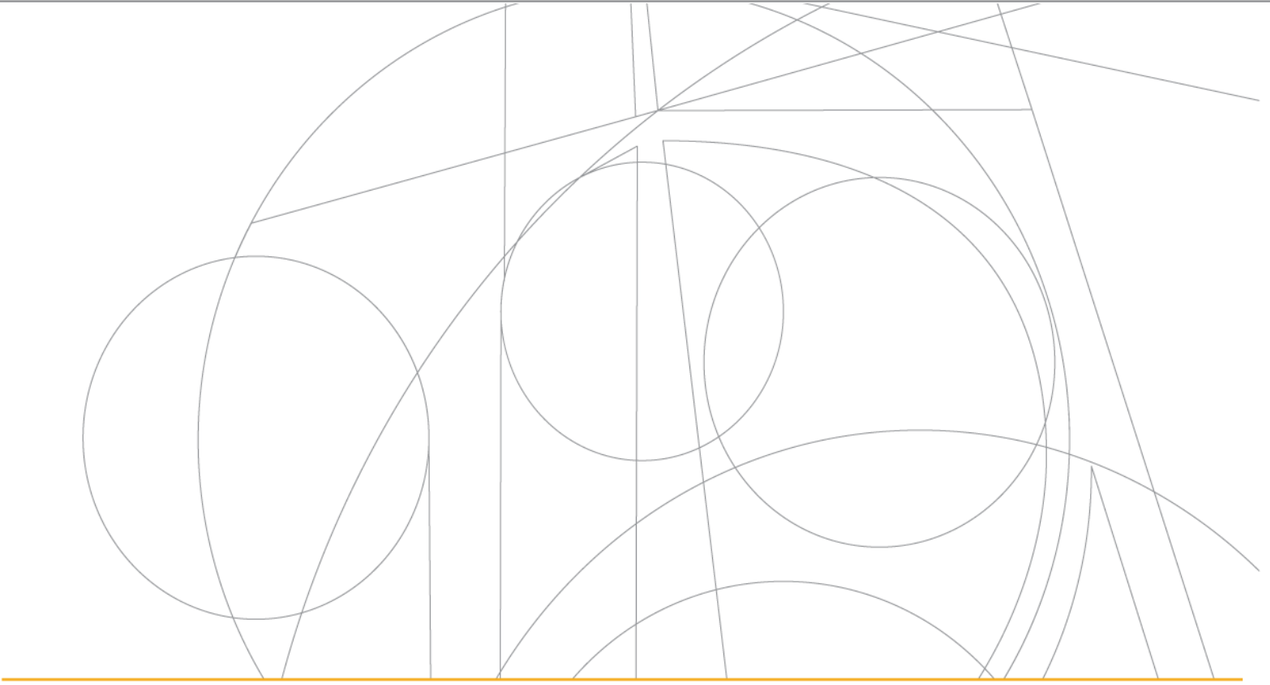
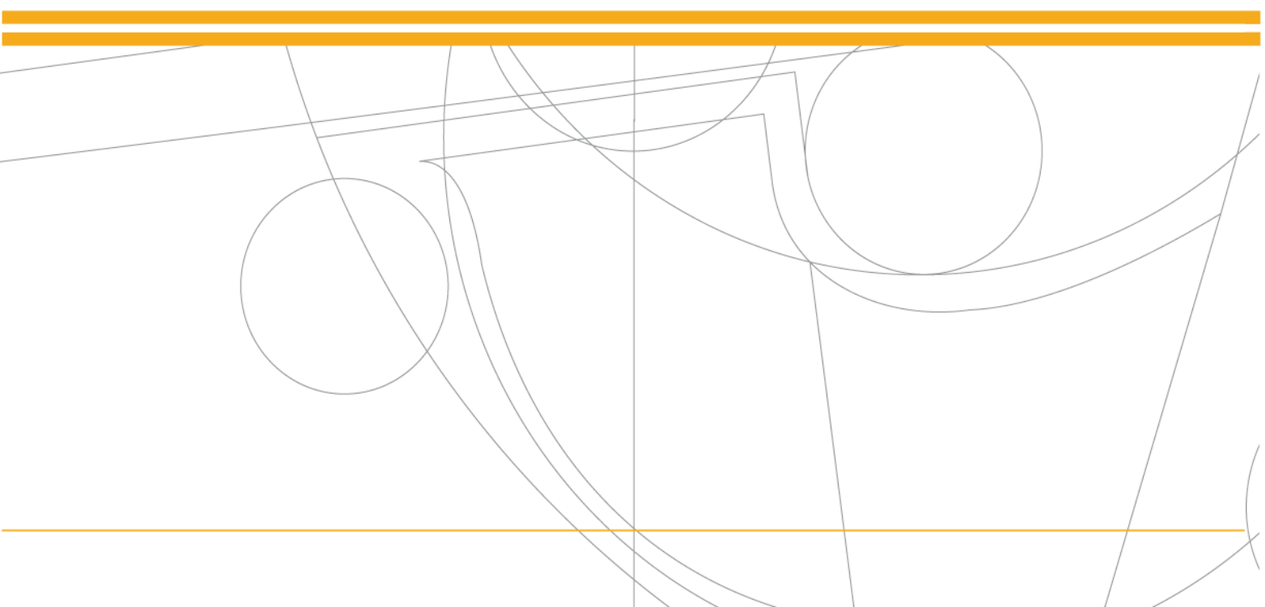


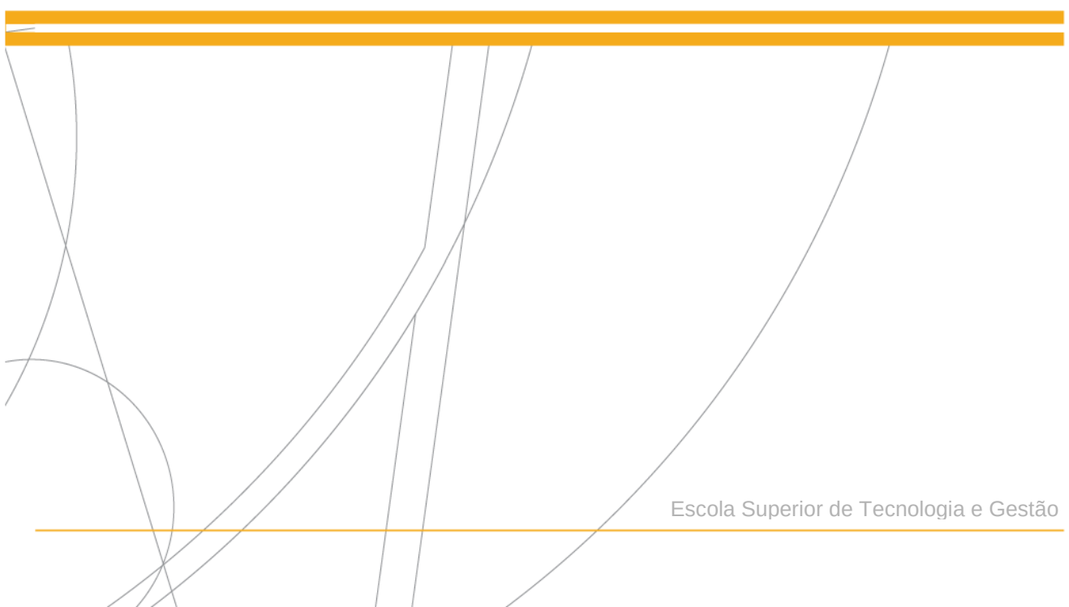
INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

EXPLORATION OF COMPUTER VISION SYSTEMS IN THE RECOGNITION OF CHARACTERISTICS IN PARTS IN NA INDUSTRIAL ENVIRONMENT

EXPLORAÇÃO DE SISTEMAS DE VISÃO POR COMPUTADOR NO RECONHECIMENTO DE CARACTETÍSTICAS NAS PEÇAS EM AMBIENTE INDUSTRIAL

João Manuel Ribeiro Rodrigues





Escola Superior de Tecnologia e Gestão

EXPLORATION OF COMPUTER VISION SYSTEMS IN THE RECOGNITION OF CHARACTERISTICS IN PARTS IN NA INDUSTRIAL ENVIRONMENT

2024

ESTG

INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

João Manuel Ribeiro Rodrigues

EXPLORATION OF COMPUTER VISION SYSTEMS IN THE RECOGNITION OF CHARACTERISTICS IN PARTS IN NA INDUSTRIAL ENVIRONMENT

EXPLORAÇÃO DE SISTEMAS DE VISÃO POR COMPUTADOR NO RECONHECIMENTO DE CARACTETÍSTICAS NAS PEÇAS EM AMBIENTE INDUSTRIAL

Nome do Curso de Mestrado
Engenharia Informática

Trabalho efectuado sob a orientação do
Professor Doutor Jorge Barbosa Ribeiro

Julho de 2024

Exploration of Computer Vision Systems in the Recognition of Characteristics in Parts in an Industrial Environment

Exploração de Sistemas de Visão por Computador
no Reconhecimento de Características
nas Peças em Ambiente Industrial

João Manuel Ribeiro Rodrigues

Master's Degree in Informatics Engineering

July, 2024

Resumo

Nos últimos anos, as áreas da *Inteligência Artificial (IA)* e do *Machine Learning (ML)* tem sido alvo de grandes melhorias, inovações técnicas e descobertas, sendo a área de visão computacional um dos grandes frutos alcançados com tais investigações e desenvolvimentos. Após a aplicação bem-sucedida de abordagens, técnicas e algoritmos associados à IA e ao ML, o potencial da sua aplicabilidade na área da visão computacional tem tido um maior destaque e sucesso em termos de aplicabilidade, principalmente em áreas como a robótica, agricultura, transporte, *self-driving* e na área industrial. Na área industrial, em particular no setor automóvel tem havido uma motivação para implementações de sistemas de visão computacional, usando IA e abordagens de ML, no sentido de substituir equipamentos de elevados custos de aquisição e de manutenção por outros com menor custo, como câmaras e scanners complementando-as com estes sistemas de visão por computador (‘*computer vision*’). Apesar de existirem no mercado fornecedores de software deste tipo de sistemas, este estudo centra-se na exploração de diferentes alternativas ao dispendioso hardware e a licenças de software que acompanham esta inovação prevista pelo âmbito global da Industria 4.0, propondo e criando um método/ferramenta com menores custos para detetar pequenos componentes e as suas características nas linhas de montagem em ambiente industrial, assim como calcular importantes métricas em ambiente real ‘chão de fábrica’, como a estimação de distância entre componentes, rotação de montagem e a distinção entre pequenos componentes, algo que não é possível de aplicar em software de licença ‘fechada’/proprietária. Dado os resultados promissores e o facto de ser baseada em licenciamento *OpenSource* foi usada e adaptada a arquitetura de visão por computador YOLO *(You Only Look Once - Real-Time Object Detection)*, complementada com um conjunto de outras ferramentas *OpenSource* que permite uma maior escalabilidade e uniformização do software no contexto das linhas de montagens e nos vários pontos e contextos de deteção de componentes.

Neste contexto, considera-se este estudo importante, uma vez que atua como porta de entrada para ferramentas de IA e visão por computador numa planta fabril de alta produção, na qual inovações deste género são por vezes postas de parte em prol de uma produção contínua e sem entraves em termos de custos de equipamentos e de manutenção, assim como em termos de eficiência de deteção dos componentes e das características associadas. Foram alcançados resultados promissores e, no final deste trabalho, é possível verificar que esta aplicação deteta corretamente vários componentes de uma determinada operação na linha de montagem, como as fitas, braçadeira ou clipes escolhidos, com uma *frame rate* elevada (comparando com outros algoritmos de visão por computador), garantindo uma deteção em tempo real no chão de fábrica.

Com a realização deste trabalho, além de um desenvolvimento personalizado da ferramenta em questão, foram alcançados pontos como medições entre componentes e deteções de modo de montagem que até hoje não eram possíveis, além de uma melhoria de tempos ciclo e de uma abertura de flexibilidade na montagem das peças, por parte dos operadores, aliado à utilização de equipamentos como câmaras e scanners com menores custos de aquisição e de manutenção.

Abstract

In recent years, the areas of Artificial Intelligence (AI) and Machine Learning (ML) have been the target of major improvements, technical innovations, and discoveries, with the area of computer vision being one of the great fruits of such research and development. After the successful application of approaches, techniques and algorithms associated with AI and ML, the potential of their applicability in the area of computer vision has been more prominent and successful in terms of applicability, especially in areas such as robotics, agriculture, transportation, self-driving and in the industrial area. In the industrial area, particularly in the automotive sector, there has been a drive to implement computer vision systems, using AI and ML approaches, in order to replace equipment with high acquisition and maintenance costs with less expensive equipment, such as cameras and scanners, complementing them with these computer vision systems. Although there are software suppliers of this type of system on the market, this study focuses on exploring different alternatives to the expensive hardware and software licenses that accompany this innovation envisaged by the global scope of Industry 4.0, proposing and creating a cheaper method/tool for detecting small components and their characteristics on assembly lines in an industrial environment, as well as calculating important metrics in a real shop floor environment, such as estimating the distance between components, assembly rotation and distinguishing between small components, something that is not possible to apply in closed license software. Given the promising results and the fact that it based under OpenSource licensing, the YOLO *(You Only Look Once - Real-Time Object Detection)* computer vision architecture was used and adapted, complemented by a set of other OpenSource tools that allow for greater scalability and standardization of the software in the context of assembly lines and in the various points and contexts of component detection.

In this context, this study is considered important, as it acts as a gateway to AI and computer vision tools in a high-production manufacturing plant, where innovations of this kind are sometimes put aside in favor of continuous and unhindered production in terms of equipment and maintenance costs, as well as in terms of the efficiency of detecting components and their associated characteristics. Promising results have been achieved and, at the end of this work, it is possible to verify that this application correctly detects various components of a given operation on the assembly line, such as the chosen tapes, clamps, or clips, with a high frame rate (compared to other computer vision algorithms), guaranteeing real-time detection on the shop floor.

With this work, in addition to the customized development of the tool in question, points have been achieved such as measurements between components and assembly mode detections that until now were not possible, in addition to an improvement in cycle times and an opening up of flexibility in the assembly of parts by operators, combined with the use of equipment such as cameras and scanners with lower acquisition and maintenance. costs.

Acknowledgments

Coming to the end of a master's degree is by no means a smooth and easy ride. It takes a great toll of dedication, perseverance, and hard work to come around all the uncertainties and difficulties that arise and reach the finish line. Because no person is ever fulfilled by itself, I would like to express my sincere gratitude to all the people who supported me throughout this journey.

To my thesis supervisors, Professor Doutor Jorge Barbosa Ribeiro, and Professor António Miguel Cruz, I would like to thank for their invaluable guidance, encouragement, and patience along this journey.

I would also like to thank those whom directly or indirectly contributed to the completion of this thesis, especially my master's degree colleagues Vasco, Marcelo, Pedro and Bruno. Coming from a non-technical, non-programming world, I saw you as role models for what I could achieve if I focused and work hard. It was a pleasure working alongside you and learning so much from you.

To my mom and to my dad, for their encouragement and support, love and affection. For the times that I was not enjoying time with both of you because I was working and pushing through this project.

To my colleagues in the Engineering Department, not only for the support and encouragement, but also for the fun and relaxing times that I passed with you along these past years. To Matheus and Esteban, for believing in the implementation of this project, for the guidance, and for aiding me with the resources to pursue this project the way I did.

Finally, I would like to thank Ana, my fiancé, for her unwavering motivation, support, and the harsh but needed words. Her kindness improved my best days, her love and support kept me going on my lowest ones. Thank you for sticking by my side, even when I felt like giving up on everything.

Thank you for helping me to achieve this goal, I will forever remember you for being part of this journey.

João Manuel Ribeiro Rodrigues

Glossary

AI – Artificial Intelligence

ANN – Artificial Neural Network

AP – Average Precision

API – Application Programmable Interface

BPMN - Process Model and Notation

DB – Data Block

DNN – Deep Neural Network

DL – Deep Learning

CNN – Convolutional Neural Network

CPU – Central Processing Unit

CUDA – Computer Unified Device Architecture

FPS – Frames per Second

GPU – Graphical Process Unit

IIoT – Industrial Internet of Things

IoT – Internet of Things

mAP – Mean Average Precision

ML – Machine Learning

OEM – Original Engineering Manufacturer

OOP – Object Oriented Programming

OP – Operation

PLC – Programmable Logic Controller

RCNN – Regional Convolution Neural Networks

SDK – Software Development Kit

SSD – Single Shot Detection

TPU – Tensor Processing Unit

YOLO – You Only Look Once

To my past self, for not believing that this day would ever come.
To my future self, as proof that hard work beats talent when talent doesn't work hard enough.
To anyone that might think that they are not enough and push through despite those thoughts.

João Manuel Ribeiro Rodrigues

Contents

List of Figures

List of Tables

Chapter 1

Introduction

First coined in 2015 in Germany by Kalus Schwab [1], the term Industry 4.0 refers to the rapid technological innovations applied to manufacturing industries in the 21st century. Popularized especially after 2016, the ideas behind Industry 4.0[1], or the 'Fourth Industrial Revolution', are based on significant changes in industrial capitalism, brought by the application of modern tools and technologies, from *Artificial Intelligence (AI)* and advanced robotics to 3D printing and nanotechnology [2, 3]. Since then, their integration into industries worldwide has leveraged and improved their development, storage control and logistics, as well as their production, assembly, and quality control processes. The integration of these types of innovative systems is possible through the combination of different Information Technology (IT) devices, network management and software programs that allow a holistic view of the entire business process, an improvement in cybersecurity systems, incorporation with Internet of Things (IoT) devices, even improvements in advanced robotics, big data, and artificial intelligence methods. Since then, companies have been gradually joining this 4th industrial revolution, seeing their processes improved and efficiency enhanced [4].

In the last couple of years, *Artificial Intelligence and Machine Learning (ML)* have revolutionized numerous industries by enabling machines to perform tasks that typically require human intelligence. Within this broader context, computer vision – a subfield of AI – focuses on enabling computers to interpret and understand visual information from the world. Recent advancements in deep learning, particularly convolutional neural networks, have significantly improved the accuracy and capabilities of computer vision systems. These systems can now recognize objects, detect anomalies, and even understand complex scenes with high precision, making them invaluable in various applications from healthcare to autonomous vehicles. The promising results of computer vision in manufacturing industries are evident in its ability to automate visual inspection processes, enhance quality control, and improve safety across multiple sectors.

The automotive sector is an important sector of industrial production, characterized by its complex manufacturing processes and high precision requirements. This industry is known to

[1] *Industry 4.0 – Definition-* https://www.sap.com/products/scm/industry-4-0/what-is-industry-4-0.html

heavily rely on advanced technologies, including expensive equipment such as high-resolution cameras, laser scanners, and other sensors, to ensure quality and conformity. The use of such equipment is crucial for tasks like component detection, measurements, and positional control. However, the high cost of these technologies can be a significant barrier to keep competitiveness on production costs in an ever-changing market. Despite this, the ability to accurately detect objects and measure distances is essential for various applications, including assembly line and process automation. The challenge lies in balancing the costs with the operational benefits to achieve efficiency and innovation in production processes.

Within the automotive and manufacturing industry, computer vision has been implemented and is found in numerous applications in it, enhancing productivity and ensuring quality control, like software applied to quality inspection, inventory management and detections in sorting systems [5, 6, 7].

That said, this project is focused on the study and implementation of a software solution built on the intersection of the Industry 4.0 principles, AI, and ML within the automotive sector, promoting a method to optimize industrial processes, enhance production efficiency, and improve production cycle times.

The following sections present the motivation and objectives as well as the methodology adopted. This work was carried out in a real-life context in a company within the automotive sector that produces and assembles airbag modules for tier 1 *Original Engineering Manufacturer (OEMs)*[2], conducted under Industry 4.0 guidelines. For confidentiality reasons, in the document we will refer only the scope of business applicability and some particularities in the automotive component production business sector activity and not refer the real designation of the company. For this reason, in the document we will refer to the company name as '*AUTOProduction*'.

1.1 Motivation and objectives

The motivation behind this work is to explore different alternatives to the expensive hardware and software licenses that accompany this innovation foreseen by the global scope of Industry 4.0, simulating in a real context and as a proof of concept a cheaper method/tool for detecting small components and their characteristics on assembly lines in an industrial environment, as well as calculating important metrics on the shop floor, such as estimating the distance between components, assembly rotation and distinguishing between small components.

Considering the historical background of Industry 4. 0 and the *AUTOProduction* business, the primary objective of this work is to build a standard application powered by AI and then implement it on the factories' shopfloor, enabling company workers (from product engineers, technicians, maintenance operators to *Information Technology (IT)* engineers and shop floor operators) to detect and check small components such as tapes, clips or plates mounted in the

[2] *OEM - https://www.sciencedirect.com/topics/engineering/original-equipment-manufacturer*

module, for example, in an easier way, in order to maintain a similar level of control as the old machine-vision system, thus opening the door to a more flexible system for positioning components, achieving substantial cost savings in the long term.

To achieve the main goal of this work, the following objectives have been defined which form the basis of the final project:

1. A comparative study of computer vision algorithms: Since the applicability of the project will be on real assembly lines, real-time detections will be necessary in order not to increase cycle times for the produced parts. It should also be accurate enough to give correct information to the workers. For this, a comparative study of the different types of algorithms that currently exist, and their history will be carried out to analyze which is the most suitable for building our application.
2. A hardware study: Always keeping a budget-friendly mentality, even when applying to a renowned multinational company, it will be necessary to carry out a budget study for the physical material that will support the application implemented on the line, from sensors (cameras or scanners) to the physical devices on which the application will run (computers or other computing devices, like Raspberry Pi or Edge Devices), in order to obtain the best results at the lowest possible cost.
3. Creating a work pipeline: From the first images taken to train an accurate vision model, annotating the objects to be identified in each image, exporting the dataset, training the model in a local environment and its iterations until it reaches the desired level of precision and efficiency and the first confidence tests, the entire work pipeline must take into account best practices and lessons learned, while being as user-friendly as possible.
4. Creation of a component position verification algorithm: More than a simple component detection is required for an industrial verification system in the automotive sector. Given the high standards required by the OEMs, it is necessary for the detected component to be verified in a specific position in the airbag module.
5. A software - PLC connection study: In every company shopfloor, the *Programmable Logic Controller (PLC)* Infrastructure stands as the true heart and brain of operations, communicating with all sectors of the plant – from the traceability system for parts produced, through cameras, sensors, safety barriers to robotic components. For the system to be viable to use, the application developed needs to have a direct communication protocol with the existing PLC infrastructure in the factory.
6. Development of a valid prototype: In order to validate the combination of software, hardware, frameworks and connections studied for the project, a prototype will be developed to run in a test environment, in a laboratory, in which we can check how it behaves and adapts to environmental variables such as camera positioning, lighting or overlapping/changing components to be detected.
7. Simulation and performance tests: To implement the developed solution on *AUTOProduction's* shop floor, the application will also be tested in a real environment, on an active assembly line in the factory, where the 'old' system for checking

components (camera and photocells) will be replaced by the developed solution, carrying out the necessary observations, adjustments and monitoring of the vision model and the application that encapsules it.

1.2 Methodology and Schedule

Each area of study has an ideal working and research methodology, depending on the topic, area, and questions it aims to answer. In this project, the methodology chosen was *Design Science Research (DSR)*, which is based on the research and development of solutions, with well-defined processes, to increase the value of the companies involved.

DSR is a relatively modern research approach with origins in the engineering field and other applied sciences. It emphasizes the creation of artefacts with the main objective of improving companies' organizational performance by solving specific problems. This methodology enables a direct or indirect increase in profits by focusing on solving problems or improving existing solutions [8].

According to [9], DSR requires the development of a new artifact adapted to solve a domain-specific problem. The construction of the artifact is considered complete when it meets the requirements and constraints of the problem it is intended to solve. DSR represents a problem-solving process in which knowledge, understanding of the problem, and the design of the solution are acquired during the construction and application of the artifact.

DSR not only addresses previously unsolved problems in a unique and innovative way, but also improves the efficiency and effectiveness of solutions to previously addressed issues. DSR is identified by six distinct activities, like shown in figure 1, that can take place sequentially or in a different order:

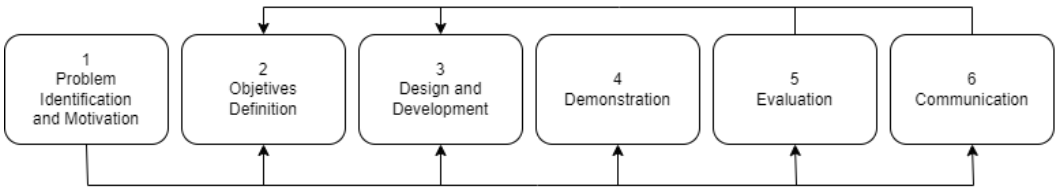


Figure 1 - Design Science Research activities (extracted from [9])

1. Identifying and motivation for the problem, gaining a good understanding of existing means and state of the art to support the importance of the solution. In this work, it is defined in the Motivation subchapter and 'State of the Art' chapter.
2. Defining the objectives of the solution, determining what is possible and feasible for the future of the project, grasping on the most likely solutions to the current problem. In this work, it is defined in 'Motivation & objectives' subchapter and Problem Analysis chapter.
3. Design and development, determining the architecture and functionality of the artifacts

and their subsequent creation. In this work it is described under ‘Design & Implementation’ chapter.

4. Demonstration of how the solution created solves the problem identified, through experimentation, simulation, or case studies. In this work, this point is described at the end of the ‘Implementation’ subchapter.
5. Evaluation of the solution, checking how the artifact performs in solving the problem, to validate it delivering on the expected results, considering the objectives previously established. This point is described in ‘Results & Evaluation’ chapter.
6. Communication of the solution, sharing the problem and solution developed and its results. In this work, the communication was achieved by the publishment of a paper relating the work done, mentioned at the conclusion.

In this context through the simulation applicability of this work, we describe in the next sections of the document the phases of the design science research methodology described above and, in the conclusion, we resume the different DSR activities.

In order to achieve the goals of the project, and by the fact of the time to implement the project, from August of 2021 to July 2024, the general phases and schedule of the work activities have been scheduled and implemented as follows:

Phase	2021				2022				2023				2024			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
State ot the art Study			X	X	X	X	X									
Phase 1 - Proof of Concept						X	X									
Phase 2 - Programming Environment Implementation								X								
Phase 3 - Smart Area Implementation and Distance Estimation										X	X					
Phase 4 - Real world's simulation													X	X		
Thesis Writing				X		X	X		X		X	X		X		

Figure 2 - Activities Schedule

In this project, the SCRUM Agile methodology was adopted to ensure a flexible and iterative approach to the project development. SCRUM emphasizes on collaboration, accountability, and continuous improvement by breaking down work into smaller, manageable increments called sprints [10], lasting 1 month. At the beginning of each sprint, goals were defined, and tasks were prioritized according to the project’s timeline and requirements. Daily self-check-ins were performed to track progress and identify any obstacles. After each sprint, a personal review of the results was carried out to evaluate the progress made and adjust the next sprint’s focus accordingly. This iterative process allowed for flexibility and continuous refinement of the solution, ensuring that the project stayed on track and met its objectives within the defined schedule.

During this period, as the tasks were completed, a summary was made of the studies and conclusions drawn which materialized in the writing of this master's thesis.

1.3 Dissertation structure

This master thesis is structured as follows: The first, and current chapter, presents some introductory notes, with contextualization and primary motivation for this work. Then the main objectives are presented along with the methodological approach used in this work.

Chapter 2 presents the State of the Art starting by a brief presentation of computer vision technology and its main applicability in manufacturing industries. Then, some works and tasks implementing machine learning applied to the manufacturing industry of interest to this project are presented. At the end of this chapter, it is presented the evolution and main architecture explanation of the main computer vision algorithm to use in this project.

Chapter 3 presents some theoretical background, concepts and definitions that are needed throughout the research, presenting the mains ideas that the project revolves around, as well tools and technologies used during this work's development and implementation.

Chapter 4 presents the analysis of the problem as well as the requirements and use case scenario to implement the solution.

Chapter 5 presents the design and implementation of the project aims to solve at *AUTOProduction's* shopfloor and its requirements, documenting the desired components to detect in the assembly line, the system architecture, as well the entire development process from the creation of the vision model to the final testing, implementation and monitoring of the application on a real assembly line.

Finally, in chapter 6 and 7 stands the results of the solution implementation in the assembly lines, its analysis and limitations are drawn, and some conclusions and future work are presented.

A list of references and an appendix are also included at the end of the document.

Chapter 2

State of the Art

This chapter presents a literature review of the most important topics on which this project is revolving on, such as an overview of the field of computer vision and its evolution and its most recent and important advances, as well as its applicability in manufacturing industries. Supported by AI techniques, data augmentation techniques (important for training a vision model) and applications in the sector are also presented, as well as measurement applications between objects that have combined with artificial intelligence to achieve their purpose. Finally, we review the architecture of the chosen vision algorithm, the YOLO algorithm, as well as its history between versions over the years. At the end, an alternative to programming between Python and PLC applications is also presented, in order to have coverage and applicability in broad industrial environments.

2.1 An overview on computer vision

Computer Vision is a branch of software development that simulates human visual systems using cameras and computers to process images [11], involving the training of learning models to make predictions or detections about image content. Computer vision aims to give computers the ability to 'see' like humans, improving the accuracy of image recognition and object detection through *Deep Learning (DL)* techniques. Nowadays, applications based on computer vision extend across various sectors, leveraging them through their ability to interpret and make inferences about visual data such as images or video. Some examples of applicability are:

- In medicine, in dermatology for example, computer vision applications are used to identify skin conditions and warts for an early detection of skin cancer [12] or in the automatic diagnosis of medical images such as X-rays, or MRI [13], among others.
- In the automotive sector, where computer vision is becoming increasingly crucial in autonomous driving functions, such as detecting other vehicles, road surfaces and tracking pedestrians for safer driving [14, 15], or for automatic license plate detection in automatic parking meter management systems and smart parking lots.
- In retail, for automatic inventory management tasks using cameras to reduce stock

shortages and optimize supply chain processes, as well as applications in the payment areas, through checkout-free stores [16].

- In manufacturing, computer vision applications are used in quality control tasks (such as defect detection in production processes on the shop floor) [17] and autonomous pick and place through the synergy between robots and automatic control of embedded cameras, among others.

Although notoriously well-known today, the subject of computer vision has been the subject of study and advances since before the turn of the century, with the creation of the still well-known Canny Edge Detector in the 1980s - a multi-stage algorithm (including Gaussian filters, gradient intensity finders, gradient thresholding and non-maximum suppression step) that allows the detection of edges in objects from an image provided to it [18]. However, only in recent years have 'bigger' advances been made in the area, which are usually categorized into three distinct eras:

Before 2014, in addition to the previously mentioned Canny Edge Detector, traditional object detection techniques were also developed, such as the Viola-Jones Detector (2001), Hog Detector (2006) and *Deformable Parts Model (DPM)* – Detector (2008) [19].

After 2014, with the advent of AI, algorithms began to be built that were truly inspired by the neural networks developed until then, and among them another distinction was made regarding them, between Single and Dual Stage. In a Dual Stage Detector algorithm, the approximate regions of the object are proposed by neural network features, which are then applied again to classify and delineate the bounding box. They achieve greater detection efficiency, but are consistently slower, due to the multiple inferences between steps. Several two-stage algorithms use convolutional neural networks, of which two can be highlighted:

1. *Region-based Convolutional Neural Networks (R-CNN)* [20] are pioneering models that have implemented deep models for object detection. First, regions of the image are selected. Then labels and bounding boxes are created according to the classes assigned to the program. During the first phase, the image is divided into around two thousand regions so that the *Convolutional Neural Networks (CNN)* can be applied to each region, respectively. The size of the region is calculated, and the correct region is inserted into the neural network. It can be inferred that a model as detailed as this has time constraints, with the training time being significantly longer compared to Sigle Stage Detectors, as YOLO, since it classifies and creates the bounding boxes individually one step at a time, and the neural network is applied to them singularly [21].
2. Mask R-CNN [21] is an advancement of Fast R-CNN, with the difference that Mask R-CNN has added a branch that allows it to additionally predict the object's mask, in parallel with the step that predicts the bounding box. It's easy to train, and it infers at 5 *Frames Per Second (FPS)*.

