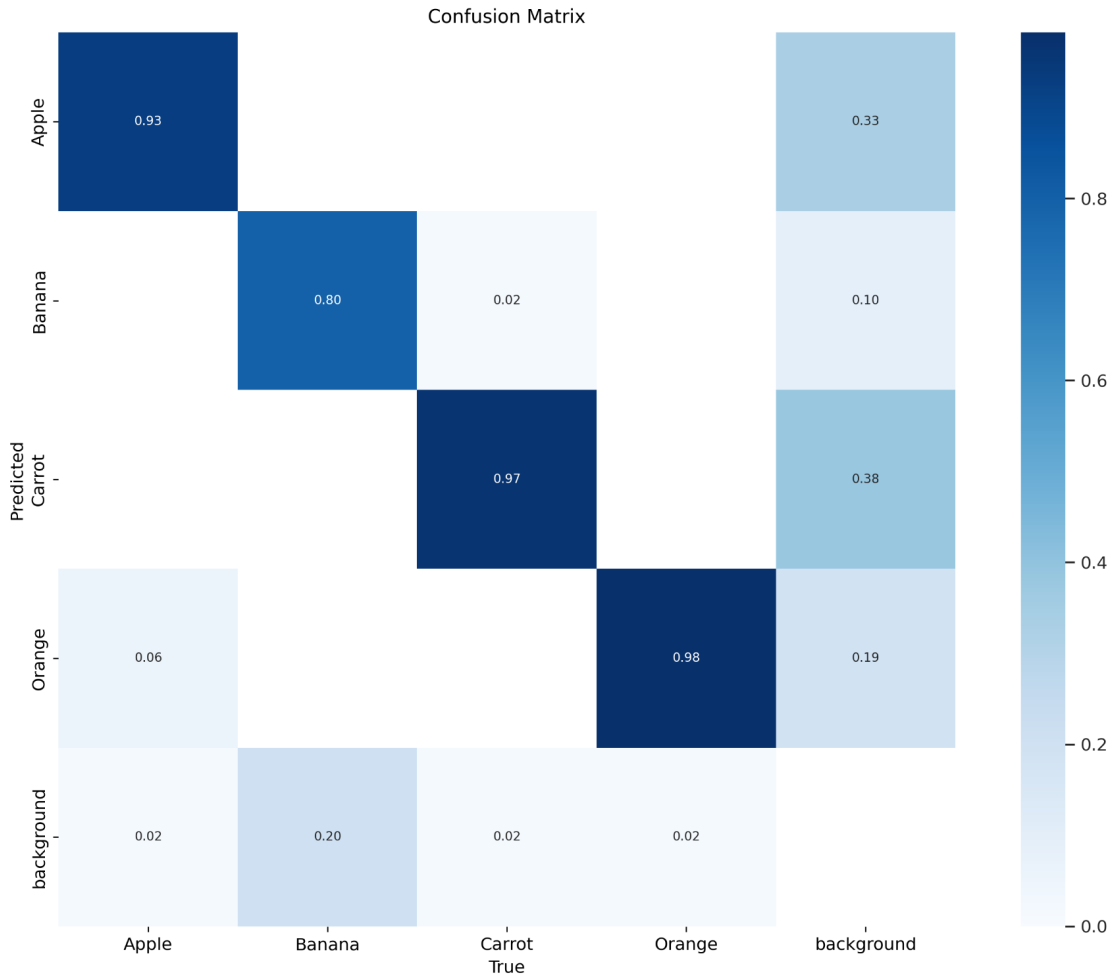

Figure 5.2: Confusion matrix for our first model

The preceding figure displays an accuracy rate of 80% for the banana class, indicating a 20% error rate with the background. This was a significant issue, leading us to determine that the model required improvement to resolve the problem, the problem shown clearly in Figure 5.3.

Figure 5.3: Problem in our first model

5.4.1.3 Training our Second Fruits Detection Model

To address this issue, we opted to use YOLOv5s (YOLOv5 small) as a pre-trained model and leverage its weights. We also increased the training epochs from 100 to 200 while keeping all other training settings constant. The resulting matrices are displayed in Figure 5.4, it provided valuable insights, leading to the determination of the optimal values for precision, recall, and mAP0.5 as 0.95, 0.97, and 0.98, respectively.

