Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Implementation of a Tic-Tac-Toe game for a robotic arm using Artificial Intelligence

Student

Jayro Zaid Paiva Mimbela

Tutor

Manuel Alò

Advisor

Mikael Andreas Bianchi

Costumer

-

Degree course

Master of Science in Engineering with specialty in Electrical Engineering

Module

Master Thesis

Year

2022

Date

August 31, 2022

Abstract

The purpose of this thesis is to highlight the potential of Artificial Intelligence applied to manufacturing processes. To illustrate this potential, the thesis aims to develop a robotic agent that is able to play the Tic-Tac-Toe game. For that, based on a previous analysis, we equipped a UR3e robot with three systems: grid recognition system, opponent's marker detection system, and camera-based positioning system. Besides the development of these three systems, the thesis presents a detailed analysis of the kinematics of manipulator robots, focusing on the UR family robots. The analysis includes the study of forward and inverse kinematics, singularities, and novel approaches to the inverse kinematics which consider singularities such as the Damped least-squares method.

Contents

Chapter 1

Thesis project description

1.1 Description

We situate the master thesis project within the framework of the fourth industrial revolution, which points to the smart automation using technologies as automation, machine learning, computer vision, robotics, etc. The aim of the project is to create a demonstrator of technologies listed above, developing a cobotic cell able to play a Tic Tac Toe game against a human opponent. The system that has to be developed comprise a robotic arm used to draw the moves in a whiteboard placed in the table, a camera to visualize the workspace and an algorithm based on AI to recognize the spatial coordinates of the image and to elaborate the moves.
We subdivide the project into two parts: in the first part, the trajectories of the robotic arm have to be elaborated, avoiding the singularities and using the kinematics libraries developed at SUPSI. Besides the collisions need to be prevented, verifying the trajectories in a digital twin before the actuation.
In the second part of the project, the image elaboration and game strategy need to be developed using AI algorithms. The cobot first has to calculate the position and orientation of the image acquired by the camera, using a laser pointer as a reference for position determination. As the second step, the grid (playing field) and moves drawn by a person have to be recognized and located in the space to compute the movements to be performed by the robot. In this project, the potentialities and the potential disadvantages of the artificial intelligence have to be highlighted.

1.1.1 Objectives

- Study the PyBullet library to compute 3D object interaction in the digital twin.
- Compute distances, collisions, and singularities between cobot joints and object surfaces.

- Elaborate a trajectory computation in order to avoid the singularities and collisions. Use, e.g. the damped least-squares method.
- Prepare the cobotic cell with the camera and implement the interface in Python.
- Using a laser pointer mounted on the robot and perpendicular to the table, implement an algorithm based on AI to test different positions and identify the projection and position of the image captured by the camera.
- Using AI, implement an algorithm to identify the Tic-Tac-Toe grid (size, orientation and 9 fields) drawn by a user.
- Implement an algorithm to determine player's moves and compute the moves to win.
- Compute the trajectory to draw the moves.

1.1.2 Technologies

- Deep Learning.
- Computer vision.
- Cobots.
- Programming language: Python.

Chapter 2

Introduction

Including the robots in industrial settings changed virtually every industry. Thereby, the robots nowadays are widely used in manufacturing, transport, search and rescue, healthcare, surgery, laboratory research, etc. [1].
Focusing in the manufacturing, from the beginning the robots play a central role, increasing the productivity of the factories by means the automation into production lines. However, although different industries employ industrial robots in their manufacturing operations, the robots mostly only deal with repetitive tasks. Therefore, endowing machines with reasoning and learning ability that allow them to realize more flexible and human-centred tasks would represent a game-changer for quite a few industry sectors [2][3].
Among the industrial robots, robot manipulators are the most common devices, and generally its motion is based on a series of pre-programming motion sequences; however, this approach for complex tasks is inefficiency, tedious and impracticable, especially if the tasks update and change constantly. Therefore, we need a robot with the abilities of perception, decision-making and learning in a complex and dynamic environment. In this context, the Artificial Intelligence (AI) opens up this possibility, releasing the robots from old-fashioned hard-programmed scenarios [2][3].
The AI, precisely, is a computational technique that makes machines intelligent. The machines can learn and make their own decision, without expecting that the human tell him what to do and how to react. In this sense, we consider a machine to be intelligent if it can analyse information and extract knowledge beyond the obvious [4].
In 1950, Alan Turing designed the known Turing Test to give an operational definition of intelligence: a computer passes the test if, after having a conversation (via online typed messages) with an interrogator for five minutes, the interrogator did not known whether the responses come from a human or form a computer. According to this test, we note that the computer would need to have the capabilities below [5]:

- **Natural language processing** so that it can communicate successfully in English.
- **Knowledge representation** to store what it knows or hears.

- **Automated reasoning** to use the stored information and to answer questions and to draw new conclusions.
- **Machine learning** to adapt to new circumstances and to detect and extrapolate patterns.

Although in the Turing Test does not consider a physical interaction between the computer and the interrogator since physical simulation of a human is unnecessary for intelligence, exits the so-called total Turing Test, which includes a video signal to allow the interrogator to test the subject's perceptual abilities, and the opportunity for the interrogator to pass physical objects "through the hatch". Thus, to pass the total Turing Test, the computer also will need the capabilities below [5]:

- **Computer vision** to perceive objects.
- **Robotics** to manipulate and move objects.

The above computer capabilities are the six disciplines that compose most of AI [5][6]. This merge between the robotics and AI birth to the smart robotic manufacturing, where robots now can carry out more complex task with a degree of intelligence. In the industrial manufacture, this new approach is being used in, e.g. robotic grasping, assembly and disassembly, peg-in-hole task, bending steel wire, grinding and polishing, weld seam welding, human-robot collaborative work, etc. [2][3].
At the research level, we can find an opportunity to show how to solve a task with intelligent requirements through the development of systems able to play games. In this sense, simply games as Sudoku or Tic-Tac-Toe are suitable for illustrating the design and implementation of an intelligent system. But at the end, the intention is to highlight the potential of intelligent machines approach to be used, e.g. in industrial applications [7][8][9][10][11].
Based on this intention, the present thesis project proposes the development of agent robotic able to play the Tic-Tac-Toe game. It is important to highlight that this only implies endowing the robot with the capabilities to play the game, without the implication of knowing how to play the game, thus the project does not focus on a decision-making system. We endow a robot manipulator with a computer vision system that allows to the robot to know the current status of the game, as well as recognize the opponent's marker.
By other hand, we also focus on the theoretical development of manipulator robot kinematics. We start with the concepts of direct and inverse kinematics, focusing on the analytical solutions of inverse kinematics (advantages and disadvantages). Based on this previous analysis, we carry out a study of singularities and new methods of inverse kinematics that take them into account. With this, we hope to have the theory to propose trajectory design strategies that consider singularity problems.
As with previous works, the purpose of the thesis is to highlight the potential of this novel approach for industry, e.g. for application in manufacturing processes.

Chapter 3

State of the art

As mentioned in the introduction, many research groups have found in the games an opportunity to show the potential of implementing technologies such as machine learning, deep learning, computer vision, robotic, etc. in manufacturing processes. In this context, games as Sudoku and the Tic-Tac-Toe, because of its simplicity, are suitable for this purpose. In this section, we will review previous research aligned with this purpose, describing the methodology implemented.

In [7], the element actuator is a robot Lego Mindstorm NXT equipped with a pen and a light sensor. The light sensor scans the grid in order to identify the non-empty and empty cells. Then, the robot repeats the scan process to identify the numbers in the corresponding cells. For the identification, the algorithm comprises a series of image processing techniques, such as thresholding, segmentation, and thinning operations. Second, the algorithm compare the processed image resultant with some characteristic features in such a way that the number is recognized based on the match with these characteristic. Now, an algorithm based on the principle of uniqueness is used to solve the Sudoku puzzle. Finally, the robot writes the final solution using the pen. More details in the cited paper.

In [8], the element actuator is again a robot Lego Mindstorm NXT equipped also with a pen and a light sensor. Once again, the light sensor is used to scan and identify the empty and non-empty cells. The robot repeat this scan process in each turn of the game to identify the opponent's moves. To identify the opponent's marker (either nought or cross), just like [7], the robot first scan the cell where have identified a marker, and for the identification, the algorithm comprises a series of image processing techniques such as thresholding, segmentation, and thinning operations. Then, the algorithm performs a matching operation. To compute the next robot's movement, a highest priority algorithm is used. Finally, on each turn, the robot draws his response based on the opponent's move using the pen. More details in the cited paper.

Unlike [7] and [8], [9] focus on the algorithm to play the Tic-Tac-Toe, i.e. the decision-making system to compute the optimal agent movements in responses to the opponent's

movements. The decision-making system is based on feedforward neural networks. Finally, they interfaced the game with the robot manipulator Scorbot-ER VII. More details in the cited paper.
In [12], now the agent robotic is an NAO robot endowed with two hands that allows it to carry out complex tasks. The research focus on the forward and inverse kinematics of the robot, just like trajectory planning algorithms. The aim is that the robot can play the Tic-Tac-Toe game by picking and putting pawns that represent the player's markers. It is worth mentioning that the research only sought to endow the robot with the ability to play the game, i.e. the robot can play but does not yet know how. More details in the cited paper.
In [11], a swarm of Nano-UAVs replaces the agent robotic to play the Tic-Tac-Toe game. Here, they use a camera, just under the grid, for the game state evaluation. To identify the non-empty and empty cells, a computer vision system (comprising a camera) identifies them using, just like [7] and [8], image processing techniques such as gray level transformation, thresholding, morphological and find counter operations. The player's marker recognition is based on a pixel density analysis. The research focus, just like [9], on the decision-making system, which is based on reinforcement learning. More details in the cited paper.
We can highlight the following points from the research reviewed:

- The grid is composed of straight lines and is drawn symmetrically.
- A light sensor, a single camera, or a vision stereoscopic system can be used as the sensor device for the grid recognition system. For the case of a light sensor or a single camera, we can locate the sensor device just under the grid to simplify the space-to-space transformation, i.e. from image coordinates to world coordinates.
- The agent robotic can be from a Lego robot, a robot manipulator, until even a swarm of Nano-UAVs.

We will take these points into account for the proposal of the system to be developed.

Chapter 4

Proposed system overview

The Tic-Tac-Toe game is a 3x3 grid puzzle with nine squares which are initially empty. Two players alternatively take a turn to fill these squares by noughts (**O**) and crosses (**X**). We use these markers to differentiate the occupied squares between the two players. The aim of the game is to fill a straight line with the same marker in either a horizontal, vertical or diagonal direction [8][9][11]. Based on the game's description, we can describe how a game between a human and a robot would be conducted by the steps below:

1. The human player (opponent) goes to the whiteboard and draws the grid.

2. Now, the robot has to identify the grid and determine the coordinates of the centroids of the nine squares that comprise it. These squares represent the playing field in which the robot can draw a nought or cross. Then, the robot has to wait for the opponent's move.

3. Once the opponent has realized his first movement, the robot has to locate the opponent's marker, i.e. the robot has to know the square that the opponent has filled, as well as which squares are empty. The empty squares represent the free playing field in which the robot can draw. Once the robot has located the opponent's move, it has to identify which marker it is: (**O** or **X**), as this way it will know which marker represents the opponent, and by default, its own.

4. Then, based on the aim of the game, the robot has to decide on the next move, i.e. in which square the robot should draw the marker that represents it.

5. Based on the decision taken, the robot has to execute its move.

6. Finally, the third through fifth steps are repeated until one player meets the aim of the game and wins. If we do not reach to this aim, i.e. the human and the robot have filled all the squares, we declare the game as a draw.

Now the question is: What systems should the robot have to follow the game flow described above? To facilitate the identification of the systems, from the steps above, we make the game flowchart shown in Figure 4.1.

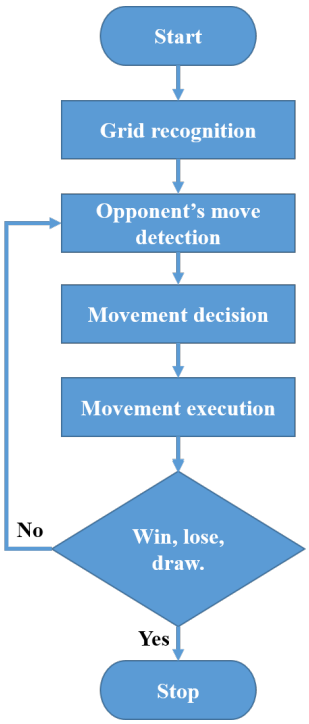

Figure 4.1: Tic-Tac-Toe flowchart

From the flowchart, we identify the tasks below:

- Grid recognition.
- Opponent's marker detection.
- Movement decision.
- Movement execution.

Based on the tasks identified, we propose to endow the robot with a computer vision system to meet with the two first tasks. In this way, we propose the development of the systems below:

- **Grid recognition system:** System that will allow to the robot to compute the coordinates of the centroids of the nine squares that comprise the 3x3 grid.
- **Opponent's marker detection system:** System that will allow to the robot to localize and recognize the opponent's marker.

Besides the tasks mentioned in the flowchart, there is an implicit task necessary so that the robot could to use correctly the information acquired by the camera. We define the system to meet this task below:

- **Camera-based positioning system:** System that will allow to the robot refers the information acquired by the camera regarding its base.

By other hand, although the motion is a robot's own capability, we need to carry out a study of its kinematics in order to design the trajectories that the robot must execute to make possible its interaction with the human in the game. It is worth mentioning that the thesis project does not focus on the decision-making system. Finally, in Figure 4.2, we present the experimental module in which we will implement the three listed systems.

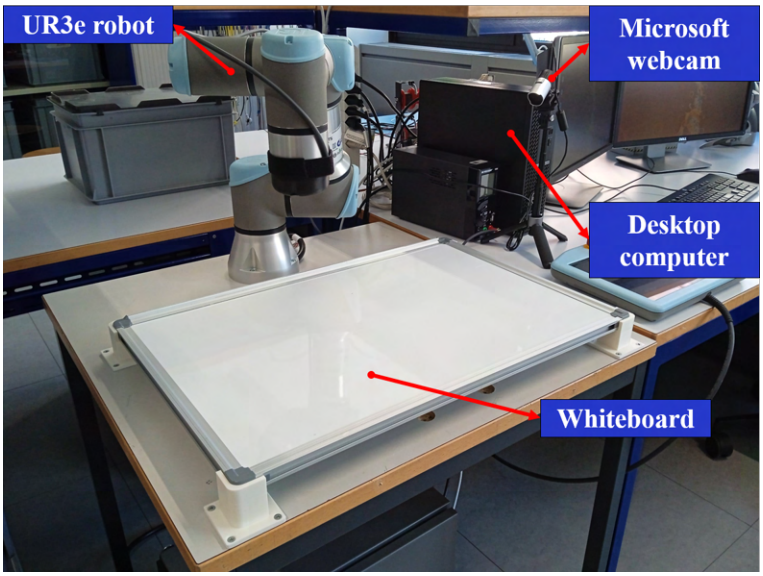

Figure 4.2: Experimental module

The experimental module is comprise by:

- **UR3e robot:** The robotic agent who will be endowed with a degree of intelligence to be able to play Tic-Tac-Toe.
- **Microsoft webcam:** The sensor device whose output (image) will be the input for the computer vision system. Therefore, for the grid recognition and opponent's marker detection systems.
- **Desktop computer:** Processing unit where we will execute all the algorithms involved in the development of the systems.
- **Whiteboard:** It represents the available playing space where the grid is drawn.

Now, we will dedicate the rest of the document to describe the design and implementation of the systems proposed. As well as the study of the kinematics of the UR3e robot.

Chapter 5

Grid recognition system

5.1 System description

The aim is to develop a system capable of computing the coordinates of the centroids of each square composing the grid. To develop the algorithms involved in the system's development, we will first make use of images got under simulation. Finally, we will use actual images to adjust the first approach to the algorithms and adapt them to actual situations.

5.2 Implementation description

To guide the reader, in Figure 5.1, we show the flowchart of the operations that comprise the grid recognition system. We will now detail the algorithms involved in each operation.

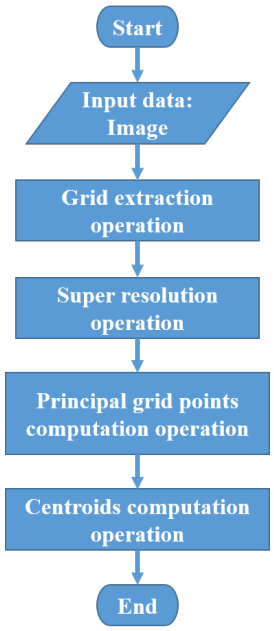

Figure 5.1: Grid recognition system flowchart

5.2.1 Grid extraction operation

We start by showing in Figure 5.2 the input image.

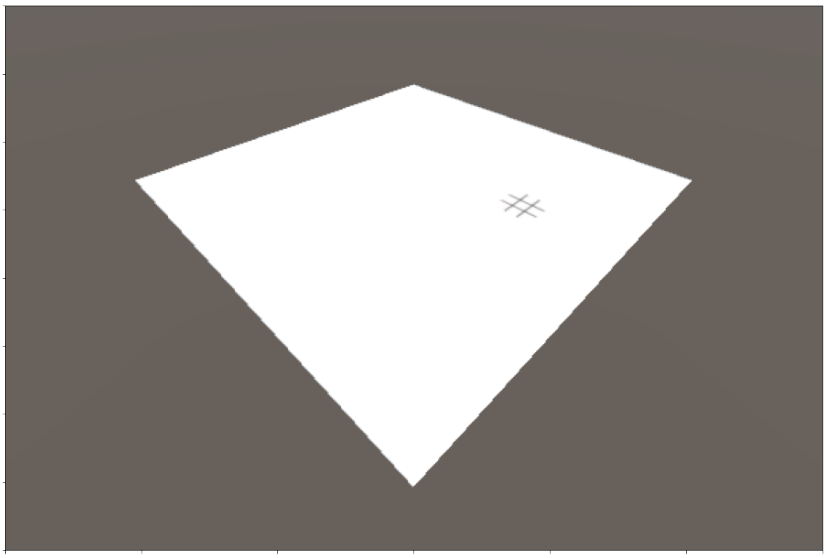

Figure 5.2: Input image

The first objective is to extract the grid from the image. For that, we propose a first step comprising detect the grid in the image. The detection algorithm is based on the Canny edge detector algorithm [13]. Therefore, because most of edge detection algorithms need a grayscale input image, we need to change the space color of the input image from RGB to grayscale [14]. In Figure 5.3, we show the input image in grayscale.

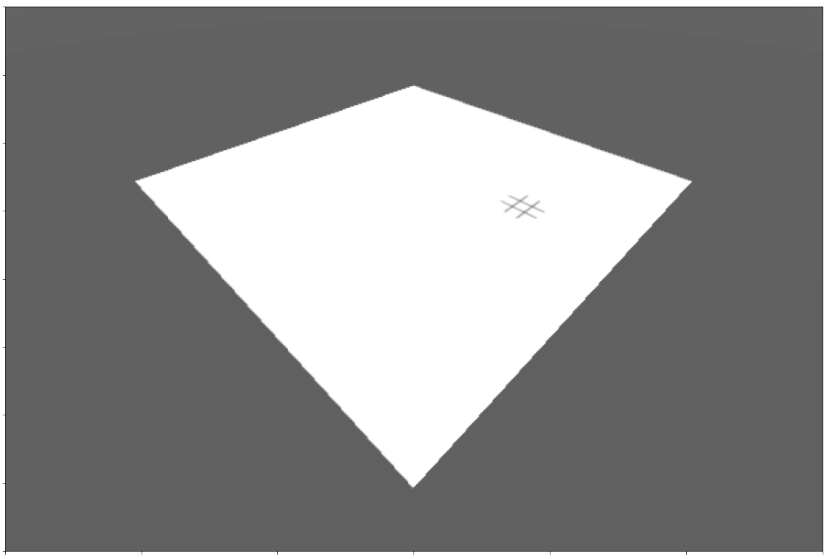

Figure 5.3: Grayscale input image

Once we have the input image in greyscale, we apply the Canny edge detector. Using $\sigma = 1$, we get the results shown in Figure 5.4. The resulting image is a binary image where the detected edges corresponding to the white pixels.

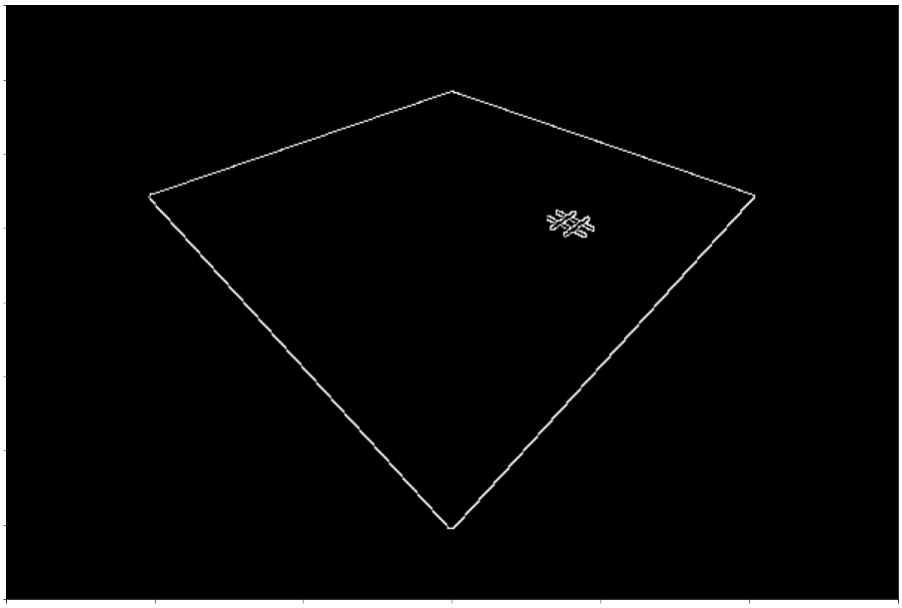

Figure 5.4: Edges detected ($\sigma = 1$)

Before continuing, we believe it is important to explain what is sigma (σ) in the Canny algorithm. The Canny algorithm is a multi-stage algorithm whose first stage is a noise reduction using a Gaussian filter (Blurring operation) [15]. Applying a Gaussian filter means convolve a Gaussian kernel with an image, where sigma represents the standard deviation of the Gaussian. This image filtering process helps to remove noise, in such a way that the higher the sigma value the image is much more blurry, while with a much lower sigma value, the blurriness is much less, allowing to the algorithm detect small and sharp lines. [16][17]. In Figure 5.5, we show how the results of the Canny algorithm vary depending on the sigma value.

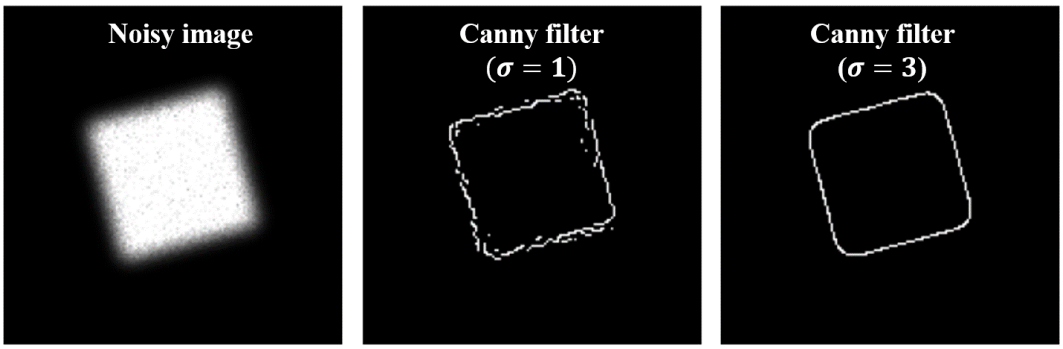

Figure 5.5: Results of the Canny algorithm as a function of sigma [13]

As we can see in the Figure 5.5, we can detect much more details in relation to the edges with $\sigma = 1$. Under this reasoning, we will remove the whiteboard edges and extract the grid. The procedure comprises first looking for a higher sigma value in such a way that we only detect the whiteboard edges. We achieve this using $\sigma = 8$. In Figure 5.6, we show the results. As we can see, we detect only the whiteboard edges.

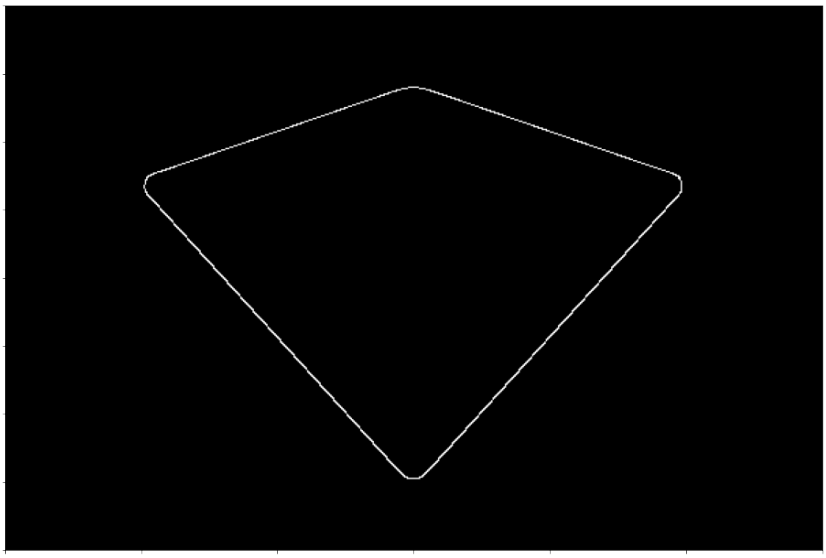

Figure 5.6: Whiteboard edges ($\sigma = 8$)

Second, from the resultant binary image, which contains only the whiteboard edges, we create a binary mask. First, we apply a morphological operation to the binary image. A morphological operation is the most common binary image operation, which changes the shape of the underlying binary image objects. Basically, we convolve the binary image with a binary structuring element (either a 3x3 box filter or more complicated disc structures), and select a binary output value depending on the thresholded result of the convolution [14]. In figure 5.7, we show the most common morphological operations applied on an image.