Single Stage Detectors, on the other hand, predict the bounding box without a region

proposal step, a process that considerably consumes less time and resources while being better implemented in real-time applications. However, it is not as effective at recognizing irregularly shaped objects or groups of small objects [19]. Within this group of detectors, two of the most popular, which have the greatest advantage due to their fast detection and simple implementation, are the following:

1. *Single-Shot Detection (SSD)*, a single-phase detector that allows multiple classes to be detected, using a single deep learning network, evaluating the result of the bounding box spaces into a set of boxes at different resolutions and scales, according to their location on the map. The detector generates an evaluation for each object present in the image and adjusts the bounding box to its size. It is easy to train and integrate into software systems that require detection components. Compared to other one-stage methods, it is much more accurate, even with small input images [22].
2. *You Only Look Once (YOLO)*, used in several commercial products from companies that use computer vision. The first detector was launched in 2016 (with its architecture being much faster than any predecessor), and it has gone on to gain more updated versions and innovations such as mosaic data enhancement, self-adversarial training, and cross-batch normalization [23].

Today, the best algorithm in terms of *Average Precision (AP)* are YOLOv8 and YOLOv9, standing as one of the fastest algorithms, with the most FPS. In addition to the 'native' implementations of detection algorithms, more and more models have been changed for Edge AI (where computing is not carried out by a local machine, but by hosting in the cloud), making it possible to turn these techniques even more scalable and with increasingly lightweight algorithms and optimizations for edge versions.

Within the two types of algorithms mentioned above, Single Stage Detectors, unlike Dual Stage Detectors (which often involve two or more separate phases of proposing and classifying regions), simplify this process into a single pass through the neural network. This approach gives them a considerable advantage over Dual Stage algorithms in the following respects:

- They become more efficient by combining region proposal and classification into a unified structure. SSDs eliminate the need for complex *Region Proposal Networks (RPN)* or external region proposal mechanisms, thus reducing computational overhead;
- Real-time performance given by the inherent simplicity of single-phase detectors blends itself well to real-time applications such as autonomous driving, video surveillance and robotics, where fast processing of images or video streams is crucial for proper operations.
- Simplicity conferred by a single-pass architecture. Single-phase detectors are conceptually simpler to implement and to optimize compared to dual-phase detectors, which often involve complex multi-phase pipelines.

2.2 Computer vision on manufacturing industries

Like previously declared, within the field of artificial intelligence, the general goal of computer vision is to teach machines how to perceive, interpret, and comprehend visual data, making use of techniques for gathering, handling, and evaluating pictures or videos in order to mimic and simulate human vision and carry out practical functions. Along with self-driving cars, checkout-free aisles, clinical imaging and early cancer diagnostic and detection, computer vision can help with critical industry processes across multiple fields [5, 6, 7], like:

- Quality Inspections: The quality control cycle is a complicated task, due to the high reliance placed on the visual understanding of the tester and adapting to constant product changes. With AI, this problem is circumvented, since a trained model can distinguish good from bad products on a production line at a speed unmatched by 'human' techniques;
- Stock Management: Flows of materials that are consumed, removed and moved every day are difficult to manage and can negatively influence the company in the long term if corrective measures are not taken. Object detection automation systems can count how much material there is in storage and locate 'lost' or poorly stored material for the rest of the company with a wide network of cameras installed in warehouses;
- Detections in the Sorting Process: Sorting items (by size, color or profile) is an extremely laborious task and is accompanied by human error (even with collaborative robots installed, the process is not completely efficient). With Object Detection and Tracking algorithms implemented, objects are classified by parameters listed and selected beforehand, substantially reducing anomalies in categorization, thus making production lines more efficient;
- Assembly Line Processes: On assembly lines that are already fully automated (large automotive OEMs, for example), each movement of robotic arms is triggered by a script. However, any unevenness in the assembled product (like a vehicle) can cause inconveniences and problems, since the assembly coordinates of those robots become misaligned. It would be important to give the machines more flexibility by teaching them to locate parts and move to different locations or components in a more dynamic and correct way.

DL methods, such machine learning, and sophisticated neural networks, have completely changed the computer vision subfield in automotive sector by allowing computers to acquire, process, and interpret visual data [24]. Nowadays, image sensors, mobile CPUs, operating systems, and application development have been leveraged from advances in computer vision.

Directed applied to manufacturing industries, computer vision projects using the YOLO computer vision framework, have been developed and implemented in several business, as is the case of software applications to detect defect on the surface of industrial parts or products [25, 26], proving to be a useful asset at a very low price for the level of control that the YOLO

platform offers to the control process. Other use cases for computer vision algorithms in manufacturing industries pass from feature detection, recognition, segmentation, and three-dimensional modeling on produced parts [27], for example, while other system applications can detect operators' accidents and send alert to managers and staff, ensuring safety of employees in case of an emergency [28].

In these scenarios, within manufacturing industries, computer vision systems can also aid in production and component assembly, defect detection or regular monitoring of special equipment used for production (machinery or EPIs) [29]. All in all, the integration of computer vision with manufacturing operations on the factories' shop floor maximizes the potential benefits of implementing Industry 4.0 initiatives, since it enhances productivity, safety, and quality control in manufacturing processes.

2.3 Data augmentation through synthetic data

Collecting, analyzing, and identifying images for a good and balanced training library (or dataset) for machine vision algorithms is usually a very complicated, time-consuming, and costly job. This task becomes even more challenging under the real conditions that factory shopfloor has to offer. Data augmentation is a technique that is mainly used to generate new data (images and labels, in the case of computer vision applications) from existing ones, and thus artificially increase the size of a dataset with less effort. In this area, it is commonly used to improve the performance of CNNs [30] using multiple methods like image rotating, cropping, flipping as well adjusting hue, saturation, exposure or noise settings [31, 32]. In 2021, Faltings et al. [33] proposed a model to generate training data through synthetic methods to read numerical digits in cast iron billets, proving that is possible to use up to 75% of synthetically created data, significantly compensating in terms of time spent collecting and annotating images – having the same result as models completely made up of real data. On top of that, the use of computer-enhanced data allowed for addressing rare events in production, like character errors or component deformations. Critical and sporadic errors in production, which are challenging for 'normal' datasets due to low occurrence frequency, were synthetically enhanced, enriching the dataset for improved accuracy.

On the other hand, by increasing the volume and variety of training data, data augmentation helps to better generalize and increase performance for unknown data [33], while also reducing overfitting by increasing the number of training data included in the machine learning model from the beginning [34]. In some cases, like the ones appointed in [35] shown that a method including of mixing-up of synthetic and real images, along with random erasing of data have shown improvements in object detection and image classification tasks.

2.4 Distance measurement techniques

Distance measurement through cameras vary between active and passive methods and techniques to determine the distance to objects or scenes captured by the camera [36]. Active distance measurement involves using a local light source to project a beam onto a target, capturing the reflected light with a sensor. On the other hand, passive distance measurement methods rely solely on the image capture, on angle measurements and position without the use of a light source, such as parallax or triangulation techniques [36, 37]. Both techniques are broadly used in long distance measurements, however, for short distance measurements, active methods prove to be more reliable on the use of light sensors to predict distances between the object and camera.

When it comes to measuring distances, through a single-focal camera, the 'simpler' AI algorithms that exist today are still not able to have a clear perception of depth of focus or point of view, since they are dependent on additional proximity sensors to do these calculations. Vajgl et at. [38] proposed a scheme by which a SSD algorithm, YOLO, can be improved in order to predict the absolute distance between objects, using only the information received by a mono focal camera. As it was possible to integrate this scheme within the original architecture of the created algorithm, the prediction vectors were extended – sharing the main weights of the model with the bounding box regressor, updating the original loss function by the part responsible for estimating distances.

The proposed new algorithm, Dist-YOLO, was designed using two different ways of treating distances (class-agnostic and class-aware), proving that agnostic classes create smaller predictive vectors than class-aware, as well as achieving better results within a satisfactory time frame. The authors also demonstrate that the different object detection and distance measurement tasks work in synergy, resulting in an increase in the accuracy of the original bounding box creation function, as well as showing that the scheme produces an average relative error of only 11% in distances up to 150 meters, which makes the algorithm a highly competitive solution with existing approaches, since it maintains the same FPS rate as the original YOLO, at 45 images per second when inferring in a graphical processing unit (GPU).

Although there are several computer vision implementations to calculate distances between large objects, proving its efficiency and promising results, after analyzing the state of the art and the success and the promising results of the YOLO framework, it was decided to explore and extend on this framework with the creation of more functionalities for this work case.

In the following subchapter it will be presented a brief history of this OpenSource framework through its various versions up until today, as well as its architecture.

2.5 YOLO framework

YOLO (or You Only Look Once) stands out in the world of artificial computer vision algorithms as a structure and framework that balances detection speed and accuracy incredibly well [39], enabling real-time detections in various areas of applicability, covering segments such as autonomous driving, robotics, or manufacturing. Since its creation, it has undergone several cycles, updates, and iterations (each one building on the discoveries of the previous version and considering its limitations, gradually improving its performance) [40].

2.5.1 Evolution

Created by Joseph Redmon and published in 2016 [41], the first version of YOLO presented for the first time an end-to-end approach to object detection, in which it became possible to aggregate detections in the image in a single pass through the neural network, unlike other algorithms that required either running a classifier hundreds of times through the chosen images, or more advanced methods of dividing this task into two phases (one to detect possible regions of interest and the other to apply the classifier to those regions [42]). Until then, object detection was seen as a classification task of separating bounding boxes. However, YOLO was built considering solving the problem recursively, associating the probability of each detection using only a single CNN pass [29, 41].

In 2017 [41], the YOLOv2 model was published by Redmon, together with Ali Farhadi. First categorized as YOLO9000 (for being able to detect up to 9000 different categories of objects), it had several improvements over the original, making it faster and more robust. One of the biggest improvements was the use of batch normalization in all the convolutional layers of the network, which makes it easier to train the model, increasing its efficiency during learning, reducing overfitting, and improving stability and performance. The second version was also able to handle higher-resolution images [43], making it better at finding smaller objects in an image (something that is practically impossible in a lower-resolution image, and therefore blurrier, image). Also, an innovation brought from Faster R-CNN implementations was the use of anchor boxes (becoming the norm in later versions), helping the algorithm to predict the size and shape of objects more accurately, even if they have more irregular shapes.

In 2019, Redmon and Ali [44] published the third version of YOLO, with even more innovations and a larger model architecture, to keep up with existing technology and maintain the frequency of real-time detections [40]. In terms of structure, the authors used a more efficient backbone architecture called Darknet-53 (due to the 53 convolutional layers) [39], contributing to faster and more accurate detections, and making it possible to detect objects at three different scales. However, with the implementation of the Darknet backbone, there was a slight increase in the computational requirements needed to run the algorithm, compared to previous models, so it was necessary to consider the hardware needs for the requirements of the application to be created and implemented.

In 2020, and for the first time without its original authors, the “official” version of YOLOv4 was published [45]. Received with some controversy and uncertainty for leaving out Redmon and Ali, the fourth version of YOLO proved to be faithful to the principles and philosophy of the versions that preceded it. Real-time detections, open source, single stage detection and the darknet-based framework [40], along with more innovations and improvements became accepted by the community as the official version. For the first time, YOLOv4 tried to find balance in its changes by categorizing bag-of-freebies and bag-of-specials. Bag-of-freebies refers to the changes made by users to the model's training methods and the increase its computational cost, but not its inference time (data augmentation being the most common modified training method), while Bag-of-Specials increases the inference time of the training, but substantially increases its accuracy. It’s also important to note that YOLOv4 has been optimized for greater and more efficient use of limited hardware resources and was, at the time of its launch, one of the best performing object detection models, being the most suitable for deployment on various types of hardware, including edge devices. After the release of version YOLOv4, Joseph Redmond announced that he was abandoning research into YOLO and the computer vision field altogether, for fear of his studies and work being abused in military operations and applications, leaving the continuation of research into the algorithm to the community of developers who have since continued to develop on top of the discoveries and work he had done [39].

Also in 2020, just one month after the YOLOv4 paper, the fifth version of YOLO, YOLOv5, was launched by Glen Jocher, founder of Ultralytics [40, 46]. The name was chosen to avoid overlap with the already released YOLOv4, however, both authors used state-of-the-art innovations in computer vision work, making the architecture of YOLOv5 quite similar to the one YOLOv4 was built on, and consequently it did not bring the necessary innovations to ‘deserve’ the title of the fifth version in the eyes of many. This was made worse by the fact that the algorithm was not accompanied by a published paper [39], unlike the four versions before it. However, what YOLOv5 gains over its predecessor is its ease of use. Written in Python (instead of C, as was the case until then), YOLOv5 is easy to install and integrate into IoT devices, combined with the fact that it is supported by the largest community of Pytorch users, which guarantees it more contributions and potential for future growth compared to the smaller Darknet community. In this way, YOLOv5 is an algorithm created and designed to be more user-friendly, with simple integration and implementation, quick installation, and much better integration with the Python ecosystem. Since its implementation, version five is also known for its flexibility and fast training times, especially for customizable datasets (necessary for application custom model creators), and it is also possible to easily change the training architecture or its settings with various hyperparameters. In addition to introducing support for instance segmentation and classification models [46], YOLOv5 was natively released with 5 different models (nano, small, medium, large, extra), with different oscillations between speed and detection accuracy, making it ideal for making a more informed choice, taking into account the hardware of the devices that will run it. For this project, YOLOv5 was chosen as the model and vision algorithm.

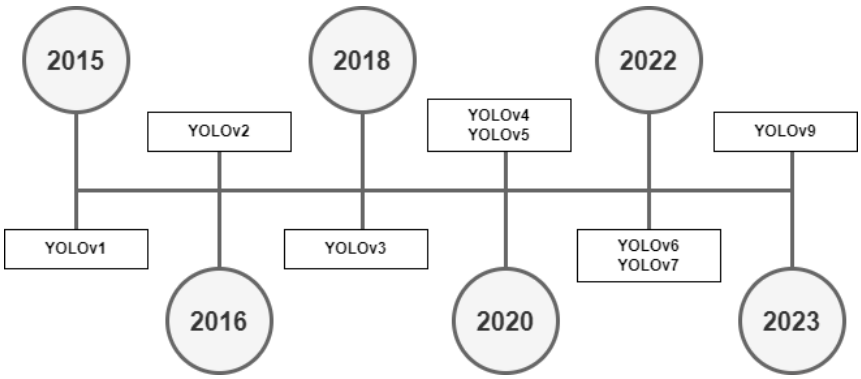


Figure 3 - YOLO Timeline Evolution

YOLOv6 was published in 2022 [47] by the Meituan Vision AI Department. Unlike different YOLO architectures that rely on anchor-based techniques for detecting objects in the image, YOLOv6 adopts an anchor-free approach, which results in an architecture that is 50% faster than most anchor-based object detectors [47]. Other significant improvements are practical enhancements such as an increase in the number of training epochs available, quantization and knowledge distillation, which makes this model very suitable for real-time industrial applications. Quantization reduces the memory and computational requirements of the trained model without necessarily compromising its performance (converting weights and activations of the neural network from high-precision floating point numbers, such as 32-bits, to lower-precision numbers, such as 8-bit integers), which reduces memory usage and speeds up the computation required and is best used in deployments with limited resources [48]. On the other hand, knowledge distillation is used to further improve the accuracy of models without incurring significant computational costs. In this process, a 'teacher' model trains a 'student' model, with the predictions of the teacher model serving as soft labels, as well as the actual labels, for training the student model. In this way, YOLOv6 uses self-distillation for training, where the student model also serves as the teacher, with teacher model being trained beforehand. [49].

Also in 2022, YOLOv7 was launched by the same authors as YOLOv4 [50] and, at the time of its publication, it surpassed all known detectors in terms of speed and accuracy in the 5 to 160 FPS range. Like YOLOv4, it was trained using the Microsoft Common Objects in Context (MS COCO) dataset, without previously trained backbones (which makes it more adaptable to smaller datasets, learning specifically detailed features for the data received, rather than relying on pre-existing weights from a different dataset), proposing some changes to the architecture and bag-of-freebies methods that improved its accuracy without affecting inference time in training time. For this effect, YOLOv7 uses an auxiliary head to train the middle layers, while the main head is mainly responsible for the model's final output. This dual-head architecture combined with the label assister mechanism, which assigns soft labels, makes the model learn from incoming data more efficiently [40]. This label assister is nothing more than an evolution of the knowledge distillation proposal in YOLOv6, in which, instead of only taking

into account the actual labels of the collected data, it also takes into account the model's predictions [51], creating soft labels, which are more adaptable and that lead to better training and overall performance. It is also important to note that YOLOv7 was the first in its long series of detection models to incorporate a pose estimation model, making it notable in a field where there was still a scarcity of real-time models for this purpose.

In 2023, the Ultralytics team published and launched YOLOv8 [5], which, like YOLOv5, was accompanied by different types of scaled versions (nano, small, medium, large and extra-large), supporting multiple tasks such as object detection, segmentation, pose estimation, tracking and classification. Using an anchor-free methodology, the YOLOv8 architecture has seen its object generalization improved during model training, being able to detect the object's midpoint (thus reducing the number of bounding boxes), which helps in later non-maximum suppression tasks by discarding incorrect predictions [40].

In 2024, the last available version of YOLO, the YOLOv9 was released [52], build on its predecessors that came before. This new model features two key innovations: Programmable Gradient Information (PGI) and the Generalized Efficient Layer Aggregation Network (GELAN). PGI addresses the issue of information loss through neural network layers by ensuring complete input information is available for target tasks, thereby enhancing the reliability of gradient information for network weight updates. GELAN, a lightweight network architecture based on gradient path planning, optimizes parameter utilization and, when combined with PGI, delivers superior performance in object detection tasks, outperforming other state-of-the-art methods both in efficiency and accuracy [52].

2.5.2 Architecture

Although there are currently more recent versions of the YOLO algorithm, for this work we opted as base of our application its 5th version, YOLOv5. Even though since the creation and implementation of YOLOv5, all subsequent models created have optimized the choice between speed and accuracy in detections [40] (offering different scales of models that adapt to different physical hardware requirements and specific applications), YOLOv5 was the one that proved to be the most 'stable' for creating a solo application, as well as having the most online documentation available on implemented projects.

Using convolutional layers, which together create a complete convolutional neural network, the architecture of the YOLOv5 algorithm can be divided into three main parts: the feature extractor, the detector, and a classifier [53], incorporated into the backbone, neck, and head of its main architecture (shown in figure 3). Between these three main parts the data information flows through a set of key points, detailed after figure 3, which illustrates the YOLOv5 main architecture, below.

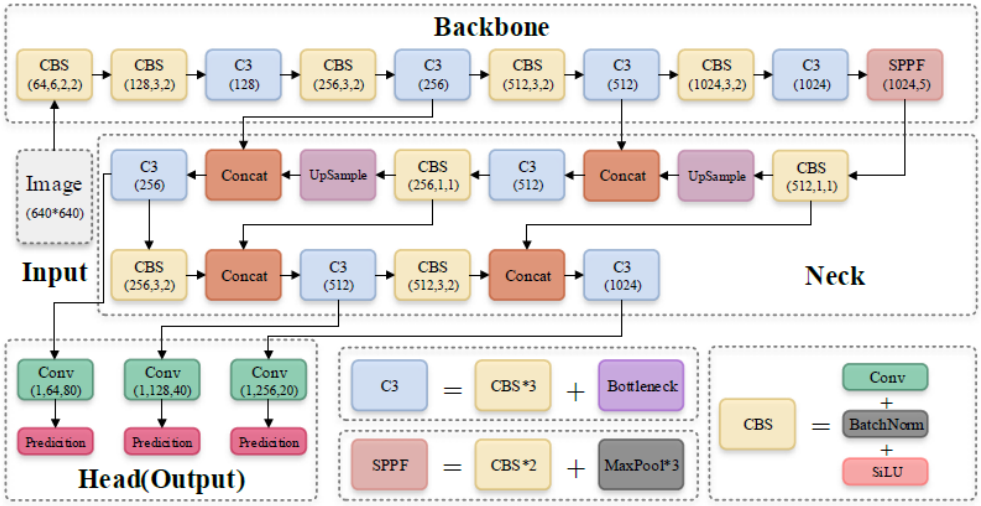


Figure 4 - YOLOv5 Architecture (extracted from [54])

The YOLOv5 Architecture is structured in two main parts, as illustrated in figure 4:

1. Input Image: Just like any other neural network applied to image detection systems, the first step involves acquiring an image (either a still image from a live camera or a video feed, capturing each frame individually) [14], typically represented as a three-dimensional array with height, width (in pixels) and three distinct RGB color channels (or just one channel, if the image is in grayscale) [17].
2. Backbone: According to [4, 14], as YOLOv5 offers different types of models for different types of tasks, from YOLOv5n (for nano models) toYOLOv5l (large models), the structure of each of the algorithm's models varies both in width and depth [53], as well as in overall performance [55].

The first key component of YOLOv5 is its backbone network, which is primarily responsible for feature extraction [14]. Depending on the model to be used, YOLO allows greater flexibility in the choice of backbone architecture, with the most popular choices being EfficientNet [56], CSPDarknet53 [54, 57] or MobileNetV3 [58], with this choice depending entirely on the balance between the size of the model, the speed of the model and the accuracy required for the specific application.

In this way, the image input passes through the backbone layers, which are typically made up of convolutional layers, pooling layers and activation functions. These layers have the function of transforming the raw image of pixels into a set of feature maps (after a series of down-samplings [59]), and the deeper the backbone layer, the more abstract and complex the features given by the extractor will be [14]. In the case of the model chosen – YOLOv5s (small) – we can expect a relatively simple feature map compared to heavier models.

3. Feature Extraction: Obtained mostly from the backbone of the structure [56], the feature maps obtained and returned by the extractor represent the visual information

extracted by the image imputed into the model, capturing textures, well-defined patterns and information about objects present in the image [53]. Features extracted from different layers of the algorithm's backbone end up having varying levels of information, allowing the network to detect objects at multiple different scales [60].

4. Detection Head: The detection head is another crucial part of YOLOv5 as a model, since it is responsible for processing the feature maps captured in 3) and making predictions based on their results, with information such as height and width, object class, confidence and coordinates [54]. The detection head typically consists of a series of convolutional layers followed by a series of output layers. The convolutional layers in the detection head help to define and enhance the features extracted from the backbone, aggregating the information between different scales. The depth of the detection head can be adjusted to control the trade-off between speed and accuracy [4]
5. Bounding Box Prediction: YOLOv5 works by dividing the input image into a grid of cells, with the number of cells in the grid depending directly on the original size of the image and the percentage of down-sampling applied to the backbone layers [59]. For each cell, the detection head predicts a fixed number of bounding boxes (usually 3 to 5), and these bounding boxes are parameterized with four different values [53] – x and y coordinates, width, and height, where x and y represent the center of the predicted bounding box [4], and the height and width redefine its shape according to the image [39].
6. Object Score: For each predicted bounding box, YOLO calculates an 'objective value' that indicates the probability of that box containing an object [39]. This value is obtained using binary classification, evaluating this detection task as a classification of the presence of an object against the absence of an object for each of the bounding boxes generated. This generated value helps in future filtering of false negatives, focusing on the bounding boxes most likely to contain objects [39, 55].
7. Class Prediction: In addition, as well as valuing the object itself, YOLO also predicts the class probability for each existing bounding box [4], with the number of classes predicted depending on the training dataset to which the model was subjected (for each n object classes, the model would predict n dimensional vectors representing the probability of each class).
8. *Non-Maximum Suppression (NMS)*: After obtaining the bounding box predictions, object scores and class probabilities for each bounding box, YOLO performs a non-maximum suppression task that is applied separately to each class to remove duplicate or overlapping bounding boxes [39]. During this step, only the bounding boxes with the highest degree of confidence (according to their class probability and object value) are kept for each object predicted in the image, resulting in an output for final detection [22].
9. Output: The final output of YOLO is a list of the objects detected in the image together with their bounding boxes and respective class labels, given as tensors. Each record in this list contains the object's class, coordinates (x, y, w, h) and degree of confidence [17, 39], and these results can be applied in detention, tracking or further analysis

applications.

2.5.3 Chosen YOLO version

As previously mentioned, the version of YOLO chosen for this project was YOLOv5. Although there are more recent versions with superior benchmarks, the documentation for the YOLOv5 project (which was the first model to be programmed in Python), together with the troubleshooting documentation, implementations and questions in forums and articles on the Internet, continues to be fed by a large community of developers, which guarantees longer-term support than the versions that followed.

Another reason for choosing the fifth version of the framework is that, not only was it the only stable version at the start of the project (as YOLOv6 and YOLOv7 were only a few months old at the time), but it is also one of the models that performs best on low-budget or lower-capacity hardware (something that directly influences the hardware on production lines). This is why the YOLOv5 was chosen and continued to be used, as it performs best under the limitations of the existing hardware in the industrial environment.

In addition to these aspects, YOLOv5 is capable of being implemented in industrial applications as is the case of software applications to detect defect on the surface of industrial parts or products [25, 26] or for feature detection, recognition, segmentation, and three-dimensional modeling on produced parts [27], making it the perfect fit for a barrier-entry AI software in the company.

2.6 Programmable Logic Controller (PLC) integration

Often recalled as the 'heart' of operations, PLCs are digital operating electronic devices, used in various industries and processes [61], being indispensable on modern machine and industrial automation, ensuring precise control, adaptability, and real-time responsiveness. They function by storing and executing specific instructions to control processes through digital or analog input/output modules, offering features like logic sequencing, timing, counting, and arithmetic operations, working in real-time, ensuring that output results are produced promptly in response to input conditions. Within the automotive sector, the PLCs are mainly used to interpret inputs from positional, electrical, or magnetic sensors, cameras, scanners, photocells or barriers, and perform actions with the servo, valves, switches and other robotic components to ease and automate the assembly process, working alongside the traceability software [62, 63].

Although there's not many papers or work cases that document a direct integration between the YOLO vision model with a PLC hardware, there is, in the field of machine learning applications, a few papers that mention the integration of python software with Programmable Logic Controllers. Rubio-Manrique et al. [64] provides a scheme to connect a python script to a PLC using the Snap7 library (that offers communication capabilities with Siemens S7 PLCs).

This suite library offers an approach for developers to create software that can read and write data directly into the PLC data blocks, to then be used to carry out controls and perform checks in the factory shop floor.

2.7 Conclusion

In this state of the art literature review, the topics most relevant to the development of the project were presented. A summary was given of the evolution of computer vision, from its beginnings and limitations to the present day where these limitations no longer exist to the extent of before, with differentiations between Single Stage Detection and Dual Stage Detection algorithms, presenting their main advantages and disadvantages. Also on computer vision, its applicability in the industrial sector was described, with an emphasis on quality inspection tasks and general processes, such as sorting tasks or production line automatic assembly. Regarding the vision algorithm chosen for this work, the YOLO framework was presented, its history and updates between versions, as well as its system architecture, which allows objects to be detected from an image input through a single pass through the neural network.

In order to idealize the application to be developed, case studies and work cases were also presented that focus on the more arduous tasks to be achieved in this work, such as the case of connecting the software to the existing PLC infrastructure on the shop floor, performing data augmentation on the real data and images in order to achieve a more composite dataset without the need to spend absurd amounts of time, and performing distance measurements between objects with the YOLO system, something that is also one of the important considerations to be implemented in this project.

Despite newer versions existing today, YOLOv5 was the chosen version for this project because of its extensive documentation and active community support, ensuring long-term reliability. At the project's inception, YOLOv5 was the only stable version, with YOLOv6 and YOLOv7 being relatively new. Additionally, YOLOv5 excels in performance on low-budget or lower-capacity hardware, which is crucial for industrial production lines. Its ability to be implemented in applications for defect detection, feature recognition and segmentation identification makes it ideal for the company's needs, providing robust and accessible AI first in-house solutions.

Chapter 3

Theorical Background

In this chapter are presented the various concepts, terminologies, technologies as well programming languages, tools and frameworks that are considered relevant for the broad picture of this project along with their explanation, deepen into more complex themes in some particular and more important cases. Moreover, it is also presented an additional literature review regarding Machine Learning, Neural Networks and Convolutional Neural Networks – themes presented in the State-of-the-Art chapter which will be explored and explained in greater depth in this chapter.

These concepts, technologies, processes, and tools are detailed in different sections.

In the first section, this paper will deepen into concepts like machine learning and basic neural networks, giving the basic explanation about it before extending to the image processing division of AI, with explanations on CNNs and image identification methods on these types of artificial neural networks. Also in these first section, it is explained what the Industrial Internet of Things implies in manufacturing industries, as well the lifecycle of MLOps methodologies and how it can be applied to assurance quality in artificial vision applications and models.

In the second section it is enumerated the main tools, languages and frameworks used on the course of this project, what do they bring to the table and the main advantages of their use.

3.1 Concepts

3.1.1 Machine Learning

Machine Learning (ML) is one of the branches of AI with the most notoriety and advances today, focusing on data analysis and its use to create algorithms that can approach the learning and retention capacity of human beings. Over the years, it has gone from being a small area of study in which computer resources limited its progress. However, nowadays, with technologies available such as graphics processing cards and cloud computing, for example, it has increasingly improved its processes, and therefore its learning capacity [65]. Within Machine

Learning there are three different fields:

a) Supervised learning is when the algorithm is trained according to a set of labeled data (that is, knowing and outlining the expected output for each piece of data imputed into the system). This type of learning aims to teach the algorithm to map the inputs with the outputs of the dataset, in order to make predictions or classifications on new, unseen data.
b) Unsupervised learning, unlike supervised learning, is dedicated to teaching the algorithm on an unlabeled dataset, with the aim of finding more abstract patterns or structure in the input data without the need for prior labels, allowing the system to generalize, analyze and discover insights or natural groupings in the data under analysis.
c) Reinforcement learning, unlike the first two, takes place in a controlled environment in which the algorithm learns to make decisions according to the reward given by an external observer. The algorithm interacts with the data and gives its output in the form of a prediction or classification, receiving feedback in the form of a reward if it is correct (or a penalty if it is wrong). This leads to and adjustment on its actions to improve the general performance of the model over time.

Between these three strands, AI applications are created with unique objectives such as object detection, identification, prediction, segmentation, forecasting, sentiment analysis, feature recognition, among others, with none of the fields being unanimously the most suitable for any of the applications. This is because, for certain AI algorithms, their performance will depend on the characteristics and extent of the dataset, as well as the objectives they are intended to achieve.

In the context of machine learning as it applies to this work, below are explained some approaches to some of the related terms used throughout.

3.1.2 Artificial Neural Networks

Inspired by the structure and operating principles of the human nervous system and brain, *Artificial Neural Networks (ANNs)* are algorithms capable of recognizing relationships between datasets automatically, using known methods that occur during the processing of information in the human brain. These networks are designed with a large number of simple units, or neurons, and nodes, separated by multiple layers and connected to each other, working in parallel to process data in real time, simulating the synapses that occur in the brain [66]. When applied to the real world, neural networks play an important role in ML applications, namely in classification, prediction, and identification processes (through supervised learning techniques), as well as in the discovery of patterns in datasets or data clusters (unsupervised learning).

