FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Fernando Kendy Marciniak Arake

End-to-end development of a demonstrator for defect detection

Aachen
2024

Fernando Kendy Marciniak Arake

End-to-end development of a demonstrator for defect detection

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.
Supervisor: Prof. Eric Aislan Antonelo, Dr.
Co-supervisor: Henrik Heymann, M.Sc.

Aachen
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Marciniak Arake, Fernando Kendy
End-to-end development of a demonstrator for defect detection / Fernando Kendy Marciniak Arake ; orientador, Eric Aislan Antonelo, coorientador, Henrik Heymann, 2024.
70 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2024.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Internet of Things. 3. Computer Vision. 4. Defect Detection. I. Aislan Antonelo, Eric . II. Heymann, Henrik . III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. IV. Título.

Fernando Kendy Marciniak Arake

End-to-end development of a demonstrator for defect detection

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, February 26th, 2024.

Prof. Marcelo De Lellis Costa de Oliveira, Dr.
Course Coordinator

Examining Board:

Prof. Eric Aislan Antonelo, Dr.
Advisor
UFSC/CTC/DAS

Documento assinado digitalmente
Eric Aislan Antonelo
Data: 24/05/2024 14:36:12-0300
CPF: ***.163.409-**
Verifique as assinaturas em https://v.ufsc.br

Henrik Heymann, M.Sc.
Supervisor
Fraunhofer IPT

Henrik Heymann
Digital unterschrieben von Henrik Heymann
Datum: 2024.06.10 10:55:45 +02'00'

Prof. Gabriel Thaler, M.Sc.
Evaluator
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Board President
UFSC/CTC/DAS

ACKNOWLEDGEMENTS

I am grateful for my family and friends. Obrigado, pai e mãe.

DISCLAIMER

Aachen, February 15th, 2024.

As representative of the Fraunhofer Institute for Production Technology in which the present work was carried out, I declare this document to be exempt from any confidential or sensitive content regarding intellectual property, that may keep it from being published by the Federal University of Santa Catarina (UFSC) to the general public, including its online availability in the Institutional Repository of the University Library (BU). Furthermore, I attest knowledge of the obligation by the author, as a student of UFSC, to deposit this document in the said Institutional Repository, for being it a Final Program Dissertation (*"Trabalho de Conclusão de Curso"*), in accordance with the *Resolução Normativa n° 126/2019/CUn*.

Henrik Heymann

Digital unterschrieben von Henrik Heymann
Datum: 2024.06.10 10:54:59 +02'00'

Henrik Heymann
Fraunhofer Institute of Production Technology

ABSTRACT

With the Industry 4.0 advent, companies started the digital transformation of their assets and processes. The Internet of Things (IoT) plays a big part in this transformation, connecting devices and machines, leading to better production control, and providing valuable procedure data. Computer Vision (CV) is another area that is taking advantage of industrial digitization. Making good use of cameras and the images they acquire together with CV techniques based on Machine Learning (ML) models, it is possible to setup a CV based system for defect detection, ensuring production quality. The main motivation surrounding the project was the planning and development of a pilot system which can be further reused by different use cases, either as a whole or as a first reference. The focus of this work was the End-to-End development of a demonstrator for defect detection. It started with the hardware selection, hardware setup (physically and connection to a specific cloud), establishment of a socket-based server, setup of a backend service in an existing web application, selection and training of a CV model, and the deployment making use of containers. A simple mockup was used for the frontend, as it was not a goal of the project. The selected use case was based on the MVTec Anomaly Detection Dataset, specifically for defect detection in cables. The goal was achieved and the whole system worked well, from data acquisition to the final deployment.

Keywords: Defect detection. IoT. Computer vision.

RESUMO

Com o advento da Indústria 4.0, empresas iniciaram a transformação digital dos seus ativos e processos. A Internet das Coisas (IoT) desempenha um papel importante nesta transformação, conectando dispositivos e máquinas, levando a um melhor controle da produção e fornecendo dados valiosos. A Visão Computacional (CV) é outra área que está aproveitando a digitalização industrial. Fazendo bom uso de câmeras e das imagens por elas adquiridas junto a técnicas de CV baseadas em modelos Aprendizado de Máquina (ML), é possível montar um sistema baseado em CV para detecção de defeitos, garantindo qualidade na produção. A principal motivação a cerca desse projeto era o planejamento e desenvolvimento de um sistema piloto que possa ser reutilizado futuramente em diferentes casos de uso, tanto o sistema todo quanto como uma referência inicial. O foco desse projeto é o desenvolvimento completo de um demonstrador para detecção de defeitos. Partindo da seleção de *hardware*, a montagem do *hardware* (fisicamente e conexão a uma *cloud* específica), definição de um servidor baseado em *sockets*, desenvolvimento de um serviço de *backend* em um aplicação *web* existente, seleção e treinamento de um modelo de CV, e a implantação do sistema fazendo uso de containers. Um modelo simples de *frontend* foi utilizado visto que não era um objetivo do projeto. O caso de uso selecionado foi baseado no *MVTec Anomaly Detection Dataset*, especificamente para a detecção de defeitos em cabos. O objetivo foi alcançado e o sistema como um todo funcionou bem, desde a aquisição de dados até o *deployment* final. **Palavras-chave**: Detecção de defeitos. Internet das Coisas. Visão computacional.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

1.1 CONTEXTUALIZATION

The industrial scenario is getting more and more influenced by the Industry 4.0 (I4.0) and its topics nowadays (deeper explanation on chapter 4). The main aspect of the I4.0 is the digitization of its processes and services, leading to better control over the many different processes in a factory. To achieve the desired level of control, information about the procedures is needed, and that is where the Internet of Things (IoT) comes in scene. IoT defines a network of devices, systems, sensors, and softwares that connects and exchange data with each other. Having IoT devices throughout the production chain allows for a detailed overview of the process, helping in getting insights and finding new solutions and optimizations. Companies are making use of real time data analysis to make decisions, ensuring that no immediate issue may influence on the final product. Many are the possibilities of improvement in the production chain, being the production quality the focus of this project. Production quality assures that the company is effectively fabricating products that are according to their standards, leading to a increase in clients satisfaction and fidelity. To assure production quality, many different techniques can be used, like Computer Vision (CV), which is the case of this project. By using a well established CV model widely used throughout both the industry and academy, YOLOv8, it was possible to achieve solid defect detection in the planned system.

The project focuses on developing a pilot system for defect detection by planning its architecture and developing the prototype. It aims at providing this first system that can be reused or used as an initial reference for further projects, while also being an easy to carry system that can be taken to fairs and events, allowing the institute (see next section) to showcase what it does and gathering interest from both the general public and possible partners. The main requisites are to have it as automated as possible - seeking for a "plug and play" prototype, to make it portable, to work in a continuous error less manner, and to be scalable.

The system is composed by a camera connected to a device which should communicate with a service at the Intelligent Quality Platform IQP, a website of Fraunhofer IPT. The service will make the defect detection, which will be displayed in the frontend of the application and physically through a light column.

1.2 THE RESEARCH INSTITUTE

The project was developed at the Fraunhofer Institute for Production Technology. Further explanations of the company and its history will be shown in the next subsections.

1.2.1 The Fraunhofer Society

The *Fraunhofer-Gesellschaft* is a German-based organization named after Joseph von Fraunhofer (1787 – 1826), a German researcher who made significant contributions to modern

optics, discovering the Fraunhofer lines. The institution was founded in Munich on March 26^{th}, 1949. Since it was the post-war period, there was an urgent need to strengthen the industry, mainly in iron and steel and mechanical engineering. Later on, in 1951, it was the first time that the *Fraunhofer-Gesellschaft* received funds from the Marshall Plan through the European Recovery Program. That was the beginning of the organization, which opened its first institute in 1954. By 1956, other institutes were opened, leaving the region of Bavaria and expanding across Germany.

The *Fraunhofer-Gesellschaft* is the leading applied research organization in the world, with 76 institutes and research units throughout Germany. The funding usually comes from three main sources: industry contracts, publicly funded projects, and federal and state government.

1.2.2 The Fraunhofer Institute for Production Technology - IPT

Many institutes are related to the *Fraunhofer-Gesellschaft*, and the Fraunhofer Institute for Production Technology IPT is one of them. Having been opened in 1980, it is located in Aachen, Nordrhein-Westfalen, Germany, and is focused on production matters, especially in manufacturing.

Figure 1 – Fraunhofer Institute for Production Technology.

Source: (Fraunhofer IPT, 2024).

1.2.3 The Production Quality Department

The production quality is the department in which the project was developed. Among its main focuses, are the consistent digitalization and cross-linking of production data and maximal resource efficiency, figures in high positions.

2 OBJECTIVES

The main objective of the project is the end-to-end development of a demonstrator for defect detection, starting with the architecture setup to the development and implementation.

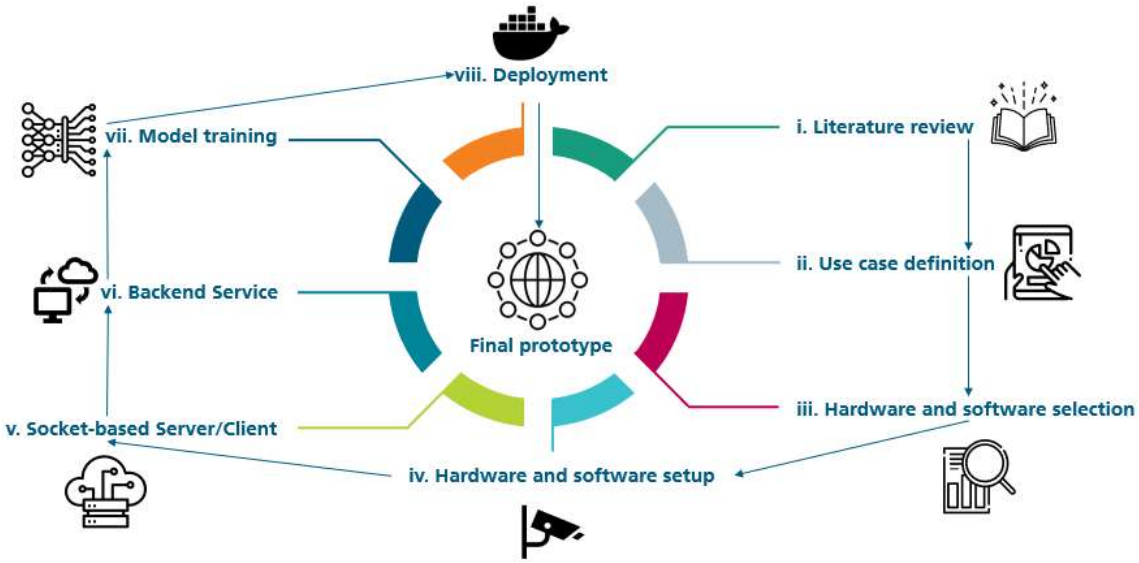
To reach the main objective, some specific objectives had to be achieved, being them:

- A solid performance of the CV defect detection model to prove its usability and its effectiveness.
- Flawless interaction of all the system components: Raspberry Pi, socket-server, backend, and frontend.
- Stablish a solid pilot system, opening the path for future improvements without changing much of the original.

3 METHODOLOGY

All of the steps described here are developed in Section 4 and 5. The idea of the project was the one described in chapter 2, but how to achieve that? Figure 2 shows the flowchart of the steps that were taken, leading to the final prototype.

Figure 2 – Methodology's flowchart.



Source: author.

1. Literature review: for a better understanding of the methods and techniques that were going to be used, it was needed to research and read about them, leading to a more solid theoretical knowledge.

2. Use case definition: knowing that it was going to be about defect detection, a use case had to be defined. The ideal use case would be industry related, being easy to use and handle. A public dataset by MVTec was chosen, containing defects in cables.

3. Hardware and software selection: based on real-time images, the full hardware setup had to be decided. The controlling device (i.e. Arduino, ESP32, Raspberry Pi) had to be easy to use, with solid documentation, while also attending some requirements, such as the automation of certain processes. A Raspberry Pi, a USB camera, and a traffic light column were selected. For the software, Python is the main programming language, and all the used systems had Linux distributions on them.

4. Hardware and software setup: after selecting the hardware, it was all assembled. The goal was to have flawless interaction between the hardware units that were previously selected, mainly worrying about the activation of the 24V light column and its control. Software

wise, the goal was to achieve the automated execution and start of the tasks the Raspberry would perform.

5. Socket-based server/client: for the communication between the Raspberry Pi and the Virtual Machine (VM), a socket-based server was used. The goal was to have a standard architecture to handle different connections and its inputs/outputs (requests, commands, returns).

6. Backend service: since the objective was to deploy this service on an existing web application of IPT, a FastAPI backend was defined. The backend would be responsible for requesting new images and, after receiving them, using a defect detection model on it, returning the image with the prediction on top of it.

7. Model training: using the YOLOv8 model and the chosen dataset, the model was trained. By using only the images available in the data set, the model did not perform that well, being biased and over fitted. Data augmentation techniques were used in order to improve the model training, leading to better performance.

8. Deployment: at the end, the complete system was deployed at the FEC making use of Docker containers.

4 THEORETICAL BACKGROUND

4.1 INDUSTRY 4.0

In the 18th century, what is called the First Industrial Revolution happened. Its most remarkable change in the way things were produced was the introduction of steam engines, which led to a rise in human productivity (SINGH; SHARMA, 2020). The so-called Second Industrial Revolution, which happened in the 19th century, was responsible for popularizing the usage of electricity and steel in the industry (AGARWAL, H.; AGARWAL, R., 2017). These two were key factors in the increase and widespread of mass production models, like Fordism. A little further, by the middle of the 20th century, the Third Industrial Revolution took place, with the most notable change being the technological revolution. It started with the movement from mechanical and analog electronic technology to digital electronics, to the massive use of microelectronic devices (computers, microprocessors, semiconductors), and the invention of the Internet by the late 60s (MOHAJAN, 2021). It is still happening and is expected to endure until the 2030s.

While being part of the 3rd Industrial Revolution, society and technology had great advances. This advance led to what is called the Fourth Industrial Revolution, or the Industry 4.0. This new revolution is built around the concept of smart factories, with the integration between men and technology through Cyber-Physical Systems (CPS) (PETRILLO et al., 2018). As defined by Zanero (2017) "In many real-world systems, computational and physical resources are strictly interconnected: embedded computers and communication networks govern physical actuators that operate in the outside world and receive inputs from sensors, creating a smart control loop capable of adaptation, autonomy, and improved efficiency". According to Petrillo et al. (2018), Industry 4.0 covers three fundamental aspects:

1. Digitization and increased integration of vertical and horizontal value chains;
2. Digitization of product and service offerings;
3. Introduction of innovative digital business models.