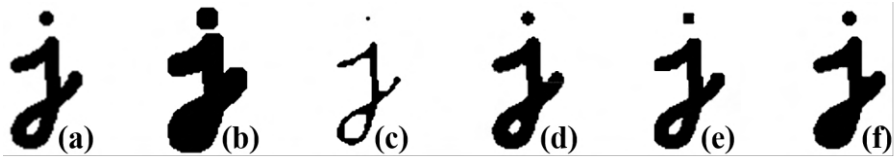

Figure 5.7: (a) Original image, (b) dilation operation, (c) erosion operation, (d) majority operation, (e) opening operation, (f) closing operation [14]

Continuing with the binary mask creation, we decide to apply a dilation operation. Then, we invert the pixel values of the resultant dilated binary image in such a way that multiplying this binary mask with the image shown in Figure 5.4 we achieve to remove the whiteboard edges and keep the grid. In Figure 5.8, we show the operations described above.

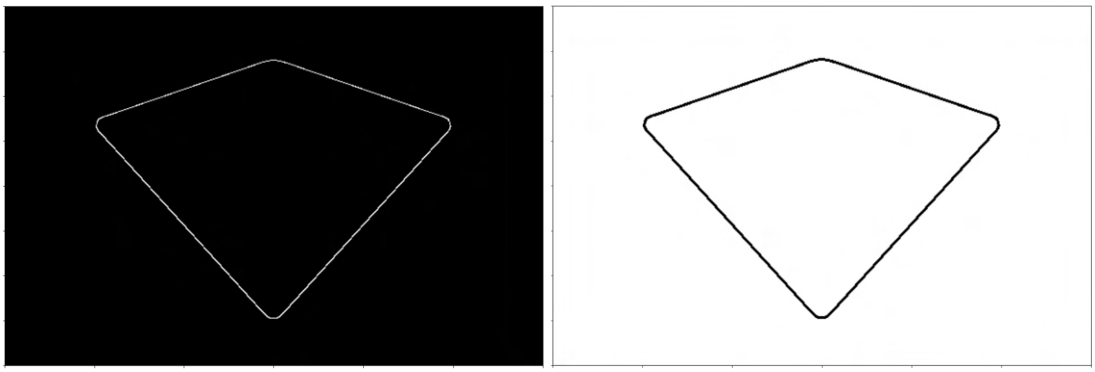

Figure 5.8: (right) Dilation operation applied to the whiteboard edges, (left) binary mask resulting from inverting the pixels values

Finally, using the binary mask, we achieve to detect the grid in the input image. In Figure 5.9, we show the results.

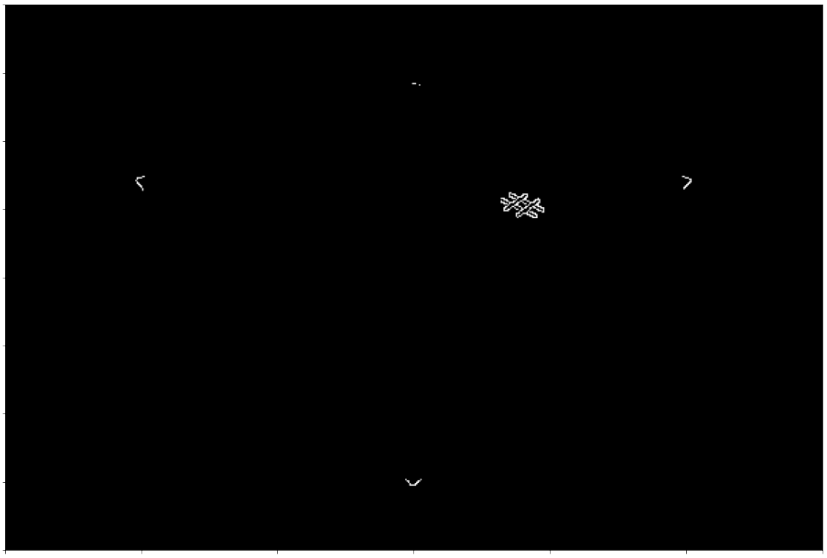

Figure 5.9: Grid detection

Although we have apparently achieved to detect only the grid, we can still observe in Figure 5.9 some tiny parts of the whiteboard edges. To solve this, the second step of the extraction operation comprising cropping the grid detected. The cropping algorithm is based on an image segmentation algorithm. Image segmentation algorithms aim to group pixels that have a similar appearance [14]. In this way, using Image labelling segmentation [18], the idea is to have, on the one hand, the region representing the grid and, on the other hand, the regions representing the remanent whiteboard edges.

Then, taking advantage of the properties of these labeled image regions [19], we can cropping the grid detected regardless of the presence of the tiny edges, e.g. in this case we decide to keep only the labeled image region with the largest area. In this way, we eliminate these tiny parts of the image. In Figure 5.10, we show the results. The grid is bounded by the blue rectangle, also computed from the properties of the labeled image region.

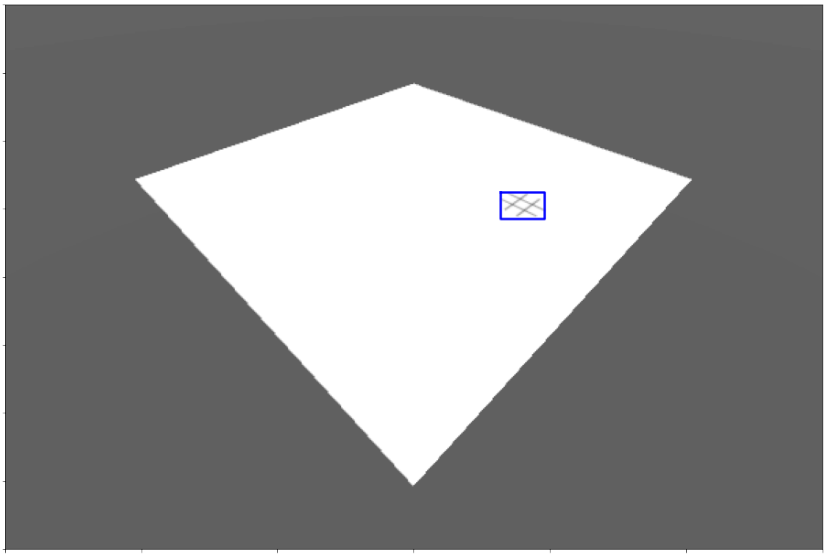

Figure 5.10: Grid detection and cropping

Finally, extracting only the region bounded by the blue rectangle, we achieve to extract the grid from the input image. In Figure 5.11, we show the grid extracted. However, as we can see, although we have successfully extracted the grid, the image got has a low resolution, i.e. the image now has a lower level of detail, which does not help the image processing [20].

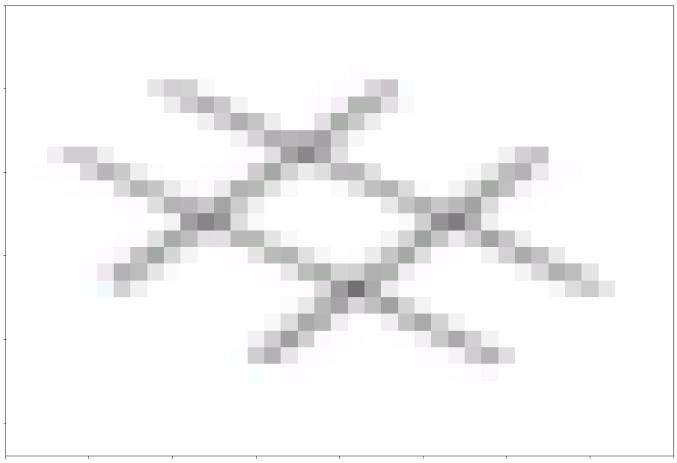

Figure 5.11: Grid extracted from the input image

5.2.2 Super resolution operation

From the extraction operation, we have an image of the grid but with a low resolution, which will not help further processing. Therefore, the second objective is to increase the resolution of the image got in Figure 5.11 without degrading the quality. For that, we propose the use of a Super resolution operation based on Deep Learning. We decide to use the EDSR model, which is a pre-trained model, i.e. we don't need to train the model ourselves to use it. This model, based on a Deep Leaning architecture, increases the input image resolution by 4x. The thesis does not focus on how this model works or the training process. We can find all this information in [21]. Using the EDSR model, we get the resultant image shown in Figure 5.12.

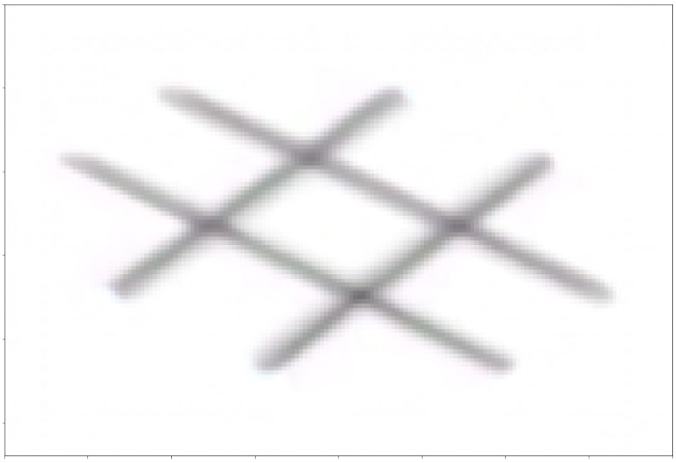

Figure 5.12: Grid extracted - resolution increased by 4

5.2.3 Principal grid points computation operation

The third objective is to compute the principal grid points. For that, we propose an algorithm based on the Straight line Hough transform algorithm, which is a method used to detect straight lines [22][23]. To use this algorithm, an edge detection pre-processing is desirable, however we propose an image pre-processing based on a Binarization and Skeletonization operations. The simplest way of binarization is thresholding. The Thresholding operation is used to create binary images from grayscale image, being a simplest way to segment objects from the background [24], and the Skeletonization operation reduce binary objects to 1 pixel wide representations [25].

We start first by changing the space color to grayscale, and then we apply a Gaussian filter with $\sigma = 3$ to the resultant image. We show these operations in Figure 5.13.

Then we apply the thresholding operation in order to have a binary image where the grid corresponds to the white pixels. Using the thresholding value: 220, we get the binary image shown in Figure 5.14.

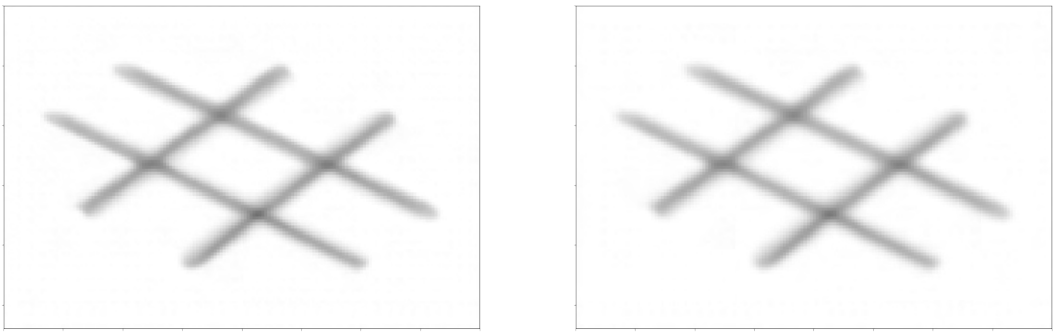

Figure 5.13: (right) Grayscale transformation, (left) blurring operation

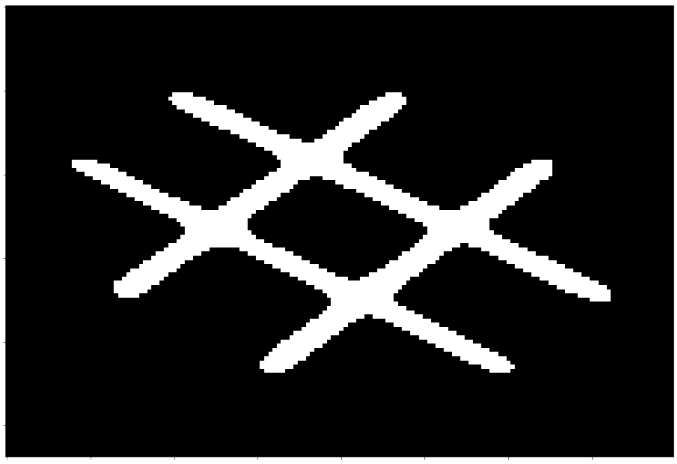

Figure 5.14: Grid binary image

Once we have the binary image, we apply the skeletonization operation. Besides to the skeletonization operation, we use a segmentation operation in order to remove some remanent whiteboard edges in the image after the thresholding operation. This can be the case when we draw the grid near to the border of the whiteboard. We show these operations in Figure 5.15.

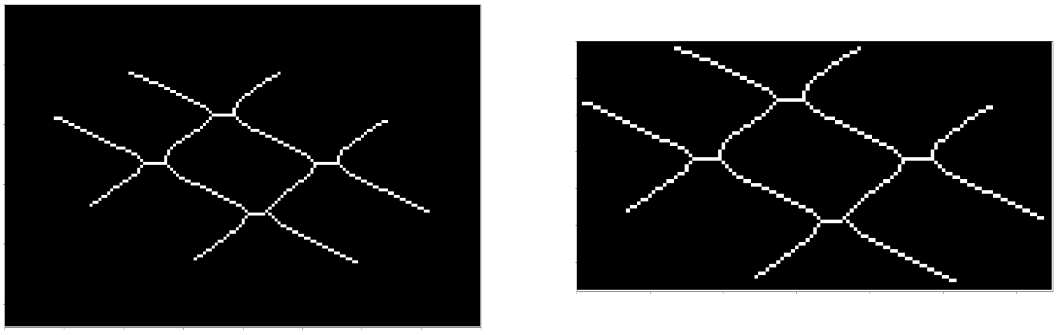

Figure 5.15: (right) Skeletonization operation to the grid binary image, (left) grid extraction using a segmentation operation

Once we have completed the pre-processing, we use the Straight line Hough transform algorithm to compute the 4 most straight lines that comprise the grid and then compute the intersection points of these lines. We show the results got in Figure 5.16.

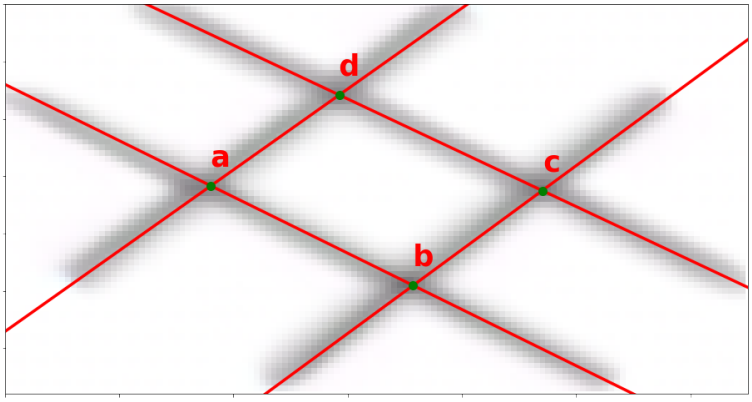

Figure 5.16: Intersection points of the 4 most straight lines

Finally, we use the central grid points (a, b, c, d) to compute, using geometric relationships, the extreme grid points. The principal grid points will be the complement between the central grid points and the extreme grid points. We show the results in Figure 5.17.

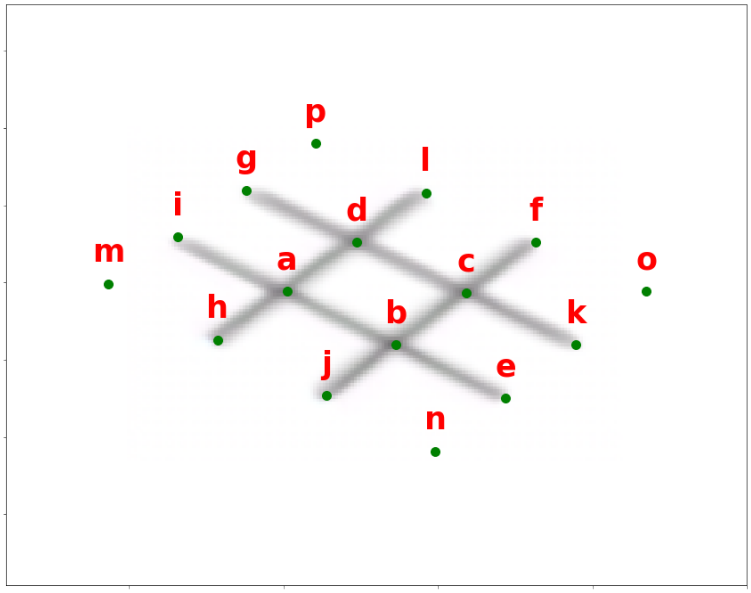

Figure 5.17: Principal grid points

5.2.4 Centroids computation operation

The final objective is to compute the coordinates of the centroids of the nine squares that comprise the grid. For that, we use geometric relationships among to the principal grid points computed. We show the results in Figure 5.18.

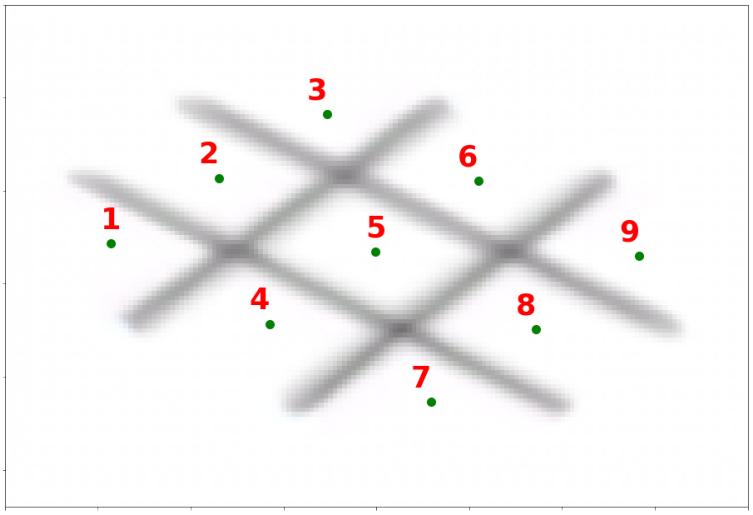

Figure 5.18: Centroids of the nine squares that comprise the grid

After detailing the operations involved in the grid recognition system, in Figures 5.19 and 5.20, we show the results after applying the grid recognition to two different simulated images. We also show the corresponding pixel coordinates of the centroids.

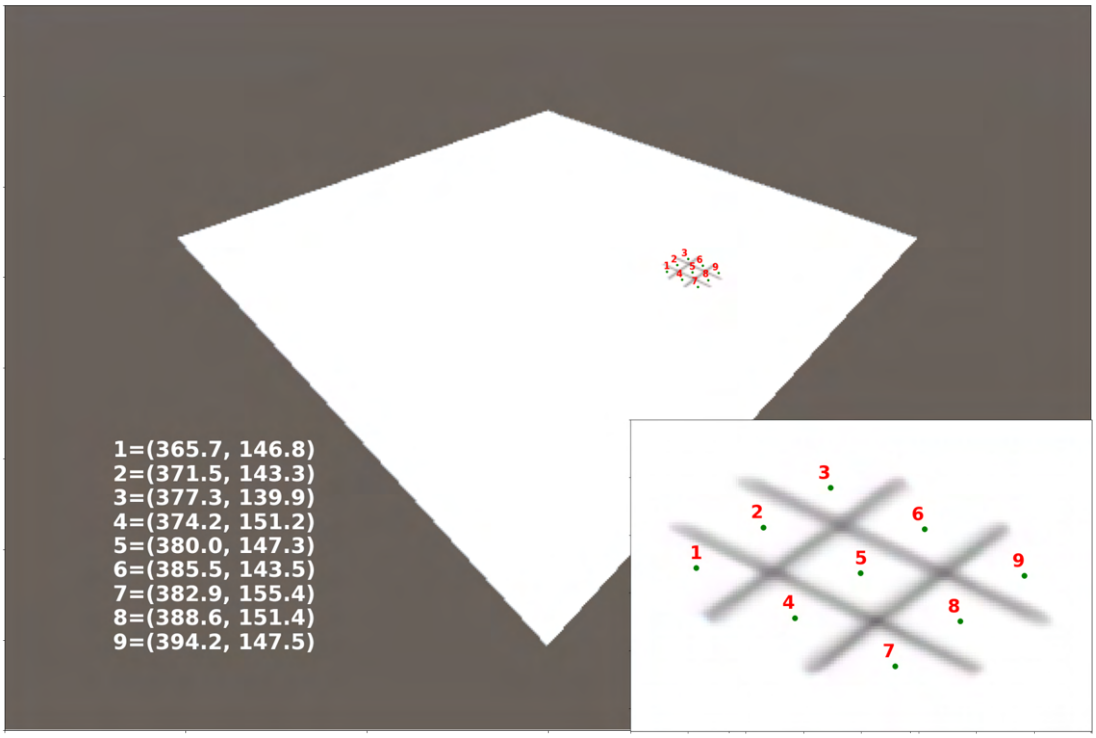

Figure 5.19: Results of the grid recognition - Image simulated 1

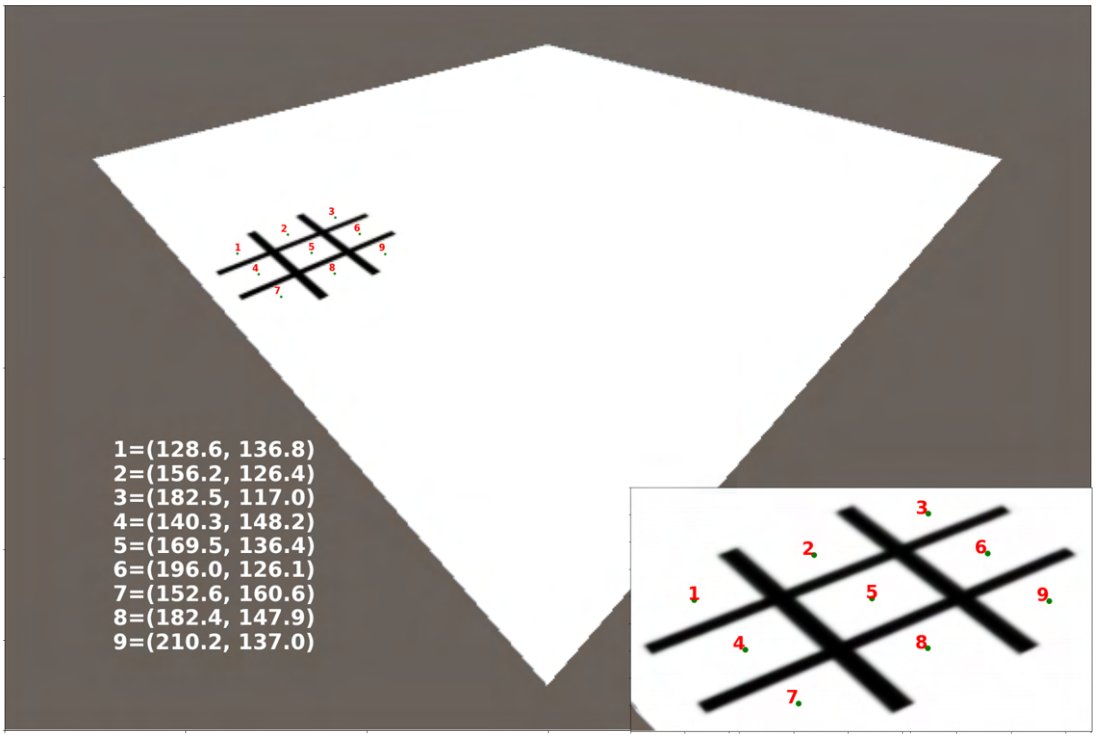

Figure 5.20: Results of the grid recognition - Image simulated 2

5.3 Real implementation

As mentioned in the chapter introduction, the first approach of the algorithms involved in the grid recognition system is based on simulated images, i.e. ideal images that unfortunately will differ from the ones got in actual tests. Therefore, in order to adapt the algorithms to actual images, we check each operation pointed out in the grid recognition system flowchart (Figure 5.1) but using images got in the experimental module. In this way, we can easily identify at which point of the algorithms involved in the different operations we have to adjust some parameters or change the strategies to meet the objectives.
Regarding the extraction operation, we apply the operation to a real image and we get the results shown in Figure 5.21.