Objectively, ANNs are represented by nodes (individual units in each layer of the neural

network, which receive information from the layer before it, modifying it in the process of transferring information to the next layer) and weights (parameters associated with the connections between nodes and which can be involved with the strength of that connection through its 'strength'). One of the most basic and well-known types of neural network is feed-forward, where data moves from an initial point (input) to its end point (output), and where the nodes of a given layer modify the results and values of the next layer through the variables embedded in the weights of the network [65], as shown in the following figure:

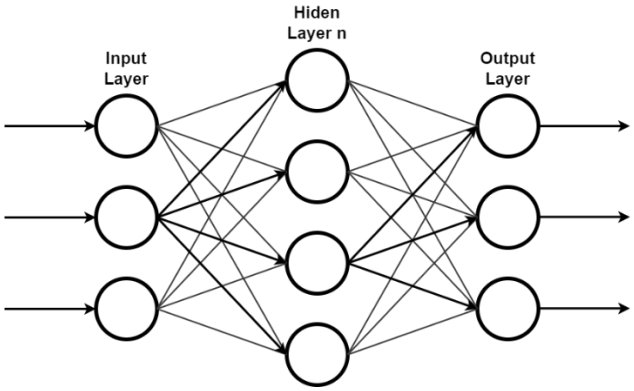


Figure 5 - Feed Forward Neural Network Diagram

In this type of neural network, although it does not have its own memory, the ability to adapt to the data supplied to it can be achieved by modifying the parameters of the weights in the connections between nodes in different layers, so that the combination of values provides the expected outputs for the inputs given to the neural network. Throughout the training of a neural network, these values are automatically adjusted using backpropagation or gradient descent algorithms [67], so that data that has already been through the learning process is not 'lost' and always has an influence on the final result, while these weights are still being adjusted.

3.1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), or also known as *ConvNets*, are an advanced class of Artificial Neural Networks that specialize in handling and processing data that has an advanced grid structure, such as images. CNNs have been designed and developed specifically to extract useful features from images, whether they are single images or extracted from video feeds, in order to achieve a singular objective, such as object detection or image classification.

The organization of a CNN follows the basic principle of a feed-forward neural network in terms of its layer structure, and there are three very important types for this type of neural network, as presented in the next figure:

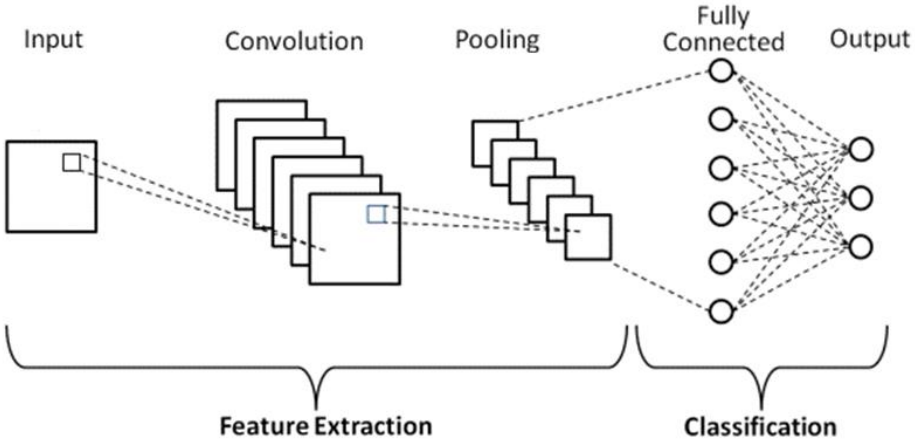


Figure 6 - CNN Diagram (extracted from [68])

The general structure of a CNN are separated in three main layers, as illustrated in figure 6:

1. Convolutional Layer: At the heart of a CNN is the convolutional layer, which is responsible for most of the calculations. This layer uses input data, filters, and feature maps to learn image features using small squares of input data. The convolution operation preserves the spatial relationships between pixels by using a kernel to detect specific features, resulting in convoluted features or activation maps.
2. Pooling Layer: Pooling layers, also known as down sampling, reduce the dimensionality of the input without learning parameters. Maximum pooling and average pooling are two primary types, which help with feature extraction while potentially discarding less relevant information.
3. Fully Connected Layer: Located at the end of the CNN architecture, the fully connected layer establishes global connectivity between neurons, transforming high-level features into final results. Unlike the convolutional and pooling layers, it connects all the neurons to those of the previous and subsequent layers, usually employing a SoftMax activation function for classification tasks.

3.1.4 Object Detection Techniques

When talking about object detection in images, there are several existing techniques that deal with the identification or classification of objects in a given image, such as bounding box prediction, segmentation (semantic and instance), template matching or classification, for example. In the context of AI applications, the two types of detection that are mostly use are bounding box identification and instance segmentation identification. They are both used for supervised learning dataset creation, with images that have 'annotations' (or labels) identified the desired parts for the vision model to detect.

Bounding box identification is an image identification method that involves drawing

rectangular boxes around objects in images [69], while instance segmentation identification detects and segments objects around their perimeter at the pixel level, providing precise outlines [70].

Whereas Both can be used in application like autonomous driving, medical imaging or surveillance, the labelling part of the dataset that trains the vision model is different. While bounding box models require for the user to only 'delimitate' the square area that surrounds the desired object and assign a label to it [69], instance segmentation needs a detailed polygonal shape to take form surrounding the object, shaping it and 'force' the vision model to generate a polygonal object in different positions [70]. In terms of time necessary for the user to label the images in dataset, bounding box methods are much faster to annotate, being necessary to only give the four different points of the rectangle. For instance segmentation datasets, the labelling part is much more time consuming, because its necessary to input all the necessary points so that the perimeter of the object is fully identified.

In this project, these two types of object identification techniques were chosen and used (instance segmentation was the technique chosen initially, present and documented in subchapters 5.3.3 - October to december 2022 and 5.3.4 - April to august 2023, with bounding box identification being used in subchapter 5.3.5 - March to may 2024, when implementing the application on a real assembly line).

3.1.5 Industrial Internet of Things (IIoT)

The *Industrial Internet of Things (IIoT)* is an extension of the *Internet of Things (IoT)*, being specifically focused on the industrial sector and their applications, connecting machines, devices, and systems to enable data exchange and automation in various areas, especially on factory floors across different activities [71]. By incorporating AI and ML, IIoT processes aim to create intelligent systems that can respond autonomously to data and instructions, increasing efficiency and real-time decision-making in sectors such as transportation, healthcare, agriculture, energy, and manufacturing [72]. Its evolution has been marked mainly by advances in remote monitoring (such as the creation of dashboards based on real time sensory data), cognitive and qualitative analysis and industrial control processes. Initially, the IIoT aimed to improve operational efficiency, as well as greater productivity and better management of industrial assets and processes through intelligent monitoring [71].

The IIoT ecosystem keeps industrial equipment, machines and cloud-based applications interconnected, highlighting the advantages and technologies and, at the same time, the challenges faced in their implementation. This evolution reflects a shift towards taking advantage of data-based information and predictive maintenance to optimize industrial operations and asset management in the era of Industry 4.0.

3.1.6 Machine Learning Operations (MLOps) lifecycle & model quality assurance

Machine Learning Operations (MLOps) is a methodology that combines DevOps principles and guidelines with machine learning processes in order to improve collaboration, speed and quality in the development of AI models [73], using various technologies to simplify the workload required, from model design to its final operations. In simple terms, MLOps aims to standardize and improve the development, monitoring and management of ML models in a production environment. This type of process is identified by the following components:

- *Continuous Integration and Development (CI/CD)*: MLOps emphasizes automating the process of integrating updates into the code, testing, and deploying models to the production environment in a consistent and seamless way;
- Model Monitoring: Continuous monitoring of deployed models to track performance metrics, detect deviations and ensure that models remain accurate and effective over time;
- Scalability and Resource Management: Ensuring that ML systems can be scaled effectively to handle variable workloads and resource demands efficiently.
- Version Control: Manage versions of datasets, models and code to monitor changes, reproduce results and facilitate collaboration between data scientists and engineers.

These mechanisms allow MLOps methodologies to be more efficient, reliable, and highly scalable, with their work cycle being the amalgamation of two well-known cycles, DevOps and DataML Operations.

The DevOps software development life cycle can be interpreted as an 'infinite' loop [74], being a cycle consisting of two different types of activities (design and development of a piece of software, and its deployment and monitoring in production), following them according to the steps below, on figure 7:

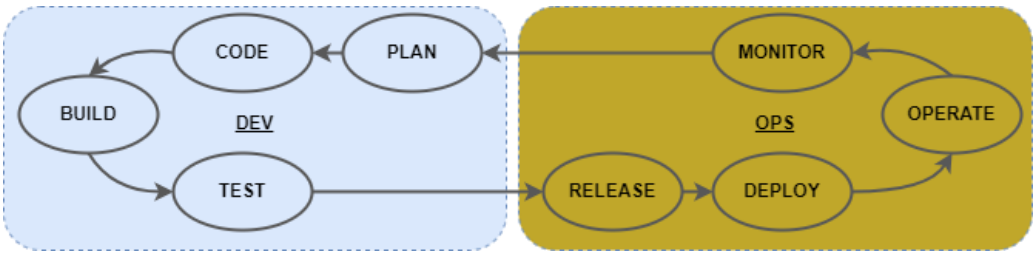


Figure 7 - DevOps Cycle (extracted from [74])

1. Planning: Definition of objectives and metrics in which the project will be evaluated, analyzing the problems faced by the end user;
2. Code: Developing end-to-end software, establishing APIs that can be part of the

application and ensuring team alignment;

3. Testing: Testing the software automatically and comprehensively;
4. Release: Version control and transition to a continuous deployment;
5. Deploy: Using deployment services, send the software to production on the designated infrastructure;
6. Operate: Gradual scaling combined with suitable software and methods such as Docker containers and Kubernetes;
7. Monitor: Continuous monitoring, according to program errors, problems, or latencies, which should be based on the severity of the problem and will restart the cycle with a replan of the application.

In a similar way, the DataML cycle can be described as follows on figure 8 with the following steps:

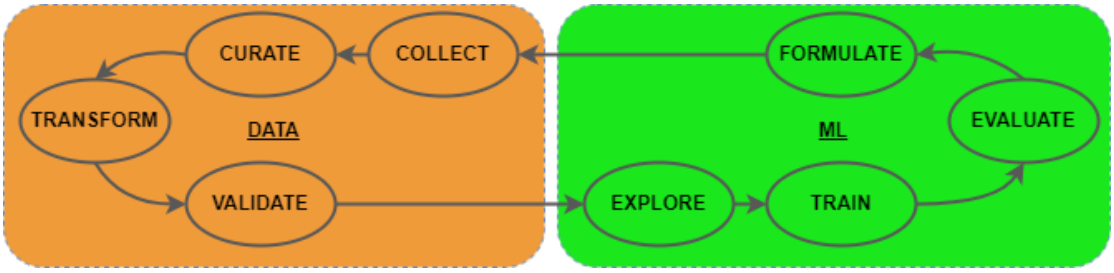


Figure 8 - DataML Cycle (extracted from [74])

1. Formulate: Definition of ML problems associated with business objectives, taking into account the associated possible False Positives and False Negatives and considering the availability of data and its evaluation metrics;
2. Collect: Data collection from available sources, storing it in a 'raw' area of the data lake / data warehouse;
3. Curate: Cleaning unnecessary data, eliminating duplicates, identifying data for supervised learning, cataloging and organizing for better access, among others;
4. Transform: Automate transformations in the data processing work pipeline. Store transformed data in a 'gold' zone of the data lake / data warehouse;
5. Validate: Implement quality checks on cataloged data;
6. Explore: Iterate on transformations and validations;
7. Train: Experiment with different models and hyperparameters, selecting the best performing models for later evaluation;
8. Evaluate: Contrast the trained model with the business objectives and associated metrics, with the possibility of reformulating the model development.

Combining the DataML and DevOps cycles, the MLOps cycle can be illustrated in figure 9, with 4 distinct but complementary stages:

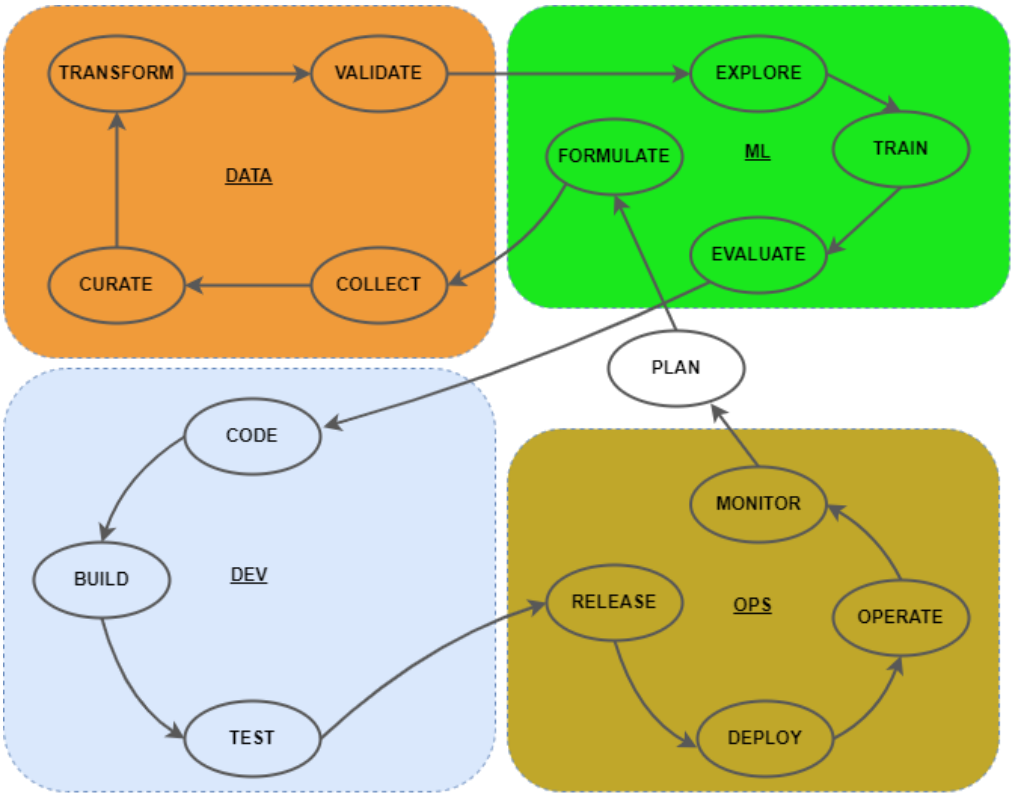


Figure 9 - MLOps Cycle (extracted from [74])

In addition to the measures conferred by MLOps, Studer et al. [75] also proposes a process model for the development of machine learning applications focused on quality assurance methodology based on the lifecycle of ML applications. Because, despite the widespread use of ML in various industries, there had been a lack of a standardized process model to increase the efficiency and success of ML projects. To fill this gap, the authors propose with CRISP-ML(Q) a comprehensive framework covering six key phases:

1. Scope Definition: This initial phase involves defining the objectives and scope of the machine learning project, laying the foundations for subsequent development phases. The tasks of understanding the business and the data are carried out simultaneously in this phase, recognizing their significant impact on the viability of the project;
2. Understanding the Business and Data: The business objectives and available data are thoroughly analyzed in this phase, as they are closely linked and can influence each other. This phase merges the tasks of understanding the activity and the data into a single phase, reflecting industry practices that have demonstrated the interdependence of these activities;
3. Data Preparation: In this phase, data is processed, cleaned, and transformed to make it suitable for modeling. Data quality and relevance are crucial aspects addressed during data preparation to ensure the accuracy and effectiveness of the subsequent modeling

phase. In our work case, it will be the data labelling and augmentation;

4. Modeling: The modeling phase involves selecting and applying machine learning algorithms to create predictive models based on the prepared data. This phase focuses on developing models that can respond effectively to the project's objectives and requirements.
5. Evaluation: Once the models have been built, they are evaluated to determine their performance, accuracy, and overall adaptability to the real data. Evaluation metrics are used to measure the effectiveness of the models and determine their suitability for implementation in real-world scenarios;
6. Implementation: The final phase involves implementing the machine learning models in operational environments. Special attention is paid to monitoring and maintaining the implemented models to ensure continuous performance and mitigate the risk of degradation over time.

One of the distinctive features of CRISP-ML(Q) is its emphasis on quality assurance methodology throughout the ML application development lifecycle. By integrating quality assurance practices at each stage, the model aims to identify and mitigate the risks associated with ML projects, drawing on both practical experience and scientific literature to ensure robustness and adaptability. This proactive approach to quality assurance is crucial to maintaining the performance and reliability of ML applications, especially in dynamic real-time environments where models must adapt to changing conditions.

In addition, the document highlights the importance of addressing legal constraints, ethical considerations, data complexity, model explainability and scalability when developing ML solutions. These factors play a significant role in ensuring the ethical and responsible implementation of ML systems, particularly in sensitive areas where fairness, transparency and trust are vital.

In conclusion, the document advocates the establishment of industry-neutral standards for the development of ML applications and calls for collaborative efforts to define a cross-sector process model for ML projects. By introducing CRISP-ML(Q) as a step towards this goal, the authors aim to provide a structured framework that can improve the success, efficiency, and quality of ML applications in various domains and industries.

3.2 Tools, Languages and Frameworks

3.2.1 Computer Unified Device Architecture (CUDA)

Computer Unified Device Architecture (CUDA), is a standard programming interface implemented in NVDIA GPUs for parallel programming. This type of programming allows for faster and more efficient management of complex problems and matrixial calculations compared to normal CPUs, and has multiple applicability in various areas, from the financial market to medical and big data applications [76]. With CUDA enabled GPUs, users can

leverage the potential of the hardware installed in their machine to use in developing and training machine learning applications, and not just rely on the local machine's CPU, making the training and inference processes exponentially faster and more accurate in the results. In short, NVIDIA's CUDA plays a key role in improving computing performance and enabling parallel processing for different applications.

3.2.2 Roboflow

Roboflow [77] is an online platform dedicated to creating computer vision datasets for object detection applications, from the beginning to the end of the model creation process. In Roboflow, users can create visual detection projects (such as object detection, segmentation or classification), label images and even train models through the platform, accessing them via an API provided by the site, or, in the case of this project, export the dataset of all the annotations on a given model output (from YOLO, COCO or tensorflow). It is also possible to perform health checks on the dataset (label control), while having access to version control of the past versions of the model and deploy models directly from the platform. This makes it an important platform in the field of computer vision due to the resources it gives the user, that otherwise would be more time-consuming and laborious tasks.

3.2.3 Open-Source Computer Vision (OpenCV)

Open-Source Computer Vision (OpenCV), is an open-source library widely used to develop computer vision applications efficiently [80]. It is designed to process and analyze visual data from an input, such as an image or video, offering different functionalities for processing and using the data, such as classifying or identifying objects, detecting borders and feature extraction, optimized to be supported by different operating systems and mobile platforms [81].

In computer vision applications, the OpenCV library is used as a mean to display the algorithm's YOLO output (as is the case in the YOLO framework), with the object detections annotated to the user, as well being able to mark symbols and write text on top of it. This makes the library useful to create a more dynamic and interactive application, where is possible to have results on the object detections inferred.

Overall, OpenCV plays an important role in enabling the development of innovative solutions in the areas of computer vision, image processing and machine learning.

3.2.4 Information exchange (with snap7 and python-snap7)

Snap7 is a high-performance, open-source communication suite designed for Siemens systems to perform information exchange between the Siemens S7 PLC devices. It utilizes industrial Ethernet interfaces for communication [82], allowing for cross-platform operations,

while supporting multiple S7 models, like the S7-200, S7-300, S7-400, and S7-1200/1500 models.

Snap7 also supports multiple programming languages, including C, C++, Python, and C#, making it versatile for different development environments [83]. It provides various functionalities such as reading and writing data blocks, inputs, outputs, and other memory areas of the PLC, as well as reading and writing variables and tags, monitoring and controlling PLCs remotely. Optimized for performance, Snap7 ensures efficient and reliable communication with PLCs, making it a valuable tool for developers and engineers in industrial automation applications, including manufacturing processes and building automation [82].

Like OpenCV, python-snap7 is an open-source code library that combines Python's user-friendly programming with traditional PLCs and communication with Snap7 Siemen Systems, offering an innovative approach to the world of industrial programming. It enables rapid prototyping as well as advanced controls within the PLC environment, flexibility that allows its users to implement satisfactory communication and automation protocols. In addition, python-snap7 facilitates seamless integration with existing software ecosystems and industrial standards, improving interoperability and synergy throughout the system [84].

3.2.5 Totally integrated automation (TIA) portal

The Totally Integrated Automation Portal (TIA) Portal, is a comprehensive engineering framework used in industrial automation to design, configure, and diagnose automation systems. It integrates various automation software tools for programming PLCs (Programmable Logic Controllers), HMIs (Human-Machine Interfaces), and other devices, providing a centralized platform for efficient control system development [85].

TIA Portal enables real-time monitoring and control of processes, enhancing system efficiency and accuracy while reducing human error risks. It facilitates the creation of automated control systems with features like remote control, system visualization, and data storage capabilities. Additionally, TIA Portal plays a crucial role in improving the security of PLCs by investigating potential vulnerabilities and enhancing protection mechanisms against cyber threats [86].

3.2.6 Python, Pytorch and Tensorflow

Python is a high-level programming language, known for being incredibly versatile and flexible, and widely used in applications for its readability and simplicity. It supports various programming styles, including object-oriented programming (OOP) [78]. Developed in 1991, the Python language allows code to be readable through a simple syntax and a dynamic typing system, suitable for various applications such as data analysis, web development and machine learning [78]. Its popularity is evident in its widespread use in multiple industry sectors, its community backup, and its role as a fundamental pillar in emerging and advanced technologies

such as artificial intelligence, machine learning and big data in renowned companies and industries. A survey revealed that Python is widely used, with 99.8% of respondents considering it to be a popular language in the digital industry [79].

On another hand, Pytorch is a very versatile deep learning architecture development framework that has gained a lot of popularity among data scientists and software engineers due to its capabilities, as it allows for the development of large-scale deep learning and artificial intelligence projects, combining tabular data with images, text and even audio in multimodal applications [87]. Pytorch's architecture was designed to be as user-friendly as possible, supporting the ecosystem inherent to Python, for better integration and debugging with other data computing libraries, while remaining efficient and compatible with hardware accelerators such as GPUs and drivers associated [88]. In essence, Pytorch provides a robust platform for developing and deploying machine learning models, offering flexibility and efficiency in manipulating complex data such as text, images or audio in the creation and development of models. Because of this, Pytorch was the DL framework chosen to implement this project, after software and algorithm studies were made.

Tensorflow is an open-source library for developing machine learning applications, developed by Google in 2015, which allows Artificial Intelligence algorithms to be run on different devices, from smartphones to large computer systems, with minimal adjustments. Tensorflow uses data-flow graphs to define operations and computations, distributing them across different machines and processing equipment, such as CPUs, GPUs and TPUs (tensor processing units) [89], thus becoming the pillar of complex neural networks and their algorithms, through its operations on tensors and its operations, such as matrix addition and multiplication. Tensorflow is therefore used extensively today in various applications, including image classification, data analysis and DL model training, making it a fundamental tool in the research and development of algorithms in the field of Machine Learning.

In this work case, the tensorflow library was used in implementation of the proof-of-concept model, document on 5.3.2).

3.2.7 Anaconda and Jupyter notebook

Anaconda is a distribution of Python and R language, designed for scientific computing and data science tasks. It simplifies package management and deployment through it package manager, conda, allowing user to create isolated environments for different projects [90].

Jupyter notebooks are part of a pre-installed anaconda libraries and are interactive web-based documents that combine text, code and outputs like tables, images and plots, used for exploration, documentation and scientific research. They allow for interactive executions, saving time on reloads, imports and conversions [91] and became especially popular due to Python's high development speed and easy integration with an extensive libraries catalog like NumPy, Pytorch and OpenCV.

For this project, jupyter notebooks were used for tasks like image gathering, application prototyping and tensor manipulation tests.

3.2.8 Software versions summary

Considering the numerous software packages and versions to be used during the project, table 1 below lists the OpenSource software and their versions in order to simplify the search for installations and dependencies associated with the project, taking into account the type of GPU installed locally.

Table 1 - Software Versions Summary

GPU Installed	Computing Power	Software	Version
Quadro M1200	5.0	CUDA SDK	11.8
		Python	3.9.3
		openCV python-snap7	4.2.0 1.4.1
		YOLOv5	7.0
		conda	22.3.0
		pytorch	1.9.0
		tensorflow	2.4.0
GTX 1650 LP 4Gb	7.5	CUDA SDK	11.8
		Python	3.9.3
		openCV python-snap7	4.2.0 1.4.1
		YOLOv5	7.0
		conda	22.3.0
		pytorch	1.9.0
RTX 4070 16 Gb	8.9	CUDA SDK	12.4
		Python	3.11.7
		openCV python-snap7	4.9.0 1.4.1
		YOLOv5	7.0
		conda	24.5.0
		pytorch	2.3.0

As can be observed from table 1, the physical GPU installed locally on the work computer has a direct influence on the version of the CUDA SDK to be used in the project. Therefore, the above versions of CUDA SDK, python, pytorch and conda are those that are documented online and those that provide compatibility between the different platforms.

3.3 Conclusion

This theoretical background chapter presents the main concepts which, although not presented in the literature review, are also important for this work. In terms of concepts, the path and foundations of machine learning and simpler neural networks were outlined, up to and including convolutional neural networks capable of extracting and processing information from images, as well as the main approaches to annotating images for training vision models. The

concept of IIoT was reviewed and how it is interconnected with companies such as *AUTOProduction* on Industry 4.0 issues, where, among other things, AI solutions are used in the production and assembly of products. The MLOps cycle was also mentioned, outlining its important CI/CD, continuous monitoring, scalability, and version control guidelines for software standardization based on ML models, as well as guidelines for a methodology for quality assurance of ML models, including scope definition, understanding of the business, data preparation, modelling, evaluation, and implementation.

With regard to the tools to be used, there was an introduction to the python language and its OpenCV and python-snap7 libraries, which are important for manipulating the YOLO display and connecting to the PLC (along with the TIA Portal interface), respectively, as well as mentions of the CUDA interface (which simplifies and improves GPU performance when training models), the Roboflow website (which standardizes the preparation of the vision model dataset), and the Pytorch framework (which makes it possible to use YOLO within an anaconda environment).

The tensorflow library, jupyter notebooks were also mentioned and used for unit testing and proof of concepts, as well as SQL Server, which will serve as a database to feed the application, both in test and production environments. At the end of the Tools subchapter, it was also provided a detailed table in which the software versions installed on the computer are associated, depending on the GPU installed locally.

Chapter 4

Problem Analysis

This chapter presents the analysis of the problem that this project aims to solve, as well as the general architecture of the system, with the different components that integrate it. Firstly, there will be a detailed introduction to the process on the assembly line, as well the Business *Process Model and Notation (BPMN)*[3] associated with, where the solution will be implemented, so that all the intrinsic components of the product to be manufactured and its quality components are considered.

This will be followed by the definition of the system requirements, both functional and non-functional, where the most important needs in terms of application functionality will be designated for the best implementation and scalability within the company, as well as the presentation of the use case diagram, where the main users of the application and their main responsibilities within the developed system will be presented and detailed.

4.1 Overall assembly process overview

Around the world, within the manufacturing industry (especially in the automotive sector) there is a great motivation to introduce innovations and techniques driven by Industry 4.0 into their processes. Although some industries are going down the path of robotization, with 33% of the companies surveyed in 2022 claiming to have invested in robotic solutions for their shop floor [93], around 24% have started investing in intelligent digital solutions, with adoption expected to be around 60% by 2025 [93, 94]. Behind these measures is, rather than a potential increase in sales or an improvement in the products manufactured, a reduction in time and an increase in efficiency [93].

Within the solutions created in an AI environment, the primary objective is a substantial reduction in automation projects, improving the efficiency of inspections, with 26% of companies choosing not to outsource this development of solutions [95], as it allows them to maintain and leverage their know-how of the processes and technologies implemented.

[3] *Process Model and Notation (BPMN)* - https://www.bpmn.org/

On a certain assembly line belonging in the *AUTOProductions*' shopfloor, the various processes (or operations, OPs) are divided into different phases, identified by the flowchart of the assembly line, in figure 10:

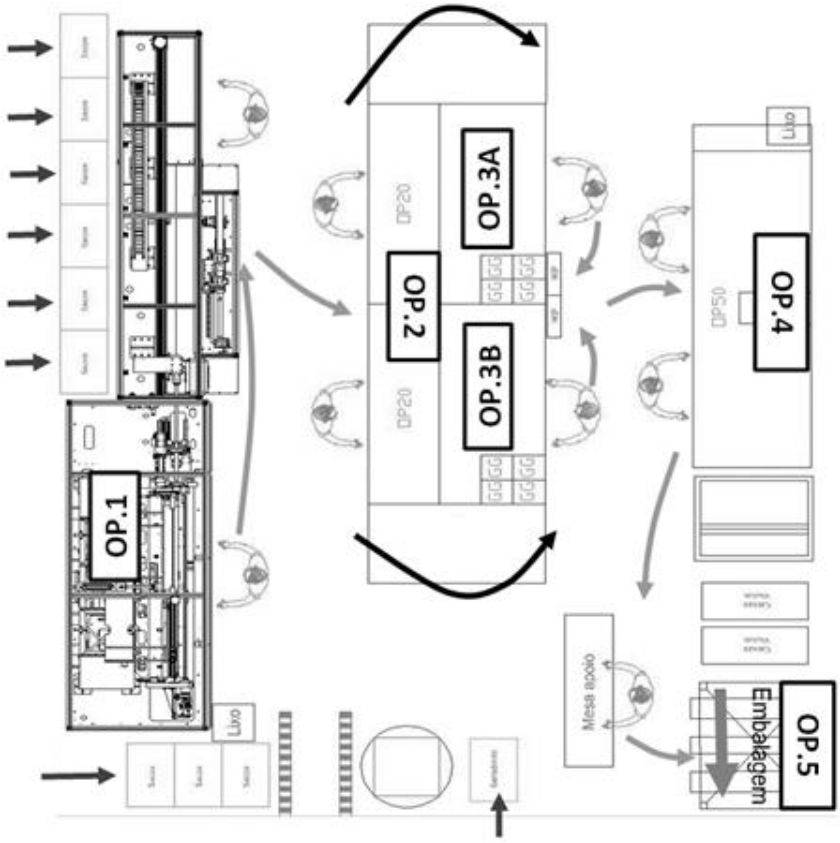


Figure 10 - Assembly Line's Process Flowchart

1. OP1: Folding the airbag bag and inserting the folded bag into its enclosed cloth retainer (sock). The worker only must insert the bag, spreading and stretching on the folding machine accordingly. The machine will then fold the bag in a criss cross sequence of 3-4cm each. Once folded, a second machine pulls the bag through a cloth retainer, thus maintaining the bag folded and ready to be manually manipulated for the next operation.

2. OP2: Placement of plastic and metallic clips, for the future installation of the airbag module on the roofs of cars, in the OEM's assembly plants. The socks mentioned on the previous step have a little square piece of fabric, with a hole in the middle (known as the bag ear), through which the plastic clip must be inserted, so to not move around and be secured in position. The metallic clip is then installed on the plastic one (being useful to secure the module in production on the following operation, and on the roof of the cars, on the OEM factories).

3. OP3A e B: Placement of the generator, installation of its support tether plates, oeticker clamp and aid tapes. The generator is the explosive component of the airbag, the one that is going to inflate the bag once a collision is made with another car or object through an electric pulse that will it, thus initiating a fast combustion reaction inside it, generating a lot of volumetric gas in a fraction of a second. The gas will then be 'stored' in the bag folded in OP1 for some seconds, and, hopefully, be determinant in the saving of the life of the car's occupants. The two tether plates are used for better supporting the module and positioning them in the car's assembly factories, while the oeticker clamp is used to fix the generator to the plates, making sure that it doesn't move anymore throughout its life span. The black tapes are general aids to secure the generator with the sock and bag, so that it holds the same position until the final verification and future installation.
4. OP4: Verification of components (plastic and metallic clips, oeticker clamp and tapes), for subsequent sewing of the plate plate, closure of the oeticker clamp and final screwing. On this step, today, this verification is being made with expensive cameras, that look for a specific color on a predetermined position for it to verify the module accordingly. It detects the presence of the plastic and metal clip, as well the bag ear, and the tapes in with six different hardware equipment's (2 cameras for 2 tapes, and 4 cameras for 4 subsets of bag ear, plastic clip and metal clip). The following sewing, clamping and screwing are made with dedicated machines that operate on the module without the needs of human interaction.
5. OP5: Final packaging. The assembled module is manually handled for the last time until it arrives at the assembly line of the OEM factory, for it to be finally installed on the car. The malleable module is folded and put on a box for it to be stored, labelled and shipped.