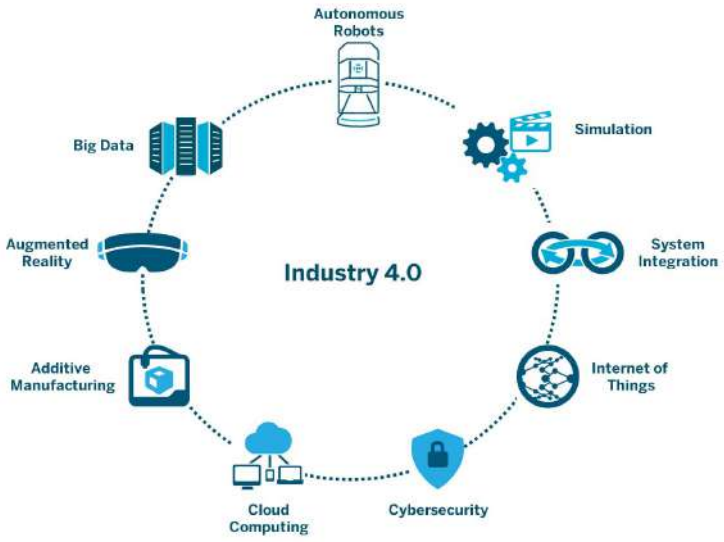
All of the aforementioned topics add up to the idea of a smart factory: digitization of each process of an industry so more control and improvements can be done over each of them.

With all the digitization happening, another concept was defined: the Internet of Things (IoT) - Figure 3. The following subsection will give an overview of IoT.

4.2 IOT

The Internet of Things can be defined as a technology that is embodied in a wide spectrum of networked products, systems, and sensors, which take advantage of advancements in

Figure 3 – Industry 4.0 and its main areas.



Source: (Aethon, 2018).

computing power, electronics miniaturization, and network interconnections to offer new capabilities not previously possible (ROSE; ELDRIDGE; CHAPIN, 2015).

A typical IoT system works through real-time data collection and exchange. It also stated that, commonly, IoT systems have the following components (AWS, 2024c):

1. Smart devices: devices that are used to collect data, user inputs, or user patterns and send and receive data from the IoT application. In the case of the project, it was a camera for image acquisition;

2. IoT application: An IoT application is a collection of services and software that integrates data received from various IoT devices. It uses machine learning or artificial intelligence (AI) technology to analyze this data and make informed decisions. These decisions are communicated back to the IoT device and the IoT device then responds intelligently to inputs (AWS, 2024c). For this project, two main services compose the IoT application: a socket server and a backend service;

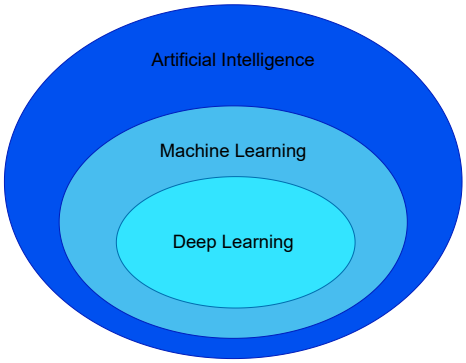
3. Graphical User Interface (GUI): management through a GUI. A webpage was used as the GUI for the project.

All three common components add up to a complete IoT system, which can be used for different purposes: smart houses, production quality, and employee safety. In this case, it was used for assurance of production quality, in specific, defect detection.

4.3 ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) can be explained, in a shallow manner, as "something" capable of replicating human capacities, such as thinking, planning, selecting, and defining. "On an operational level for business use, AI is a set of technologies that are based primarily on machine learning and deep learning, used for data analytics, predictions and forecasting, object categorization, natural language processing, recommendations, intelligent data retrieval, and more" (GOOGLE, 2024b). Figure 4 shows how AI and its subgroups (ML, DL) are related.

Figure 4 – AI and its subgroups.



Source: author.

4.3.1 Machine Learning

It would be easier to explain to a child what is the difference between a racing car and a common car through examples, rather than trying to find rules or conditions that define a racing car (JANIESCH; ZSCHECH; HEINRICH, 2021). This phrase works extremely well to define the idea behind Machine Learning (ML). The computer, as a ML system, is fed by data so it can by itself assess characteristics and patterns of what is being watched. Jordan and Mitchell (2015) defined Machine Learning as an improvement in a performance metric as a task is performed, through some sort of training experience.

Depending on what is being assessed, it is expected that the machine is able to take actions that match the desired behavior, whether it is a movie recommendation, autonomous driving, or, in this case, defect detection. Based on (MAHESH, 2020), ML can be defined by the following subgroups:

- Supervised Learning: a ML task that aims to learn a function that maps one input and one output based on an input-output reference. Labeled data is provided, for example, animal images. After training the model, it is expected that a picture of a dog will be classified as such.

- Unsupervised Learning: making opposition to Supervised Learning, there are no correct answers. Non-labeled data is provided to the model, being the model responsible for extracting characteristics and patterns from it. Going back to the animals example, the model might classify the animals not based on their species, but on their characteristics from, like animals with and without wings (birds, in general).

- Reinforcement Learning: Reinforcement Learning is an area of ML focused on how an agent will take actions in a given environment so a certain reward can be maximized. Good examples of this are old games played by AI. In the case of the popular game Pong, the agent has as a reward the scored points. It considers these before taking action, always looking to maximize the reward.

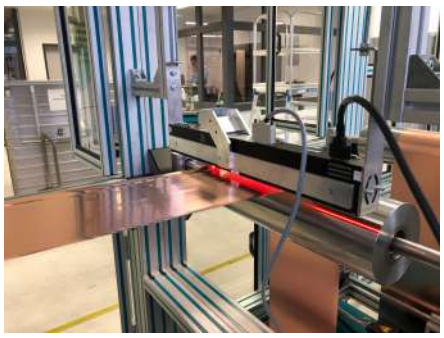
4.3.2 Computer Vision

According to Azure (2024), "Computer vision is a field of computer science that focuses on enabling computers to identify and understand objects and people in images and videos. Like other types of AI, computer vision seeks to perform and automate tasks that replicate human capabilities. In this case, computer vision seeks to replicate both the way humans see and the way humans make sense of what they see."

In modern Computer Vision (CV), Deep Learning has become the most used method to achieve high-quality CV products. "Computer vision uses deep learning to form neural networks that guide systems in their image processing and analysis. Convolutional neural networks (CNN) techniques enable deep learning inference for image classification and object detection. Once fully trained, computer vision models can perform object recognition, detect and recognize people, and even track movement" (INTEL, 2024).

The most common use cases for CV in industrial cases are object detection and recognition, packaging inspection, sorting and counting, and quality inspection, which is the case of this project. Figure 5 shows a setup using a Contact Image Sensor (CIS) in a manufacturing environment.

Figure 5 – Use of a CIS for quality inspection in an ion-lithium battery production.



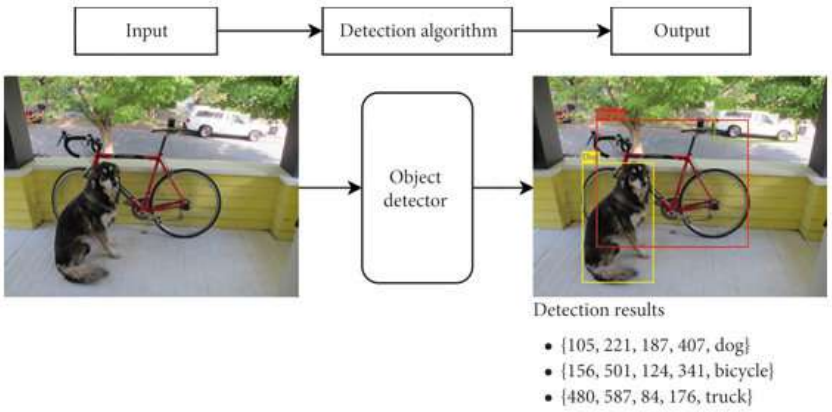
Source: (Alexander Kreppein, 2023).

4.3.2.1 Object detection

The statement by Mathworks (2024) defines clearly what is object detection in CV: "Object detection is a computer vision technique for locating instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results. When humans look at images or videos, we can recognize and locate objects of interest within a matter of moments. The goal of object detection is to replicate this intelligence using a computer".

Object detection algorithms will take an image as input and return both the coordinates of the bounding boxes and the detected object (the format of these outputs may vary from model to model). In the case of Figure 6, an image containing a dog, a bicycle, and a truck is used as input. After being processed by the model, the given output is made of a set of four coordinates that represent the boundary boxes ($x_{\min}, x_{\max}, y_{\min}, y_{\max}$) and a string that names the detected object (HASSAN; KHALIL; AHMAD, 2020).

Figure 6 – Object detection working flow.



Source: (HASSAN; KHALIL; AHMAD, 2020).

For this project, instead of detecting objects (cars, trees, cups), object detection techniques were used for defect detection in cables. The following section will introduce and explain the model that was used.

4.3.2.2 YOLOv8 model

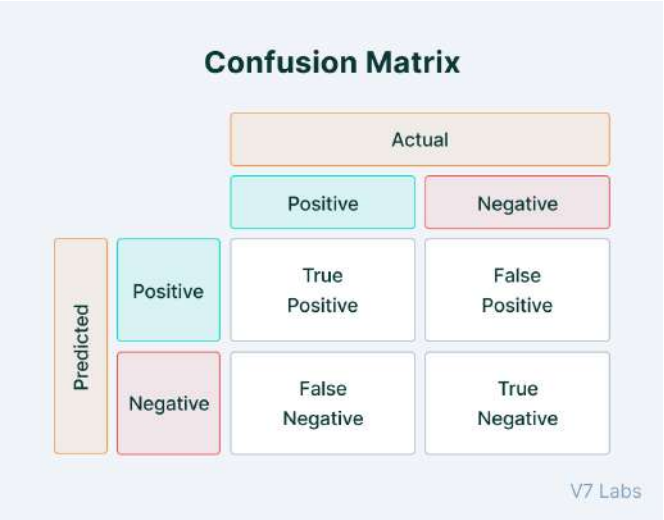
You Only Look Once (YOLO) is an object detection and image segmentation model developed by Joseph Redmon and Ali Farhadi at the University of Washington, being launched in 2015. Many versions of the model have been released in the past years and their main versions, according to the documentation by Ultralytics (2024), are:

- YOLOv3: released in 2018, enhanced the performance of the previous versions by using a more efficient backbone network, multiple anchors, and spatial pyramid pooling.
- YOLOv5: in 2020, the 5th version was released. It significantly improved the performance and added new features, like hyperparameter optimization, integrated experiment tracking, and automatic export to popular export formats.
- YOLOv8: launched in 2023, it is the latest version of the model. It supports many different tasks, including detection, segmentation, pose estimation, tracking, and classification. It is very versatile and allows users to use it for many different applications.

The differences between performance for the latest versions can be seen in Figure 10. The metric is the Mean Average Precision (mAP), commonly used to analyze the performance of object detection models. It is based on four other sub metrics (SHAH, 2022):

1. Confusion Matrix: based on 4 attributes. Represents how many predictions are correct and incorrect per class, helping in understanding the classes that are being confused by the model as other classes (TIWARI, 2022).
    - True Positives (TP): the prediction is correct.
    - True Negatives (TN): no prediction is made, which is expected. In object detection, it is not taken into account. It can be said that it is correctly detecting the background of an object as background, so no detection.
    - False Positives (FP): the prediction is incorrect, no object should be detected.
    - False Negatives (FN): no prediction is made, but it should.

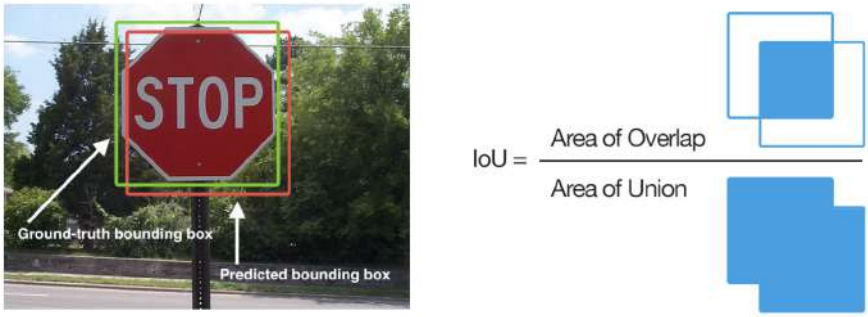
Figure 7 – Confusion Matrix.



Source: (SHAH, 2022).

2. Intersection over Union (IoU): indicates the overlap of the predicted bounding box coordinates to the ground truth boxes. Higher IoU values mean that the prediction closely resembles the ground truth box coordinates, the closer to one the better (Figure 8).

Figure 8 – Intersection over Union.



Source: (SHAH, 2022).

3. Precision: it measures how well you can find true positives (TP) out of all positive predictions (TP + FP). For example, if the precision of the model is 0.753, it means that the prediction should be correct around 75% of the time.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

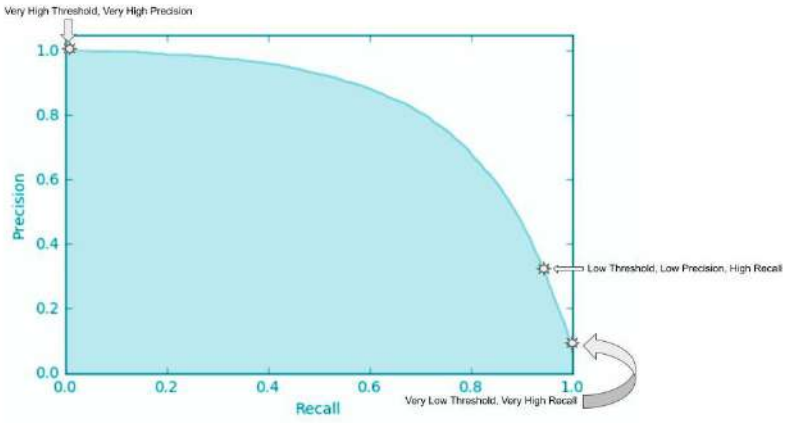
4. Recall: it measures how well you can find true positives (TP) out of all the real positives available (TP + FN). The closer to 1 the better, meaning that the amount of FN is very low.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

Having all these metrics defined, it is possible to plot the precision-recall curve (figure 9). This graph "shows how to recall changes for a given precision and vice versa in a computer vision model" (SOLAWETZ, 2020). The metric of interest for this curve is the Average Precision (AP) which is defined as the weighted mean of precisions at each threshold with the increase in recall from the previous threshold used as the weight. In this case, the threshold is the confidence score assigned to the predicted bounding boxes. So for each different confidence score, a threshold is set.

