UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea in
Ingegneria Informatica e dell'Automazione

Sistema per il rilevamento e il riconoscimento della forma per macchine per la lavorazione dei tondini d'acciaio

Detection and shape recognition system applied to rebar processing machines

Relatore:

Prof.ssa Silvia Maria Zanoli

Tesi di Laurea di:

Claudio Tomaiuolo

Correlatore:

Ing. Luca Giulioni

Anno Accademico 2020/2021

Abstract

In this thesis computer vision techniques has been applied for the automatic detection of steel rebars modelled by a Bi-directional automatic stirrup bender machine. Furthermore, recognition techniques has been applied so to classify the detected objects. The project, carried out at Schnell Spa[1], is part of a larger project which integrates also an anthropomorphic arm and aims at automating the process of picking and placing rebars processed by the bender machines.

To develop this system, an automatic object detection algorithm of the Object Segmentation type was used, based on the Detectron2 neural network algorithm, developed by Facebook, which allows the various steel rebars in space to be recognised in the various layers of each frame provided by the prediction, and manipulated according to the purpose of the project.

[1] https://www.schnell.it/it/

Sommario

In questa tesi sono state applicate tecniche di visione artifiiale per il rilevamento automatico di vari oggetti in acciaio prodotti da una macchina piegatrice di staffe automatica bi-direzionale. Inoltre, sono state applicate tecniche di riconoscimento per classificare gli oggetti rilevati. Il progetto, realizzato presso la Schnell Spa[1], fa parte di un progetto più ampio che integra anche un braccio antropomorfo e mira ad automatizzare il processo di prelievo e posizionamento dei tondini modellati dalla staffatrice.
Per sviluppare questo sistema è stato utilizzato un algoritmo di riconoscimento automatico degli oggetti del tipo Object Segmentation, basato sull'algoritmo di rete neurale Detectron2, sviluppato da Facebook, che permette di riconoscere i vari tondini d'acciaio nello spazio nei vari strati di ogni fotogramma fornito dalla previsione, e di manipolarle secondo lo scopo del progetto.

[1] https://www.schnell.it/it/

Acknowledgments

I would like to thank, first of all, Professor Silvia Maria Zanoli, my thesis advisor for approving my internship and thesis proposal, on which it is based, and for her support during this period.

I would also like to thank Luca Giulioni, for his support and for the great professionalism he shared with me in researching and developing this project. I would like to thank Schnell Spa for hosting me and giving me the opportunity to realize this thesis.

Finally, I would like to thank Alessandro for putting up with me during my days at Schnell Spa, Federico and Giovannino, for making me a better person with your presence during all these years in Ancona, and Giovanni Bernardini, who with his enthusiasm and lightness helped me to reach the end of this university journey.

Contents

List of Figures

Introduction

In recent decades, the arrival of 'Industry 4.0' has brought a renewal of the technologies used in industrial automation to improve working conditions, create new business models and increase the productivity and production quality of plants[6].

The term automation means whatever is needed - a tool, process or system - to make one or more machines operate automatically without direct human activity. Thus, industrial automation uses mechanical, electronic and computer technologies to control and manage flows of materials, information and energy resources in one or more industrial production processes.

Industrial automation was born with the first industrial revolution, when engineers and industries were required to make huge investments in automation: first James Watt's steam engine, then Eugenio Barsanti and Felice Matteucci's internal combustion engine, and later electronics, all of which made it possible to achieve considerable technological progress. Since then, it has grown exponentially, in terms of fields of application and technological performance.

Originally, the aim was to make the machine do repetitive and alienating actions that were previously carried out by workers. This aim has remained, but has evolved to include expectations in terms of optimisation of resources and speed of execution, millimetric precision in operations where it is needed, the possibility of preserving the health and life of employees in operations, which in the short or long term, can be risky and/or harmful[8].

Automation applied in the factory or industry has allowed a significant improvement in the production chain and in each individual machine. Machines are now able to acquire information, understand and manage it, and act accordingly[7].

The concepts of industrial automation and Industry 4.0 go hand in hand. Achieving robotic control of increasingly autonomous industrial machinery, thanks to increasingly advanced and up-to-date technology, leads to the concept of Industry 4.0. Artificial neural networks play a key role in this technological revolution.

The implementation of a neural network was unthinkable just a few decades ago, but today some might say that science fiction has finally caught up with reality.

The artificial neural circuits are the basis of sophisticated forms of artificial intelligence, always more evolved, able to learn by exploiting mechanisms similar (at least in part) to those of human intelligence.

Artificial neural networks are now able to solve certain categories of problems, coming ever closer to the efficiency of our brains, and even finding solutions that are inaccessible to the human mind.

Much has been achieved since the birth of the concept of the artificial neuron. In many different scientific fields, from biomedicine to data mining, neural networks are in daily use.

Continuous progress is making it possible to obtain increasingly sophisticated circuits. In short, everything indicates that neural networks and machine learning will be an important part of the foundations of the hyper-technological world we are entering. Artificial intelligence is the key architecture for machines to function like the human brain and be able to make decisions autonomously, i.e. to simulate human cognitive functions such as learning and problem solving[9]. The word ROBOT, first used in 1920 by the Czech writer Karel Čapek to define the 'artificial operator' in his theatrical drama, is coming to life in these decades.

This thesis focuses on the problem of developing an automatic object detection and recognition system to increase the automation process in the field of steel rod processing machines. The work, carried out at Schnell Spa, is part of a long-term project whose final objective is to automate the process of picking up and placing rebar through the use of an anthropomorphic robotic arm.

This work is organised as follows: the first chapter "Problem Statement" illustrates the background and the needs at the basis of the project treated in this thesis; the second chapter "Computer vision" provides an introduction to the world of computer vision, defining what a digital image is and listing the various processing and recognition techniques; the first tests are instead reported in the third chapter "First Approach" with the implementation of the functions used and the related problems encountered; "Instance Segmentation" is the title of the fourth chapter, which describes the implementation and the training of the neural network introduced in the object detection system; the fifth chapter "Shape Recognition" explains, instead, the techniques used to study the shape of the steel rods and to recognize their angles; finally, the "Conclusion and Future Works" are reported.

Chapter 1

Problem Statement

This project aims to limit human intervention in the process of picking up pieces machined by a steel rebar bending machine.

The machine we are considering performs bending in three dimensions and a final cut, allowing the steel rods to be shaped for their intended purpose. The raw material is manually loaded into the machine, which pulls the bar with motors and applies the standard construction rod ribbing to the raw iron bar, after which it shapes it. At the time of cutting, the final product falls onto a horizontal plane, where it is picked up by an employee. However, it is only possible to approach the machine when the process is complete and the machined rod is in a static position on the final plane. During the cutting process, it sometimes happens that the finished product is thrown with force towards the table or floor.

Schnell's long-term project is to remove human intervention from this industrial process. The aim is to develop an automatic pick-and-place system using a robotic arm (Kawasaki RS030N2, Appendix A.5) that picks up the shaped product a moment before cutting and deposits it on a given plane.

The aim of the thesis, which focuses on computer vision (specifically on object detection and recognition), is to detect the bended rods pieces in the images and in any situation they may be in and to recognize their shape and orientation in space, in order to be able to calculate the best possible gripping points in the future.

The work in the company was organised in two parts: the first one was a preliminary study of Python programming and the use of the various image processing techniques using the functions

of the OpenCV library (Appendix A.1); the second was the programming of an artificial neural network that would make the rebar recognition system intelligent and dynamic.

Chapter 2

Computer vision

Computer vision systems can be used successfully in factory automation, quality control and other stages of a production process. These technologies make it possible to detect defects, improve product quality, reduce production costs and automate various production processes[1].

In order to achieve the goal set for this thesis, a computer vision algorithm have been developed so to extract information required that is the detection of objects made from the machining of steel rebars and their recognition. In order to better understand the functioning of the algorithm and the selection criteria of the techniques used, it is good to first give an overview of what computer vision is, the fields of operation and how a digital image is represented and processed

2.1 What is computer vision?

Computer Vision is an interdisciplinary field that studies algorithms and techniques to enable computers to reproduce functions and processes of the human visual apparatus. Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information[3]. Seeing is thus understood not only as the acquisition of a two-dimensional photograph of an area, but above all as the interpretation of the content of that area. Understanding the content means not only being able to recognise objects, people or animals within a single image or in sequence (video), but above all being able to extract useful information for their processing, at increasingly higher levels of abstraction and understanding[1]. In other words, it is the ability to reconstruct a context around the image, giving it real meaning, transforming the large matrices of numbers that compose it into descriptions of the world that make sense, through models constructed with the help of geometry, physics, statistics, to make appropriate decisions[2].

A machine vision system consists of the integration of optical, electronic and mechanical components that allow the acquisition of images both in the visible light spectrum and outside it (infrared, ultraviolet, X-rays, etc.). The result of the processing is the recognition of certain characteristics of the image for various purposes of control, classification, selection, etc.

2.2 Recognition techniques

In recent years, there has been a strong increase in attention towards Computer Vision, thanks to the advent of Artificial Intelligence and increasingly advanced Machine Learning techniques, which have made it possible to achieve performances almost comparable to human ones, and to the increasing attention given to digital content, mostly represented by digital images and videos[10].

Computer vision algorithms can investigate an image in greater or lesser depth, depending on the techniques used, the type of image and the type of task performed:

- **Image Classification**: analysis of image content and assignment of a label (e.g. dog, cat);
- **Object Detection**: identification of one or more entities within an image;
- **Image Segmentation**: subdivision of an image into sections (e.g. to highlight the pixels of a medical report showing a tumour);
- **Face Recognition**: recognition of people's faces;
- **Action Recognition**: identification of one or more entities and their relationship in time and space, in order to identify and describe specific actions (e.g. a footballer hitting the ball with his head);
- **Visual Relationship Detection**: understanding the relationship between objects in an image;
- **Emotion Recognition**: detecting the sentiment of an image;
- **Image Editing**: modification of an image (e.g. obscuring sensitive data).

Among these, companies are currently focusing mainly on Image Classification and Object Detection solutions. But the development of Computer Vision solutions involves many difficulties, such as (the most important) the creation of a sufficiently large dataset for training the algorithm and teaching the algorithm to understand the image even in the presence of transformations (e.g. non-optimal lighting conditions, deformation or partial coverage of the subject, scale variations).

2.3 Industrial applications

Machine vision systems have taken on a key role, especially in the industrial and manufacturing sectors, thanks to the possibility of being integrated directly into production lines and factory environments. The most frequent uses of this technology are:

- **Predictive maintenance**: Computer vision algorithms for monitoring industrial assets - mainly machinery - with a view to predictive maintenance (avoiding downtime by acting on possible failures or malfunctions);
- **Product monitoring**: systems for quality control and analysis of possible product defects, in order to guarantee the highest level of customer satisfaction and limit possible problems in the after-sales phase;
- **Safety in the workplace**: systems to monitor images of the plant, workers and their actions, in order to identify any risk situations and/or accidents that are harmful to people or the environment.

