NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENT OF
MECHANICAL AND INDUSTRIAL ENGINEERING

PEDRO MIGUEL RODRIGUES PINTO GARCIA
BSc in Mechanical Engineering Sciences

AUTONOMOUS TRIGGER-TASK-SEQUENCING SYSTEM FOR HUMAN-ROBOT COLLABORATIVE ASSEMBLY: A DEEP LEARNING APPROACH ON VISUAL TASK RECOGNITION

MASTER'S IN MECHANICAL ENGINEERING
NOVA University Lisbon
October 2021

DEPARTAMENT OF
MECHANICAL AND INDUSTRIAL ENGINEERING

Autonomous trigger-task-sequencing system for Human-Robot Collaborative assembly: A Deep Learning approach on Visual Task Recognition

PEDRO MIGUEL RODRIGUES PINTO GARCIA

BSc in Mechanical Engineering Sciences

Adviser: PhD. Nuno Alberto Marques Mendes
Assistant Professor, NOVA School of Science and Technology

Examination Committee:

Chair: PhD. Telmo Jorge Gomes dos Santos,
Associate Professor with Habilitation, NOVA School of Science and Technology

Adviser: PhD. Nuno Alberto Marques Mendes
Assistant Professor, NOVA School of Science and Technology

Members: PhD. Pedro Mariano Simões Neto,
Assistant Professor with Habilitation, Faculty of Sciences and Technology of University of Coimbra

MASTER'S IN MECHANICAL ENGINEERING

NOVA University Lisbon
October 2021

Autonomous trigger-task-sequencing system for Human-Robot Collaborative assembly: A Deep Learning approach on Visual Task Recognition

“Invention is the most important product of man's creative brain. The ultimate purpose is the complete mastery of mind over the material world, the harnessing of human nature to human needs.”

Nikola Tesla

ACKNOWLEDGMENTS

I would like to start by acknowledging the Research and Development Unit for Mechanical and Industrial Engineering integrated in the Department of Mechanical and Industrial Engineering (UNIDEMI) of NOVA School of Science and Technology, for providing most of the necessary equipment to develop my dissertation. I am also thankful for the professors of this department that inspired and taught me throughout my educational journey at NOVA University, helping me understand how interesting and gratifying working as an engineer could be. I would also like to specifically acknowledge my adviser for proposing a topic of study that sparked great interest in me and that allowed me to learn about the fields of deep learning and robotics, two fields that I had been greatly interested in for a while.

I am also grateful for all the colleagues I have met over the past 5 years of studying at NOVA that contributed to making my academic journey as incredible as it turned out to be. I would particularly express my gratitude towards the inner circle of friends that university gave me, who became important people in my life and that I want to maintain connected to in the future. Besides the ones the university gave me, I am also thankful for all my closest friends outside the university environment, with whom I have been delighted with the chance of sharing experiences and memories throughout the years and were a source of great support throughout some of the down moments during the development of this master's thesis.

Furthermore, I want to thank all my family, but in a greater extent to my parents, uncles, and grandparents, who unconditionally supported me in my decisions throughout life, even when skeptical at first, enabling me to study what I wanted at university and experiencing many other incredible things throughout my journey, which contributed in one way or another to the individual I am today and to the engineer I am about to become.

Last but not the least, I am grateful towards my wonderful girlfriend for all the love, support, care, and partnership throughout the adventures of life.

ABSTRACT

Human-robot collaborative (HRC) assembly environments are becoming increasingly important as the paradigm of manufacturing is shifting from mass production towards mass customization, and the introduction of a HRC system could significantly improve the flexibility and intelligence of automation. To efficiently accomplish tasks in HRC assembly environments, a robot needs to be able to understand its surroundings by detecting, recognizing, and locating the presence of humans and targeted objects, as well as to have a higher level of understanding beyond what is being currently seen, i.e., to fully understand a sequence of operations and tasks related to each step of the assembly process.

Within the scope of the Internet of Things (IoT) and Artificial Intelligence (AI), the most widely used framework to enable robot vision (or vision in other type of cyber-physical systems) is deep learning, precisely by means of deep learning algorithms based on Convolutional Neural Networks (CNNs).

Hence, in this study a system capable of managing a HRC assembly process of a mechanical component was developed, where computer vision was enabled by CNN-based methods and the assembly task recognition made from solely visual RGB data of the components in working space. A model for components' detection was firstly developed by comparing two main CNN-based framework branches – the Faster Region-based Convolutional Neural Network (Faster R-CNN) and the You Only Look Once (YOLO). Then, a system that correlated the current state of the working space – i.e., whether certain components were already correctly assembled – with the progression of the assembly sequence was developed with the best-performing 11 class object detector, the YOLOv3. This framework was the only capable of detecting a small object classes in comparison with the other benchmarked frameworks and presented a *mean average precision* (mAP) of 72.26% over the test dataset. Using YOLOv3 as the computer vision-enabling framework, a successful and efficient HRC industrial demonstrator was created.

KEYWORDS

Visual assembly task recognition.
Human-Robot Collaborative assembly.
Online Object Detection.
YOLO.
Faster R-CNN.

RESUMO

Os ambientes de colaboração homem-robô (CHR) têm vindo a ganhar relevância à medida que o paradigma do fabrico industrial se vindo a alterar da tradicional produção em massa e em grandes volumes, para uma tipologia de fabrico mais customizada. A introdução da CHR pode aumentar significativamente a flexibilidade e grau de inteligência dos sistemas de automação. Para realizar eficientemente as tarefas pré-definidas em ambientes de CHR, os robôs necessitam de ter a capacidade de compreender o seu ambiente envolvente e de detetar, reconhecer e localizar a presença de humanos e de objetos chave, mas também de realizar associações de alto-nível, i.e., compreender em detalhe e processar a sequência de ações necessárias para realizar cada uma das tarefas de montagem pré-definidas.

No âmbito da Internet das Coisas (IoC) e da Inteligência Artificial (IA), a estrutura que atualmente mais se utiliza para tornar a visão computacional uma realidade é o *deep learning* - Aprendizagem Profunda (AP) – nomeadamente os algoritmos de AP baseados na arquitetura das *Convolutional Neural Networks* (CNNs).

No presente estudo, um sistema capaz de gerir todo o processo de montagem em CHR de um componente mecânico foi desenvolvido, onde a visão computacional deste sistema foi habilitada por algoritmos em arquiteturas CNN, sendo o reconhecimento visual das operações de montagem realizado com base em imagens dos componentes dispersos pelo espaço de trabalho adquiridas por uma câmara de vídeo. Inicialmente, um modelo para deteção de componentes foi desenvolvido comparando-se nesta etapa dois ramos de CNNs comumente utilizados para a deteção de objetos – as *Faster Region-based Convolutional Neural Networks* (*Faster* R-CNN) e as *You Only Look Once* (YOLO). De seguida, um sistema que correlacionava o estado atual do espaço de trabalho – i.e., quais componentes já se encontrariam montados – com o andamento da sequência de montagem foi desenvolvido com o modelo que apresentou os melhores resultados, o YOLOv3. Este modelo foi o único capaz de detetar a classe de objetos de dimensões mais reduzidas em comparação com os outros modelos e apresentou uma *mean average precision* (mAP) de 72,26% no *dataset* de teste. Por fim, um eficiente demonstrador industrial da montagem de todo o processo em CHR foi desenvolvido e implementado com sucesso.

PALAVRAS-CHAVE

Reconhecimento visual de operações de montagem.
Colaboração Homem-robô.
Deteção de objetos *online*.
YOLO.
Faster R-CNN.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND ABBREVIATIONS

AI	Artificial intelligence
CNN	Convolutional neural network
DCNN	Deep convolutional neural network
FDM	Fused deposition modeling
FN	False negatives
FP	False positives
FPS	Frames per second
HRC	Human-robot collaborative
i4.0	Industry 4.0
IoT	Internet of things
IoU	Intersection over union
mAP	Mean average precision
R-CNN	Region-based convolutional neural network
RGB	Red Green Blue
RPN	Region proposal network
TP	True positives
YOLO	You only look once

1. INTRODUCTION

1.1. Motivation

Quality, efficiency, and productivity are three of the most dominant success indicators in modern industry and for this reason organizations drive a big amount of their investments and dedication to increase and improve these indicators continuously.

With this vision, the Industry 4.0 (i4.0) paradigm underlines several approaches with contemporary technologies and ways of thinking which have been showing that the use of real time data boosts efficiency, flexibility, productivity and production quality levels [1], when implemented correctly and in the right contexts. Within this scope, combining concepts as Internet of Things (IoT) and Artificial Intelligence (AI) – two of the several foundations of i4.0 [2] – enables robots and machines to work autonomously on the factory floor with minimal or even any human intervention, by providing machines the capability of recognizing, analyzing and act according to the received information on the surroundings from external sensors (e.g. a RGB camera). One of the most widely used frameworks for enabling robots and machines to have visual perception of their surroundings, and the one explored in this work, is deep learning with focus on the concept of deep Convolutional Neural Networks (DCNNs).

However, instead of solely using autonomous robots to perform a variety of tasks, there is also value in combining human taskforce with autonomous robots in the so-called human-robot collaborative (HRC) environments. Furthermore, HRC assembly is a potential way of combining the best of two worlds, i.e., by promoting an environment where humans could safely teamwork with autonomous robots to carry out assembly tasks, it could be possible to maximize the potential of each one of the executors, the versatility and flexibility of the human worker and the automation and robustness of the robot.

Therefore, the success of the implementation of a HRC assembly environment is mainly dependent on the capability of the robot to properly understand its surroundings in order to act correctly, safely, and at the appropriate time instance. Hence the interest in developing a study on a deep learning-based framework that could maximize the aforementioned capability of surrounding perception.

1.2. Objectives

The goal of this dissertation study is to develop a system capable of correctly managing the HRC assembly process of a selected mechanical component comprised of both human and robot tasks which could happen simultaneously or not. In order to achieve this purpose, a key element must be developed: a framework capable of visually recognizing every task through the correlation of the visual video input of the working space with a higher level of understanding of the entire assembly process and current progress.

Throughout the development of this study, it is expected to achieve the following milestones:

1. Creation of a suitable dataset with images associated to every task and part of the mechanical component.
2. Development of a multi-class object detector.
3. Elaboration of an effective management system that correlates the detected objects with the current assembly progress, enabling in this way task recognition.
4. Deployment and testing of the management system with an industrial robotic demonstrator.

1.3. Structure of the dissertation

The main layout of the present document is divided in 8 chapters.

Chapter 2 is comprised by the literature review where different approaches for enabling computer vision are defined, an overview on generic deep learning methods for image classification and object detection is given, main deep learning methods and limitations and opportunities are identified, and several works in the field are analyzed.

Chapter 3 is a brief exposition of the theoretical fundamentals of DCNN frameworks, such as Region-based Convolutional Neural Networks (R-CNNs) or You Only Look Once (YOLO).

Chapters 4 presents a generic methodology for implementing a human-robot collaborative (HRC) environment. **Chapter 5** refers to the experimental procedure for the studied HRC assembly process. In in **Chapter 6,** the results of both the multi-class object detector development and the deployment of this detector in the industrial demonstrator are presented and discussed. **Chapter 7** refers to the final remarks regarding the outcomes of this work, as well as suggestions for further developments. **Chapters 8** and **9** are the references and appendix, respectively.

2. LITERATURE REVIEW

In line with one of the cornerstones of Industry 4.0 (i4.0) – leveraging the connectivity of machines within industrial production environments – human workers have to intuitively collaborate with flexible robotic cyber-physical systems that are required to take care of their own tasks as well as provide assistance to the worker when necessary [3]. These systems are usually called Human-robot collaborative (HRC) systems and HRC assembly or production environments have been one of the topics in the limelight of Artificial Intelligence (AI) applications in industry.

In such assembly systems, it is necessary to develop a communication framework between both sides, i.e., between the human and the robotic system. So, the robotic hardware needs to have the capability of recognizing the tasks that are currently being done and understand abstract concepts such an assembly sequence, in order to move at the required time instance and to autonomously teamwork with the human worker throughout the entire assembly process.

The following review is comprised of the comparison between contemporary ways of enabling task recognition in HRC environments and on the exposition of different frameworks that enhance these environments with the key capability of computer vision.

2.1. Coordination and interaction in HRC environments

In HRC assembly environments, the emphasis is given to the human worker and the robot has to adapt its pre-defined assembly tasks or movements on the fly and in accordance with the current progress of the entire assembly process, and/or to the actions of the human, with the goal of maximizing the potential of each one of the partakers, the versatility and flexibility of the human worker and the automation and robustness of the robot.

The coordination and interaction between the human and the robot proposed in the several related works can be broadly divided into two main categories according to the source of the data retrieved and processed by the HRC system, as presented in Figure 2.1. One category refers to the human worker's action or activity recognition via processing data as key-points in the body of the human (e.g., human joints) [4], visual and inertial sensorial data referred to certain hand gestures or signals [5], or even vocal sounds [6] that can be used to trigger the robotic part to perform its movements and tasks. The task recognition capability is achieved in systems incumbent to this category by correlating the aforementioned types of data with a specific task or

set of tasks in a predefined assembly sequence. The other category refers to types of HRC systems that only process data related to the assembly components, i.e., not directly dependent from the actions of the human worker and only based on the existence or inexistence of already assembled components or other objects (e.g., tools), as well as on their poses within the working cell useful space [6]-[7], which can be correlated with a certain time instance within the pre-defined assembly sequence, and trigger certain commands to the robotic hardware when required.

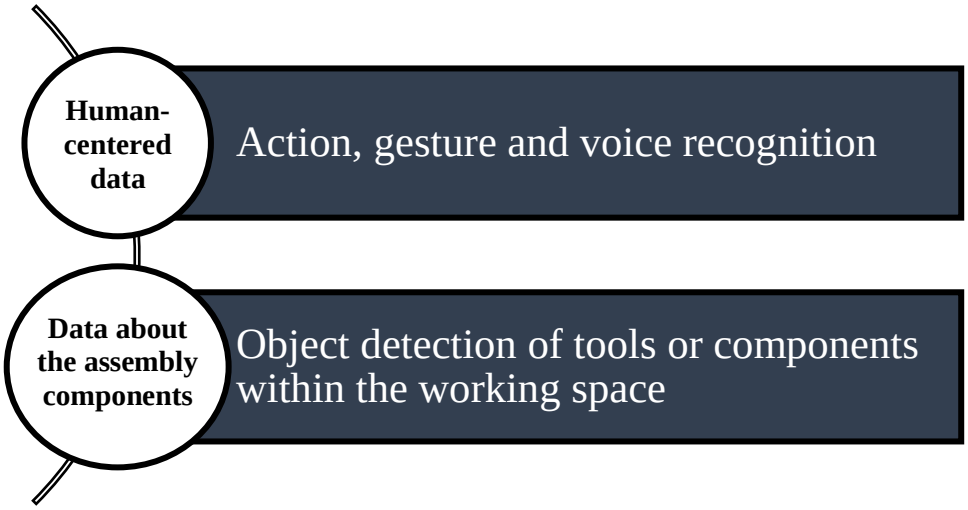


Figure 2.1 - Two different approaches to retrieve data for task recognition in industrial assembly.

Human related data is usually acquired by means of haptic and inertial sensors, brainwave signals or external visual sensors (e.g., an RGB camera or the Microsoft's Kinect sensor) with the goal of enabling action, gesture, or voice recognition and understanding actions of the human worker. On the other hand, data related only to the assembly components is mainly acquired through captured images or video of the working space by external visual sensors, and therefore, it can be considered as a problem of image recognition or object detection.

2.1.1. Human-centered methods for assembly task recognition

During the research for this work, it was identified that human centered methods are well established for assembly task recognition, as there are several publications and studies on human action/behavior recognition and prediction, which enables a higher level of understanding related with the assembly tasks, i.e., there is a pre-defined correlation between certain key movements or actions done by the human with the entire assembly progress.

Xiahne et al. [9] proposed a custom 3D CNN, which was a CNN framework built from scratch used to identify a series of images that were dynamic in time. The training dataset was comprised of 7 labeled video segments of each one of the 7 different assembly tasks to be carried out performed by human workers, and after some testing and tuning, this framework achieves a

mean average precision (mAP) of 82% for human action recognition, however, correlating these identified actions with the assembly tasks recognition was not successful, according to the authors.

By processing data collected from a Myo armband equipped with Inertial Measurement Unit (IMU) and surface electromyography (sEMG), Tao et al. [5] proposed a CNN framework also built from scratch, and capable of receiving 2 different inputs, one from the IMU and the other from the sEMG. A custom dataset with 6 different classes of common assembly activities (e.g.: grabbing a tool or tightening a nut using a wrench) was created, and a mAP of 89% for action recognition was achieved.

To not only enable action recognition, but also to estimate the operation times for repetitive assembly actions, Chen et al. [4] proposed a combined approach of human pose estimation and tool detection. For human join coordinate estimation, a Convolutional Pose Machine (CPM) model was used, and to assess whether a tool is being used, 2D CNN, 3D CNN and YOLOv3 were compared. To train these networks, a custom dataset was built, and it was comprised by 39 video clips of 13 operators performing 3 different types of different assembly actions, which originated 1493 labeled video frames. YOLOv3 performed slightly better on action recognition accuracy than 3D CNN (92.8% against 88.9%), and 2D CNN performed the worst, with a mAP of 82%.

An alternative approach to CNNs in order to enable human-robot collaborative assembly was presented by Cheng et al. [10]. They proposed a Long Short-term memory (LSTM) artificial neural network for human motion classification and target object estimation and trained it over a custom dataset with 50 trials for each action class. The overall classification of the 4 action classes was only 30% but the assembly plan recognition accuracy remained higher than 85%. The average time completion time was reduced by 29.1%.