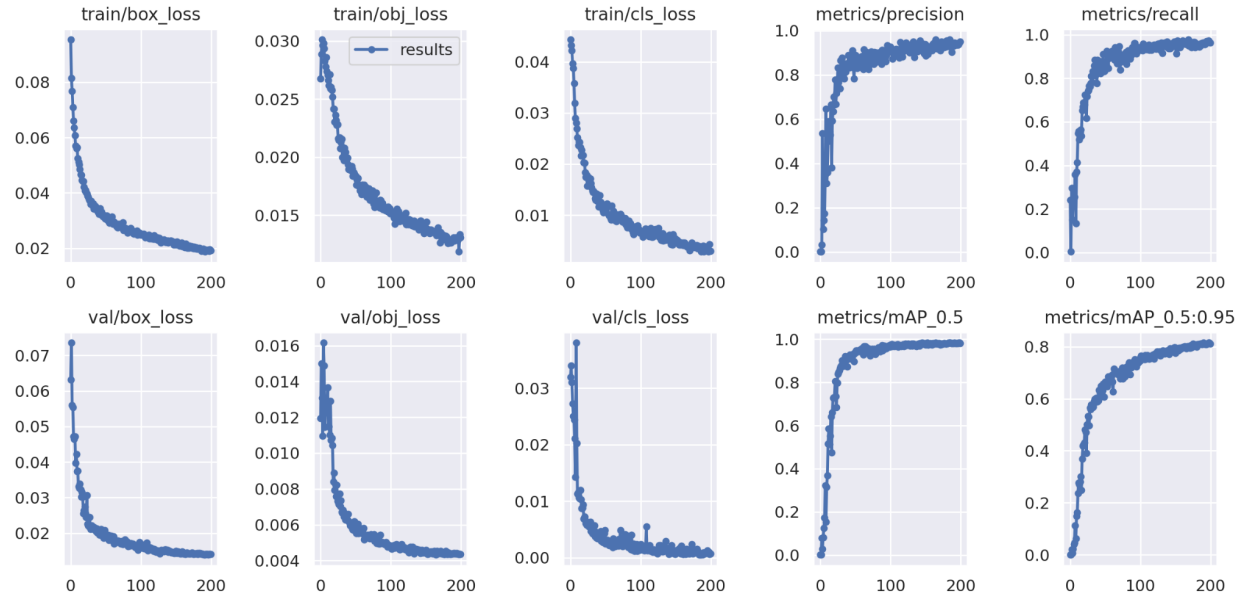

Figure 5.4: Metrics for our second model

5.4.1.4 Experimental Analysis of The Second Fruits Detection Model

The previous matrices indicate that the model had a precision rate of 95%. We then tested the model using the webcam and found that it produced accurate results without any misclassifications. This is evident from the confusion matrix shown in Figure 5.5.