The whole assembly line process can also be represented in the BPMN diagram described in figure 11:

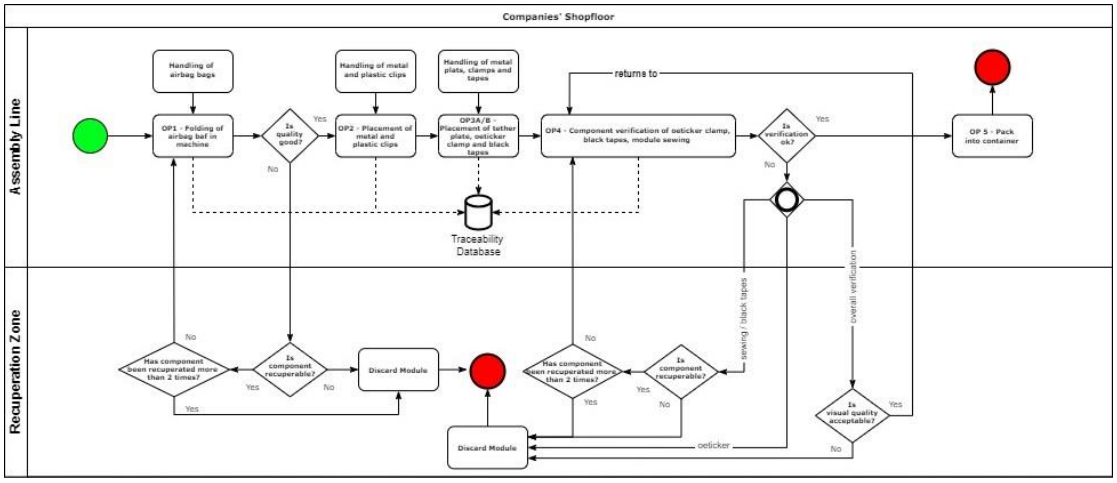


Figure 11 - Assembly Line BPMN

The Business Process Model and Notation diagram above shows the broad assembly line process for any of the products in AUTOProduction's catalog. Although the diagram begins with the transportation of the unitary components that will make up the final product to the production line, its assembly begins with OP1 with the first bag folding, followed by OP2 and its installation of clips, OP3 and the installation of tapes, clamps and tether plate, ending with the quality check in OP4 and packaging in OP5, with subsequent transportation to the warehouse as a finished product.

It's important to note that between each major OP step there are checkpoints for the module to be assembled (manual inspection after OP1, and machine inspection in OP4). Depending on the quality and defect of the part, it enters a rework cycle. If a particular part has not undergone rework more than twice, it is allowed to enter the assembly cycle again (since after two times, the components risk being degraded by successive manipulations, compromising the safety of the airbag). Also depending on the component in question, there is no rework possible, and the module must be discarded if it is not in good condition (while black tapes are easier to rework, a metal clamp could never be inserted into the module again, as it is a non-malleable material).

4.2 Requirements

Before developing the solution presented throughout the paper, some requirements were outlined that needed to be implemented for the project to be successfully implemented on *AUTOProductions*' shopfloor assembly lines, with the possibility of scaling to other areas, and even to other factory plants, governed by some of the factories' standards. This section of the paper provides a detailed overview of all the *Functional Requirements (FR)* and Non-*Functional Requirements (NFR)* for the project. By outlining each FR and NFR, we have established a clear understanding of what is required to meet the project's objectives. The FRs

define the specific features and functions that the system must perform in order to meet the users' needs. On the other hand, the NFR describes the system's performance criteria, encompassing factors such as reliability, ease of use, scalability and security, which are essential to ensure overall success.

By encompassing the FRs and NFRs in this section, we ensure a comprehensive understanding of what needs to be accomplished, enabling efficient and effective work to achieve the project's objectives.

4.2.1 Functional requirements

The Functional Requirements represent a description of the services that the software should provide, detailing the behavior and results of a software system or its components. It includes calculations, data manipulation, business processes, user interactions or any other specific functionality that defines the likely operations of the system. Functional Requirements in Software Engineering are also known as Functional Specifications [96]. All the FRs for the project are meticulously listed and described in reference table 2, which includes a detailed description of each requirement, with its unique code, a clear description, and a priority level.

Table 2 - Functional Requirements

Code	Description	Priority
FR01	The Application Owner should deploy a computer vision model and monitor its performance	High
FR02	The Application Owner should train and evaluate a computer vision model	High
FR03	The Computer Vision Engineer should be able to label components pictures and add them to the system	High
FR04	The PLC Engineer should configurate PLC client to ensure smart area connectivity and detection activities	High
FR05	The Manufacturing Engineer should configurate smart areas to detect components in the verification stage	Medium
FR06	The Shopfloor Worker should assemble the module and wait for detections from the application	Medium
FR07	The Quality Technician should evaluate and validate the trained vision model	Low

4.2.2 Non-functional requirements

Non-Functional Requirements specify the quality attributes of a software system, evaluating it based on responsiveness, usability, security, portability, and other critical non-functional standards [96]. All the project's NFRs are presented in detail in reference table 3, which provides an exhaustive list of each requirement, along with its unique code and a clear description.

Table 3 - Non-Functional Requirements

Code	Description
NFR01	Needs to perform detections in under 100ms
NFR02	Must ensure precise control over the positioning of components within the module
NFR03	Needs to have direct communication with PLC Infrastructure
NFR04	Needs to be developed on a tight budget
NFR05	Could provide training support for employees in component differentiation
NFR06	All code needs to be written in English
NFR07	All files related to the computer vision model should be stored in a dedicated, well-organized folder structure within the project repository
NFR08	All external components used in project needs to be in their most stable versions
NFR09	All external components used in project must be open source

4.3 Use case diagram

The use case diagram captures the functionality and requirements of the system through the representation of actors and their use cases. Use cases describe the various services, tasks, and functions that the system must perform. They represent high-level functionality and illustrate how users interact with the system, serving as fundamental elements of Unified Modeling language modeling.

Figure 11 provides a visual representation of the use case diagram, illustrating the actions that each actor can perform. Table 4 describes the actors and their corresponding actions and tasks in more detail.

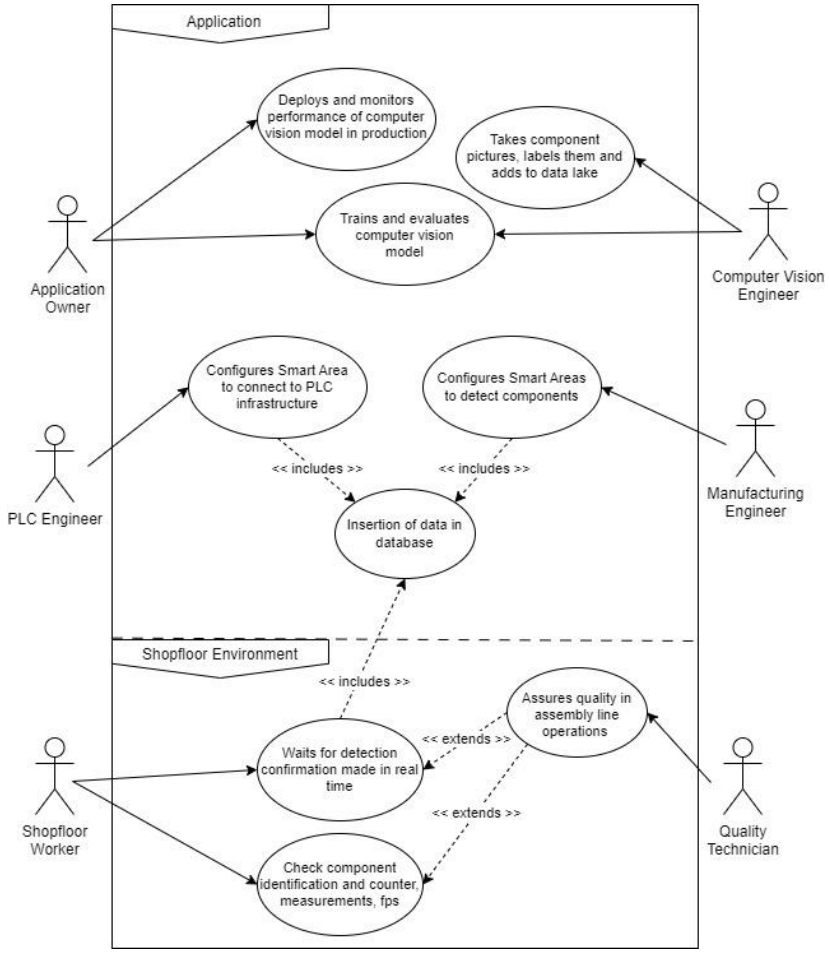


Figure 12 - Use Case Diagram

Table 4 - Application Users

User Type	**Description**
Application Owner	This user deploys trained models into production
Computer Vision Engineer	This user is responsible for the vision model development
PLC Engineer	This user is responsible for the connections to the app on the PLC side
Manufacturing Engineer	This user has rights to create smart area configuration
Quality Technician	This user checks and monitors detections
Shopfloor Worker	This user observes detections

As illustrated in figure 12 and table 4 above, among *AUTOProduction's* factory shopfloor environment, it is expected that the application developed will have several interacting agents with different responsibilities. Below is a short summary of the responsibilities and use cases for each of them, separating them between application users, and non-system application users:

System Users:

- Application Owner: This is the user who will have the most general and holistic view of the developed application, with the main responsibility of deploying and remotely

monitoring the performance of the computer vision models, as well as assisting in the training and evaluation of the vision models achieved by the computer vision engineer.

- CV Engineer: This user is responsible for creating datasets for the vision models, creating projects, organizing them, acquiring photographs, and creating annotations for the final vision model (as this is a very time-consuming task), and assisting the application owner with local training.
- PLC Engineer: This user, focusing solely on the shop floor's PLC infrastructure, will be responsible for programming and providing the connection settings on the PLC side, so that the application has its smart area functions ensured.
- Manufacturing Engineer: This user has responsibility for the components to be detected in the module, configuring the smart detection areas to the desired size, and expecting a certain type of component for it.

Non-System Users:

- Quality Technician: Ensures and validates the checks obtained by the application, in order to guarantee the vision model's good performance.
- Shopfloor Worker: This user is responsible for assembling the airbag module on the line, waiting for the application's checks and visual identifications, such as distance measurements or reference identification.

4.4 Conclusion

In this chapter, the problem analysis and overall assembly line process where the project will be inserted are thoroughly examined, as well the functional and non-functional requirements prompted for the application, as well the use case diagram and the user that are going to interact with it.

Understanding the manufacturing process as a whole, through the assembly line process flowchart and the BPMN diagram (figures 10 and 11, respectively), and considerating the importance of the components to be identified in the module, their usefulness and their safety features for the end consumer, we can conclude that the precise detection of certain components such as tapes and clips becomes essential for the assembly of the product on the assembly line. Likewise, it is necessary to understand the importance of fast assembly cycles so that the assembly line has sufficient cadence to supply the OEMs with the quantities of material requested.

As such, the highest priority functional requirements were defined as the need for the application to carry out detections in real time, identifying them correctly and controlling their position in the module in the appropriate OPs, as well as having a direct connection to the PLC infrastructure and being carried out under a tight budget, thus enabling both applicability in new production lines and in old ones that may be susceptible to changes in the future.

The use case diagram was also examined in this chapter, identifying the main agents that will interact with the application and/or the surrounding architecture (such as the creation of vision models and their deployment), and defining their responsibilities and the needs that the

application must provide, which will then be implemented in the next Design and Implementation chapter.

Chapter 5

Design and Implementation

This chapter encapsulates the design and implementation process of the project as a whole, based on the requirements previously established, the project objectives clarified, and the use case diagram drawn. In this way, the entire process is documented in this chapter, from deciding which components to detect with our solution, the proposed architecture for the program, the process of collecting data, identifying, training and evaluating the vision model, the component integration cycle and the structure of the database that will support our program, as well as the integration of the component verification algorithm and subsequent communication with the existing PLC network.

5.1 Desired components to detect

During the manufacturing process of this type of airbag module, there are several components that are of critical importance for the good production of the piece. Mounting components such as the various clips and metallic plates are crucial to continuous, speedy and predictable production in OEM's factories, as are intrinsically safe airbag module components such as black tapes (which hold soft and malleable parts of the airbag module up until the final screwing) and metallic clamp, which will ensure that the all-important airbag generator will not move during and after the manufacturing process need to be checked in order to ensure product compliance and safety.

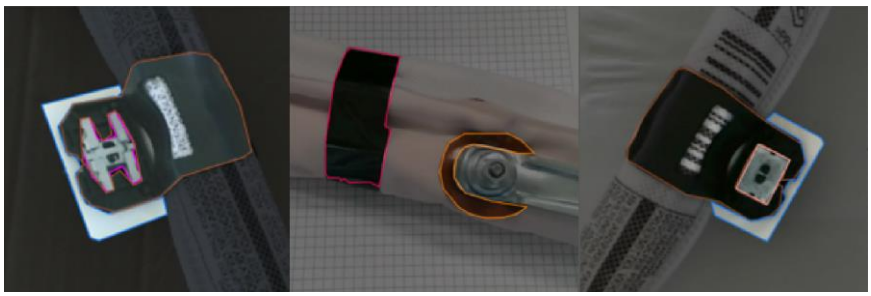


Figure 13 - Identifiable Components

Until today, this verification in OP 4 is done on assembly lines with the support of machine vision tools, such as positional cameras (which check the overall color at a given position in the image. For example, if a black tape is positioned at the image location where the black color is expected, the camera will pass a positive signal to the robots, communicating through the PLC system, in order to proceed with the tether sewing, tightening of the clamp and final screwing. At the moment, 6 different inputs from 6 cameras are needed to proceed with the final process (2 inputs for the two black tapes present in the module and 4 inputs for confirming the presence of a metallic clip, a plastic one, and the bag ear – an important component that guarantees that the clips were placed in the correct position).

5.2 System architecture

In Figure 14, below, we can have a better perception of the architecture of the intended system:

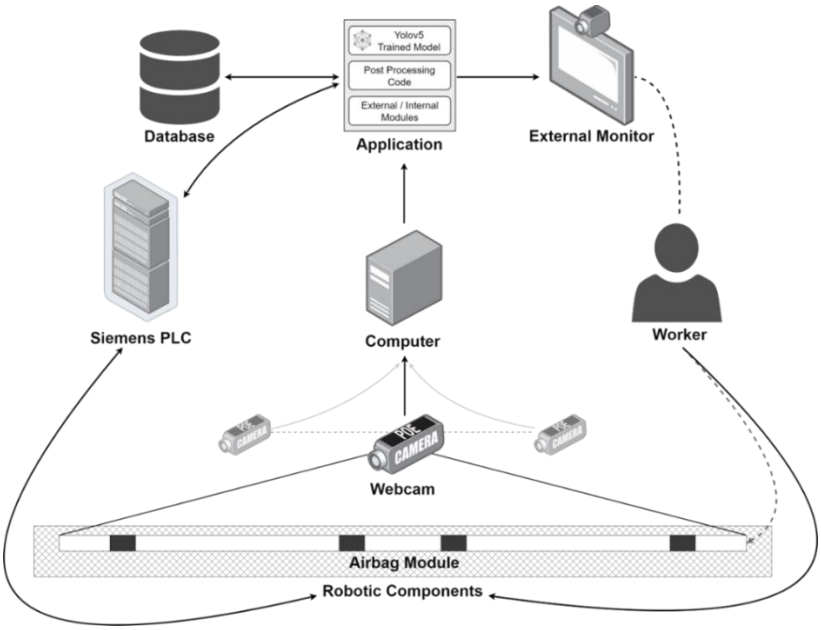


Figure 14 - Proposed System Architecture

The manufacturing worker places the airbag module on the robotic supports, holding it using the metal clips installed in Operation 2. After the module is secured on the robotic arms and machines, the worker waits for the OK signal check provided by the vision model YOLO, which shows the components it is detecting in real time on an external monitor. The image for this verification will be captured by an AUSDOM web camera, which, connected to a dedicated computer on the production line (at least, in the first instance), will feed the detection model with several images per second, allowing monitoring of the detected components. When the operator verifies that all the components are being detected at the same time, with the components of distance and positioning of tapes in accordance, he will give input to the machine, through a manual button, with communication through the PLC system, to proceed with the operation cycle, proceeding to tightening, sewing and final screwing. This cycle will only advance if all the components are being detected, requiring a visual inspection of the module, or adjustments in its position if any component is not being detected by the vision system, thus avoiding issues with OK verifications, when, in reality, the module produced does not guarantee all safety and compliance conditions for it to be packaged. To not only detect the desired components in the module assembled on the production line but also the correct position of all of them, it is necessary to create registries on how many verification control fields an OP is expecting and which components are this fields waiting for. For this, it is created on *AUTOProductions's* physical Microsoft SQL Server database a list of areas which are expecting different components (like tapes and/or clips), with the desired assembly line number, operation number and connection details with the PLC Infrastructure.

5.3 Implementation

This section describes the complete implementation of the project, from the first proof of concept prior to the start of the state-of-the-art studies on the shop floor, followed by the construction and development of an accurate and effective vision model for the intended application and criteria, implementation of the object identification and detection algorithm and connection to the PLC system. The project management methodology adopted for the development of the project was *SCRUM*[4] *AGILE Methodoloy* with 1-month sprints. However, it is important to note that the project did not develop continuously over time but had start and stop intervals due to other tasks in the company. However, at the start of each sprint it was verified that all the dependencies were up to date, and all the functionalities were still operational. Likewise, at the start of each sprint, the objectives for the sprint were outlined, which can be described in tables 5, 10 and 11, referring to the months of October to December 2022, April to August 2023 and March to May 2024, detailed in the following sub chapters.

[4] SCRUM AGILE Methodology - https://www.scrum.org/

5.3.1 Phase 1 - Proof of concept

In August 2022, the first opportunity arose to create a proof-of-concept for the current project, in which a prototype was created that validated the development of the application that would be worked on in the coming months.

As a first approach, a model was developed in Python using the Tensorflow library, mentioned in 3.2.8), due to its simplicity of use. The aim of the trained vision model was to detect black strips under the airbag module, achieving positive results, as can be seen in figure 15, below.

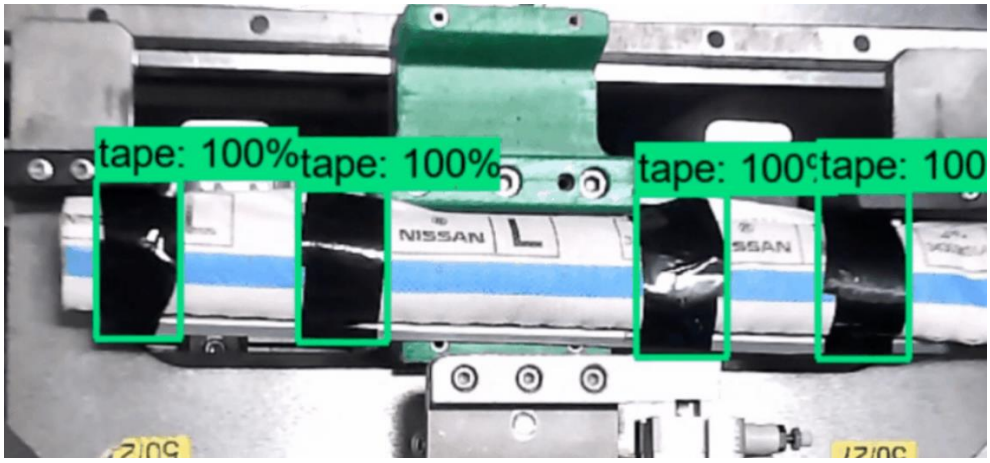


Figure 15 - Tensorflow Proof-of-Concept

Despite the good results achieved with the proof-of-concept, as well as reasonable frame rates (in the region of 5-8 per second), it was decided to opt for more intensive research and reassessment of existing machine vision algorithms, object identification methods and vision model training modalities before delving deeper into the subject (which turned out to be an asset, in hindsight).

Nonetheless, this proof-of-concept phase proved to be important as it served as a launch pad for the project that was to follow, requiring various tasks to be carried out and various topics to be researched in order for the model application to function properly.

Among the work carried out, the following should be mentioned:

5.3.1.1 General dependencies installation on work computer

One of the initial points in this first implementation phase is to install the general software dependencies on the work computer.

Just as mentioned in 3.2.11 – Software versions summary, regarding the CUDA SDK to be used on the local machine, this version information is provided by NVIDIA's own

benchmarks and guidelines, in which the SDK is associated with the computing power (from 1.0 to > 10.0) of the GPU installed on the local computer [97]. By installing the correct software versions and drivers, we can check their installation as follows:

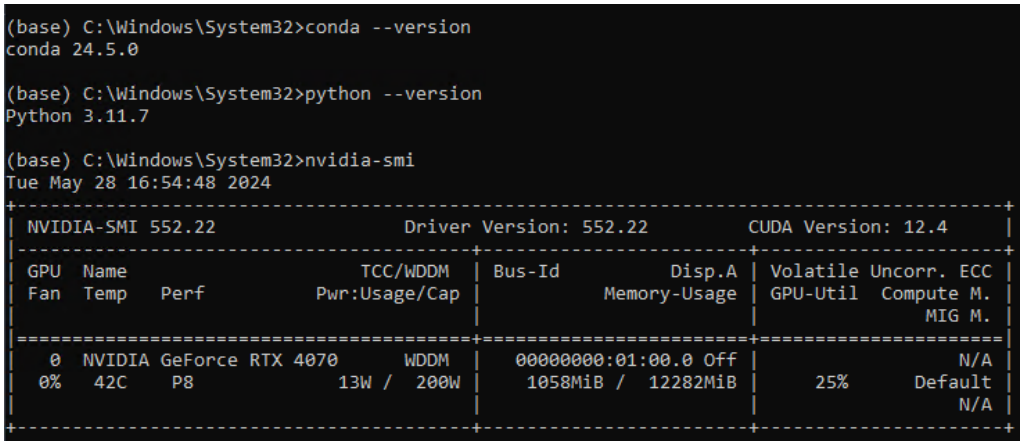


Figure 16 - Dependencies Check

As presented by figure 16, to check the versions installed of anaconda and python, the command prompt commands used return the specific versions installed for each package (24.5.0 for conda, 3.11.7 for python). To check the GPU's system management interface, the nvidia-smi command used returns the local computers' installed GPU latest CUDA SDK version available, as well the driver version, to check if it's the most updated one. The computer regarding figure 15 has a RTX 4070 installed.

5.3.1.2 Data gathering

Also in this first phase, the script for capturing images from a local webcam was built inside a Jupyter notebook (shown in Appendix A.1), which can be run on any computer, without the need for the Python or CUDA libraries (just python and anaconda to run the notebook, and openCV to handle the image capture). The script takes into account the different types of labels we want and the number of images for each one. At each time interval, the script takes a print of what the webcam is capturing at the moment, assigns a unique name to the generated image and saves it in a folder so that the vision model can then be trained.

The purpose of this script is to simplify the creation of a dataset of images with n labels needed for the vision model, with x images captured and stored with a unique identifier per label.

5.3.1.3 Conclusion

This chapter highlights the successful creation of a proof-of-concept for the application development project initiated in August 2022. A prototype model was developed using TensorFlow, demonstrating the capability to detect black strips under airbag modules with promising results and acceptable frame rates. In this phase was underscored the necessity for further research and reassessment of computer vision algorithms and object identification methods, ultimately for the benefit of the project's progress. Key activities included the installation of essential software dependencies, verification of GPU compatibility, and the development of a data-gathering script for image capture, which streamlined the dataset creation process for the vision model. This initial work laid the base for the subsequent development phases, ensuring the project's efficacy and scalability easiness.

5.3.2 Phase 2 – Programming Environment Implementation

The first sprint after the conception and development of the Proof-of-Concept took place between October and December of 2022, in which the focus was mainly on the development from the ground up of the custom vision model to be applied in the assembly line. In this way, we can list the priority tasks to be carried out, and their importance for the development of the project, as follows:

Table 5 - October to December 2022 Tasks

Task	**Importance**
Pytorch and YOLOv5 Installment	Very High
Vision Model Inference	Very High
Vision Model – Data Labelling	High
Vision Model – Model Export and Model Training	High
Vision Model – Model Evaluation	High
Integration Loop Definition	High
Project Folder Structure Definition	Medium
Integration Costs	Low

5.3.2.1 Pytorch and YOLOv5 installment

With the primary dependencies already installed, it is also needed to have the main framework from the Pytorch repository and YOLOv5 up and running. Both libraries are straight forward in their installation requirements.

5.3.2.1.1 Pytorch

Opening the Anaconda console (the main environment where the application will be running) with administrative rights, the first thing we had to do was disable SSL checks (since the network is administered globally by the company, the Pytorch package ended up being 'blocked' by the company's own guidelines).

After that, Pytorch's organizational website indicates the command we should run locally, taking into account the system requirements. In our case, the programming language is Python installed on Windows, with Conda as the main package and with CUDA 12.4 previously installed and explained at point 5.3).

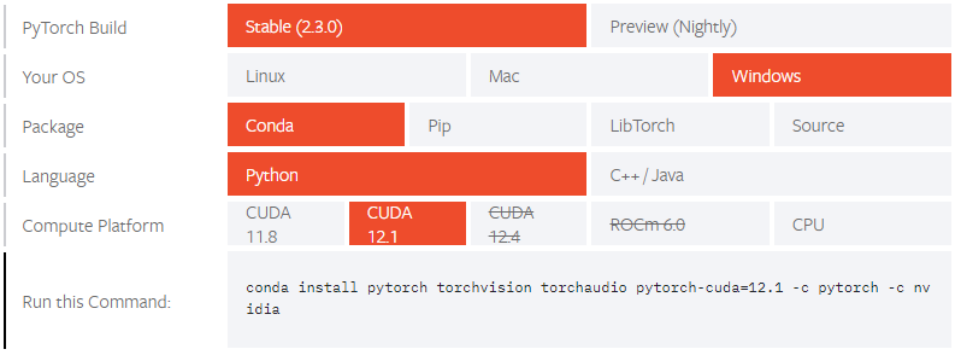


Figure 17 - Pytorch Installment Command

After running the command in the Anaconda console, we can check its installation within the Python interpreter and whether we have GPU enable on the system with the following commands:

```
(base) C:\Windows\System32>python
Python 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> import torch
>>> import torchvision
>>> import torchaudio

>>> print(torch.__version__)
2.3.0
>>> print(torchvision.__version__)
0.18.0
>>> print(torchaudio.__version__)
2.3.0

>>> print(torch.cuda.is_available())
True
>>> print(torch.cuda.device_count())
1
>>> print(torch.cuda.current_device())
0
>>> print(torch.cuda.device(0))
<torch.cuda.device object at 0x0000019F1ED7C7D0>
>>> print(torch.cuda.get_device_name(0))
NVIDIA GeForce RTX 4070
```

Figure 18 - Check Pytorch Dependencies

5.3.2.1.2 YOLOv5

To install the YOLOv5 package, a clone was made of the project's main github repository (available at https://github.com/ultralytics/yolov5), and once this was complete, the requirements described in the repository's txt file were also installed via the anaconda console with the following set of commands:

```
cd path\to\YOLOv5
conda install --yes --file requirements.txt
```

5.3.2.1.3 Comparison between default YOLOv5 models

Along with the main YOLOv5 repository come 'default' vision models, which are pre-trained on the *Common Object in Context (COCO)* dataset (large-scale object detection dataset, captioned for 80 classes). The default models are YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large) and YOLOv5x (extra-large) [46], with the 'smallest' models enabling more images to be inferred per second, while the 'heaviest' models have higher efficiency and accuracy parameters.

Table 6 - Default YOLOv5 Models Benchmarks

Model	mAP val 50-95	mAP val 50	Speed CPU (ms)	Speed V100 b32 (ms)	Params
YOLOv5n	28.0	45.7	45	0.6	1.9
YOLOv5s	37.4	56.8	98	0.9	7.2
YOLOv5m	45.4	64.1	224	1.7	21.2
YOLOv5l	49.0	67.3	430	2.7	46.5
YOLOv5x	50.7	68.9	766	4.8	86.7

During the development of the application, some additional graphics cards were tested and purchased by the company in order to get a comprehensive view of how the GPUs behave in training and image inference tasks. Of the *AUTOProduction's* existing GPUs, the Quadro M1200 is the GPU in the work laptop and the Quadro M2000 is installed in a workstation dedicated to building and rendering AutoCAD projects[5]. Of the GPUs purchased later, the GTX 1650 Low Profile 4Gb was chosen to upgrade the computers on the production lines, while the RTX 4070 was purchased to build a computer that will be used to train vision models more intensively. Below is a table comparing the level of GPU performance in the different tasks mentioned:

[5] *AUTODESK – Autocad Projects and Quadro M2000 - https://www.autodesk.com/pt/support/system-requirements/certified-graphics-hardware/autocad*

Table 7 - Train, Inference and Temperature Metrics on GPUs

GPU	Train	Inference Yolov5s	Inference Yolov5l	Temperature
Quadro M1200	10-12seg per epoch	20 FPS	1 FPS	68-70º
GTX 1650 LP	Incapaz de treinar*	20 FPS	2 FPS	58-64º
Quadro M2000	6-8seg per epoch	40 FPS	10 FPS	55-61º
RTX 4070	<1 seg per epoch	>60 FPS	25 FPS	53-59º

*due to a bug in the YOLOv5 code with color tensor and anaconda environment, it was not possible to train models on the GTX 1650 computer. Documented in [98].

From table 7 we can observe how essential a high-end GPU is in the process of training a vision model (reducing a task that would take 18h - training time on the work laptop, to less than 10 mins, just as it is important to upgrade with a low-end GPU on the factory shopfloor pcs to achieve real-time detections).

5.3.2.2 Vision model development

Developing a computer vision model from the ground up is an idea and task that seems complicated and unfeasible to anyone who has never worked with datasets or image processing in the context of ML applications. However, by following a series of simple but important steps, it becomes easy to implement. From the first image capture, through training the model to its final evaluation, this part of the document will focus only on the tasks assigned to training the vision model, as well as the best practices learned and achieved to get an accurate and effective model.

5.3.2.2.1 Data collection

The data collection stage is where the visual information (images in our case) that will be fed into the vision model training are collected.

As mentioned in point of the Data Gathering part of the Proof-of-Concept 5.3.2) subchapter, data collection was achieved using the jupyter notebook created. Five different labels were inserted into it (black_tape, metal_clip, plastic_clip, bag_ear, oeticker_plate). For each of the labels, 100 different photos were taken (totaling around 500 photos), which were placed in a local folder for later analysis.