Several other studies have been done with similar approaches and purpose, but the ones referred to in the present section represent the main different research paths regarding human-centered methods for assembly task recognition at present.

2.1.2. Generic object detection methods for components-based task recognition

In combination with Augmented Reality (AR), Ze-Hao et al. [8] proposed an integrated assembly assistance system based on Faster R-CNN that localized the correct tool to be used in the current task and displays AR assembly instructions by means of a display device. The network was pre-trained on PASCAL VOC [11] dataset and then fine-tuned (i.e., transfer learning was performed with a new dataset) over a custom synthetic dataset (3D CAD models) to detect and localize 5

different classes of tools, reaching a mAP of 84.7% for tool detection and spatial localization within the working space.

With a similar purpose, K-B. Park et al. [12] performed different experiments in order to propose two different applications, one for task assembly and maintenance assistance and other for human-robot interaction, in which the 3D position and pose detection of the real objects were performed by the Faster R-CNN, with ResNet-101 as a backbone network. This network was pre-trained on MS COCO dataset [13] and fine-tuned with a custom dataset where several classes were added, and plus a parallel Fully Connect Network (FCN), which results in the Mask R-CNN framework [14] for object instance segmentation. Focusing on their second experiment, this approach performed robust object detection and localization with position and angular errors inferior to 0.03m and 5º, respectively.

Liu et al. [15] compared Mask R-CNN with ResNet-101 and ResNet-50 as backbone networks to other neural network methods. A custom dataset was built with 80 assembled chassis images collected on the assembly production-line which was then augmented to 2500 image by means random rotation, translation, scaling, shearing and elastic transformation. Firstly, the Mask R-CNN was employed to detect and localize the different components within the chassis, originating 22 classes on one side of the chassis and 17 on the other, and outperforming the other methods with detection accuracy of 93.7% under different light intensities and conditions. Then, running on 800 chassis, ResNet-101 showed better mAP than ResNet-50 (78.1% against 73.6%). but showed greater computational time of 271 ms.

An overview of some characteristics of the learning environment and detection performance of each one of the review methods is compiled in Table 2.1.r

Table 2.1 - Overview of related work and their performances. The first 4 methods refer to human worker action recognition and the last 3 methods to visual recognition.

Method	**Backbone network**	**Pre-train dataset**	**Additional dataset size**	**Classes to detect**	**Detection performance**
3D CNN [34]	N/A	none	7 labeled video frames of human assembly activity	7 common assembly actions	mAP 82.0%
Custom CNN [35]	N/A	none	Muscle activation and inertial signals	6 common assembly actions	mAP 89.0%

YOLOv3 3D CNN 2D CNN [36]	DarkNet-53 N/A N/A	Yes, for YOLOv3, but dataset not specified	1560 labeled video frames	3 common assembly actions	mAP 92.8% 88.9% 82.0%	
LSTM [37]	N/A	none	50 video trials for each class	4 common assembly actions	Accuracy 85.0.%	
Faster R-CNN [38]	AlexNet	PASCAL VOC	Synthetic dataset (2000 images per class)	5 different tools	mAP 84.7%	
Faster R-CNN + Mask [40]	ResNet-101	COCO	200 images for each class	> 7 different objects	N/A (reduced localization error)	
Faster R-CNN + Mask [9]	ResNet-101 ResNet-50	Yes, but dataset not specified	2500 images of the assembled chassis	22 + 17 components (front and rear)	Accuracy 93.7%	mAP 78.1% 73.6%

2.2. Computer vision-enabling methods

All the aforementioned methods for granting a HRC system the ability of task recognition have in common the necessity of on an initial stage, enhancing the system with computer vision, which is something critical in enabling these cyber-physical systems with the capability of properly understanding the surroundings in their vicinity for recognizing and following the assembly tasks accordingly.

Computer vision can be defined as the research area that focuses on studying processes in which a computer could automatically process and interpretate an image or video acquired by external sensors with a high level of understanding [16]. Thus, the final goal of computer vision is to enable visual perception of a computer regarding a sensed scene, which could be used as a cornerstone for further decision-making (e.g., an autonomous vehicle stopping or changing its motion when a human is detected in its vicinity).

Today, a wide variety of methods for enabling computer vision have been proposed and explored; nonetheless, and as shown in Figure 2.2, these methods can be broadly divided into two main categories: reference image detection and non-reference image detection [15].

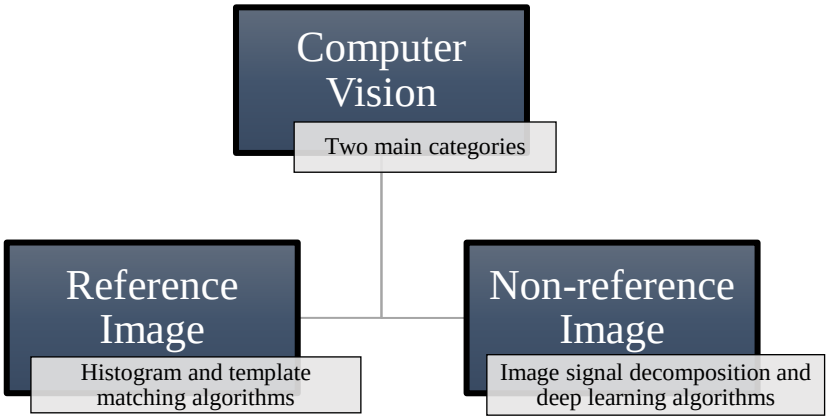


Figure 2.2 - Two main categories of computer vision.

2.2.1. Reference image detection

In reference image detection methods, a comparison between the image to be classified and a pre-defined standard reference one is made and the degree of similarity between areas of interest in the image to be classified and the reference image is measured. This measurement is usually made by means of algorithms as the histogram and the template matching algorithms [15]. Several approaches on reference image classification and object detection can be seen in [17]–[22].

Despite of relatively simple implementation, and showing effectiveness under static conditions, these methods are obsolete when the objects to be detected are moving (e.g.: during an assembly process where parts move from one place to another and vary pose), or when images to be detected show a same object or area of interest in a different pose than the reference image. Thus, reference image methods were considered inadequate for the purpose of this work.

2.2.2. Non-reference image detection

On the other hand, non-reference image detection methods are those in which the used algorithms extract features and characteristics from the images used as training data and then perform classifications on never seen data, according to pre-defined criteria or rules without the use of any reference image. As presented in Figure 2.2, this kind of methods may use image signal decomposition algorithms and achieve up to 95% detection accuracy [23]–[25], but have only shown reliability on images with prominent foreground, which is usually not the case in scenes of common factory floor environments.

On the other hand, the most extensive part of non-reference image detection frameworks are deep learning methods. Most widely used in generic image classification and object detection, with special focus on deep neural networks, and showing superior results than other computer

vision methods in different real-life applications due to their greater ability of automatically learning feature representations from data [26].

For these reasons and from this point forward, only computer vision methods based on deep learning techniques will be taken into consideration, for being the most commonly used at the present [27].

The concept of deep learning can be defined as a cluster of machine learning methods that create a multilayered representation of the input data. The manipulation and transformation of the data in each layer is usually trained through algorithms similar to backpropagation by using a dataset with known labelled images (or other way of data format) as training data [27]. Deep learning methods are the ones that are based on standard Artificial Neural Networks [28], which are a type of network architecture which aims to mimic the process of learning in a biological human brain so that classifications and decisions based on new input data can be made with higher accuracy.

In 2012, Krizhevsky et al. proposed a deep Convolutional Neural Network (DCNN) named AlexNet [29] that achieved at the time record breaking image classification accuracy in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Since this milestone and proof of considerable superiority of a deep learning method over other image processing methods, the field of computer vision has been shifting most of its research towards deep learning approaches [30].

2.3. Generic image classification

Generic image classification is known as the process of identifying one single class instance, in a given 2D image and it was the first way of implementing deep learning frameworks for image processing and understanding [27]. For instance, when proposed for the first time in 2012, the DCNN AlexNet [29] was a framework for generic image classification.

A visualization of how generic image classification can be carried out is shown in Figure *2.3*. As it is possible to observe, the network can either be trained – by means of a training dataset of images – to identify and localize within the image specific objects or entities (e.g., Mona Lisa or Eiffel Tower), or to classify more generic object class categories (e.g., car or cat), depending on the labelling carried out while creating the training dataset.

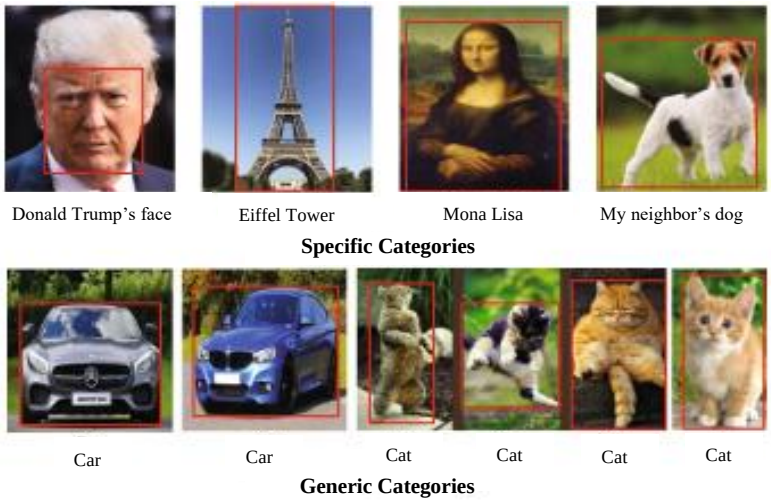


Figure 2.3 - Visualization of generic image classification. Adapted from [26].

Most of the early image classification deep learning models were trained on the first version of ImageNet [31], a large-scale ontology dataset of approximately 3.2 million training images and 12 subtrees, and throughout the following years a new version of ImageNet and 3 other popular datasets were developed for this purpose: PASCAL VOC, MS COCO and Open Images [26]. Preceded several breakthroughs in the field, deep learning methods for image classification of trivial items, such as everyday objects, plants or animals, achieved state-of-the-art recognition accuracies of 99.79%, and in some cases out-performing the human eye [32].

However, in real-life applications, the usual problem is not as simple as classifying a set of noiseless high-resolution stationary images, but rather the opposite. Hence, initial DCNNs, as for e.g., AlexNet, were not yet suitable for the complexity, variability, and randomness of the real-world. Nevertheless, these findings were the cornerstone for researchers to further develop new and more complex methods, enabling the implementation of deep learning in a wide range of applications that process higher dimensional data (e.g., video, or sensorial data).

2.4. Object detection

At present, object detection or recognition is the most predominant research area in deep learning in what computer vision is concerned, and it can be broadly defined as the process of determining whether there are any instances of pre-defined object categories in an image and, if positive, to return the spatial location and extent, as well as the classification label of each one of the detected object instances [26].

Object detection differs from generic image classification insofar as it concerns the classification and localization of 2 or more object categories instead of just one, as exposed in Figure 2.4. In this case, the object category is *race car*, the localization is made via bounding

boxes and the classification is made by showing the label of each object and its degree of confidence between 0 and 100%.

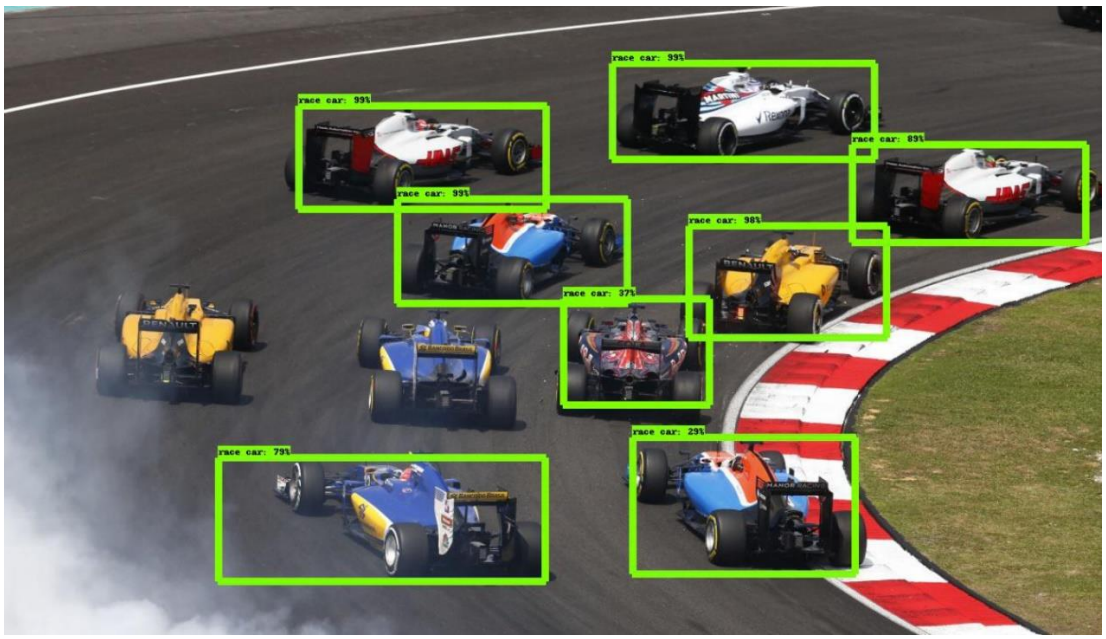


Figure 2.4 - Object detection in a given image. Object category is *race car* and the localization is made with bounding boxes [33].

2.4.1. Ways of defining the spatial location and extent of an object

The spatial location and extent of a detected object in a given image can be either determined by means of a bounding box (an axis-aligned rectangle that coarsely encloses the detected object), a precise pixelwise segmentation mask or a closed boundary around each detected object. The main different approaches to highlight the both the extent and location of an object are illustrated in Figure 2.5.

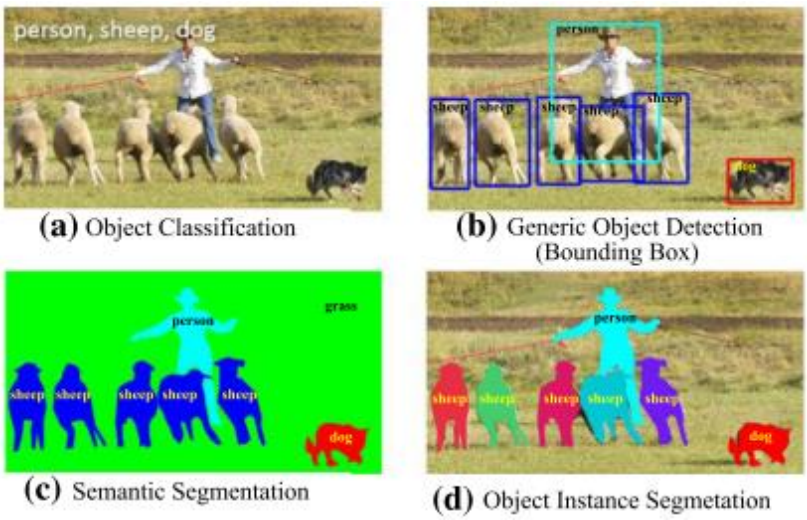


Figure 2.5 - Approaches to object detection: a) image level object classification, b) bounding box level object detection, c) pixel-wise semantic segmentation, d) instance level segmentation. Adapted from [26].

In Figure 2.4 a) it is shown a simple problem object classification or categorization, with the goal of determining the presence of specific object categories without outputting their location or extent. On the other hand, Figures 2.4 b), c) and d) illustrate approaches within the scope of object detection, with the latter opposing to semantic segmentation and bounding boxes by distinguishing different instances of the same object class within the same image.

2.4.2. Most relevant deep learning frameworks in the last decade

Succeeding the turning point hallmarked by AlexNet in 2012, several publications that enabled the evolution from classic generic image classification towards more different general object categories within the same image were made. The beginning of what is called object detection can be traced back to when a DCNN used for image classification was upgraded with this purpose, resulting in the creation of the Region-based Convolutional Neural Network (R-CNN) detector by Girshick et al. [34].

A chronological representation of the most relevant publications in the field of deep learning towards object detection are shown in Figure 2.6.

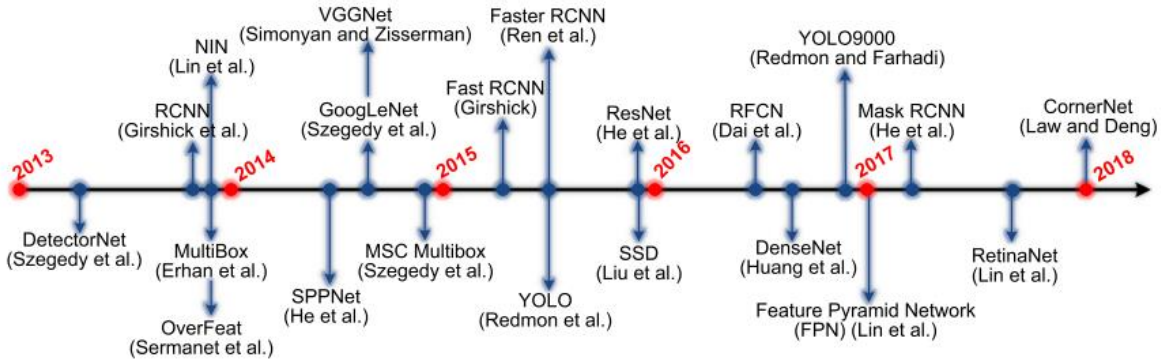


Figure 2.6 - Main deep learning breakthroughs in object detection [26].