To start calculating the mAP, it is needed to define an IoU threshold. With the IoU threshold defined, it is possible to define the TP (IoU > IoU threshold), FP (IoU < IoU threshold), and FN (missing prediction). Having them, the Precision and Recall for each threshold (confidence) is calculated. Then, for each class of the model, a precision-recall curve is generated, and from that, the AP is calculated. To finish it off, the mAP is the mean of all the APs that were calculated. Equation (3) defines mAP, being N the number of classes available.

Figure 9 – Precision-recall curve.



Source: (CHUGH, 2023) .

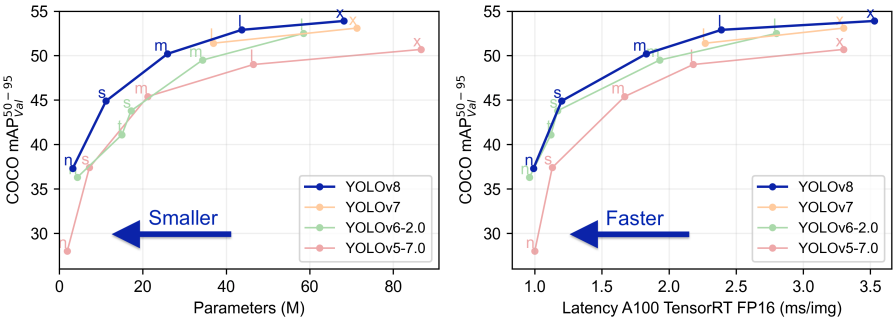
$$\text{mAP} = \frac{1}{N}\sum_{n=1}^{N} \text{AP}_n \tag{3}$$

The advantage of using mAP to measure the performance of the model is that it takes into account all the classes available in a single metric, giving robustness to the analysis. The higher the mAP the better the model is across its classes. Depending on the IoU threshold that was chosen, the terminology changes. So for a fixed IoU of 0.5, the term is mAP@0.5. In the case of a range of IoU thresholds, from 0.10 to 0.90 with a step size of 0.05, the term would be mAP@0.10:0.05:0.90.

With the understanding of what mAP is, it is possible to go back to the YOLO models and check their overall performances comparing each other, as seen in Figure 10. It is notable that the newest version of the model is the best one, requiring fewer parameters to reach better performance. It is also faster than the old ones.

For the model that was used in this project, YOLOv8, the output is as follows: a set of bounding boxes that enclose the detected objects in the format $(x_{\text{center}}, y_{\text{center}}, \text{width}, \text{height})$, the probabilities of the object belonging to each of the possible classes, and an integer that displays the maximum number of possible detected objects. The architecture of the model can be seen in Figure 32. The architecture of the model will not be further commented on nor explained in this document.

Figure 10 – Performance of the latest YOLO models.

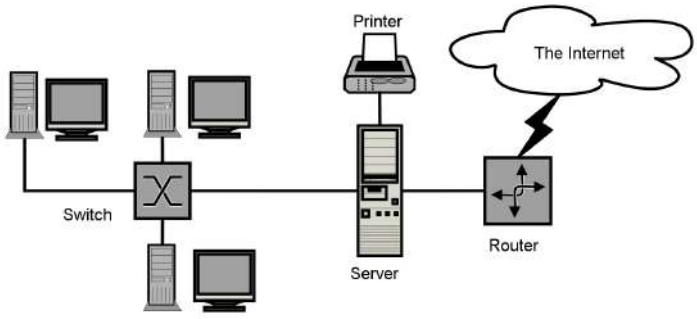


Source: (ULTRALYTICS, 2023).

4.4 COMPUTER NETWORKS

As said by IBM (2024), "A computer network comprises two or more computers that are connected—either by cables (wired) or WiFi (wireless)—to transmit, exchange, or share data and resources. You build a computer network using hardware (e.g., routers, switches, access points, and cables) and software (e.g., operating systems or business applications)".

Figure 11 – Network example.



Source: (JACOBS, 2023).

Some terms and concepts are important to have a clearer understanding of Computer Networks, being them:

1. IP address: it is a 32-bit number. It uniquely identifies a host (e.g., computer, printer, router) on a TCP/IP network (MICROSOFT, 2022).

2. Nodes: A network node can be defined as the connection point among network devices, allowing to send data from one endpoint to the other (SOLARWINDS, 2024).

3. Ports: identifies a specific connection between network devices. Each port is identified by a number. If you think of an IP address as comparable to the address of a hotel, then ports are the suites or room numbers within that hotel (IBM, 2024).

There are many network types, LAN (Local Area Network) and VPN (Virtual Private Network) are the most important ones for this project. LAN defines a computer network in which the devices are relatively close to each other (e.g., school, supermarket, company building). LANs are usually private. VPN, according to (IBM, 2024), is "A VPN is a secure, point-to-point connection between two network endpoints. A VPN establishes an encrypted channel that keeps a user's identity and access credentials, as well as any data transferred, inaccessible to hackers".

4.4.1 Socket programming

IBM (2021b) defines a socket as a communications connection point (endpoint) that you can name and address in a network. Socket programming uses sockets to establish communication between different devices in a network.

Sockets behave as follows (IBM, 2021a):

- A socket will keep existing as long as the process maintains itself connected to it.
- A socket can be named and used to communicate with other sockets (in the same domain).
- Sockets perform the communication when the server accepts a connection from them.

In Figure 12, it is possible to see a client/server setup based on sockets. First, sockets are defined, both for the server and the client. Then, the socket is bound to a specific address and port by the bind() method. Right after, the server keeps on listening until the client asks to establish a connection. After accepting, both the server and client can either send or receive data from each other, being the connection closed whenever one of them finishes it.

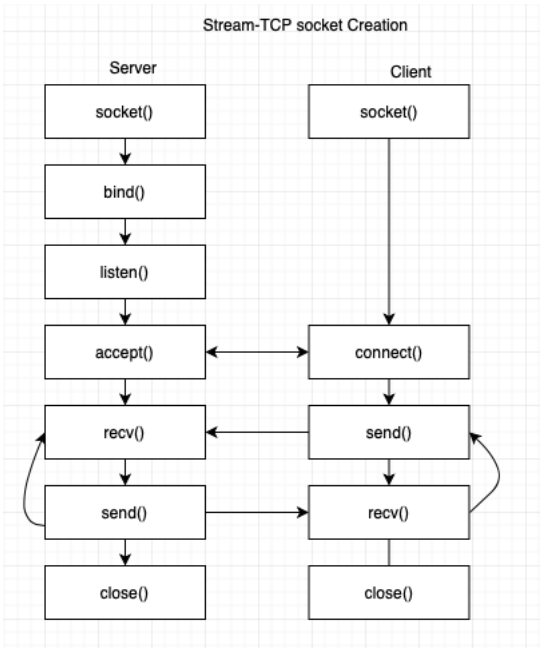
The protocol used for the communication between these devices is TCP/IP, which will be explained in subsection 4.4.2.

4.4.2 Communication Protocols

According to IBM (2023), protocols are sets of rules for message formats and procedures that allow machines and application programs to exchange information. There are many communication protocols, being the main ones for this project:

1. TCP/IP: based on the Transmission Control Protocol (TCP) and the Internet Protocol (IP). TCP is responsible for receiving the data, dividing it into packets, adding a destination address, and passing it ahead. Then, the packet is used to define an IP datagram, which has a header and trailer. The datagram is passed on using, for example, an ethernet cable (IBM, 2023).

Figure 12 – Client/Server setup using sockets.



Source: (PRODEVELOPERTUTORIAL, 2020).

2. HTTP: "The Hypertext Transfer Protocol (HTTP) is the foundation of the World Wide Web, and is used to load webpages using hypertext links. HTTP is an application layer protocol designed to transfer information between networked devices and runs on top of other layers of the network protocol stack. A typical flow over HTTP involves a client machine making a request to a server, which then sends a response message" (CLOUDFLARE, 2024).

4.5 MULTITHREADING

For Multithreading, 2 concepts are important to understand: process and thread. A process is a program that is running in some system. Dividing this program into smaller independent units, results in threads. A collection of threads makes up a process (GOEL, 2023).

Multithreading becomes extremely useful for cases where it is needed to handle more than one task at the same time. In the case of this project, where a Raspberry should keep on listening to a socket while also acquiring images, multithreading is extremely useful. Not only it allow both tasks to be run simultaneously, but it also helps in making better use of the available resources.

4.6 WEB DEVELOPMENT

Dharmen (2024) defined web development as follows: "Web development refers to the creating, building, and maintaining of websites. It includes aspects such as web design, web publishing, web programming, and database management. It is the creation of an application that works over the internet i.e. websites."

Backend

The backend is the "back" part of a website, where users can not interact with nor visualize it. It is responsible for setting up the logics that will be presented to the user through the front end (webpage).

4.7 CONTAINER

Containers are commonly used in software development. "A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another" (DOCKER, 2024b).

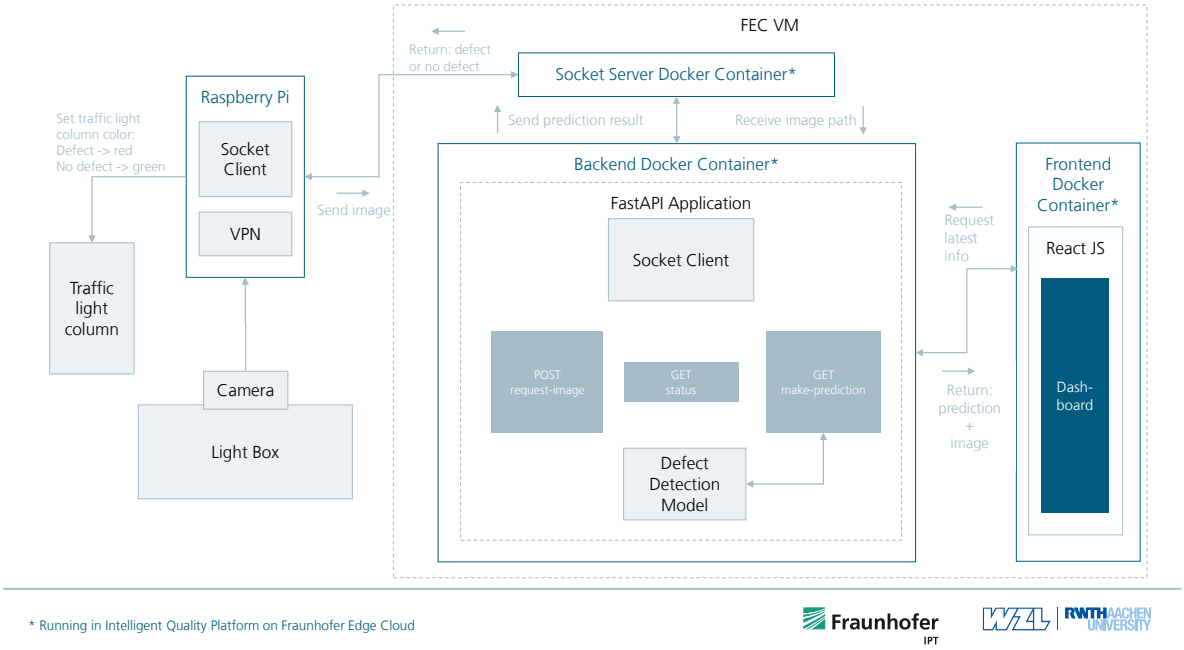
The biggest advantage of using containers is the isolation that they bring, meaning that independently of the device that is running the containerized application, the application will always work the same.

5 DEMONSTRATOR DEVELOPMENT

5.1 TECHNICAL SETUP

Having the methodology defined in Chapter 3, and after reviewing and checking on the needed knowledge in Chapter 4, it was possible to define the final technical setup of the project, as seen in Figure 13. It shows all that is running on the FEC (server, backend, frontend) and what is running outside of it (Raspberry Pi (client, VPN)). All the main interactions between each part are displayed, mostly showing what is being requested and returned.

Figure 13 – Technical setup.



Source: author.

5.2 USE CASE: DEFECT DETECTION IN CABLES

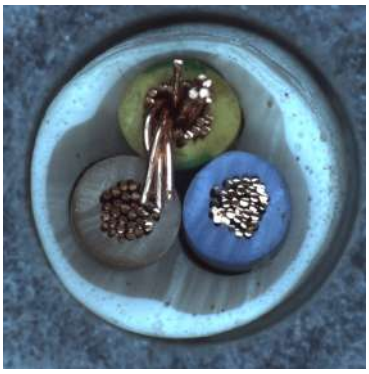
As said previously, the focus of the project is on the end-to-end development of a demonstrator for defect detection. So, it was needed to have some data to detect defects.

The online available MVTec anomaly detection dataset (MVTec AD) was the chosen dataset. As defined by MVTec (2024) "MVTec AD is a dataset for benchmarking anomaly detection methods with a focus on industrial inspection. It contains over 5000 high-resolution images divided into fifteen different object and texture categories". Among all these available images, which cover different industrial processes, defects on cables were selected. This use case was the chosen one due to it being industry related while also having defects that were easier to identify. As the focus of the project was to define the architecture of the system and

how to implement it, the use case should be real-life related, but also easier to handle and work with.

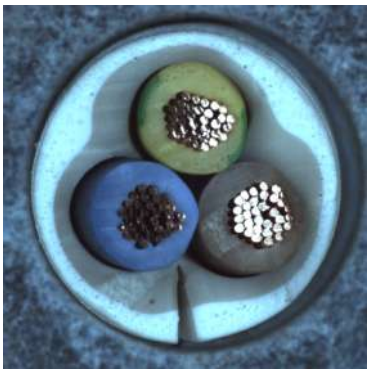
Figure 14 – Images used to train the model.

(a) Good cable.

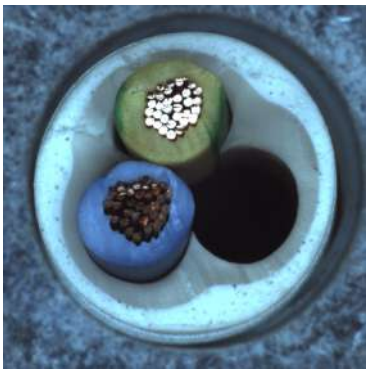


(b) Defected cable: bent wires.

(c) Defected cable: cut inner insulation.



(d) Defected cable: cut outer insulation.



(e) Defected cable: missing cable.

Source: (MVTEC, 2024).

In Figure 14, it is possible to see all the classes that were considered in this project. Figure 14a shows a cable that is in perfect conditions. In Figures 14b, 14c, 14d, and 14e defected cables are shown, along with their defects names.

5.3 HARDWARE AND SOFTWARE SELECTION

Many different tools were required for the development of the project, both software and hardware. This first part will present them.

5.3.1 Hardware

Talking about the hardware, first, the main ones will be all listed, and then each one will get a deeper explanation.