Thus, they range from object recognition to biometrics, from smart surveillance (smart or cloud-based surveillance cameras to analyse recorded images and identify infringements) to movement tracking and diagnostic analysis in telemedicine. The areas in which Computer Vision is used are increasing all the time[11].

2.4 Image processing

Image processing is a method of performing certain operations on an image in order to obtain an improved image or to extract some useful information from it. It is a type of signal processing in which the input is an image and the output may be an image or features associated with that image[12]. Nowadays, image processing is one of the fastest growing technologies.

Image processing basically includes the following three steps:

- Importing the image using image capture tools;
- Analysis and manipulation of the image;
- Output where the result may be an altered image or a report based on image analysis.

There are two types of methods used for image processing: analogue and digital processing. Analogue image processing can be used for hard copies such as prints and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in the manipulation of digital images using computers. The three general stages that all types of data must undergo when using digital techniques are pre-processing, enhancement and display, and information extraction[2].

2.4.1 Image as a value matrix

An image, at the digital level, can be defined as a 2D function, f(x, y), where x and y are spatial (plane) coordinates, and the magnitude of f at any pair of coordinates (x, y) is called the intensity or grey level of the image at that point. When x, y and the amplitude values of f are all finite, discrete, we call the image a digital image, which can be processed by a computer. A digital image is composed of a finite number of elements, each of which has a particular position and value, commonly called pixels, which contain intensity and colour information[2].

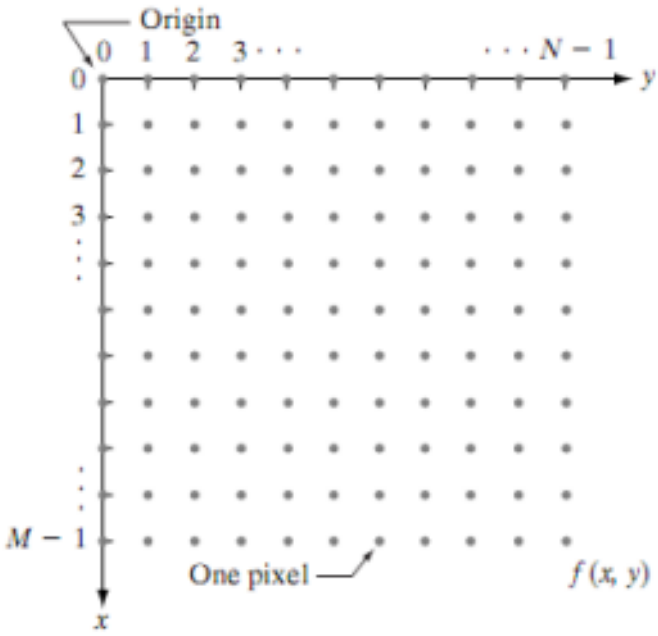

Figure 2.1. Coordinate convention used to represent digital images

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}.$$

Figure 2.2. The equation for defining a digital image

Pixels are arranged in the form of a matrix, having M rows and N columns. The coordinate values (x, y) now become discrete quantities. For clarity and notational convenience, integer values are used to indicate these discrete values. Thus, the coordinate values at the origin are (x, y) = (0, 0).

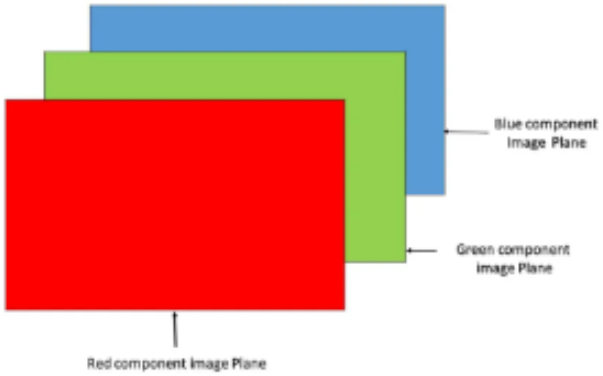

Figure 2.3. RGB image is a three layered image

The two most common types of images:

- RGB image: Contains three layers of 2D image, these layers are the red, green and blue channels.
- Greyscale image: These images contain shades of black and white and contain only one channel.

2.4.2 Levels of processing

In general, there are three levels of processing namely: low, mid and high-level processes[13].

Low-level image processing: involves tasks such as pre-processing the image to reduce noise, improving contrast, sharpening the image, etc. In this type of process, both the input and output are images.

Mid-level image processing: involves tasks such as image segmentation, image description, object recognition, etc. In this type of process, the input is generally images but the output is generally image attributes.

High-level image processing: involves making sense of a group of recognised objects. This process is normally associated with Computer Vision.

2.4.3 Fundamental steps in digital image processing[2]

1. **Image acquisition** is the first process. Note that acquisition could be as simple as receiving an image that is already in digital form. Generally, the image acquisition phase involves pre-processing, such as resizing.
2. **Image enhancement (low-level)** is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Enhancement is a very subjective area of image processing.
3. **Image restoration (low-level)** is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.
4. **Color image processing** is an area that has been gaining in importance because of the significant increase in the use of digital images over the Internet. It deals with applying colour models to digital images.
5. **Wavelets** are the foundation for representing images in various degrees of resolution.
6. **Compression**, as the name implies, deals with techniques for reducing the storage required to save an image, or the bandwidth required to transmit it.
7. **Morphological processing** deals with tools for extracting image components that are useful in the representation and description of shape.
8. **Segmentation (mid-level)** procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually. In general, the more accurate the segmentation, the more likely recognition is to succeed.
9. **Representation** almost always follows the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region (i.e., the set of pixels separating one image region from another) or all the points in the region itself. In either case, converting the data to a form suitable for computer processing is necessary. The first decision that must

be made is whether the data should be represented as a boundary or as a complete region. Boundary representation is appropriate when the focus is on external shape characteristics, such as corners and inflections. Regional representation is appropriate when the focus is on internal properties, such as texture or skeletal shape. In some applications, these representations complement each other. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. A method must also be specified for describing the data so that features of interest are highlighted.

10. **Description (high level)**, also called feature selection, deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.
11. **Recognition (high-level)** is the process that assigns a label (e.g., "vehicle") to an object based on its descriptors.

Chapter 3

First approach

To write the algorithm, it was decided to use the Python 3 programming language (Appendix A.1), due to its versatility and ease of writing, as well as the number of libraries available on the web, and excellent for processing and manipulating images, videos and footage in real time.

Unfortunately, I had never used Python before, so the first step was to study the programming language and its lexical structures and rules and to organise the workstation, using Visual Studio Code (Appendix A.1) as a code editor and Anaconda3 (Appendix A.1) to simplify the download and import of the various libraries needed and to create different working environments for testing the various algorithms.

The second step was to explore the world of Computer Vision, familiarising ourselves with OpenCV (Appendix A.2), the most famous library in the field of computer vision, and then with NumPy (Appendix A.2), to manage and manipulate the large arrays of pixels that make up the images and, therefore, the frames that make up video and filming in real time[14].

3.1 Find rebar in the image

OpenCV[4] provides many functions for image and video manipulation. The colour matrix it works with is BRG (blue-red-green), instead of the classic RGB (red-green-blue)[4]. However, to simplify the various image processing operations, it is possible to transform the working matrix to the classic RGB by applying the *cvtColor()* function, which converts an input image from one colour space to another. This function was also useful for transforming the image to black and white, to make it easier to find the steel rod within the matrix. In a black and white image, the pixel values take on values in a grey scale, with black (value 0) and white (value 255) at the two extremes, creating contrast between the elements in an image and the background.

An image with values ranging from 0 to 255 is an input requirement for applying the *Threshold()* function, which declares a threshold value against which each pixel is compared: if the pixel value is less than the declared threshold, it is set to 0, otherwise it is set to the maximum value, which in this case is 255. Its utility is to create a clear separation between the pixels that correspond to the steel rods and those that correspond to the background, or at least to parts of the image that are not of interest for the purpose of the project[15].

The next step is to detect the edges of the rebar. Fortunately, this method is so widely used in the world of computer vision, that OpenCV provides the *Canny()* (Canny Edge Detection)[16] function that implements this not very trivial operation.

The result of the implementations of the functions just mentioned follows.

Figure 3.1. Original rebar image

Figure 3.2. Image obtained after applying the cvtColor() function to make it black and white

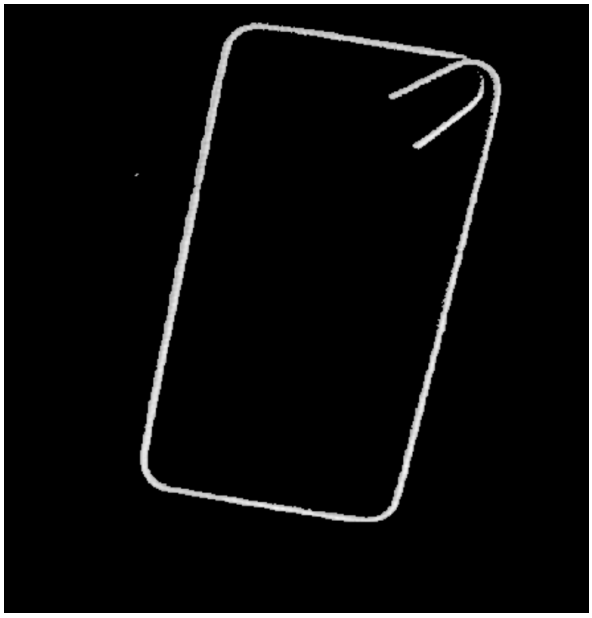

Figure 3.3. Image obtaine f

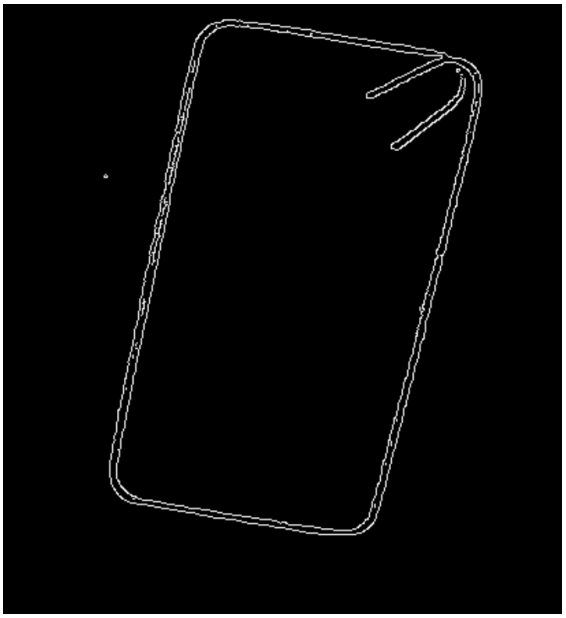

Figure 3.4. Image obtained after applying the canny() function

The final result, obtained after the implementation of the *Canny()* function, is a matrix consisting of rows and columns with inside values of 0 for black and 255 for white, representing the edges of the steel rod.