There were the main frameworks which allowed the implementation of deep learning in several real-life applications such as robot/computer vision, consumer electronics, security, autonomous vehicles, human-computer interaction, intelligent video surveillance and augmented reality. The frameworks presented in Figure 2.6, can be divided in two main categories [26]:

1. Region-based or two-stage detectors, such as SPP-Net [35], ResNet [36] and CornerNet [37].
2. Unified or one-stage detectors, such as YOLO [38] and SSD [39].

Region-based detectors are computationally expensive frameworks due to the region proposal stage but perform slightly better on popular dataset benchmarks than one-stage detectors.

On the other hand, these unified or one-stage detectors refer to detectors that the detection does not involve the generation of region proposals, and are greatly faster in making detections than two-stage detectors [26]. Further explanation on this will be provided in section 3.

Some applications of DCNNs in other areas than industry, such as health care, agriculture or drug research can be found in [40]–[42]. Additionally, an in-depth review of the different methods based on the frameworks presented in Figure 2.6 can be found [27].

2.5. Limitations and opportunities for components-based task recognition

Deep learning applications in real-life object detection problems, face a few common-ground limitations:

- *Generalization of the model* – existence of tricky edge cases that contribute to a poor capability in generalizing certain detections models (e.g., an autonomous vehicle trained to detect a certain topology of traffic lights, of a certain height, size and shape, characteristic to a certain city or country. When the car drives in another city or country, it might perform differently).
- *Biased learning* – typically happens when the dataset is not lengthy enough and there exists an increased imbalance of object classes (e.g., in a double-class classification problem of a cat and dog, if 80% of the dataset images have cats and the other 20% dogs, the classifier had a biased learning process).
- *Necessity of good and big quantities of data and powerful hardware support*, as deep learning models for object detection are computationally expensive.
- *Difficulties in model explainability* – which is the ability of the human understanding the relation between the featured values of an instance and its prediction model. This is connected to the fact that human developer has little control on the entirety of the algorithm, as only the training data and certain training parameters can be tuned and changed by the developer. Once the training process has started, the models automatically adapt their parameters, and the meaning of each one of these adjustments is usually unknown for the human developer.

To increase object detection accuracy, two good practices were identified as a pattern in the best performing methods in sections 2.1.1 and 2.1.2. These techniques are the artificial dataset augmentation and performing the process transfer learning (by means of a custom dataset) on pre-trained networks on one of the popular benchmark datasets, in order to adapt the selected network to the desired application. Besides these good practices, three factors were also identified as crucial for a high detection performance: the adopted backbone network, the selected detection

architecture (e.g.: Faster R-CNN or Fast R-CNN) and the quality and an increased dimension of the training dataset.

Moreover, one main issue inherent to CNN-based architectures is overfitting [6], which refers to when the model starts learning in detail the noise of the training data to the extent that it negatively affects its performance. However, three main techniques were identified to tackle this issue: one is to increase generalization within the dataset, which can be made by means of data augmentation, other is to use a different dataset as validation data during training, and the last is to enable dropout (which is the deactivation of different neurons in every gradient descent step) in architectures with an increased number of layers and training epochs.

The ideal object detector, capable of tackling most of the identified limitations holds certain key characteristics, which are shown in Figure 2.7.

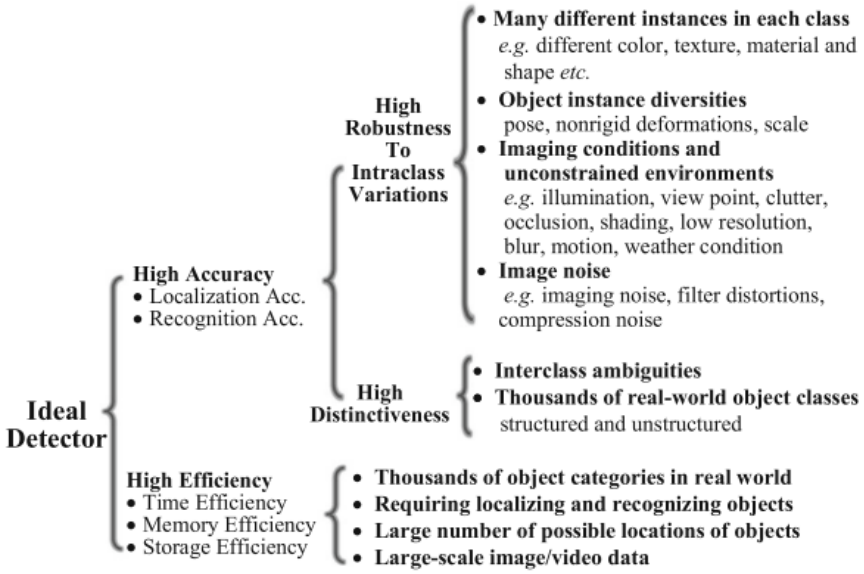


Figure 2.7 - Taxonomy of the main challenges in object detection [26].

Regarding the HRC assembly environments, the task recognition was found to be commonly made by means of human action recognition and prediction, i.e., by processing visual, inertial, or other kind of data related to the human worker. These inputs would then be used to trigger commands in the robotic arm to prompt its assistance to the human. This approach exposes a main drawback which is the high sensitivity false positives (FP) because a FP can have a high negative impact on the effectiveness of a HRC assembly system, for e.g., a human making a movement similar to the trigger-movement and commanding wrongly the robotic hardware to perform an action when it should not, could have drastic consequences on the working piece. On the other hand, using only the online object detections of the assembled components as triggers

to command the robotic hardware could tackle this FP-related sensitivity issue, as the robotic-trigger-to-command part would be directly disaggregated from the human worker.

Moreover, publications on deep learning methods focused only on processing visual data about the current state of the assembly, i.e., recognizing the topology, pose and position of the already-assembled components to enable task recognition, were not found during the literature research made for this study. This fact exposes an opportunity to create new developments in the field of HRC assembly environments by exploring a system that does not directly depend on human movements, gestures or sounds as triggers for commanding and guiding a certain process, but rather on the components detected within the coworking area, which is a problem of common object detection, something that is tackled with deep learning frameworks, and the selected path for this work.

3. BACKGROUND OF THE PROPOSED METHOD

3.1. Convolutional Neural Network

Convolutional Neural Networks (CNNs), also known as deep Convolutional Neural Networks (DCNNs) – which are the most representative models of what deep learning stands for– are detectors which consist of multiple convolution layers, pooling layers, and activation functions. Usually, each layer has a number of different convolutional kernels, a nonlinear activation function and often, a pooling mechanism to lower the dimensionality of the output data. It allows the extraction of higher representations of the image content, and unlike classic image recognition models where features were defined by hand, DCNNs process image's raw pixel data during training which enables automatic feature extraction, improving drastically the detection performance compared to classic models [27].

The basic CNN block is the hallmarked of this kind of networks and is schematized in Figure 3.1. It has the function of extracting the features from the input data, so the CNN is also given the name of feature extractor or detector [32] and, as shown in Figure 3.1, it is comprised by three main operations or stages: the convolution, the activation and finally the max pooling.

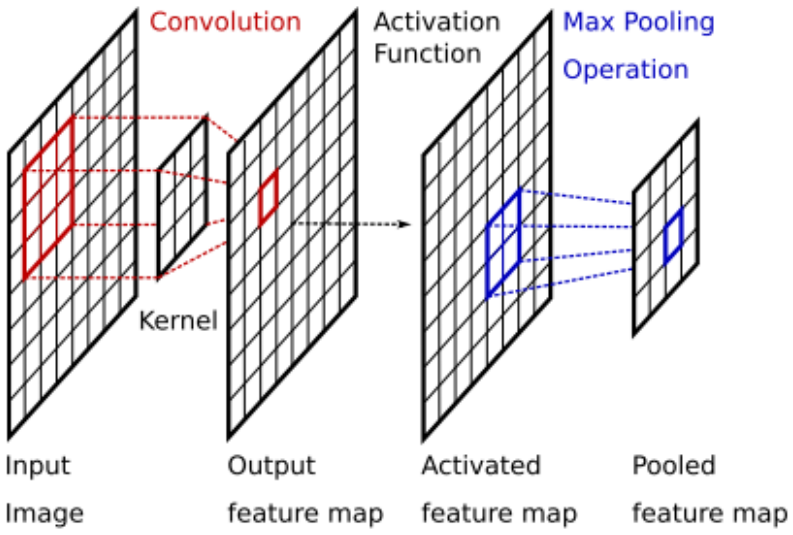


Figure 3.1 - Basic CNN block. A single layer is shown, which applies a kernel on an input filter, followed by an activation function and a max pooling operation [27].

3.1.1. Convolution

Instead of connecting every pixel of the input image to a layer of hidden neurons, the convolutional operation connects small and localized regions of the input regions to a neuron in a hidden layer and it is responsible for edge detection. These smaller regions within the input image are usually called local receptive fields (or filter or kernel), and the convolution operation can be interpreted as striding a local receptive field across the entire input image, pixel by pixel (or with a different stride length), and for each step of this process, a connection is made to a different neuron in the first hidden layer [32].

The convolution operation can be conceptually visualized as shown in Figure 3.2. In this example, given an input image with a 4x4 pixel size and a 3x3 filter or receptive field, the output of the convolution operation is a 2x2 feature map. The length of each stride is accordingly 1 pixel.

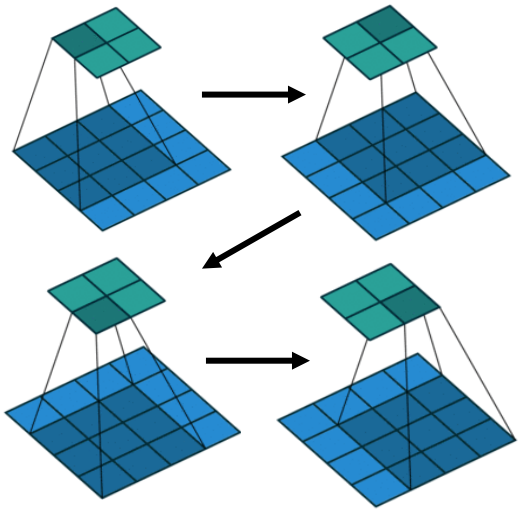


Figure 3.2 - A conceptual visualization of the convolution operation on a 4x4 input image (bottom blue square) by means of a 3x3 receptive field. Adapted from [43].

A single convolution over an input feature map x^{L-1} at a feature map from the previous layer L-1 and convolved with a 2D convolutional kernel (or filter, or weight) w^{L} can be mathematically expressed by equation (3.1).

$$x^{L-1} * w^{L} \tag{3.1}$$

This convolution appears, however, over a sequence of layers that comprise the entire deep learning model, and are subject to a nonlinear operation σ, such that, according to equation (3.2), a convolution between all the N^{L-1} input feature maps (x_i^{L-1}) and the corresponding weights $w_{i,j}^{L}$ plus a bias term b_j^{L}, happen.

$$x_j^L = \sigma\left(\sum_{i=1}^{N^{L-1}} x_i^{L-1} * w_{i,j}^L + b_j^L\right) \tag{3.2}$$

The nonlinear function σ (·), or activation function, is typically a rectified linear unit (ReLU) for each element (convolution) and it's expressed by equation (3.3) [26].

$$\sigma(x) = \max\{x, 0\} \tag{3.3}$$

3.1.2. Max Pooling

Pooling is mainly divided in average, global and max pooling, where the latter is the most widely used in DCNN architectures. It is usually done after a convolution, and respective activation function, as it corresponds to the down-sampling/up-sampling of feature maps [26].

The max pooling operation is conceptually schematized in Figure 3.3, highlighting its capability of decreasing the computational load by reducing the spatial volume of the input feature map, i.e., reducing the dimension of the input feature map.

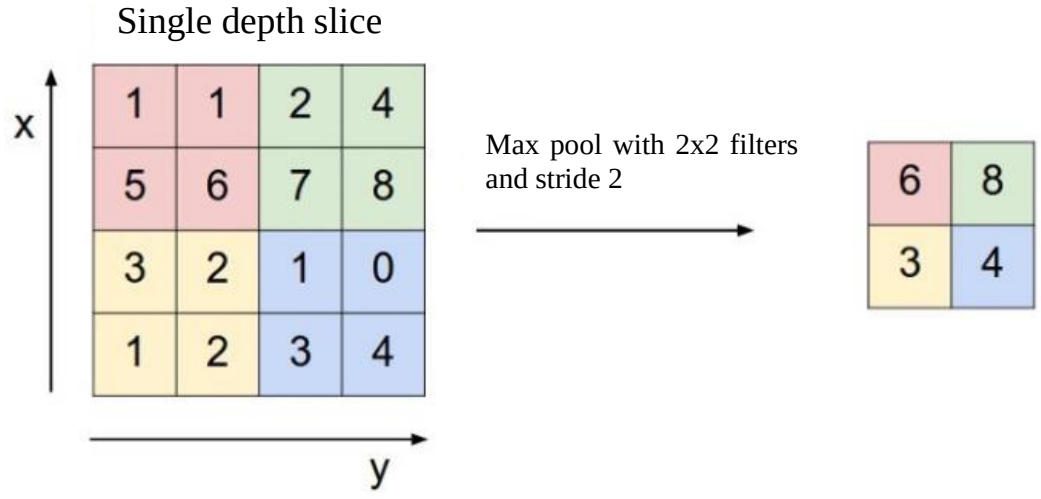


Figure 3.3 - Max pooling operation with a 2x2 filter and stride of 2. Adapted from [43].

3.1.3. Overview of a generic CNN architecture

The basic CNN block shown in Figure 3.1 is not by itself responsible for the classification of an input image, rather only for the features extraction, as previously stated. So, the CNN models require an additional set of layers after the basic CNN block responsible for performing the classification of each one of the object classes. These additional layers are a set of fully connected layers followed by an output layer (with the categories' labels). This set of layers involve weights,

biases, and fully connected neurons, and because they are responsible for elaborating the classification of the given input image, are usually referred to as classifier [26].

A schematic representation of a typical DCNN, the VGGNet [44], being fed with an RGB image, is shown in Figure 3.4. This network is comprised by 11 weighted layers, 8 convolution layers, 3 fully connected layers, 5 max pooling layers and a softmax classification layer. The last three fully connected layers take features from the top convolutional layer as input in vector form. The final layer is a C-way softmax function, with C being the number of classes that the network classifies. The whole network can be trained from labeled training data by optimizing an objective function (e.g., mean squared error or cross entropy loss) via stochastic gradient descent [26].

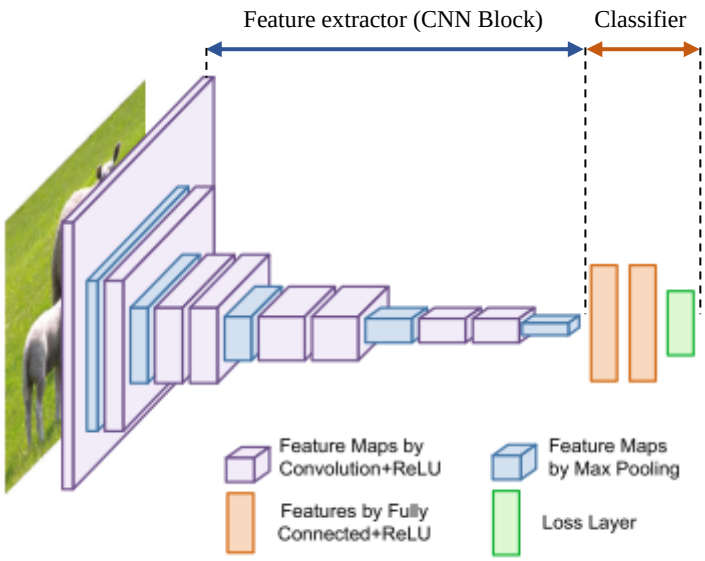


Figure 3.4 - Architecture of VGGNet, a typical CNN with 11 layers. Adapted from [26].

3.2. Main categories of CNN-based detection frameworks

Deep learning frameworks for object feature representation, recognition, or detection, can be organized into two main categories [26]:

1. Two stage detection frameworks, which include a pre-processing step for generating object proposals.
2. One stage detection frameworks, or region proposal free frameworks, with a single proposed method which does not separate the process of the detection proposal.

3.2.1. Region-based (two stage) frameworks

As presented in section 3.1.1. the convolution operation in CNNs was initially made by striding a filter across the entire input image. However, it is self-evident that an object to be detect in an image might not be displaced throughout its entire extent, but rather in a smaller region of interest within the image (e.g.: detecting two chairs in an image with these chairs inside of a big room without anything else in it). When this is the case, striding the filter throughout the entire image leads to unnecessary computational load and time.

Girshick et al. [34] tackled this problem when they proposed the Region-based Convolutional Neural Network (R-CNN) framework – an integration of AlexNet with a region proposal selective search – that instead of analyzing the entire image when it is normally not necessary, it only analyzed region proposals made in a pre-processor step by the selective search algorithm. These region proposals were smaller candidate frames of the entire input image where an object would be more likely to be exist. More information regarding the region proposals algorithm can be found in [45].

Therefore, region-based or two stage detection frameworks and the ones inspired on the R-CNN framework, and connected to a region proposals step, prior to both the feature extraction and the classifier set of layers.

However, despite the improvement in detection accuracy and training speed compared to CNN, the R-CNN framework still had notable bottlenecks: the extraction of CNN features from thousands of warped region proposals per image, which made training expensive in time and space, and slow object detection. Girshick, one of the creators of R-CNN, proposed the Fast R-CNN framework [46], increasing training and detection speed as well as detection accuracy. The main reason for this significant speed increase was that instead of performing the convolution operation in each one of the region proposals, it was instead processed only one convolution per image.