The three main hardware components of the project are:

1. Raspberry Pi 4 B 8 GB;

2. Arducam B029201 4K 8MP;
3. Signal Tower TC-9539296.

Raspberry Pi 4 B

The Raspberry Pi Foundation is the company responsible for the Raspberry Pi in all its versions. They have as their main mission "to enable young people to realize their full potential through the power of computing and digital technologies" (FOUNDATION, 2023). They also want to reach as many people as possible, in three main areas: education, non-formal learning, and research.

With this contextualization about who developed the Raspberry Pi, it is reasonable to take a look at the device now. The idea of the Pi is to be a single-board computer while providing good performance and having an affordable price. The first device, Raspberry Pi 1 B, was released in 2012 with a single-core 700MHz CPU and 256MB RAM. Since 2012, many versions have been developed and made available to the public. This project used the Raspberry Pi 4 B, which is the fourth generation. The exact specifications of it are as follows (further details can be seen here):

- Processor: ARM Cortex-A72;
- Processor cores: 4 x;
- RAM: 8 GB;

The other options, compared to the Raspberry Pi, would be either an Arduino or an ESP32. For the functionality of acquiring images, either one of them would work perfectly. The main reason why the Raspberry was selected over the other options is the automation of the tasks that would be running on the device. Remembering that the final prototype is a demonstrator that would be taken around to fairs and events, having a plug and play device is a must. It is possible to easily run Linux distros in a Raspberry, being possible to use systemd services to automate tasks, such as running the VPN connection service and the image acquisition script.

ArduCam B029201 4K 8MP

Being developed by ArduCam, the camera that was used in the project is the B029201 4K 8MP. Based on the 1/4" Sony IMX219 image sensor, this camera provides high-quality images in a small device. The main advantage over other options, is that it is a USB camera, so basically Plug and Play. It is assembled in a metal case with a rotatable bracket.

Signal Tower TC-9539296

The idea of the project is to show how to build and set up a defect detection system, which is commonly used in industrial processes. It was decided to have an indicator by the

Figure 15 – Raspberry Pi 4 B 8GB.

Source: (TRU, 2024a).

Figure 16 – ArduCam B029201.

Source: (ARDUCAM, 2023).

camera, just like in real industry. For this, a 24V signal tower containing three different LED colors (basically a traffic light) was used.

5.3.2 Software tools

In this section, the software tools that were used are going to be shown and explained.

Linux

For both the VM running the backend service and the Raspberry Pi that is acquiring the images, Linux is the operating system. The two of them are running Ubuntu distros. The Raspberry is using Ubuntu Desktop, which includes a graphical user interface, making it easier

Figure 17 – Signal Tower TC-9539296.

Source: (TRU, 2024b).

to check on the webcam. The VM runs Ubuntu with no graphical interface.

systemd

Another used tool was the systemd, which according to Red Hat (2024) is a system and service manager for Linux operating systems. It was used to create some specific services for the project.

Python

Python is a high-level programming language that had its first release around 1991. Among the main goals of Python, is the focus on code readability and ease of use. It supports OOP and is one of the most popular programming languages. It is mainly used for data science/machine learning and web development (backend), but it has been acquiring space in other sorts of applications, like hardware programming (an area that used to be dominated by high-performance programming languages), which also involves this project. The main Python libraries used will be described in sequence. The minor and less complex ones will be shortly described in the Development chapter as they appear.

OpenCV

"OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products" (OPENCV, 2020). It was mostly used for image saving and acquisition.

threading

Threading is a library that helps implement multithreading in Python programs.

socket

This Python library helps write code to create and interact with socket objects, i.e. a socket server.

os

The OS module on Python provides ease of use to interact with the operating system. It helps define paths, create/delete folders and files, and access some of the operating system dependencies.

FastAPI

According to its own documentation, FastAPI is a modern, high-performance, web framework for building APIs with Python. It was used for the backend of the project.

ultralytics

Ultralytics is a company focused on AI. They focus on creating and providing the best possible models through their library. The YOLOv8 model used is theirs.

PyTorch - torchvision

PyTorch is a well known machine learning framework. In the context of this project, the torchvision package was used. As stated in their documentation, it is a package that contains popular datasets, model architectures, and common image transformations for computer vision (which is the reason why it was used).

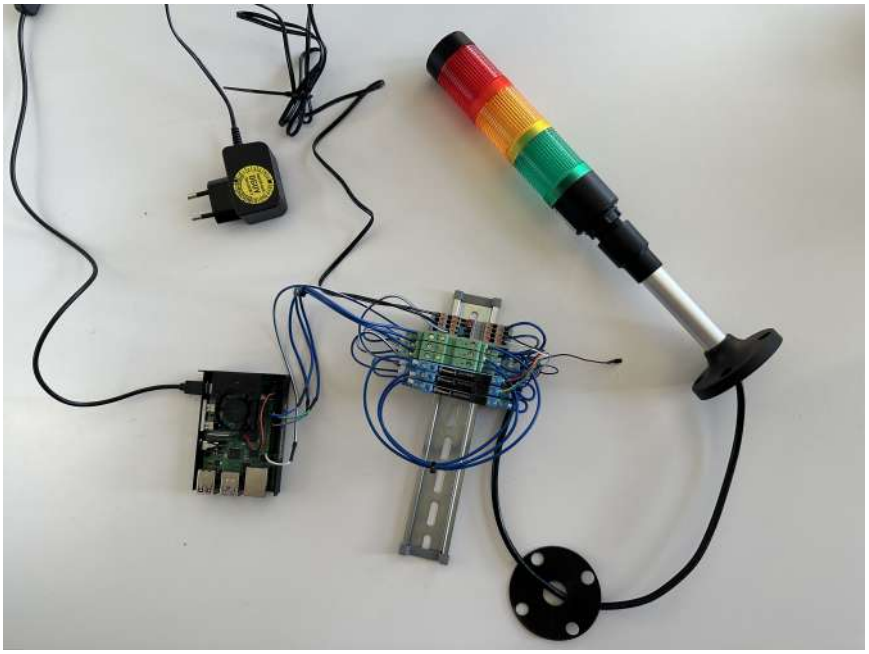
5.4 HARDWARE SETUP

Starting the project, after selecting the hardware, it was needed to assemble it. The signal tower needs 24V as supply voltage, while the Raspberry Pi can only provide 3.3V. To solve that, relays were used. As a first glimpse of the setup, in Figure 18, the Raspberry Pi sends low-voltage signals to the relays, where the high-voltage circuit is then activated, allowing the control of the signal tower.

Talking a little about relays, they are switches that turn circuits on or off. They take an electrical signal as input to either connect or disconnect another circuit. Some of the common usages are:

- Control of high voltage circuits/devices by low voltage ones, mediated by the relay.
- Protection of electrical systems and minimization of damages due to over currents/voltages (PRASAD, 2022).

Figure 18 – Hardware setup.



Source: author.

Going back to Figure 18, the relay configuration was made by the electrical department of IPT. The green relay is the one that is effectively taking the output of Raspberry's GPIO pins. When these are received, an internal high-voltage circuit is activated, providing the 24V output. This output is going through another relay, which takes 24V as input and can output as high as 250V. This second one, in this case, is working as a security gadget for the circuit. In applications where bigger voltages are needed, the second relay would be able to provide it.

As described in Raspberry (2024a)'s documentation, the device has multiple pins available. Figure 19 shows all the available pins. There are different types:

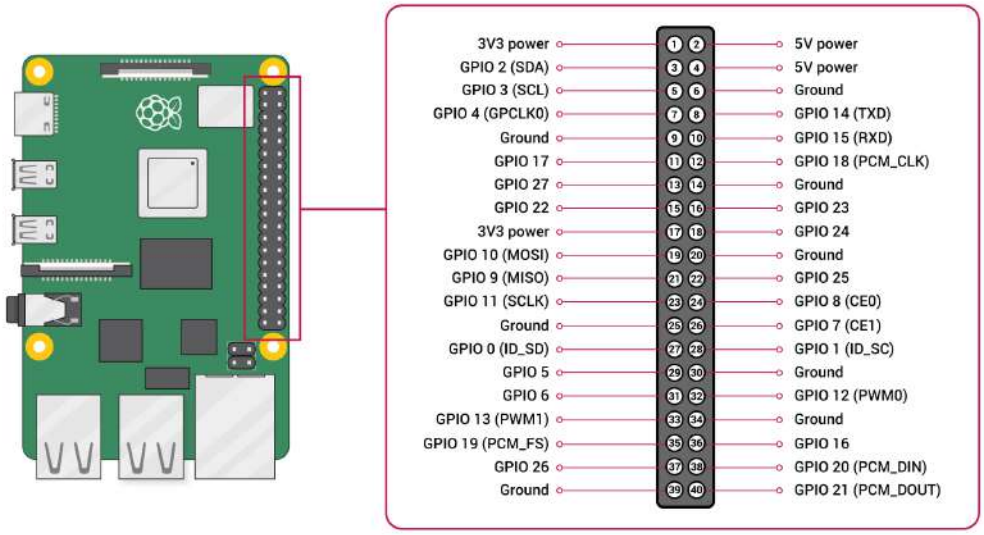
- Power: there are 3.3V and 5V pins for power supply.
- Ground: pins to ground the circuit.
- GPIO: general-purpose input/output pins. Multipurpose pins that can be used to interact with external devices by reading/sending digital signals.

GPIO pins were used to communicate with the signal tower. A test script was used to test the full setup.

```
from gpiozero import LED
from time import sleep

green = LED(25)
yellow = LED(22)
```

Figure 19 – Raspberry Pi's pins.



Source: (RASPBERRY, 2024b).

```
red = LED(23)

while True:
    yellow.off()
    red.on()
    sleep(.5)
    red.off()
    green.on()
    sleep(.5)
    green.off()
    yellow.on()
    sleep(.5)
```

It uses GPIO pins 22, 23, and 25 to define LED objects, from the gpiozero library. The LED object defines them as output pins, and using the on() and off() methods, it is possible to activate and deactivate the pins, sending signals to the relay that activates the signal tower. With the test code, it was possible to conclude that the signal tower was receiving the commands from the Raspberry Pi, which in this case, was simulating a very quick traffic light.

5.5 SOFTWARE SETUP - RASPBERRY PI

The code that is running on the Raspberry Pi is divided into two parts: the script responsible for both the socket client and image acquisition and the systemd service responsible for connecting the VPN.

5.5.1 VPN connection

To connect the different devices, mainly the Raspberry and the VM that is running both the server and the backend service, a VPN connection was used. An OpenVPN file was used to establish the connection. A command is called taking the file as a parameter and a password is required to establish the connection. A systemd service was defined so the VPN would automatically connect as the Raspberry Pi was turned on.

First, a script was created. It started defining that the script should be interpreted using the expect command. The timeout was set to -1, so it waits indefinitely until the desired pattern is returned. In this case, it is the password. The spawn command creates a new process responsible for running the OpenVPN file. Then, the "Password" is expected. The password is sent, and the last expect command waits for the process to end before finishing it.

```
#!/usr/bin/expect -f

set timeout -1
spawn sudo openvpn /path/to/ovpn/file/vpn.ovpn
expect -re "Password"
send "password\r"
expect eof
```

With the script created, a systemd service file was defined. It starts with a [Unit] section containing the description of the service and conditions for the service to start. The conditions are initialization of the basic network services and when the network is online. The [Service] describes the execution of the service. It defines the vpnsetup.sh file should be executed using the expect command. It also states that the service should always restart in case of failure. The final section, [Install], defines the service should be started in the default target (after all system initialization is complete).

```
[Unit]
Description=Connect to VPN on startup
After=network.target network-online.target

[Service]
ExecStart=/usr/bin/expect /home/demo/Desktop/vpn/vpnsetup.sh
Type=simple
Restart=always

[Install]
WantedBy=default.target
```

With the vpnrun.service file ready, some commands were required to activate and run the service. "daemon-reload" is used so systemd can detect the newly created service file. Then, using "enable" followed by "start", the service is up and running.

5.5.2 Image acquisition and socket client

The first part of this code is used to define the used libraries and some global variables. The logging library is used for more flexible handling of logs in Python. It provides a more complete error diagnosis, auxiliating in debugging the code. The IP_ADDRESS and PORT variables will be further explained in the next section, along with the client_socket. A threading.Lock() object is attributed to a variable. It is useful in multithreading when a resource is being accessed by multiple threads, which could lead to issues. The lock prevents different threads access the same resource at the same time (e.g. the current_frame variable is being updated and another function tries to use it). The previously cited LED objects define the pins used to communicate with the traffic light column.

```
import cv2
import threading
import socket
import logging
import pickle
import struct
import time
from datetime import datetime
from gpiozero import LED

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

IP_ADDRESS = "10.34.97.244"
PORT = 5003

client_socket = None
current_frame = None
exit_flag = False
lock = threading.Lock()

green = LED(25)
yellow = LED(22)
red = LED(23)
```

The connect_to_server() function is responsible for establishing the connection to the server. In case it is unable to do it, it will wait for 5 seconds, and then try again.