Code:

```
#Read image
img = cv.imread('tondino test.jpg')
img = cv.resize(img, dsize=(500, 500))

#Convert to RGB format
image = cv.cvtColor(img, cv.COLOR_BGR2RGB)

#Convert to Black and White
gray = cv.cvtColor(image,cv.COLOR_RGB2GRAY)

#Threshold
ret, thresh = cv.threshold(gray,170,255,cv.THRESH_TOZERO)

#Detect edges of the shape
edges = cv.Canny(thresh,50,150,apertureSize = 7)
```

3.2 Understand rebar geometry

Once we have found the steel rod in the image, with its edges, we need to study its shape and geometry, in order to subsequently understand the best gripping points for the robotic arm in the pick and place process.

The Hough Line Transform function, provided by the OpenCV[4] library, seems to be the best solution for this purpose.

The **Hough Line Transform** is a method of extracting features from objects to detect simple shapes such as circles and lines in an image. OpenCV implements two types of functions: *HoughLines()* which returns us the detected lines, through two values (ρ, θ), a rho (perpendicular distance of the line from the origin in pixels) and a theta (angle of the line in radians), and draws them along the whole image; *HoughLinesP()* (or Probabilistic Hough Line Transform) much more efficient, returns us the extremes (x1,y1,x2,y2) of each detected line and draws it on the object in the image[17]. The usefulness of using one method over the other obviously depends on the purpose and type of output you want to receive.

The result of implementing both methods follows.

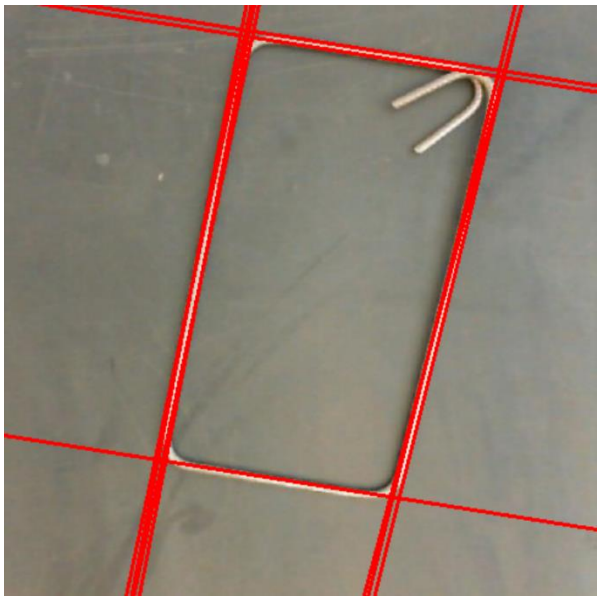

Figure 3.5. Result of applying the Hough Line Transform

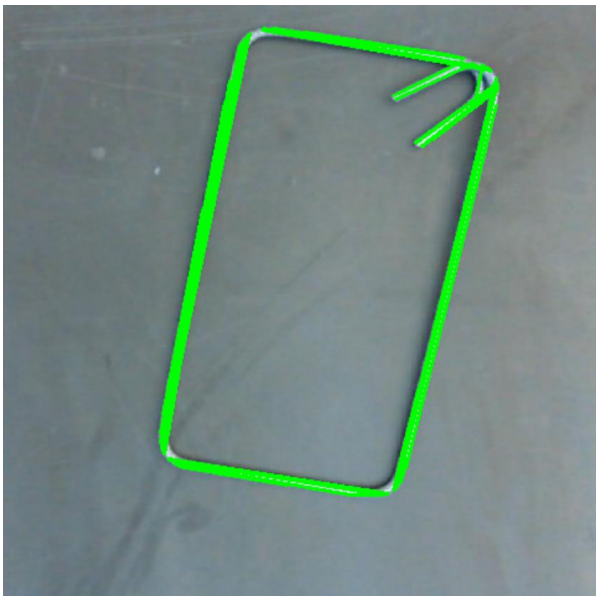

Figure 3.6. Result of applying the Probabilistic Hough Line Transform

In addition, here are some frames extracted from the result of applying the algorithm to a video.

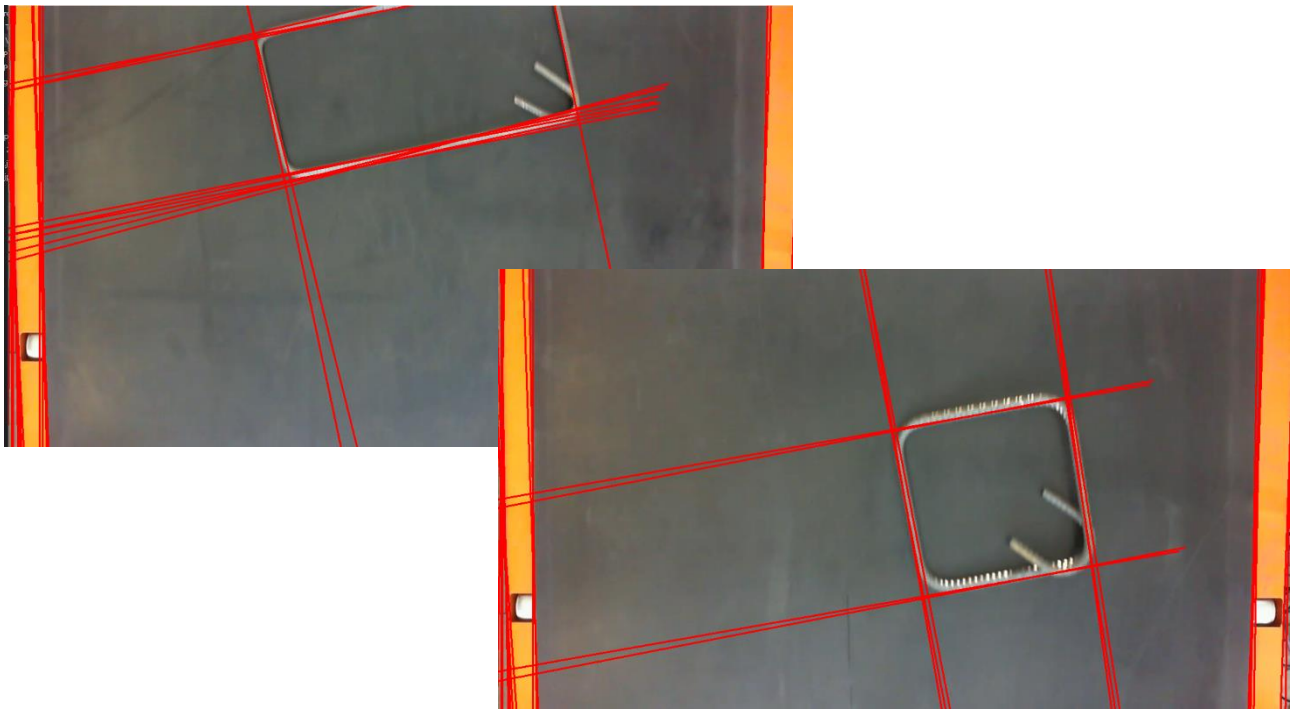

Figure 3.7. Frames extracted from a video to which the Hough Transform has been applied

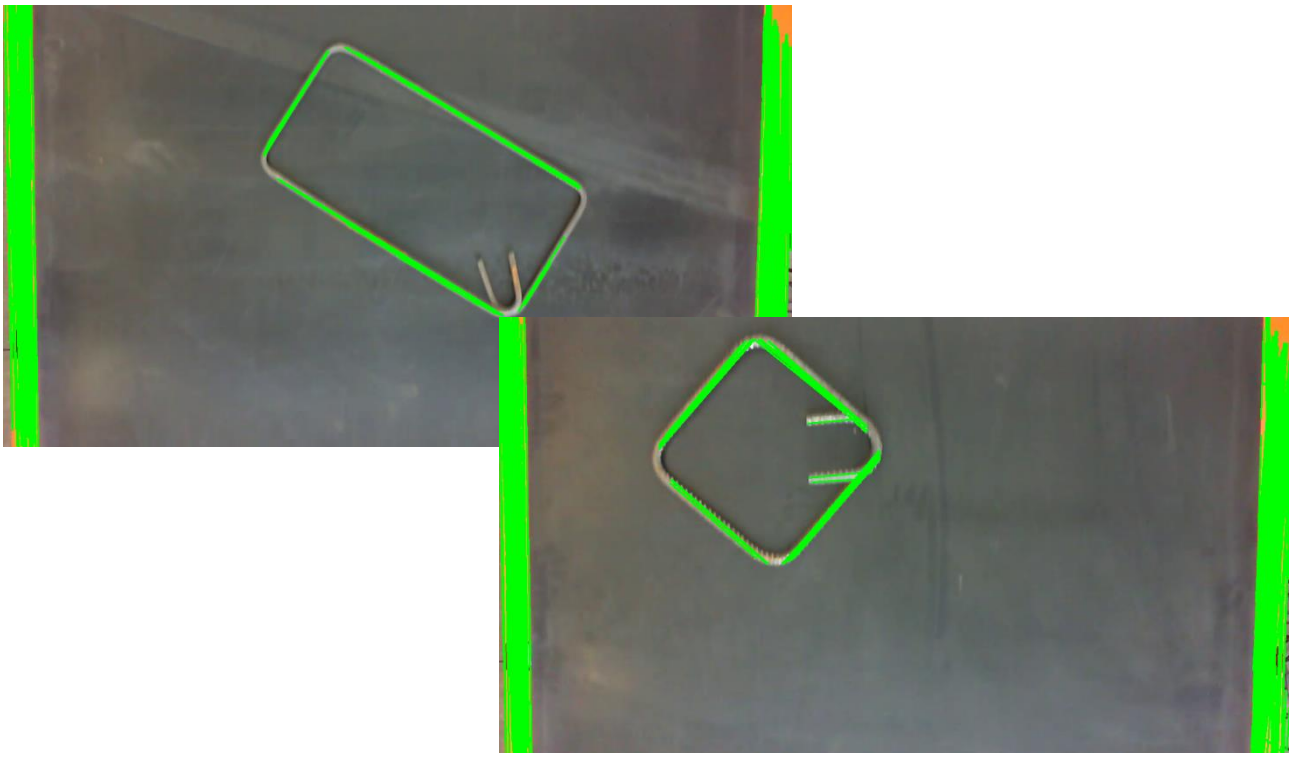

Figure 3.8. Frames extracted from a video to which the Probabilistic Hough Transform has been applied

Hough Line Transform code:

```
#Detect lines in the image using the Hough Line Transform
lines = cv.HoughLines(edges,1,np.pi/180,100)
if lines is not None:
    for line in lines:
        rho,theta = line[0]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))
        cv.line(img,(x1,y1),(x2,y2),(0,0,255),2)
else:
   print("No line detected in image")
```

Probabilistic Hough Line Transform code:

```
#Detect lines in the image using the Probabilistic Hough Line Transform
lines = cv.HoughLinesP(edges, 1, np.pi/180, 20, np.array([]), 1, 100)
for line in lines:
    for x1, y1, x2, y2 in line:
        cv.line(image, (x1, y1), (x2, y2), (0,255,0), 2)
```

3.3 First results

The first result was satisfactory. You can distinguish the rod in the image from the background and the lines have also been applied for a possible study for the best gripping points. Unfortunately, it cannot be implemented, unless the operating environment is placed in a huge box, with a static lighting system inside. In fact, the parameters set in the functions are customised according to the conditions of the objects taken as input. If the conditions of the inputs change, the parameters used in the various functions must also be readjusted. Any change in light, colour or shadows (day-night cycle, switching on and off the various lights in the building, changing the room and the lighting angles) would change the operating environment and require a new parameter setting, which would defeat the purpose of recognising the rods on the machines to automate the Pick and Place process by the robotic arm.