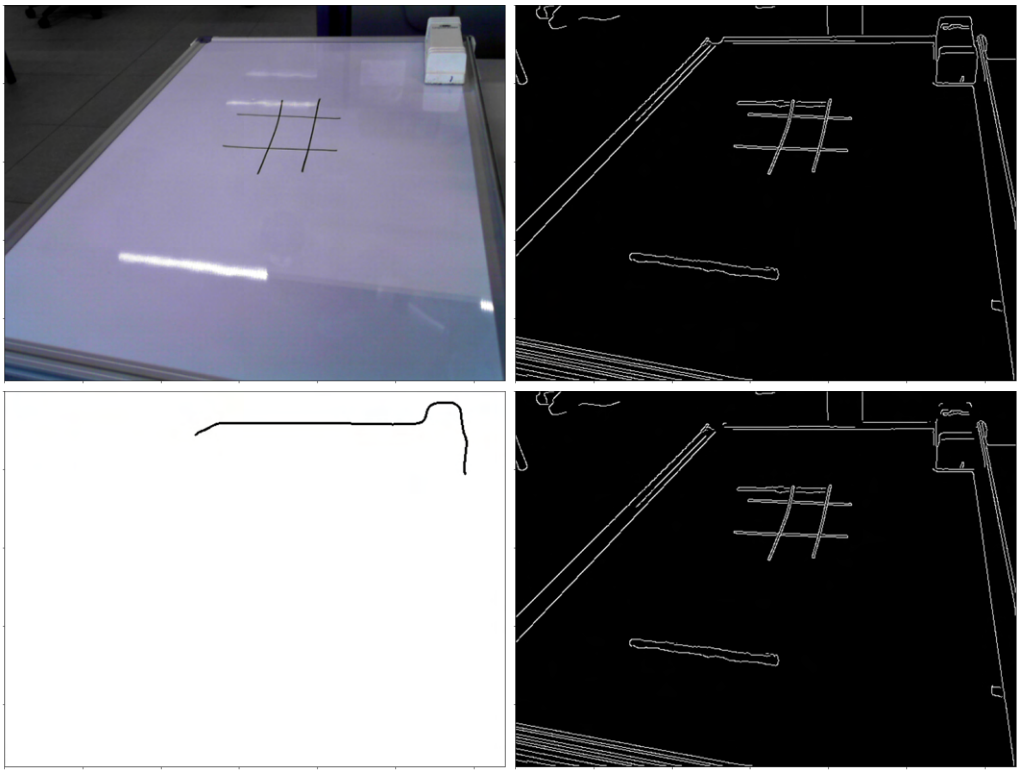

Figure 5.21: Grid extraction operation: (upper left corner) Input image, (upper right corner) edge detection ($\sigma = 1$), (lower left corner) binary mask creation ($\sigma = 1$), (lower right corner) results after applying the binary mask

As we can see, we do not get a useful binary mask to remove the whiteboard edges. This is because the sigma value is too high. Therefore, the first adjustment that we will make is to reduce this value. Based on trial and error, we decide to use $\sigma = 4$. Besides, we will apply a Gaussian filter ($\sigma = 5$) before applying the Canny algorithm, a greater dilatation to the binary mask, and a final dilation operation to the resulting image after applying the binary mask. We show the new results in Figure 5.22.

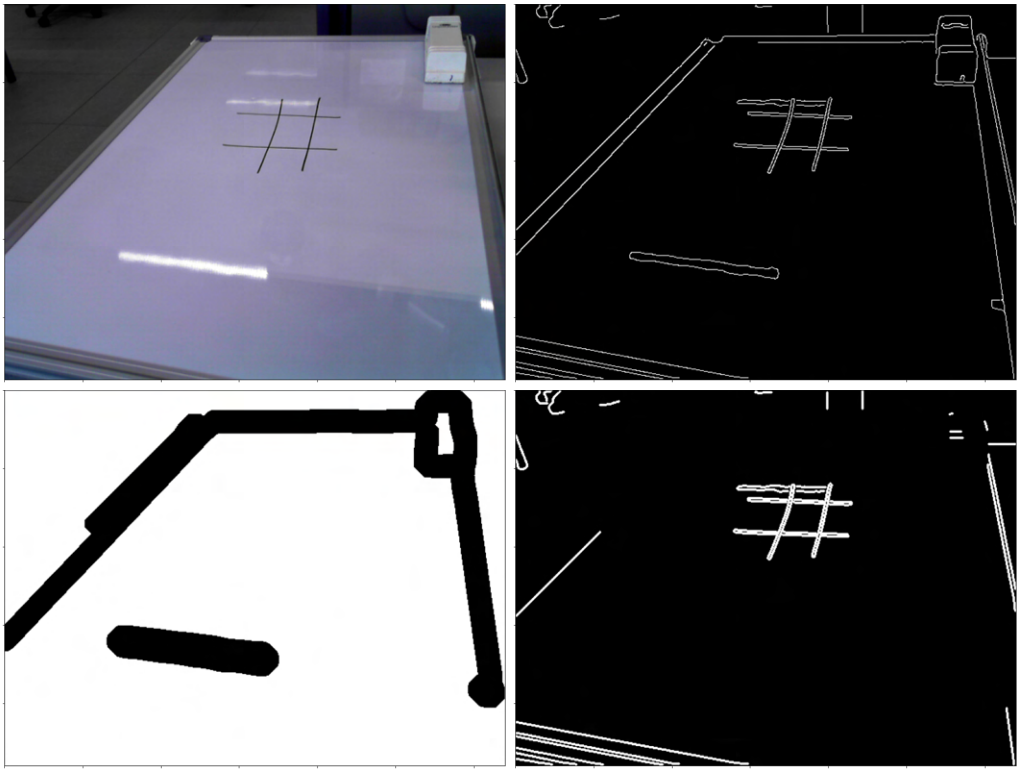

Figure 5.22: Grid extraction operation: (upper left corner) Input image, (upper right corner) edge detection ($\sigma = 1$), (lower left corner) binary mask creation ($\sigma = 4$), (lower right corner) results after applying the binary mask and a dilation operation

From the results shown, we can observe that we have been able to remove some of the whiteboard edges. However, we still have some edges of the whiteboard and of objects present in the images, which we could not remove.
Following the extraction operation, we apply a segmentation operation. As explained above, the idea is to group the edges into regions in such a way that we can cropping and extracting the region that corresponds to the grid. In Figure 5.23, we show the results after applying the complete extraction operation to some different actual images.

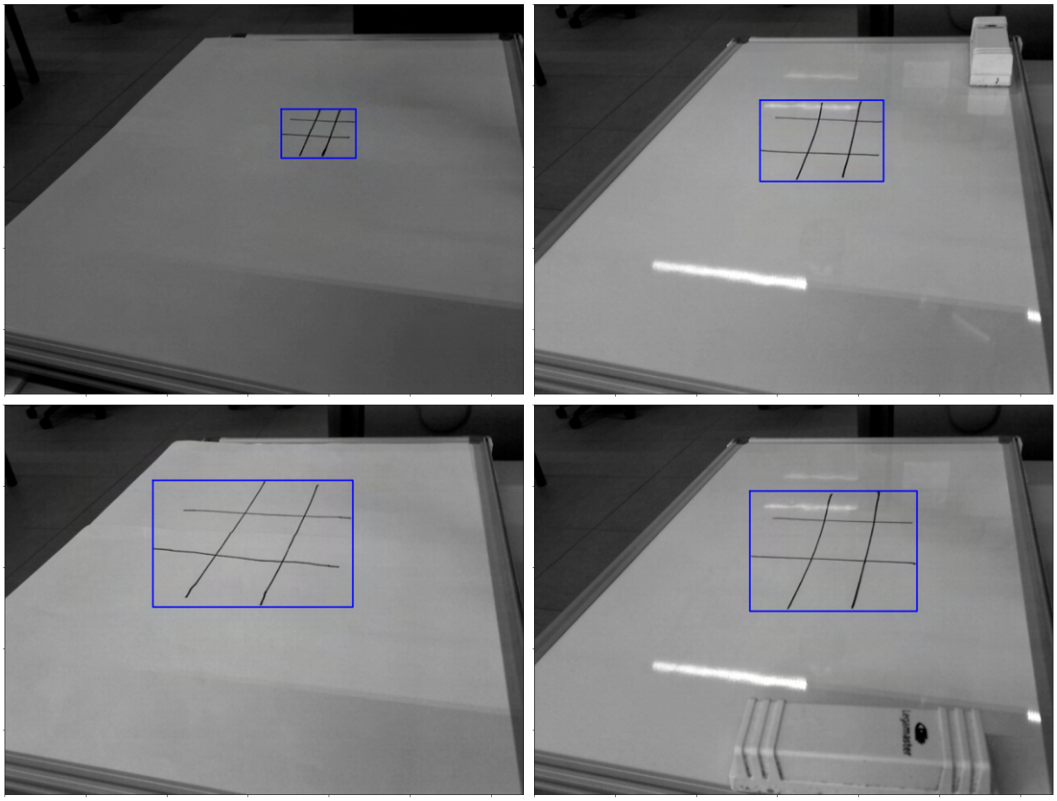

Figure 5.23: Grid detection and cropping results

As we can see, we have successfully carried out the extraction operation. As shown in the results, regardless of the effects of reflectance or the presence of other objects in the image, we extract the grid.

Regarding the Super resolution operation, we choose one of the extracted grids and apply the super resolution technique. In Figure 5.24, we show a grid image extracted and its corresponding enhanced version (resolution x4).

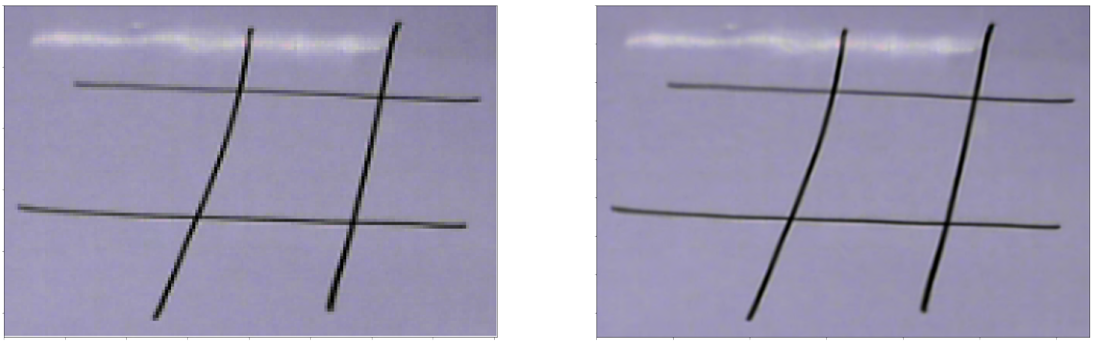

Figure 5.24: Super resolution results

5.3.1 Principal grid points computation operation improvement

5.3.1.1 Binarization operation improvement

In the principal grid points computation operation, the binarization operation is one of the most crucial operations for the pre-processing necessary to use the Straight line Hough transform. In Section 5.2.3, we have fixed a thresholding value of 220, however we were dealing with simulated images under constant luminance conditions. Therefore, it is no longer possible to use a simple thresholding operation, since the images are now subject to different lighting conditions. To solve this, we propose the use of the Otsu's binarization method, which avoid to establish a fixed thresholding value, determining automatically an optimal global threshold value from the image histogram [26]. The method is suitable for bimodal images, which is our case since we want to separate two different pixel values: the pixels belonging to the grid, and those belonging to the background. In Figure 5.25, we show the results after applying the Otsu's method to different actual images (with different lighting conditions) and to apply an erosion and dilation operations to the resultant binary image.

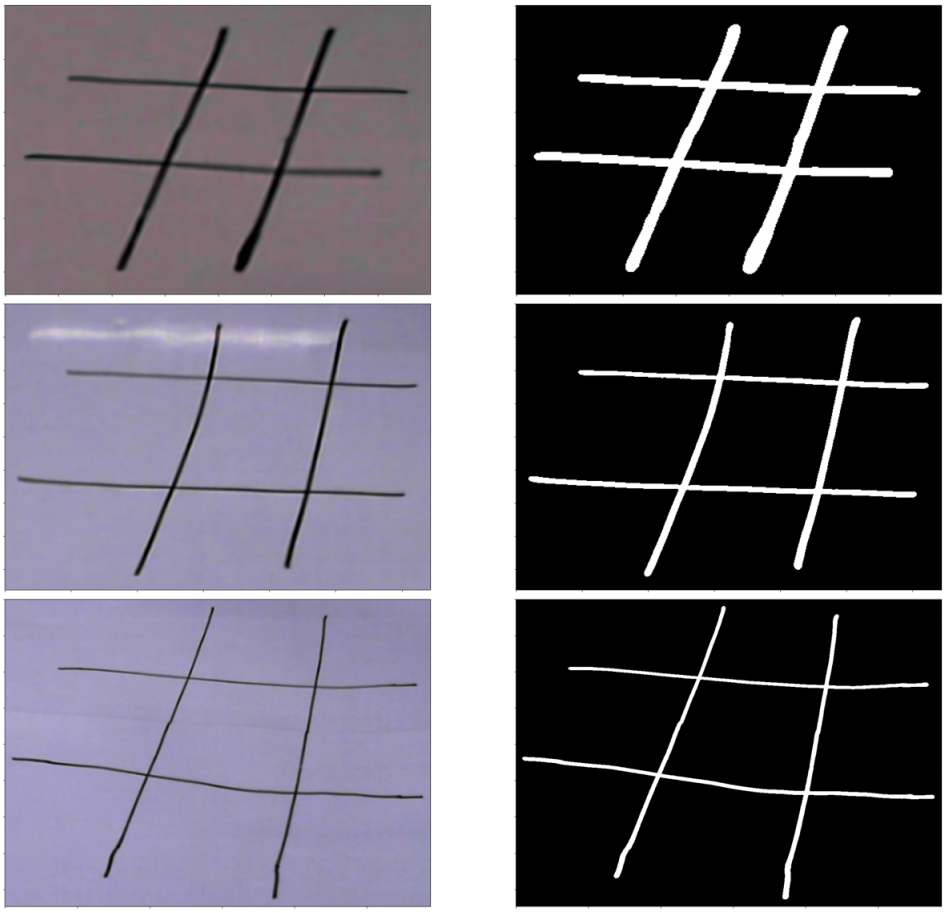

Figure 5.25: Results after applying the Otsu's method

5.3.1.2 Points computation method improvement

In Section 5.2.3, based on the central grid points, we proposed the use of geometric relationships to compute the extreme grid points and, with this, the principal grid points. However, we were considering a perfectly symmetric grid. Thus, it is not possible to remain with this approach to compute the extreme grid points because, in an actual test, the opponent can design the grid asymmetrically. Based on this, we propose a new approach comprising first, the computation of the convex polygon containing the grid in the binary image, second, the computation of the 8 straightest lines composing the polygon, and finally, the computation of the intersections of these 8 lines.

We start by computing the convex hull of the grid in one of the binary images shown in Figure 5.25. The convex hull of a binary image is a set of pixels included in the smallest convex polygon surrounding all white pixels in the image [27]. We show the results in Figure 5.26.

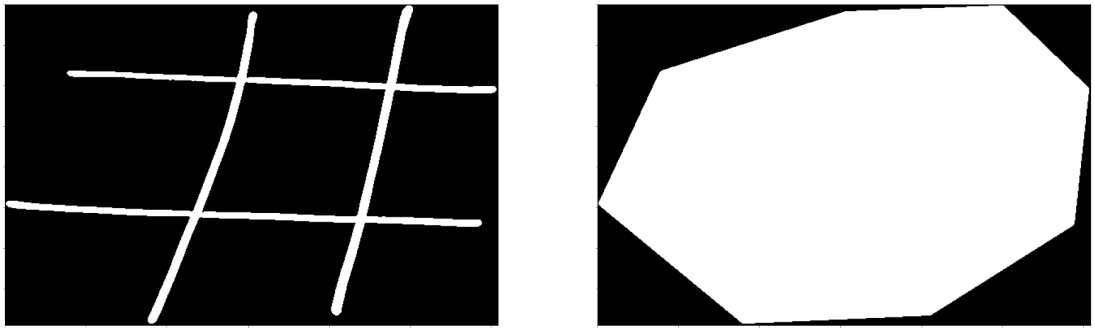

Figure 5.26: (right) Grid binary image, (left) convex hull operation

Now, we apply a Gaussian filter to the binary image got in the convex hull operation, and then we apply a Canny edge detection. We show the results in Figure 5.27.

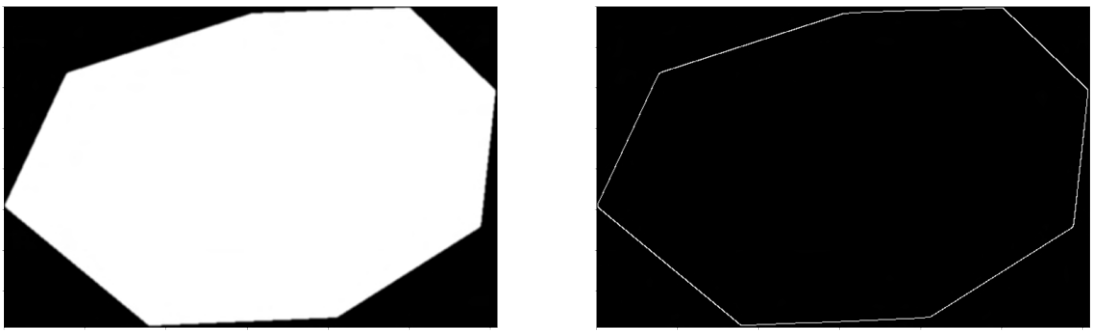

Figure 5.27: (right) Blurring operation, (left) edge detection operation

Once we have the edges of the polygon surrounding the grid in the binary image, we use the Straight line Hough transform to compute the 8 most straight lines that comprise the polygon and then compute the intersections of these lines.

We show the results in Figure 5.28.

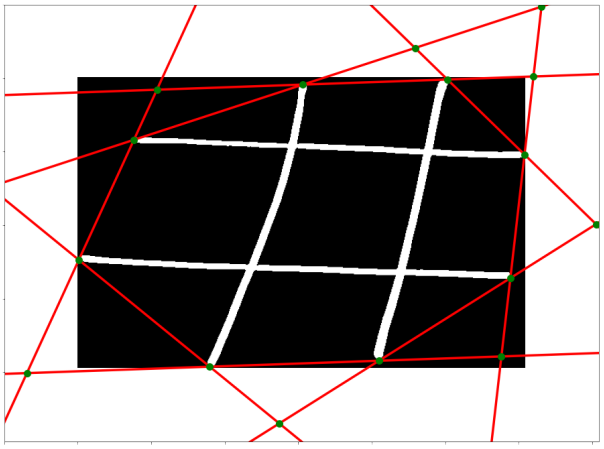

Figure 5.28: Intersection points of the 8 most straight lines

As we can see in Figure 5.28, we have additionally inserted the grid. With this, we want to highlight that it is possible to compute the 12 extreme grid points from the lines intersection. However, as we can see, because of the geometry of the polygon, we have over 12 intersections.

To solve this, we propose a new approach which comprises first computing the extreme grid points assuming that the grid has been drawn symmetrically (Section 5.2.3), and second, using these points as a reference to determine which of the points got from the intersection of the 8 lines are the extreme grid points. In Figure 5.29, we show the extreme grid points computed assuming a symmetrical grid (blue points), and the points got from the lines intersection (green points).

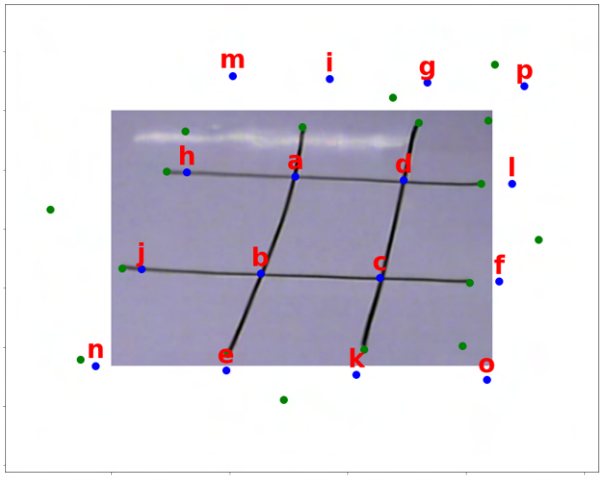

Figure 5.29: Blue points: principal grid points computed assuming a symmetrical grid, green points: intersection points computed from the lines intersection

The determination of the extreme grid points comprises the following steps:

1. For each blue point, we look for the three nearest green points.
2. Then, we computing the distance from each green point to the straight line of the grid containing the blue point.
3. Finally, the point with the shortest distance to the line will be the extreme grid point.

For example, in Figure 5.30, we observe enclosed in a blue circle the three green points closest to the blue point, which corresponds to the extreme grid point “g”. If we now compute the distance from each green point to the straight line passing through “d” and “g” (red line), we can conclude that the green point enclosed in the red circle is the extreme grid point.

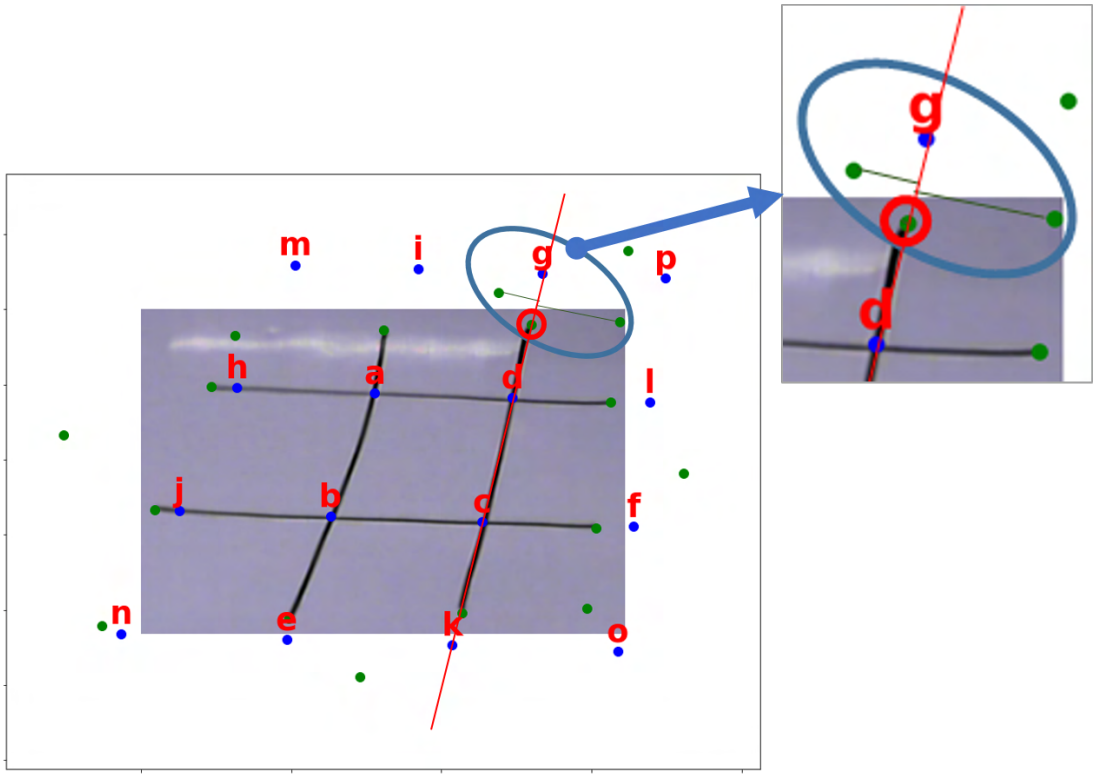

Figure 5.30: New approach to compute the extreme grid points

Finally, the principal grid points will be the complement between the extreme and central grid points. In Figure 5.31, we show the results after applying the new approach to compute the extreme grid points to some actual images. Considering that the grid is composed of straight lines, we can still calculate the central grid points according to Section 5.2.3.

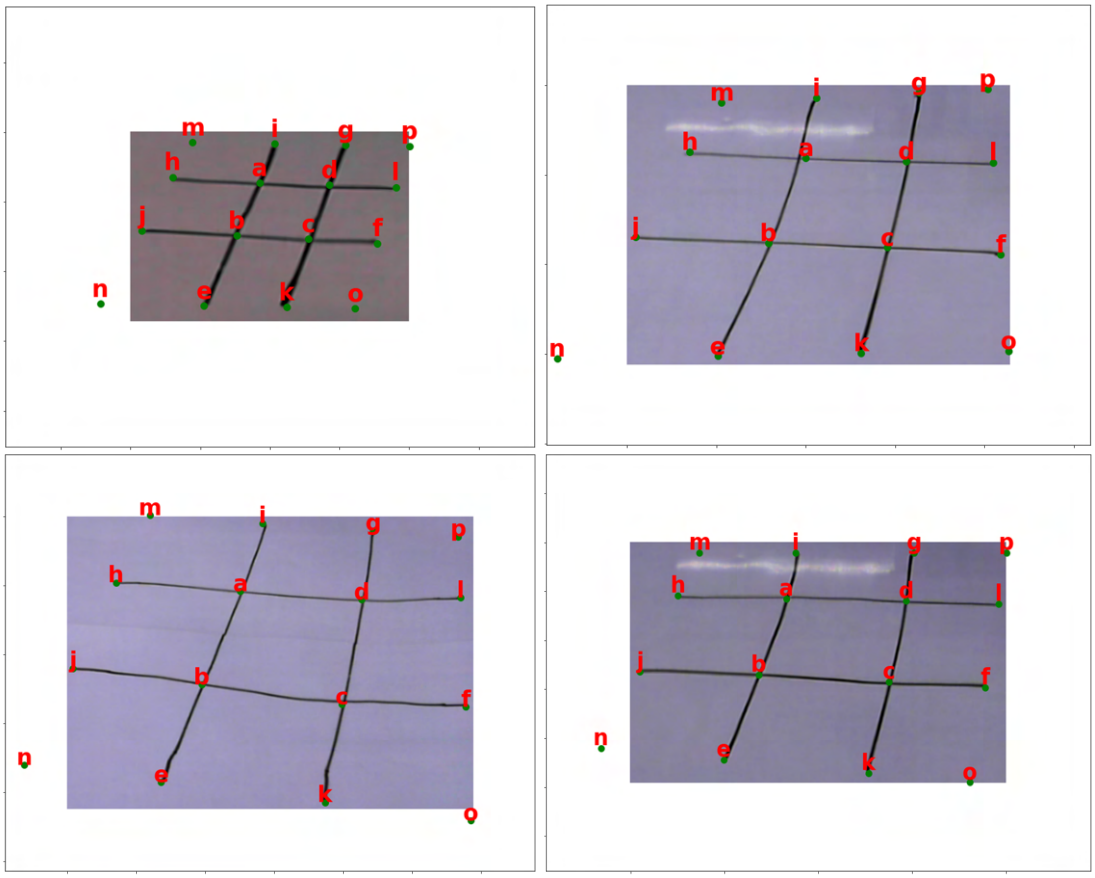

Figure 5.31: Principal grid points computation results - new approach

To conclude, according to the centroids computation operation (Section 5.2.4), from the principal grid points, we compute the centroids of the nine squares that comprise the grid. In Figures 5.32 and 5.33, we show the results after applying the adapted grid recognition to two different actual images. We also show the corresponding pixel coordinates of the centroids.