The function that is actively in charge of acquiring the images is camera_thread(). It checks for the camera availability, in case it is not possible, an error is raised and after 5 seconds another attempt is made. In case it connects and the exit_flag is False, it keeps on updating the current_frame making use of the lock that was previously defined.

```
def connect_to_server():
    global client_socket
    while not exit_flag:
```

```
            try:
                client_socket = socket.socket(socket.AF_INET, socket.
                    SOCK_STREAM)
                client_socket.connect((HOST, PORT))
                logger.info("Connected to server.")
                return
            except socket.error as e:
                logger.error(f"Unable to connect to server, retrying:
                    {e}")
                time.sleep(5)  # Wait 5 seconds before retrying

    def camera_thread():
        global current_frame
        while not exit_flag:
            try:
                cam = cv2.VideoCapture(0)
                if not cam.isOpened():
                    raise IOError("Camera not accessible!")
                logger.info("Camera connected!")
                while not exit_flag:
                    ret, frame = cam.read()
                    if not ret:
                        logger.error("Error in grabbing frame!")
                        break
                    with lock:
                        current_frame = frame
                cam.release()
            except IOError as e:
                logger.error(f"{e}. Retrying in 5 seconds")
                time.sleep(5)
```

The last function is called main(). It is responsible for connecting to the server, acquiring the images, and sending them over to the server. It defines a thread for the camera_thread() function and calls the connect_to_server() function. Once connected, it waits for the commands that will arrive. In case the command is "capture_image", it uses the lock and defines a dictionary containing the current image and a timestamp of when the image was acquired. Then it is sent to the server. When the command is "state", it will check the value of state (normal or defect) and activate the corresponding LED. the final command, "close", closes the connection.

```
    def main():
        global exit_flag, client_socket
        client_socket = None
        thread = threading.Thread(target=camera_thread)
        thread.start()
        try:
            while not exit_flag:
                if client_socket is None:
```

```
                connect_to_server()
            try:
                command = client_socket.recv(1024).decode('utf-8')
                if command == 'capture_image':
                    with lock:
                        if current_frame is not None:
                            data_dict = {
                                'timestamp': datetime.now().
                                    strftime('%Y-%m-%d_%H-%M-%S'),
                                'image': current_frame
                            }
                            send_data = pickle.dumps(data_dict)
                            data_size = struct.pack("I", len(
                                send_data))
                            client_socket.sendall(data_size +
                                send_data)
                elif command.startswith('state:'):
                    state_value = int(command.split(':')[1])
                    if state_value == 1:
                        logger.info("Normal!")
                        green.on()
                        time.sleep(2)
                        green.off()
                    elif state_value == 2:
                        logger.warning("Defect!")
                        red.on()
                        time.sleep(2)
                        red.off()
                elif command == 'close':
                    logger.info("Server requested shutdown.")
                    client_socket.sendall(b'Client shutting down.'
                        )
                    break
            except socket.error as e:
                logger.error(f"Socket error, attempting to
                    reconnect: {e}")
                if client_socket:
                    client_socket.close()
                    client_socket = None
    except KeyboardInterrupt:
        logger.info("Interrupted by user.")
    exit_flag = True
    thread.join()
    if client_socket:
        client_socket.close()
```

The last part was setting up a systemd service to automatically connect the Raspberry Pi with the server, making it ready to use right after startup. It has the same structure as the VPN

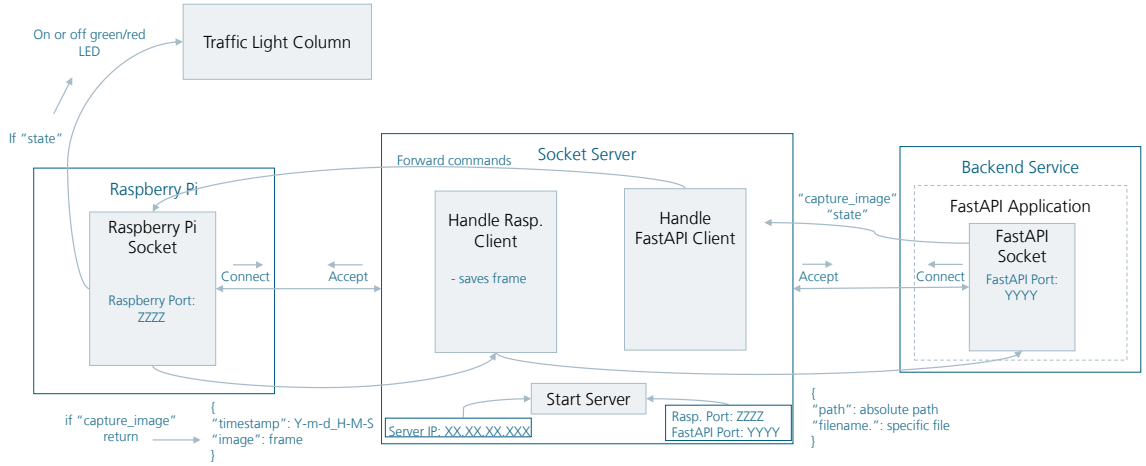
service, with some differences in the calls and actions. Under [Service], the path in which the Python script is located was defined in WorkingDirectory. The ExecStart parameter is where the script and interpreter are defined. The rest is the same as the VPN service.

```
[Unit]
Description=Camera script service!
After=network.target network-online.target
[Service]
Type=simple
User=demo
WorkingDirectory=/path/to/python/script/
ExecStart=/path/to/python/interpreter/python camerascript.py
Restart=always
[Install]
WantedBy=multi-user.target
```

5.6 CLIENT/SERVER USING SOCKETS

Since many devices were used in the project and they were dependent on the data that would come from each one, an architecture had to be defined for this information exchange. It was defined that a client/server architecture based on sockets would be used. Figure 20 shows the full architecture that was used, along with its relations, which will be further explained in this section.

Figure 20 – Client/Server socket-based setup.



Source: author.

5.6.1 Server

Starting with the server, it contains 4 functions:

1. start_server();
2. accept_connections(server_socket, handler_function);
3. handle_raspberry_pi_client(client_socket, address);
4. handle_fastapi_client(client_socket, address).

It first defines the used libraries. Some variables are also declared, being the name self-explained. The PATH one indicates where the images that were acquired should be stored. They are called "raw" due to being straight out of the camera, with no processing nor prediction made on them. One variable for each socket is also defined.

```
import socket
import threading
import cv2
import pickle
import struct
import os
import json

IP_ADDRESS = "XX.XX.XX.XXX"
FASTAPI_PORT = YYYY
RASP_PORT = ZZZZ
PATH = "/path/to/save/raw/image"

fastapi_client_socket = None
raspberry_pi_socket = None
```

The start_server() function is the one responsible for starting the server. As explained previously in subsection 4.4.1, the server has to do some processes before being able to accept connections from different clients. It starts by defining the socket types and then binding specific addresses and ports to each socket, one for the Raspberry Pi and one for the FastAPI service. After the binding part, the server is ready to listen and wait for new connection requests. Multithreading is used to allow the simultaneous connection of both sockets.

```
def start_server():
    fastapi_server_socket = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
    raspberry_server_socket = socket.socket(socket.AF_INET, socket
        .SOCK_STREAM)

    fastapi_server_socket.bind((IP_ADDRESS, FASTAPI_PORT))
    raspberry_server_socket.bind((IP_ADDRESS, RASP_PORT))
```

```
    fastapi_server_socket.listen()
    raspberry_server_socket.listen()

    print("Server listening for FastAPI and Raspberry Pi clients
        ...")

    threading.Thread(target=accept_connections, args=(
        fastapi_server_socket, handle_fastapi_client)).start()
    threading.Thread(target=accept_connections, args=(
        raspberry_server_socket, handle_raspberry_pi_client)).start
        ()

def accept_connections(server_socket, handler_function):
    while True:
        client_socket, address = server_socket.accept()
        threading.Thread(target=handler_function, args=(
            client_socket, address)).start()
```

The accept_connections(server_socket, handler_function) function is responsible for accepting the connections from the Raspberry Pi and FastAPI clients. It takes as arguments the socket of each one of them and also the functions that are responsible for handling the clients. It runs a multithread for the handler functions.

For the handle_raspberry_pi_client(client_socket, address) function, which is responsible for the operations that are performed with the Raspberry Pi client, it takes as input the socket and its address. The raspberry_pi_socket variable is defined as global due to its usage in other functions. It keeps on listening for any information that might come from the client. Once there is data available, it checks for the size of the message based on the data_size_header variable. A dictionary is being sent by the Raspberry Pi, containing an image and a timestamp of when it was acquired. The image is saved. It checks for the existence of the FastAPI socket, and sends a JSON containing the path where the raw images are being stored and the specific name of the current image.

```
def handle_raspberry_pi_client(client_socket, address):
    global raspberry_pi_socket
    raspberry_pi_socket = client_socket
    print(f"Connected to Raspberry Pi client: {address}")
    try:
        connection = client_socket.makefile('rb')
        while True:
            data_size_header = connection.read(4)
            if not data_size_header:
                break
            data_size = struct.unpack("I", data_size_header)[0]
            data = connection.read(data_size)
            if not data:
```

```
                    break

                image_data = pickle.loads(data, fix_imports=False,
                    encoding="bytes")
                frame = image_data['image']
                timestamp = image_data['timestamp']
                image_filename = f"image_{timestamp}.jpg"
                cv2.imwrite(os.path.join(PATH, image_filename), frame)

                if fastapi_client_socket:
                    saved_image_filename = os.path.join(PATH,
                        image_filename)
                    saved_image_dict = {
                        'path': saved_image_filename,
                        'filename': image_filename
                    }
                    saved_image_json = json.dumps(saved_image_dict)
                    fastapi_client_socket.sendall(saved_image_json.
                        encode())

                print(f"Received image from Raspberry Pi saved as '{
                    image_filename}'")
        finally:
            raspberry_pi_socket = None
            client_socket.close()
```

The last function, handle_fastapi_client(client_socket, address), handles the FastAPI client connection. It takes the commands that are sent by the FastAPI client and forwards them to the Raspberry Pi. The commands are "capture_image" for acquiring a new frame, "state" for the result of the prediction, and "close" to close the connection.

```
def handle_fastapi_client(client_socket, address):
    global fastapi_client_socket
    fastapi_client_socket = client_socket
    print(f"Connected to FastAPI client: {address}")
    try:
        while True:
            command = client_socket.recv(1024).decode()
            if command == 'capture_image':
                print("FastAPI requested an image capture")
                if raspberry_pi_socket:
                    raspberry_pi_socket.sendall(command.encode())
                else:
                    print("Raspberry Pi client is not connected.")
            elif command.startswith('state:'):
                state_value = command.split(':')[1]
                if raspberry_pi_socket:
                    raspberry_pi_socket.sendall(f"state:{
```

```
                    state_value}".encode())
            elif command == 'close':
                break
    finally:
        fastapi_client_socket = None
        client_socket.close()
```

5.6.2 Clients

Both the clients, Raspberry Pi and FastAPI service, are pretty similar. They have three global variables: the IP_ADDRESS, PORT, and client_socket. The first two are used to connect to the server, and the last one to interact with the socket throughout the code (i.e. send commands and receive the image path).

5.7 BACKEND SERVICE

This backend service was built around the idea of integrating it to an existing web application of the company: the Intelligent Quality Platform (IQP). IQP is an initiative by IPT built to demonstrate the extensive range of machine learning use cases to the industry, including predictive maintenance of machinery, prediction of product quality, and detection of machine/products anomalies. It focuses on having various ML applications in a standardized platform for easier access and control.

The backend was built using FastAPI. This framework, as stated previously, is designed to build fast efficient APIs in Python. APIs, according to AWS (2024a) are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols. FastAPI works based on REST APIs. It stands for Representational State Transfer and works based on defining a set of functions. In the scope of this project, the following methods were used:

- GET: this type of request is used to retrieve information. It will never modify it, only accessing it and, for example, display it on the frontend.
- POST: request used to create new resources (i.e. request an image to be takena and saved).
- ON EVENT: endpoints based on the ocurrency of events. Usually used in the startup and/or shutdown of an application.

Before explaining the endpoints, some libraries were called and variables defined. The PIL library is responsible for handling images, to open and interact with. Then the IP_ADDRESS and PORT were set. Some global variables for the client socket object, filename, and the saving path to the images post-prediction. To end, some FastAPI configurations were made. A FastAPI object, app, was defined and the addresses that should be able to interact with the backend were also set.

```
from fastapi import FastAPI, UploadFile
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import FileResponse
from ultralytics import YOLO
import cv2
import socket
import os
from PIL import Image, ImageDraw
import json
import numpy as np
import time

IP_ADDRESS = "XX.XX.XX.XXX"  # Server's IP
PORT = YYYY                  # Port for FastAPI to connect to the
    socket server
client_socket = None
filename = None
SAVE_PATH = '/path/to/save/predicted/images/'

app = FastAPI()

# CORS middleware setup
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:WWWW", "http://localhost:GGGG
        "],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

For the ON EVENT methods, the startup one was used to stablish the connection with the socket-based server. The second one, shutdown, was used to close this connection.

```
@app.on_event("startup")
async def startup_event():
    global client_socket
    client_socket = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
    client_socket.connect((IP_ADDRESS, PORT))
    print("connected")

@app.on_event("shutdown")
async def shutdown_event():
    global client_socket
    if client_socket:
        client_socket.close()
```

The POST method, used in "request-image", is responsible for requesting images. The command is sent to the server, which forwards it to the Raspberry Pi. After having the image saved, the server forwards the path and name of the most recent frame. The prediction is made on top of this image, and a new image, with the prediction (bounding box) on it, is saved. Then, based on the prediction, a command is sent to the server about it being normal or defective.

```
@app.post("/request-image/")
    async def request_image():
        global client_socket
        global filename
        try:
            client_socket.sendall(b"capture_image")  # Send
                command to capture image
            # Wait for the server to send back the image path
            image_path_data = client_socket.recv(1024)
            image_path_data = image_path_data.decode('utf-8').
                strip()
            image_path_data = json.loads(image_path_data)
            image_path = image_path_data['path']
            filename = image_path_data['filename']
            if image_path:
                if os.path.exists(image_path):
                    pred = prediction(image_path)
                    if len(pred) > 0:
                        state = 2
                    else:
                        state = 1
                    annotated = annotate(Image.open(image_path),
                        pred)
                    save_prediction(annotated, filename)
                    time.sleep(5)
                    state_message = f"state:{state}"
                    client_socket.sendall(state_message.encode())
                    return {"status": "Success", "message": "Image
                        received and processed", "prediction":
                        pred, "state": state}
                else:
                    return {"status": "Failure", "message": "Image
                        file not found"}
            else:
                return {"status": "Failure", "message": "No image
                    path received from the server"}
        except Exception as e:
            print(f"Error during image request: {e}")
            return {"status": "Failure", "message": f"Error during
                image request: {e}"}
```

Some extra functions were used in the "request-image" endpoint, being "prediction",

"annotate", and "save_prediction". The "prediction" function is responsible for predicting on top of the acquired image. It returns a list containing the bounding boxes coordinates, the defect names, and the confidence of the prediction (0 to 1, the higher the better). The "annotate" function takes as input the image that the prediction was made on and the output of that prediction. It takes the image and draws the bounding boxes on top of it. The last one, "save_prediction", saves the image with the prediction so the front end can access and display it.

```
def prediction(image_path):
    model = YOLO('best.pt')
    image = cv2.imread(image_path)
    results = model.predict(image)
    result = results[0]
    output = []
    for box in result.boxes:
        x1, y1, x2, y2 = [round(x) for x in box.xyxy[0].tolist()]
        class_id = box.cls[0].item()
        prob = round(box.conf[0].item(), 2)
        output.append([x1, y1, x2, y2, result.names[class_id],
            prob])
    return output

def annotate(image, boxes):
    draw = ImageDraw.Draw(image)
    for box in boxes:
        x1, y1, x2, y2, object_type, probability = box
        draw.rectangle([(x1, y1), (x2, y2)], outline="red", width
            =3)
        draw.text((x1, y1 - 10), f"{object_type} ({probability})",
             fill="red")
    return image

def save_prediction(image, filename):
    image_np = np.array(image).astype(np.uint8)
    image_np_bgr = cv2.cvtColor(image_np, cv2.COLOR_RGB2BGR)
    cv2.imwrite(os.path.join(SAVE_PATH, filename), image_np_bgr)
```

The GET method, used in "last-predicted-image" is responsible for getting the last image that was saved (using the global variable "filename"). It is used by the frontend to display the prediction. The other GET endpoint was set for testing purposes, to check if the backend was working.