Another major problem is in the operating environment for which the system was designed. The working surface of the stirring machine (Schnell Prima 133, appendix A.5) is made of a very reflective metal that blends in with the rod, which makes it difficult to distinguish the rod from the background, even more so with the traditional techniques mentioned above.

The many functions provided by OpenCV are not enough. What is needed is an algorithm that can recognise the rod in any operating environment: a neural network.

Chapter 4

Rebar detection system

The decision-making capacity of a well-trained neural network makes it possible to automate industrial production processes, limiting human intervention, especially in risky or harmful cases that would endanger their health or their very lives.
Below is a brief description of what an artificial neural network is, how it is composed and how it works, to better understand all the subsequent contents and details of their implementation.

4.1 The neural network

An **Artificial Neural Network** (also abbreviated as **NN**) is a computational model consisting of artificial 'neurons', loosely inspired by the simplification of a biological neural network[19].

The mathematical models used are admittedly too simple to gain an understanding of biological neural networks, but they are used to attempt to solve artificial intelligence engineering problems such as those used in various technological fields.

Neural networks enable computers to solve problems independently and improve their ability to interpret input[18].

4.1.1 Composition

Artificial neural networks can be described as models consisting of at least two layers, one input and one output, and optional hidden layers. The more complex the problem to be solved, the more layers are necessary. Each layer of the network contains a certain number of ''specialised'' artificial neurons[18].

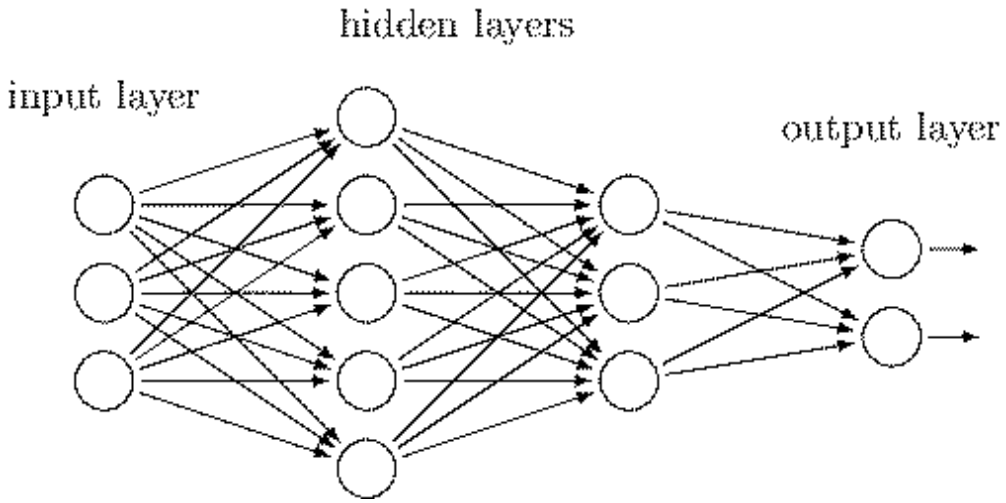

Figure 4.1. Layered model of a neural network[20]

4.1.2 How it works

The information processing in a Neural Network always follows the same procedure: the information, in the form of patterns or signals, is transferred and processed by the neurons of the input layer. Each neuron is assigned a weight, so that the neurons have a different importance.

In the next step, an activation function and a threshold value calculate and weight the output value of the neuron. Depending on the information evaluation and weighting, other connected neurons are activated to a greater or lesser extent.

By means of these processes, an algorithm is modelled which produces a result for each input. With each training, the weighting and thus the algorithm is modified so that the network provides increasingly accurate results[18].

4.1.3 Machine Learning

Machine Learning (ML) is a branch of artificial intelligence that uses statistical methods to improve the performance of an algorithm. In the field of informatics, Machine Learning is a variant of traditional programming, in which, in a machine, the ability to learn something from the input data

is set up autonomously, without explicit instructions, to elaborate patterns and make predictions on these[21].
Currently, Machine Learning is used everywhere. When we interact with banks, shop online or use social media, machine learning algorithms are used, making our experience efficient, easy and safe.

4.2 Algorithm selection

4.2.1 Object Detection

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings or cars) in digital images and videos[24]. Humans recognise a multitude of objects in images with little effort, despite the fact that the image of objects may vary somewhat in different viewpoints, different formats/scales or rotation[10]. Moreover, objects can be recognised even when they are partially out of sight. This task is still a challenge for Computer Vision in general.

There are many libraries and models available on the web: OpenVINO by Intel, DIGITS5 by Nvidia, YOLO (known for its very high FPS but lacking in functions and not very ''automatic''). OpenCV itself provides functions for Object Detection with Deep Learning[5].

Exploring the subject on the net, however, the attention has shifted to Instance Segmentation, capable of detecting and delineating every distinct object of interest that appears in an image, proposed by Detectron2, a library developed by Facebook, famous for its ease of implementation, the many functions it offers and the quantity of models already trained and material available on the net.

4.2.2 Detectron2

Detectron2[22] (Appendix A.3) is the next generation library, developed by Facebook AI Research, that provides state-of-the-art detection and segmentation algorithms. It is the successor to Detectron and maskrcnn-benchmark and has been completely rewritten on the basis of Pytorch.
It is implemented in a number of machine vision research projects and production applications by Facebook.
It includes a number of features such as Object Detection, Segmentation Detection, Panoptic Segmentation, Pose Detection, Densepose, Cascade R-CNN, rotated bounding boxes, PointRend, DeepLab, etc., and provides a large set of already trained models and baselines (Detectron2 Model Zoo).

Figure 4.2. Instance Segmentation

Figure 4.3. Panoptic Segmentation

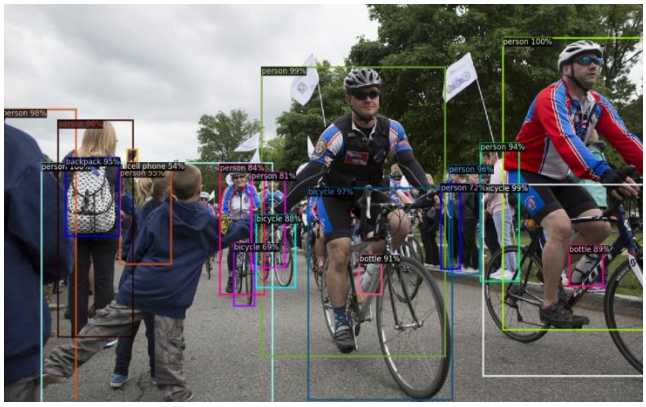

Figure 4.4. Person/Object Detection

Figure 4.5. Densepose

Figure 4.6. Pose Detection

4.2.3 Model Zoo and Baselines

As already pointed out, Detectron2 provides several models (which make up the NN layers) and already trained baselines. Following the documentation on the official website, all baselines were trained with Detectron2 itself in September-October 2019 with datasets provided by Model Zoo, on a Big Basin server with 8 NVIDIA V100 & NVLink GPUs. Obviously, there was no such high-performance server/computer at the company, so the choice of which model to use had to be weighted[22].

The choice of model was based on the performance data in this table taken from the official web page. The choice fell on the R101-FPN model, which has a relatively lower train time and inference time than the others, except for the R50-FPN, but compared with the latter has a significantly higher box AP and mask AP (parameters indicating the veracity of the model's predictions), in addition to a relatively low memory occupation (train mem).

COCO Instance Segmentation Baselines with Mask R-CNN

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	mask AP	model id
R50-C4	1x	0.584	0.110	5.2	36.8	32.2	137259246
R50-DC5	1x	0.471	0.076	6.5	38.3	34.2	137260150
R50-FPN	1x	0.261	0.043	3.4	38.6	35.2	137260431
R50-C4	3x	0.575	0.111	5.2	39.8	34.4	137849525
R50-DC5	3x	0.470	0.076	6.5	40.0	35.9	137849551
R50-FPN	3x	0.261	0.043	3.4	41.0	37.2	137849600
R101-C4	3x	0.652	0.145	6.3	42.6	36.7	138363239
R101-DC5	3x	0.545	0.092	7.6	41.9	37.3	138363294
R101-FPN	3x	0.340	0.056	4.6	42.9	38.6	138205316
X101-FPN	3x	0.690	0.103	7.2	44.3	39.5	139653917

Figure 4.7. Detectron2 baselines list

4.3 Model creation

4.3.1 Dataset

The **dataset** used was composed of images provided by the company, which portrayed the steel bars of different diameters and shapes, in different positions and places, and obviously with different illuminations, and others taken with a smartphone (LG Velvet) with different lenses, wide-angle and ultra-wide-angle, and with the Intel RealSense D455 camera (Appendix A.4). The use of different devices and lenses is intentional, due to the fact that the neural network does not have to get used to the angle of view or the peculiarities of a lens, which then characterise the output image, and thus, the input and final prediction of the neural network.

The final dataset was composed of about 250 images, then subdivided in: 70% for the training (train), 20% for the validation of the predictions (validation) and the remaining 10% for the verification of the final result (test).

All images, except those used for testing, were labelled manually. The tool used is made available on the **MakeSense.ai** website (Appendix A.3) precisely to shape parts of interest in the images and give them a label. This work is done in order to tell the neural network "what to focus on" (silhouette) when analysing the input and "what it represents" (label).

4.3.2 Training

The network training was set to a maximum of 3500 iterations, with a checkpoint function that saved the created model every 500 iterations.

Unfortunately, the training time on the computer provided by the company (Appendix A.4) was far too long. We are talking about 15 hours of training time. This is due to the lack of an Nvidia GPU with CUDA architecture in the supplied equipment. CUDA architecture is a hardware technology developed by nVidia that is used for parallel computing[23]. Detectron2 is optimised for this architecture.