Nevertheless, further developments were made on the Fast R-CNN framework, as it stood out that runtime was still negatively impacted by the region proposals step. Hence, Ren et al. proposed the Faster R-CNN [47] framework, where the selective algorithm employed in R-CNN and Fast R-CNN to create the region proposals was replaced by a CNN, nominated Region Proposal Network (RPN), which was an efficient and accurate network capable of generating region proposals accordingly [26]. A high-level diagram of the Faster R-CNN framework for generic object detection, highlighting the different main parts of the framework can be seen in Figure 3.5.

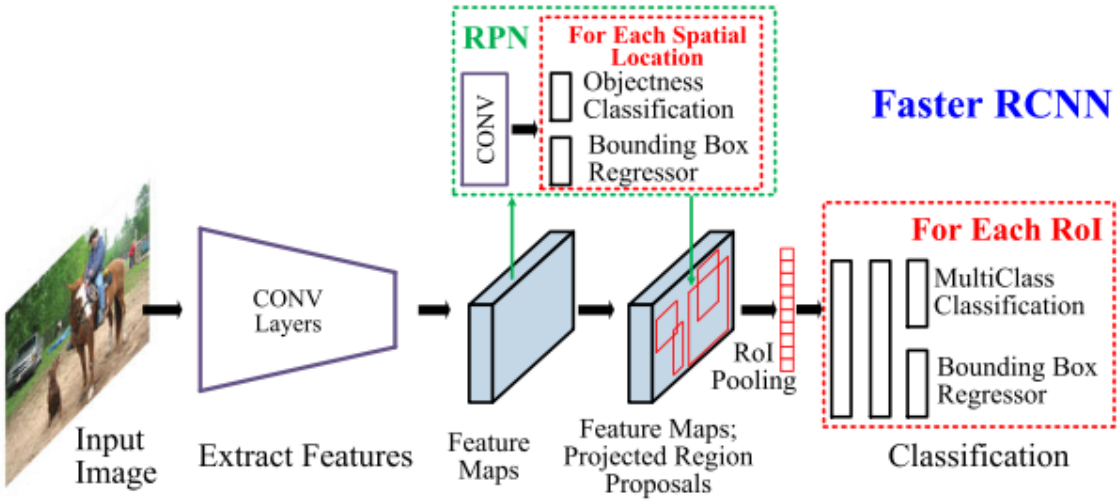


Figure 3.5 – High-level diagram of the Faster R-CNN framework for object detection [26]. RoI pooling refers to the region of interest pooling, i.e., a max pooling operation within the region proposals.

The developments from R-CNN till Faster R-CNN not only increased detection performance but also greatly decreased the detection time during tests. The degree of improvement of the detection time throughout these developments can be seen in Figure 3.6.

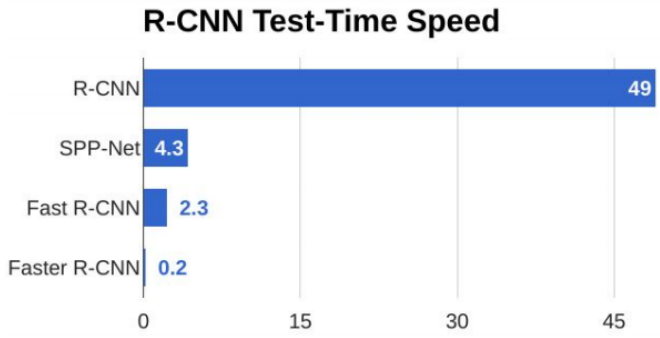


Figure 3.6 - Run test-time speed comparison between R-CNN, SPP-Net, Fast R-CNN, and Faster R-CNN [48]. Time is in seconds and same image input size for all models.

Nonetheless, the region-based frameworks still require a great computation effort to be ran and trained and are much slower than the one stage detectors, that will be explained in section 3.2.2. Despite this, all the state-of-art results on popular dataset benchmarks belong to these frameworks.

3.2.2. Region proposal free (one stage) detection frameworks

Despite the domination of region-based pipeline frameworks since R-CNN, such that the leading results on popular benchmark datasets are all based on Faster R-CNN, these frameworks are computationally expensive for current mobile/wearable devices, which have limited storage and computational capability. Hence, instead of trying to optimize the individual components of a complex region-based pipeline, researchers have begun to develop one stage detection strategies [26].

These one stage detection frameworks refer to architectures that, unlike the region-based ones, directly predict class probabilities and bounding box offsets from full images with a single feed-forward CNN in a monolithic setting that does not involve region proposal generation, encapsulating all computation in a single network. Since the whole pipeline is a single network, it can be optimized end-to-end directly on detection performance.

The effects of the elimination of the region proposals section, had a great decrease in computational load and increase on the detection speed. As a matter of perspective, for a 300x300 input image, the one stage detector SSD [39] achieved 74.3% *mean average precision* (mAP) on the VOC2007 test at 59 *frames per second* (FPS) versus Faster R-CNN 7 FPS / mAP 73.2% or first version of YOLO [38] 45 FPS /mAP 63.4% [26].

Despite the several region proposal free frameworks proposed during the years, one of the most relevant branches of research and real-life applications is undoubtedly the one based on the YOLO architecture, which was firstly proposed by Redmon et al. [38]. In order to highlight the contrast between this kind of frameworks and the two stage detectors shown in Figure 3.5, a high-level diagram of the YOLO framework – a representative framework of one stage detectors – is shown in Figure 3.7.

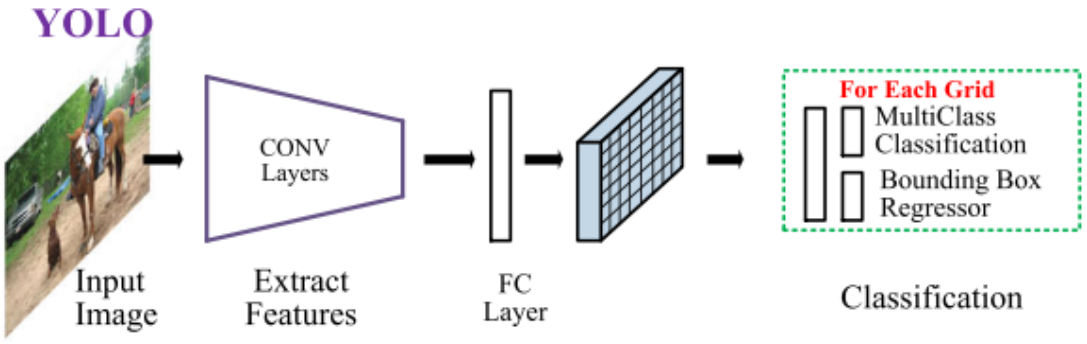


Figure 3.7 - High-level diagram of the Faster R-CNN framework for object detection [26]. FC layer stands for fully connected layer.

The first version of the YOLO framework was the starting point for further developments within this family of algorithms that have the characteristic of diving the input images into a grid system in which each cell in the grid is responsible for detecting objects within itself. The first development by Redmon et al. was YOLOv2 (also given the name of YOLO9000) [49], an improved version of YOLO, capable of detecting 9000 object categories in real-time and at higher speeds and accuracy than the previous version, however it still struggled in detecting smaller objects. Followed by this, the YOLOv3 [50] framework was also proposed by Redmon et al., to tackle the poor performance of the previous with the smaller objects. This was an improved version of YOLOv2 in detection speed and performance, as well in detecting smaller objects by adding detection at multiple scales to help detect smaller objects and separating the loss function used for training into mean squared error for bounding box regression and binary cross-entropy for object classification to help improve detection accuracy.

4. GENERIC METHODOLOGY

Throughout this chapter the methodology of creating and implementing a generic human-robot collaborative (HRC) system where decisions are automatically made based on visual data from the processes or operations occurring inside the common working space, is presented.

The creation of a HRC system can be divided into 2 distinct parts. The first part, refers to the development and validation of the computer vision-enabling part of the system which, as identified during the literature review for this work, is commonly done by means of deep learning frameworks. On the other hand, the second part refers to the implementation of the computer vision framework with the selected hardware to safely teamwork and perform certain movements and tasks in synchrony with a human worker, e.g., a robotic arm fetching tools or other objects for the human.

4.1. Deep learning for enabling computer vision

The initial step for creating the HRC system is to select and adapt a pre-existing framework that enables the system to understand its surroundings in which it would be operating, and as stated throughout section 2, deep learning CNN-based approaches are the most widely employed for visual scene understanding, and the frameworks selected for this study.

Enabling computer vision in a custom HRC environment by means of deep learning frameworks requires a sequence of milestones to be accomplished:

1. Firstly, it is necessary to define the physical human-robot working space where both the human and the collaborative piece of hardware (e.g., a robotic arm) will teamwork.
2. Afterwards, a set of relevant images of the objects and scenes of interest to detect for the correct operation of the HRC environment need to be acquired, for the further training of the deep learning models.
3. After the image acquisition, the manual labelling and bounding boxes for each one of the images with each one of the predefined object class categories has to be made, concluding in this way the creation of the dataset.
4. Succeeded to this, the specific pre-trained deep learning models for enabling computer vision are selected and prepared for training. More specifically for this study, 2 different

major branches of deep learning in object detection were selected for further performance comparison: the Faster R-CNN [47] branch and the YOLO [38] branch.

5. The dataset is then divided into training dataset, validation dataset to be used during the retraining of each of one the deep learning models and testing dataset to assess and compare their performance.
6. After this, the training dataset can be augmented to create higher intraclass variations and the images resized to the same input size of the first layer of each one of the detectors.
7. Sequentially, all the chosen detectors are retrained (process also known as transfer learning or fine-tuning) over the custom dataset.
8. The object detectors can then be compared with the metrics of *precision vs recall* curves (PR curves) and *mean average precision* (mAP), whereas other metrics could have also been selected depending on the desired application (e.g., ROC curves and the area under these curves – AUC).
9. At last, the detection speeds of each one of the object detectors can also be measured over a same image, and these values used for comparison and decision-making.

4.2. Implementation with the physical hardware

The second and last step for creating the HRC environment is the development of an integrator algorithm that will be the brain of the system and allow the deployment of the deep learning framework with the collaborative piece of hardware. The final goal of this algorithm is that the presence of certain key detected objects acts as a trigger for commanding the hardware to perform a certain movement at an exact time instance.

The feasibility of the mentioned system is not only connected to the object detection performance of the chosen deep learning model, but also to a key higher level of understanding regarding the entire process to be done, i.e., according to which components currently being detected, it is necessary that the system possesses the ability of correlating this visual input to the current progress of the process, defining accordingly which actions the hardware still needs to perform, and which ones are already done.

To achieve this capability of a higher level of understanding that correlates the detected components with the pace of the entire process sequence, it is necessary to complete the subsequent sequence of milestones:

1. Define the entire assembly task sequence and allocate the tasks to be done by the human and the tasks to be done by the collaborative piece of hardware.
2. Define and program the movements to be done by the robot in order to complete its pre-allocated tasks.

3. Decide which key components should be detected in order to trigger the command of each one to be done by collaborative piece of hardware, at the required time instance.
4. Develop an algorithm that integrates the deployment of this HRC system.

The algorithm used for deploying the HRC system that automatically commands the collaborative hardware can be structure as shown in the flowchart presented in Figure 4.1.

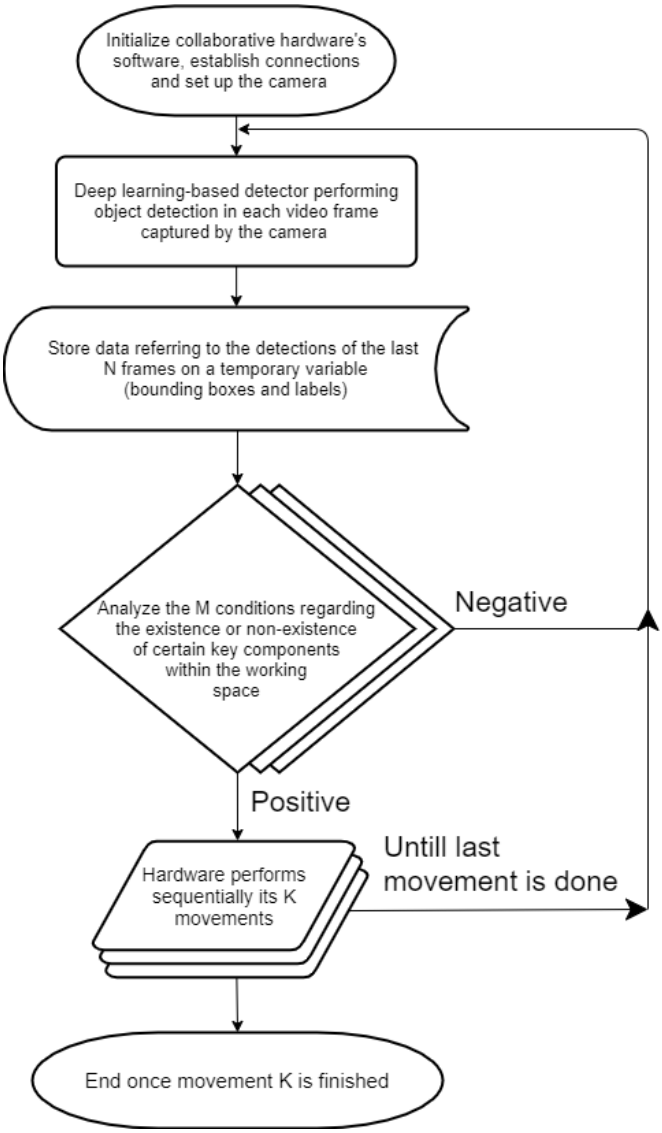


Figure 4.1 - Deployment algorithm of the HRC environment: Computer vision framework with the collaborative piece of hardware.

The initial steps of the deployment algorithm are initializing all the required software for the HRC interface, setting up the camera to continuously acquire images and resizing its output to be used as an input by the deep learning object detector. Then, the algorithm processes and performs object detection on these input frames and the detection results (labels and bounding boxes) of a N number of frames are stored in a temporary variable. Afterwards, it is necessary to

evaluate whether in a certain M number of these N frames (where M<N) the predefined objects and other conditions are satisfied (e.g., final joint positions of a robotic arm) – where these conditions and objects are coupled to a specific movement and task of the predefined assembly sequence. If all the conditions and the minimum number of detected objects instances for a specific task **are satisfied**, the system commands the collaborative piece of hardware to perform a movement or a set of movements for this task and makes the system move forward to the next task and movement, overriding the temporary variable with the detections of the N frames with new ones and proceeding to new detections. On the other hand, if all the conditions and the minimum number of detected objects instances **are not satisfied**, no commands are triggered in the hardware, and the temporary variable with the detections of the N frames is overridden with new detections continuously until all the conditions for a certain movement are satisfied. Once the last hardware movement, of an entire group of a K number of predefined hardware movements is completed, the system is shutdown, because the entire assembly process will be finished.

5. EXPERIMENTAL PROCEDURE

This chapter describes in detail the experimental part of this work, which was divided in two main stages. The first stage was referred to the training and comparison of performance between 4 different widely used deep learning algorithms for enabling computer vision, with the relevant object categories for the desired collaborative assembly of a pre-chosen mechanical component. On the other hand, the second stage was referred to the deployment of the best performing algorithm with a robotic arm by means of a system capable of managing the HRC assembly process of a chosen mechanical component.

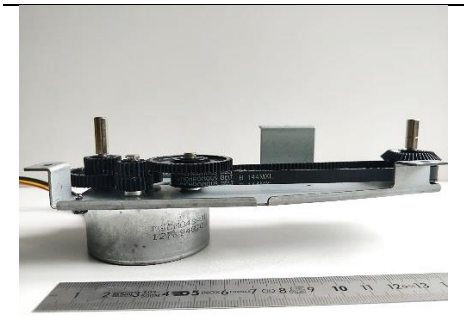
5.1. Equipment

5.1.1. Selection of equipment and components

The selected equipment, components, and software to perform the experimental procedure are presented in Table 5.1.

Table 5.1 - Main characteristics of the equipment, components and software used for this work.