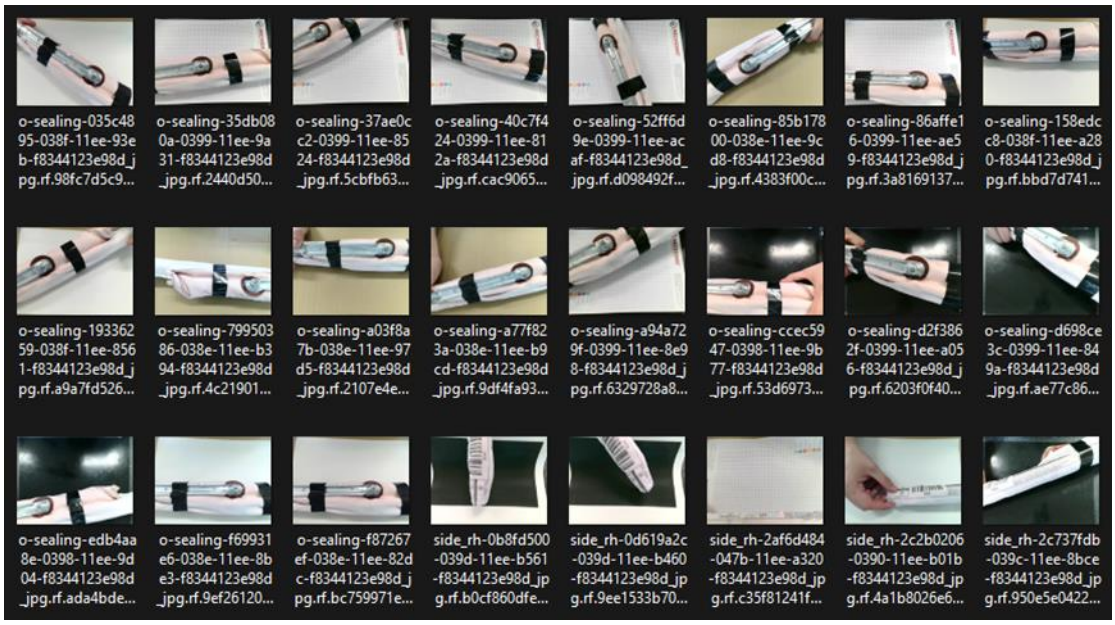


Figure 19 - Pictures Collected

An important point to remember is that, for the best possible training and generalization of the components, it is important that the images of the components differ from each other - be it by brightness, proximity to the camera, rotation of the component, light saturation, as well as different image backgrounds that contain (or don't contain) the desired component. In this way, we can give the model greater adaptability to the vision model, which will behave better and achieve better results when the image environment on the shop floor changes or deregulates (a constant in an industrial environment). After analyzing the photos taken, around 150 were eliminated because they were out of focus, had parts of the components blurred or cut off, or there were too many photos without any components present (although the presence of these photos was important for training the model's True Negatives, it was only considered necessary to have around ten of these photos).

5.3.2.2.2 Data labelling

In the data labeling stage, the previously collected images undergo a process of object identification using external tools. In terms of object identification, we can mention the two largest and most widely used annotation methods for identifying objects, bounding box, and instance segmentation.

In bounding box identification, components are identified in the image by a rectangle with 4 coordinate points, as shown in the figure below, showing the object’s class (0), and the four coordinate points that delimit it:

Figure 20 - Bounding Box Coordinates and Example

Identification by bounding boxes is the go-to method for object detection applications due to its speed of execution in image processing and easy integration with the applications developed. However, in the course of our work, some inconsistencies with this model were found, especially when it comes to components that are very close to each other, or components that overlap others (in our case, the bag_ear component 'encapsules' the metal_clip component within itself, just as the plastic_clip encompasses the metal_clip and is very close to the bag_ear. Because of these particularities, with bounding box identifications in a fully trained model, the model would end up with duplicate detections, which would invalidate the project for this type of module.

With segmentation identification, on the other hand, components are not identified in the image by just 4 points, but by multiple points, creating an irregular polygon tailored to the component in question, as shown in the figure 20 below:

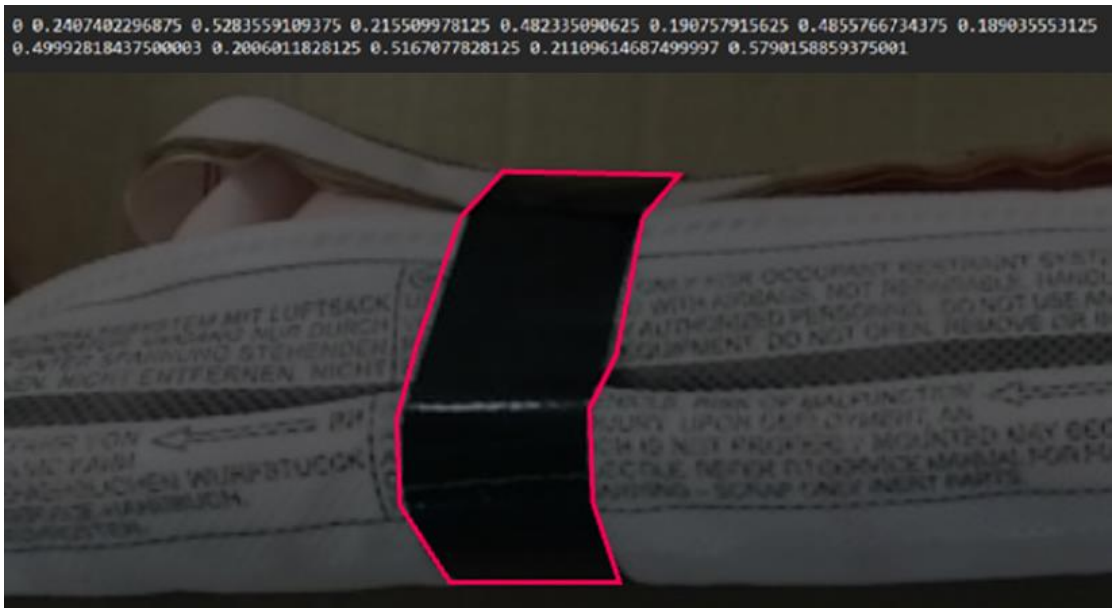


Figure 21 - Instance Segmentation Coordinates and Example

Due to the greater complexity of the data to be analyzed by the vision model, this method becomes more intensive in terms of hardware resources (it allocates more energy and capacity to the GPU and CPU) and is therefore significantly slower compared to the vision model used for bounding boxes (on par with minus 3-5 FPS). However, it is a state-of-the-art approach, more effective than bounding box identification, since it does not run the risk of having duplicate detections, and it is ideal for detecting objects that are close together.

Since it detects objects by their shape and can then show them individually with different colored masks, it was discussed with the company's shopfloor training department that the tool could also serve as a means of supporting/training new workers in differentiating and identifying different components on assembly lines on the shop floor, so segmentation identification was chosen for this case work.

Although not mentioned in the Proof-of-Concept subchapter, the vision model trained in August was trained with labeled data from the now-deprecated labelImg (open-source project, with repository on github) [99], which has since been merged with the open-source Label Studio project [100]. However, for this project, the aforementioned Roboflow website was used for the image annotations.

As previously mentioned, Roboflow is an online platform that simplifies the process of annotating, training, monitoring, and version controlling machine vision models. Below are the steps for creating a new dataset and annotating it:

1. Select the 'Create New Project' option, giving the project a name and choosing the type of vision project we want to develop (Object Detection by bounding boxes or Instance Segmentation). For this case, Instance Segmentation was chosen for the reasons mentioned above.
2. Upload Data. Within the project created and by clicking on the 'upload data' option,

we can upload the images captured on the Data Collection (5.3.3.2.1) sub chapter to the project so that they can be annotated at the next point.

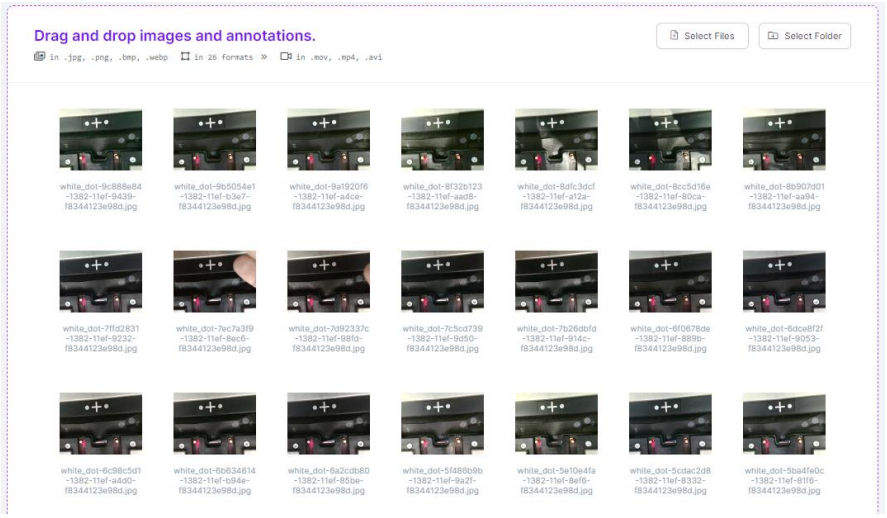


Figure 22 - Uploaded Data on Roboflow Project

3. Once the images have been uploaded, they need to be annotated for the classes required and desired. Roboflow offers paid auto-labeling or human labeling services (with the support of an external team), helping the user to create a dataset of images in what is the most time-consuming and detailed task of the whole process. In this example, we used manual annotations. In this way, we initially annotated around 350 images that we considered good enough to include in the dataset. The polygon annotations (mandatory for object detection by instance segmentation) were done point by point for all the desired components, as shown in the figures 23 and 24 below:

Figure 23 - Bag Ear Instance Segmentation Identification

Figure 24 - Metal Clip Instance Segmentation Identification

4. Once annotated, we can check the health check of the vision model, another analysis provided by Roboflow

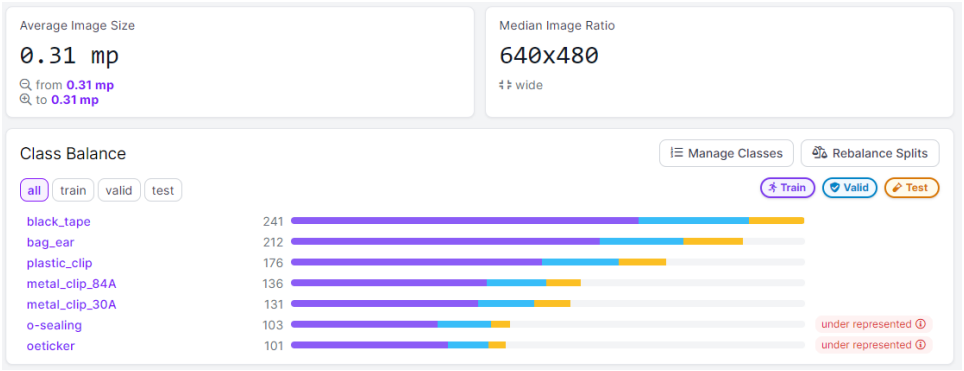


Figure 25 - Dataset Health Check

As we can see from figure 25, we have an overview of the state of the dataset. Since all the images were taken with the same camera, with the same resolution and image ration, there is no variation in this information. However, it is important to note that in the class balance shown above, there is an under representation of the 'o-sealing' and 'oeticker' classes, with fewer images from these classes having been annotated in relation to the others, which turns out not to be a problem as the components (their shape and color) are quite distinct from any of the others mentioned. Likewise, the percentages of each class between 'train', 'valid' and 'test' will be used at the Model Training point (5.3.3.2.4 sub chapter) when training the vision model locally.

5.3.3.2.3 Model export

Once the dataset has been annotated (or updated, with new photos, labels, and new annotations), we can generate a new version for training by clicking on the 'Generate' option in the project workspace. Once the source images that will construct the dataset have been configured, as well as the percentages for training - validation - testing (70 - 20 - 10), we can also adjust the configuration for data augmentation.

As mentioned in 2.3), data augmentation techniques are especially important for the proper management of an image dataset. If we were to imagine a vision model for detecting errors in the sewing of a thread in a sewing process, the dataset would have to be made according to the real errors that appear in an industrial environment. However, if errors of this kind don't occur very often, the dataset becomes 'poor' and doesn't generalize a sewing error as accurately as a dataset with thousands of images would. With data augmentation tools, however, it is possible

to ‘transform’ one image and its annotations into several, using cropping, rotation, and skewing techniques, as well as adjustments to saturation, brightness and color exposure levels. In our case, the data augmentation process will not leverage any less represented component but will add an extra set of images to the dataset with reference to the images already annotated, going from the original 350 images to around 1600, using the data augmentation configuration stated on figure 26.

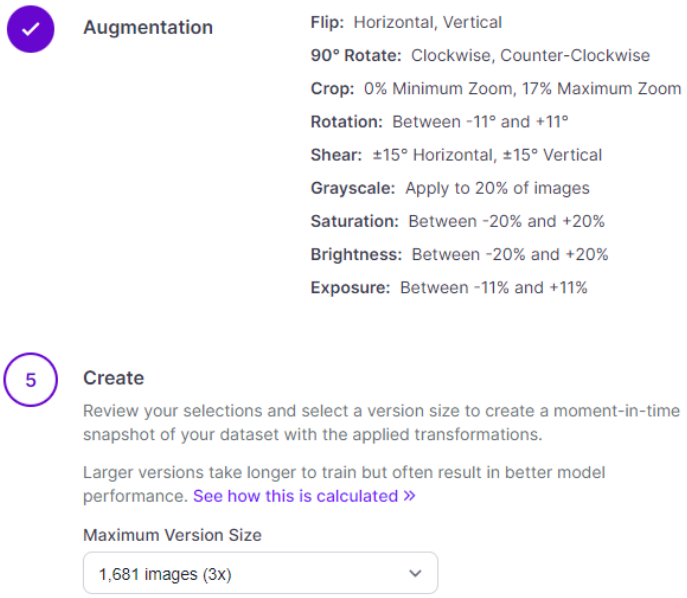


Figure 26 - Data Augmentation Configuration on Roboflow

Once the new version has been generated, it is possible to export the model in different versions (YOLOv5, YOLOv7, Tensorflow, OpenAI, COCO, among others). For our chosen work case, we chose the TXT format - YOLOv5 Pytorch, downloading the compressed folder to our computer.

5.3.3.2.4 Model training

Inside the folder of the model generated and downloaded to our computer, there will be 3 folders (test, train and valid), where the respective images and annotations made on them will be located, along with the data.yaml file, being an intrinsic part of YOLO’s K-fold cross validation method.

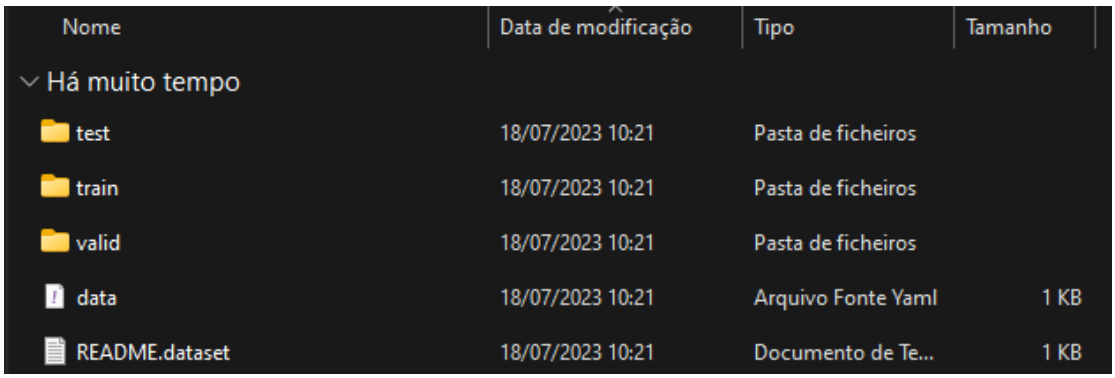


Figure 27 - Exported Model Folder Structure

The data.yaml file is the file that will bridge the gap between the ever-constant vision model training script and the variable data we want to train it with. It contains the relative paths for each of the image folders, as well as the number of classes (nc) listed in the dataset and their respective names. As this will not be the only folder for the vision models created and trained for the company, it was decided to replace the relative path with an absolute path.

```
train: <absolute/path/to>/train/images
val:  <absolute/path/to>/valid/images
test:  <absolute/path/to>/test/images

nc: 7
names: ['bag_ear', 'black_tape', 'metal_clip_30A', 'metal_clip_84A',
        'o-sealing', 'oeticker', 'plastic_clip']
```

Figure 28 - data.yaml File Configuration

Since YOLOv5 is already installed and available in the work computer, the script to train the model stands as follows:

```
    cd path\to\YOLOv5
    python train.py --img i --batch b --epochs e --data
<path\to\data.yaml> --weights <default_yolov5_model>
```

```
(base) C:\Users\hummer\Desktop\industrial-vision\yolov5>python train.py --img 320 --batch 32 --epochs 5000 --data C:\Use
rs\hummer\Desktop\white_dot_on_fabric\data.yaml --weights yolov5s.pt
train: weights=yolov5s.pt, cfg=, data=C:\Users\hummer\Desktop\white_dot_on_fabric\data.yaml, hyp=data\hyps\hyp.scratch-l
ow.yaml, epochs=5000, batch_size=32, imgsz=320, rect=False, resume=False, nosave=False, noval=False, noautoanchor=False,
 noplots=False, evolve=None, bucket=, cache=None, image_weights=False, device=, multi_scale=False, single_cls=False, opt
imizer=SGD, sync_bn=False, workers=8, project=runs\train, name=exp, exist_ok=False, quad=False, cos_lr=False, label_smoo
thing=0.0, patience=100, freeze=[0], save_period=-1, seed=0, local_rank=-1, entity=None, upload_dataset=False, bbox_inte
rval=-1, artifact_alias=latest
github: skipping check (not a git repository), for updates see https://github.com/ultralytics/yolov5
YOLOv5  2023-7-4 Python-3.11.7 torch-2.3.0 CUDA:0 (NVIDIA GeForce RTX 4070, 12282MiB)


hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup
_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hs
v_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.
0, mixup=0.0, copy_paste=0.0
ClearML: run 'pip install clearml' to automatically track, visualize and remotely train YOLOv5  in ClearML
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5  runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs\train', view at http://localhost:6006/
Overriding model.yaml nc=80 with nc=1


Transferred 343/349 items from yolov5s.pt
AMP: checks passed
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 60 weight(decay=0.0005), 60 bias
train: Scanning C:\Users\hummer\Desktop\white_dot_on_fabric\train\labels.cache... 291 images, 3 backgrounds, 0 corrupt:
val: Scanning C:\Users\hummer\Desktop\white_dot_on_fabric\valid\labels.cache... 28 images, 1 backgrounds, 0 corrupt: 10
```

Figure 29 - YOLOv5 Training Script and Execution

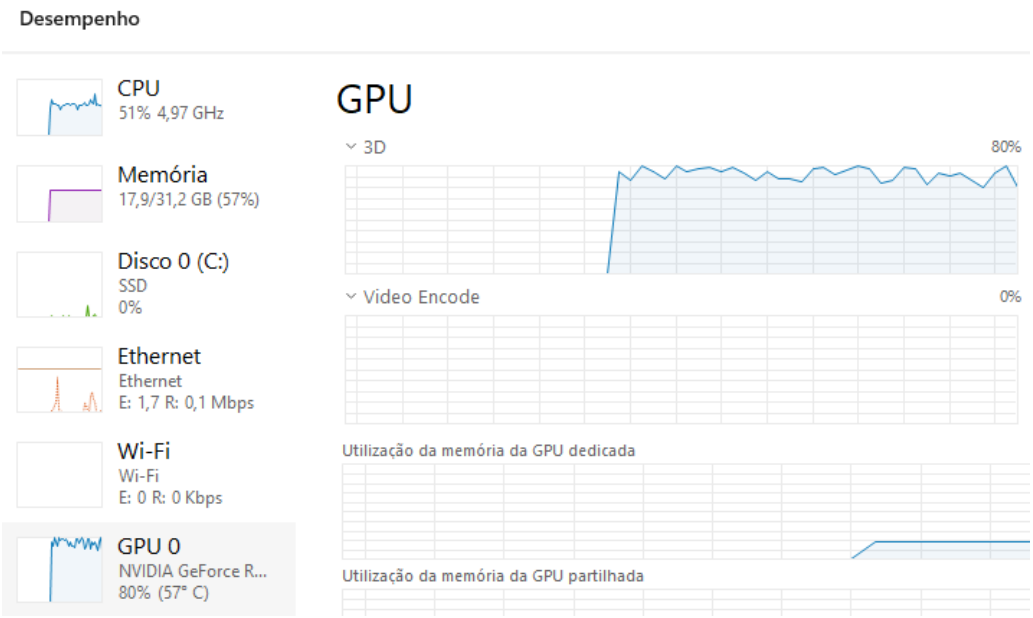


Figure 30 - GPU and CPU Performance while Training

Among the parameters listed above, YOLO offers an even wider range of hyperparameters for more complete model training, with more detailed information or customized for different tasks. Below is a short list explaining the most commonly used and important hyperparameters for machine vision model training:

- **batch**: size of the model batch. Depends on the GPU's memory allocation;
- **epochs**: number of epochs (or training steps) to train the model;

- **data**: path to the yaml file containing information about the dataset (9.3.d5);
- **workers**: number of CPU workers (optional when working with GPU);
- **cfg**: desired model architecture;
- **weights**: 'default' model on which we want to start training the new model. In the first instance, the default model would be yolov5n (nano) or yolov5s (small);
- **name**: configuration of the model's training logs;
- **hyp**: configurable yaml file with descriptions of the hyperparameter choices.

After the training script has checked all the associated GPU, image and label dependencies, as well as the availability of the default weights file, training will begin until the designated number of epochs has been reached (although it will abort further epochs if the mean average precision (mAP) has not improved since the last 100 epochs to date.

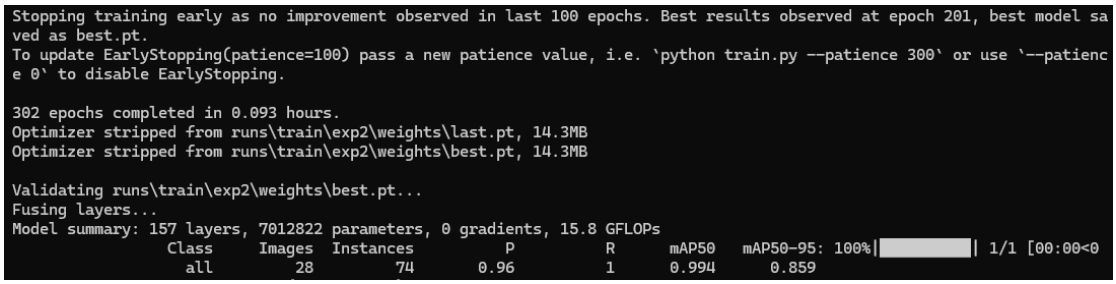


Figure 31 - Model Training Stop Example

At the end of the training, the model created will be saved in the runs subfolder within the YOLOv5 folder, in a new folder, ‘exp’, in which you will find the best.pt file, the model that achieved the best metrics throughout and until the end of the training process.

5.3.3.2.5 Model evaluation

Within the ‘exp’ subfolder (which was generated in the previous training step), visual outputs are generated through which we can check and verify in greater detail the initial conditions of the trained model as a whole (such as correlograms – figure 31) and graphs with the final training results (figure 32).

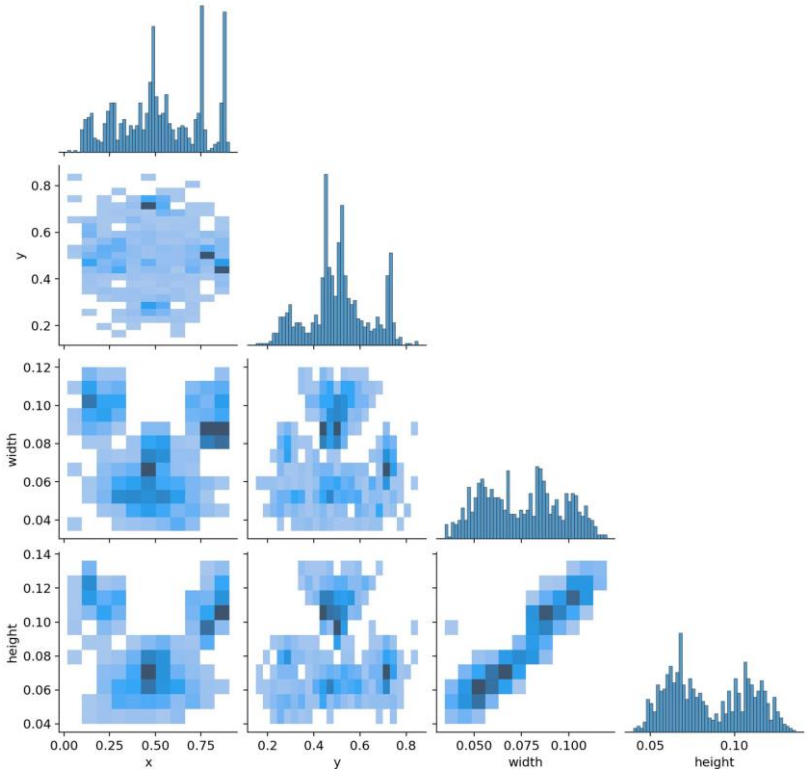


Figure 32 - Model Training Correlograms Evaluation

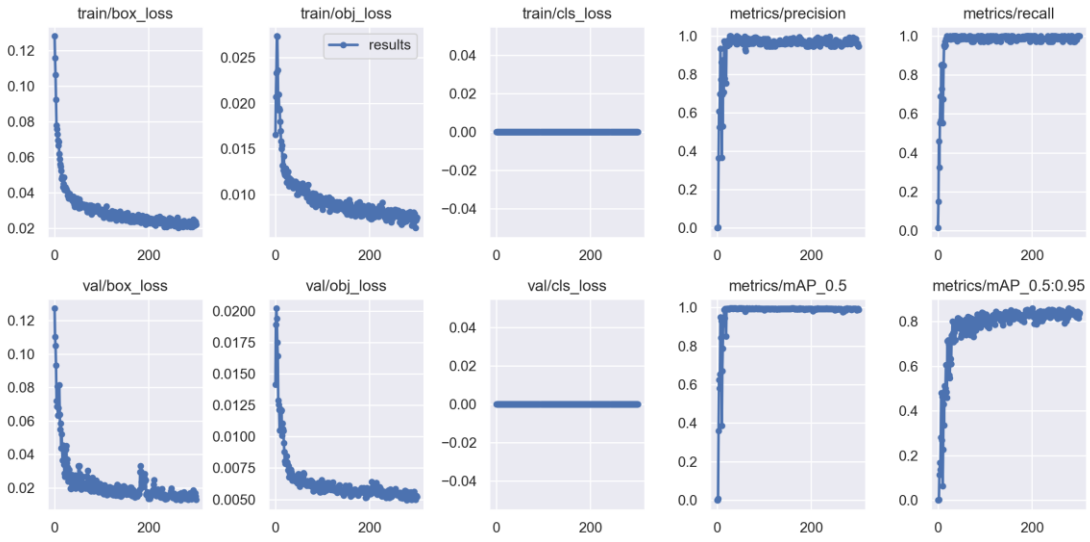


Figure 33 - Model Training Graph Evaluation

With regard to figure 33, it is important to note that, with the vision model trained, we look at the most important result metrics, Precision, Recall and Loss, two of which are directly related to the True Positive / True Negative / False Positive / False Negative terms, which are explained bellow:

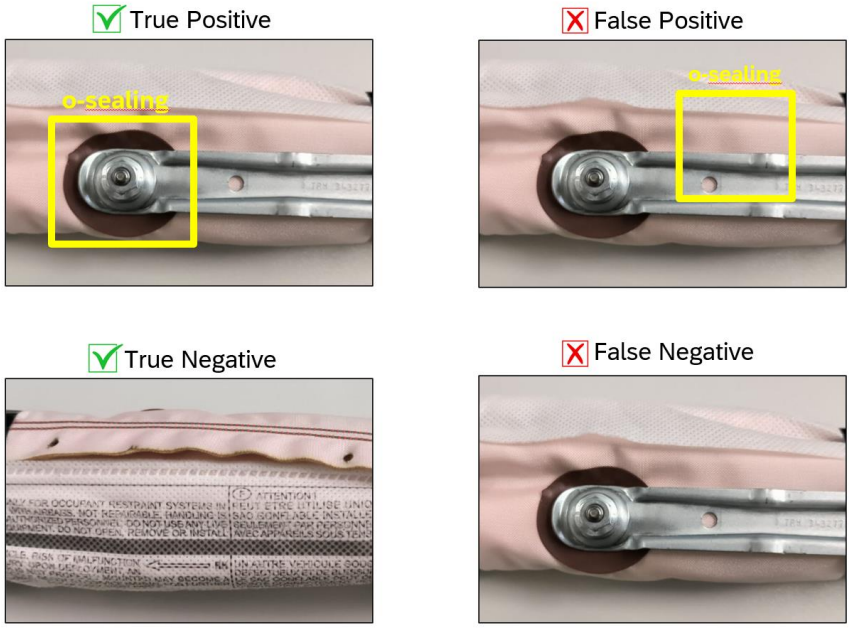


Figure 34 - Model Classification Outcomes

In figure 34, we can see the behavior of the four possible outcomes when regarding detections from a computer vision application, evaluating for just one component, o-sealing (brown rubber disk, positioned behind the metal plate). In the True Positive event, the model has verified a positive detection in the correct position of the component. In True Negative, no bounding boxes are detected in the image because there is no component to detect. In the case of False Positive, a bounding box is detected, but not in the correct place in the image, leaving the desired component waiting to be detected. Finally, in the False Negative event, no bounding box is detected in the image, even though there are components present in it.

Precision

Precision is a measure of the accuracy of the positive predictions made by the model. It is the ratio of true positive predictions (correctly predicted positive instances) to the total predicted positives (sum of true positives and false positives).

$$Precision = \frac{True\ Positives}{True\ Positives\ + False\ Positives}$$

Precision answers the question: 'Of all the instances that the model predicted as positive, how many were actually positive?'

Recall

Recall (or true positive rate) is a measure of the model's ability to find all the relevant

positive instances in the dataset. It is the ratio of true positive predictions to the total actual positives (sum of true positives and false negatives)

$$Recall = \frac{True\ Positives}{True\ Positives\ + False\ Negatives}$$

Recall answers the question: 'Of all the actual positive instances, how many did the model correctly identify?'

Loss

Loss is a measure of the error in the model's predictions. It quantifies how well or poorly the model is performing. In the context of neural networks and vision models, the loss function is used during the training process to update the model weights. Loss helps in training the model by minimizing the error between the predicted outputs and the actual labels during training.

5.3.3.2.6 Model inference

With the vision model trained and the 'best.pt' file - referring to the best with the best metrics - inside the 'exp' folder, we can finally infer the vision model trained in instance segmentation as follows:

```
cd path\to\YOLOv5
python segment\predict.py --weights <path\to\best.pt> -source <camera_index>
```

Among the parameters listed above, the following are important to note:

- **source**: Index of the camera that will be used for detection (can be 0 to n, where n is the number of cameras connected to the pc, USB or integrated)
- **weights**: absolute path to the best.pt file, hosted on the data lake, referring to the vision model in use.

Not used in this project, but relevant for a more customizable application:

- **conf-thres**: Minimum confidence threshold for any label detected in the model inference (minimum default is 0.3)
- **max-det**: Maximum detections allowed per frame, ideal for limiting excessive detections in the inference
- **device**: If you have more than one GPU installed on your computer, you can choose device '0', '1', etc, or 'cpu'.

- **classes**: In a model with several trained classes, you can initially filter by the classes '0 2 3', for example.

In our case, using the trained model we achieved positive results, in which we were able to verify the components in the image, as we can see from figure 35 below.