A more efficient and complete alternative is the online programming environment provided by Google, Google-Colab (Appendix A.1), which provides very high computing power in reading algorithms and, most importantly, includes Nvidia GPUs with CUDA architecture, which significantly reduced the training time of the neural network to about 4 hours.

Training Code:

```
#Set trainer
cfg = get_cfg()
#cfg.MODEL.DEVICE = 'cpu' #Desactive on G-Colab
cfg.merge_from_file(model_zoo.get_config_file("COCO-
InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml"))
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml")  #initialize from model zoo
cfg.DATASETS.TRAIN = ("my_dataset_train",)
cfg.DATASETS.TEST = ("my_dataset_val",)
cfg.DATALOADER.NUM_WORKERS = 2
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.REFERENCE_WORLD_SIZE = 0 #number of GPUs. It takes no effect when set to 0.
cfg.SOLVER.BASE_LR = 0.00025  # pick a good LR
cfg.SOLVER.CHECKPOINT_PERIOD = 500 # Save model every 500 iterations
cfg.SOLVER.MAX_ITER = 3500  # you will need to train longer for a practical dataset
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 90   # (default: 512)
```

```
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1   # only has one class (Tondino)

#validate against our validation set.
class CocoTrainer(DefaultTrainer):
  @classmethod

  def build_evaluator(cls, cfg, dataset_name, output_folder=None):

    if output_folder is None:
        os.makedirs("coco_eval", exist_ok=True)
        output_folder = "coco_eval"

    return COCOEvaluator(dataset_name, cfg, False, output_folder)

#Start training
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg) #change with AugTrainer(cfg) to activate Data Augmentation
trainer.resume_or_load(resume=False)
trainer.train()
```

4.3.3 Total_loss

The Total_loss parameter represents the learning level of the network. The lower it is, the more efficient the network will be at making predictions[24]. In the graph below, it can be seen that from a certain point onwards, the value no longer falls significantly and oscillates more or less at the same level. This indicates that after a certain number of iterations (just after 3000), the model goes into overfitting, i.e. it learns by heart the content of the input templates and is no longer able to make predictions on inputs other than those on which it was trained.

The model chosen to continue the project is that of 3000 iterations, which from the comparisons made with the models of 2000, 2500 and 3500 iterations has resulted to be the most efficient.

Figure 4.8. Total_loss diagram

4.3.4 Data Augmentation

To compensate for the small size of the data set, there is a method called Data Augmentation. The Detectron2 library already includes this function with various implementation techniques.

Data Augmentation[25] means, literally, 'data augmentation'. It is a set of techniques that expand the available dataset without collecting new items, by applying controlled random changes to existing data, making modified copies of it. Labels created on images obviously follow these changes.

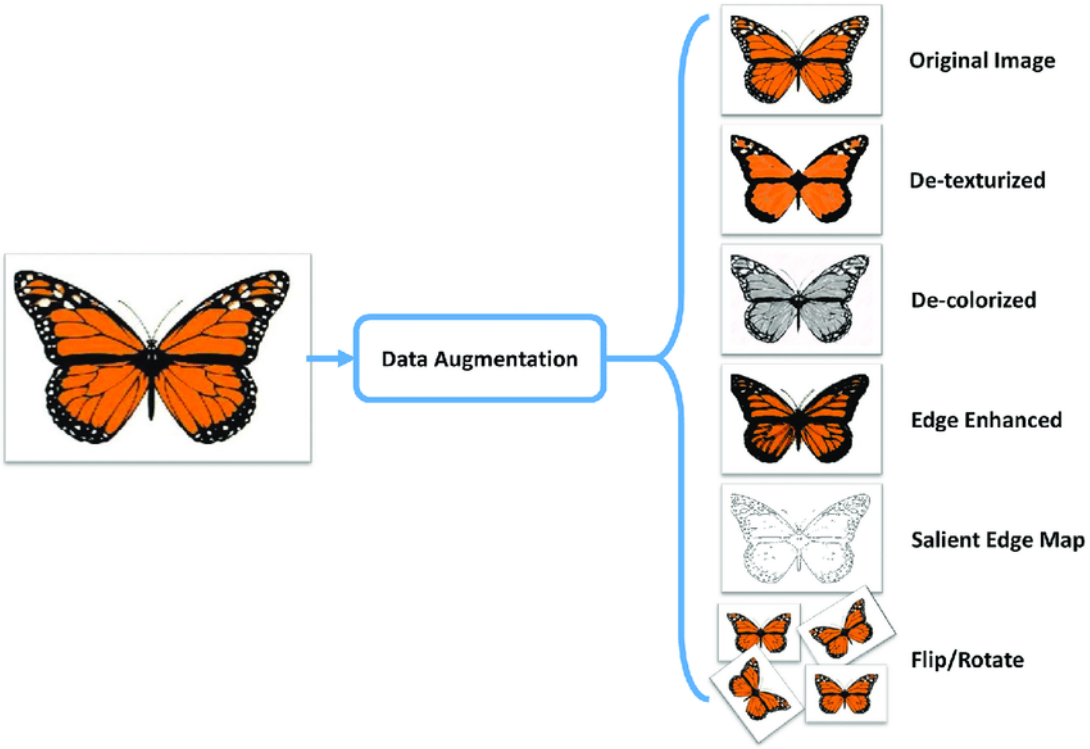

Figure 4.9. Data Augmentation techniques

In the case under consideration, there are few filters that can be used to modify and augment the dataset. The images collected were aimed in some way at training the neural network to recognise the rods mainly by their ribbings, providing it with a pattern to be able to distinguish them from other objects that might have the same shape.

Chapter 5

Results

The result of the neural network implementation was more than excellent. The model is able to recognise steel rods in almost all environments. Sometimes it makes mistakes, taking parts of the image that do not belong to the steel rod or in some positions the steel rod is recognised in spots. These inaccuracies are surely due to the dataset used for training, which is really too small. To create a good model, you normally need thousands and thousands of input data. Building a large dataset and then applying the labels manually would have taken a long time. However, there is a technique to increase the data from what you already have.

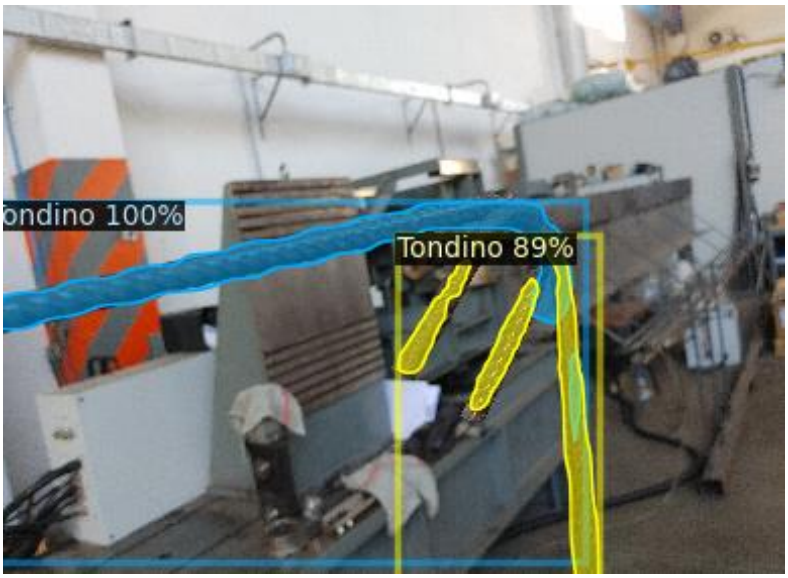

Figure 5.1. Neural network rebar prediction example 1

Figure 5.2. Neural network rebar prediction example 2

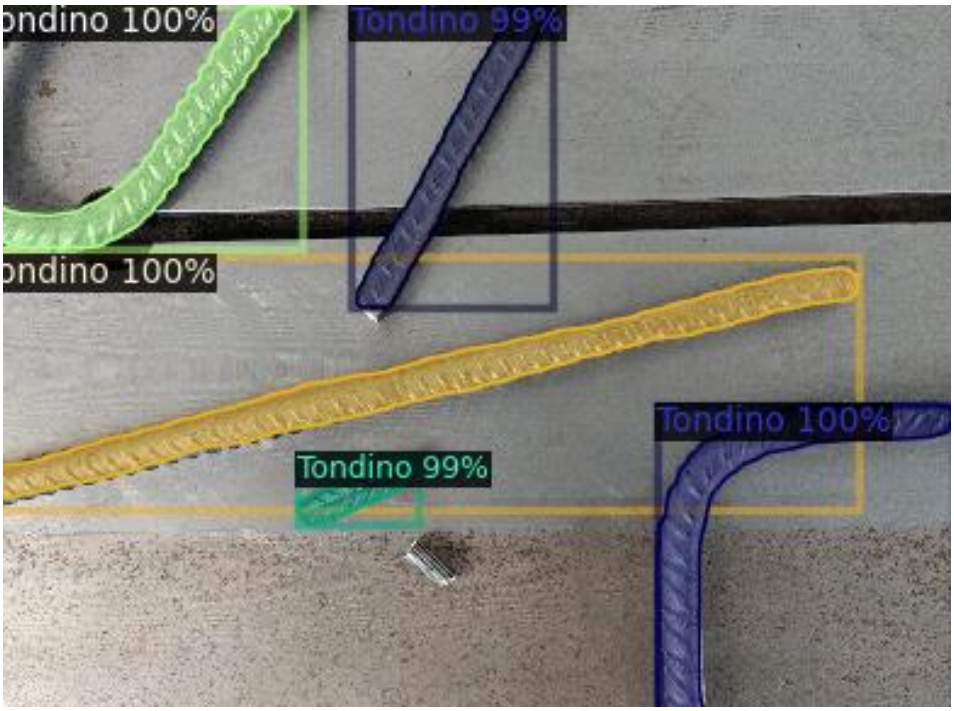

Figure 5.3. Neural network rebar prediction example 3

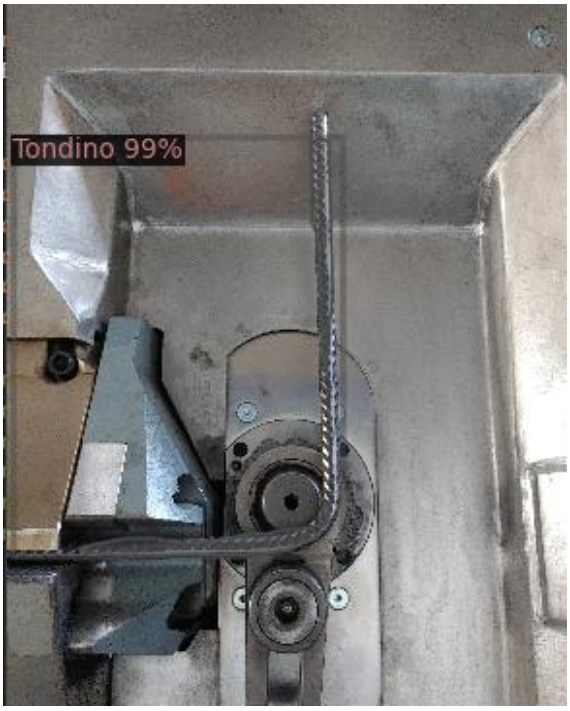

Figure 5.4. Neural network rebar prediction example 4

5.1 Data augmentation implementation

The modifications used to train a new model that included the increase of the data were: *Resize()* for resizing, *RandomBrightness()* for random changes in brightness, *RandomFlip()* for rotations in random directions, *RandomCrop()* for random cuts.