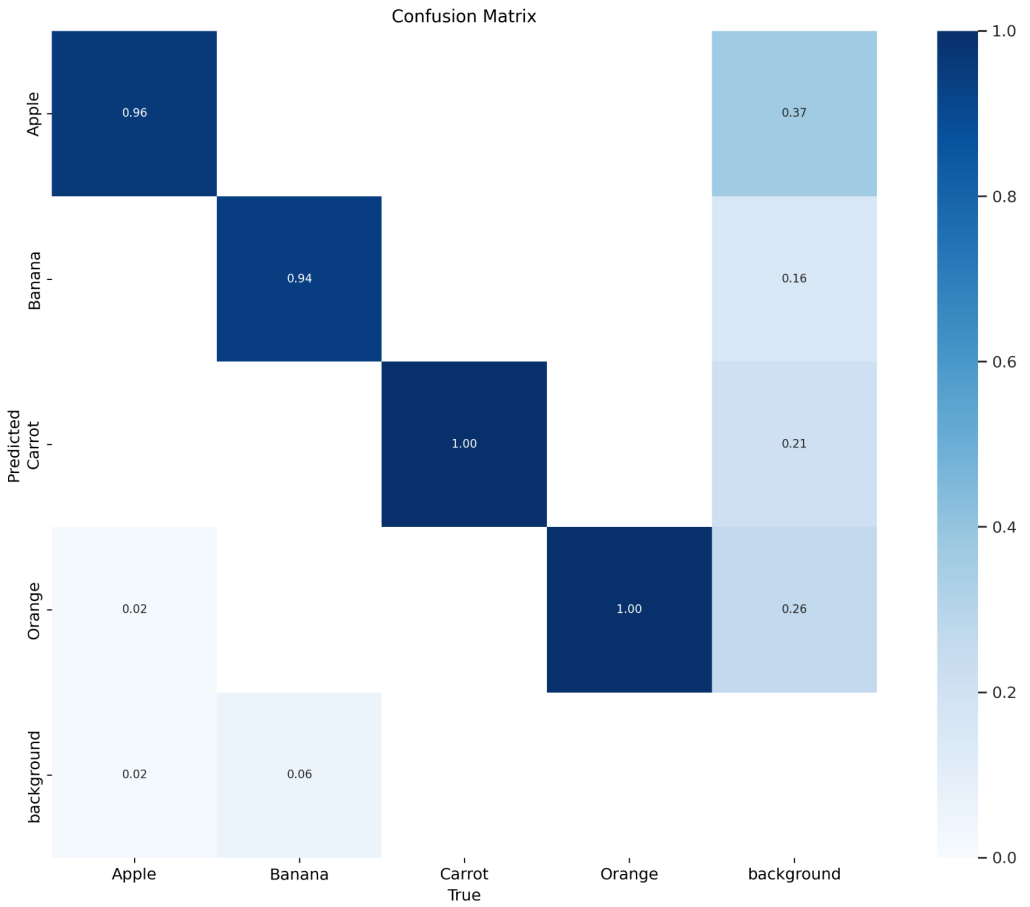

Figure 5.6: Confusion matrix for our second model

Examining the confusion matrix, we can observe a decrease in background misclassification from 0.20 to 0.06. However, this value is negligible as the cameras will only detect the conveyor as a background. In Figure 5.7, we demonstrate how we resolved the issue illustrated in Figure 5.3.

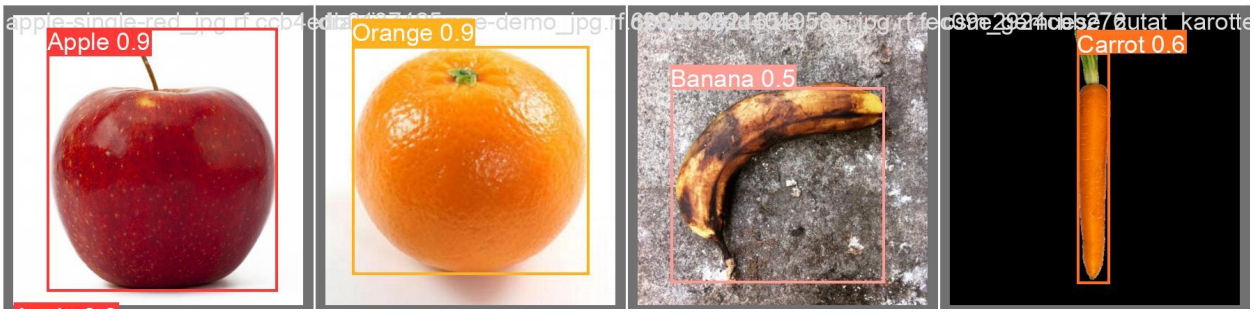

Figure 5.7: Resolve our first model problem

5.4.2 Rotten Apple Detection Model

We worked with a diverse dataset that consisted of 978 images from two categories: 520 fresh apple images and 551 rotten apple images. During the training phase, we used 374 images for fresh apples and 371 for rotten apples.

5.4.2.1 Training Rotten Apple Detection Model

Our training process involved training the model from scratch using YOLOv5 as a pre-trained model. Due to hardware constraints on Colab, we opted for a batch size of 16, where only one GPU was allowed. For image size, we used 416x416 as the minimum to avoid slowing down the training process. The training ran for 40 epochs and took 5 hours, 4 minutes, and 44 seconds to complete. Figure 5.8 reveals that the best model achieved values of 0.90, 0.93, and 0.94 for precision, recall, and mAP0.5, respectively.