Equipment/component/software	Relevant characteristics	Representative figures
Robotic arm (plus stabilizing structure)	Model: **ViperX 300 Robot Arm Trossen Robotics** **5 degrees of freedom version** Full **360º rotation** Precision: **± 1mm** Working payload: **750g** Software: **ROS, RViz and Gazebo**	 
Camera (plus tripod)	Model: **SONY DSC-RX0M2G** Lens: **Wide-angle ZEISS Tessar T* 24mm F4** # of Pixels: **15.3 Megapixels** Recording quality: **up to 4k movies** Sensor type: **1.0 type (13.2mmx8.8mm) Exmor TS CMOS sensor, aspect ratio 3:2**	 

Computer	Model: **MSI GP76 Leopard 10 UG** CPU: **i7-10870H** GPU: **NVIDIA GeForce RTX 3070 8GB GDDR6 Laptop** RAM: **16GB DDR4** Storage capacity: **1TB SSD** OS: **Windows and Ubuntu**	
Mechanical component to be assembled plus a screwdriver	Assembly with **8 different types of sub-components:** 1 base, 4 cogwheels, 1 belt, 2 screws and 1 stepper motor**.**	  
Linksys BEFW11S4 Wireless-B Broadband Router	Standard: **802.11b** Ethernet speed: **10/100 Mbps** Frequency: **2.4 GHz**	

MATLAB	Required Add-ons: **MATLAB Support Package for USB Webcams.** **Computer Vision, Deep learning, Image Processing, Image Acquisition, Statistics and Machine Learning Toolboxes.** **ROS Toolbox and Robotics System Toolbox**. **Image Labeler App.**	

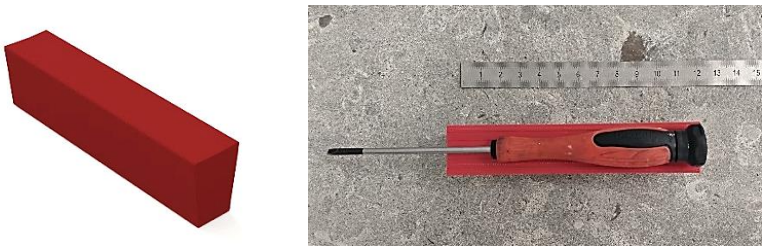
5.1.2. Design and additive manufacturing of support components

To ease the assembly process of the mechanical component and facilitate tasks such as screwing or the alignment of two or more components, an assembly support was manufactured in Polylactic Acid (PLA) filament by means of Fused Deposition Modeling (FDM) technology. The manufactured support shown in Figure 5.1 a), also has the function of maintaining all the components parallel to the working space surface to enable a more ergonomic assembly and maintain the components in a roughly fixed pose throughout the entire process. Figure 5.1 b) and c) illustrate the pose of the bottom and top of the fully assembled mechanical component when mounted on top of the PLA support.

a) b) c)

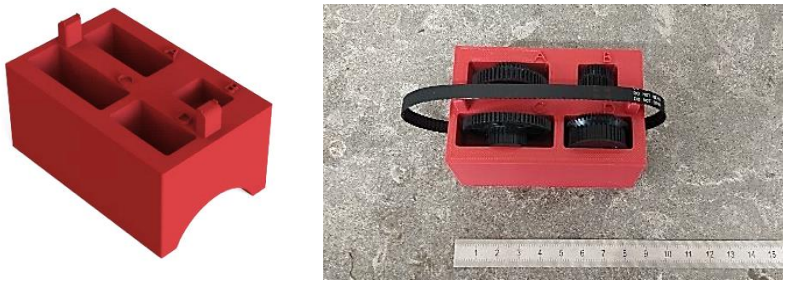
Figure 5.1 - a) CAD representation of the assembly support. b) Bottom part of the component mounted on the support. c) Top part of the component mounted on the support.

Besides the support, three more parts were manufactured: two similar screwdriver holders with the function of holding a screwdriver in a specific position and pose, and one box suitable for storing all the cogwheels and the belt. Figure 5.2 and Figure 5.3, a) and b) show a representation of the screwdriver holder and the wheel's box, respectively.



a) b)

Figure 5.2 - a) CAD representation of the screwdriver holder; b) Screwdriver mounted on the holder in a desired pose and position.



a) b)

Figure 5.3 - a) CAD representation of the wheel box. b) Cogwheels and belt stored inside the wheel box.

5.2. First stage: Detector creation, benchmarking, and selection

5.2.1. Experimental setup

For the experimental procedure of the first stage, the used layout is illustrated in Figure 5.4. This layout was put in place to acquire all the necessary images for the dataset that was further used to fine-tune 4 different deep learning models. On the other hand, the training and performance assessment of said algorithms, which are inherent to this first stage, were done using the computer specified in Table 5.1.

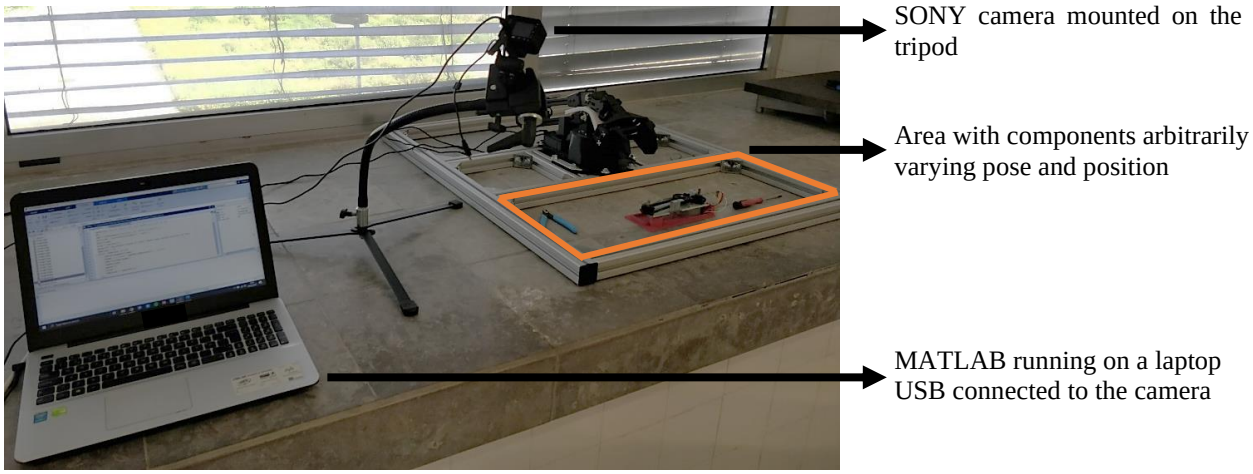


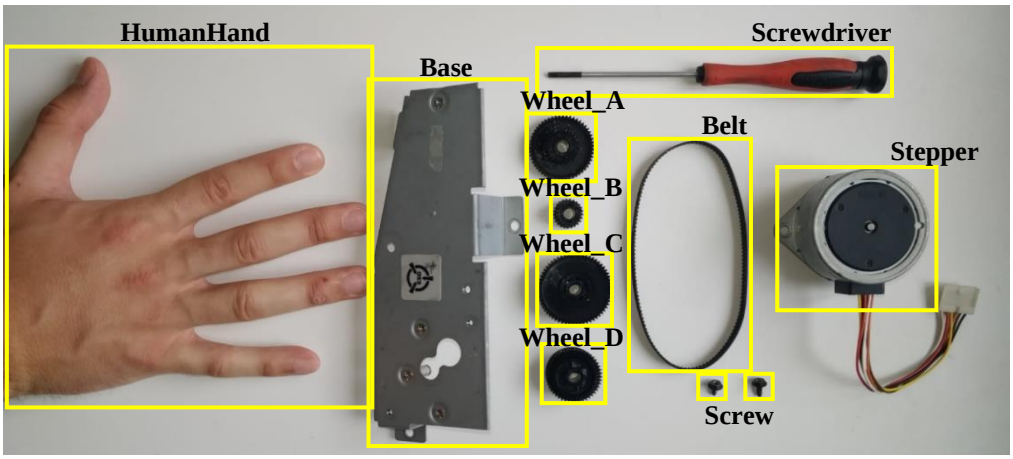
Figure 5.4 - Experimental setup for dataset creation during the first stage of the experimental procedure.

5.2.2. Methodology

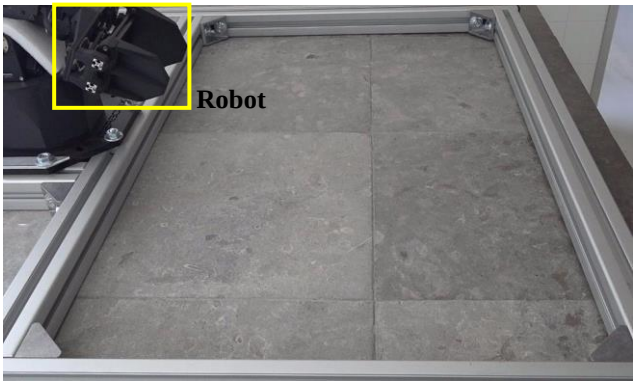
5.2.2.1. Dataset

After setting up the experimental layout for this first stage, the dataset of images was created. This dataset was comprised of 3858 images with each of an initial size of 576x1024 pixels, and they were acquired with the SONY DSC-RX0M2G camera, connected to MATLAB as a USB camera. For this purpose, a script was developed in MATLAB, and it can be analyzed in both appendixes 9.1 and 9.2.

Following this step, all the acquired images were manually labeled using Image Labeler App from MATLAB to create a training dataset comprised of 11 different object category classes. To tackle class imbalance, it was previously aimed to have around 1100 instances of each one of the object categories. These 11 categories were labelled: *'Base', 'Stepper', 'Belt', 'Robot', 'Screwdriver', 'Screw', 'HumanHand', 'Wheel_A', 'Wheel_B', 'Wheel_C'* and *'Wheel_D'*, and a visual representation of these object classes is presented in Figure 5.5 a) and b), where the first represents 10 of the 11 categories and the latter represents only the '*Robot*' class, which refers to the gripper of the robotic arm.



a)



b)

Figure 5.5 - a) 10 out of the 11 object class categories. b) *Robot* object class category.

So that the pre-defined goal of roughly 1100 instances for each class would be achieved, the images acquired by the camera were a mixture of images taken while a human was continuously performing the assembly of the mechanical component and images showing a smaller set of components or just an individual object class per image. An overview of the global dataset object instances for each object label is shown in Table 5.2.

Table 5.2 - Overview of the number of object class instances in the dataset and in how many images each label class exists.

Label	Label count	Image count
Base	1078	1074
Stepper	1104	1092
Belt	1071	1071
Robot	1059	1059
Screwdriver	1104	1103
Screw	1140	881
HumanHand	1126	770
Wheel_A	1107	1107
Wheel_B	1078	1076
Wheel_C	1072	1072
Wheel_D	1053	1053

The entire unlabeled and labeled dataset used for this application can be found in appendix 9.1.

5.2.2.2. Algorithms and training

After the creation of the dataset, 4 different deep learning models were subjected to a transfer learning process, i.e., pre-trained models for image recognition were trained once again but with the dataset created for this human-robot collaborative assembly application, adjusting in this way its weights and biases for the desired object detections. The used software was MATLAB.

Before the training of any algorithm, the initial dataset with the 3858 images, was once randomly divided into training (80% of the initial dataset), validation (10%), and test (10%) datasets, and kept the same for the 4 models. The training and validation datasets were given use during the process of training and the test dataset was used to assess the performance of each one of the models. In addition, data augmentation was done on the training dataset, where 50% of all the images were randomly flipped horizontally as an effort of tackling problems as overfitting and prompt increased intraclass variation within the training dataset.

The 4 to-be-compared deep learning models were Faster R-CNNs ResNet-50 and ResNet-101 [36], YOLOv2 [49] and YOLOv3 [50], i.e., 2 models based on the Faster R-CNN framework and 2 models based on YOLO. Several relevant characteristics about these algorithms, as well as the training parameters selected for each one, are summarized in Table 5.3.

As it can be observed, the epochs and mini-batch sizes inherent to the Faster R-CNN based models were greatly lower than the YOLO based models; this was due to the greater computational cost of running and training both ResNet-50 and ResNet-101 compared to Darknet-19 and Darknet-53, resulting in a greater training time and memory usage of the available graphics processing unit (GPU). The number of epochs and the mini-batch size were decided on a trial-and-error process to not overshoot the memory of the available GPU during training. The input size of the Faster R-CNNs was altered in order to maintain a similar width to height ratio of the initial ratio of the images acquired by the camera (576x1024), and the input size of both YOLOv2 and YOLOv3 were kept the same as the original pre-trained models offered by MATLAB.

Table 5.3 – Deep learning models' characteristics and training parameters.

Architecture	**Faster R-CNN**	**Faster R-CNN**	**YOLOv2**	**YOLOv3**
Backbone network	ResNet-50	ResNet-101	Darknet-19	Darknet-53
Input layer size in pixels [height width]	[224 396]	[224 396]	[256 256]	[608 608]
Pretraining dataset	ImageNet [31]	ImageNet	ImageNet	COCO [13]
Number of initial object categories	1000	1000	1000	80
Hardware (GPU)	NVIDIA GeForce RTX 3070 8GB GDDR6 Laptop	NVIDIA GeForce RTX 3070 8GB GDDR6 Laptop	NVIDIA GeForce RTX 3070 8GB GDDR6 Laptop	NVIDIA GeForce RTX 3070 8GB GDDR6 Laptop
Training parameters				
Solver	Stochastic gradient descent (sgdm)	sgdm	sgdm	sgdm
Number of epochs	4	3	50	30
Mini-batch size	1	1	32	4
Initial learning rate	1E-3	1E-3	1E-4	1E-3
Dropout	Yes	Yes	Yes	Yes
L_2 regularization	1E-4	1E-4	1E-4	5E-4
Validation dataset	Yes	Yes	Yes	Yes

5.2.2.3. Performance assessment metrics

Succeeding the training of the 4 deep learning models, only the best performing was deployed for the second stage of this experimental procedure, described in section 5.3.

Because the existence of a positive class was more interesting than the existence of a negative class for the system to be deployed, i.e., it was more relevant to know whether the detected object was the actually the correct one rather than knowing if it was a false or true negative, and also because the problem was of multi-class object detection, rather than a binary

classification problem [51], the metrics used to assess and compare each one of the algorithms were the *precision vs recall* curve, also known as PR curve, and its *mean average precision* (mAP). Besides this, the detection speeds of each one of the object detectors was also measured. For this, a same image was processed 50 times by each one of the deep learning models, and the average value of the detection speeds over all detections was the result used for comparison.

5.3. Second stage: Deployment

5.3.1. Experimental setup

The experimental setup for this second stage was different from the one used in the first stage of this experimental procedure.

In this setup, it was necessary to run Ubuntu - Linux operating system (OS) on a Virtual Machine because the Robot Operating System (ROS) used to command the ViperX 300 robotic arm ran on Ubuntu and the native OS of the used laptop was Windows. Moreover, since the deep learning model would be running on MATLAB, the robotic arm was also indirectly commanded with the same software. A static IP address was established for Windows and Ubuntu OSs, to enable the connection between MATLAB running on Windows and the ROS Master node running on Ubuntu and for this purpose, an external router was used to create a local network to allow the usage of a secure IP, not imposing any communication limitations between both OSs. In addition, it was necessary to determine fixed world coordinates for the items that would be grasped by the robot. A schematic representation of the entire layout for this deployment stage can be seen in Figure 5.6.

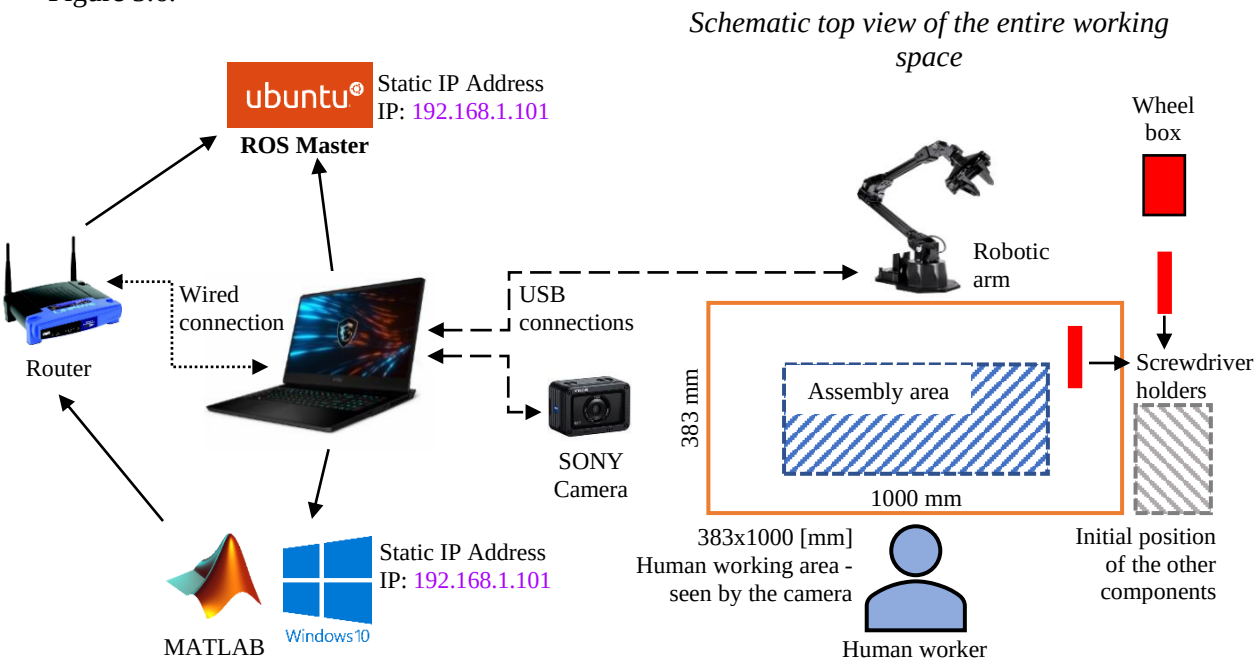
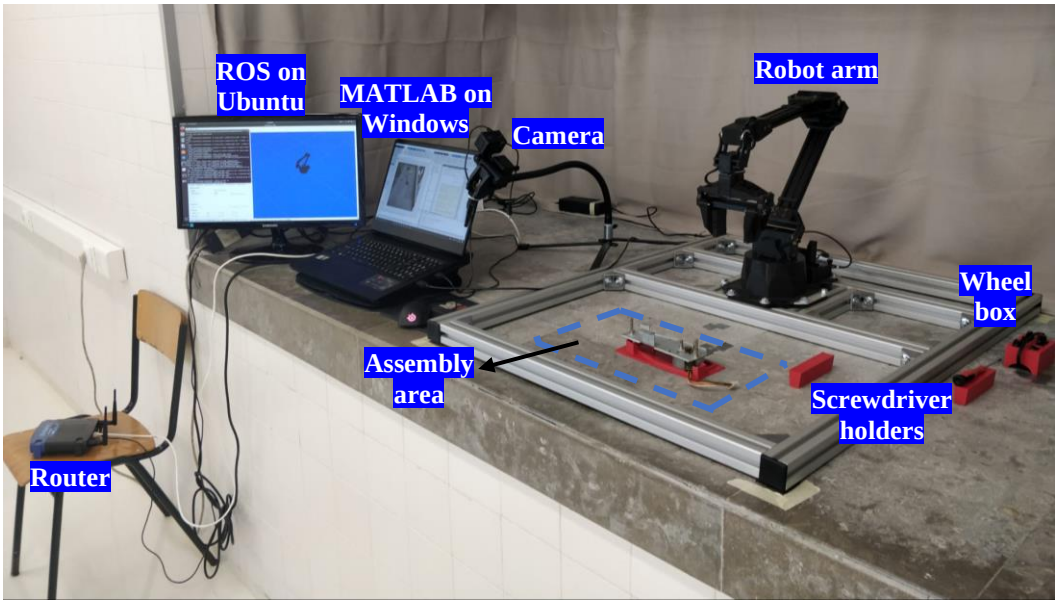
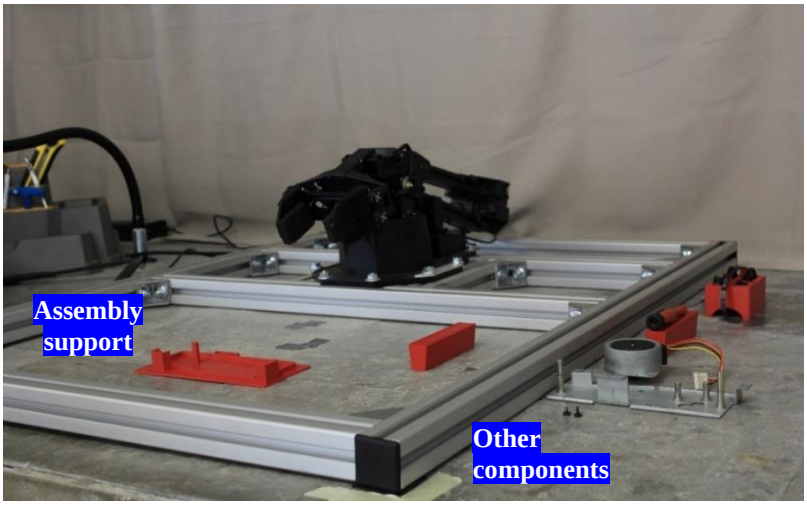


Figure 5.6 - Deployment stage experimental layout. At the left, it is represented the laptop running simultaneously MATLAB on Windows and ROS on Ubuntu inside a virtual machine, wired connected to the router, and USB connected to the camera and robot. At the right, it is shown a schematic top view of the working space.