With these modifications the dataset increased to about a thousand elements.

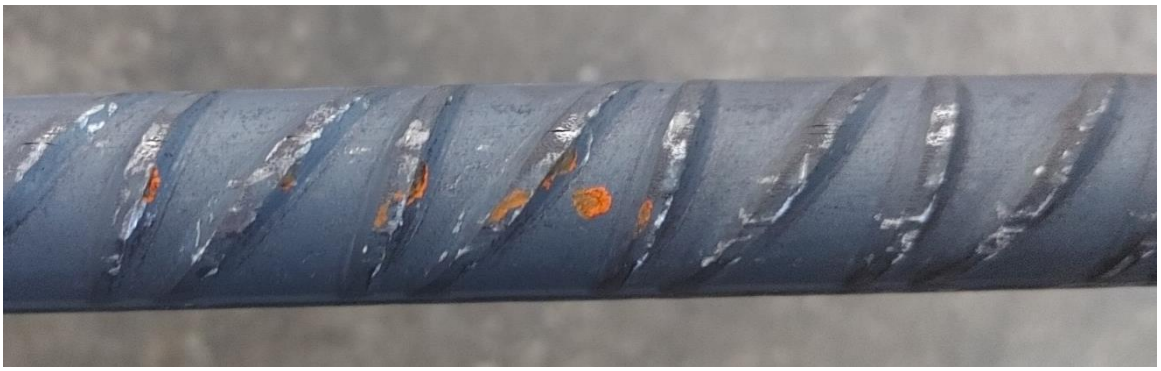

Figure 5.5. A rebar with focus on its ribbing

Unfortunately, against all expectations, the resulting model was less accurate than the first one, finding "rebar" that is not there and committing several inaccuracies in the recognition of real steel rod by also taking parts of the background or other elements.

The project therefore continued with the DA-free model with 3000 iterations, which was the most efficient of all those trained.

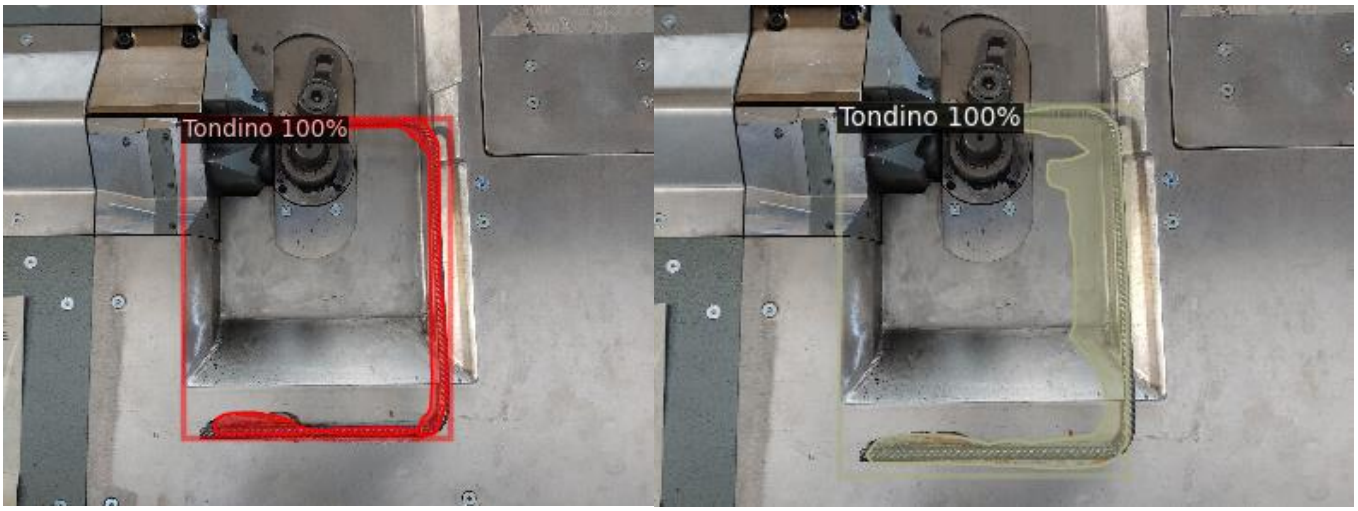

Figure 5.6. Comparison between the original model (left) and the model with the data increase applied (right) 1

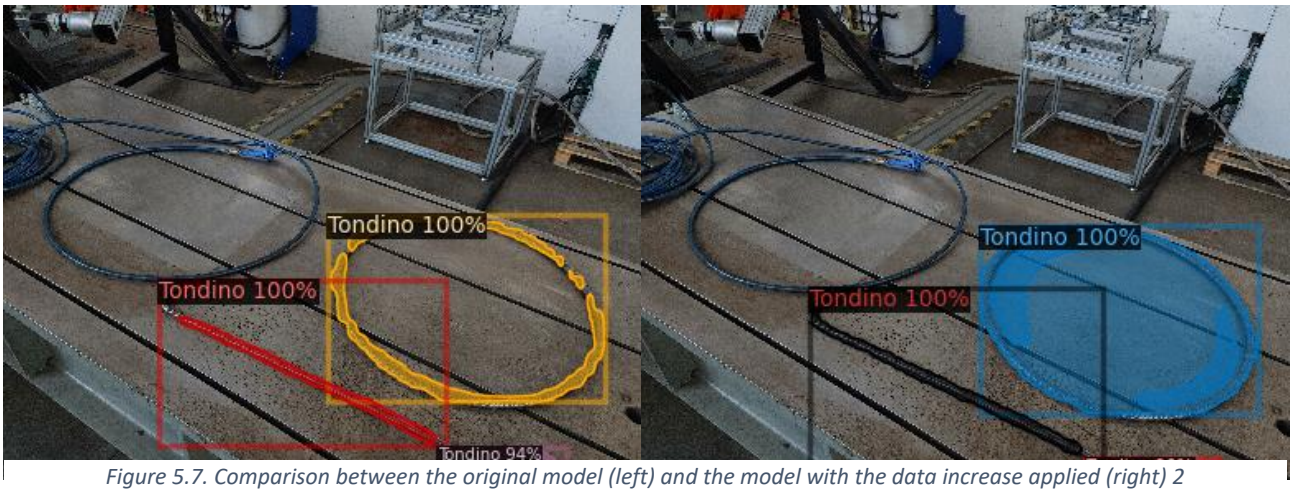

Figure 5.7. Comparison between the original model (left) and the model with the data increase applied (right) 2

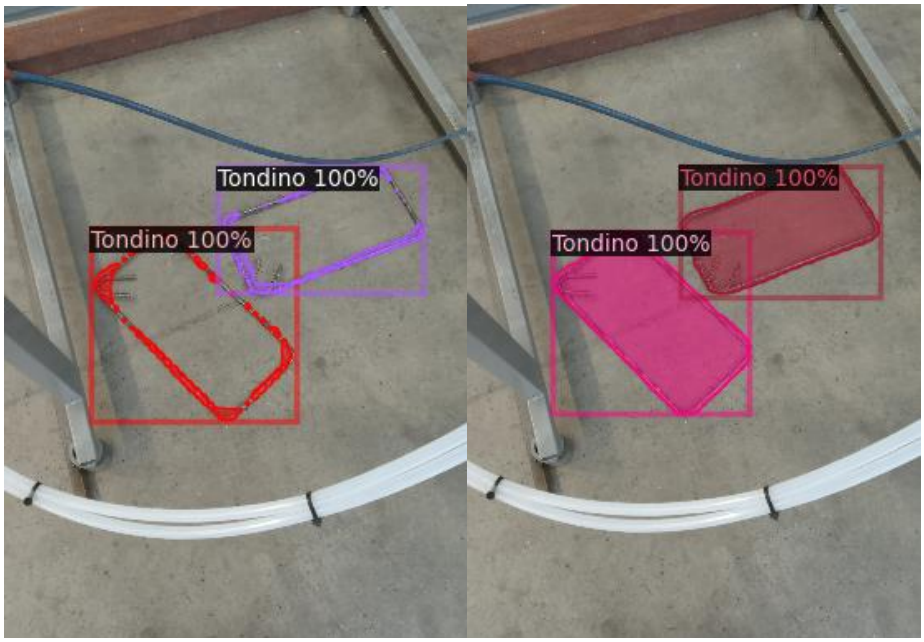

Figure 5.8. Comparison between the original model (left) and the model with the data increase applied (right) 3

Data Augmentation code:

```
#To activate Data Augmentation from Detectron2 library
def custom_mapper(dataset_dict):
    # Implement a mapper, similar to the default DatasetMapper, but with your own
customizations
    dataset_dict = copy.deepcopy(dataset_dict)  # it will be modified by code below
    image = utils.read_image(dataset_dict["file_name"], format="BGR")
    transform_list = [T.Resize((800,800)),
                      T.RandomBrightness(0.9, 1.1),
                      T.RandomFlip(prob=0.5, horizontal=False, vertical=True),
                      T.RandomFlip(prob=0.5, horizontal=True, vertical=False),
                      T.RandomCrop("absolute", (640, 640))
                      ]
    image, transforms = T.apply_transform_gens(transform_list, image)
    dataset_dict["image"] = torch.as_tensor(image.transpose(2, 0, 1).astype("float32"))
```

```
    annos = [
        utils.transform_instance_annotations(obj, transforms, image.shape[:2])
        for obj in dataset_dict.pop("annotations")
        if obj.get("iscrowd", 0) == 0
    ]
    instances = utils.annotations_to_instances(annos, image.shape[:2])
    dataset_dict["instances"] = utils.filter_empty_instances(instances)
    return dataset_dict

# Data Augmentation Trainer
class AugTrainer(DefaultTrainer):
    @classmethod

    def build_train_loader(cls, cfg):
      return build_detection_train_loader(cfg, mapper=custom_mapper)
```

5.2 Shape Recognition

After having developed an efficient system to detect steel rods in an intelligent way, it is necessary to study the shape of the objects to be picked up by the stirring machine, in order to find the best possible gripping points. The robotic arm, in fact, has a special extension consisting of two grippers placed at a right angle, capable of gripping the steel rods with one or both gripping points.