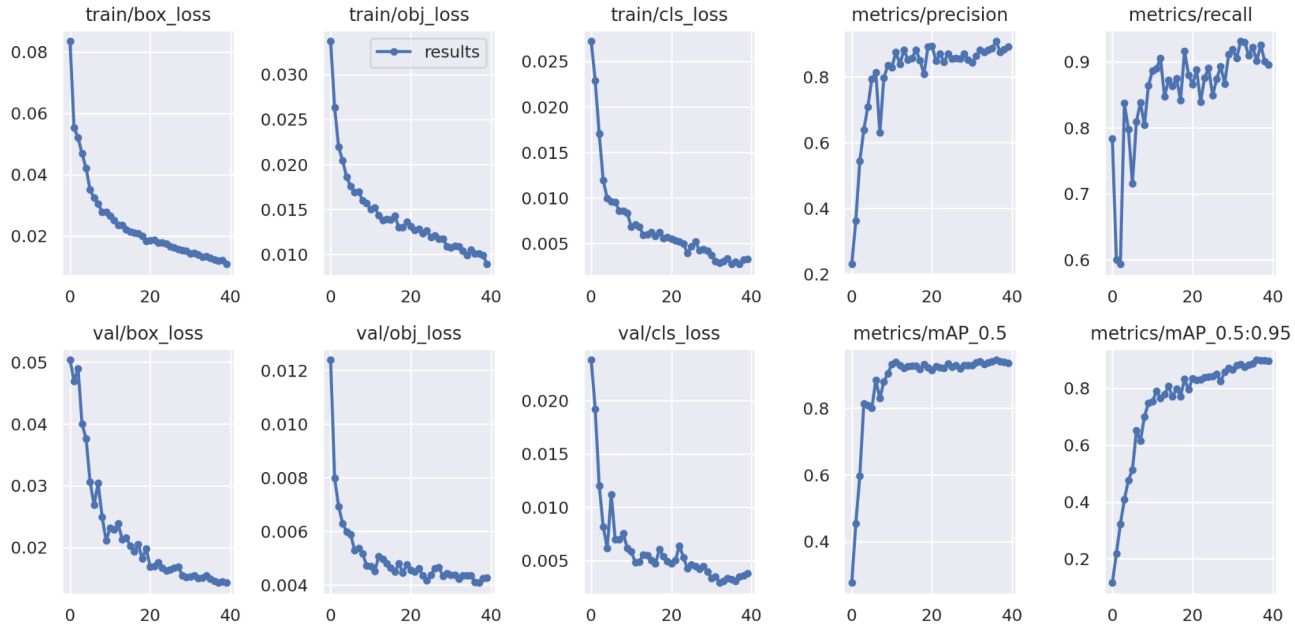

Figure 5.8: Matrics for rotten apple model

5.4.2.2 Experimental Analysis of The Rotten Apple Detection Model

Precision and recall are two important metrics used to evaluate the performance of a machine-learning model. The precision measures the accuracy of the positive predictions made by the model, while the recall measures the percentage of true positives that were correctly identified by the model.

In the case of the previous matrices, the precision was found to be 90%, which may not be considered great in some contexts, but it's not entirely bad either. On the other hand, the recall was 93%, which is an impressive result, indicating that the model can correctly identify a high percentage of true positives.

To further test the performance of the model, we decided to use our webcam and see how it would perform in a real-world scenario. Our observations confirmed that the model was indeed performing well, as we were able to accurately identify a high number of true positives.

We have included the corresponding confusion matrix in Figure 5.9, which provides a visual representation of the performance of the model. The confusion matrix displays the number of true positives, true negatives, false positives, and false negatives, enabling us to evaluate the model's accuracy, precision, and recall. Based on our results, we can confidently say that the model is performing well, and we can continue to rely on it for our future predictions.