The physical setup of the experimental procedure of the deployment stage, the initial positions of all the components and robot before the beginning of the assembly process as well as the point of view of the camera can be seen in Figure 5.7 a), b) and c), respectively.



a)



b)

c)

Figure 5.7 - a) Physical overview of the deployment's experimental setup. b) Initial position of the robot and other components, as well as support parts. c) Camera's point of view.

5.3.2. Methodology

The purpose of this deployment stage was to create a proof-of-concept system capable of managing the human-robot collaborative assembly process of the selected mechanical component, by providing the robot the capability of autonomously deciding when to perform a certain pre-programmed movement in order to assist the human in a specific task.

Firstly, the assembly tasks sequence and topology were defined, as well as the additional necessary equipment besides the components to be assembled (e.g., the screwdriver or the box that stored the cogwheels and belt), as it is presented in detail in Table 5.4.

Table 5.4 - Assembly task sequence and description.

Task number	Task Description	Extra-component necessities
1	Mount the bottom part of *Base* component on the assembly support	Assembly support
2	Mount the *Stepper* on *Base.*	N/A
3	Insert both S*crew* components (bolts) in the correct holes to fix *Stepper* to *Base.*	N/A
4	Screw the bolts with the screwdriver.	Screwdriver
5	Turn the half-assembled mechanical component upside down (top part).	N/A
6	Assemble wheels A,B,C and D and the *Belt.*	Wheel box storing cogwheels and belt

Once the assembly sequence was established, the movements that the robotic arm was expected to do were defined. The ViperX 300 robotic arm was programmed to perform the 4 different movements described in Table 5.5, with the goal of assisting the human during the assembly process, e.g., to pick up a tool which was outside of the reach of the human.

Table 5.5 - Robot movements sequence and description.

Movement	Movement description
1	Pick up screwdriver from outside the human working area (HWA) and drop it on the screwdriver holder in it.
2	Pick up the wheel box from outside the HWA and drop in a specific place inside of it.
3	Pick up the screwdriver laying on the inner screwdriver holder and carry it to the outer one.
4	Pick up and take away the empty wheel box to the initial position outside the HWA.

The aforementioned robot movements were only to be performed according to the progress of the assembly tasks done by the human. It was defined that the robot would:

- Perform movement 1 only when the human was performing inserting the bolts in place (task 3).
- Perform movement 2 only when the human started screwing (task 4).

- Perform movement 3 only when the human turned the half-assembled mechanical component upside (task 5).
- Perform movement 4 only when all the components were assembled (completed task 6).

In order to correlate the video data retrieved by the camera with the desired higher level of understanding of the assembly progress, a deployment algorithm was developed in MATLAB. This algorithm allowed the deep learning model to process each frame captured by the camera in an online loop, and whenever certain key components would be correctly detected, the robot would know when to perform one of the 4 pre-defined movements. The flowchart presented in Figure 5.8 underlines the structure of the mentioned deployment algorithm. For this case study, the number of frames, N – from which the detection results would be stored on a temporary variable – was 10, and regarding the movements 1, 2, 3 and 4 (K = 4), the values of M were 7, 5, 3 and 2, respectively. The MATLAB code regarding this algorithm is presented in appendixes 9.1 and 9.3.

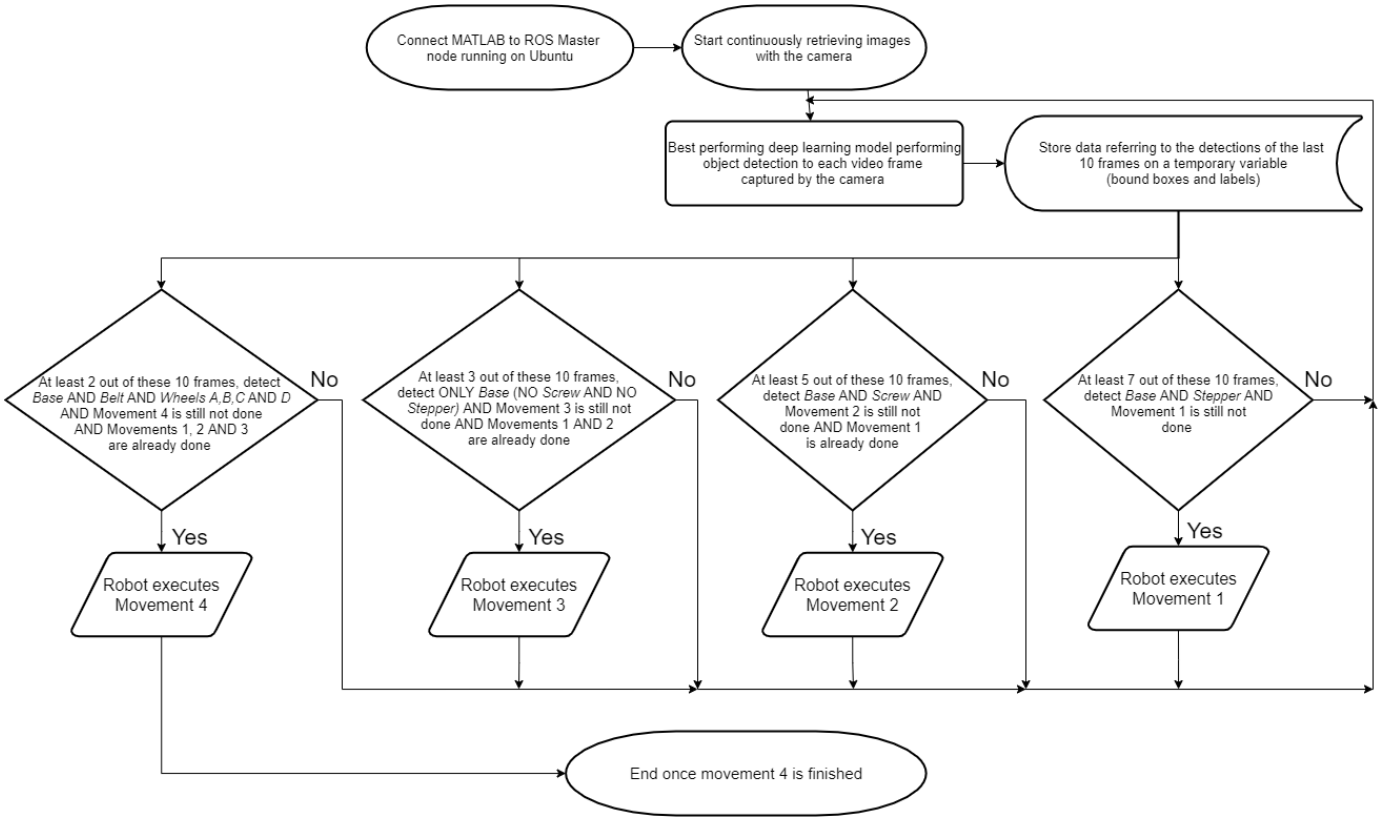


Figure 5.8 - Deployment algorithm of the human-robot collaborative assembly system with ViperX300 robot arm.

6. RESULTS AND DISCUSSION

In this chapter, the results obtained from the 1st and 2nd stages of the already mentioned experimental procedure are exposed and discussed upon.

6.1. First stage results – selected algorithm for further deployment

As stated in section 5.2.2.2, 10% of the initial dataset was not used during the training of the deep learning models. However, it was used as a test dataset for the performance assessment and comparison between the 4 different object detectors.

The performance assessment metrics selected for this work, the *precision vs recall* curves (PR curves), with the respective the *mean average precision* (mAP), are presented in Figure 6.2 a), b), c) and d).

The precision – the vertical axis of the PR curves – is a measure of the deep learning model's ability of correctly classifying each object class instance, i.e., how accurate the true positive (TP) predictions are. So, it is defined as the ratio between all the true positives and the total predicted positives – true positives plus the false positives (FP) – as stated in equation (6.1).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{6.1}$$

On the other hand, the recall, or also known as sensitivity – the horizontal axis – is a measure of the classifier's ability of correctly identifying all the object instances of a determined class, i.e., how well the positive classes are predicted. It is defined as the ratio of TP and total of ground truth positives – the sum of TP with the false negatives (FN), as shown in equation (6.2).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.2}$$

The mAP is instead calculated by taking the mean of the average precision over all queries regarding each object class, and/or over all *intersection over union* (IoU) thresholds, depending on different detection challenges that might be necessary to tackle. For this work only the mAP with IoU of 0.5, a common mAP metric, was considered.

The IoU is in turn the ratio of the area of intersection and area of union of the predicted bounding box and ground truth bounding box, as it is illustrated in Figure 6.1.

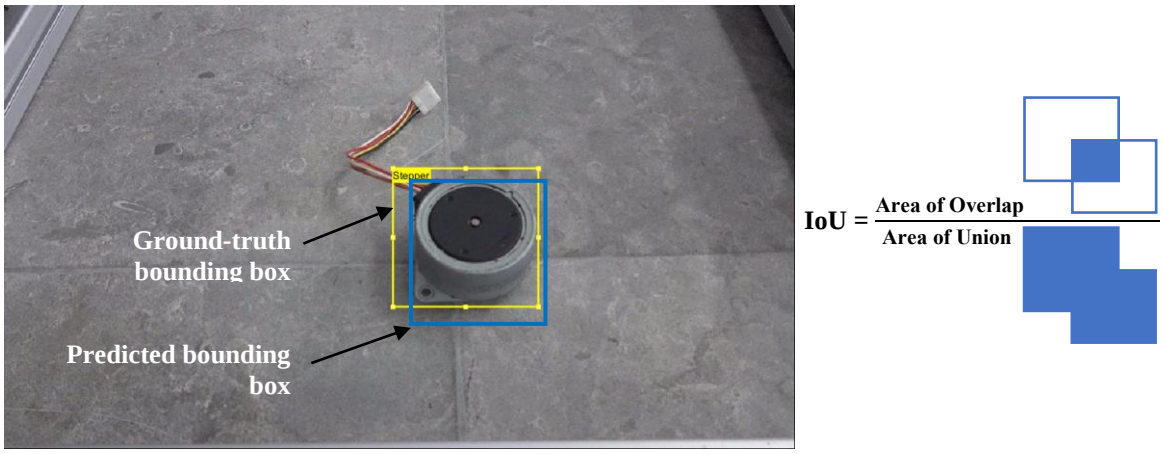
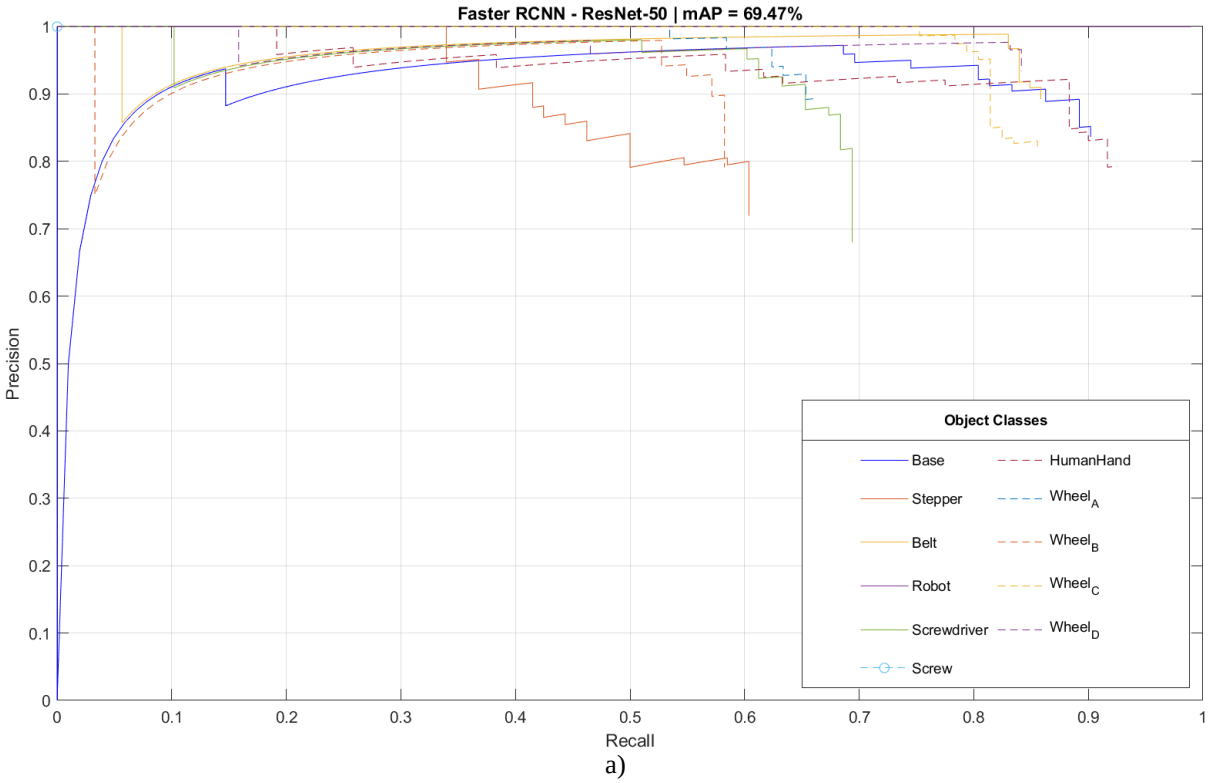
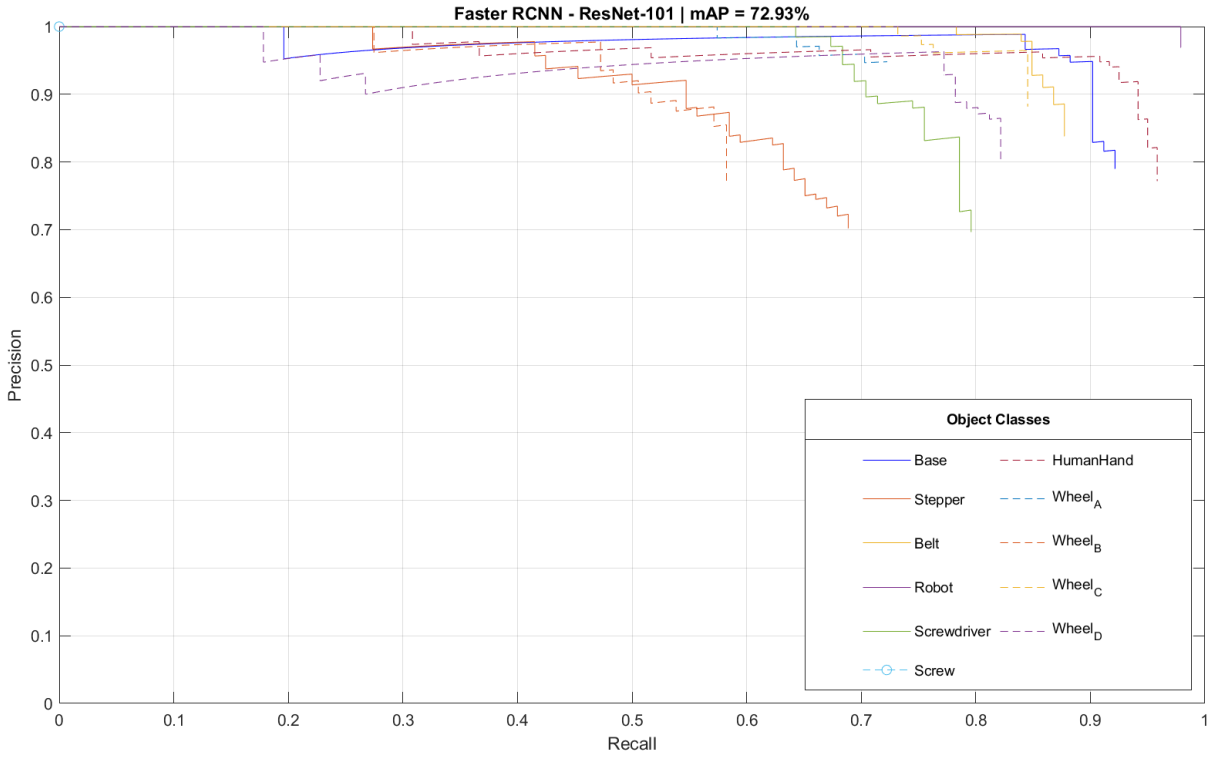


Figure 6.1 - Representation of the concept of Intersection over Union (IoU).