5.2.1 Reapplying the Hough Line Transform

The trained neural network model outputs images with a silhouette and a label indicating the rebar. However, the received images are composed of several layers containing the position of the various steel rods found. These layers can be isolated to provide a matrix of rows and columns with 0 for black and 1 for white, which corresponds to the presence of the steel rod.

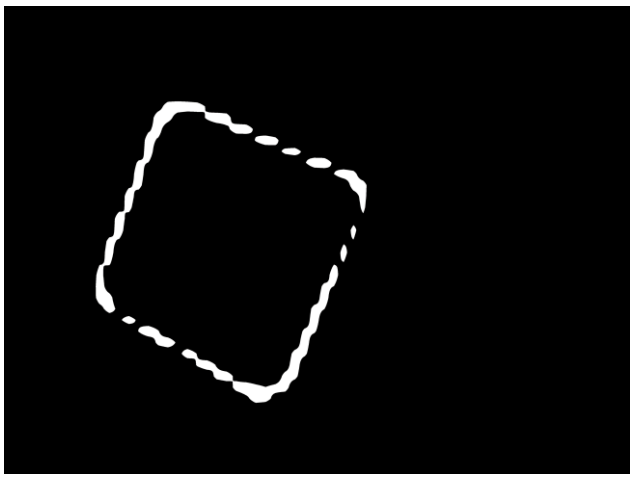

Figure 5.9. Matrix obtained with neural network prediction

On the stirrup machine, however, the rod is always and only one, so the various layers have been superimposed with an OR operation, which would also have corrected any errors made in the prediction, if it had mistakenly recognised two rods instead of one, also due to any overlapping parts due to the bending carried out by the machine.

Isolation and OR-operation code:

```
# Run Detectron2 inference on test images
cfg.MODEL.WEIGHTS = os.path.join("Models/model_3000.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.8   # set the testing threshold for this model
predictor = DefaultPredictor(cfg)

# Prediction and Show Image
filenames = ['tondino_dataset/tondino/Test/20211020_162551.jpg']
for imageName in filenames:
  im = cv2.imread(imageName)
  outputs = predictor(im)
  v = Visualizer(im[:, :, ::-1],
                 metadata=my_dataset_train_metadata,
                 scale=0.2
                 )
```

```
  out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
  cv2.imshow('segmented',out.get_image()[:, :, ::-1])


#OR-operation
newim = outputs['instances'].pred_masks.numpy() #on Colab, add .cpu() before .numpy()
for x in range(1,newim.shape[0]):
   newim[0] = np.logical_or(newim[0],newim[x])
newim = newim[0,:,:].astype(np.uint8)*255  #convert to an unsigned byte
```

To optimise the use of the Hough Line Transform, a function included in the Scikit-image library (Appendix A.2) was applied.

Skeletonize() is a function that literally skeletons binary objects into long representations of 1 pixel. It is useful for extraction functions and/or for representing the topology of an object. Skeletonisation takes place in several stages. In each of them, the object's edges are identified and a one-pixel outline is removed, provided that the removal does not result in the object being split into multiple objects, thus ensuring that one object always remains one[26].

In contrast to *Canny()*, found in OpenCV, which returns edges and therefore two lines, *skeletonize()* returns only one, the middle one.

Figure 5.10. Implementation of skeletonize()

The "body" received from the implementation of *skeletonize()* consists of a line 1 pixel thick. This "skeleton" lends itself very well to the application of the Hough Line Transform, returning bundles of lines for each very small segment belonging to the steel rod.

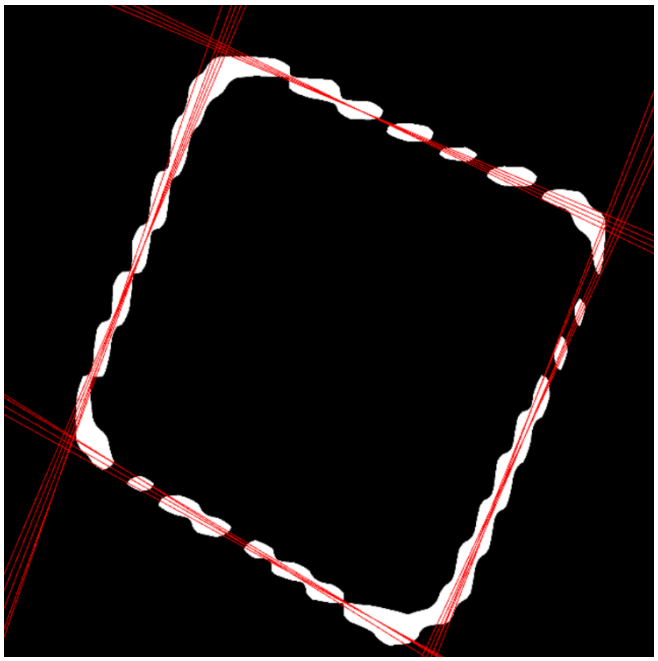

Figure 5.11. The Hough Line Transform applied to the result obtained from skeletonize()

Skeletonize and Hough Line Transform Implementation Code:

```
#Skeletonize
skel = skeletonize(newim/255)
skel = skel.astype(np.uint8)*255

#We need to draw coloured lines on the image
rgbnewim1 = cv2.cvtColor(newim,cv2.COLOR_GRAY2RGB)

# Draw Hough Lines
DistanceDrawLines = int(((newim.shape[1]+newim.shape[0])/2) * 10/100)

lines = cv2.HoughLines(skel, 15, np.pi/180, DistanceDrawLines)
if lines is not None:
  for line in lines:
    rho,theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 10000*(-b))
    y1 = int(y0 + 10000*(a))
    x2 = int(x0 - 10000*(-b))
    y2 = int(y0 - 10000*(a))
    cv2.line(rgbnewim1,(x1,y1),(x2,y2),(0,0,255),2)
```

As already explained in chapter 3, the Hough Line Transform represents the detected lines through two values (ρ, θ), a rho (perpendicular distance of the line from the origin in pixels) and a theta (angle of the line in radians). To find the central line, the average of these two values was calculated for each line bundle.

Code:

```
# Distance among lines
  DistanceLines = int(((newim.shape[1]+newim.shape[0])/2) * 15/100) #
  Radians=0.25 #Radians'difference among near lines

  # To resolve Hough Transformation Rapresentation Problems (Negative Rho)
  def resolveLine (line):
    if (line[0]<0):
      line[0]=-line[0]
      line[1]=line[1]-np.pi
    return line

  # From several lines to One central line
  newMatrix = np.zeros((len(lines),2))
  checkMatrix = np.full(len(lines), False)
  countNewLines = 0
  a = 0
  for i in range(0,len(lines)):
    if (checkMatrix[i]==False):
      lines[i][0] = resolveLine(lines[i][0])
      checkMatrix[i]=True
      centreR = 0
      centreT = 0
      centreR, centreT  = lines[i][0]
      k=1
      for j in range(i+1,len(lines)):
        if (checkMatrix[j]==False):
          lines[j][0] = resolveLine(lines[j][0])
          if (((lines[i][0][0]-lines[j][0][0]<DistanceLines) & (lines[i][0][0]-lines[j][0][0]>-DistanceLines))
& ((lines[i][0][1]-lines[j][0][1]<Radians) & (lines[i][0][1]-lines[j][0][1]>-Radians))):
            checkMatrix[j]=True
            centreR = centreR + lines[j][0][0]
            centreT = centreT + lines[j][0][1]
            k=k+1
      newMatrix[countNewLines][0] = (centreR/k)
      newMatrix[countNewLines][1] = (centreT/k)

      if (newMatrix[countNewLines][1]<0):
        newMatrix[countNewLines][0]=-newMatrix[countNewLines][0]
        newMatrix[countNewLines][1]=newMatrix[countNewLines][1]+np.pi

      a = np.cos(newMatrix[countNewLines][1])
      b = np.sin(newMatrix[countNewLines][1])
      x0 = a*newMatrix[countNewLines][0]
      y0 = b*newMatrix[countNewLines][0]
      x1 = int(x0 + 10000*(-b))
      y1 = int(y0 + 10000*(a))
```

```
    x2 = int(x0 - 10000*(-b))
    y2 = int(y0 - 10000*(a))
    cv2.line(rgbnewim2,(x1,y1),(x2,y2),(0,0,255),3)
    # cv2.imshow("oneline",rgbnewim2)
    # if cv2.waitKey(0) & 0xff == 27:
    #   cv2.destroyAllWindows()
    countNewLines+=1
# detelete zeros from newMatrix array
for p in range(len(lines),0,-1):
  if (newMatrix[p-1][0]==0):
    newMatrix = np.delete(newMatrix, p-1, axis=0)
```

5.2.2 Intersections between lines

The two grippers of the robotic arm for extracting the rebar from the machine are placed at a 90° angle and gripping can be done with one or both of the grippers. However, the rebar should be gripped with both grippers. Steel bars with a large diameter and bent into particular shapes, taken at one point, may have such a weight that they self-fold downwards by the force of gravity, changing the shape just given.

By then finding the intersections between the measured lines and calculating the angles they form, it is possible to work out whether there are angles of plus or minus 90° suitable for the particular grip of the robotic arm.

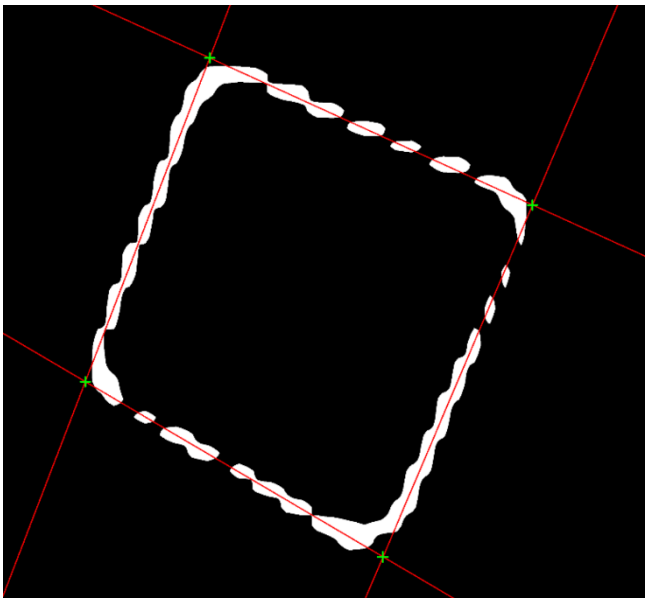

Figure 5.12. Intersections found between central lines

Code:

```
#Intersection between lines
 if newMatrix is not None:
  intersections = np.zeros((len(lines)*10,2))
  e=0
  for x in range(0,len(newMatrix)):
   for y in range(x+1,len(newMatrix)):
    rho1, theta1 = newMatrix[x]
    rho2, theta2 = newMatrix[y]
    A = np.array([[np.cos(theta1), np.sin(theta1)],
                  [np.cos(theta2), np.sin(theta2)]])
    B = np.array([[rho1], [rho2]])
    x0, y0 = np.linalg.solve(A, B)
    if ((x0>=0 and x0<=rgbnewim2.shape[0]) and (y0>=0 and y0<=rgbnewim2.shape[1])):
     intersections[e] = int(np.round(x0)), int(np.round(y0))
     e+=1
   # detelete zeros from array
  for p in range(len(intersections),0,-1):
   if (intersections[p-1][0]==0):
    intersections = np.delete(intersections, p-1, axis=0)
```

```
#Draw intersection
if intersections is not None:
  pt = np.zeros(2)
  for w in range(0,len(intersections)):
    length = 20
    x = int(intersections[w][0]) #cv2.line want integer variables
    y = int(intersections[w][1])
    cv2.line(rgbnewim2,(x, y+length),(x, y-length),(0, 255, 0),5) # vertical line
    cv2.line(rgbnewim2,(x-length, y),(x+length, y),(0, 255, 0),5)
```

Conclusion

The goal of this thesis was to develop a rebar recognition system, which would be able to operate under any conditions and in any environment. Various computer vision techniques made available by OpenCV were studied; however they did not meet the requirements of the project. The implementation of the Detectron2 neural network for instance segmentation had a significantly more efficient result, allowing the computer to recognise the rods in a "smart" way under any conditions.