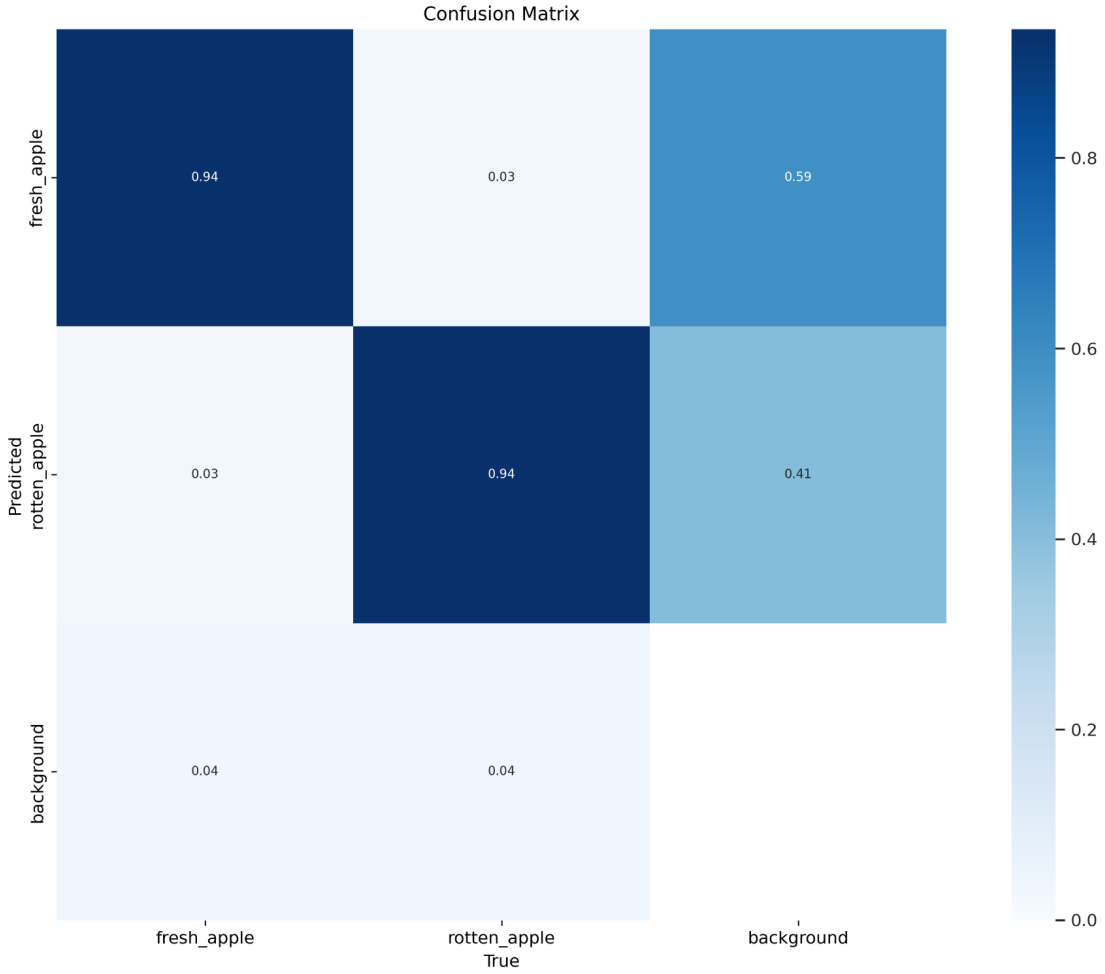

Figure 5.9: Confusion matrix for rotten apple model

The image presented in Figure 5.10 provides a snapshot from the testing phase:

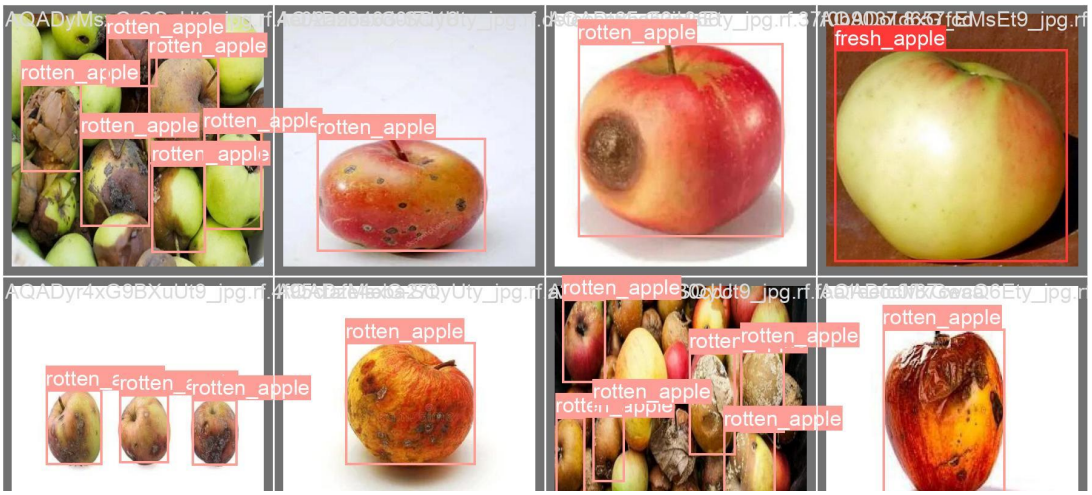

Figure 5.10: Rotten apple model - Snapshot from the testing images

5.4.3 Rotten Orange Detection Model

Our dataset comprised 353 images, which are divided into two categories: fresh oranges and rotten oranges. Out of these, there were 161 fresh orange images and 187 rotten orange images. To train our model, we utilized 117 images of fresh oranges and 124 images of rotten oranges.