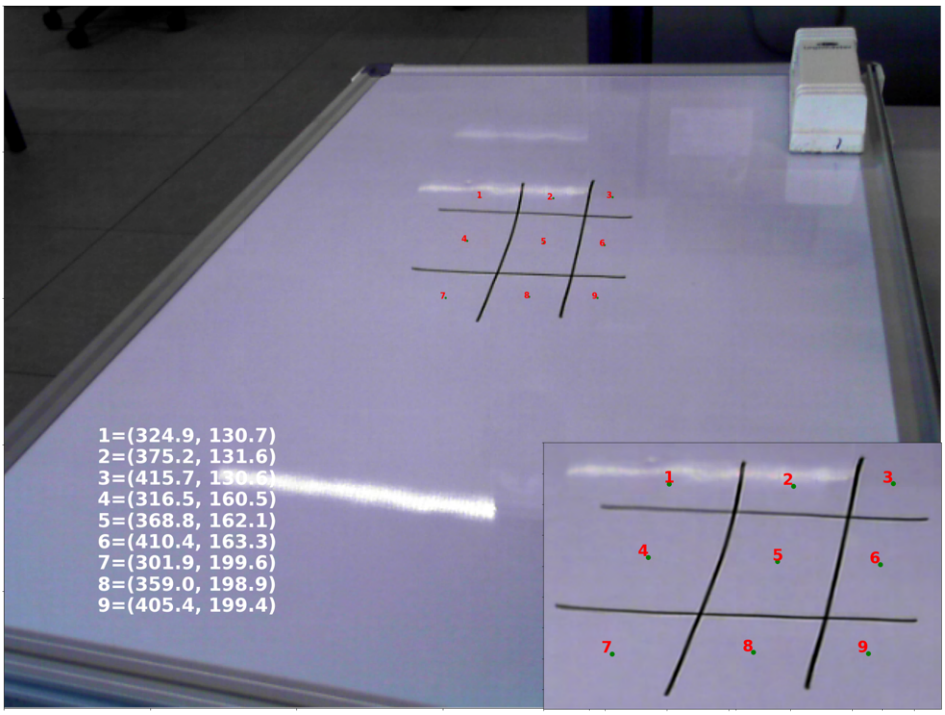

Figure 5.32: Results of the grid recognition - Actual image 1

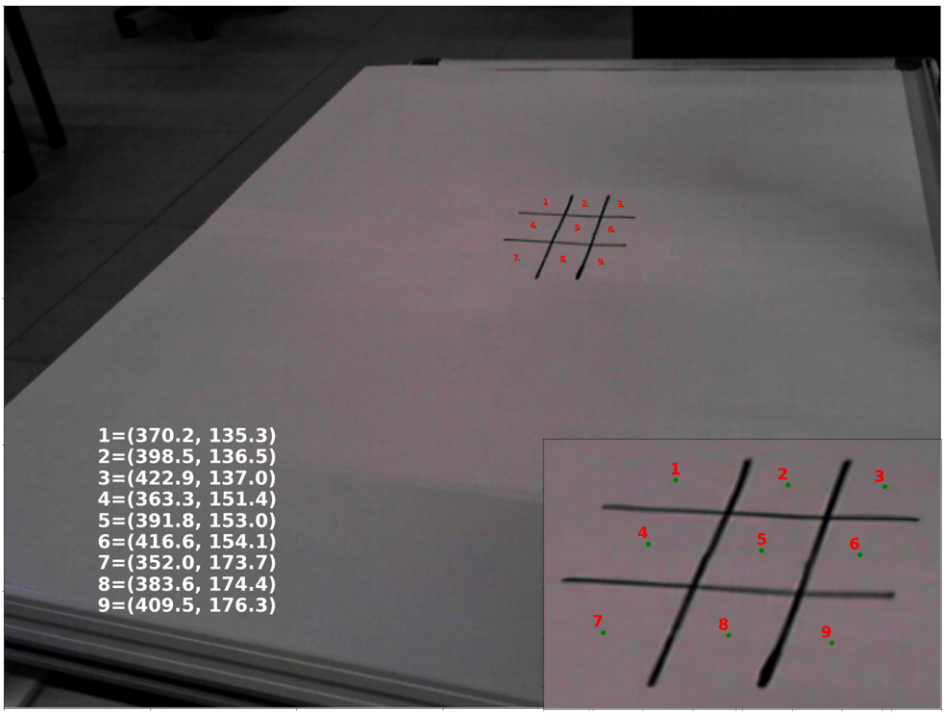

Figure 5.33: Results of the grid recognition - Actual image 2

5.4 Highlighted points

- Unlike [7][8][9][10][11][12], it is unnecessary for the grid to be drawn symmetrically. However, the condition to be maintained is that the grid is composed of straight lines.
- Regarding the grid extraction process, we can detect the grid even in the presence of some reflection of the illumination on the whiteboard. However, this does not imply that we actually draw the grid over some reflection, since the algorithm presents problems when this is the case, as shown in Figure 5.34.

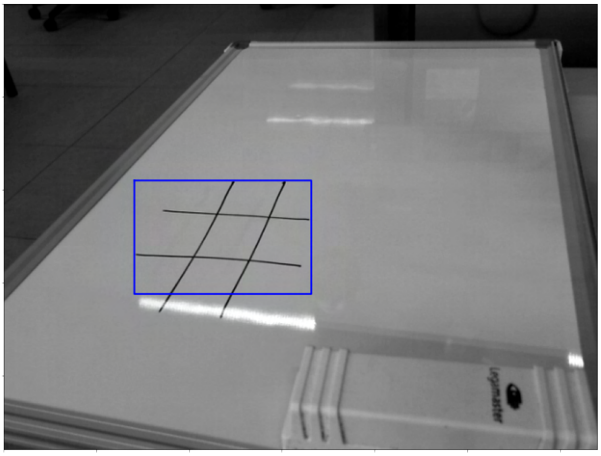

Figure 5.34: Reflection problem

Therefore, for an optimal performance of the grid recognition, we recommend the following options:

  - Draw the grid away from any reflections.
  - Change the reflectance effects of the whiteboard by adding a paper cover to its surface.

- The first step comprising the extraction operation is the grid detection. For that, we have used a strategy that comprises using the Canny edge detection algorithm for two different sigma values. However, we choose these fixed values by trial and error. An alternative improvement to perform this detection operation is to switch from an image processing based strategy to a deep learning model based strategy.

Chapter 6

Opponent's marker detection system

6.1 System description

The aim is to develop a system capable of recognising the opponent's marker, and therefore the marker to be used by the robot. For that, we propose the development of a Convolutional Neural Network (CNN) model, whose goal is to recognise whether the marker used by the opponent is a nought or a cross.

6.2 Implementation description

6.2.1 Model dataset

Deep learning models, unlike machine learning models, are much more sensitive to data quality, as they can learn minute details from the input data. Therefore, data quality is crucial for the development of these models [28]. Based on this, we have searched for the most suitable data to meet our aim. For the development of this detection system, we have decided to use the NIST Special Database 19 [29], in particular the data comprising images of the characters that comprise the alphabet. We show some of these in Figure 6.1.

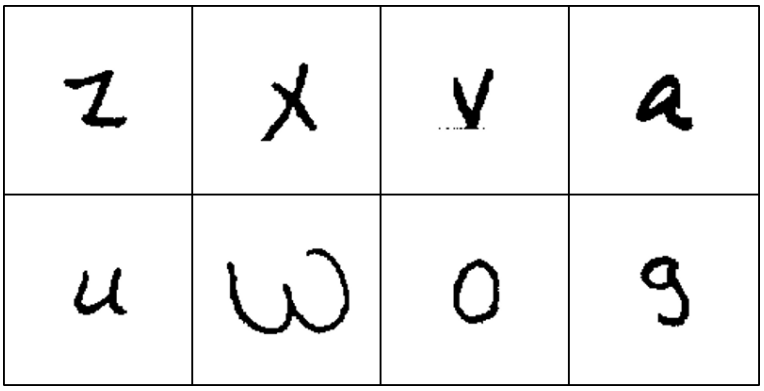

Figure 6.1: NIST Special Database 19 - Alphabet characters

Alignment with the objectives of the system, from the various characters available to us, we decide to use the characters "o" and "x" since we can basically use them to represent the nought and the cross. The dataset comprises 1000 images for each marker.

6.2.2 CNN modelling

The objective is to design and create a deep learning model based on Convolutional Neural Networks from the dataset shown in Section 6.2.1.
In Deep Learning, we will find the well-known Convolutional Neural Networks (CNNs), a kind of Feedforward Neural Network (FNN) which, instead of using fully connected layers, use convolutional layers in which a mathematical operation called convolution is used, being the crucial component that make them widely used for image processing and computer vision [30][14]
In Figure 6.2, we show the architecture chosen for the CNN model. From it, we will unpack the key concepts behind this type of neural networks.

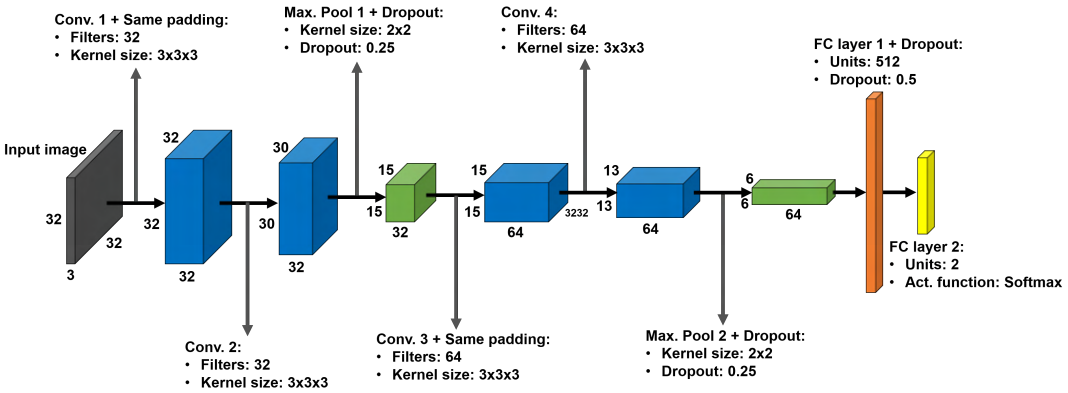

Figure 6.2: CNN architecture

The term networks is because the most common structure used in deep learning models is a chain structure such that a model is the composition of many functions. Each function in the chain is called a layer, and each one comprises units (neurons) [31]. Based on the proposed architecture, we will identify the three main types of layers to build CNNs:

- **Convolutional layers:** When talked of FNNs, we talk about networks which are composed of dense layers which learn global patterns in their input feature space. However, the CNNs instead of using dense layer, using convolution layers which allow learn local patterns. Regarding the images, these patterns are found in small 2D windows. For example, in Figure 6.3, we show how an image can be broken into local patterns (edges, textures, etc.) [30].

  In a nutshell, the convolutional layers are the core building block of the CNNs, which do most of the computational heavy lifting. In these layers, the convolution operation

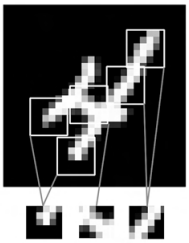

Figure 6.3: Local patterns in an image [30]

is carried out on 3D tensors, called feature maps. Each feature map is a 3D tensor with two spatial axes and a depth axis. For an RGB image, the depth axis is 3 that corresponding to the three channels: red, green and blue [32][33].

The principal goal of the convolution operation is to extract high-level information from the input feature map, and the element to achieve it is called Kernel (or Filter), being essentially the dot product between the kernel and local regions of the input, what is behind the convolution operation [30].

In this way, the convolution operation involves extracting patches from the input feature map and applies the same transformation to all these patches, producing an output feature map. This output map is still a 3D tensor whose depth depends on the convolutional layer, i.e. the number of kernels. In practice, we do not have convolutional layers formed by a single kernel, but we have layers formed by a set of them [32][33][30]. We illustrate the convolution operation in Figure 6.4 [30].

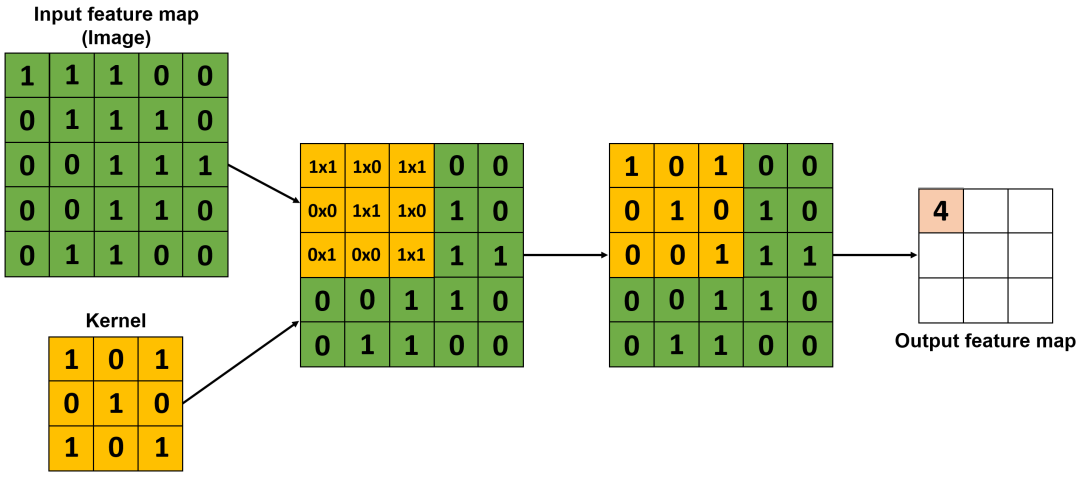

Figure 6.4: Convolution operation

We have an input feature map of size 5x5x1 and a kernel of size 3x3x1. The convolution works by sliding this kernel over the input feature map and applying the dot product. In this way, we extract the surrounding features [30].

It is worth mentioning that when we are working with RGB images, as in our case, the kernel will have a depth equal to the number of channels in the image, i.e. the kernel will have a depth of 3 [34].

The convolution operation leads us to two results: an output with a lower dimensionality than the input or an output where the dimensionality increases or remains the same as the input. We get the first result when apply Valid padding and the other one when we apply a Same padding [32].

- Same padding: We augment the dimension of the input before to apply the convolution operation. In this way, after the convolution, we will get an output with the same dimensions as the input. The proposed architecture has two convolutional layers applying the same padding: Conv. 1 and Conv. 3.
- Valid padding: If we perform the convolution operation without padding. The proposed architecture has two convolutional layers applying the valid padding: Conv. 2 and Conv. 4.

Based on the proposed CNN architecture, in Figure 6.5, we show the output maps after applying the first convolutional layer (Conv. 1), using the same padding, to a input image.

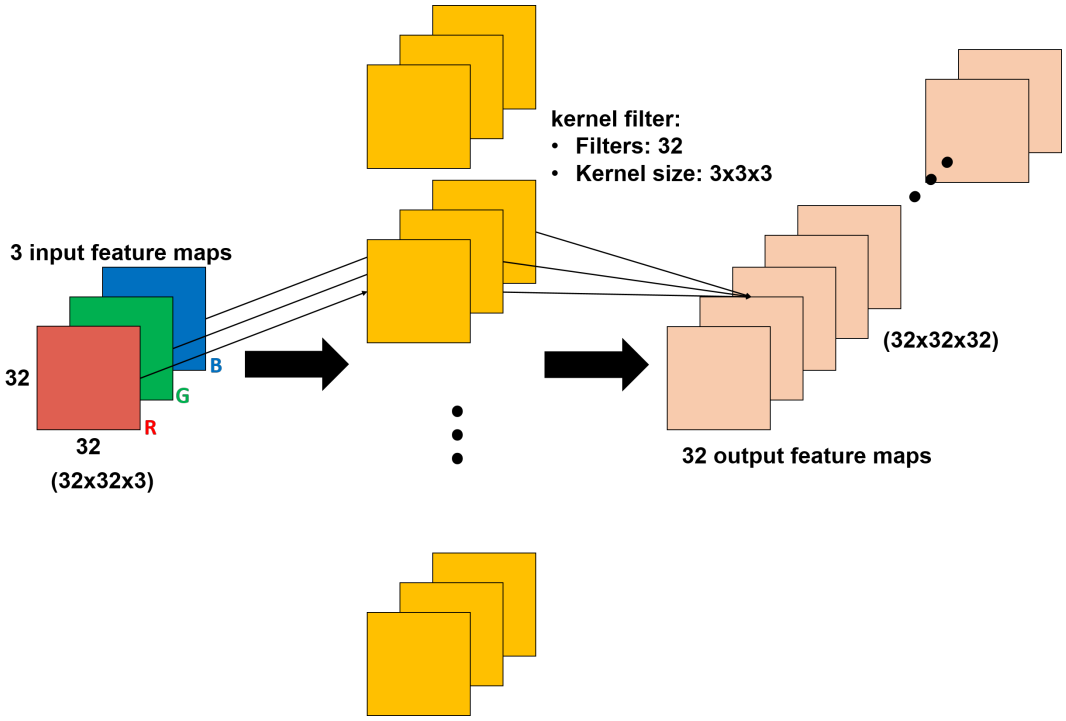

Figure 6.5: Convolutional layer 1

The input map to the first convolutional layer is an image of size 32x32x3 (RGB channels). The convolutional layer comprises 32 kernels of size 3x3x3. After the convolu-

tion operation, we get 32 output maps of size 32x32. Each output map is known as a response map [30].

Regarding the parameters that are learned in the training process, these corresponding to each value that comprises the kernel. For example, a kernel of size 3x3 will have 9 parameters (weights) to learn, plus a bias. In total, we have to learn 10 parameters. Following this logic, the first convolutional layer has to learn (3x3x3x32) weights + 32 biases = 896 parameters.

- **Pooling layers:** The pooling layers have as function the reduction of the dimensionality of the convolved feature. This means to decrease the computational power required to process the data. We have two types [32]:
  - Max pooling: Returns the maximum value from the portion of the image covered by the kernel. We illustrate this operation in Figure 6.6.

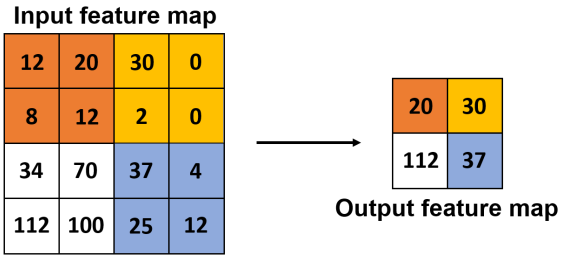

Figure 6.6: Max pooling operation

  - Average pooling: Returns the average of all the values from the portion of the image covered by the kernel. We illustrate this operation in Figure 6.7.

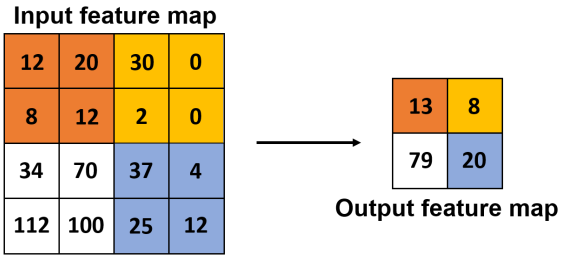

Figure 6.7: Average pooling operation

Because max pooling, besides discarding noisy activation completely, performs denoising along with dimensionality reduction, it is much better than average pooling, which only performs dimensionality reduction as a noise suppression mechanism. Therefore, max pooling layers are widely used [32].

The proposed architecture has two max pooling layers: Max. Pool 1 and Max. Pool 2. We can compute the size of the output feature maps using the formulas found in

[35]. For example, at the output of the max pooling layer 1 (Max. Pool 1), we will get 32 output maps of size 15x15. It is worth mentioning that there are no parameters to learn in these layers.

The goal of the joint work between the convolutional layers and the max pooling layers is looking for low-levels details as the image passes through the network. Therefore, after going through by both layers, we have enabled the model to understand features. To achieve this, we have as the last layer a fully connected layer for classification purposes [32].

- **Fully connected (FC) layers:** Unlike the convolutional layers, these layers have full connections to all activations in the previous layer, being the better way of learning non-linear combinations of the high-level features that turn out as outputs of the combination of the convolutional and pooling layers. However, before using these layers, we have to flatten the output of the convolutional and pooling layers into a column vector such that this flattened output will be the feed to a feedforward neural network. The last FC layer (the output layer) in the architecture, for a classification task, uses the Softmax activation as the activation function [32].

  The proposed architecture has two fully connected layers: FC layer 1 and FC layer 2. The number of neurons in the FC layer 2 is related to the classes in the dataset, in this case 2: nought or cross.

  Like the convolutional layers, these layers contain parameters that the network must learn in the training process. For example, FC layer 1 has 512 neurons. After flattening the output of the Max. Pool 2 layer, we have a column vector of size 2304. Therefore, the FC layer 1 has to learn (512x2304) weights + 512 biases = 1180160 parameters.

It is worth mentioning that the proposed architecture is based on an architecture already explored in the master sessions of the Machine Learning in Computer Vision course, and which is the common type of CNN that comprises many convolutional layers preceding subsampling (pooling) layers, while the ending layers are FC layers [36]. Therefore, we will not focus on optimizing the architecture or the principal training parameters. Finally, we summarise the CNN model architecture below:

- Convolutional layer 1: 32 filters, kernel size = 3x3, padding = 1.
- Convolutional layer 2: 32 filters, kernel size = 3x3.
- Max pooling layer 1: 2x2 windows.
- Dropout layer 1: 0.25
- Convolutional layer 3: 64 filters, kernel size = 3x3, padding = 1.

- Convolutional layer 4: 64 filters, kernel size = 3x3.
- Max pooling layer 2: 2x2 windows.
- Dropout layer 2: 0.25.
- Fully connected layer: 512 neurons.
- Dropout layer 3: 0.5.
- Output layer: 2 neurons, activation function: Softmax.

Regarding the dropout layers, these layers, during the training process, randomly removed some fraction of units from the network (or, equivalently, sets input units to 0). It is a regularization technique that helps prevent overfitting [37][14]. We apply these layers at the output of both max pooling layers, such that in the training process we randomly deactivate the 25% of the processing units (neurons) in the network. We use again the dropout layer at the output of the FC layer 1.
Having already detailed what concerns the architecture of the proposed model, we now focus on the training process. The training process, whether for the CNNs or the FNNs, is based on the gradient computation of a cost (or loss) function by using a propagation algorithm, and the updating of the parameters (weights and biases) of the network by using a gradient descent algorithm (gradient-based optimizer).
The goal is to find the optimal weights and biases that minimize the cost function, or the error between the predicted and the actual output. Therefore, carrying out model training involves the choice of both the cost function and the optimization algorithm. It is worth mentioning that the choice of cost function is closely related to the form of the output units, i.e. the type of task that the network will perform: regression, classification, etc. Another choice is the activation functions of the hidden layers. Finally, to measure the performance level of the model, we also need to choose the error metric, which also depends on the type of task that the network will perform [31][38][39].
Based on the above, we define the principal training parameters below:

- Training process parameters:
  - Optimizer: Adam.
  - Loss function: Categorical cross-entropy.
  - Activation functions (hidden layers): ReLU.
  - Activation function (output layer): Softmax.
  - Batch size: 16.
  - Epochs: 5.
- Model evaluation:

– Metric evaluation: Accuracy.

Adam is an optimizer that implements the Adam algorithm. The Adam optimization is a stochastic gradient descent method that is based on an adaptive estimation of first-order and second-order moments. In a keras implementation, the first argument is the learning rate [40]. The learning rate is perhaps the most important hyperparameter. A hyperparameter is a parameter that is not learned during the training process. The learning rate controls the effective model capacity better than other hyperparameters [31]. For the training process, we will use a learning rate equal to 0.001.

Regarding the loss function, the cross-entropy is the function commonly used to measure the performance of CNN models, having as output a probability distribution. This function is widely used in multi-class classification problems [36].

Finally, other important parameters in a gradient-based optimizer are the epochs and the batch size. The epochs refer to the number of times that the entire training data has passed through the network [41]. The batch size refers to the number of training samples used in one iteration [42]. An iteration refers to the numbers of times that the parameters are updated, such that one iteration is one gradient update [43].

In Figure 6.8, to illustrate the training process regarding the iterations, batch size and epochs, we show the training cycle for one epoch.

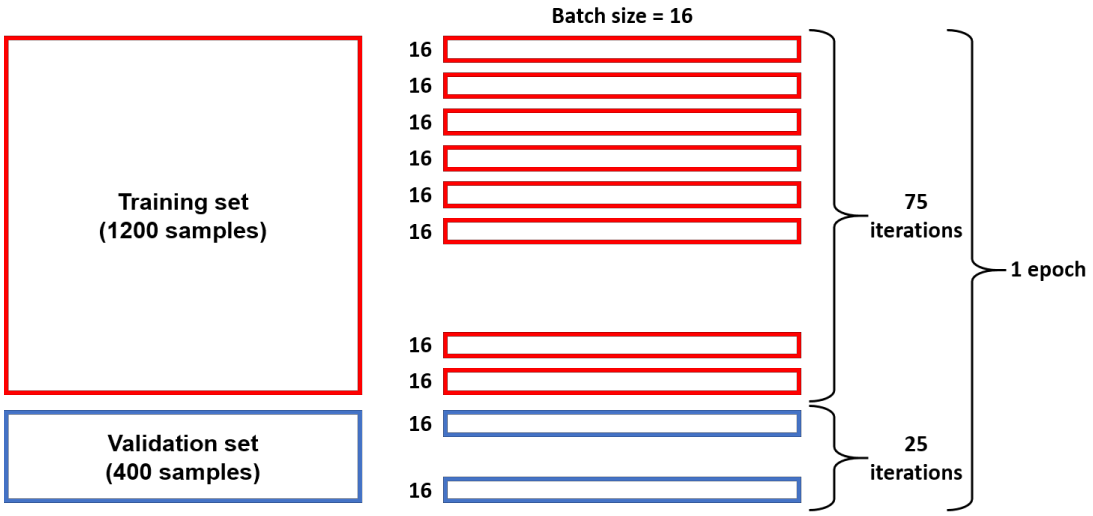

Figure 6.8: Batch size, steps and epochs

Based on the above, we can state that, in the training process, the parameters (weights and biases) will be updated 75 times in one epoch.