```
@app.get("/")
async def home():
    return {"status": "ok"}

@app.get("/last-predicted-image")
async def last_predicted_image():
    if filename:
```

```
            return FileResponse(SAVE_PATH + filename)
        else:
            return None
```

5.8 YOLOV8 TRAINING

This section will talk about the YOLOv8 model and, mostly, the data that was used to train it. Since the use case is based on a dataset that is available online, it was decided to print some of the images to test the model later (none of the images that were printed were used to train the model).

YOLOv8 was selected as the model due to it being extremely efficient, as seen in 4, and easy to use, bringing good performance and allowing to focus on the other parts of the system's architecture. The reason to selecting it is similar to the use case, being easy to use while still being a model that is effectively used in the industry.

5.8.1 Original data set

In Figure 21, it is possible to see the metrics of the training and validation using only the original data set, with no preprocessing nor data augmentation. The most relevant metrics when evaluating an object detection model, the Mean Average Precision (mAP), is not stable and changed very aggressively through the training epochs. Compared to what can be seen in Chapter 6 (Figure 28) where data augmentation techniques were used, the performance improved drastically. Figure 22 shows a prediction made by the model trained with only the original data set, showing that even though it is able to predict a defect correctly, it also fails, predicting a non-existing defect. To fix these issues, data augmentation techniques were used, improving the overall performance of the model.

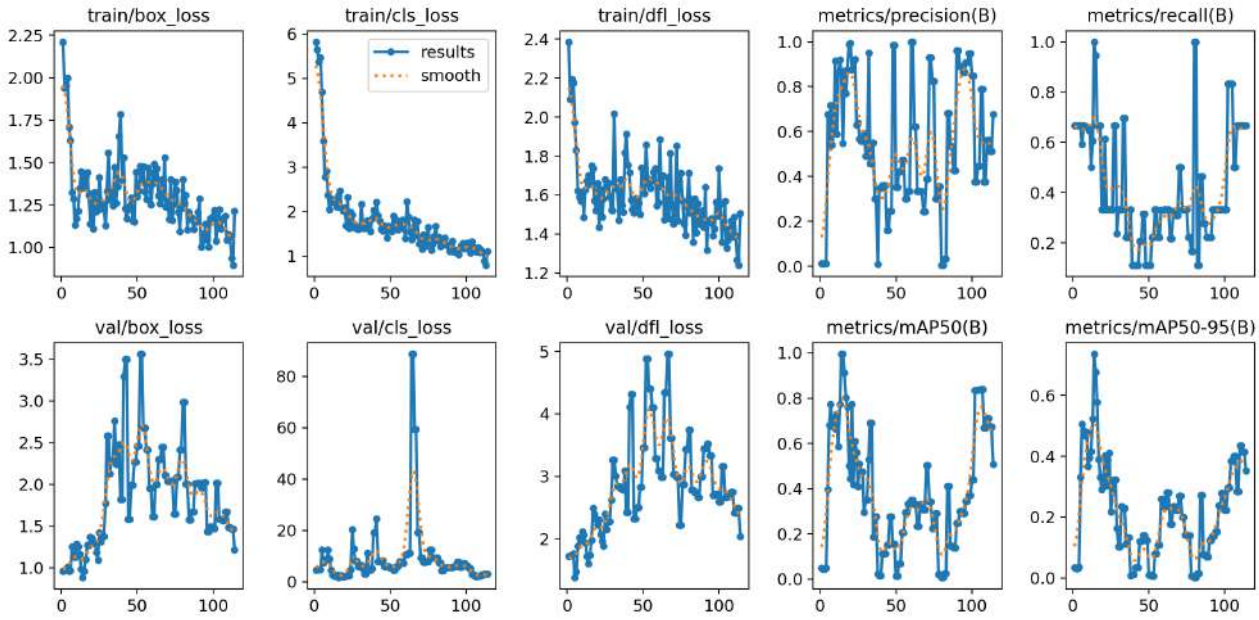
5.8.2 Data Preprocessing

The model YOLOv8 has very detailed documentation, provided by Ultralytics (2024). It is easy to use the model, being the main concern of the data that is being fed to it. The developed system works by getting images from the camera connected to the Raspberry Pi. Figure 23 shows the same image from 2 perspectives, 23a shows what is being acquired by the camera while 23b shows what is the original from the dataset.

It is possible to see the difference between both images, especially the brightness and "blur" differences, comparing the acquired and dataset images. To begin with, the chosen images, which represent most of the dataset (just 2 or 3 of each type of defect were left for printing and testing), were all preprocessed.

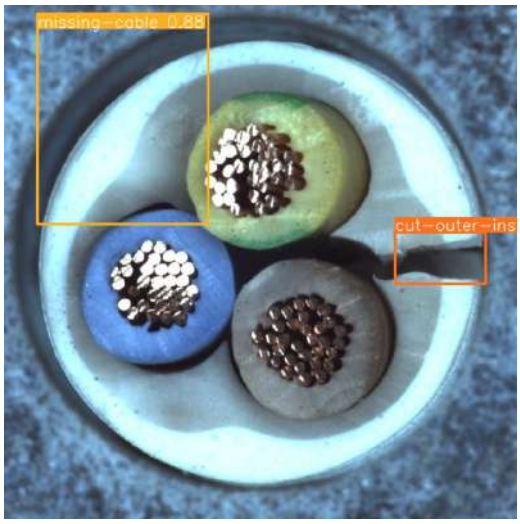
This preprocessing was based on Data Augmentation techniques. AWS (2024b) defined: "Data augmentation is the process of artificially generating new data from existing data, primarily to train new machine learning (ML) models. ML models require large and varied datasets for

Figure 21 – Train/Validation loss and performance metrics with no data preprocessing.



Source: author.

Figure 22 – Post prediction.



Source: author.

initial training, but sourcing sufficiently diverse real-world datasets can be challenging because of data silos, regulations, and other limitations. Data augmentation artificially increases the dataset by making small changes to the original data."

Looking for images that were more similar to the ones acquired by the camera, two functions available in PyTorch's torchvision package were used: GaussianBlur() and ColorJit-

Figure 23 – Camera acquired image vs dataset image.

(a) Camera acquired image. (b) Dataset image.

Source: author / (MVTEC, 2024).

ter(). The first one was used to get the images a little more "blurry", while the second changed the brightness of the image, making it more similar to the images acquired. Figure 24 shows the results after these first two steps, which were applied to all the chosen data. With this new dataset, many different data augmentation techniques were applied.

Figure 24 – Example image after using GaussianBlur() and ColorJitter() methods.

Source: author (based on (MVTEC, 2024).

The following piece of code were used to perform the desired augmentation:

```
import os
from PIL import Image
import torchvision.transforms as transforms

ORIGINAL_PATH = "path/to/dataset"
```

```
AUGMENTED_PATH = "path/to/save/augmented"

augmentations = [
    transforms.RandomHorizontalFlip(1),
    transforms.RandomVerticalFlip(1),
    transforms.RandomRotation((0,10)),
    transforms.RandomRotation((-10,0)),
    transforms.ColorJitter(contrast=(0.8,0.95))
    # transforms.GaussianBlur(kernel_size=(5,9),sigma=(5)),
    # transforms.ColorJitter(brightness=(1.2,1.4)),
]

def augment():
    for image_name in os.listdir(ORIGINAL_PATH):
        image_path = os.path.join(ORIGINAL_PATH, image_name)
        img = Image.open(image_path)
        for idx, augmentation in enumerate(augmentations):
            augmented_img = augmentation(img)
            augmented_img_path = os.path.join(AUGMENTED_PATH, f"{
                image_name.split('.')[0]}_aug_{idx}.png")
            augmented_img.save(augmented_img_path)
        print(f"Image {image_name} just got augmented!")
    print("Data augmentation and saving completed.")
    pass

if __name__ == "__main__":
    augment()
```

The chosen augmentations were:

- RandomHorizontalFlip: takes as input the probability of flipping the image horizontally, in this case, 100%.
- RandomVerticalFlip: same as the previous one, but flips vertically.
- RandomRotation: applies a random rotation on the image. In this case, a random angle between [0, 10] and [-10, 0].
- ColorJitter(contrast): randomly changes the contrast. In this case, a random value between [0.8, 0.95].

Figure 25 shows all the performed augmentations. From each of the initial pictures of the dataset (total of 37) another 5 were generated, leading to a final dataset of size 222. The train/test split will be covered in the next subsection.

Having the 222 images dataset defined, the only thing left to do was label the images. Making use of Roboflow, an online tool that allows an easy way to annotate images for object detection tasks. This part demanded quite some time, being a very manual task.

Figure 25 – Augmented images.

(a) Horizontal flip.

(b) Vertical flip.

(c) Random rotation.

(d) Contrast.

Source: author (based on MVTec, 2024).

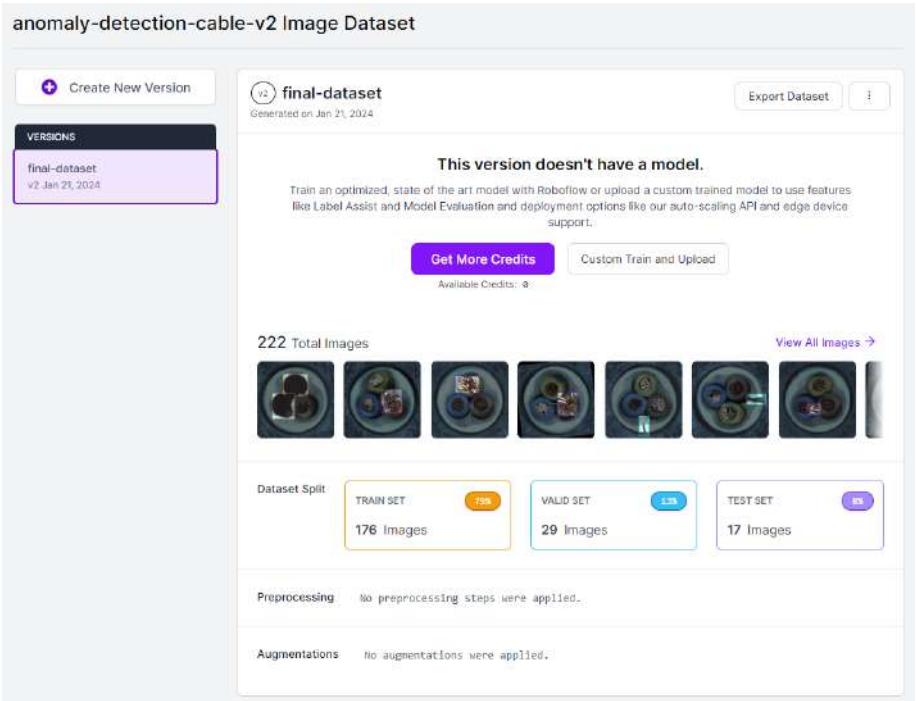
After uploading the images to Roboflow's platform, the next step is doing the annotation. Figure 26 shows the process of annotating them. Using the Bounding Box Tool, the one on the right of Figure 26, it is possible to define the bounding boxes that will define the object to detect. After setting the bounding box, a class definition is required, in this case, a missing cable defect. After having the images ready, the dataset creation is set. The 222 images are separated into 4 classes: bent wires, cut inner insulation, cut outer insulation, and missing cable. Right after, the size of the dataset is defined. The train/validation/test split is 79% for training, 13% for validation, and 8% for testing. For the preprocessing steps that Roboflow provides, none were applied. Same for the augmentations. Figure 27 shows the full configuration of the dataset.

Another solid feature of Roboflow is the possibility of exporting the dataset to YOLOv8 format.

Figure 26 – Roboflow's annotating tool.

Source: author.

Figure 27 – Dataset full configuration.



Source: author.

5.8.3 Model training

YOLOv8's documentation is very rich and complete, making it quite simple to use. For the training, a Google Colab notebook was used. "Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education" (GOOGLE, 2024a). To check on the used notebook, click here.

To begin with the training, the environment was set up. A GPU option was selected and the needed libraries were installed: ultralytics and roboflow. When exporting a dataset from roboflow, it provides a code snippet to download it. By running the cell, the dataset is ready for use. Ultralytics provides two ways of using its tools: running a Python script or through CLI (Command Line Interface) commands. In this case, CLI was used. Many parameters can be taken in a call, being the main and used ones:

- Task: define the task to be performed. In this case, object (defect) detection.
- Mode: what to do with the model. Can be training, validating, predicting, exporting, tracking, and benchmarking.
- Model: choose which of the available YOLOv8 models will be used. The available options are: nano, small, medium, large, and extra large. Varying from the small one which has less complex layers and architecture, to the more complex extra large one, which posseses higher accuracy and is heavier (demands more powerful hardware).
- Data: indicates the dataset path.
- Epochs: define how many epochs will be used to train the model. Ultralytics provides built-in early stopping functionalities, so even if a higher number of epochs is defined, once the training does not improve significantly it stops.
- Imgsz: image size defines the size of the input images.

```
!yolo task=detect mode=train model=yolov8m.pt data={dataset.
    location}/data.yaml epochs=500 imgsz=640
```

After finishing the training, the model's weights are saved in two files: best and last. Best being the top results and last the final one (either after all the epochs or after the early stopping).

5.9 DEPLOYMENT USING DOCKER CONTAINERS

The final part of the project was to deploy it making use of docker containers. Docker is a platform designed to help developers build, share, and run container applications (DOCKER, 2024b).

The containers were defined as displayed in Figure 13. One for the server, one for the backend, and one for the frontend (the frontend implementation and deployment are not part of this project). Two Dockerfile files were created, one for the server and one for the backend. Starting with the server, it defines that the container should run a 3.11 version of Python. The working directory is the same as where the file is located. The RUN command is used to install pip and upgrade it if needed (pip is the standard Python packages manager). It copies the "requirements.txt" file available in the directory and installs all the libraries listed there. The system is updated and some needed resources are installed. Then, the rest of the files are copied. This second copy is important because Docker works sequentially, meaning that each command inside the Dockerfile is considered a layer. So every time the container is rebuilt, docker will try to access and reuse layers from older builds. Since the "requirements.txt" rarely changes, these layers will be reused, and only the changes in the actual code/application are going to be rebuilt. It is also relevant that it is one of the last commands because all layers after the changed one are going to be rebuilt (DOCKER, 2024a). Once again, due to the layered organization of docker, the system is asked to be updated. The last line defines the command that should be run, in this case, the execution of the server script.