5.4.3.1 Training Rotten Orange Detection Model

To train our model, we began by using YOLOv5l (YOLOv5 large) as a pre-trained model and trained it from scratch. Because of the limitations of Colab's hardware, we had to employ a batch size of 16. To avoid slowing down the training process, we used an image size of 416x416 as the minimum. The training process lasted for 100 epochs and took 5 hours, 13 minutes, and 32 seconds to complete. The results of the training are presented in Figure 5.11, it provided valuable insights, leading to the determination of the optimal values for precision, recall, and mAP0.5 as 0.95, 0.96, and 0.97, respectively.

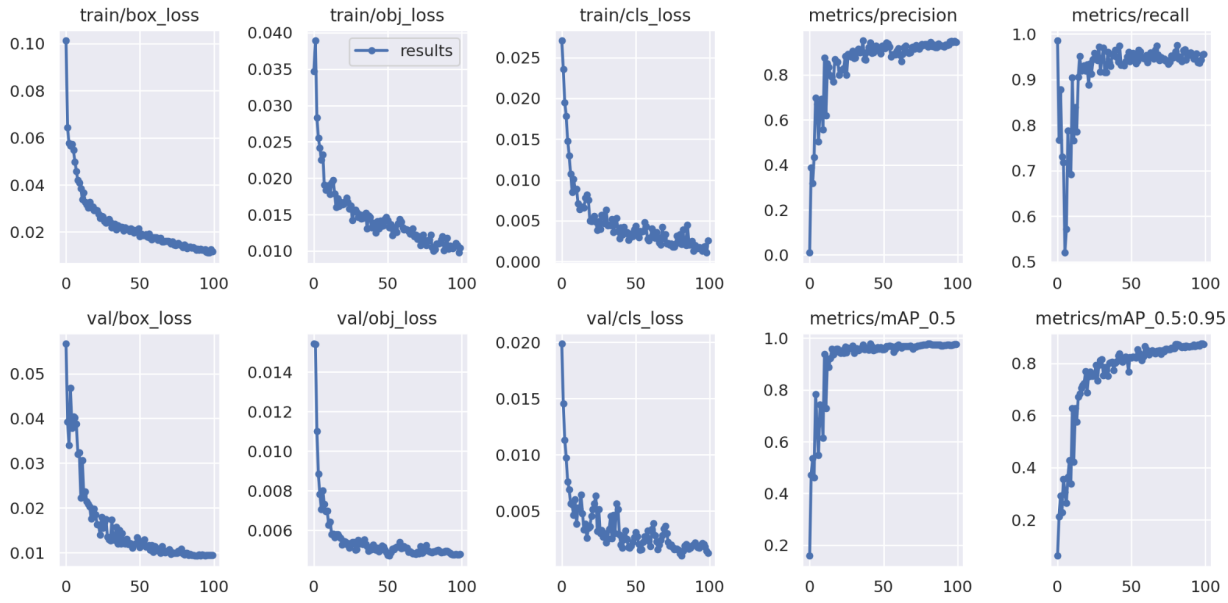

Figure 5.11: Matrics for the rotten orange model

5.4.3.2 Experimental Analysis for The Rotten Orange Detection Model

The precision score of the previous matrices was 95%, which is quite high. We further tested the model and obtained the expected results, demonstrating its effectiveness. The corresponding

confusion matrix is in Figure 5.12. highlights the model's ability to accurately identify true positives and negatives.