Now, having defined the network architecture and the training parameters, we create the model from the dataset. In Figure 6.9, we show the learning curves that point out the training and validation loss.

From the curves shown above, we can state that we do not have overfitting or underfitting

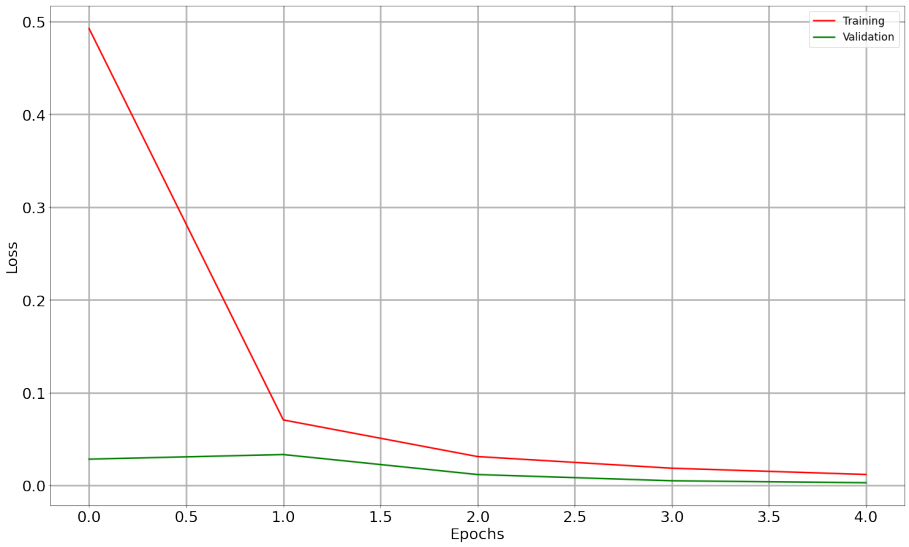

Figure 6.9: Batch size, steps and epochs

problems. Overfitting problems refers to when the model learns patterns from the input data but it cannot predict new outcomes for new input data: a too large gap between the training error and test error. Underfitting problems refers to when the model could not learn any pattern from the training data: a too large training error and test error [31][44].
Regarding the evaluation of the model performance, we will use the confusion matrix. The confusion matrix is a very popular table for the performance evaluation of classification problems, either for binary or multi-class classification [45]. In our case (binary classification), the confusion matrix will have the form shown in Figure 6.10.

		Predicted class	
		cross	nought
Predicted class	cross	True cross (TC)	False cross (FC)
	nought	False nought (FN)	True nought (TN)

Figure 6.10: Confusion matrix - Binary classification

Where TC refers to the observations which are predicted as a cross and are actually a cross. FC refers to the observations which are predicted as cross but are actually a nought. TN refers to the observations which are predicted as a nought and are actually a nought. FN

refers to the observations which are predicted as a nought but are actually a cross [46]. The observations belong to the test dataset. In our case, we have a test dataset comprising 400 images.

Based on the confusion matrix, we can compute the model accuracy using the formula below [47][45]:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

The accuracy represents the number of observations that the model has correctly predicted. In worth mentioning that exists other metric such as the precision, sensitivity, specificity or F1 score [46].

Finally, in Figure 6.11, we show the confusion matrix after evaluating the model created.

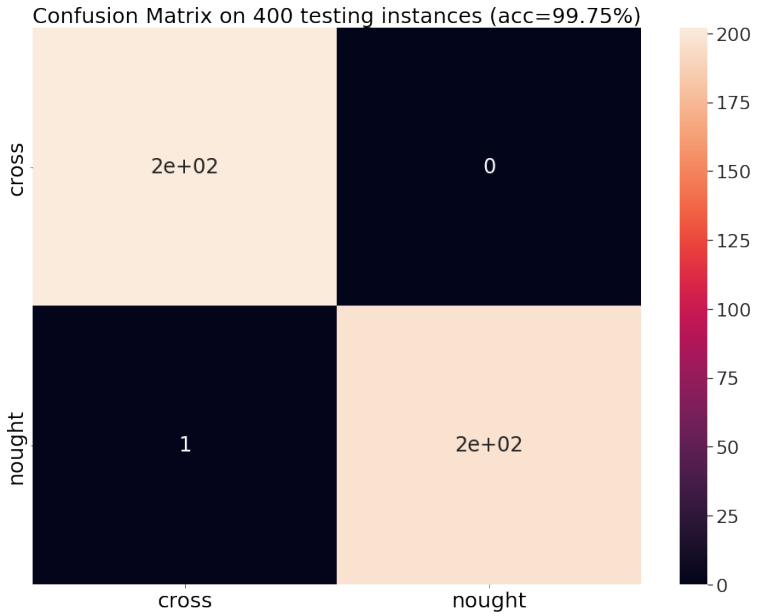

Figure 6.11: Confusion matrix

As we can see in the resultant confusion matrix, we have a CNN model with an accuracy of 99.75%. Last, we show in Figure 6.12 some results after using the model and predict the marker type over the test dataset.

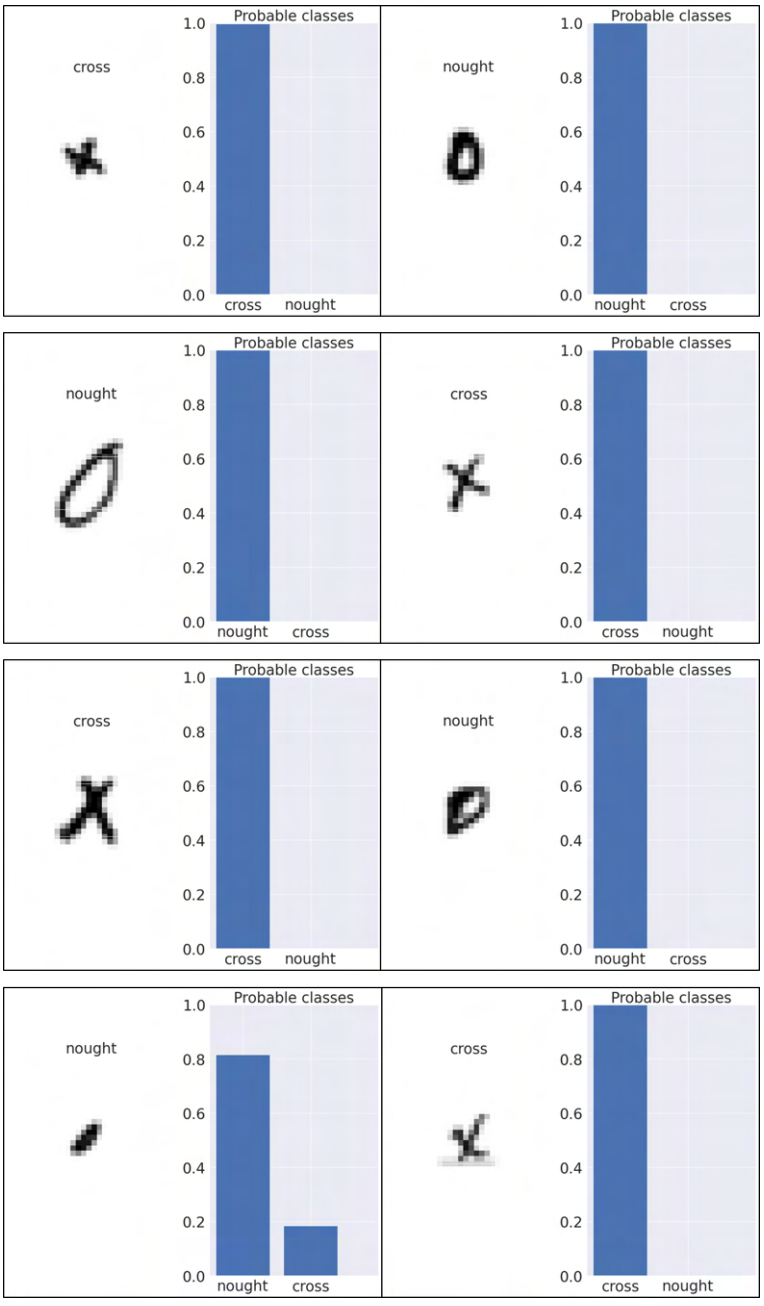

Figure 6.12: CNN model results after predicting the marker type over some test images

6.3 Highlighted points

- The high accuracy of the model attracts a lot of attention since it is almost 100%. However, it is also clear that we do not have an overfitting problem (Figure 6.9). Therefore, the cause of this phenomenon is not related to the architecture or training parameters. What remains to be analyzed is the dataset, probably causing this result. We first show some of the training data in Figures 6.13 and 6.14.

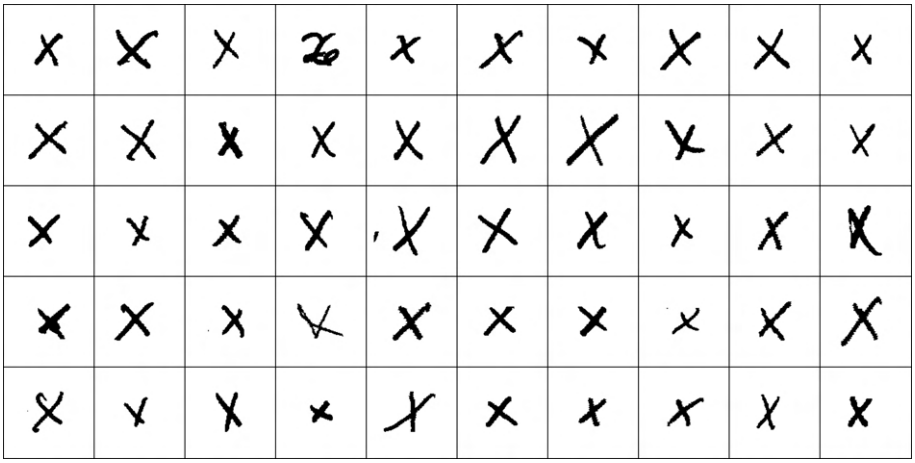

Figure 6.13: Cross dataset

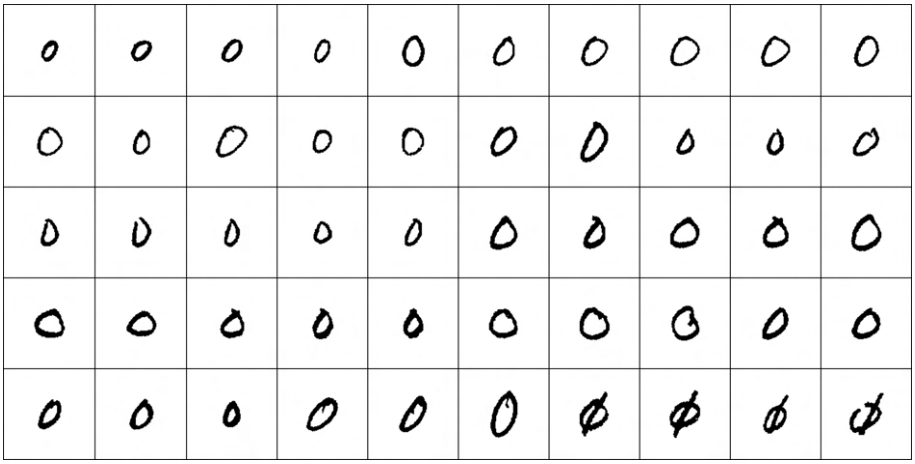

Figure 6.14: Nought dataset

As we can see, from a design point of view, the data set has little variety, since we can see that all the samples consider a nought or cross design centered on the image and with no considerable transformation such as rotation, scaling, etc.

However, this detail makes sense when remembering what is the original purpose of the data. The Special Database 19 is intended for character recognition tasks.

Therefore, the data has inherent semantics, so we probably wouldn't find a flipped "h" or "m", because it would not have a semantic sense.

Instead, the purpose we are looking for is different. We are using the characters "o" and "x" to represent the noughts and crosses, but for us there is only a meaning of correspondence, therefore both markers could be designed without following some type of design consideration.

Based on the above, we can state that the results got in relation to accuracy are because the CNN model is facing a rather simple classification task. Since, not only does the data have a uniformity of design, but it is also only a matter of classifying two classes. However, because of this, when carrying out the tests in the experimental module, we must maintain the design criteria of the noughts and crosses. To further adapt the model to our specific task, we can use augmentation techniques, and in this way, to have images where the noughts and crosses present transformations such as rotation, scaling, etc.

- Despite the great potential that CCNs have for extracting features automatically, the major problem one faces when trying to use this type of network, as with other deep leaning model, is the dataset. Deep learning models are extremely data-hungry, demanding extensive amounts of data for an optimal performance. Just to give a flavour, the CIFAR-10 dataset comprises 60000 images, ImageNet database comprises 14197122 images, and the MNIST database comprises 70000 images.

Chapter 7

Camera-based positioning system

7.1 System description

Either using image processing or deep learning techniques, we can already abstract information from the images, but the resulting information belongs to the pixels space, i.e. it is relative to the camera reference frame. However, the actuating element, i.e. the UR3e robot, needs to receive this information but relative to his reference frame. Therefore, we need to find a way to referencing the resulting information of the camera to the robot reference frame. In this sense, the aim is to develop a system capable of computing the 2D coordinates of a point in the real world based on its 2D coordinates in an image. In this way, the information that we have already gained, e.g. the centroids of each square that comprises the grid, can be used by the robot.
Using the theory of the Pinhole camera model, we can describe a linear relationship between a point belonging to the real world (mm) and its corresponding coordinate in the images (pixels) by the following equation [48]:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = T \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{7.1}$$

Where $P = (X, Y, 1)$ is the homogeneous coordinates of the point in the world and $p = (u, v, 1)$ is the homogeneous coordinates of the point in the image. T is a 3x3 homogeneous transformation matrix that comprises scaling, rotation, and translation transformations and has the form below:

$$T = \begin{bmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ 0 & 0 & 1 \end{bmatrix} \tag{7.2}$$

Mathematically, the objective is to compute this transformation matrix, which will allow us to get the coordinates of a point in the world from its corresponding coordinates in the image.

Replacing (7.2) in equation (7.1), and given corresponding coordinates (X_i, Y_i, u_i, v_i), we can write the following transformation equation:

$$\begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} = \begin{bmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

Rearranging the equation above in the form $Az = b$, where z contains our unknowns $(t_{00}, ...t_{12})$, we get the equation below:

$$\begin{bmatrix} u_i & v_i & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_i & v_i & 1 \end{bmatrix} = \begin{bmatrix} t_{00} \\ t_{01} \\ t_{02} \\ t_{10} \\ t_{11} \\ t_{12} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \tag{7.3}$$

From the equation (7.3), we can observe that a correspondence of coordinates offers two linear equations with six unknowns. Therefore, to solve the linear system of equations, we need at least three correspondences. However, it is better to have over three correspondences and solve an overdetermined system with a least-squares method.
Although we can use the equation (7.3) to meet our objective. The equation is based on the general pinhole camera model, where linearity is preserved. Therefore, the equation does not consider the effects of the non-linear distortion on the images caused by the camera lens [49]. The lens distortion is a deviation of the ideal projection considered in the pinhole camera model, in which straight lines in the real world are no longer straight in the image [50]. We can observe some examples of lens distortion in Figure 7.1.

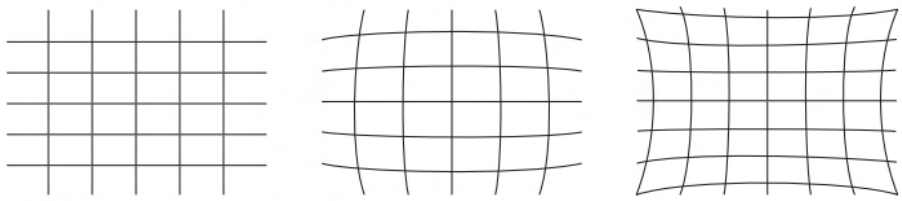

Figure 7.1: Examples of lens distortions: (left) Ideal rectangular grid, (centre) barrel distortion, (right) pincushion distortion [50]

To consider the lens distortion in our mathematical formulation, we can model it using the Brown-Conrady polynomial model, which is a mapping from distorted coordinates to rectified coordinates, or vice versa. We can find the lens distortion parameters of this model with distortion calibration methods. Finally, to remove the lens distortion, the simplest technique is to use directly the model for all the pixels in the output undistorted image [49][50].

To summarise the above, considering the camera model and the distortion model of the lens, we can achieve our objective by following the steps below:

1. Acquisition of a certain number of corresponding coordinates, i.e. the coordinates of a point in the world (mm) and its corresponding coordinates in the image (pixels).
2. After a distortion calibration, use of the lens distortion model to compute the rectified pixel coordinates in the image.
3. Use of the rectified pixel coordinates to solve the equation 7.3 by the least-squares method, and to get the homogeneous matrix T.

In this context, we propose an alternative approach that replaces the steps 2 and 3 with the design and creation of a deep learning model based on feedforward neural networks. Regarding the first step, we will develop a subsystem for the acquisition of the corresponding coordinates. As in previous systems, we will first make use of images got under simulation to develop the algorithms involved in the system's development, and then we will use actual images to adjust the first approach to the algorithms and adapt them to actual situations.

7.2 Implementation description

7.2.1 Data acquisition process

The first objective is to develop a subsystem for the data acquisition. For that, we propose the steps below:

1. Equip the robot with a dot laser, which we will install in the end-effector.
2. Command a position to the robot, so that it moves the projection of the dot laser onto the surface of the whiteboard.
3. Capture an image containing the dot laser projected on the surface of the whiteboard, defined from now on as the laser point.
4. Get the coordinates of the laser point in the image.
5. Save in a database, both the position commanded to the robot and the coordinates of the laser point.
6. Repeat steps from 2 to 5 according to the amount of data we consider appropriate.

As mentioned in Section 7.1, first we will use raw data got by simulation. This data includes the position commanded to the robot (step 2), and an image containing to the laser point (step 3). Therefore, following the sequence of steps, the next step, and the first objective, is to compute the coordinates of the laser point (step 4).
We start by showing in Figure 7.2 the input image.

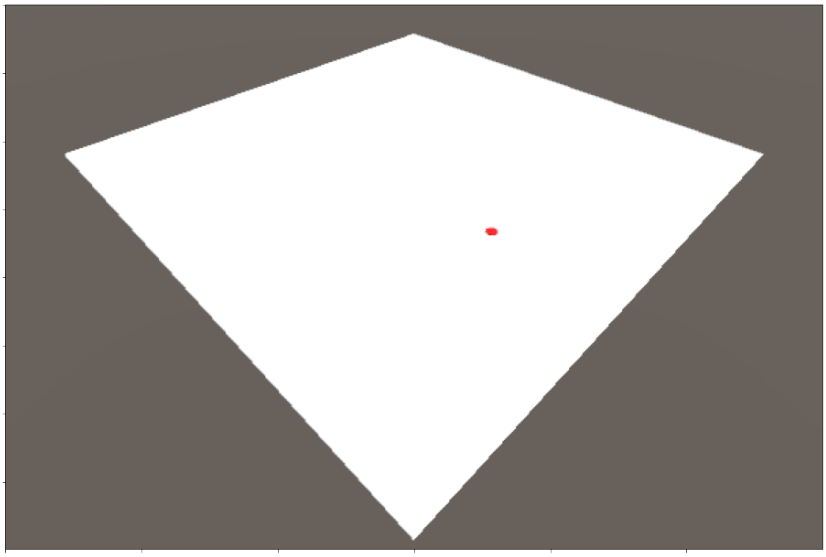

Figure 7.2: Input image

According the first objective, we are facing a single-object localization task which is one of the most common tasks of computer vision [51]. First, we start by changing the color space of the image from RGB to HSV space. The color space is a mathematical model to represent the color information in function as color components. In this way, the HSV color space is the most suitable space to describe the human visual experience being closer to the human's color perception compared with the RGB color space [52]. We show the results after applying the color space changing to the input image in Figure 7.3.

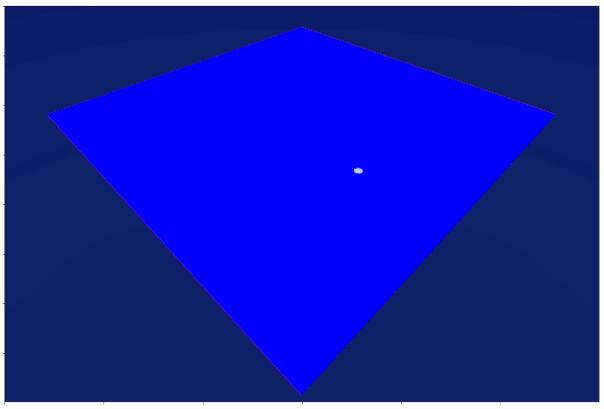

Figure 7.3: Input image in HSV color space

Likewise, in Figure 7.4, we show its three components: Hue channel, Saturation channel and Value channel. The Hue channel (H channel) indicates the color type, e.g. red, blue, yellow, etc. The Saturation channel (S channel) indicates how pure the color is. The Value channel (V channel) indicates how much brightness is a color [52].

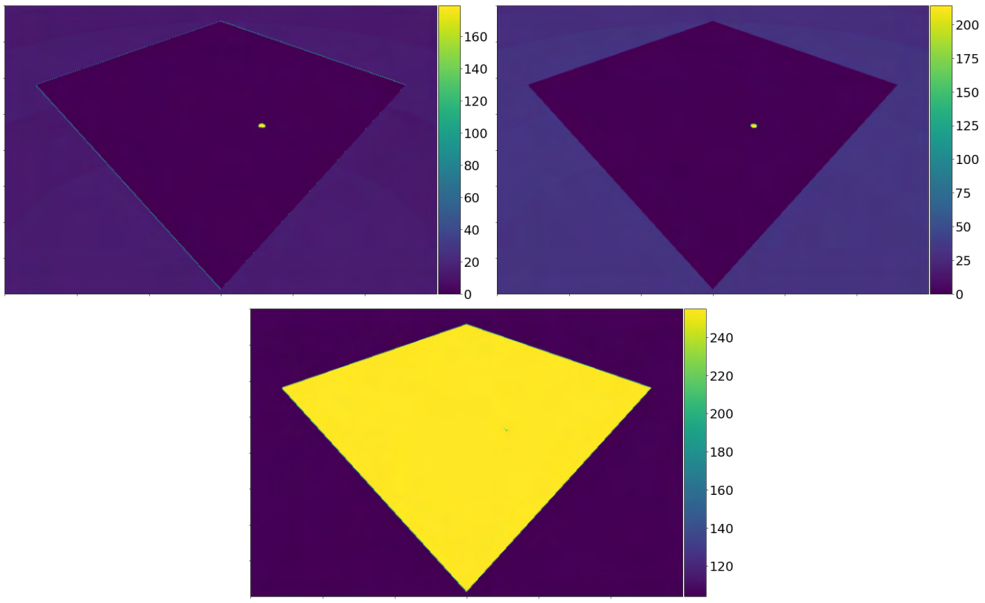

Figure 7.4: (left corner) H channel, (right corner) S channel, (bottom) V channel

From the results shown in Figure 7.4, we can state that it is possible to analyze either the H channel or the S channel to compute the coordinates of the laser point in the image. We decide to analyze the channel that indicates the color type: the Hue channel.
We propose a thresholding operation to detect the laser point. From this operation, we will have a binary image where the white pixels represent the the laser point. Finally, we apply to the resultant binary image a segmentation operation in such a way that we can use the properties of the labelled region that contain the laser point. In this case, we are interesting in computing the coordinates of the centroid of the region.
We start with the thresholding operation. First, we plot the histogram of the pixel distribution in the H channel of the input image. We show this histogram in Figure 7.5.

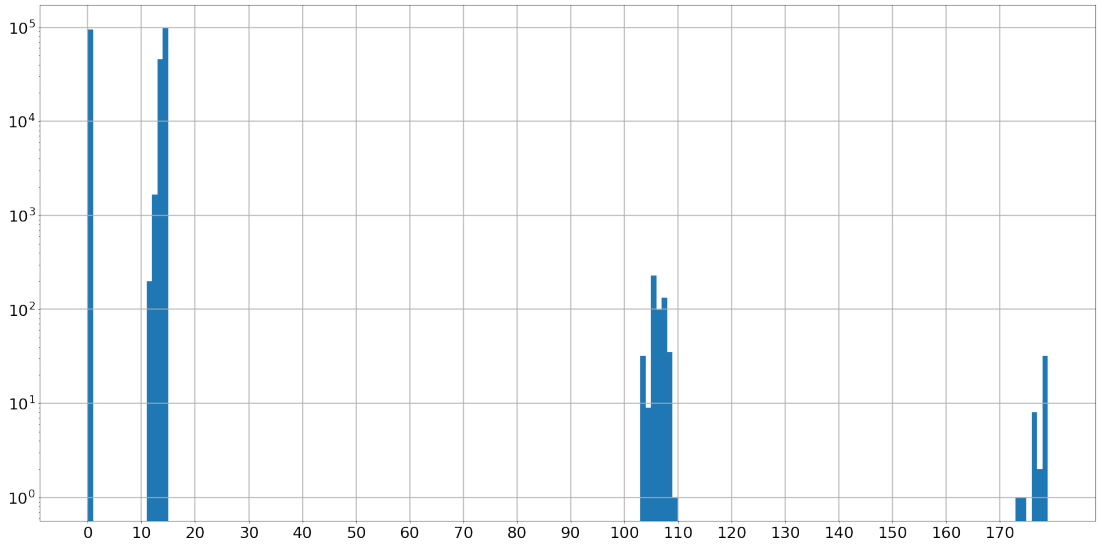

Figure 7.5: Histogram of the pixel distribution in the H channel

Based on the histogram shown in Figure 7.5, we decide to carry out a simple thresholding operation varying the thresholding range according to the pixel distribution. In this way, in Figure 7.6, we show the results after applying a thresholding operation using the ranges: $[0 - 100]$, $[100 - 120]$ and $[120 - 180]$. The green area is the region that represents to the pixels belonging to each thresholding range.