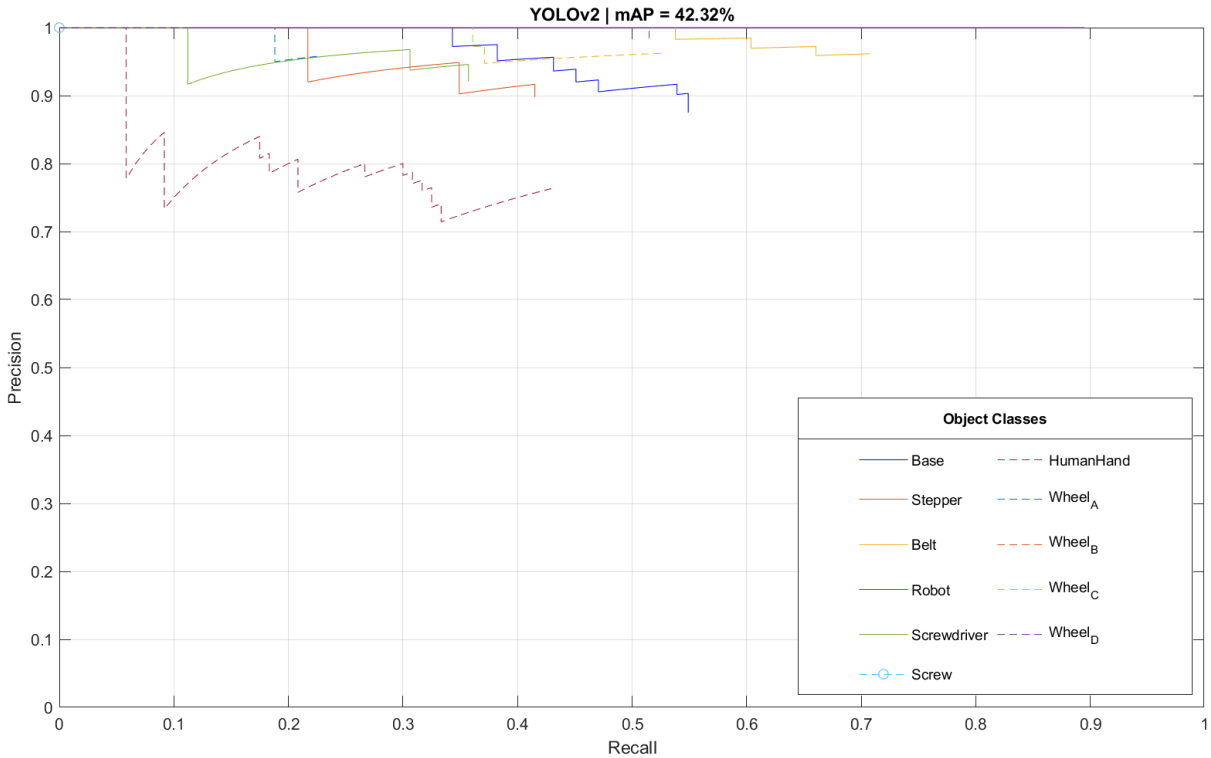
Consequently, the PR curves shown in Figure 6.2, express the trade-off between precision and recall for different threshold values and for each one of the deep learning models. A good classifier is one that showcases a high area under the curve referring to each one of the object classes as well as an approximately horizontal trend throughout the different values of precision and recall. The area of the curve for a single object class in a perfect classifier equal to 1.



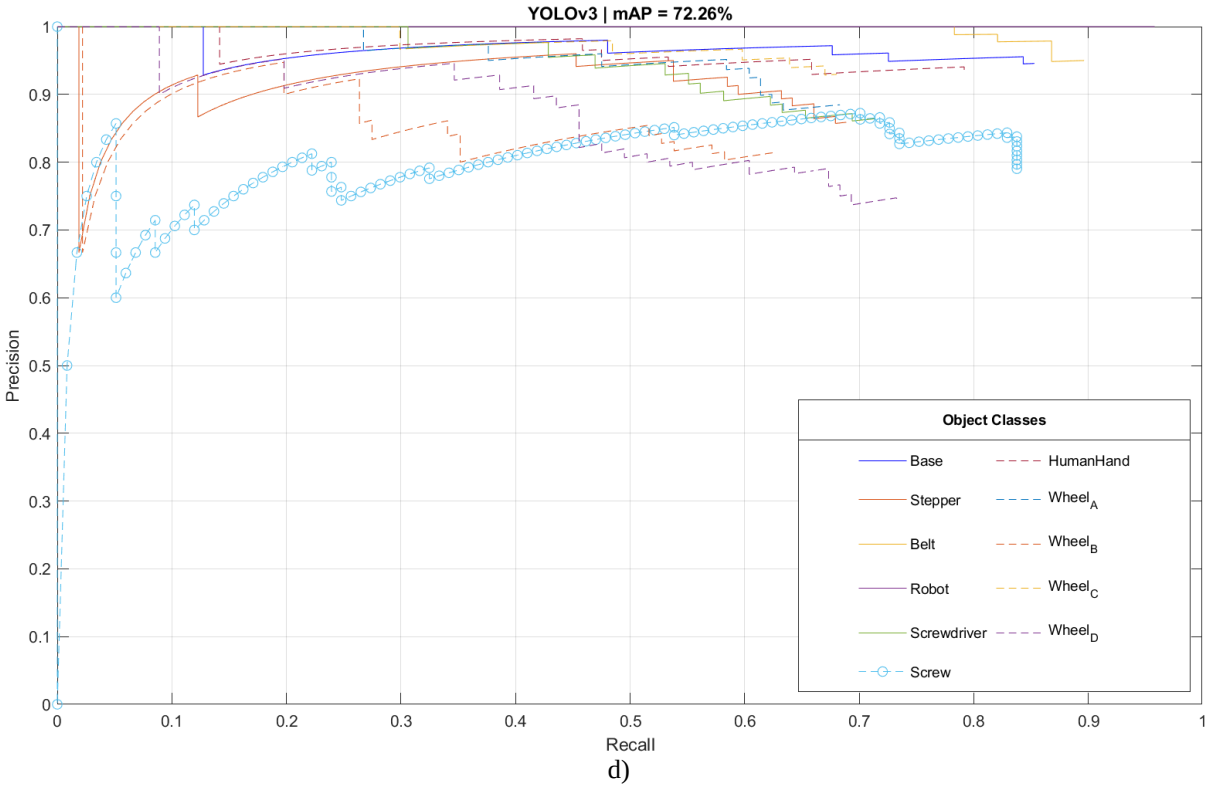
a)


Faster RCNN - ResNet-101 | mAP = 72.93%
Precision
Recall
Object Classes
Base
Stepper
Belt
Robot
Screwdriver
Screw
HumanHand
Wheel A
Wheel B
Wheel C
Wheel D

b)


YOLOv2 | mAP = 42.32%
Precision
Recall
Object Classes
Base
Stepper
Belt
Robot
Screwdriver
Screw
HumanHand
Wheel A
Wheel B
Wheel C
Wheel D

c)



d)

Figure 6.2 - PR curves and mAP of the 4 different deep learning classifiers. a) Faster R-CNN – ResNet-50. b) Faster R-CNN – ResNet-101. c) YOLOv2 – Darknet-19. d) YOLOv3 – Darknet-53.

A condensed overview of the mAP with an IoU threshold of 0.5 of each one of the classifiers is presented in Table 6.1.

Table 6.1 - Mean average precision of each classifier with a 0.5 IoU.

Classifier	**mAP_{50} (%)**
ResNet-50	69.47
ResNet-101	**72.93**
Darknet-19	42.32
Darknet-53	**72.26**

Furthermore, the detection results of each one of the images that comprised the dataset for each one of the deep learning models, can be seen in appendix 9.1, under the folder *Detected-TestImages*.

According to these results, the two best performing deep learning models in regard to the mAP metric were the Faster R-CNN architecture with ResNet-101 as backbone network and YOLOv3 architecture with Darknet-53 as backbone network. Both algorithms exposed an approximately constant precision level for different levels of recall (horizontal trendline), which means both are robust for the test data, nevertheless, while the ResNet-101 classifier showed a better understanding regarding the differences between *Wheels_A,B,C* and *D*, the Darknet-53 classifier also showed a fair understanding of these differences and, in addition, was the only

classifier capable of detecting the small *Screw* object class, which was necessary to be detected for the correct deployment of the human-robot collaborative (HRC) assembly system.

Moreover, the results of the tests done to determine the detection speed of each one of the deep learning frameworks are presented in Table 6.2. The average detection speed was an average of 50 measurements for a single previously resized image, where in each one of the measurements the required time for the detection of the labels, bounding boxes and the calculation of the confidence of each detection was determined.

Table 6.2 - Detection speeds of each one of the detectors (processed on GPU). Tests done with 50 detections on the same image.

Classifier	**Average detection time per image [seconds]**	**Standard deviation**
ResNet-50 [224x396]	0.346	0.043
ResNet-101 [224x396]	0.403	0.036
Darknet-19 [256x256]	**0.031**	**0.009**
Darknet-53 [608x608]	0.551	0.060

When analyzing these results, it is necessary to remember that the detection speed is not only related to the model's architecture but also deeply related to the input image size of the detector, i.e., higher input sizes require higher computational resources, which explains why the YOLOv3 detector has the slowest detection speed compared to the others, while processing 2 to 3 times higher image sizes. Nonetheless, comparing Darknet-19 to ResNet-50 and ResNet-101 – which have approximately similar image input sizes – it is clear the advantage of using YOLO-based architectures instead of Faster R-CNN ones for online applications – such as similar HRC systems – where an order of magnitude higher detection speed contributes for a better performance of the system.

Thus, for all the above-mentioned reasons and results, the selected deep learning model for the 2nd stage – the deployment of the HRC assembly system – was **YOLOv3** with **Darknet-53** as backbone network.

6.2. Second stage results – deployment of the HRC collaborative system

The final goal of this work was to create and deploy a HRC assembly system, capable of enabling a reliable assembly process of a mechanical component, where a human co-worked with a robotic arm inside the same working space, by taking advantage of the flexibility and dexterity of the human and the automation, robustness, and reach of the robotic arm.

During the deployment tests of the HRC system, it was verified that:

- The YOLOv3 presented a detection performance of 72.26% (mAP) over the test dataset and it was possible to tune the deployment algorithm's parameters mentioned in Figure 5.8, in order to increase the efficiency and reduce idle times of the assembly process.
- The vision system was capable of triggering the robot to move at the exact necessary time, helping sequentially the human throughout the entire assembly process.
- Since the robot had pre-programmed movements, it was also necessary that the human possessed a thorough understanding of each one of the movements (tasks) that the robotic arm would perform, in order to work with it safely and efficiently.
- The proof-of-concept of the HRC assembly system worked successfully.

A demonstration video of the working physical system was made and can be seen in the following link: HRC-Thesis.mp4 - Google Drive, and a sequence of images illustrating the moment where the system automatically triggered the robotic arm to grasp the wheel box for the human, while he was performing the screwing task simultaneously is shown in Figure 6.3.

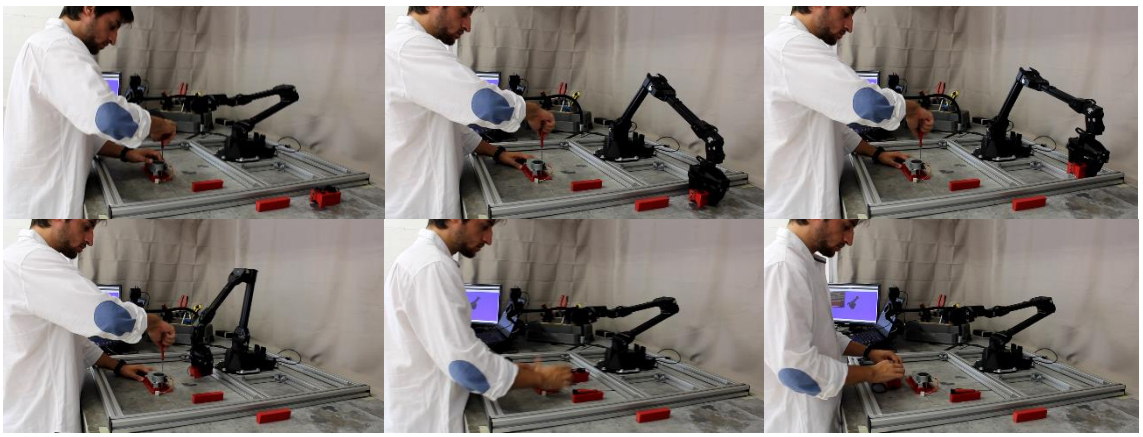


Figure 6.3 – Sequence of images showing the robotic arm being automatically triggered to grasp the wheel box for the human (movement 2) while the screwing task is in progress.

7. CONCLUSION AND FUTURE WORKS

7.1. Conclusion

During this work, a comparison between 4 different common deep learning frameworks (Faster R-CNN with ResNet-50, Faster R-CNN with ResNet-101, YOLOv2 with Darknet-19 and YOLOv3 with Darknet-53) in regard to detection performance over a custom dataset was firstly made.

According to the final goal of creating a cyber-physical industrial demonstrator of a human-robot collaborative (HRC) assembly process, it was determined that the YOLOv3 framework was more adequate for the visual task recognition part due to the following reasons:

1. The best performing detectors in terms of mAP values where YOLOv3 and ResNet-101 (72.26% and 72.93%, respectively). However, YOLOv3 was the only model capable of detecting the small *Screw* object class, which was something important for the correct functioning of the HRC system.
2. Despite having a similar mAP of the ResNet-101 framework and showing in comparison a worse capability in distinguishing the object classes *Wheel_A, Wheel_B, Wheel_C* and *Wheel_D*, the YOLOv3 was the only framework capable of detecting the small *Screw* object class, and because of the nature of the defined assembly sequence of the selected mechanical component, it was determined as more important to correctly detect the *Screw* object class rather than distinguishing perfectly all the object classes referred to the cogwheels.
3. Owing to the inherent downside of the Faster R-CNN framework of requiring higher computational resources and having a slower detection speed (for a fixed image input size), due to the much higher number of layers and parameters in comparison to the YOLO framework, the deployment of the ResNet-101 detector with the available hardware was discarded for the creation of the HRC demonstrator. Moreover, the YOLOv3 framework provided a recognition output in an acceptable time frame for the proposed HRC application.

After selecting the YOLOv3 as the object detection framework to be used in the deployment phase of the HRC assembly environment, an algorithm for this purpose was developed in MATLAB. This algorithm had the purpose of granting to the system a higher-level

of understanding of the entire assembly sequence, as it correlated the detected components in a certain time instance with the assembly tasks' progression, commanding the robotic arm as necessary and correctly.

The YOLOv3 framework presented a high mAP of 72.26% mAP over the test dataset, and with such high-performing model it was possible to tune the parameters of the deployment algorithm in order to create a system capable of managing the entire HRC assembly process, where the robotic arm was automatically commanded to move at the required time instance, according to the current state of the assembly process. The success of the implementation of this industrial demonstrator is exemplified in Figure 6.3, and can also be seen in the demonstration video of the HRC demonstrator in appendix 9.1.

Thus, this work outputs an efficient and viable HRC assembly system where the task recognition is solely performed by taking as input the visual data from a RGB camera, which is further processed and analyzed by the deep learning YOLOv3 framework and, according to the current state of the assembly process, i.e., to which components are already correctly assembled, commands the robot move on the fly.

7.2. Suggestion for future works

Notwithstanding the success and positive results achieved with this dissertation, there are several suggestions for future improvements and further work opportunities:

1. Increase human safety by implementing a collision-avoidance technology, e.g., cover the robotic arm with capacitive sensors, that could detect the presence of a human hand, arm, or others in the vicinity of the robot due to the fluctuations in the magnetic field created by the sensors.
2. Increase the training dataset and retrain all object detectors.
3. Increase the number of epochs during the retraining of ResNet-50 and ResNet-101 on a computer with a higher processing power, to evaluate how the epochs number influences the performance of these detectors.
4. Explore the object detection with the more recent versions of the You Only Look Once model, i.e., YOLOv4 [52] and YOLOv5 [53].
5. Improve the autonomous behavior of the system by commanding the robot to move to the positions interpolated by the predicted bounding box coordinates and picking up the objects accordingly, instead of picking up them from pre-defined world coordinates.
6. Vary the management system parameters in order to reduce idle times during the assembly process.