```
#Create a ubuntu base image with python 3 installed.
FROM python:3.11
#Set the working directory
WORKDIR /
RUN python -m pip install --upgrade pip
COPY requirements.txt /
RUN pip3 install -r /requirements.txt
RUN apt-get update
RUN apt-get install -y libgl1-mesa-glx
#Add app files
COPY . .
RUN apt-get -y update
#Run the command
CMD ["python3", "server.py"]
```

The backend Dockerfile is basically the same, being the main difference the script that will be executed and the working directory.

Since it was needed to handle multiple containers, the "docker-compose" CLI was used. It runs based on a .yml file, in which the desired container configurations are defined. Under "services", the two were set as live_demo_service_dev and live_demo_socket_dev. The server one is set to always restart and to be built based on the Dockerfile from the socketserver folder. The name is set and ports are defined. It maps the port number from the host device to the container one, separated by a ":". The same goes for the "volumes" argument, having the device path mapped to the container one. The backend service is set to depend on the server service, so it will only activate after the server is started. The rest of the configurations are the same, except for the path and ports.

```
version: '3.7'
services:
    live_demo_service_dev:
        depends_on:
            - live_demo_socket_dev
        restart: always
        build: ./
        container_name: live_demo_service_dev
        ports:
            - TTTT:TTTT
        volumes:
            - /path/to/save/raw/image:/path/container/raw

    live_demo_socket_dev:
        restart: always
        build: ./socketserver
        container_name: live_demo_socket_dev
        ports:
            - YYYY:YYYY
            - ZZZZ:ZZZZ
        volumes:
            - /path/to/save/raw/image:/path/container/raw
```

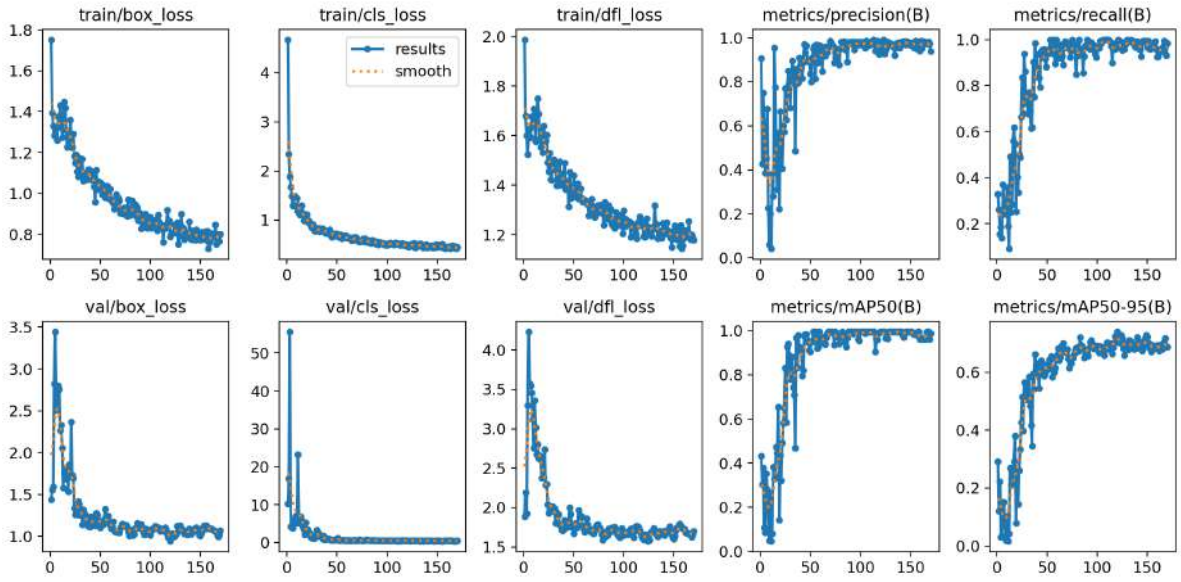
6 RESULTS

This chapter will present the results of the project, checking each part of the development chapter, and testing it out. First, the trained model will be evaluated. Then, Software setup - Raspberry Pi, Client/Server using sockets, and Backend service sections are going to be presented together. The Docker container deployment will come as last.

MODEL PERFORMANCE

Starting with the trained model, its performance was evaluated. Figure 28 shows the train and validation losses along with some metrics for the model performance. Both precision (P) and recall (R) curves indicate a good performance. The specific values of the validation can be seen in Table 1. It shows that both P and R had good results. The mAP50 had really good results, with a percentage of 99.5%. The mAP50-95, which considers the IoU threshold to vary, has a lower value of 0.742 across all classes. Although showing a lower value on the mAP50-95, the model performance is satisfying for the given purpose. The full Colab notebook with the training, validating, and testing can be seen here. A prediction for each type of defect can be seen in Figure 29.

Figure 28 – Train/Validation loss and performance metrics.



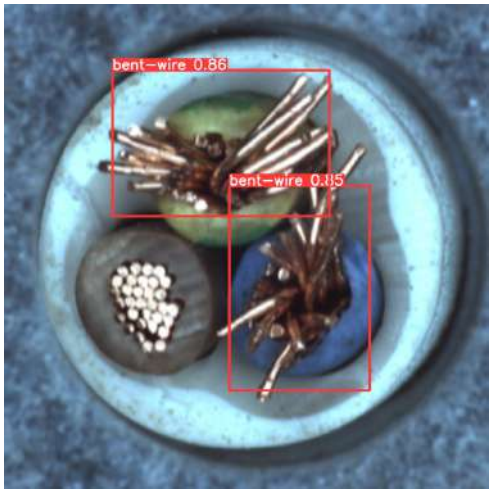
Source: author.

Table 1 – Validating results.

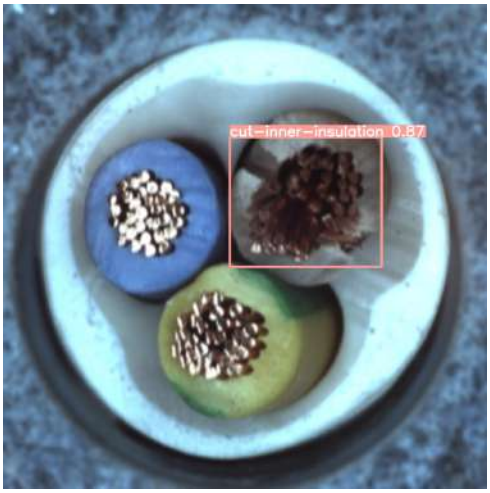
Class	Images	Instances	Box(P	Recall	mAP50	mAP50-95
all	29	33	0.943	0.99	0.995	0.742
bent-wire	29	7	0.905	1	0.995	0.663
cut-inner-insulation	29	11	0.898	1	0.995	0.762
cut-outer-insulation	29	6	1	0.959	0.995	0.608
missing-cable	29	9	0.969	1	0.995	0.936

Source: author.

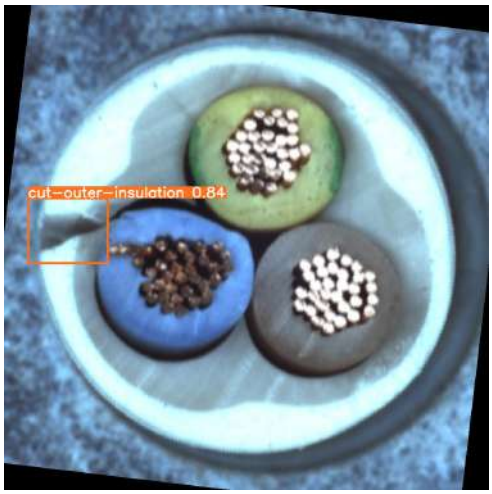
Figure 29 – Images post prediction.



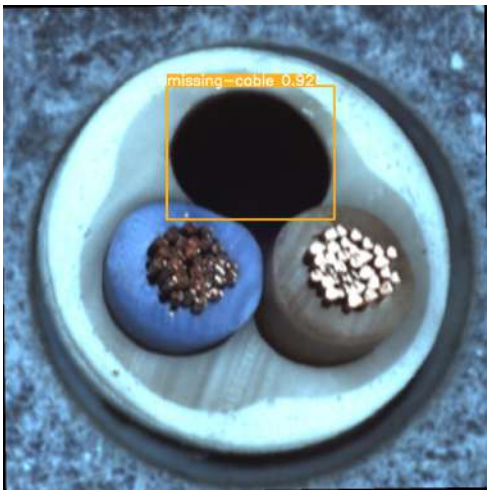
(a) Bent wire.



(b) Cut inner insulation.



(c) Cut outer insulation.



(d) Missing cable.

Source: author.

FULL TESTING: RASPBERRY PI, SERVER, BACKEND, FRONTEND

In Chapter 5, the Raspberry Pi setup was defined, the socket-based server was set, and the backend service was constructed. Now, for this section, all of them were tested together. Having the Server online, it will wait to accept any connection request, in the case, from the Raspberry and the backend. Some "print" statements were used to check on the connections as follows:

```
Server listening for FastAPI and Raspberry Pi clients...
Connected to FastAPI client: ('FastAPI IP', port)
Connected to Raspberry Pi client: ('Raspberry IP', port)
```

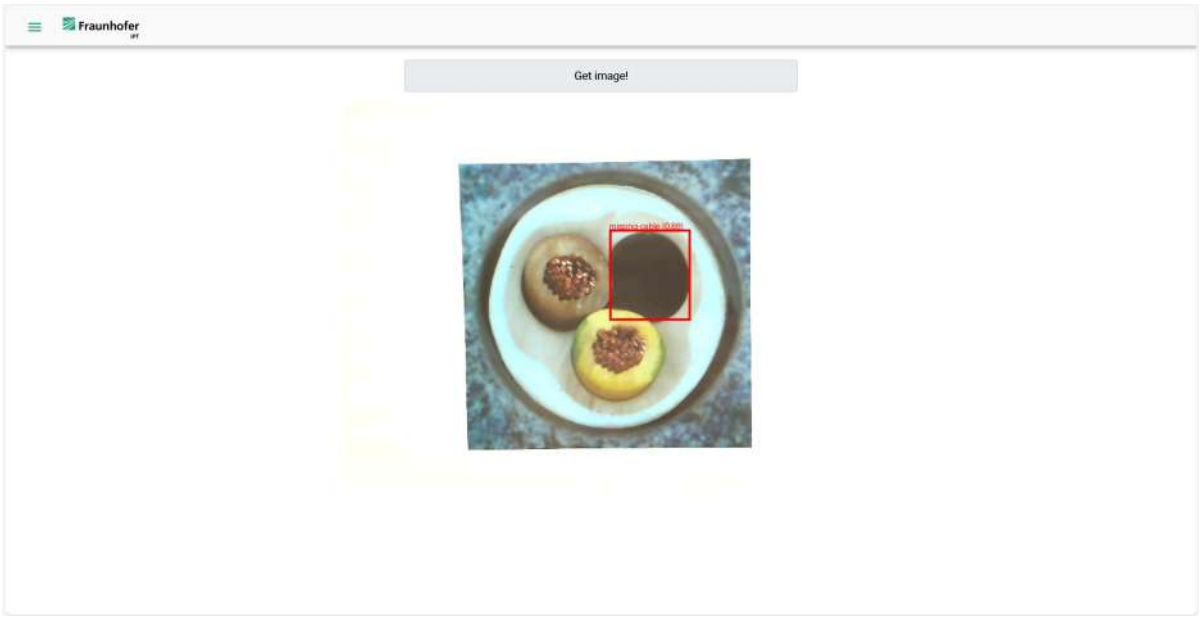
Having both connected, it means that the system is ready. As stated before, the frontend part was not covered in this project. By accessing the front end, a button is available, and after clicking it, the backend is called, using the POST method followed by the GET. After requesting the frontend, the backend outputs the prediction results and shows that both requests worked. Figure 30 shows the frontend with the prediction.