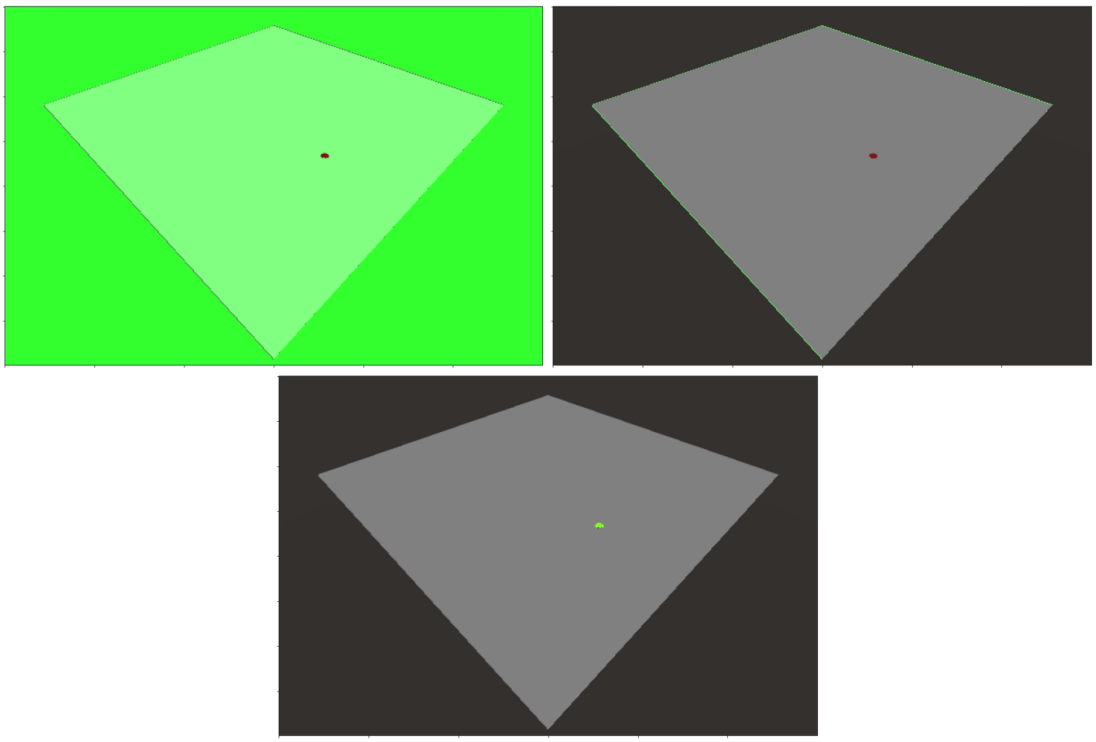

Figure 7.6: Thresholding operation: (left corner) thresholding range: $[0-100]$, (right corner) thresholding range: $[100-120]$, (bottom) thresholding range: $[120-180]$

Based on the results above, we can clearly state that we can detect the laser point just by using the thresholding range: [120-180]. In Figure 7.7, we show the resultant binary image after applying the corresponding thresholding operation.

Figure 7.7: Laser point detection

Then, we apply a segmentation operation to the binary image above and compute the coordinates of the centroid of the region containing to the laser point. In Figure 7.8, we show the results after applying the first approach to the laser point localization algorithm to other simulated images.

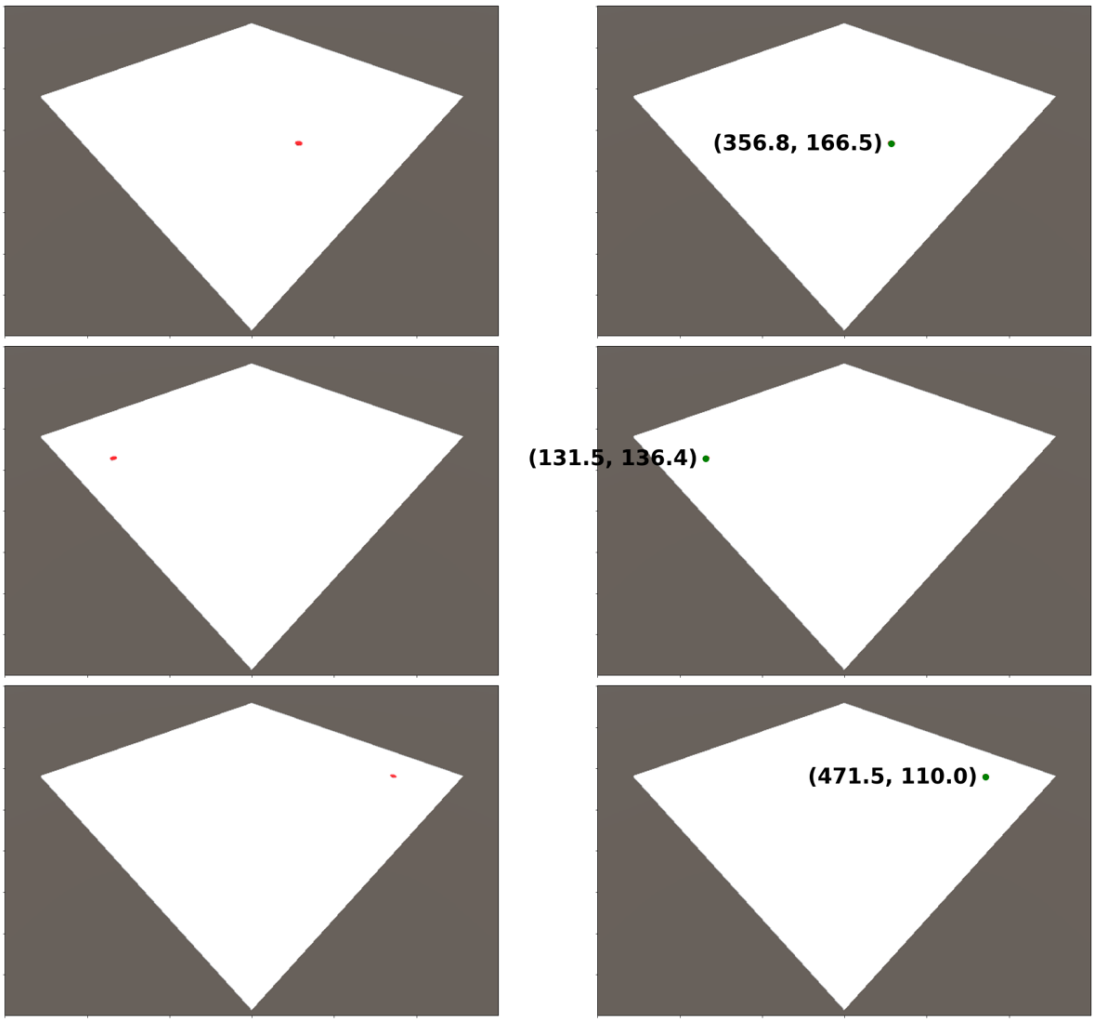

Figure 7.8: Results after applying the laser point localization algorithm

To fully accomplish our first objective, first we save the coordinates of the laser point with the position commanded to the robot (step 5), and finally we repeat the same procedure with the entire raw data got by simulation (step 6). We have a dataset comprising 1000 images and 1000 positions relative to the robot base. Therefore, at the end of the acquisition process using this raw data, we will have 1000 points of correspondence. We will use this dataset to fulfill subsequent objectives.

7.2.2 Feedforward neural network modelling

The second objective is to design and create a deep learning model based on feedforward neural networks from the data acquired in Section 7.2.1.
The feedforward neural networks are the quintessential deep learning models. The goal of these models is to approximate a function mapping $\boldsymbol{y^*} = f(\boldsymbol{x}, \boldsymbol{\theta})$ which maps input examples $\boldsymbol{x}$ to output examples $\boldsymbol{y}$. The $\boldsymbol{\theta}$ are the model parameters, comprising the weights and biases of the network. The model learns these parameters through a training process such that we drive $f(\boldsymbol{x})$ to match $f^*(\boldsymbol{x})$ [31].
As a deep learning model, like CNNs, its training process is based on the computation of the gradient of a cost function and the updating of the network parameters using a gradient descent algorithm. Therefore, as with CNNs, we need to properly choose a cost function, the optimizer, the activation functions for the hidden layers and metrics to measure the performance of the model. Just like CNNs, another important aspect is the model architecture, in this case, how many layers and neurons by layer will have the network.
Based on the above requirements for the model design, we define the architecture and the principal training parameters below.

- Model architecture:
  - Two inputs: which refer to the pixel coordinates of the laser point.
  - Two outputs: which refer to the coordinates of the laser point regarding the base of the robot.
  - Three hidden layer: 64, 128, 64 neurons respectively.

    Regarding the hidden layers, their design is an active area of research. Determining a final design requires a lot of iteration and optimization [31][53]. In this sense, the thesis does not focus on this area, so the number of layers in the proposed model is based on empirical experience.
- Training process parameters:
  - Optimizer: Adam.
  - Loss function: Mean Squared Error (MSE).
  - Activation functions: ReLU.
- Model evaluation:
  - Metric evaluation: Absolute Mean Error (MAE).
  - Evaluation technique: k-fold cross-validation.

The k-fold cross-validation is a cross-validation technique in which we use all the data to evaluate the model, being very useful when the dataset is small. The k-fold split the dataset into k non-overlapping subsets in such a way that we can estimate the test error by taking the mean of the test error over k trials. In the trial i, we use the i-th subset as the test set and we use the remaining data as the training set. In this way, this technique allows us to create different models from the available data and keep the best one [31]. We will use this evaluation technique because we focus on creating a model using as little data as possible. Regarding the learning rate, we will perform a grid search that comprises choosing a small finite set of values and analyze the behavior of the training process after training a model for every finite value. Then, we choose the value corresponding to the best model performance [31]. In this way, in Figures 7.9 and 7.10, we show the learning training and validation curves, respectively.

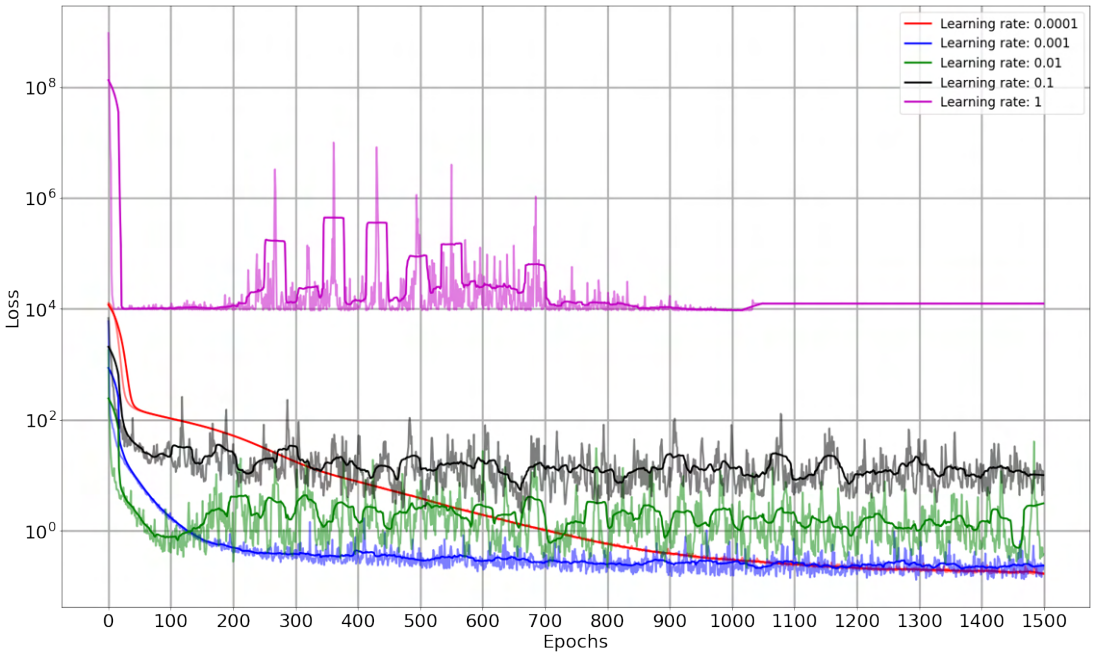

Figure 7.9: Train learning curve

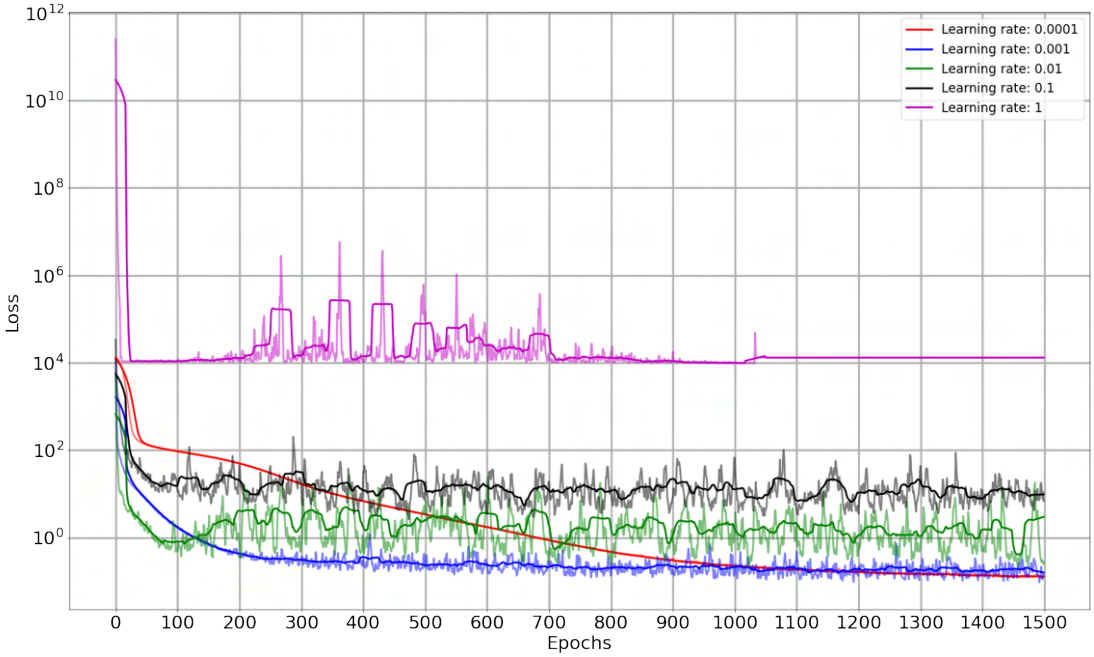

Figure 7.10: Validation learning curve

From the results got in both learning curves, we conclude that the best value for the leaning rate for our task is 0.001 (blue line). Regarding the number of epochs, as we can observe in the learning curves, we have carried out the training of the model using a 1500 epochs by default.

Thus, just like the learning rate, we will carry out a grid search to find the optimal number of epochs. In Figure 7.11, we show a graph whose x-axis is the number of epochs and the y-axis is the average MAE, i.e. the average of the absolute mean error in the x-coordinates and y-coordinates. Based on the graph, we conclude that the best value for the number of epochs is 1000, since from this value, the average MAE stabilises.

Finally, with all the design parameters already defined, we create the model from the raw data got under simulation. In Figure 7.12, we show the learning curves that points out the training and validation loss. From the curves shown above, we can state that we do not have overfitting or underfitting problems.

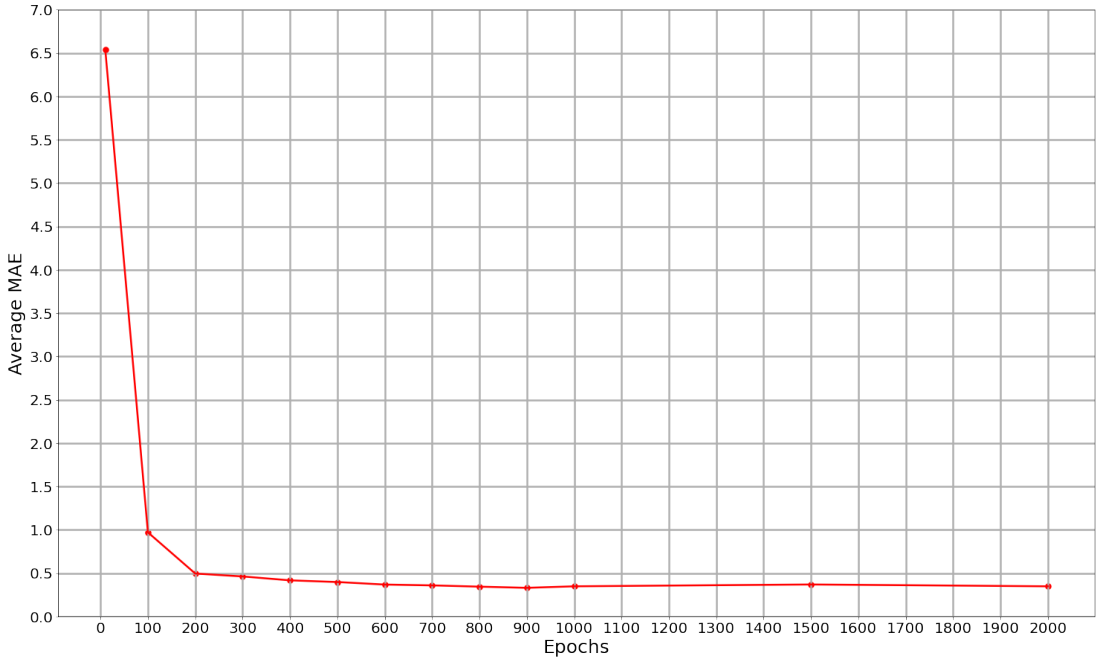

Figure 7.11: Epochs vs Average MAE

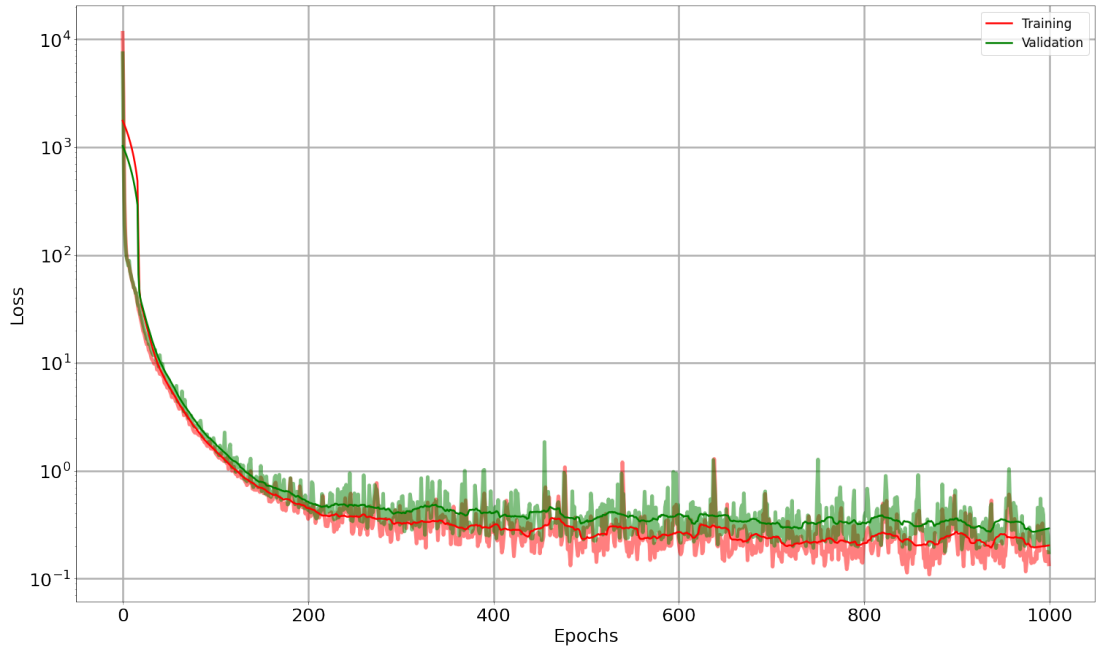

Figure 7.12: Learning curves

7.2.3 Positioning system evaluation

The positioning systems are used to determine a position coordinates, and we expect that the position errors are random and follow a normal distribution [54]. Based on this, we decide to evaluate the model performance, i.e. the positioning system performance, based on a statistical analysis of the measurement error, which will be the difference between the coordinates predicted using the model and the actual coordinates [55]. We will analyze the 99.7% confidence interval ($\pm 3\sigma$) of the distribution of the coordinate errors. A 99.7% confidence interval means there is a 99.7% chance that the coordinate error estimated lies within the bounds of this interval. The narrower is, the greater is precision in the estimate. The wider is, the less certainty there is in the estimate [56]. First, we check the normal distribution of the coordinate errors in both x-coordinate and y-coordinate. For that, we use the probability plot tool. In the probability plot, we compare the data to a theoretical distribution such that the closer the plot comes to a straight line, the more the data fits a normal distribution [57][58]. We show the probability plot for both coordinates in Figure 7.13.

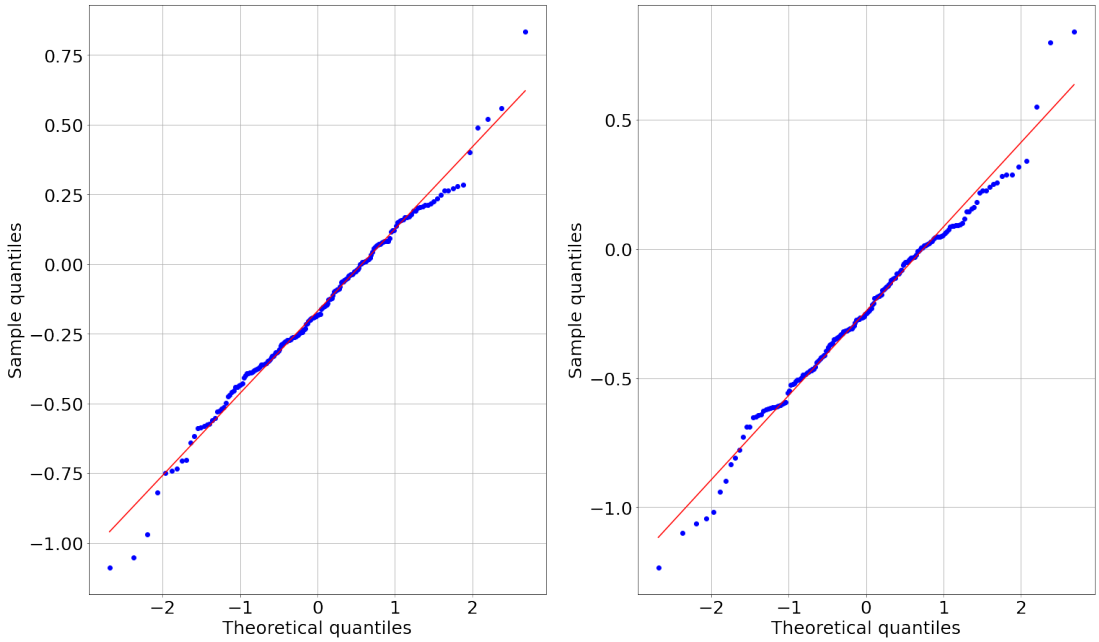

Figure 7.13: Probability plot: (left) measurement errors in x-coordinate, (right) measurement errors in y-coordinate

As we can see, we can state that the coordinate errors follow a normal distribution. Now, to analyze the 99.7% confidence interval, in Figures 7.14 and 7.15, we plot the box plot and the probability distribution for both coordinates.