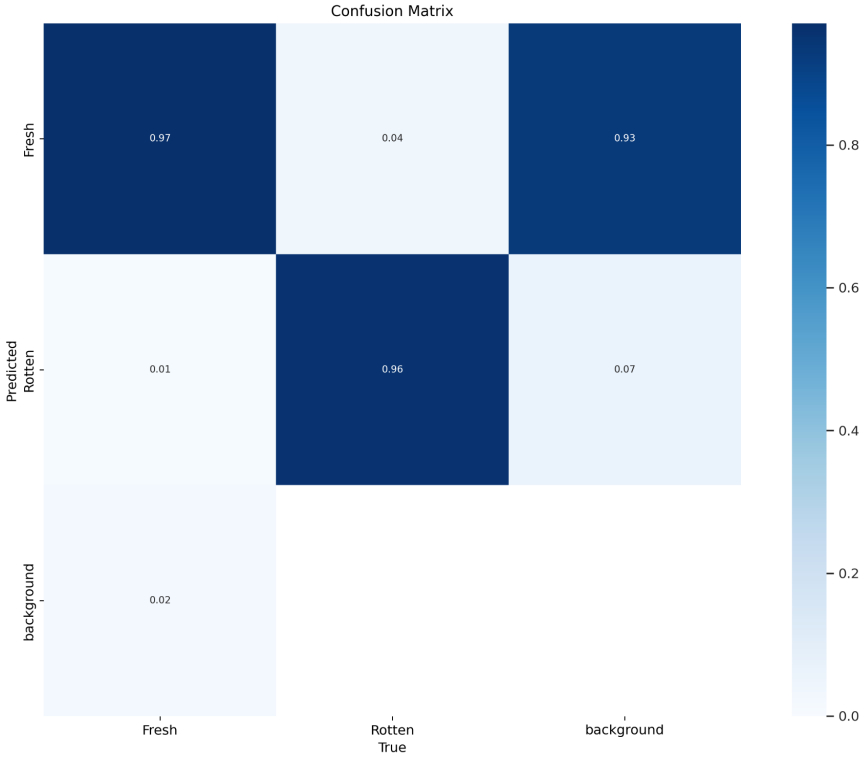

Figure 5.12: Confusion matrix for the rotten orange model

The image presented in Figure 5.13 provides a snapshot from the testing phase:

Figure 5.13: Rotten orange model - Snapshot from the testing images

5.5 Testing the System

When we initially tested our models with the system components, we noticed that the number of frames was lower than expected, only 10 frames per second. We investigated and discovered that each image was being processed by three models, even though only two models (fruit detection and rotten detection for apples or oranges) were needed in the worst-case scenario.

To address this issue, we optimized the process by passing each frame through the fruit detection model first. If the detected fruit was either an apple or an orange, we then passed the frame to the rotten detection model to determine its condition. This approach significantly improved the processing speed, increasing the number of frames per second to 20+.

Chapter 6

Summary and Future Work

6.1 Summary

In this project, we aimed to overcome the limitations of manual sorting in manufacturing, such as high labor costs and sorting errors. We evaluated various options for hardware components and selected the most suitable for our system. Additionally, we explored different software approaches and chose the fastest one to reduce complexity. Our system comprises one stepper motor for the rotational cylinder, two webcams, one ultrasonic sensor, and a conveyor belt system, which are all connected to a standalone processor and an Arduino Mega 2560. By using the YOLOv5 object detection model, we underwent multiple training phases until we obtained a model that met our requirements. Our system can classify objects according to their class and guide them to their intended slot with highly accurate detection rates, eliminating the need for human intervention.

6.2 Future Work

Our project has future work planned to optimize the categorization process in manufacturing. To achieve this goal, we plan to explore additional enhancements, such as expanding the system's capabilities to handle multiple objects, fruits, or products simultaneously. This will significantly speed up the categorization process. We also aim to improve the system's speed by upgrading hardware, optimizing software, and implementing machine learning algorithms.

Furthermore, we plan to introduce software updates that will enable the system to classify the same fruit into categories based on its condition and specific criteria. This feature is essential in the manufacturing and selling process. To reduce the error rate, we will add more cameras from different angles to cover a broader surface area of the fruits on the conveyor.

All these improvements are critical for ensuring the system's efficiency and scalability, leading to increased productivity and reduced costs in manufacturing.

References

[1] “Deep Learning”, investopedia.com/, 2020. [Online],Retrieved from: https://www.investopedia.com/terms/d/deep-learning.asp

[2] “What Is Deep Learning?”, mathworks.com/, 2020. [Online],Retrieved from: https://www.mathworks.com/discovery/deep-learning.html

[3]“Why Deep Learning over Traditional Machine Learning?”,towardsdatascience.com/, 2018. [Online],Retrieved from: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063

[4] “Computer Vision”, sas.com/, 2020. [Online],Retrieved from: https://www.sas.com/en_us/insights/analytics/computer-vision.html

[5] “Feature Engineering for Images: A Valuable Introduction to the HOG Feature Descriptor”, analyticsvidhya.com/, 2019. [Online],Retrieved from: https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introductionhog-feature-descriptor/

[6] E. Ideas, H. Qabajah, K. Abdeen, and L. Tamimi, "Object Sorting System Using Robotic Arm," 2018.