8. REFERENCES

[1] F. Yu and T. Schweisfurth, “Industry 4.0 technology implementation in SMEs – A survey in the Danish-German border region,” *Int. J. Innov. Stud.*, vol. 4, no. 3, pp. 76–84, 2020, doi: 10.1016/j.ijis.2020.05.001.

[2] L. Da Xu, E. L. Xu, and L. Li, “Industry 4.0: State of the art and future trends,” *Int. J. Prod. Res.*, vol. 56, no. 8, pp. 2941–2962, 2018, doi: 10.1080/00207543.2018.1444806.

[3] C. Pohlt, T. Schlegl, and S. Wachsmuth, “Human work activity recognition for working cells in industrial production contexts,” *Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern.*, vol. 2019-Octob, pp. 4225–4230, 2019, doi: 10.1109/SMC.2019.8913873.

[4] C. Chen, T. Wang, D. Li, and J. Hong, “Repetitive assembly action recognition based on object detection and pose estimation,” *J. Manuf. Syst.*, vol. 55, no. September 2019, pp. 325–333, 2020, doi: 10.1016/j.jmsy.2020.04.018.

[5] L. Ze-Hao, M. C. Leu, W. Tao, Z. Lai, and C. Leu, “Worker Activity Recognition in Smart Manufacturing Using IMU and sEMG Signals with Convolutional Neural Networks,” 2018.

[6] R. A. Khalil, G. S. Member, N. Saeed, S. Member, and M. Masood, “Deep Learning in the Industrial Internet of Things : Potentials , Challenges , and Emerging Applications,” vol. X, no. X, pp. 1–29, 2020.

[7] K. B. Park, S. H. Choi, M. Kim, and J. Y. Lee, “Deep learning-based mobile augmented reality for task assistance using 3D spatial mapping and snapshot-based RGB-D data,” *Comput. Ind. Eng.*, vol. 146, no. June, 2020, doi: 10.1016/j.cie.2020.106585.

[8] Z. H. Lai, W. Tao, M. C. Leu, and Z. Yin, “Smart augmented reality instructional system for mechanical assembly towards worker-centered intelligent manufacturing,” *J. Manuf. Syst.*, vol. 55, no. July 2019, pp. 69–81, 2020, doi: 10.1016/j.jmsy.2020.02.010.

[9] X. Wen, H. Chen, and Q. Hong, “Human Assembly Task Recognition in Human-Robot Collaboration based on 3D CNN,” *9th IEEE Int. Conf. Cyber Technol. Autom. Control Intell. Syst. CYBER 2019*, pp. 1230–1234, 2019, doi:

10.1109/CYBER46603.2019.9066597.

[10] Y. Cheng, L. Sun, C. Liu, and M. Tomizuka, “Towards efficient human-robot collaboration with robust plan recognition and trajectory prediction,” *arXiv*, vol. 5, no. 2, pp. 2602–2609, 2019.

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *Int. J. Comput. Vis.*, 2010, doi: 10.1007/s11263-009-0275-4.

[12] K.-B. Park, S. H. Choi, M. Kim, and J. Y. Lee, “Deep learning-based mobile augmented reality for task assistance using 3D spatial mapping and snapshot-based RGB-D data,” *Comput. Ind. Eng.*, vol. 146, p. 106585, 2020, doi: https://doi.org/10.1016/j.cie.2020.106585.

[13] Tsung-Yi Lin *et al.*, “COCO - Common Objects in Context,” *COCO Dataset*, 2018.

[14] K. He, G. Gkioxari, P. Dollar, and R. Girshick, “Mask R-CNN,” 2017, doi: 10.1109/ICCV.2017.322.

[15] G. Liu, B. He, S. Liu, and J. Huang, “Chassis assembly detection and identification based on deep learning component instance segmentation,” *Symmetry (Basel).*, vol. 11, no. 8, 2019, doi: 10.3390/sym11081001.

[16] W. E. Snyder and H. Qi, *Fundamentals of Computer Vision*. Cambridge University Press, 2017.

[17] L. Zeng, W. Xiong, and Y. Zhai, “Gun bore flaw image matching based on improved SIFT descriptor,” 2013, doi: 10.1117/12.2014430.

[18] H. W. Kim and S. I. Yoo, “Defect detection using feature point matching for non-repetitive patterned images,” *Pattern Anal. Appl.*, 2014, doi: 10.1007/s10044-012-0305-7.

[19] M. Kumar, N. K. Singh, M. Kumar, and A. Kumar Vishwakarma, “A novel approach of standard data base generation for defect detection in bare PCB,” 2015, doi: 10.1109/CCAA.2015.7148363.

[20] M. W. Tahir, N. A. Zaidi, R. Blank, P. P. Vinayaka, M. J. Vellekoop, and W. Lang, “Detection of fungus through an optical sensor system using the histogram of oriented gradients,” 2017, doi: 10.1109/ICSENS.2016.7808537.

[21] J. Huang, P. Jia, and G. Liu, “An OVR-SVM Based Machine Vision Evaluation Method for Standard Component Assembly,” 2017, doi: 10.2991/ammee-17.2017.150.

[22] A. Srisaila, S. Kranthi, K. Pranathi, and P. M. Latha, “Tag Identification for Vehicles by Using Connected Components based Segmentation and Template Matching Based Recognition,” 2018, doi: 10.1109/ICECA.2018.8474774.

[23] S. Minaee and Y. Wang, “Screen Content Image Segmentation Using Robust Regression and Sparse Decomposition,” *IEEE J. Emerg. Sel. Top. Circuits Syst.*, 2016, doi: 10.1109/JETCAS.2016.2597701.

[24] S. Minaee, “Image Segmentation Using Subspace Representation and Sparse Decomposition,” *arXiv*. 2018.

[25] S. Minaee and Y. Wang, “An ADMM Approach to Masked Signal Decomposition Using Subspace Representation,” *IEEE Trans. Image Process.*, 2019, doi: 10.1109/TIP.2019.2894966.

[26] L. Liu *et al.*, “Deep Learning for Generic Object Detection: A Survey,” *Int. J. Comput. Vis.*, vol. 128, no. 2, pp. 261–318, 2020, doi: 10.1007/s11263-019-01247-4.

[27] T. Georgiou, Y. Liu, W. Chen, and M. Lew, “A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision,” *Int. J. Multimed. Inf. Retr.*, vol. 9, no. 3, pp. 135–170, 2020, doi: 10.1007/s13735-019-00183-w.

[28] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015, doi: 10.1016/j.neunet.2014.09.003.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “2012 AlexNet,” *Adv. Neural Inf. Process. Syst.*, 2012.

[30] M. Z. Alom *et al.*, “The history began from AlexNet: A comprehensive survey on deep learning approaches,” *arXiv*. 2018.

[31] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, “ImageNet: A large-scale hierarchical image database,” 2009, doi: 10.1109/cvprw.2009.5206848.

[32] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[33] J. Le, “The 5 Computer Vision techniques that will change how you see the world,” 2018. https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-

see-the-world-1ee19334354b.

[34] R. Girshick, J. Donahue, T. Darrell, U. C. Berkeley, and J. Malik, “R-CNN,” *1311.2524v5*, 2014.

[35] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2015, doi: 10.1109/TPAMI.2015.2389824.

[36] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2016.

[37] H. Law and J. Deng, “CornerNet,” *Eur. Conf. Comput. Vision(ECCV)*, 2018.

[38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[39] W. Liu *et al.*, “SSD: Single shot multibox detector,” 2016, doi: 10.1007/978-3-319-46448-0_2.

[40] J. M. Philip, S. Durga, and D. Esther, “Deep Learning Application in IoT Health Care: A Survey,” in *Intelligence in Big Data Technologies---Beyond the Hype*, 2021, pp. 199–208.

[41] A. Lavecchia, “Deep learning in drug discovery: opportunities, challenges and future prospects,” *Drug Discov. Today*, vol. 24, no. 10, pp. 2017–2032, 2019, doi: 10.1016/j.drudis.2019.07.006.

[42] L. Santos, F. Neves Dos Santos, P. Moura Oliveira, and P. Shinde, “Deep Learning Applications in Agriculture: A Short Review,” 2020, pp. 139–151.

[43] Stanford University, “CS231n: Convolutional Neural Networks for Visual Recognition,” 2020. .

[44] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-scale Image Recognition,” *ICLR 2015*, pp. 1–14, 2015.

[45] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, “Selective Search for Object Recognition,” 2012.

[46] R. Girshick, “Fast R-CNN,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.

[47] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[48] F. Li, J. Johnson, and S. Yeung, “Lecture 11 : Detection and Segmentation,” 2017.

[49] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017, doi: 10.1109/CVPR.2017.690.

[50] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” 2018, [Online]. Available: http://arxiv.org/abs/1804.02767.

[51] J. Davis and M. Goadrich, “The relationship between precision-recall and ROC curves,” *ACM Int. Conf. Proceeding Ser.*, vol. 148, no. June 2006, pp. 233–240, 2006, doi: 10.1145/1143844.1143874.

[52] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” 2020, [Online]. Available: http://arxiv.org/abs/2004.10934.

[53] G. Jocher, “YOLOv5.” 2020, [Online]. Available: https://github.com/ultralytics/yolov5.

9. APPENDIX

9.1. Github link

https://github.com/P-Garcia-Dev/Deployment-of-a-Human-Robot-Collaborative-assembly-environment---Master-s-Thesis

9.2. Script developed for the creation of the dataset

```
%% Script for image dataset acquisition
%by Pedro Garcia, July 2021

clc
clear
imaqreset

%%
cam = videoinput('winvideo', 2, 'MJPG_1024x576'); %SONY Camera indexing in local machine
start(cam);

pause(5); %Necessary pause to start image edge webcam software by SONY

ImgCount = 0;

while (ImgCount <= 3858)
   img=getsnapshot(cam);

   %Names and path on local machine where .jpg image files were saved
   fname = ['C:\Users\Pedro G\Desktop\Tese\Development\Dataset\AssemblyImg\AssemblyImg'
num2str(ImgCount+1) '.jpg'];

   %Save images in the desired path
   imwrite(img, fname);

   %Display image
   imshow(img)
   pause(1)
   close

   ImgCount = ImgCount+1;
end
```

9.3. Script developed for the deployment of the HRC assembly process

```
%% Script for implementing and managing an online Human-robot collaborative assembly process
%
% by Pedro Garcia, August of 2021
%
% Run section by section.
% ROS Master of the robotic arm must already be running either on a Virtual Machine or on a different machine with Ubuntu
%
% The deep learning algorithm used for this system is a 11 object class fine-tuned YOLOv3 neural
% network, initially trained with 80 classes over COCO dataset.

clear
clc
imaqreset

%Establish connection to ROS Master and ROS Action Client, load the trained YOLOv3 multi-object detector (11 classes)
load('C:\Users\Pedro G\Desktop\Tese\Development\Algorithms\YOLO\YOLOv3_DarkNet53_Trained.mat','yolov3Detector','inputSize');
YOLOv3 = yolov3Detector;
imgSz = inputSize(1,1:2); %Input size of the input layer of the YOLOv3 detector

ipaddress = '192.168.1.101';
rosinit(ipaddress)

[Arm, GoalMsg] = rosactionclient('/vx300/arm_controller/follow_joint_trajectory');
GoalMsg.Trajectory.JointNames = {'elbow', ...
                    'shoulder', ...
                    'waist', ...
                    'wrist_angle',...
                    'wrist_rotate',...
                    };

%Create video object with the SONY Camera
vidobj = imaq.VideoDevice('winvideo', 2);

%% Deployment with the Robot and the computer vision-enabling deep learning model

%Initialize each joints' movevent status as 'none' because the movements haven't yet been done
statusArm1 = 'none';
statusArm2 = 'none';
statusArm3 = 'none';
statusArm4 = 'none';
```

```
statusArm5 = 'none';
statusArm6 = 'none';
statusArm7 = 'none';
statusArm8 = 'none';
statusArm9 = 'none';
statusArm10 = 'none';
statusArm11 = 'none';
statusArm12 = 'none';
statusArm13 = 'none';

%Initialize labels variable - frames captured in each cycle framesN
framesN = 10;
labels = cell(1,framesN);

while strcmp(statusArm13,'none') == 1
%Detection of the components - Assessment of detections made every 10 frames
    for i=1:framesN
        frame = step(vidobj);
        frameSz1 = imresize(frame,imgSz);

        [bboxes,~,detLabel] = detect(YOLOv3,frameSz1);
        labels{1,i} = detLabel;

        detImg = insertObjectAnnotation(frameSz1,'Rectangle',bboxes,cellstr(detLabel));
        imshow(detImg)
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Movement 1 - pick screwdriver and place it inside the working space
count1 = 0;

    for n=1:length(labels)
        %Objects to detect: Base and Stepper
        if strcmp(statusArm1,'none') == 1 && strcmp(statusArm2,'none') == 1 && strcmp(statusArm3,'none') == 1 && strcmp(statusArm4,'none') == 1 && ismember('Base',labels{1,n}) == 1 && ismember('Stepper',labels{1,n}) == 1 && ismember('Screwdriver',labels{1,n}) == 0 && ismember('Belt',labels{1,n}) == 0 && ismember('Screw',labels{1,n}) == 0 && ismember('Wheel_A',labels{1,n}) == 0 && ismember('Wheel_B',labels{1,n}) == 0 && ismember('Wheel_C',labels{1,n}) == 0 && ismember('Wheel_D',labels{1,n}) == 0
            count1 = count1 + 1;
        end
    end

    if count1 >= 7 %if 7 out of the last 10 frames detect all the Objects to detect, execute movement 1
        [statusArm1,statusArm2,statusArm3,statusArm4] = screwdriverIn(Arm,GoalMsg);
```

```
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Movement 2 - pick wheel box and place it inside the working space
count2 = 0;

    for m=1:length(labels)
        %Objects to detect: Base and Screw
        if strcmp(statusArm1,'succeeded') == 1 && strcmp(statusArm2,'succeeded') == 1 && strcmp(statusArm3,'succeeded') == 1 && strcmp(statusArm4,'succeeded') == 1 && strcmp(statusArm5,'none') == 1 && strcmp(statusArm6,'none') == 1 && strcmp(statusArm7,'none') == 1 && ismember('Base',labels{1,m}) == 1 && ismember('Screw',labels{1,m}) == 1 && ismember('Belt',labels{1,m}) == 0 && ismember('Wheel_A',labels{1,m}) == 0 && ismember('Wheel_B',labels{1,m}) == 0 && ismember('Wheel_C',labels{1,m}) == 0 && ismember('Wheel_D',labels{1,m}) == 0
            count2 = count2 + 1;
        end
    end

    if count2 >= 5 %if 5 out of the last 10 frames detect all the Object to detect and all the prior movements were already made, execute movement 2
        [statusArm5,statusArm6,statusArm7] = wheelboxIn(Arm,GoalMsg);
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Movement 3 - pick screwdriver and place it outside of the working space
count3 = 0;

    for k=1:length(labels)
        %Objects to detect: Base
        if strcmp(statusArm1,'succeeded') == 1 && strcmp(statusArm2,'succeeded') == 1 && strcmp(statusArm3,'succeeded') == 1 && strcmp(statusArm4,'succeeded') == 1 && strcmp(statusArm5,'succeeded') == 1 && strcmp(statusArm6,'succeeded') == 1 && strcmp(statusArm7,'succeeded') == 1 && strcmp(statusArm8,'none') == 1 && strcmp(statusArm9,'none') == 1 && strcmp(statusArm10,'none') == 1 && ismember('Base',labels{1,k}) == 1 && ismember('Screw',labels{1,k}) == 0
            count3 = count3 + 1;
        end
    end

    if count3 >= 3 %if 3 out of the last 10 frames detect all the Objects to detect and all the prior movements were already made, execute movement 3
        [statusArm8,statusArm9,statusArm10] = screwdriverOut(Arm,GoalMsg);
    end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Movement 4 - pick wheel box and place it outside of the working space
count4 = 0;

    for p=1:length(labels)
        %Objects to detect: Base, Belt, Wheel_A, Wheel_B
        if strcmp(statusArm1,'succeeded') == 1 && strcmp(statusArm2,'succeeded') == 1 && strcmp(statusArm3,'succeeded') == 1 && strcmp(statusArm4,'succeeded') == 1 && strcmp(statusArm5,'succeeded') == 1 && strcmp(statusArm6,'succeeded') == 1 && strcmp(statusArm7,'succeeded') == 1 && strcmp(statusArm8,'succeeded') == 1 && strcmp(statusArm9,'succeeded') == 1 && strcmp(statusArm10,'succeeded') == 1 && ismember('Base',labels{1,p}) == 1 && ismember('Belt',labels{1,p}) == 1 && ismember('Wheel_A',labels{1,p}) == 1 && ismember('Wheel_B',labels{1,p}) == 1
            count4 = count4 + 1;
        end
    end

    if count4 >= 2 %if 2 out of the last 10 frames detect all the Object to detect and all the prior movements were already made, execute movement 4
        [statusArm11,statusArm12,statusArm13] = wheelboxOut(Arm,GoalMsg);
        pause(1.0)
    end

labels = cell(1,framesN);
end

rosshutdown
```

2021 PEDRO PINTO GARCIA AUTONOMOUS TRIGGER-TASK-SEQUENCING SYSTEM FOR HUMAN-ROBOT COLABORATIVE ASSEMBLY: A DEEP LEARNING APPROACH ON VISUAL TASK RECOGNITION

NOVA SCHOOL OF SCIENCE & TECHNOLOGY