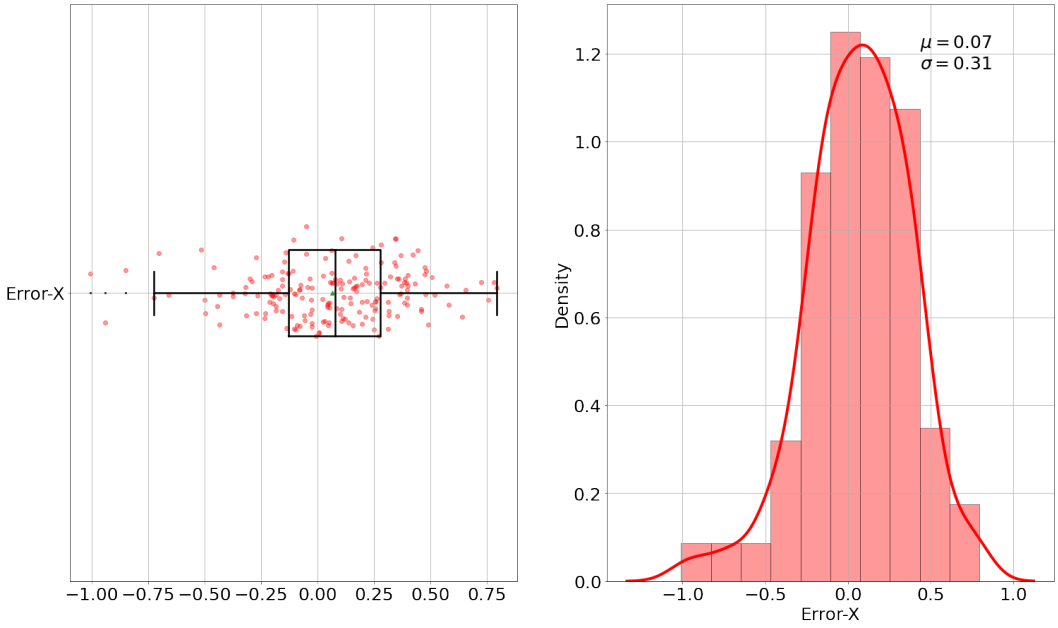

Figure 7.14: x-coordinate error: (left) Box plot, (right) probability distribution

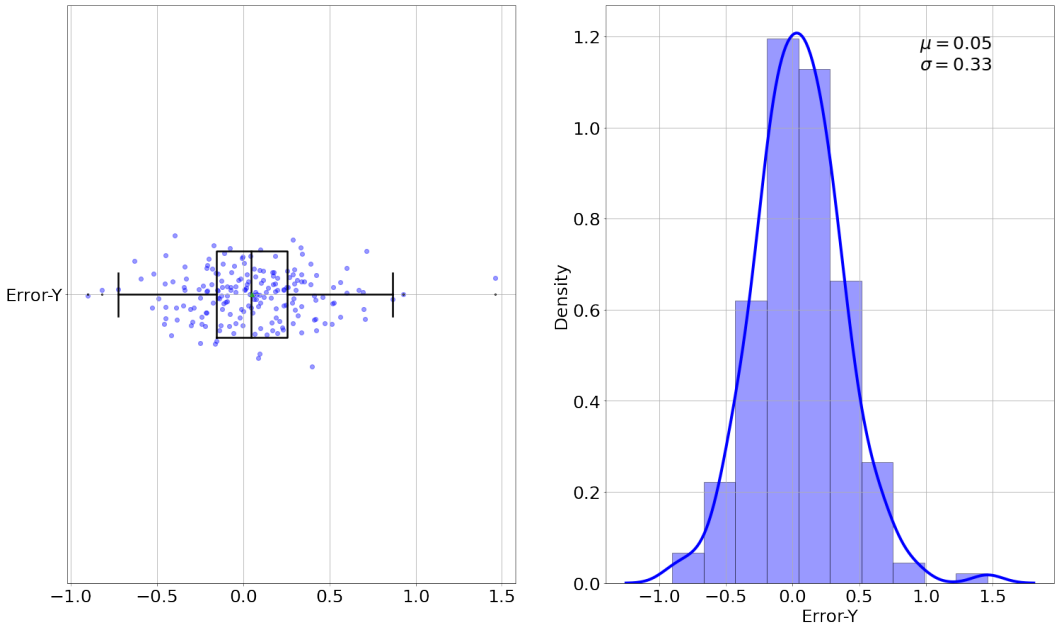

Figure 7.15: y-coordinate error: (left) Box plot, (right) probability distribution

According to the graphs, the coordinate errors distribution in the x-coordinate has a standard deviation of 0.31 mm, while in the y-coordinate has a standard deviation of 0.33 mm. Therefore, based on this statistical analysis, we state that the positioning system has a precision of $\pm 0.93 \approx \pm 1$ mm in the x-coordinate, and a precision of $\pm 0.99 \approx \pm 1$ mm in the y-coordinate.

7.2.4 Data size analysis

From the data acquisition process (Section 7.2.1), we know that getting a real sample involves the robot performing a movement action, so each sample acquisition takes a certain amount of time. In order to optimize this process, we will carry out an analysis of the amount of data needed for the model. For that, we randomly partition the dataset got at the end of the Section 7.2.1 into subdatasets ranging from 50 samples. For each subdataset, we will train and evaluate 10 different models according to the evaluation described in Section 7.2.3. Finally, we compute the average 99.7% confidence interval for each subdataset and plot them. In this way, at the end, we expect to get a plot in which, as the data size increases, the average 99.7% confidence interval decreases. In Figures 7.16 and 7.17, we show this graph for each coordinate.

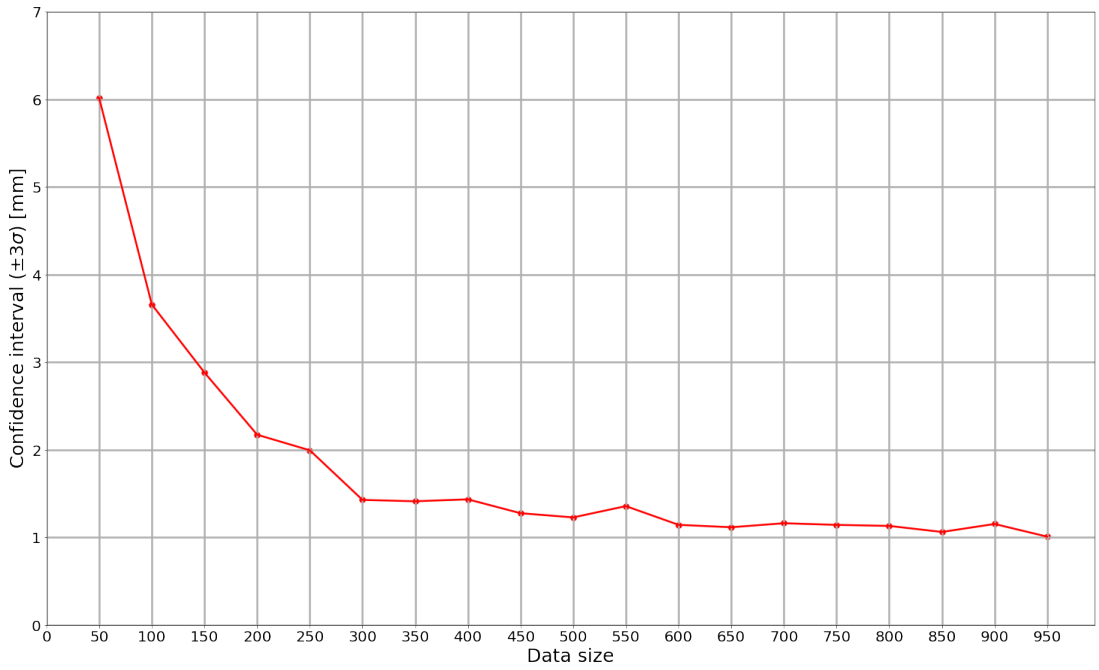

Figure 7.16: Data size analysis in x-coordinate

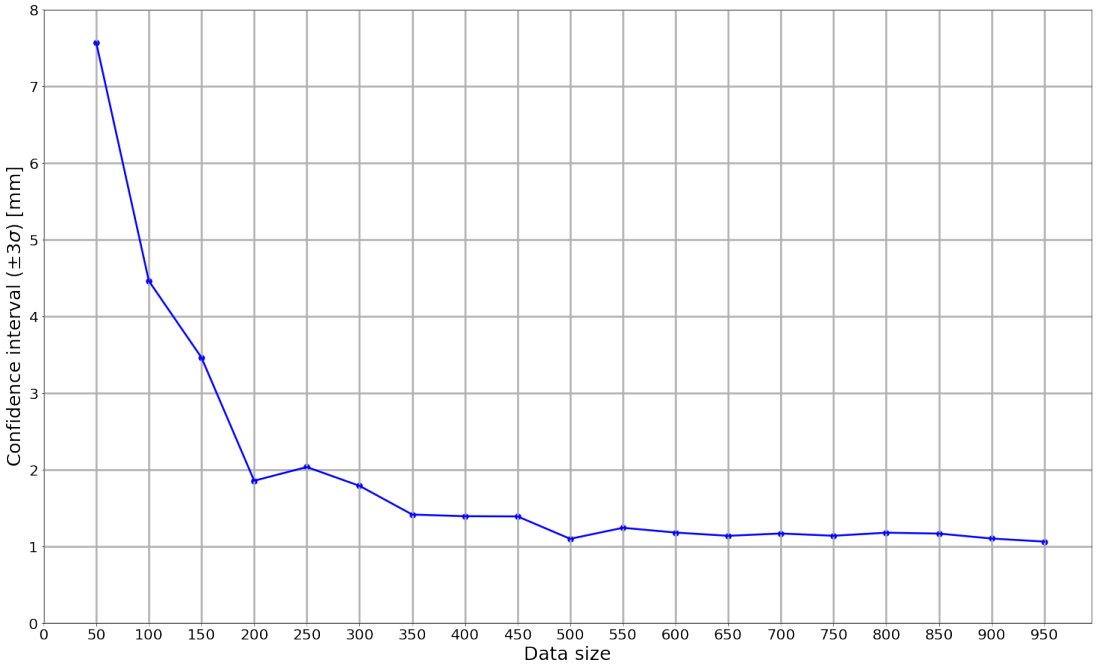

Figure 7.17: Data size analysis in y-coordinate

In order to further explore the performance of the model with a smaller amount of data, we will do the same procedure as explained above, but this time we will partition the dataset into subsets of data ranging from 10 to 100 samples. In Figures 7.18 and 7.19, we show the results for each coordinate.

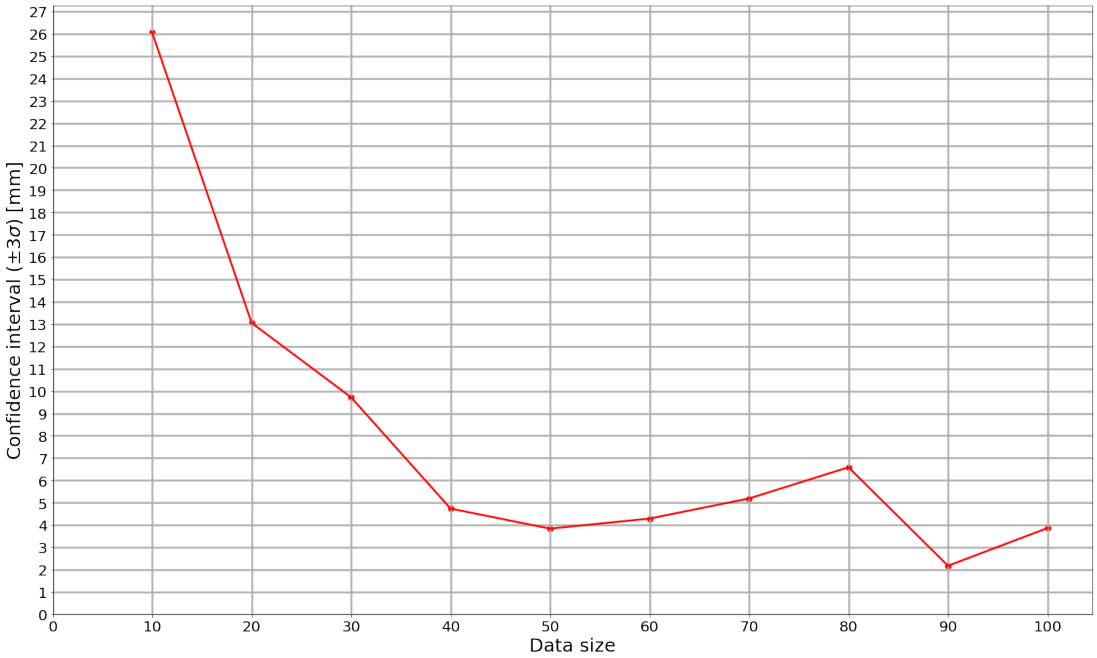

Figure 7.18: Data size analysis in x-coordinate (10-100 samples)

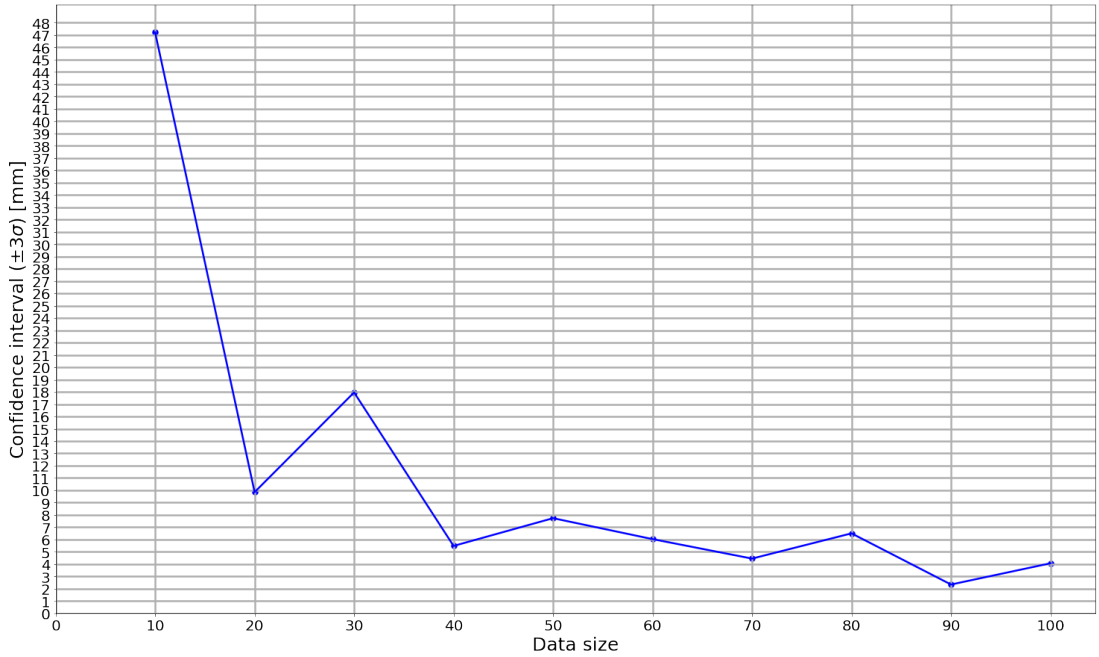

Figure 7.19: Data size analysis in y-coordinate (10-100 samples)

From the graphs above, making a compromise between the data size and an acceptable precision, we arrived at a trade-off of 300 samples to get an approximate accuracy in the x-coordinate of 2 mm, and in the y-coordinate of 2 mm.

7.2.5 Repeatability analysis

Repeatable, replicable and reproducible methods and results are very important for scientific research. In the deep learning field, the repeatability of deep learning models is important and fundamental in order to have replicable and reproducible models. Repeatability is the ability to re-train the models and getting the same results under the same training conditions on multiple trials. However, we normally assume the repeatability of deep learning models to hold. Therefore, it is a good practice to check the model repeatability prior to deploy or publish a deep learning model, since if the model is not repeatable, then the model is not replicable nor reproducible [59][60].

Therefore, in this section, we will perform a repeatability analysis comprising training and evaluating 100 different models under the same training and evaluation parameters. For each model, out of the 1000 initial samples got in Section 7.2.1, we choose 300 samples randomly for each training, data size set in section 7.2.4. In Figures 7.20 and 7.21, we show the resulting graphs.

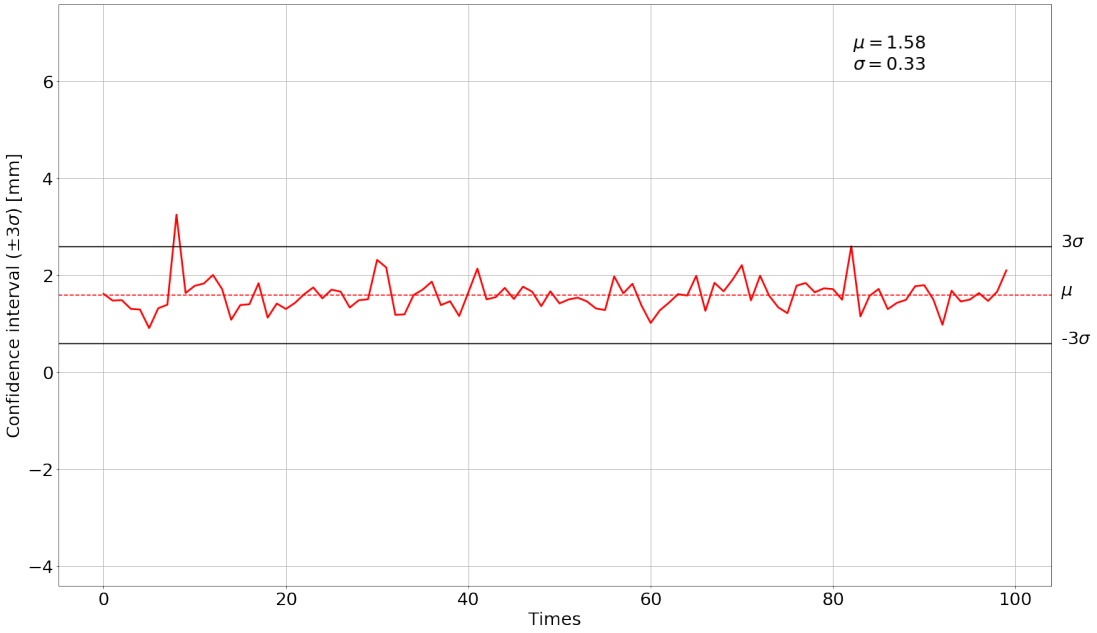

Figure 7.20: Repeatability analysis-x-coordinate

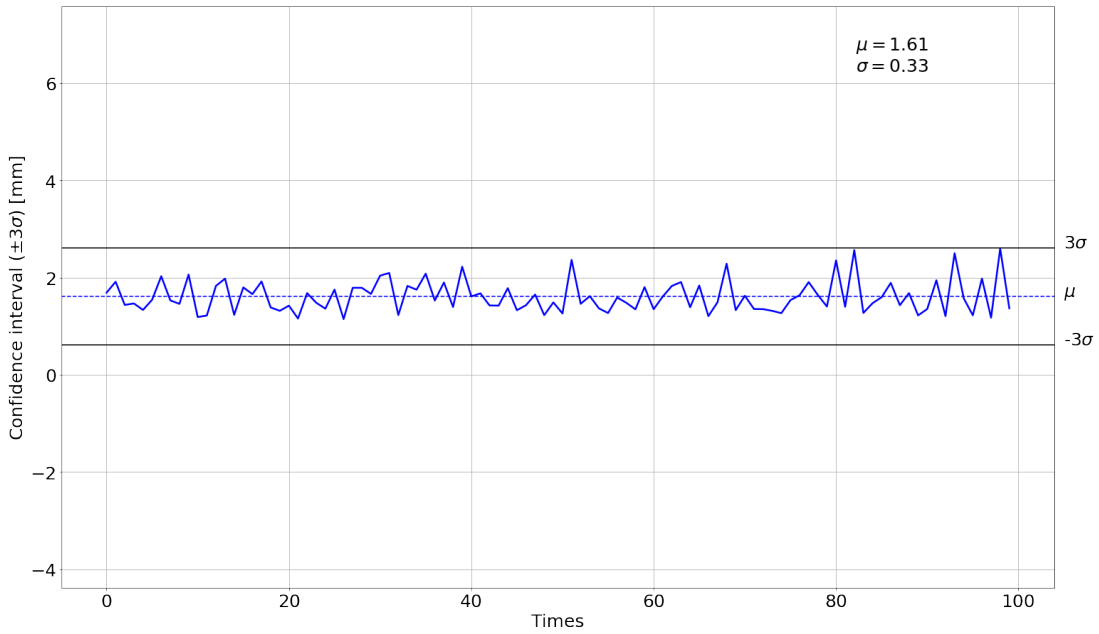

Figure 7.21: Repeatability analysis-y-coordinate

From the graphs, we can state that regardless of the calibration process of the system, i.e. the acquisition of the data followed by the creation of the model, we will have a positioning system whose precision in the x-coordinate will be within the range: $[0.59, 2.57]$ mm, while in the y-coordinate it will be within the range: $[0.62, 2.60]$ mm.

7.3 Real implementation

As mentioned in the system description (Section 7.1), the raw data used comes from a simulation. Being, the images of the laser point a fundamental piece of the development of the positioning system, since they are the input data for the creation of the model. Therefore, the real implementation of the positioning system is only subject to the correct computation of the laser point coordinates in the actual images acquired in the experimental module. For the real implementation and based on the analysis in Section 7.2.4, we have acquired 300 images samples.
Therefore, in order to adapt the localization algorithm to actual images, we will apply the algorithm to some of the acquired images. The aim is to adjust the parameters or strategies of the algorithm to meet the objectives.

7.3.1 Laser point localization algorithm improvement

We start by showing in Figure 7.22 a real input image.

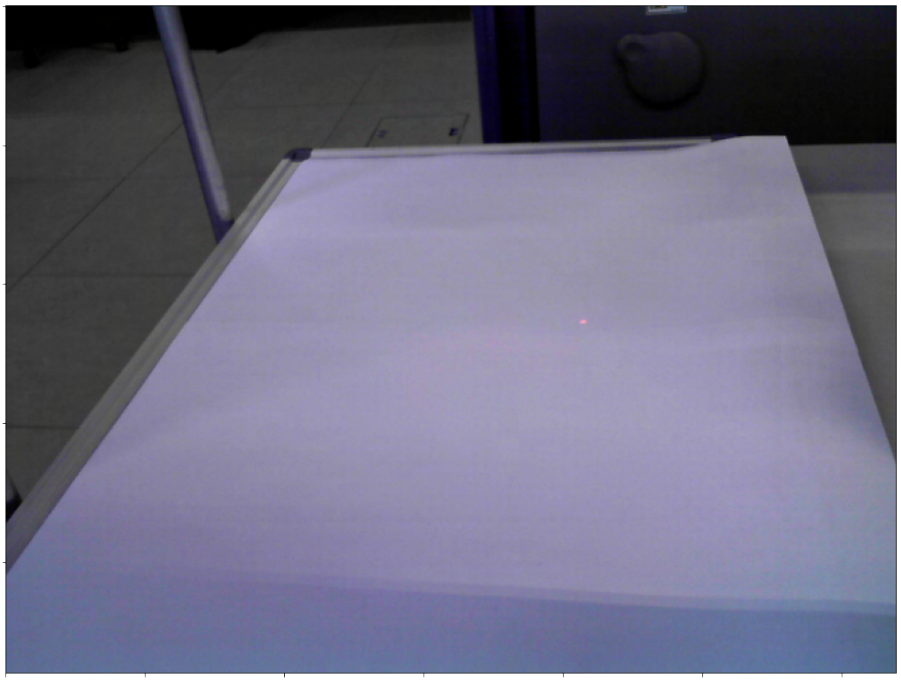

Figure 7.22: Input real image

According to the algorithm proposed in Section 7.2.1, the first step is to change the colour space of the image from RGB to HSV. Thus, just like with the simulated images, in Figure 7.23, we show the input image in the HSV colour space, and in Figure 7.24, we show its three components.

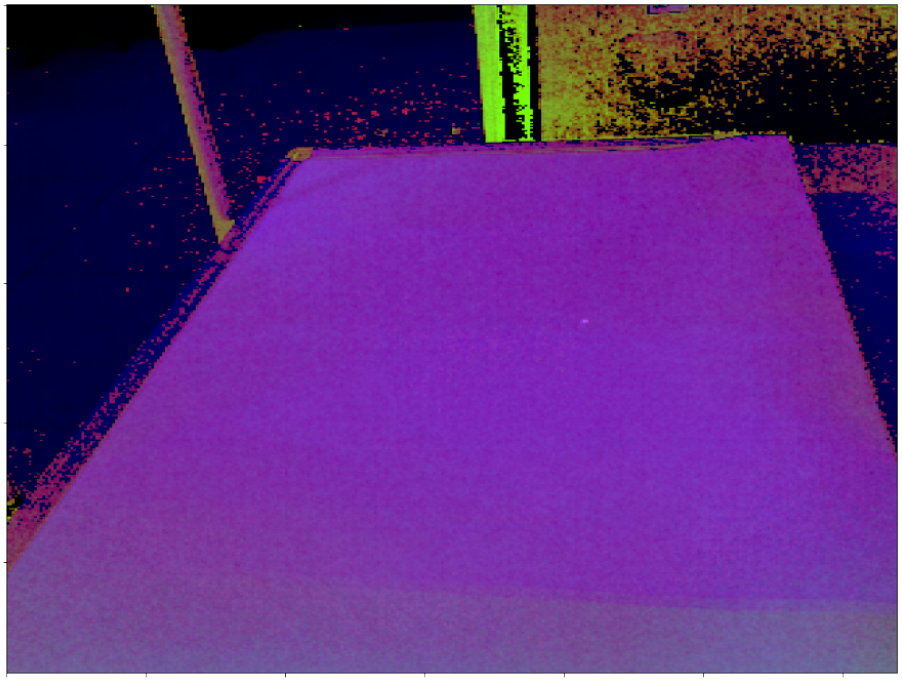

Figure 7.23: Input real image in HSV color space

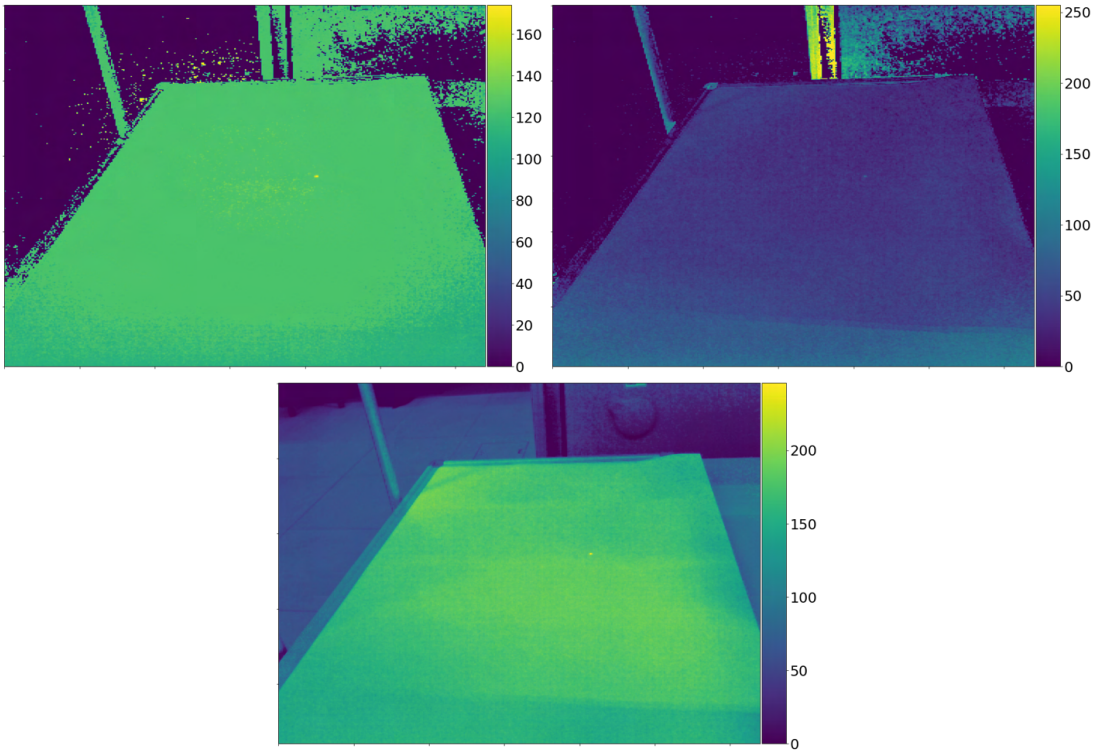

Figure 7.24: (left corner) H channel, (right corner) S channel, (bottom) V channel

From the results shown in Figure 7.24, this time we note we can analyze the H channel and the V channel to compute the point laser coordinates. This makes sense since the laser point, besides being red, has a high brightness.
Following the algorithm, the first objective is to detect the laser point in the image using a thresholding operation. We first plot the histogram of the pixel distribution in the H-channel and the V-channel, but this time we calculate the histograms considering all the 300 images. With this we are taking into consideration that, unlike the simulated images, each image is subject to variations in its pixel distribution. In Figures 7.25 and 7.26, we show both histograms.