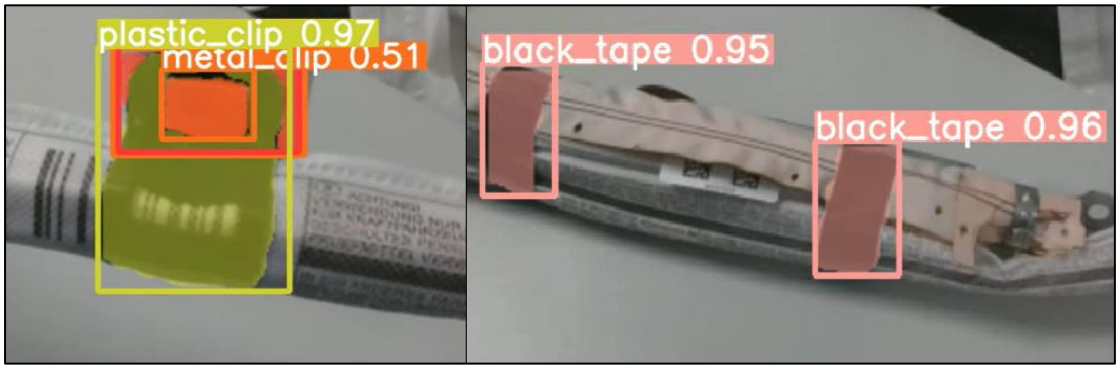


Figure 35 - Model Inference on Different Components

It is important to note that although the model inference shown in figure 34 detects the desired components quite well, its ability to do so is only achieved through the work carried out and documented in chapters 5.3.3.2.1) to 5.3.3.2.5). This is because, for there to be total generalization of the components to be detected, it is necessary to exhaustively capture them in different positions, rotations and with variations in angles and distances in positioning, as well as adjustments to the brightness and obstruction of the components and perform a detailed and attentive labelling process. By taking these steps to create a dataset for a vision model, we can prevent against only detecting components in the center of the image (if the components identified are only in a centered position), or only when the environment is very bright (if the components are only very well defined due to a well-lit environment). Likewise, exhaustive training with more than 500 epochs (or until a mAP plateau is reached) is necessary in order to achieve the best precision, recall and loss values, which are necessary for a vision model that generalizes and detects the necessary objects without problems.

From the way the dataset has been managed since its inception, we can see that the vision model has no great difficulty in detecting the desired components when the environment is well-lit, and proves to be quite resilient in detecting them when the environment is darker. This results in a model that can be used 24 hours a day on the shop floor, adapting to natural daylight, late afternoon darkness and artificial light at night, resulting in a program that does not require high maintenance to continue the manufacturing process on the assembly lines.

5.3.2.3 Integration loop definition and Integration costs

Once the vision model has been trained and evaluated, we can conclude that the integration

loop for new vision models into the assembly lines will follow the one defined in figure 35, in which the pictures captured in the first instance are annotated on the Roboflow platform (creating the dataset that will feed the training) and then exported.

Once a dataset has been created, the vision model will be trained on an office computer, equipped with an Nvidia CUDA enabled graphics card for faster and more accurate training, enabled by the parallel computing properties of the NVIDIA GPU.

The pytorch file of the vision model generated by the training will be deployed on Lenovo workstations that are installed on each production line on the shop floor, which will be called by the application made on top of the YOLOv5 instance segmentation (or bounding box detection) file.

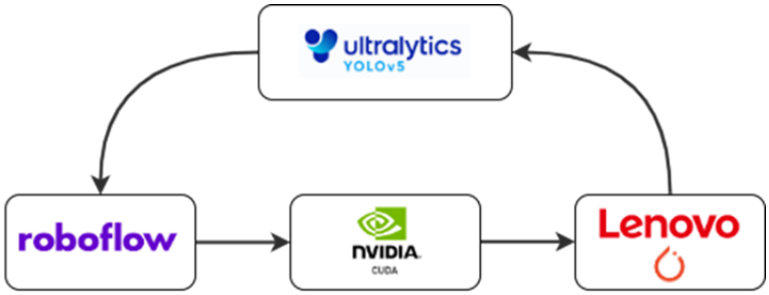


Figure 36 - Integration Loop

After the first deployment, it is expected that there will be regular monitoring of the vision model, its performance, and its effectiveness in combating the intended problem. If there are any anomalies and the vision model needs to be updated, it will be back to square one (with a new collection of images to train the vision model again).

In terms of integration costs, since the solution developed will be implemented on existing production lines, with hardware and peripherals already installed, it is necessary to take into account the budget required for the computer to be able to run the application in the desired modes - in real time and maintaining the same existing cycle times.

In terms of software, as considered in points 3.2), 5.3.2) and 5.3.3.1), it was decided to use open-source tools such as Python and Anaconda to build the application, as well as using the CUDA SDK (together with the updated GPU drivers to be installed on the computer).

In terms of hardware, it is necessary to research the number of vision models that the line will have. For lines with 1-2 vision models in use, all that is needed is an upgrade to the computer's GPU (which is non-existent in current assembly lines), as well as USB cameras and their mechanical supports. If there are no USB ports available on the current computer, a USB port expansion card is also required. Table 8 shows the associated cost specifications.

Table 8 - Computer Upgrade Cost Estimation

Material	**Unit**	**Cost**
Nvidia GTX 1650 4Gb Low Profile	1	250€
USB Expander PCI Card	1	40€
USB Webcamera High Definition	2	70€
Mechanical Supports	2	30€
Total		390€

If the assembly line expects to run more than 2 vision models at the same time, requiring resources that the existing single computer will not allow to perform well, the installation of a dedicated computer on the line will be considered. Although the costs increase exponentially, on a line where 3 or more vision models are needed, it is a cost that will bring the benefit of running the desired solutions in real time, so that they can be monitored properly as a result. In addition to a pc with a more powerful GPU than the previous one (3060 12Gb vs 1650 4Gb), USB cameras, mechanical supports and a USB port expansion card are also required, as detailed in table 9 below.

Table 9 - Dedicated Computer Upgrade Cost Estimation

Material	**Unit**	**Cost**
Lenovo Workstation	1	1200-1400€
Nvidia GTX 3060 12Gb	1	400-600€
USB Expander PCI Card	1	40€
USB Webcamera High Definition	4	210€
Mechanical Supports	4	60€
Total		1910-2310€

Regarding USB Cameras, it was noted that many camera models on the market have an intrinsic distortion effect in their image capture (which would make it impossible to correctly measure distances between certain components). Since a poorly captured image would lead to poor results, a low-budget camera was chosen that has an integrated system that self-corrects for image distortion, leading to more ‘real’ images and a more accurate measurement of distances between components.

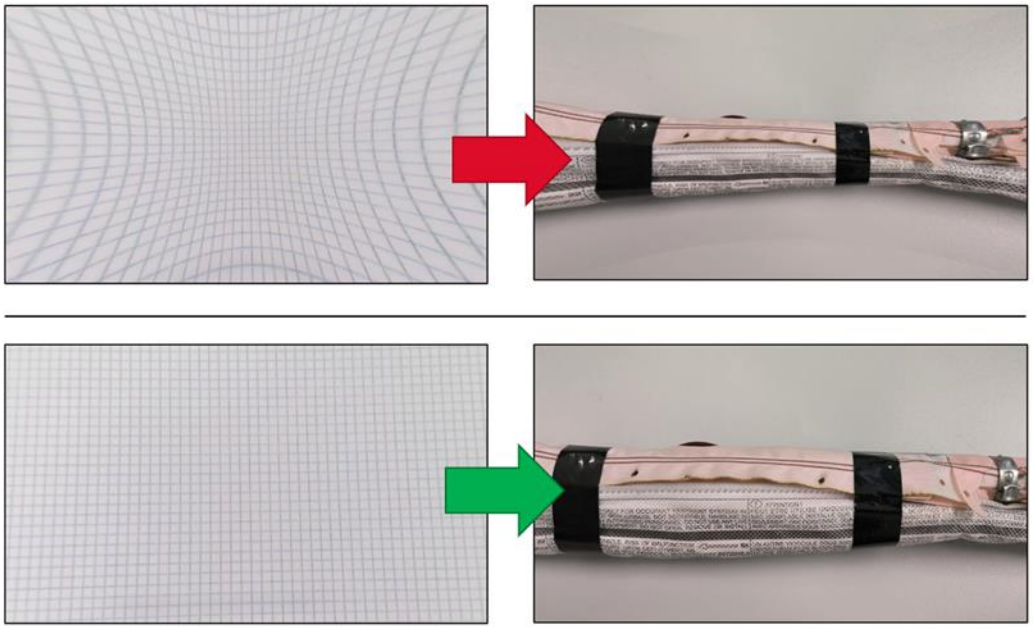


Figure 37 - Effects of Distortion (Exagerated Effect) and No Distortion in Images

As can be seen from figure 37, the effects of distortion inherent in USB cameras (especially low-budget ones) can vary in magnitude, and in the initial tests of the project only one webcam with a high degree of distortion was available. Because of this, the distances calculated between tapes ended up being very skewed due to the distortion, with distances of 30cm calculated between tapes 14cm apart. Only because of this large error margin in distance estimation was chosen to acquire new hardware for image capture.

By opting for AUSDOM webcams, which self-correct for this distortion, we get a more 'real' image, capable of obtaining images that are more faithful to reality and on which we can run the distance measurement algorithm more reliably and accurately.

5.3.2.4 Conclusion

At the end of this sprint, the important tasks of developing the vision model were completed, from the Data Collection part, through the Labelling, Model Export and Training process on a computer thanks to the computational capabilities of the locally installed GPU, making it possible to infer about the model and check its positive evaluation in the detection and identification by segmentation of different small components close to each other. The work pipelines were also defined, thus providing a standard model of how to create vision models, as well as assessing the costs inherent in modifying the assembly lines on the shop floor in the case of work in progress. Also important was the standardization and organization of a data lake to safeguard all the information inherent in the vision models created in the factory, so as to be able to update, improve or transfer vision models from one line to another with identical components.

5.3.3 Phase 3 - Smart Area Implementation and Distance Estimation

After an interval after almost four months between the last implementation stage, due to factory needs, external training and other work in the backlog, the second stage took into account the learning from the first one, in which we were able to create, train, deploy and evaluate the vision model and start building on top of the YOLOv5 vision algorithm in order to implement the desired levels of control for the desired production line. The integration of the application into the shop floor PLC infrastructure is also pending. The main work carried out at this stage is listed in the following table:

Table 10 - April to August 2023 Tasks

Task	**Importance**
Smart Area Algorithm Creation	Very High
PLC Connection	Very High
Executable Architecture	High
Database Structure	High
Distance Estimation Between Components	Medium
Performance Tunning on Vision Models	Low

5.3.3.1 Smart area implementation

In a visual control model, especially in an industrial environment where standards and process guidelines are implemented, reviewed, and audited on a regular basis, simply detecting components on an external display is not enough for the high demands of robust controls on assembly lines. If the desired component was detected in the previous sprint was a big improvement and discovery, its positional control in the airbag module is just as important, or even more so, as it is in industrial camera software. Therefore, this stage will take the instance segmentation code developed by the Ultralytics team for YOLOv5 and implement the positional controls so that it can be applied to the company's production lines, building on top of the original source code to take into consideration this component positional need.

In resume, the idea behind smart areas is to implement a 'search area' that will only validate object detection if and only if the object for which it was previously configured is within its boundaries. Figure 5.27 shows the functionality of a smart area, with three different examples.

Figure 38 - Smart Area Functionality

As illustrated in figure 38, the state of a smart area goes through three different phases:

1. **OK (green) phase**: When all the components to be detected are within its boundaries;
2. **NOK phase (red)**: When none of the components to be detected are within its boundaries;
3. **In between phase (yellow)**: When only some of the components are within its edges.

5.3.3.1.1 Database structure

To achieve the functionalities illustrated on figure 38, firstly, a table was created in SQL Server with a connection to the production network (which also supports the traceability application for modules produced), where the following fields were inserted:

- **id**: auto incremental int;
- **name**: string;
- **x**: int;
- **y**: int;
- **width**: int;
- **height**: int;
- **class1**: int;
- **class2**: int;
- **class3**: int;
- **class4**: int;
- **class5**: int;
- **line_number**: int;

- **line_op**: int;
- **plc_db**: string;
- **plc_offset**: string;
- **model**: int

In which, ‘name’ will be the name of the chosen smart area, which will contain and verify up to 5 different labels of the vision model (class1 to class5), referring to the desired assembly line number and OP (‘line_number’ and ‘line_op’), and also to the model number chosen (model number referred to in chapter 5.3.3.5) of folder structure). The position and dimensions of the smart area will be calculated in relation to the area’s initial drawing point in the application (x,y), with associated width and height. Still to be implemented, the DB and offset for communication with the PLC infrastructure are also configured for each area.

5.3.3.1.2 Tensor manipulation

By inferring the YOLOv5 script on instance segmentation, we can verify the information that each polygon provides by retrieving it by a call method, shown of figure 39.

```
tensor([ 7.35000e+02,  3.00000e+02,  8.05000e+02,  4.12000e+02,  8.26561e-01,  1.00000e+00,  5.66429e-01,  1.62278e-01,
-2.04818e-01,  8.00906e-01,  6.76602e-01,  2.66960e-01, -2.16296e-02, -2.87571e-01, -3.37936e-01,  1.11808e-01, -6.97001
e-01, -4.59540e-03, -6.64327e-01,  1.57419e-01, -1.46675e-01, -2.16916e-02,
         4.15252e-01,  1.58940e-01,  3.90911e-02, -2.12688e-03,  3.02539e-01,  2.11636e-02, -2.40038e-01, -1.99040e-01,
-1.92307e-01, -3.71538e-01,  3.41033e-01,  1.26928e-02, -7.05006e-01,  4.21718e-01, -3.39314e-02,  3.13877e-01], device=
'cuda:0')
```

Figure 39 - Tensor Retrieved Information

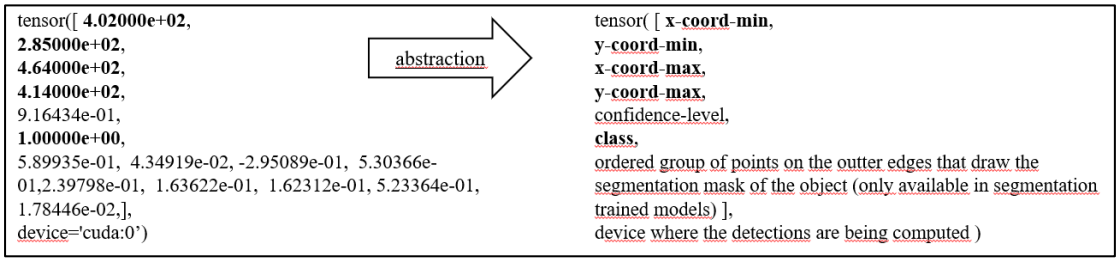


Figure 40 - Tensor Example and Abstraction

Another example of a tensor and its abstraction would be the example in figure 40, in which we see that the information contained in a tensor is grouped and returned in an orderly fashion - initial (x,y) points, final (x,y) points, degree of confidence, class number, list of external points for drawing, and the device on which the detection is being made. This order in the information makes it much accessible to extract information from each tensor (i.e., from each detection made in each frame) and contrast it with the classes expected by the smart area.

5.3.3.1.3 Smart area algorithm

With the database specifications for the creation and configuration of the areas established, as well as the manipulation and extraction of data by the tensors given by the vision algorithm achieved and understood, the following algorithm was developed for the desired solution, illustrated in figure 41 and the specification of the smart area algorithm.

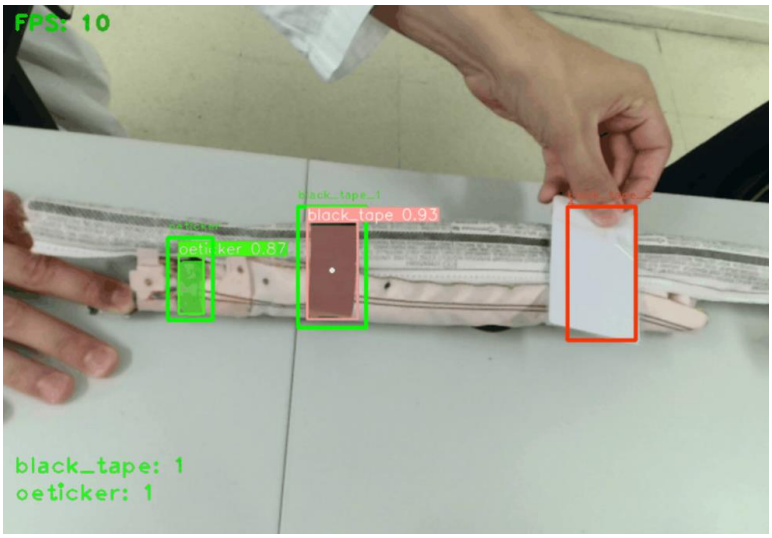


Figure 41 - Instance Segmentation Inference with Smart Vision Areas

---

For each frame: ***(15fps, for example)***
For each bounding box predicted: ***(2, at the moment of figure xxx)***
For each area loaded from database: ***(3, one red and two green)***
If (current bounding box class) is equal to (one of the classes of the current area):
Define holder to store current area; ***(holder = current area)***
If (bounding box class) is equal to (one of the classes defined in the area):
Return the class; ***(class = [detection class, True/False, area index])***
If (index of current area) is equal to (holder area index):
If (detection coordinates) is inside (area coordinates):
Add (detection instance) to (current area [ok verifications]);
For each area loaded from database: ***(second loop to assess verifications)***
If (area [ok verifications]) are equal to (area [classes]):
Turn the area green;
Write positive detection values into a txt file;
Else if (area [ok verifications]) more than 0 and less than (area [classes]):
Turn the area yellow;
Write positive and negative detection values into a txt file;
Else: ***(no detections made for current area)***
Turn the area red;
Write negative detection values into a txt file;

Reset values of verification status of each area, to repeat on next frame;

5.3.3.2 Distance estimation between components

As defined in 5.3.4.1.2, Tensor Manipulation, it is possible to access the spatial information of each object detected (bounding box, or segmented polygon) through its information tensor. In order to obtain the rough estimation of distances between specific components (in our case, the black_tape components need to be placed 14 +/- 1cm apart), an optional condition requested by OEMs in some modules produced by *AUTOProduction* on its assembly lines,

Extracting the same class information like in figure 39 and 41, the centroid of the bounding box that encompasses the segmented polygon is calculated, from the initial and final x and y coordinates. From there, the application calculates the linear distance between each of the existing centroids and draws a line between all of them, if there are any. The width of the black tape is defined in an external parameter file (such as a txt file or database registry that allows quick reading and modification, if necessary) and loaded into the program at the start of the inferences.

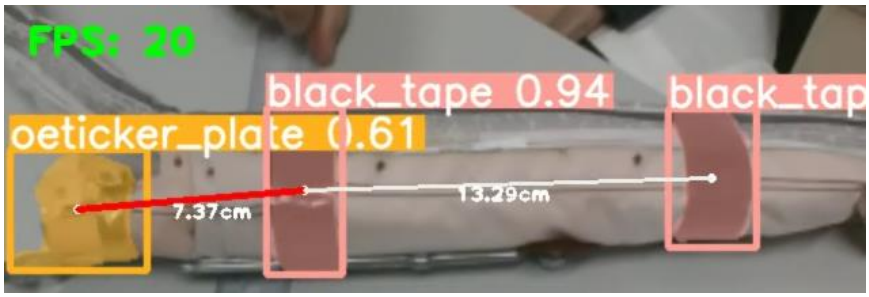


Figure 42 - Distance Measurement Between Tapes

Being one of the most ‘unstable’ points of the application developed, not being a critical element to the assembly or traceability of the part assembled, it still is one of the functionalities that needs more attention to its constraints so that the best and most real measurements can be made between the desired components.

For each measurement calculated between tapes, the tape distance is calibrated with the tape width value from the txt file or database, and by the width of the leftmost bounding box on the display (so that it can be precisely above the tape in question, reducing errors due to the curvature of the tape in the module), with all distances being directly equivalent in proportion.

It was noted that the most correct measurements were achieved when the components were arranged vertically or horizontally in relation to the camera’s field of view, with a clear start and end of the object, given by the lateral coordinates of the bounding boxes (something that is not possible with a diagonal position, which requires measuring from the vertex of the bounding box, which usually does not represent the actual start of the object). Another constraint is that

our measurement parameter (leftmost bounding box, as mentioned above) must be directly centered under the camera, as this is the point where there is the least image distortion and the least rotation in the component.

In Figure 42, we know that a tape measure 25mm in length, thus, by aligning it just below the camera (and thus avoiding image rounding inherent to the perspective focuses provided by the cameras), we can have a prediction of the proportional distance between the centroids of the components. In the image above, the real distance between the tapes was of 14.2cm on the side in display, having an error rate of about 6.5%, when all the conditions for a good measurement are achieved.

5.3.3.3 PLC connection and integration

Once the vision model and smart area algorithm have been developed, it is also necessary to connect to the PLC infrastructure, since this is where all the machine and robotic component management is done in the shopfloor.

As mentioned in the Tools, Languages and Frameworks chapter, libraries like PyPLC or python-snap7 works as a python wrapper to the Siemens Snap7 protocol, allowing python source code to communicate with devices and hardware in order to implement communication functions, data exchange and control functions.

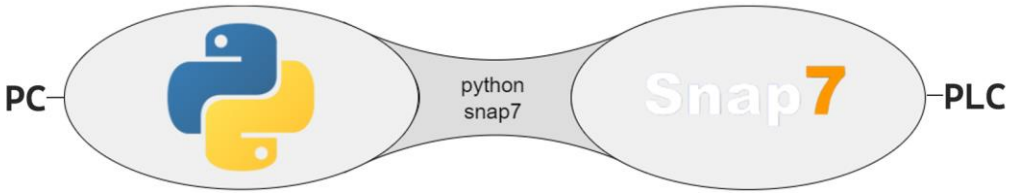


Figure 43 - Python-Snap7 Interaction between PC and PLC Infrastructures

To install python-snap7, we can simply use the following command in powershell with administration elevation, downloading and installing the most up-to-date package files with the pip install command on the python-snap7 library.

5.3.4.3.1 General configuration

In order to achieve the desired connection between the python script and the Siemens PLC, it must be configured to allow this connection. Although the python-snap7 library is compatible with the entire Siemens S7 range, in more recent models (such as the S7-1200 and S7-1500), new security measures have been installed in the system to prevent possible computer attacks, making it impossible to create important control functions in a PLC system. However, data exchange functions (such as the passing information from the python script to the PLC CPU)

using only the Data Blocks (DB) are still possible and enough with the library in question, with creating and managing SCADA functions falling out of the scope of this work.

Accessing the PLC through the TIA Portal software to create or modify a project directly on the CPU of the S7-1500 PLC (the most recent and standard in the company's requirements), we need to configure its connection properties (allow access with PUT/GET communications from a remote partner, as well as disabling optimized access to blocks, where the information passed by the solution will be stored).

Figure 44 - Permit Access with PUT/GET Communication in the PLC

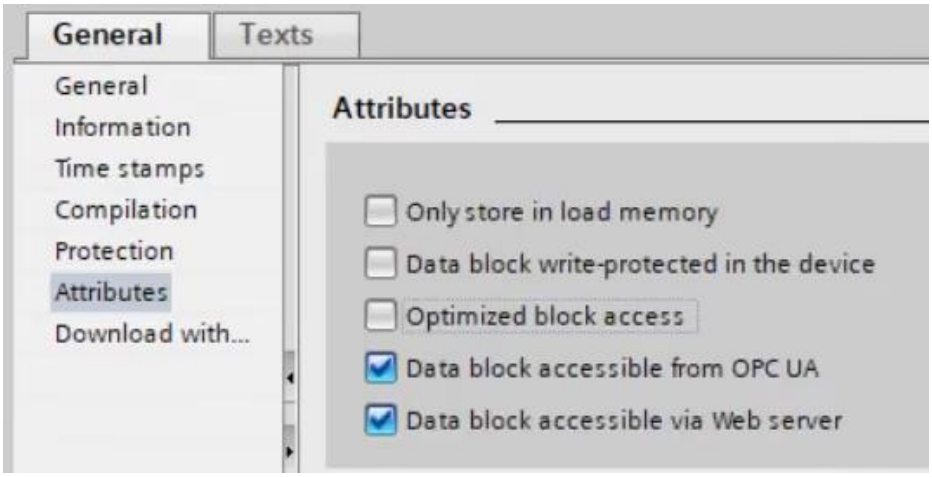


Figure 45 - Disabling Optimized Block Access in the PLC

Only in this way will our python script be able to connect to the PLC and read/write values to it.

5.3.4.3.2 PLC data block configuration

A PLC DB works similarly to a database for the PLC, since it is where various values are stored and functions are run based on them, and its structure is extremely important for PLC engineers in order to follow best practices and keep their code organized. The company standard for the structure of DB information is as follows:

IN VALUES (values given by the PLC to modify the program's behaviour):

- **In_Enable**: bool, given when the PLC is ready to execute the functions;
- **In_Reset**: bool, given by the PLC when a reset is required;
- **In_N_Program**: word, given by the PLC to modify the program to be used;

- **In_Trigger**: bool, given by the PLC to force a true trigger;

OUT VALUES (values given by Python during the program):

- **Out_Ready**: bool, given when the script is started;
- **Out_N_Program**: word, calculated by the --weights to script path;
- **Out_End_Cycle**: bool, positive value at the end of each frame;
- **Out_Result**: bool, positive value if all smart areas are green;
- **Out_Unit_Result_n**: bool, positive value and unit per area if the smart area is green.

Since the DB created has been set up without optimized block access, when compiling the program for the CPU, an 'offset' property and column will be created in the DB, which will act as an internal pointer to the DB line in question, like the ones shown in figure 46.

	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...
1	Static							
2	In_Enable	Bool	0.0	false				
3	In_Reset	Bool	0.1	false				
4	In_N_Programa	Int	2.0	0				
5	In_Trigger	Bool	4.0	false				
6	Out_Ready	Bool	4.1	false				
7	Out_N_Programa	Word	6.0	16#0				
8	Out_Fim_Ciclo	Bool	8.0	false				
9	Out_Result	Bool	8.1	false				
10	Out_Unit_Result_1	Bool	8.2	false				
11	Out_Unit_Result_2	Bool	8.3	false				
12	Out_Unit_Result_3	Bool	8.4	false				
13	Out_Unit_Result_4	Bool	8.5	false				
14	Out_Unit_Result_5	Bool	8.6	false				

Figure 46 - PLC Data Block Configuration

Since an assembly line can have several USB cameras connected to the PC, and the PC is running multiple instances of the code, it is necessary to create DBs that can hold the data that is interpreted and returned by the python script for the camera in use. In other words, if, for example, a computer is connected to 3 USB cameras, there must also be 3 DBs created in the PLC so that data can be read and written in an organized manner (although the simplest and most straightforward way is to create a series of DBs that can be used and modified in the future).

5.3.4.3.3 Managing different cameras & solution architecture

As explained in sub chapter 5.3.3.4 - Integration Costs, in order to use more cameras on a computer, it is needed a USB Port expansion card in the computer (if all the others are already in use with other peripherals, such as a mouse, keyboard, RFID Reader, ethernet adapters or RS232 adapters). In this way, it is possible to index a number to the USB camera in

use (0, 1, 2, 3, etc), something that would not be possible with the installation of a simple USB Hub. For each camera connected to the PC via USB, an instance of the developed python script (application.py) will be created (with different --weights and a different --source parameter). However, the PLC and the python-snap7 wrapper only allow the IP connection to be made once. So, as described in the smart area algorithm, the values for each camera are written to a txt file for each camera, which is then interpreted by a second python script (plc_slave.py).

With this information, we can illustrate how the application works as a whole as follows:

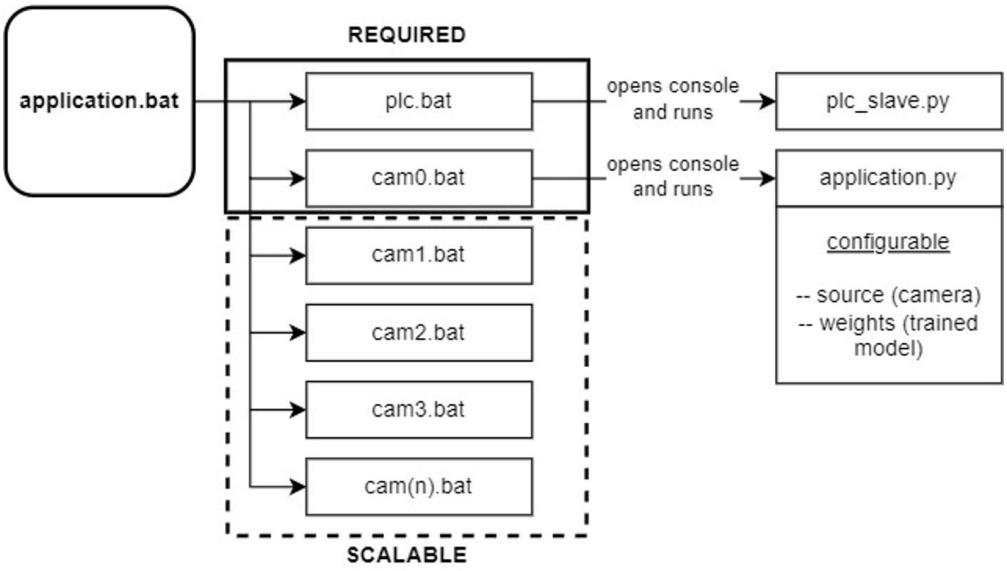


Figure 47 - Application File Requirements

As illustrated in figure 47, the application is started with the opening of a bat file, which will then run at least the plc.bat and cam0.bat files (at minimum). While the plc.bat file will open the anaconda console and run the plc_slave file (responsible for connecting to the PLC, checking values detected in txt files and reading/writing values in the PLC DBs), the cam0.bat file will create a second instance of an anaconda console and run the application.py file (configured to --source 0 and with the desired --weights). In case of some scalability being necessary (with the connection of more cameras in the assembly line), the solution will only require the enabling of the remaining bat files for the desired number of cameras. This way, when executing the bat file, the solution will create various instances of the application.py script, running the improved YOLOv5 script with a set camera and a custom weights parameter.

In depth, we can illustrate the process of starting the application, checking and writing detections and communicating to the PLC as follows:

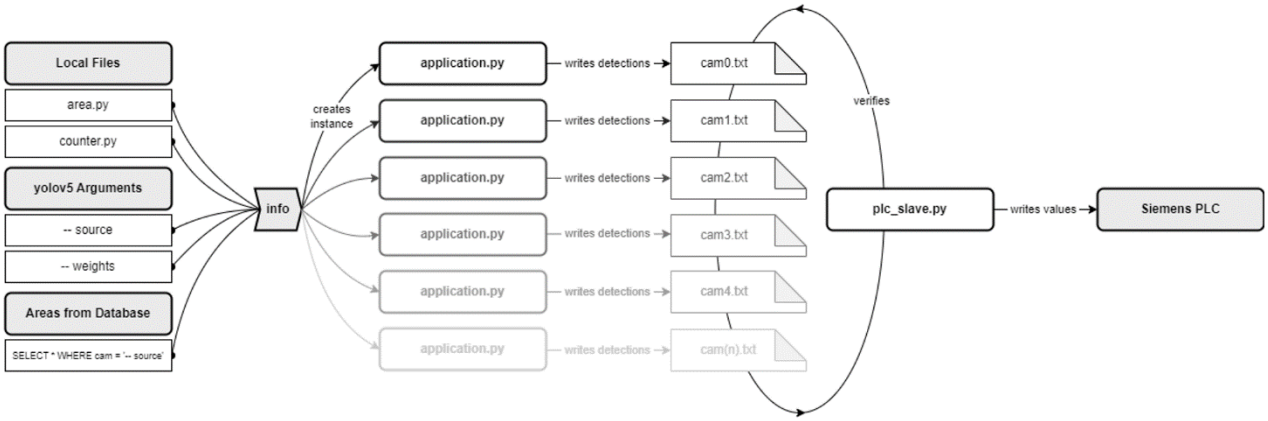


Figure 48 - Application Architecture

The functionalities of the code in the plc_slave.py file are represented by the following instructions:

Create PLC client instance;
client connects ('PLC IP', 'PLC Rack', 'PLC Slot');
Write Out_Ready value into PLC
For each area loaded from database:
    Create PLC area observer; ***(lineNumber, lineOp, plcDB, plcOffset, model in use)***
While True:
    Write '0' into bool value into Out_Fim_Ciclo;
    For each txt file in use:
        Write '0' into bool value into Out_Fim_Ciclo;
        For each PLC Observers:
            Write Out_N_Programa value into PLC; ***(weights model in use)***
            If txt file is the correspondent for the PLC Observer:
                For each area detection line in file:
                    If value is equal to 1:
                        Write '1' into bool value into Out_Unit_Result_n;
                    If value is equal to 0:
                        Write '0' into bool value into Out_Unit_Result_n;
                If (all values in file) are equal to 1:
                    Write '0' into bool value into Out_Result;
            Write '1' into bool value into Out_Fim_Ciclo;

5.3.4.3.4 Conclusion

At the end of this implementation stage, the discoveries made in the first sprint were leveraged and applied to the reality and needs of the factory and its workers. The smart area algorithm was developed (which not only detects a particular component in the image, but also determines whether it is correctly positioned), as well as defining the process for a single connection to the PLC infrastructure and functionalities on its exchanging of data. A standard was created for the creation of DBs within the PLC CPU and how the area detection information reaches it. The tool for estimating images between components was also developed, achieving positive results with a relatively low budget.