```
BACKEND OUTPUT

0: 480x640 1 missing-cable, 622.5ms
Speed: 6.8ms preprocess, 622.5ms inference, 3.3ms postprocess per
   image at shape (1, 3, 480, 640)
INFO:     127.0.0.1:44122 - "POST /request-image/ HTTP/1.1" 200 OK
INFO:     127.0.0.1:44122 - "GET /last-predicted-image HTTP/1.1"
   200 OK
```

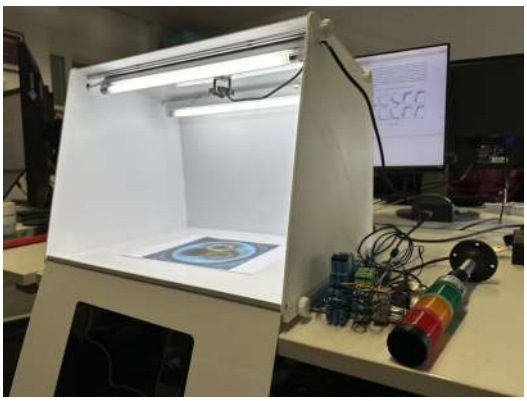
To conclude the results chapter, Figure 31 shows the final prototype that was built. The camera is fixed on the upper part, between two light sources. The image to be predicted is put right under it. All the images that are being used as tests were never seen by the model, they were separated before subsection 5.8.2.

Figure 30 – Frontend on the IQP web application.

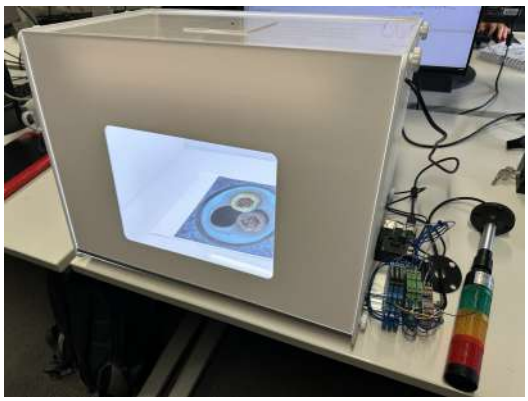


Source: author.

Figure 31 – Final prototype.



(a) Final prototype: front part opened.



(b) Final prototype: front part closed.

Source: author.

7 CONCLUSION

Digital transformation in production is a hot topic in the industry right now, with more companies looking forward to implementing new techniques and methods that will help assure safety and quality in their production. The digitization of their assets becomes a must in the context of Industry 4.0. Based on that, this project aimed at developing a demonstrator for defect detection, covering all the steps. Starting with the hardware and architecture setup, data acquisition and transmission, training of a CV model for defect detection, setup of a backend service on an existing web application, and the deployment making use of Docker containers. The developed prototype will be used to showcase what Fraunhofer IPT does and give a first example to companies that are looking forward to the digitization of their assets.

It started off with a solid literature review, in which the main concepts that would be needed were thoroughly reviewed/learned. The first contact with some technologies (backend, containers) was extremely enriching, providing a valuable skill to the author's professional life. The hardware and software selection went smoothly, being reasonably discussed between the author and the local supervisor. Having the hardware set up, the software that would run on the Raspberry Pi was developed. By making use of systemd services, the scripts that were defined would automatically start as the device turned on, automating the process of connecting to both the VPN and the socket-based server. The server was defined, along with specific functions to handle the different clients. Further on, the backend service was developed. Working based on RESTAPI methods, it fulfilled all the desired functionalities, making the predictions and returning the results to the traffic light column connected to the Raspberry Pi. To make the prediction, a CV model was trained. The data preprocessing was the main step towards the training of a YOLOv8 model. The last step covered the deployment of the service on the IQP web application of IPT.

The results were satisfying as the whole architecture that was set worked well. The images were being acquired and accordingly sent to the server, which was responsible for storing them. The server sending the image path over to the backend service worked well too, allowing the backend to make predictions on the acquired images and storing them. The model performed well with the real-time acquired images, which was a concern at first, given that the dataset images had differences compared to the ones coming from the camera. The size of the dataset was another concern. Data augmentation techniques were performed to address both these issues. The serial communication between the Raspberry Pi and the traffic light column went smoothly. The demonstrator was able to accomplish the defined goals and opened the path to further improvements.

For future work, that are some things that were already discussed between the author and the company's supervisor. The main ones being:

1. Frontend: define a virtual traffic light column, displaying the same output as the real one along with a timestamp. Also, displays an indication of the connected devices (Raspberry,

server).

2. Check on the image resolution of the prediction, making it higher.

3. Videostream: instead of receiving single images every request, set the backend so a live video will be displayed, with the predictions being made on top of it. By changing the RESTAPI methods to a Websocket-based endpoint it should be possible.

By making use of many different technologies and concepts, the overall knowledge that was gained in the development of this project is enormous. Many topics that compose the curriculum of the Automation and Control Engineering course were used. To cite the main ones: electronics, programming, computer networks, and artificial intelligence. The final result was satisfying and future work will definitely add up to the already existing system.

REFERENCES

AGARWAL, Harshit; AGARWAL, Rashi. First Industrial Revolution and Second Industrial Revolution: Technological Differences and the Differences in Banking and Financing of the Firms. **Saudi Journal of Humanities and Social Sciences**, 2017. Available from: https://www.academia.edu/41330434/First_Industrial_Revolution_and_Second_Industrial_Revolution_Technological_Differences_and_the_Differences_in_Banking_and_Financing_of_the_Firms.

ARDUCAM. **4K 8MP IMX219 autofocus USB camera module with metal case**. [S.l.: s.n.], Sept. 2023. Available from: https://www.arducam.com/product/arducam-autofoucs-imx219-usb-camera-b029201/.

AWS, Amazon. **What is an API (Application Programming Interface)?** [S.l.: s.n.], 2024a. https://aws.amazon.com/what-is/api/.

AWS, Amazon. **What Is Data Augmentation**. [S.l.: s.n.], 2024b. https://aws.amazon.com/what-is/data-augmentation/.

AWS, Amazon. **What is IoT (Internet of Things)?** [S.l.: s.n.], 2024c. https://aws.amazon.com/what-is/iot/.

AZURE, Microsoft. **What is computer vison?** [S.l.: s.n.], 2024. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-computer-vision#object-classification.

CHUGH, Vidhi. **Precision-Recall Curve in Python Tutorial**. [S.l.: s.n.], 2023. https://www.datacamp.com/tutorial/precision-recall-curve-tutorial.

CLOUDFLARE. **What is HTTP?** [S.l.: s.n.], 2024. https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/.

DHARMEN. **Web Development**. [S.l.: s.n.], 2024. https://www.geeksforgeeks.org/web-development/.

DOCKER. **Docker Docs - Layers**. [S.l.: s.n.], 2024a. https://docs.docker.com/build/guide/layers/.

DOCKER. **What is a Container?** [S.l.: s.n.], 2024b. https://www.docker.com/resources/what-container/.

FOUNDATION, Raspberry Pi. **Raspberry Pi Foundation - About us**. [S.l.: s.n.], Sept. 2023. Available from: https://www.raspberrypi.org/about/.

GOEL, Ashwin. **Multithreading in Operating System**. [S.l.: s.n.], 2023. https://www.geeksforgeeks.org/multithreading-in-operating-system/.

GOOGLE. **Google Colaboratory**. [S.l.: s.n.], 2024a. https://colab.google/.

GOOGLE. **What is Artificial Intelligence (AI)?** [S.l.: s.n.], 2024b. https://cloud.google.com/learn/what-is-artificial-intelligence?hl=en.

HASSAN, Ehtesham; KHALIL, Yasser; AHMAD, Imtiaz. Learning feature fusion in deep learning-based object detector. **Journal of Engineering**, Hindawi Limited, v. 2020, p. 1–11, 2020.

IBM. **Socket characteristics**. [S.l.: s.n.], 2021a. https://www.ibm.com/docs/en/i/7.1?topic=programming-socket-characteristics.

IBM. **Socket programming**. [S.l.: s.n.], 2021b. https://www.ibm.com/docs/en/i/7.1?topic=communications-socket-programming.

IBM. **TCP/IP protocols**. [S.l.: s.n.], 2023. https://www.ibm.com/docs/hu/aix/7.1?topic=protocol-tcpip-protocols.

IBM. **What is networking?** [S.l.: s.n.], 2024. https://www.ibm.com/topics/networking.

INTEL. **What Is Computer Vision?** [S.l.: s.n.], 2024. https://www.intel.com.br/content/www/br/pt/internet-of-things/computer-vision/overview.html.

JACOBS, Steven. **Help configuring firewalls routers and switches and wireless solutions**. [S.l.: s.n.], 2023. https://de.fiverr.com/stevenjaco/help-configuring-firewalls-routers-and-switches-and-wireless-solutions.

JANIESCH, Christian; ZSCHECH, Patrick; HEINRICH, Kai. Machine learning and deep learning. **Electronic Markets**, Springer, v. 31, n. 3, p. 685–695, 2021.

JORDAN, Michael I; MITCHELL, Tom M. Machine learning: Trends, perspectives, and prospects. **Science**, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015.

MAHESH, Batta. Machine learning algorithms-a review. **International Journal of Science and Research (IJSR).[Internet]**, v. 9, n. 1, p. 381–386, 2020.

MATHWORKS. **What Is Object Detection?** [S.l.: s.n.], 2024. https://www.mathworks.com/discovery/object-detection.html.

MICROSOFT. **Understand TCP/IP addressing and subnetting basics**. [S.l.: s.n.], 2022. https://learn.microsoft.com/en-us/troubleshoot/windows-client/networking/tcpip-addressing-and-subnetting.

MOHAJAN, Haradhan. Third Industrial Revolution Brings Global Development. **Journal of Social Sciences and Humanities**, v. 7, n. 4, p. 239–251, Dec. 2021.

MVTEC. **The MVTec anomaly detection dataset (MVTec AD)**. [S.l.: s.n.], 2024. https://www.mvtec.com/company/research/datasets/mvtec-ad.

OPENCV. **About**. [S.l.: s.n.], Nov. 2020. Available from: https://opencv.org/about/.

PETRILLO, Antonella; DE FELICE, Fabio; CIOFFI, Raffaele; ZOMPARELLI, Federico. Fourth Industrial Revolution: Current Practices, Challenges, and Opportunities. In: **IntechOpen**. Ed. by Antonella Petrillo. [S.l.]: IntechOpen, 2018. chap. 1. DOI: 10.5772/intechopen.72304.

PRASAD, Leela. **What is Relay? How it Works? Types, Applications, Testing**. [S.l.: s.n.], 2022. https://www.electronicshub.org/what-is-relay-and-how-it-works/.

PRODEVELOPERTUTORIAL. **Linux System Programming: Creating TCP sockets**. [S.l.: s.n.], 2020. https://www.prodevelopertutorial.com/linux-system-programming-creating-tcp-sockets/.

RASPBERRY. **Physical Computing with Python**. [S.l.: s.n.], 2024a. https://projects.raspberrypi.org/en/projects/physical-computing/1.

RASPBERRY. **Raspberry Pi Documentation**. [S.l.: s.n.], 2024b. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.

RED HAT, Inc. **Chapter 10. managing services with Systemd Red Hat Enterprise**. [S.l.: s.n.], 2024. Available from: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/chap-managing_services_with_systemd.

ROSE, Karen; ELDRIDGE, Scott; CHAPIN, Lyman. The internet of things: An overview. **The internet society (ISOC)**, Reston, VA, v. 80, p. 1–50, 2015.

SHAH, Deval. **Mean Average Precision (mAP) Explained: Everything You Need to Know**. [S.l.: s.n.], 2022. https://www.v7labs.com/blog/mean-average-precision.

SINGH, Bikram Jit; SHARMA, Ashwani. Evolution of Industrial Revolutions: A Review. **International Journal of Innovative Technology and Exploring Engineering**, v. 9, n. 11, p. 66–73, 2020. DOI: 10.35940/ijitee.I7144.0991120.

SOLARWINDS. **What Is a Network Node?** [S.l.: s.n.], 2024. https://www.solarwinds.com/resources/it-glossary/network-node.

SOLAWETZ, Jacob. **What is Mean Average Precision (mAP) in Object Detection?** [S.l.: s.n.], 2020. https://blog.roboflow.com/mean-average-precision/.

TIWARI, Ashish. Chapter 2 - Supervised learning: From theory to applications. In: PANDEY, Rajiv; KHATRI, Sunil Kumar; SINGH, Neeraj kumar; VERMA, Parul (Eds.). **Artificial Intelligence and Machine Learning for EDGE Computing**. [S.l.]: Academic Press, 2022. P. 23–32. ISBN 978-0-12-824054-0. DOI: https://doi.org/10.1016/B978-0-12-824054-0.00026-5. Available from: https://www.sciencedirect.com/science/article/pii/B9780128240540000265.

TRU. **TRU COMPONENTS Pure Set Raspberry Pi® 4 B 8 GB**. [S.l.: s.n.], 2024a. https://www.conrad.de/de/p/tru-components-pure-set-raspberry-pi-4-b-8-gb-4-x-1-5-ghz-inkl-netzteil-inkl-gehaeuse-2299384.html?refresh=true.

TRU. **TRU COMPONENTS Signal tower TC-9539296**. [S.l.: s.n.], 2024b. https://www.conrad.com/en/p/tru-components-signal-tower-tc-9539296-led-red-yellow-green-1-pc-s-2384824.html.

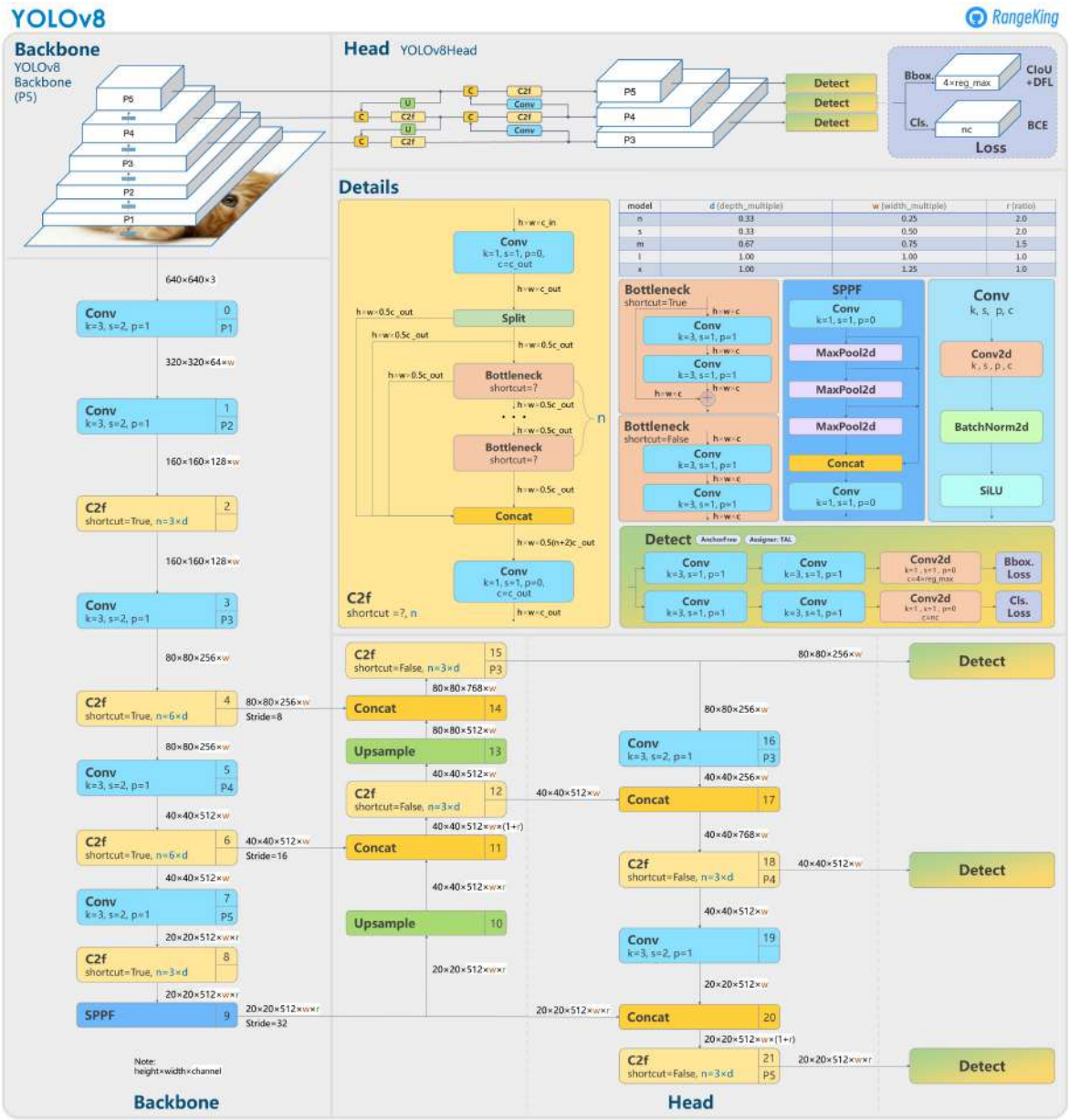
ULTRALYTICS. **Ultralytics YOLOv8 - github**. [S.l.: s.n.], 2023. https://github.com/ultralytics/ultralytics?tab=readme-ov-file.

ULTRALYTICS. **Ultralytics YOLOv8 Docs**. [S.l.: s.n.], 2024. https://docs.ultralytics.com/.

ZANERO, Stefano. Cyber-Physical Systems. **Computer**, v. 50, n. 4, p. 14–16, 2017. DOI: 10.1109/MC.2017.105.

ANNEX A –

Figure 32 – Architecture of the YOLOv8 model.



Source: (Range King, 2023).