[7] A. AL-Hraine, D. Msulam, and S. Khader, "Potato Sorting Machine Based on LabVIEW with Image Processing strategy," 2019.

[8]S. Abu Zeneh, S. Zaro, S. Alsharif, and A. Halawani, "Object Categorization for Industrial Applications," 2022.

[9] A. Aqeel, “Introduction to arduino mega 2560,” *The Engineering Projects*, 21-Jun-2021. [Online]. Available: https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-mega-2560.html.

[10] “720p HD USB webcam with built-in microphone,” *Adesso Inc*, 15-Nov-2022. [Online]. Available: https://www.adesso.com/product/720p-hd-usb-webcam-with-built-in-microphone-taa-compliant/.

[11] “SJ4000 SERIES”, sjcam.com/, 2020. [Online], Retrieved from: https://sjcam.com/product/sj4000/

[12] “How do conveyor systems work,” *Dezhou Qunfeng Machinery Manufacturing Group*, 2017. [Online]. Available: https://www.peaks-eco.com/news/how-do-conveyor-systems-work-394.html.

[13] “Stepper Motor,” *Wikipedia*, 06-Oct-2022. [Online]. Available: https://en.wikipedia.org/wiki/Stepper_motor.

[14] “Raspberry Pi”, researchgate.net/, 2020. [Online], Retrieved from: https://www.conrad.com/p/raspberry-pi-4-b-1-gb-4-x-15-ghz-raspberry-pi-2138864

[15] L. Rover, "Raspberry Pi or Arduino: When to Choose Which?", LeoRover Tech, 2021. [Online]. Available: https://www.leorover.tech/post/raspberry-pi-or-arduino-when-to-choose-which.

[16] Raspberry Pi Foundation, "Raspberry Pi 4 Model B," in IEEE Xplore, 2022. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ [Accessed: May 21, 2023].

[17] Raspberry Pi Foundation, "Raspberry Pi Camera Module V2," in IEEE Xplore, 2022. [Online]. Available: https://www.raspberrypi.org/products/camera-module-v2/ [Accessed: May 21, 2023].

[18] B. Rosenblatt, "Raspberry Pi vs. Nvidia Jetson Nano: Which is better?," VentureBeat, Mar. 4, 2021. [Online]. Available: https://venturebeat.com/2021/03/04/raspberry-pi-vs-nvidia-jetson-nano-which-is-better/ [Accessed: May 21, 2023].

[19] Nvidia, "GeForce GTX 1050 Specifications," [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/gtx-1050/specifications/ [Accessed: May 21, 2023].

[20] “What is Python”,python.org/, 2020. [Online], Retrieved from: https://www.python.org/doc/essays/blurb/.

[21] “pytorch”,https://github.com//, 2021. [Online],Retrieved from: https://github.com/pytorch/pytorch#nvidia-jetson-platforms

[22] Du, J. “Understanding of Object Detection Based on CNN Family and YOLO”, Retrieved from New Research and Development Center of Hisense, Qingdao 266071, China,2018, pp. The 1-9, Accessed at https://iopscience.iop.org/article/10.1088/1742-6596/1004/1/012029/pdf

[23] “R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms”, https://towardsdatascience.com//, 2018. [Online], Retrieved from: 44 https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[24] “YOLOv5 New Version - Improvements And Evaluation”, 2020 . [Online],Retrieved from: https://blog.roboflow.com/yolov5-improvements-and-evaluation/

[25] Xu, R. , Lin, H. , Lu, K. , Cao, L. & Liu, Y.. (2021). A Forest Fire Detection System Based on Ensemble Learning. Forests. 12. 217. 10.3390/f12020217.

[26] “YOLOv5 compared to Faster RCNN. Who wins?”, 2020. [Online], Retrieved from https://towardsdatascience.com/yolov5-compared-to-faster-rcnn-who-wins-a771cd6c9fb4

[27] “Mean Average Precision (mAP)”, 2020 . [Online],Retrieved from: https://www.kdnuggets.com/2021/03/evaluating-object-detection-models-using-mean-average-precision.html#:~:text=To%20evaluate%20object%20detection%20models,model%20is%20in%20its%20detections

[28] Shah Deval. Mean Average Precision. URL:https://www.v7labs.com/blog/mean-average-precision. (accessed: 4.10.2023).f

[29]Arie Lihi Gur. The practical guide for Object Detection with YOLOv5 algorithm. URL:https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorhm-74c04aac4843. (accessed:22.12.2022).