5.3.4 Phase 4 – Real World Simulation

After creating a proof-of-work application align with the company's needs, another substantial interval between tasks and implementations on the project have passed, as the factory went through a lengthy process of phasing out old lines of end-of-life products and bringing in new lines of products being launched. In March, the green light was given for the application to be implemented experimentally on a production line. Since it wasn't possible to reuse the vision model developed previously (the original line already contained cameras and photocells capable of providing the necessary detections), it was opted for a more recent line on which the final check of the airbag module was troublesome. This short subchapter will then focus on the implementation process on a real shop floor line, with the main tasks relating to this implementation and integration on it being:

Table 11 - March to May 2024 Tasks

Task	**Importance**
Determining work case requirements	High
Vision model development	High
Application implementation	High
Application monitoring	Medium

5.3.4.1 Work case

On another assembly line on the shop floor, a small module is manufactured with a black sock and white inscriptions on its fabric. In the final PO (verification), on the day of implementation, this confirmation of the inscriptions (3 dots) was done with an infrared sensor, capable of detecting inscriptions by the contrast and outline of the white dot on the fabric. However, on some of the socks supplied, the amount of white ink on the fabric is tiny and not

as well delineated as it should be. Although this can be checked with the naked eye, at infrared level it is not possible to 'find' the dots in the image to be checked when triggering the sensor, making it an ideal case for implementation.

5.3.4.2 Model vision development

Since this work case it is not a question of identifying similar or close components, nor is there a need to detect components by their contour (only by their presence or non-presence), the type of detection chosen was by bounding boxes, making the application less recursive intensive.

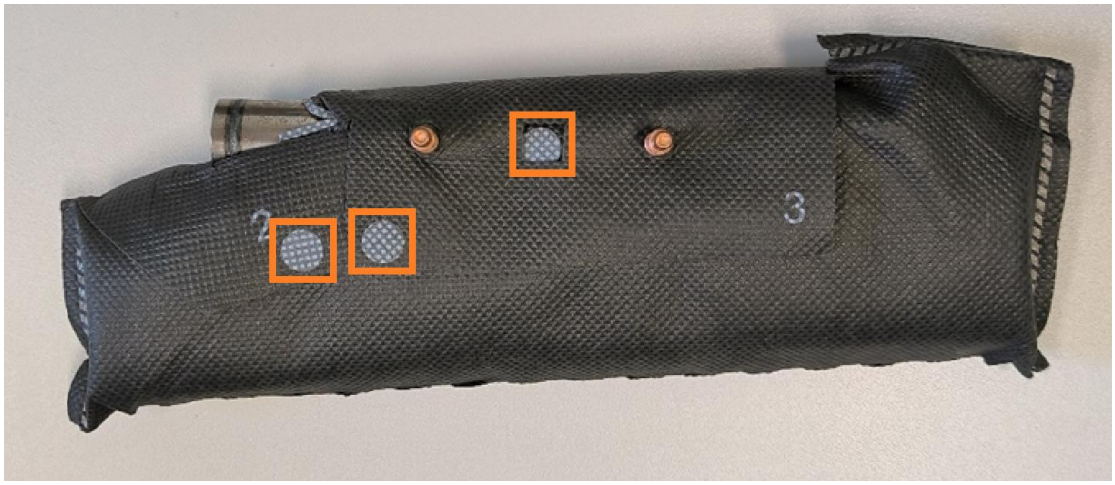


Figure 49 - White Inscriptions to Detect

As in point 5.3.3.2.1) – Data Labelling, a new project was created in Roboflow and the model was named ‘white_dot_on_fabric’, aggregating 140 images with a mix of images with 1, 2 or 3 points, or no visible points (important for training true negatives), as well as differences in background and luminosity. The dataset was annotated, data augmented, exported and trained under the RTX 4070, achieving good Precision, Recall and Loss metrics. Inferring on the new vision model, we were able to easily detect white dots on the fabric with an image frequency of around 20 FPS.

5.3.4.3 Application implementation and Application monitoring

With regard to implementation on the production line, a workstation dedicated to image processing and component verification was installed, with all the dependencies installed locally and connected to a single USB camera (fixed by a mechanical support) in the final OP. The ethernet network card was configured to be in the same IP range as the PLC CPU to ensure a

valid connection.

For the application and positional control, 3 smart areas were configured to detect 3 different white spots on the module's fabric, as well as the DB and offset settings for each area. As expected, the application runs at a stable speed, allowing real-time detections, as well as faster read and write communication with the PLC compared to the cameras and scanners used.

In terms of the application monitoring, one of the reasons for installing a dedicated computer on the assembly line in the first instance was to be able to keep the hardware and the infrared sensor operational while testing the new design. In this way, production would not be compromised in the event of a program failure.

That said, the application ran for two weeks under the supervision of engineers and technicians in order to guarantee its effectiveness, as well as improving the positioning of the camera and the brightness adjustment of the part and the PLC communication and data exchange settings. During the 10 days of production, the application was started by the engineering department at the beginning of the day, reverting to the 'old' camera and photocell sensor check at the end of the working day - to ensure that there would be no compromises to production if something failed for the night shift.

Regarding the vision model, there were no hitches or bad model detections (False Positives or False Negatives), and the most regular image failures were initially due to the USB camera being incorrectly positioned (something that was corrected in the first few hours of production).

In terms of hardware, the GPU installed in the computer maintained a steady frame rate of 15-20 FPS continuously for 8 hours of production without any visible wear and tear (despite the GPU reaching higher temperatures, around 70º). The USB camera also reached higher temperatures, becoming noticeably hot after an hour continuously feeding images, but without showing any defects to date.

At the end of the 10 days of production, and with a mostly positive evaluation from the engineers, technicians, shift leaders and workers, the solution developed was definitively implemented on the production line, removing the infrared sensor from the final check OP.

5.3.4.4 Conclusion

At the end of this implementation stage on a real assembly line, a 'review' was made of the entire process studied and developed so far. The assembly line problem was analyzed, and a beginning-to-end vision model was created according to the specifics of the component to be detected and the requirements and needs of the line and workers. The necessary hardware was provisionally installed on the assembly line and the application was activated in degraded mode to run tests in a production environment for 10 working days. With the general approval of the key players in the operation, it was decided to definitively implement the solution on the assembly line, and it has been in operation on the line ever since.

5.4 Discussion

In this design and implementation chapter, both the functional and non-functional requirements, the use case diagram presented in the previous chapter, as well as the intrinsic characteristics of the overall assembly process present on the *AUTOProduction* shopfloor, were taken as main requirements and were taken into consideration in order to design and implement the intended solution. At the beginning of the chapter, the general architecture of the system was presented, showing how the software to be developed would behave and communicate with the network of cameras to be installed, with the database, with the shopfloor operators (via the external display), and with the existing PLC network, as well as listing and showing some of the components to be identified and verified in the OP of the assembly line.

After a successful proof of concept, the basic implementation of the application was divided into three different phases:

In 2022, from October to December, the important tasks of developing a customized end-to-end vision model were carried out (from creating the dataset using the first photos of the components and annotating them for inference in instance segmentation, through exporting the dataset and followed by training the vision model using it). This first phase also saw the first creation and organization of the data lake where the data from the vision models built since then would be allocated, as well as the calculation of integration costs on an assembly line and a review of the integration loop for vision models in an industrial environment.

From April to August 2023, the built YOLOv5 source script was improved so that, inferring on the customizable models trained in the previous step, it could behave according to the initial requirements. In this way, the improvements to the YOLO code took into account the creation of smart areas for detecting and verifying the positioning of components, displaying visual indicators (such as visible component counters and FPS), as well as measurements between certain components via a rudimentary webcam. Also, in this stage of development, the connection file with the PLC network was studied and developed, which, in addition to handling the initial connection, is also responsible for reading and writing the checks obtained in real time in the DBs for the smart areas in the PLC.

In the last phase, from March to May 2024, it was possible to test the developed application in a real environment, having developed a new vision model (this time, by bounding boxes), so that the faulty process step in the assembly line could be replaced, having been monitored during a test period, obtaining very good results. This way, at the end of this test period, the developed solution could be fully implemented on a real assembly line on the factory shopfloor.

Chapter 6

Results and Evaluation

This chapter will analyze the results obtained when using the application developed to detect certain components on the assembly lines, especially when fully integrating the solution mentioned and documented in chapter 5.3.5) – March to May 2024. This evaluation takes into account whether the intended results meet the primary objectives of the work, whether the type of verification system developed is in line with the approaches implemented in the company today, how it performs on the assembly line (after an extended test period), and the general opinion of the people most involved with the systems on the shop floor (technicians, shift leaders and workers).

6.1 Component detection quality and positional control

Regarding the accuracy and effectiveness of the vision models trained in identifying components, we can see that both inferences in a test environment and in an industrial environment showed a high level of accuracy in detecting and identifying components, whether by bounding box or instance segmentation identification. Although we only check the results of the vision model in the program's inference, all the work that influences directly the results of the vision model is done in the creation of the dataset, with the capturing and labelling of different images in terms of component positioning, background and degree of luminosity, in order to guarantee a better generalization of the components to be detected.

In a test dataset, in order to correctly identify the internal material reference of the metal clips in the factory, a model was trained that was able to infer from the YOLOv5s (small) model in real time, correctly detecting and identifying the different clips by segmentation, proving that a dataset with a large number of varied images, careful data labelling (the most time-consuming part of the entire model vision development process), combined with good data augmentation techniques, are enough to recognize and identify most components, no matter how small or similar they may be.

Although the vision model detects the component in the image, being built on a mathematical and statistical model, it can never prove that a component is 100% present in the

image, or, on the contrary, that it is completely absent from the image (unlike existing camera systems, which only detect by presence/non-presence outputting a boolean value). As it is a probabilistic model, the conformity of the detections in the image collected from the dataset and the model generated must be guaranteed by the technicians and process engineers, so that there are no anomalies during production on the assembly line.

Figure 50 - Component Identification by Reference

Regarding the positional control of components in the image, its efficiency is confirmed by delimiting the area over which a given component or group of components (very close together) is accepted or not accepted. As long as all the detection areas do not have their criteria met (due to poor positioning of the module in the OP, different visibility conditions, image obstruction or poor assembly of the module), the airbag module will not have its verification cycle completed and will not be able to proceed down the assembly line, ensuring that no module unverified is ever received by the OEM. On the other hand, if the areas are 'complete' with the group of components to be detected, the connections are transmitted to the PLC to follow their assembly course.

6.2 Real time inferences comparisons

In the case work presented, the vision models were trained on the default YOLOv5n (nano) and YOLOv5s (small) models, due to their ease of detection at short/medium distances and the possibility of real-time inference, with a high number of frames per second.

However, for more 'delicate' detection tasks, with small components and/or requiring detection at a greater distance (occupying a smaller number of pixels in the image), a heavier vision model will be required, such as medium, large or extra-large, also provided in the YOLO default package.

Table 12 - YOLOv5 Benchmarks on 1650 GPU

Model	mAP val 50-95	mAP val 50	FPS
YOLOv5n	0.79	0.89	22.2
YOLOv5s	0.83	0.94	10.2
YOLOv5l	0.91	0.98	2.3
YOLOv5s-seg	0.82	0.93	17.8
YOLOv5l-seg	0.89	0.97	2.0

As described in table 12, from the data taken from csv results file, after training the vision models in bounding box and instance segmentation, with datasets identified in bounding box and instance segmentation with a wide variety of images per dataset and data augmentation performed.

Due to training with abundant data and careful annotations throughout the dataset, the results of mAP^{50-95} and mAP^{50} were even superior to the benchmark provided by Ultralytics (mentioned in sub chapter 5.3.3.1.3) - Comparison Between Default YOLOv5 Models). In this way, a custom dataset highly dedicated to a particular operation is achieved, which, depending on the level of detail and precision required, can output results in real time to the user.

6.3 PLC connection

The necessary connection to the PLC infrastructure on the shop floor is also guaranteed after testing. Although the python-snap7 library is not fully compatible with the Siemens S7-1200 and S7-1500, as it is not possible to communicate with Function Blocks and Organization Blocks, or even create functions outside the physical CPU. However, only the transfer of information to and from DBs proved necessary and enough for the scope of the project.

With regard to the latter, it was found that it provides instant communication between the python script and the CPU, making it possible to read and write new values without considerable time delays, enabling the application to quickly and continuously infer its detections and verify data in the DBs.

6.4 Limitations

As this is a new project with low-budget material to be implemented directly on the factory floor, there are some system, hardware and process limitations that need special attention.

Since the system developed will be implemented locally on the assembly lines, at least in the first instance, the installation of dependencies and software - and hardware (in the case of GPUs, as well as USB cameras), will have to be done individually and in stages. In terms of integration on the company's shop floor, this won't be a big issue or problem, since it's a lengthy process with several parties involved who need to approve and outline objectives for the solution, tasks that are time-consuming that doesn't directly influence the integration schedule. However, in terms of scalability within the group, especially in teams more focused on Research & Development and with more agile implementation cycles, a longer rollout like

this could cause some concerns.

As mentioned in point 5.3.3.4) – Integration Costs, although it was opted for a webcam that auto-corrects for image distortions, the truth is that there are always distortions associated (especially in low-end affordable ones, which the project called for), so the distance estimation will never be 100% correct. However, even with the cameras used during the project, which are capable of providing less distorted images, the application calculates distances with error percentages of around 5-15% when the parameters studied for better distance estimation are met (the leftmost component being precisely above the camera, polygonal mask of the segmented object aligning with the bounding box rectangle as much as possible and the module being in a horizontal position, so as to avoid distortions or rounding in its appearance).

Also, in terms of the production process, it was noted that the new solution causes some confusion withing the assembly line workers when performing the verification step on the final OP. This is because, with the 'old' machine vision cameras or photocells implemented to date, workers have to wait around 4-5 seconds for the verification cycle to take place (accompanied by a camera flash, sensor trigger or audible photographic click). As the implemented solution is running infinitely and does not transmit a noticeable audible or visual signal, workers were left with the idea that the application was not working or was giving an error, as 'nothing was happening' when the part was positioned in the verification OP - when in fact the verification was carried out and confirmed within the first seconds after it was positioned. Because of this, the workers were made aware of this new process of validating components on the line, and it will have to be part of the workers' training and part of the assembly line process documentation when the application is installed on the other lines and OPs.

Chapter 7

Conclusion and future work

In conclusion, we can say that the objectives of this research and development work have been met and exceeded. In this way, in addition to studying within the computer vision field its main algorithms, developing a custom vision model for an industrial environment and integrating it with a PLC infrastructure, it was also possible to integrate the proposed solution into a real assembly line on *AUTOProduction*s's factory shopfloor, something that was not originally part of the scope of this project.

Due to the evolution of processes, products and standards imposed on suppliers in the automotive sector, companies like *AUTOProduction* are currently in a transitional period, from old-fashioned manufacturing methods (such as using more manual labor, visual inspections or outdated technology) to recent proposals encouraged by Industry 4.0 (which encourages the use of more robotization, inspection checks with modern cameras and scanners, along with many others), in order to remain competitive with other companies in an ever-changing market.

In an attempt to solve the problem of inspecting and checking components on assembly lines in order to gradually replace expensive cameras, scanners and photocells, the proposed solution was to develop an application supported by machine learning tools to carry out these detections and checks. As this is an idea for the complete replacement of old hardware like cameras and sensors, this work took into account all the parts involved in the assembly process, from the necessary hardware, development of the machine vision model, factory and worker needs, system architecture and data organization, as well as connections to other infrastructures, such as PLC networks and production networks.

The computer vision algorithm chosen was YOLOv5, which, despite having undergone numerous updates and versions, is still today the most documented version and offers a simple integration with Python, the programming language of choice. Along this easy to use and easy to integrate functionalities, the YOLO algorithm is the most notorious in the open-source computer vision algorithms field one, enabling real-time detections compared to other vision algorithms on the market, thus making it possible to reduce seconds on an assembled piece cycle time (or minutes, in a continuous production), needed on a production line during the working day.

Alongside the model vision development, the software developed was built with a budget-

friendly vision, opting for open-source technologies that didn't involve user costs or licensing costs (as is the norm with proprietary machine-vision products), also opting for inexpensive material in which the cost-benefit for the project was leveraged (as was the case with USB cameras with auto distortion correction and the NIVIDIA 1650 GPU's, needed to run the application and provide data to the shopfloor operators on the production line in real time).

Given the needs and requirements of the *AUTOProductions's* factory, the connection to the PLC was also developed, tested, and fully implemented at the end of the work. In this way, all the automation and safety regulations can be ensured with the application, as well as ensuring the reading and transfer of data relating to component detections in real time between the application and the PLC infrastructure.

Having chosen the Design Science Research methodology at the start of the project, its development followed according to its proposed activities. The main motivation was to create cheaper software to detect small components on assembly lines, defining its objectives according to the needs of the factory to be implemented, its processes and the main agents (such as engineers, technicians, and workers). Its design and development were carried out using the source code of the YOLOv5 framework, which was modified and improved in order to obtain positioning checks in addition to the 'simpler' identification in the image, as well as ensuring connection and communication with the existing PLC network. The demonstration was carried out both in a test context, in laboratory, and in an industrial environment, on a real assembly line on the shop floor. The results were also evaluated and validated during the test period, during which the software developed was subject to validation and scrutiny by other departments, through assembly tests and lighting variations on the line. For the communication of the developed solution, the problem, solution, and its results were also shared, submitted and approved in a research paper which was published at the 2nd EPIA 2024[6] - International Conference on Artificial Intelligence.

In short, this work serves as an entry document and validation of the project's functionalities for the company in which it was developed, proving its usefulness, inherent low cost and time efficiency in its detections, and could revolutionize the way in which this type of varied checks is carried out on the factory floor – both in the factory and in the global group in which it operates.

In terms of future work, as this is still a recent project with only one integration into the *AUTOProduction's* assembly line, there are still many improvements to be made in the short and medium term, in order to guarantee the good use, better monitoring and ease of scalability of the application within the company and group in which it operates.

In the future, one of the tasks will be to create a local alternative to the Roboflow service, the chosen website for labelling images and managing projects and datasets. In order not to lose the information and work on the platform in case of it shutting down, an alternative will need to be implemented in-house or opting for a local open-source project like [100], so that it can annotate images and perform data augmentation without associated costs.

In terms of energy, the application's continuous inference during the working day is also

[6] EPIA 2024 - https://epia2024.pt/

an important point to keep in mind, since the energy cost of having an additional computer running a vision program can lead to increased costs (albeit small, compared to the *AUTOProduction's* daily energy costs). In any case, an analysis will be carried out by implementing an activity observer on the assembly line (in the plc_slave.py file), which will check whether the line is working within a time period (by reading other values from the PLC) and will shut down the application as long as there is no activity on it.

In terms of network infrastructure, a solution will be devised and negotiated to have just one group of central computers running the vision algorithm and transmitting the output to the assembly line computers as a service, without the need for mandatory upgrades for each line and guaranteeing a centralization of resources for the project in the long run.

Workers and shift leaders will also have to be re-educated about the processes that will change with the new implementations on the production lines, since, with continuous inferences from the application, there will no longer be a waiting time of 4 to 5 seconds between positioning the part and the final check or visual or audio cues - since with the solution presented, the check is made when the part is positioned in the OP.

Bibliography

[1] Bekmurzaeva, Rashiya & Kovalev, G.S. (2023). Industry 4.0: The Fourth Industrial Revolution. SHS Web of Conferences. 172. https://doi.org/10.1051/shsconf/202317202011

[2] Philbeck, T., & Davis, N. (2018). THE FOURTH INDUSTRIAL REVOLUTION: SHAPING A NEW ERA. Journal of International Affairs, 72(1), 17–22. Available on: https://www.jstor.org/stable/26588339. (Accessed in 14-06-2024).

[3] Secretary of State for Business, Energy and Industrial Strategy, Gov.UK (2019). Regulation for the Fourth Industrial. Policy paper, Gov.UK. Available on: https://www.gov.uk/government/publications/regulation-for-the-fourth-industrial-revolution/regulation-for-the-fourth-industrial-revolution. (Accessed in 14-06-2024).

[4] Ultralytics. (n.d.). Architecture Resume. Available on: https://docs.ultralytics.com/pt/yolov5/tutorials/architecture_description. (Accessed in 14-06-2024).

[5] James, G., Shanahan., Liang, Dai. (2020). Introduction to Computer Vision and Real Time Deep Learning-based Object Detection. https://doi.org/10.1145/3394486.3406713

[6] Prijs, J., Liao, Z., Ashkani-Esfahani, S., Olczak, J., Gordon, M., Jayakumar, P., Jutte, P. C., Jaarsma, R. L., IJpma, F. F. A., Doornberg, J. N., & Machine Learning Consortium (2022). Artificial intelligence and computer vision in orthopaedic trauma : the why, what, and how. The bone & joint journal, 104-B(8), 911–914. https://doi.org/10.1302/0301-620X.104B8.BJJ-2022-0119.R1

[7] Kari, Pulli., Anatoly, Baksheev., Kirill, Kornyakov., Victor, Eruhimov (2012). Real-time computer vision with OpenCV. Communications of The ACM, https://doi.org/10.1145/2184319.2184337

[8] Cruz, E. F., & da Cruz, A. M. R. (2020). Design Science Research for is/it projects: Focus on digital transformation. In 2020 15th Iberian Conference on Information Systems and Technologies (CISTI) (pp. 1-6). DOI: https://doi.org/0.23919/CISTI49556.2020.9140972

[9] Hevner, A., & Chatterjee, S. (2010). Design research in information systems: theory and practice (Vol. 22). Springer Science & Business Media. https://doi.org/10.1007/978-1-4419-5653-8

[10] Strauss, M., Lang, S. (2011). Agile Process Management in an Industrial R&D Department. DOI: https://doi.org/10.1007/978-3-642-23471-2_17

Trivedi, K.S. (2023). Computer Vision. In: Microsoft Azure AI Fundamentals Certification Companion. Certification Study Companion Series. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-9221-1_4

[11] Sharifani, K., & Amini, M. (2023). Machine learning and deep learning: A review of methods and applications. World Information Technology and Engineering Journal, 10(07), pp:3897-3904.

[12] Shivangi Jain, Vandana jagtap, Nitin Pise (2015). Computer Aided Melanoma Skin Cancer Detection Using Image Processing. https://doi.org/10.1016/j.procs.2015.04.209

[13] Alexander Selvikvåg Lundervold, Arvid Lundervold (2019) An overview of deep learning in medical imaging focusing on MRI. https://doi.org/10.1016/j.zemedi.2018.11.002

[14] F. Sun, Z. Li, Z. Li (2021). A traffic flow detection system based on YOLOv5, https://doi.org/10.1109/AINIT54228.2021.00095

[15] D. -L. Nguyen, X. -T. Vo, A. Priadana and K. -H. Jo (2023). Vehicle Detector Based on Improved YOLOv5 Architecture for Traffic Management and Control Systems. https://doi.org/10.1109/IECON51785.2023.10311764

[16] James, N., Antony, N. T., Shaji, S. P., Baby, S., & Annakutty, J. (2021). Automated checkout for stores: A computer vision approach. REVISTA GEINTEC-GESTAO INOVACAO E TECNOLOGIAS, 11(3), 1830-1841.

[17] M. Jindal, N. Raj, P. Saranya and S. V (2022). Aircraft Detection from Remote Sensing Images using YOLOV5 Architecture, 2022 6th International Conference on Devices, Circuits and Systems (ICDCS), Coimbatore, India, 2022, pp. 332-336. https://doi.org/10.1109/ICDCS54290.2022.9780777

[18] Deriche, R. (1987). Using Canny's criteria to derive a recursively implemented optimal edge detector. International journal of computer vision, 1(2), 167-187. https://doi.org/10.1007/BF00123164

[19] Boesch, G. (2024, June 14). Object detection in 2024: The definitive guide. viso.ai. Available on: https://viso.ai/deep-learning/object-detection. (Accessed in 14-06-2024).

[20] Xu, X., Zhao, M., Shi, P., Ren, R., He, X., Wei, X., & Yang, H. (2022). Crack detection and comparison study based on faster R-CNN and mask R-CNN. Sensors, 22(3), 1215. https://doi.org/10.3390/s22031215

[21] A. S, S. A and N. B. S (2023). Comparison of Faster RCNN and YOLO V3 for Video Anomaly Localization, 2023 International Conference on Power, Instrumentation, Control and Computing (PICC), Thrissur, India, 2023, pp. 1-4. https://doi.org/10.1109/PICC57976.2023.10142815

[22] Neubeck, A., & Van Gool, L. (2006). Efficient non-maximum suppression. In 18th international conference on pattern recognition (ICPR'06) (Vol. 3, pp. 850-855). https://doi.org/10.1109/ICPR.2006.479

[23] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788). https://doi.org/10.1109/CVPR.2016.91

[24] Ciprian, Orhei., Silviu, Vert., Muguras, Mocofan., Radu, Vasiu. (2021). End-To-End

Computer Vision Framework: An Open-Source Platform for Research and Education. Sensors. https://doi.org/10.3390/S21113691

[25] Liu B, Gao F, Li Y. Cost-Sensitive YOLOv5 for Detecting Surface Defects of Industrial Products. Sensors. 2023; 23(5):2610. https://doi.org/10.3390/s23052610

[26] Le, Hai & Zhang, Lu & Liu, Yan. (2022). Surface Defect Detection of Industrial Parts Based on YOLOv5. IEEE Access. PP. 1-1. https://doi.org/10.1109/ACCESS.2022.3228687

[27] Zhou, Longfei & Zhang, Lin & Konz, Nicholas (2023). Computer Vision Techniques in Manufacturing. https://doi.org/10.1109/tsmc.2022.3166397

[28] Srishti, Shetty. (2023). Computer Vision for Industrial Safety and Productivity. https://doi.org/10.1109/CSCITA55725.2023.10104764

[29] Swarit, Anand, Singh., Aitha, Sudheer, Kumar., Pratik, C., Sorathiya., K.A., Desai. (2022). Vision-Sensor Fusion-Based Low-Cost Dimension Measurement System for Machining Shop Floor. Volume 1. https://doi.org/10.1115/msec2022-85442

[30] Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, Santamaría J, Fadhel MA, Al-Amidie M, Farhan L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data. 2021;8(1):53. https://doi.org/10.1186/s40537-021-00444-8

[31] Fariba, Lotfi., Fatemeh, Tohidian., Mansour, Jamzad., Hamid, Beigy. (2023). Image Data Augmentation and Convolutional Feature Map Visualizations in Computer Vision Applications. Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-031-26507-5_2

[32] K. Nanthini, D. Sivabalaselvamani, K. Chitra, P. Gokul, S. KavinKumar and S. Kishore. (2023). A Survey on Data Augmentation Techniques. In 7th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2023, pp. 913-920. https://doi.org/10.1109/iccmc56507.2023.10084010

[33] Faltings, U.; Bettinger, T.; Barth, S.; Schäfer, M. (2022). Impact on Inference Model Performance for ML Tasks Using Real-Life Training Data and Synthetic Training Data from GANs. 2022, 13, 9. https://doi.org/10.3390/info13010009

[34] X. Dai, X. Zhao, F. Cen and F. Zhu (2022). Data Augmentation Using Mixup and Random Erasing. In IEEE International Conference on Networking, Sensing and Control (ICNSC), Shanghai, China, 2022, pp. 1-6. https://doi.org/10.1109/icnsc55942.2022.10004083

[35] Garcea, Fabio & Serra, Alessio & Lamberti, F. & Morra, Lia. (2022). Data augmentation for medical imaging: A systematic literature review. Computers in Biology and Medicine. 152. https://doi.org/10.1016/j.compbiomed.2022.106391

[36] M. Masud, P. Vazquez, M. R. U. Rehman, A. Elahi, W. Wijns and A. Shahzad. (2023). Measurement Techniques and Challenges of Wireless LC Resonant Sensors: A Review. doi: 10.1109/ACCESS.2023.3309300

[37] Robert, Beverly., Georgios, Smaragdakis., Anja, Feldmann. (2018). Passive and Active Measurement. https://doi.org/10.1007/978-3-319-76481-8

[38] Vajgl, M.; Hurtik, P.; Nejezchleba, T. (2022) Dist-YOLO: Fast Object Detection with Distance Estimation. Appl. Sci. 2022, 12, 1354. https://doi.org/10.3390/app12031354

[39] Thuan, D. (2021). Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detention algorithm.

[40] Terven J, Córdova-Esparza D-M, Romero-González J-A. (2023). A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. Machine Learning and Knowledge Extraction. 2023. 5(4):1680-1716. https://doi.org/10.3390/make5040083

[41] Redmon, Joseph & Farhadi, Ali. (2017). YOLO9000: Better, Faster, Stronger. 6517-6525.https://doi.org/10.1109/CVPR.2017.690

[42] Lijuan, Liu., Yanping, Wang., Wanle, Chi. (2020). Image Recognition Technology Based on Machine Learning. IEEE Access. https://doi.org/10.1109/ACCESS.2020.3021590

[43] Kamal, A. (2021, November 12). Yolo, Yolov2 and Yolov3: All you want to know. Medium. Available on: https://amrokamal-47691.medium.com/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899. (Accessed in 14-06-2024).

[44] Redmon, J.; Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv:1804.02767. https://doi.org/10.48550/arXiv.1804.02767

[45] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934. https://doi.org/10.48550/arXiv.2004.10934

[46] Ultralytics. (2022). Ultralytics/yolov5: Yolov5. GitHub. https://github.com/ultralytics/yolov5. (Accessed in 14-06-2024).

[47] Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., ... & Wei, X. (2022). YOLOv6: A single-stage object detection framework for industrial applications. arXiv preprint arXiv:2209.02976. https://doi.org/10.48550/arXiv.2209.02976

[48] Feng, C., Zhong, Y., Gao, Y., Scott, M. R., & Huang, W. (2021, October). Tood: Task-aligned one-stage object detection. In 2021 IEEE/CVF International Conference on Computer Vision (ICCV) (pp. 3490-3499). IEEE Computer Society. https://doi.org/10.1109/ICCV48922.2021.00349

[49] Shu, C., Liu, Y., Gao, J., Yan, Z., & Shen, C. (2021). Channel-wise knowledge distillation for dense prediction. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 5311-5320). https://doi.org/10.48550/arXiv.2011.13256

[50] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2023). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 7464-7475). https://doi.org/10.48550/arXiv.2207.02696

[51] Wang, C. Y., Yeh, I. H., & Liao, H. Y. M. (2021). You only learn one representation: Unified network for multiple tasks. arXiv preprint arXiv:2105.04206. https://doi.org/10.48550/arXiv.2105.04206

[52] Ultralytics. (2024, June 20). YOLOV9. YOLOv9 - Ultralytics YOLO Docs. Available on: https://docs.ultralytics.com/models/yolov9/#information-bottleneck-principle. (Accessed in 14-06-2024).