The margins of error are small and are mainly due to particular shapes modelled by the stirring machine, where some segments of the rod are superimposed, which makes the the algorithm recognise two rods that are actually two parts of the same rod piece. An other cause of error was due to the light reflected by the working base of the bending machine, which being made of metal, makes it difficult to distinguish the rod from the background.

The object detection system was then completed with a study of the shape of the object of interest, finding the coordinates of the straight lines representing the center steel rod and their intersections. The aim was to find the 90° angles of intersection in order to identify the best gripping points for the robotic arm's unique type of grasp. The algorithm is obviously dependent on the efficiency of the predictions of the neural network.

The image dataset used for training the neural network was far too small. It would be necessary to set up a dataset of at least a thousand images, all manually labelled. With this in mind, it is possible to re-train the neural network to create a new model, which will certainly perform better.

A computer with a CUDA architecture GPU would facilitate a redesign of the neural network, reducing the time needed to process algorithms, test and train future models (Google-Colab imposes limited daily usage times).

As far as the long-term project is concerned, the system dealt with in this thesis is currently not implementable in an automatic pick and place process, due to the lack of information on the third dimension. Furthermore, it will obviously be necessary to program the movement of the robotic arm based on the detection of the rods in space and the best pick-up point. A similar detection

system can also be implemented in different machines, aimed at different rod shaping techniques (such as welding).

Appendix A

Setup

Here the list of software and hardware tools used:

A.1 Programming environment

Figure A.1. Visual Studio Code Logo

Visual Studio Code 1.61.2[2]

Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

Figure A.2. Python Logo

Python 3.8.11[3]

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. It supports multiple programming paradigms beyond object-oriented programming, such as procedural and functional programming. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. For several years now, it has become one of the main programming languages for writing algorithms based on Neural Networks and thus on Artificial Intelligence.

Figure A.3. Anaconda3 Logo

Anaconda3[4]

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

Figure A.4. Google-Colab Logo

Google-Colab[5]

Google-Colab is a Jupyter notebook environment that runs in the browser using Google Cloud. Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more.

[2] https://code.visualstudio.com/

[3] https://www.python.org/

[4] https://docs.anaconda.com/

[5] https://colab.research.google.com/

A.2 Python's libraries

Figure A.5. OpenCV Logo

OpenCV 4.5.3.56[6]

The Open Source Computer Vision Library is a cross-platform software library for real-time computer vision. It consists of over 500 useful functions in the field of Image Processing and Computer Vision. The library's strengths are its completeness and freedom of use (open source).

Figure A.6. NumPy Logo

NumPy 1.19.5[7]

NumPy is an open source library for the Python programming language, which adds support for large matrices and multidimensional arrays, along with a large collection of high-level mathematical functions, in order to operate efficiently on these data structures.

Figure A.7. Matplotlib Logo

Matplotlib 3.4.3[8]

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Figure A.8. Scikit-image Logo

Scikit-image 0.18.3[9]

Scikit-image is a collection of algorithms for image processing.

A.3 Neural Network Tools

Figure A.9. Detectron2 Logo

Detectron2 0.5[10]

Detectron2 is Facebook AI Research's next generation library that provides state-of-the-art detection and segmentation algorithms. It is the successor of Detectron and maskrcnn-benchmark. It supports several computer vision research projects and production applications in Facebook.

Figure A.10. Mask R-CNN Logo

Mask R-CNN R101 FPN 3x[11]

Mask R-CNN is a Convolutional Neural Network (CNN) and state-of-the-art in terms of image segmentation. It detects objects in an image and generates a high-quality segmentation mask for each instance.

Figure A.11. MakeSense.ai Logo

MakeSense.ai[12]

Makesense.ai is a free-to-use online tool for labeling photos. It is perfect for small computer vision deep learning projects, making the process of preparing a dataset much easier and faster.

[6] https://opencv.org/

[7] https://numpy.org/

[8] https://matplotlib.org/

[9] https://scikit-image.org/

[10] https://detectron2.readthedocs.io/en/latest/tutorials/install.html

[11] https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

[12] https://www.makesense.ai/

A.4 Hardware

Figure A.12. Intel RealSense D455

Intel RealSense D455[13]

The Intel® RealSense™ Depth Camera D455 is an USB-powered camera that includes wider field of view depth sensors and a RGB sensor.

PC[14]

Dell OptiPlex 3070 MFF (JX26T):
CPU: Intel(R) Core(TM) i5-9500T CPU @2.20Hz
SSD: 256 GB
RAM: 8 GB 2666MHz DDR4
Graphic: Intel(R) UHD Graphics 630
SO: Windows 10 Pro 64 bit

A.5 Machinery for which the neural network was designed

Figure A.13. Kawasaki RS030N2

Kawasaki RS030N2[15]

A 6 DoF manipulator RS030N robot from Kawasaki.

Figure A.14. Schnell Prima 133

Schnell Prima 133[16]

Bi-directional automatic stirrup bender from coil to produce stirrups and cut-to-size bars.

[13] https://www.intelrealsense.com/depth-camera-d455/
[14] https://www.dell.com/it-it/work/shop/desktop-e-workstation/optiplex-3070-micro/spd/optiplex-3070-micro
[15] https://robotics.kawasaki.com/en1/products/robots/small-medium-payloads/RS030N
[16] https://www.schnellgroup.com/en/Products/Automatic-stirrup-bender-from-coil-Prima-13/

Bibliography

[1] S. Khan, H. Rahmani, S.A.A. Shah, and M. Bennamoun, *A guide to convolutional neural networks for computer vision*, Synthesis Lectures on Computer Vision, 1(8), 1–207, 2018.

[2] R.C. Gonzales and R.E. Woods, *Digital image processing*, Prentice Hall, 2002.

[3] D.H. Ballard and C.M. Brown, *Computer Vision Prentice Hall*, 1982.

[4] G. Bradski, *The OpenCV Library*, Dr. Dobb's Journal of Software Tools, 2000.

[5] K. He, and X. Zhang, S.Ren, and J. Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE conference on computer vision and pattern recognition, 770-778, 2016.

[6] K. Zhou, Taigang Liu and Lifeng Zhou, *Industry 4.0: Towards future industrial opportunities and challenges*, 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 2147-2152, 2015.

[7] S. -Y. Chien, M. Lewis, K. Sycara, A. Kumru and J. -S. Liu, *Influence of Culture, Transparency, Trust, and Degree of Automation on Automation Use*, in IEEE Transactions on Human-Machine Systems, vol. 50, no. 3, pp. 205-214, June 2020.

[8] F. Butera, *L'automazione e il futuro del lavoro operaio*, Studi organizzativi, 2, 82, 2014.

[9] B. Bajic, A. Rikalovic, N. Suzic and V. Piuri, *Industry 4.0 Implementation Challenges and Opportunities: A Managerial Perspective*, in IEEE Systems Journal, vol. 15, no. 1, pp. 546-559, March 2021.

[10] B. Zhang, *Computer vision vs. human vision*, 9th IEEE International Conference on Cognitive Informatics (ICCI'10), pp. 3-3, 2010.

[11] A. Fabijanska, M. Kuzanski, D. Sankowski and L. Jackowska-Strumillo, *Application of image processing and analysis in selected industrial computer vision systems*, 2008 International Conference on Perspective Technologies and Methods in MEMS Design, pp. 27-31, 2008.

[12] P. Chakravorty, *What Is a Signal? [Lecture Notes]*, in IEEE Signal Processing Magazine, vol. 35, no. 5, pp. 175-177, Sept. 2018.

[13] B. Gidas, *A renormalization group approach to image processing problems*, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 2, pp. 164-180, Feb. 1989.

[14] Harris, C.R., Millman, K.J., van der Walt, S.J. *et al., Array programming with NumPy*, Nature 585**,** 357–362, 2020.

[15] Adelson, E. H., Anderson, C. H., Bergen, J. R., Burt, P. J. & Ogden, J. M., *Pyramid methods in image processing*, RCA Engineer, 29, 33—41, 1984.

[16] W. Rong, Z. Li, W. Zhang and L. Sun, *An improved Canny edge detection algorithm*, 2014 IEEE International Conference on Mechatronics and Automation, pp. 577-582, 2014.

[17] Duda, R. O. and P. E. Hart, *Use of the Hough Transformation to Detect Lines and Curves in Pictures*, Comm. ACM, Vol. 15, pp. 11–15, January, 1972.

[18] Haykin, *Neural Networks and Learning Machines*, 3rd edition, 2008.

[19] Ernesto Burattini e Roberto Cordeschi, *Intelligenza Artificiale*, Roma, Carocci, ISBN 88-430-2011-0, 2001.

[20] Paolo medici*, Reti Neurali,* 2017. *http://www.ce.unipr.it/~medici/geometry/node107.html*

[21] Bishop, Christopher M., *Pattern Recognition and Machine Learning*, Print, 2006.

[22] Facebook Research, *Detectron2*, Github Repository, https://github.com/facebookresearch/detectron2.

[23] *GPU computing - BOINC*, on boinc.berkeley.edu (last access 7th febbraio 2016).

[24] R. L. Galvez, A. A. Bandala, E. P. Dadios, R. R. P. Vicerra and J. M. Z. Maningo, *Object Detection Using Convolutional Neural Networks*, TENCON 2018 - 2018 IEEE Region 10 Conference, pp. 2023-2027, 2018.

[25] Shorten, C., Khoshgoftaar, T.M., *A survey on Image Data Augmentation for Deep Learning*, J Big Data 6, 60, 2019.

[26] *Skeletonize*, on Scikit-Image https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html.