[53] Snegireva, D., & Perkova, A. (2021, September). Traffic sign recognition application using yolov5 architecture. In 2021 International Russian Automation Conference (RusAutoCon)

(pp. 1002-1007). IEEE. https://doi.org/10.1109/RusAutoCon52004.2021.9537355

[54] Tsang, S.-H. (2023, June 11). Brief review: Yolov5 for object detection. Medium. Available on: https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a. (Accessed in 14-06-2024).

[55] Lut, M., Abd Latib, L., Ayob, M. A., & Rohaziat, N. (2023). YOLOv5 Models Comparison of Under Extrusion Failure Detection in FDM 3D Printing. In 2023 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS) (pp. 39-43). IEEE. https://doi.org/10.1109/I2CACIS57635.2023.10193388

[56] Zhang, A., & Zhu, X. (2023). Research on ship target detection based on improved YOLOv5 algorithm. In 2023 5th International Conference on Communications, Information System and Computer Engineering (CISCE) (pp. 459-463). IEEE.

[57] Liu, H., Sun, F., Gu, J., & Deng, L. (2022). Sf-yolov5: A lightweight small object detection algorithm based on improved feature fusion mode. Sensors, 22(15), 5817. https://doi.org/10.3390/s22155817

[58] Yan, X., Ding, S., & Shi, W. (2023). MobileNetV3-YOLOv5-based Network Model for Pedestrian Detection. In 2023 IEEE 10th International Conference on Cyber Security and Cloud Computing (CSCloud)/2023 IEEE 9th International Conference on Edge Computing and Scalable Cloud (EdgeCom) (pp. 144-149). IEEE. https://doi.org/10.1109/CSCloud-EdgeCom58631.2023.00033

[59] Wu, S., Wang, J., Liu, L., Chen, D., Lu, H., Xu, C., ... & Wang, Q. (2023). Enhanced YOLOv5 Object Detection Algorithm for Accurate Detection of Adult Rhynchophorus ferrugineus. Insects, 14(8), 698. https://www.mdpi.com/2075-4450/14/8/698

[60] Li Z, Song J, Qiao K, Li C, Zhang Y, Li Z. (2022). Research on efficient feature extraction: Improving YOLOv5 backbone for facial expression detection in live streaming scenes. Front Comput Neurosci. 2022 Aug 10;16:980063. https://doi.org/10.3389/fncom.2022.980063

[61] Ahuja, D., & Chaudhary, N. (2012). Programmable Logic Controller. International Journal of Information and Computer Science, 1, 115-120. Available on: https://www.researchgate.net/publication/338539915_Programmable_logic_controller/citation/download. (Accessed in 14-06-2024).

[62] Yu, Xia, Lu. (2022). Application analysis of PLC techniques in automatic production line of automobile. https://doi.org/10.1117/12.2645715

[63] Coroiu, Laura., Linawati, Linawati. (2023). Versatile Control of an Automotive Assembly Line by using Programmable Logic Controllers. 2023 17th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, Romania, 2023, pp. 1-4 https://doi.org/10.1109/EMES58375.2023.10171789

[64] Rubio-Manrique, S., Cuní, G., Fernandez-Carreiras, D., Reszela, Z., & Rubio, A. PyPLC, A Versatile PLC-to-PC python interface. PCaPAC. Available on: https://accelconf.web.cern.ch/PCaPAC2014/papers/fpo011.pdf. (Accessed in 14-06-2024).

[65] C. M. Bishop (1994). Neural networks and their applications, Review of Scientific Instruments, pp. 1803-1832. https://doi.org/10.1063/1.1144830

[66] Mamidi, Sai, Akash., Fiitjee. (2015). Artificial intelligence & neural networks. doi: IJSCAI-IRAJ-DOIONLINE-3369. Available on: https://www.digitalxplore.org/up_proc/pdf/178-144281240253-60.pdf. (Accessed in 14-06-2024).

[67] S.-i. Amari (1993). Backpropagation and stochastic gradient descent method, Neurocomputing, pp. 185-196. https://doi.org/10.1016/0925-2312(93)90006-O

[68] Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network upGrad blog. (n.d.). Basic CNN architecture: Explaining 5 layers of Convolutional Neural Network. upGrad blog. Available on: https://www.upgrad.com/blog/basic-cnn-architecture. (Accessed in 14-06-2024).

[69] Ahmad, Fauzi., Sarifuddin, Madenda., Ernastuti., Eri, Prasetyo, Wibowo., Anis, Fitri, Nur, Masruriyah. (2020). The Importance of Bounding Box in Motion Detection. 2020 Fifth International Conference on Informatics and Computing (ICIC), Gorontalo, Indonesia, 2020, pp. 1-5. https://doi.org/10.1109/ICIC50835.2020.9288604

[70] N. Ottakath and S-Al-Maadeed (2023). Vehicle Instance Segmentation Polygonal Dataset for Private Surveillance System. Sensors 2023, 23, 3642. https://doi.org/10.20944/preprints202212.0475.v3

[71] P., Raja., Dr.S.Senthil, kumar., Digvijay, Yadav., Dr., Taresh, Singh. (2023). The Internet of Things (IOT): A Review of Concepts, Technologies, and Applications. International journal of information technology and computer engineering. nternational Journal of Information Technology &Amp; Computer Engineering (IJITC) ISSN : 2455-5290, 3(02), 21–32.https://doi.org/10.55529/ijitc.32.21.32

[72] Martyna, E., Górska. (2023). Internet of Things (IoT). https://doi.org/10.1007/978-3-031-27765-8_6

[73] Julian, Soh., Priyanshi, Singh. (2020). Machine Learning Operations. Pring Verlag Editors. https://doi.org/10.1007/978-1-4842-6405-8_8

[74] Gupta, S. C. (2022). Mlops: Machine learning life cycle. Machine Learning for Developers. Available on: https://www.ml4devs.com/articles/mlops-machine-learning-life-cycle/. (Accessed in 14-06-2024).

[75] Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Müller, K. R. (2021). Towards CRISP-ML (Q): a machine learning process model with quality assurance methodology. Machine learning and knowledge extraction, 3(2), 392-413. https://doi.org/10.3390/make3020020

[76] Vinu, Sankar, Sadasivan., Mahdi, Soltanolkotabi., Soheil, Feizi. (2023). CUDA: Convolution-based Unlearnable Datasets. arXiv.org, https://doi.org/10.48550/arXiv.2303.04278

[77] Dwyer, B., Nelson, J., Hansen, T., et. al. (2024). Roboflow (Version 1.0) [Software]. Available on: https://roboflow.com

[78] Saphalya, Peta. (2022). Python- An Appetite for the Software Industry. International journal of programming languages and applications, https://doi.org/10.5121/ijpla.2022.12401

[79] Aniket, M., Wazarkar. (2021). Python: A Quintessential approach towards Data Science.

International Journal for Research in Applied Science and Engineering Technology. https://doi.org/10.22214/IJRASET.2021.35683

[80] Fernando, Richter, Vidal., Naghmeh, Ivaki., Nuno, Laranjeiro. (2023). OpenSCV: An Open Hierarchical Taxonomy for Smart Contract Vulnerabilities. arXiv.org, https://doi.org/10.48550/arXiv.2303.14523

[81] Muthu, Kumar., Rahul, Bhatt. (2022). An approach towards Real-Time Object Detector Using Open CV. https://doi.org/10.1109/SMART55829.2022.10046751

[82] Bo, Zheng., Xu, Junjie., Li, Helin., Xing, Junwei., Huadong, Zhao., Guoning, Liu. (2017). Development of Remotely Monitoring and Control System for Siemens 840D sl NC Machine Tool Using Snap 7 Codes. https://doi.org/10.2991/EAME-17.2017.26

[83] Nardella, D. (2016). Snap7 homepage. Snap7 Homepage. Available on: https://snap7.sourceforge.net/ . (Accessed in 14-06-2024).

[84] Introduction. python. (2019). Available on: https://python-snap7.readthedocs.io/en/latest/introduction.html. (Accessed in 14-06-2024).

[85] Junchen, Ou., Lin, Sha. (2019). Design of automotive brakejoint size detection system based on TIA Portal. https://doi.org/10.1109/EITCE47263.2019.9094820

[86] Henry, Hui., Kieran, McLaughlin. (2018). Investigating Current PLC Security Issues Regarding Siemens S7 Communications and TIA Portal. https://doi.org/10.14236/EWIC/ICS2018.8

[87] Hiroshi, Ishimoto. (2022). Introduction to PyTorch, Tensors, and Tensor Operations. https://doi.org/10.1007/978-1-4842-8925-9_1

[88] Junan, Pan., Zhihao, Zhao. (2022). Deep Autoencoder Model Construction Based on Pytorch. arXiv.org, https://doi.org/10.48550/arXiv.2208.08231

[89] Miomir, Nikšić. (2022). Introduction to TensorFlow. https://doi.org/10.1007/978-1-4842-8835-1_2

[90] Damien, Rolon-Mérette., Matt, Ross., Thaddé, Rolon-Mérette., Kinsey, Church. (2020). Introduction to Anaconda and Python: Installation and setup. https://doi.org/ https://doi.org/10.20982/TQMP.16.5.S003

[91] A., I., Yakimchik. (2019). Jupyter Notebook: a system for interactive scientific computing. D oi: 10.24028/GZH.0203-3100.V41I2.2019.164458. https://journals.uran.ua/geofizicheskiy/article/view/164458

[92] Elvis, C., Foster., Shripad, V., Godbole. (2016). Overview of Microsoft SQL Server. https://doi.org/10.1007/978-1-4842-1191-5_25

[93] PWC Automotive Supplier survey 2021/2022. (2023). Available on: https://www.pwc.com/id/en/publications/cips/automotive/automotive-supplier-survey.pdf . (Accessed in 14-06-2024).

[94] Deloitte-CN-CIP-ai-Manufacturing-Application-survey-en- ... (n.d.-a). Available on: https://www2.deloitte.com/content/dam/Deloitte/cn/Documents/cip/deloitte-cn-cip-ai-manufacturing-application-survey-en-200116.pdf . (Accessed in 14-06-2024).

[95] 2020 State of AI-Based Machine Vision High confidence, growing adoption and many

challenges -trends and insights based on a survey of 110 companies. (n.d.). Retrieved May 20, 2024, from https://landing.ai/wp-content/uploads/2020/11/MachineVisionSurvey.pdf

[96] Functional Vs Nonfunctional Requirements: The 2024 Guide, A. (n.d.). Functional vs. nonfunctional requirements: The 2024 guide. https://agilemania.com/functional-vs-nonfunctional-requirements

[97] Wikipedia Contributors. (2020, February 6). CUDA. Wikipedia; Wikimedia Foundation. Available on: https://en.wikipedia.org/wiki/CUDA. (Accessed in 14-06-2024).

[98] Ultralytics Train too slow on yolov5-6.1 compared with yolov5-5.0 · Issue #7470 · ultralytics/yolov5. (n.d.). GitHub. Retrieved May 03, 2024, from https://github.com/ultralytics/yolov5/issues/7470

[99] HumanSignal/labelImg. (2023, August 17). GitHub. https://github.com/HumanSignal/labelImg

[100] HumanSignal/label-studio. (2023, October 26). GitHub. https://github.com/HumanSignal/label-studio

Appendix

A1. Project folder Structure

Given the global scope of *AUTOProductions'* business, organization is often times regarded as more important than the innovation itself. Because of this, it is necessary that all the files related to the project, from the first photos collected for the dataset, through the exported dataset (with the test, validation, and train folders, as well as the data.yaml file), to the final trained model are stored in a directory where it is easy to search for, modify and scale files.

On a server with connection to the production computer network, a new folder was created, 'industrial_vision', which itself contains a sub-folder ('data_lake'), in which all the files relating to the trained vision models will be stored.

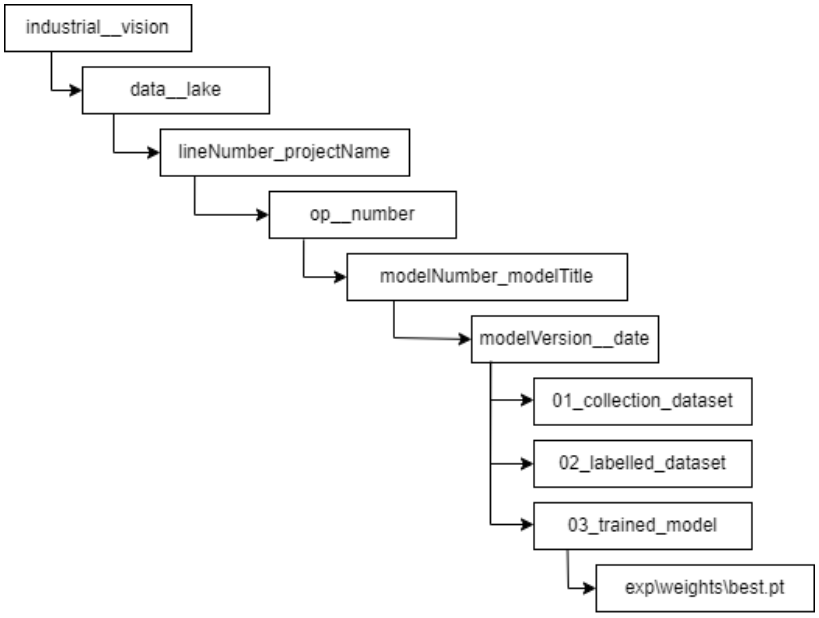


Figure 51 - Data Lake Folder Structure

As illustrated in figure 51, inside the 'data_lake' folder, the production lines will be

identified by their project number and name, followed by the desired operation, the view model number and its title (for example, '001_final_components' and '002_metallic_clips', if we want to have more than one checkpoint in an operation), the current version of the model and its execution date (for example, a model update would be version '002'). Finally, within this version folder, the images used for training, the annotated dataset and the trained model will be stored in the exp\weights folder, as generated by the training script used in 5.3.3.2.4) (Model Training).

A.2 Code

This appendix contains the main source codes for the application inference model based on the YOLOv5 original script, as well the plc_slave script mentioned in 5.3.4.3.3). Regarding the YOLOv5 improved script, some abstractions were made in the code bellow, so to not take to much unnecessary space, like the module imports, database connection, as well parts of the original YOLO like parts of the dataloader and inference mode warmup. In the same way, in the plc_slave file, the database connection was abstracted.

File “application.py”

```
# <IMPORT EXTERNAL MODULES FROM YOLOv5 INITIAL MODEL>
# <IMPORT LOCAL MODULES (area and counter)>

index = 0 #INDEX FOR THE AREAS CREATED
objIndex = 1 #INDEX FOR THE OBJECTS GENERATED
currentAr = 0 #DECLARE THE VARIABLE THAT WILL HOLD THE CURRENT AREA (THAT WILL BE
DRAWED)
areasList = [] #DECLARE THE VARIABLE THAT WILL HOLD THE ARRAY OF SINGLE AREAS CREATED
areas = area.Areas() #creates a array of areas to hold the areas created
areaCounter = 0

CAM = 0 # CAMERA IN USE
FILEPATH = 'app/detection_files/' # DETECTION FILES PATH FOLDER

nPrograma = ''
lineNumber = ''
lineOp = ''
modelNumber = ''

areasToInsert = []
areasToUpdate = []

# <MAIN DATABASE CONNECTION>

def pre_run_weight(weight):
    # TO GENERATE AND WRITE ON DETECTION FILE THE N_PROGRAMA TO PLC STANDART
    global nPrograma, lineNumber, lineOp, modelNumber
    weight0 = weight
```

```
    weight0_str = weight0[0]

    data_lake_pattern = re.search(r'data__lake\\(\d+)', weight0_str)
    op_pattern = re.search(r'\\op(\d+)\\(\d+)__.*\\(\d+)v__.*\\(\d+)', weight0_str)

    lineTemp = f"{int(data_lake_pattern.group(1)):04}" if data_lake_pattern else None
    opTemp = f"{int(op_pattern.group(1)):04}" if op_pattern else None
    modelTemp = f"{int(op_pattern.group(2)):04}" if op_pattern else None
    versionTemp = f"{int(op_pattern.group(3)):04}" if op_pattern else None

    nPrograma = f'l{lineTemp}o{opTemp}m{modelTemp}v{versionTemp}'
    lineNumber = int(lineTemp)
    lineOp = int(opTemp)
    modelNumber = int(modelTemp)

def pre_run_source(source):
    # TO RETREIVE THE CAMERA RECEIVED IN THE BEGIN SCRIPT TO LOAD THE ASSIGNED SMART
AREAS
    global CAM, FILEPATH, nPrograma, lineNumber, lineOp
    source0 = source
    selectSourceQuery = f'SELECT cam FROM [industrial__vision].[dbo].[sources] WHERE
source = {source0}'
    cursorObject.execute(selectSourceQuery)
    CAM = cursorObject.fetchmany()[0][0]
    FILEPATH += f'cam{CAM[-1]}.txt'

    detectionFile = open(FILEPATH, 'a')
    detectionFile.write(f'N_Programa | {nPrograma}\n') # WRITE N_PROGRAMA ON DETECTION
FILE
    detectionFile.write(f'Camera | {CAM}\n\n') # WRITE CAMERA NUMBER ON DETECTION FILE
    detectionFile.close()

    #SELECT ALL FROM AREA_COORDS DB TABLE
    selectAreaQuery = f"SELECT * FROM [industrial__vision].[dbo].[areas] WHERE
line_number = '{lineNumber}' and line_op = '{lineOp}' and model = {modelNumber}"
    cursorObject.execute(selectAreaQuery)
    myresult = cursorObject.fetchall()

    #LOAD ALL AREAS
    for areaLoaded in myresult:
        areasList.append(list(areaLoaded))

    for row in areasList: #for each area from the list of areas of db
        areaCarregada = area.Area(row[1], row[2], ...) #Create a area object with the
fields from db
        index = row[0] #Pass the global index value to the current index + 1
        areaCarregada.index = index
        areaCarregada.classes = [row[6], row[7], row[8], row[9], row[10]] #Define the
area classes, with the values from the db
        areaCarregada.defineClasses() #Define the classes for the current area
        areas.add_area(areaCarregada) # add the area to the areas array

        detectionFile = open(FILEPATH, 'a')
        detectionFile.write(f'{areaCarregada.plcOffset} | AREA{areaCarregada.index} |
0\n')
        detectionFile.close()

def checkClickEvents(event, x, y, flags, param): #DEFINE THE CLICK FUNCTION INSIDE THE
OPENCV DASHBOARD
```

```
    global index, currentAr #make index and currentAr as a global variable, to hold
the same values

    if event == cv2.EVENT_LBUTTONDOWN: #If left click is pressed
        currentAr = 0 #currentAr disappears
        for area0 in areas.getAreas(): #for area created in the areas array
            area0.selected = False #turn the area selection to false, at the beginning
            if area0.position[0] < x < area0.position[0] + area0.length and
area0.position[1] < y < area0.position[1] + area0.height: #if mouse pointer is inside
the area
                area0.selected = True #area is selected, so turn it to true
                currentAr = area0 #the currentAr holder is equal to the area0
                area0.changed = True #and area0 is changed for the meantime

    if event == cv2.EVENT_MOUSEMOVE:
        # DRAG AND DROP ON SMART AREA
        if currentAr:
            currentAr.position[0] = int(x - currentAr.length/2)
            currentAr.position[1] = int(y - currentAr.height/2)

    if event == cv2.EVENT_LBUTTONUP:
        currentAr = 0 #currentAr disappears
        for area0 in areas.getAreas(): #for area created in the areas array
            area0.selected = False #turn the area selection to false, at the beginning

def checkAreas():
    for area0 in areas.getAreas():
        classes = [0, None, None, None, None]
        for i in range(len(area0.classes)):
            classes[i] = area0.classes[i]
            areaInfo = (area0.name, area0.position[0], area0.position[1],
area0.length, area0.height, classes[0], classes[1], classes[2], classes[3],
classes[4])
        if area0.changed:
            areasToUpdate.append(areaInfo)
            print(f'Area Updated: {area0.name} - (x, y): {area0.position[0],
area0.position[1]}, length: {area0.length}, height: {area0.height}')
            #UPDATE QUERY
            cursorObject.executemany('UPDATE [industrial__vision].[dbo].[areas] SET x
= ?, y = ?, width = ?, height = ? WHERE name = ?', [(area0.position[0],
area0.position[1], area0.length, area0.height, area0.name)])
    mydb.commit()
    mydb.close()

# ROOT RESOLVE, MODELS, UTILS AND TORCH IMPORTS FROM YOLOv5 INITIAL MODEL

@smart_inference_mode()
def run(
        weights=ROOT / 'yolov5s.pt',  # model path or triton URL
        source=ROOT / 'data/images',  # file/dir/URL/glob/screen/0(webcam)
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        # ...
        # from YOLOv5 initial model
):
    global cursorObject, index, objIndex, currentAr, areasList, areas, CAM #importar
as variaveis gerais para que sejam tratadas globalmente
    source = str(source)
    save_img = not nosave and not source.endswith('.txt')  # save inference images
```

```
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    webcam = source.isnumeric() or source.endswith('.streams') or (is_url and not
is_file)
    screenshot = source.lower().startswith('screen')
    if is_url and is_file:
        source = check_file(source)  # download

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment
run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
exist_ok=True)  # make dir

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    startTime = 0 # FOR FPS CALCULATION
    print(f'\n{model.names}\n')

    contador = counter.Counter(30, 620, model.names)
    contador.createTopics()

    # Dataloader
    bs = 1  # batch_size
    if webcam:
        view_img = check_imshow(warn=True)
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt,
vid_stride=vid_stride)
        bs = len(dataset)
    elif screenshot:
        dataset = LoadScreenshots(source, img_size=imgsz, stride=stride, auto=pt)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt,
vid_stride=vid_stride)
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz))  # warmup
    seen, windows, dt = 0, [], (Profile(), Profile(), Profile())
    ##############################################################
    for path, im, im0s, vid_cap, s in dataset:
        with dt[0]:
            im = torch.from_numpy(im).to(model.device)
            im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
            im /= 255  # 0 - 255 to 0.0 - 1.0
            if len(im.shape) == 3:
                im = im[None]  # expand for batch dim

        # Inference
        with dt[1]:
            visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if
visualize else False
            pred = model(im, augment=augment, visualize=visualize)

        # NMS
```

```
        with dt[2]:
            pred = non_max_suppression(pred, conf_thres, iou_thres, classes,
agnostic_nms, max_det=max_det)

        # Process predictions
        ############################################################
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                im0 = cv2.resize(im0, (0, 0), fx = 1.6, fy = 1.6) # resize window to
1.6x the size
                s += f'{i}: '
            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

            #FPS CALCULATION
            currentTime = time.time()
            fps = 1/(currentTime - startTime)
            startTime = currentTime

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode ==
'image' else f'_{frame}')  # im.txt
            s += '%gx%g ' % im.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop
            annotator = Annotator(im0, line_width=line_thickness, example=str(names))

            cv2.namedWindow(str(p)) #Creates a named windows, as frame cap
            cv2.setMouseCallback(str(p), checkClickEvents)

            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, 5].unique():
                    n = (det[:, 5] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

                for area0 in areas.getAreas():
                    area0.oks = []

                items = []
                indice = {}
                contador.preupdate()

                # Write results
                # LOOP THROUGH EACH BOUNDING BOXES PREDICTED
                ############################################################
                # for *xyxy, conf, cls in reversed(det):
                for j, (*xyxy, conf, cls) in enumerate(reversed(det[:, :6])):
                    # Determine which class is being detected
                    objectClass = det[j][5]

                    items.append(det[j][5].item())
```

```
                    # SHOW FPS
                    cv2.putText(im0, "FPS: " + str(int(fps)), (30, 50),
cv2.FONT_HERSHEY_PLAIN, 2, (0,170,0), 3)

                    if currentAr: #Se currentAr existir
                        cv2.rectangle(im0, (currentAr.position[0],
currentAr.position[1]), (currentAr.position[0]+currentAr.length,
currentAr.position[1]+currentAr.height), currentAr.color, currentAr.thickness)
#Desenhar o retangulo

                    for indArea in areas.getAreas(): #Para cada Area que existir
                        cv2.rectangle(im0, (indArea.position[0], indArea.position[1]),
(indArea.position[0]+indArea.length, indArea.position[1]+indArea.height),
indArea.color, indArea.thickness) #Desenhar o retangulo da Area
                        cv2.putText(im0, str(f'{indArea.name}'), (indArea.position[0],
indArea.position[1]-10), cv2.FONT_HERSHEY_PLAIN, 1, indArea.color, 1) #Desenhar o
index

                    for area0 in areas.getAreas(): #Para cada area no array de areas
                        if det[j][5] in area0.classes: #se a classe de deteção fizer
parte da classe que a area aceita
                            holder = area0.defineHolder(det[j][5])
                            if area0.index == holder[2]:
                                #SE A DETEÇAO ESTA DENTRO DA AREA
                                if det[j][0] > area0.position[0] and det[j][2] <
area0.position[0] + area0.length: #se a largura da deteção estiver dentro da largura
da area
                                    if det[j][1] > area0.position[1] and det[j][3] <
area0.position[1] + area0.height: #se a altura da deteção estiver dentro da altura da
area
                                        area0.oks.append(det[j][5].item())

                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-
1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  #
label format
                        with open(f'{txt_path}.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')

                    if save_img or save_crop or view_img:  # Add bbox to image
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if hide_conf else
f'{names[c]} {conf:.2f}')
                        annotator.box_label(xyxy, label, color=colors(c, True))
                    if save_crop:
                        save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] /
f'{p.stem}.jpg', BGR=True)
                else:

                    for item in items:
                        if item not in indice:
                            indice[item] = 0
                        indice[item] += 1

                    contador.update(indice)
                    contador.show(im0)

            # Stream results
```

```
            im0 = annotator.result()
            if view_img:
                result = 0
                for area0 in areas.getAreas():
                        areaNum = 'AREA' + str(area0.index)
                        if set(area0.oks) == set(area0.classes):
                            area0.color = (0, 255, 0) #alterar a cor da area para
verde
                            area0.thickness = 3
                            with open(FILEPATH, 'r') as file:
                                lines = file.readlines()

                            modified_lines = []
                            for line in lines:
                                line = line.rstrip()  # Remove trailing newline
characterprint(line)

                                if line and line.startswith(f'{area0.plcOffset}') and
areaNum in line:
                                    line = str(area0.plcOffset) + ' | ' + 'AREA' +
str(area0.index) + ' | ' + str(1)  # Replace last character with '1'
                                modified_lines.append(line)

                            with open(FILEPATH, 'w') as file:
                                file.write('\n'.join(modified_lines))

                        else:
                            area0.color = (0, 50, 255)
                            area0.thickness = 3
                            result += 1

                            with open(FILEPATH, 'r') as file:
                                lines = file.readlines()

                            modified_lines = []
                            for line in lines:
                                line = line.rstrip()  # Remove trailing newline
character
                                if line and line.startswith(f'{area0.plcOffset}') and
areaNum in line:
                                    line = str(area0.plcOffset) + ' | ' + 'AREA' +
str(area0.index) + ' | ' + str(0)  # Replace last character with '1'
                                modified_lines.append(line)

                            with open(FILEPATH, 'w') as file:
                                file.write('\n'.join(modified_lines))
                else:
                    #print()
                    fpoint = (1024, 767)
                    if result == 0:
                        #############
                        ## SEND TO PLC THE FIM DE CICLO VALUE
                        #############
                        #print('VERIFICATION OK', seen)
                        cv2.rectangle(im0, (0, 0), fpoint, (0, 255, 0), 8)
                        # writeBool(int(GLOBAL_DB), int(GLOBAL_OUT_RESULT[0]),
int(GLOBAL_OUT_RESULT[-1]), 1)
```

```
                    else:
                        cv2.rectangle(im0, (0, 0), fpoint, (0, 255, 255), 8)
                        # writeBool(int(GLOBAL_DB), int(GLOBAL_OUT_RESULT[0]),
int(GLOBAL_OUT_RESULT[-1]), 0)
                cv2.imshow(str(p), im0)

                #CONTROLOS DO DESENHO E GRAVAÇÃO DAS AREAS DE INTERESSE
                ch = cv2.waitKey(1)
                if ch & 0xFF == 27: #esc key
                    checkAreas()
                    open(FILEPATH, 'w').close() #Deletes records from the previous
detection session that is ending
                    exit()
                elif ch & 0xFF == ord('w'): #w key
                    currentAr.height -= 2
                elif ch & 0xFF == ord('s'): #s key
                    currentAr.height += 2
                elif ch & 0xFF == ord('a'): #a key
                    currentAr.length -= 2
                elif ch & 0xFF == ord('d'): #d key
                    currentAr.length += 2
                elif ch & 0xFF == ord('m'): #m - info
                    print(contador.topics)


def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt',
help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / 'data/images',
help='file/dir/URL/glob/screen/0(webcam)')
    # ...
    # Pass other arguments that come with YOLOv5
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    return opt

def main(opt):
    pre_run_weight(vars(opt)['weights'])
    pre_run_source(vars(opt)['source'])
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))

if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
```

File “plc_slave.py”

```
import pyodbc
import argparse
import area

# PLC SNAP7 CONNECTION LIBRARY
import snap7
from snap7 import util
```

```
lineNumber = ''
opNumber = ''

# <MAIN DATABASE CONNECTION>

client = snap7.client.Client()
client.connect('192.168.0.1', 0, 1)

print('PLC Connected: ', client.get_connected())
print('PLC Status: ', client.get_cpu_state())
print()

plcAreas = []

# #SELECT ALL FROM AREA_COORDS DB TABLE
selectAreaQuery = f'SELECT id, db, plc_offset, cam FROM area'
selectAreaQuery = f"SELECT id, line_number, line_op, plc_db, plc_offset, model FROM
[industrial__vision].[dbo].[areas]"
cursorObject.execute(selectAreaQuery)
myresult = cursorObject.fetchall()

for row in myresult: #for each area from the list of areas of db
    areaCarregada = area.plcArea(f'AREA{row[0]}', row[1], row[2], row[3], row[4],
row[5]) #Create a area object with the fields from db
    plcAreas.append(areaCarregada)

for object in plcAreas:
    print(object.id, object.lineNumber, object.lineOp, object.plcDB, object.plcOffset,
object.model)
else:
    print()

def writeBool(db_number, start_offset, bit_offset, value):
    reading = client.db_read(db_number, start_offset, 1)
    snap7.util.set_bool(reading, 0, bit_offset, value)
    client.db_write(db_number, start_offset, reading)
    return None

PATH = '<path\\to\\industrial-vision\\yolov5\\>'

PATHS = [PATH + 'app\\detection_files\\cam0.txt',
         PATH + 'app\\detection_files\\cam1.txt',
         PATH + 'app\\detection_files\\cam2.txt',
         PATH + 'app\\detection_files\\cam3.txt',
         PATH + 'app\\detection_files\\cam4.txt',
         PATH + 'app\\detection_files\\cam5.txt',
         PATH + 'app\\detection_files\\cam6.txt',
         PATH + 'app\\detection_files\\cam7.txt',
         PATH + 'app\\detection_files\\cam8.txt',
         PATH + 'app\\detection_files\\cam9.txt']

while True:
    for path in PATHS: #For each detection file (cam0, cam1, cam2, cam3...)
        with open(path, 'r') as file:
            lines = file.readlines()

            currentLine = ''
            lineCounter = 0
```

```
            areaCounter = 0

            for line in lines: #For each line in the detection file (area1, area2,
area3...)
                for plcArea in plcAreas: #For each plcArea in the list of plcAreas
                    if line:
                        currentLine = line
                    line = line.rstrip()

                    if str(line[-1]) == str(1) and plcArea.id in line:
                        print(plcArea.db, int(plcArea.offset[:1]),
int(plcArea.offset[-1:]), 1)
                        writeBool(plcArea.db, int(plcArea.offset[:1]),
int(plcArea.offset[-1:]), 1)
                        areaCounter += 1
                    elif str(line[-1]) == str(0) and plcArea.id in line:
                        writeBool(plcArea.db, int(plcArea.offset[:1]),
int(plcArea.offset[-1:]), 0)

                    lineCounter += 1
```