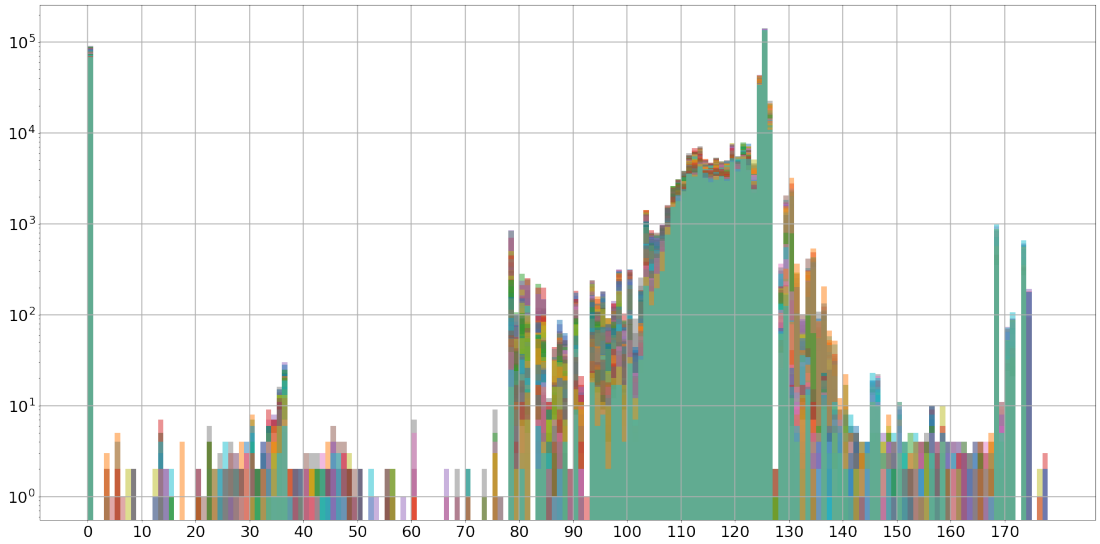

Figure 7.25: Histogram of the pixel distribution in the H channel - 300 images

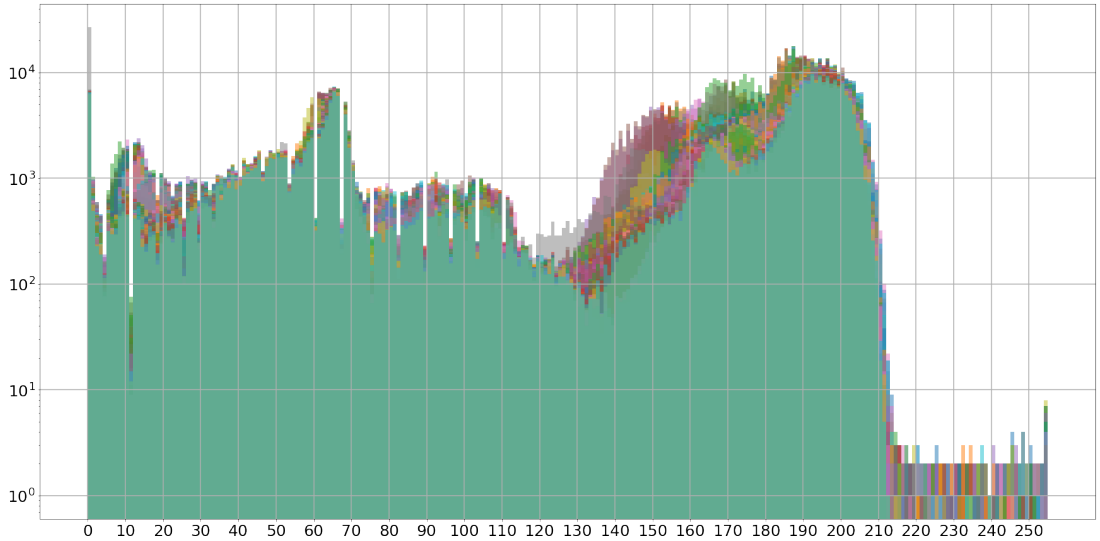

Figure 7.26: Histogram of the pixel distribution in the V channel - 300 images

As we can see in both histograms, although we can identify certain thresholding ranges with which to perform the thresholding operation, these vary for each image. Therefore, it would not be appropriate to use fixed thresholding ranges, as we proposed with the simulated images. To solve this, we propose the use of the Multi-Otsu Thresholding, which, from a designated number of classes, automatically calculates the respective thresholding values [61]. In this way, by applying the algorithm to the image under analysis, we get the results shown in Figures 7.27 and 7.28.

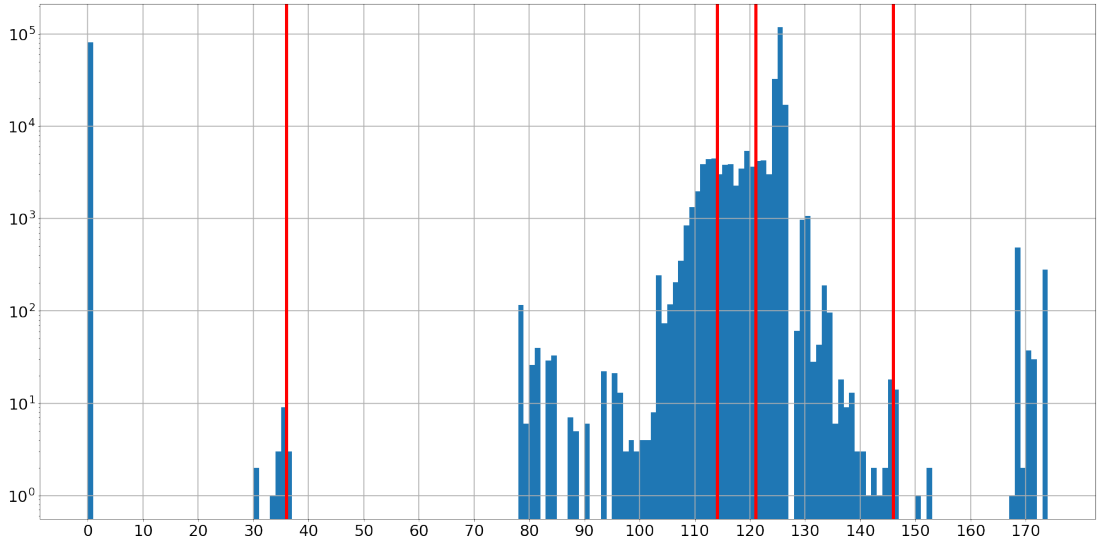

Figure 7.27: Thresholding ranges computation - H channel

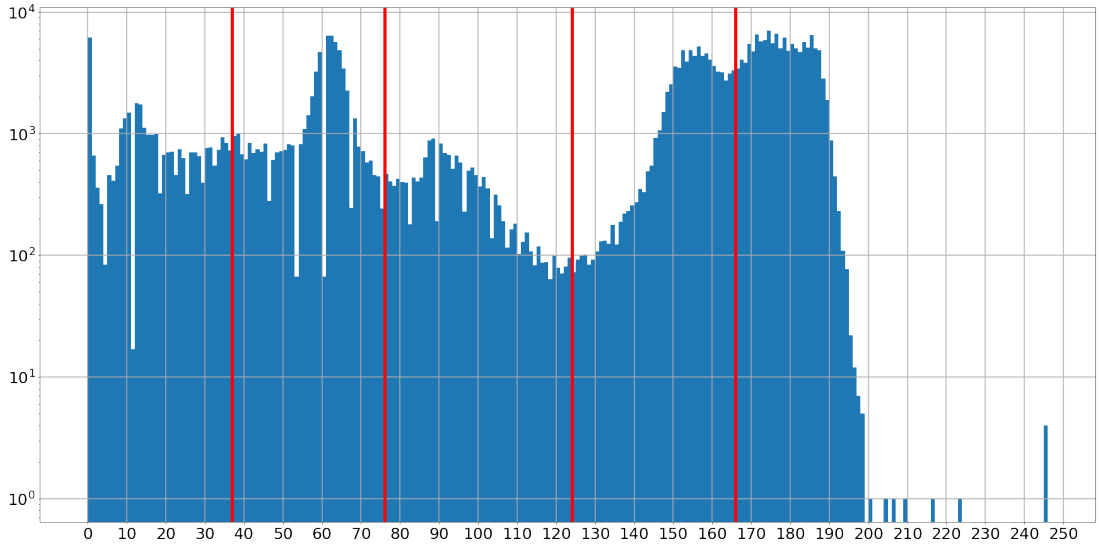

Figure 7.28: Thresholding ranges computation - V channel

The red lines in each histogram represent the computed limits for each thresholding range. In this case, we are using the algorithm to distinguish 5 classes (regions) in the image, thus we will have 5 thresholding ranges. In Figures 7.29 and 7.30, we show the resulting binary images after applying each thresholding range to each channel. The threshold values shown are those calculated by the algorithm for this particular image. The green area is the region that represents to the pixels belonging to each thresholding range.

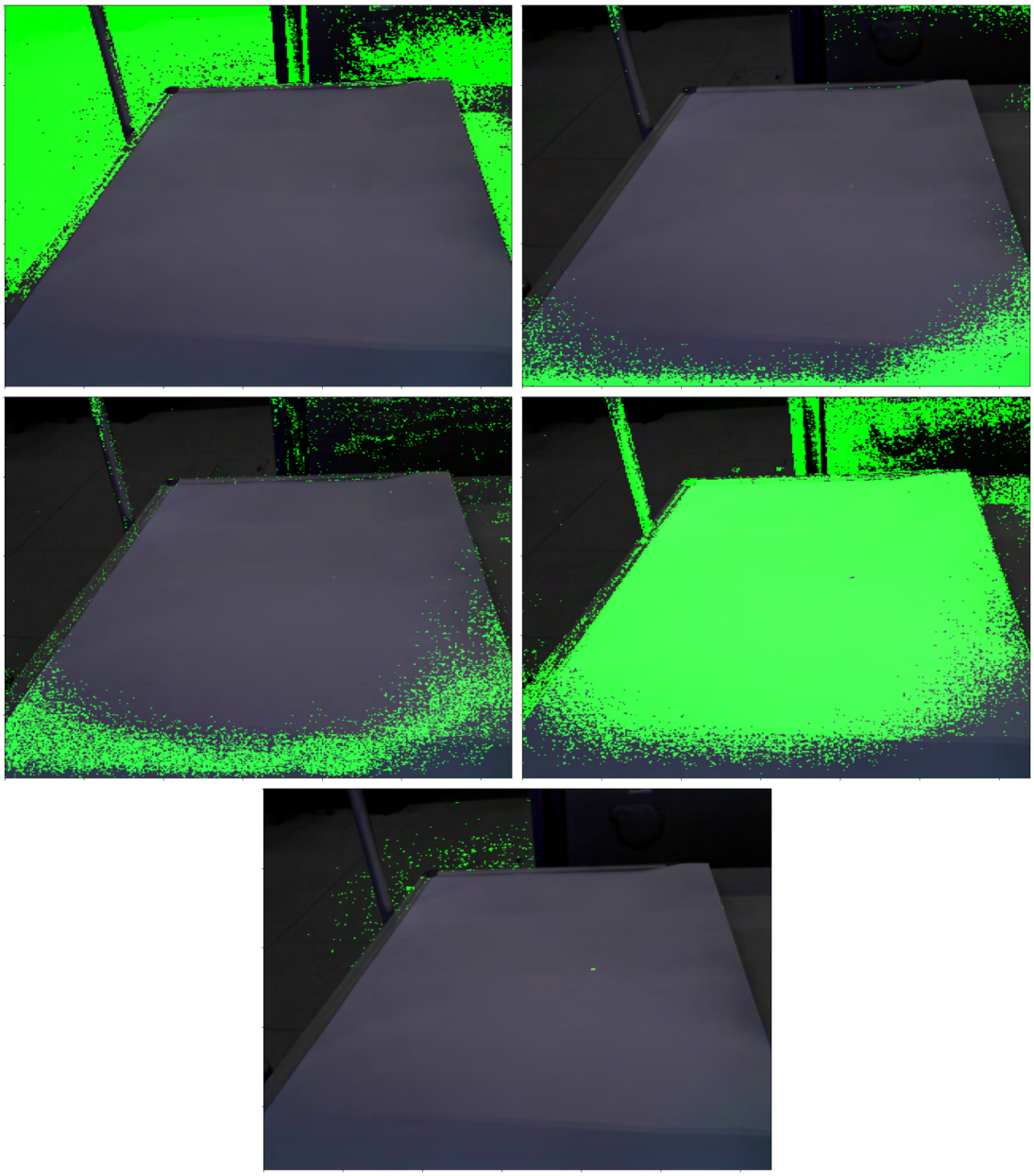

Figure 7.29: Thresholding operation in H channel: (upper left corner) thresholding range: $[0-36]$, (upper right corner) thresholding range: $[36-114]$, (lower left corner) thresholding range: $[114-121]$, (lower right corner) thresholding range: $[121-146]$, (bottom) thresholding range: $[146-180]$

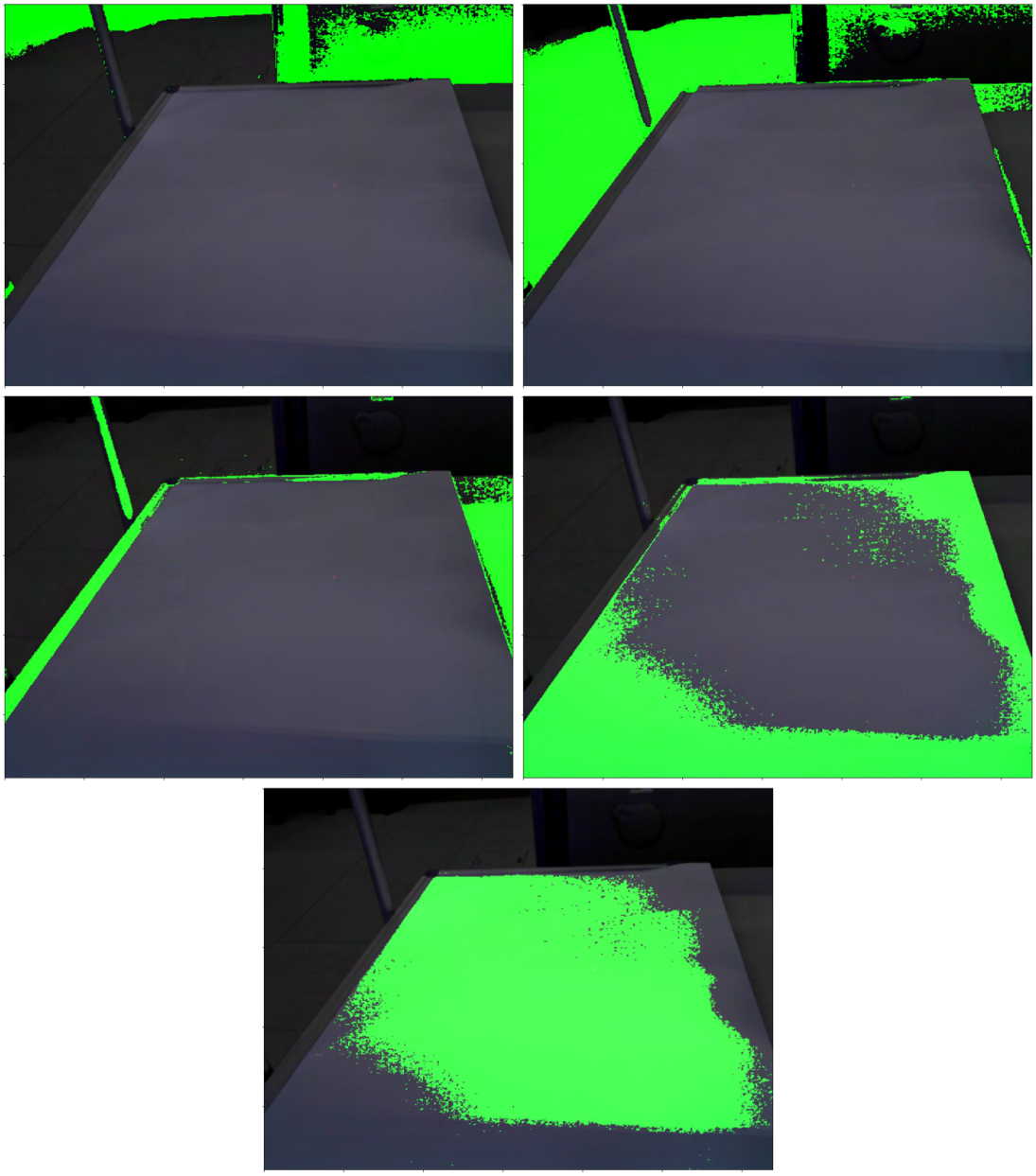

Figure 7.30: Thresholding operation in V channel: (upper left corner) thresholding range: $[0-37]$, (upper right corner) thresholding range: $[37-76]$, (lower left corner) thresholding range: $[76-124]$, (lower right corner) thresholding range: $[124-166]$, (bottom) thresholding range: $[166-255]$

Based on the results shown above, we decide to use the last computed thresholding ranges for each channel in such a way that, by multiplying both binary masks, we will detect the laser point. We show the resulting binary image after multiplying both binary masks in Figure 7.31.

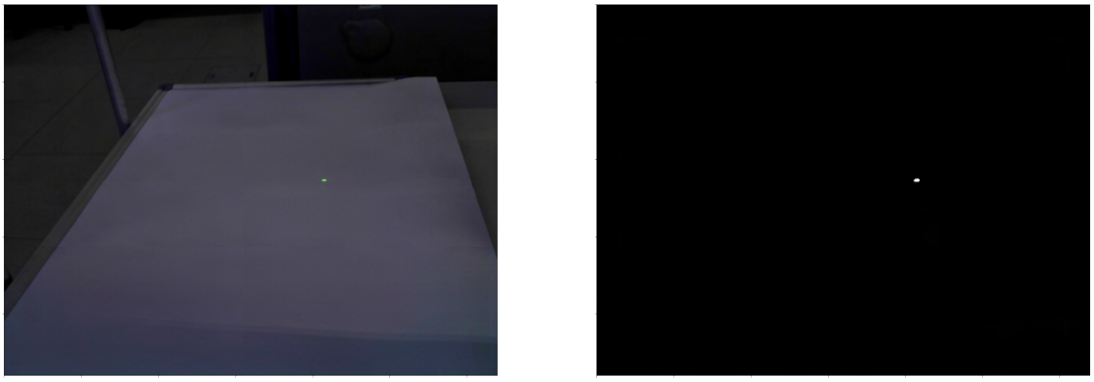

Figure 7.31: Laser point detection

Following the algorithm, once we have detected the laser point, we apply a segmentation operation and compute the centroid of the region containing the laser point. Finally, in Figure 7.32, we show the results after applying the new approach for the laser point localization algorithm to other real images.

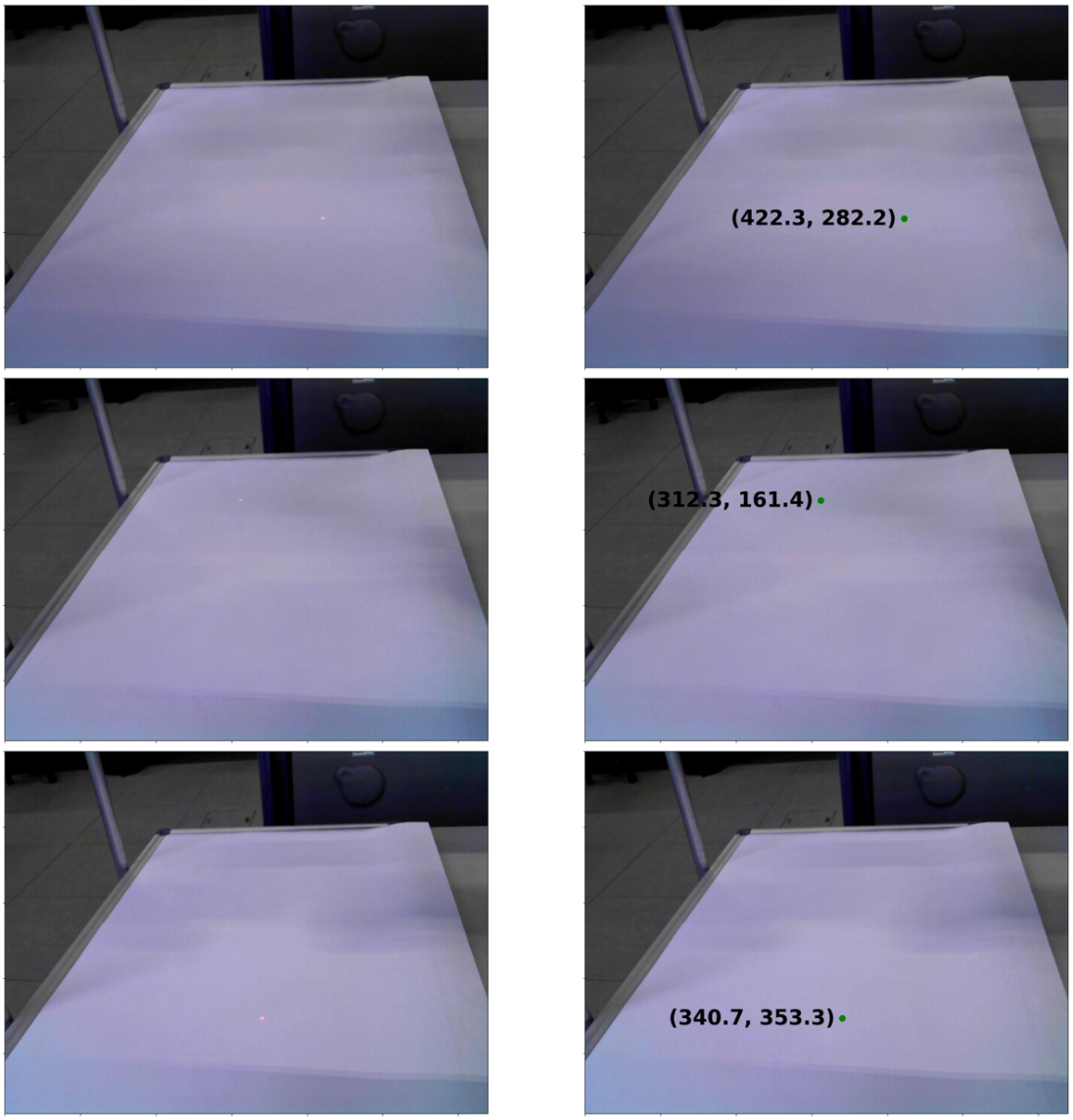

Figure 7.32: Results after applying the new laser point localization algorithm

7.4 Highlighted points

- As suggested in section 5.4, in order to avoid the reflectance problems because of the material of the whiteboard, we have added a paper cover to its surface.
- We can think of the positioning system as a means of spatially locating itself in the real world (with origin at the robot's base) from the world seen by the camera. Therefore, the system is closely linked to the conditions in which the camera is observing the real world. Based on this, the creation of the deep learning model, which is the heart of the positioning system, will only take place once, as long as the camera's localisation (position and orientation) remains unchanged. Thus, only if we change the camera location, we will be forced to re-develop the positioning system, i.e. to redo the process of data acquisition and creation of the deep learning model.
- As mentioned in the system description (Section 7.1), we have proposed a new alternative that replaces the use of the pin-hole camera model and the lens distortion model with a model based on feedforward neural networks. Although we have had good results, reaching a positioning system with an accuracy close to 2 mm, the thesis has not focused on making a comparison between the two proposals. Future work could contemplate this analysis.

Chapter 8

Robot kinematics

As mentioned in the introduction, we will dedicate a chapter to study the kinematics of a robot manipulator, in particular, the UR3e robot.
Considering an articulated mechanical structure as a system of rigid objects (links) connected by joints (prismatic or revolute), the kinematics refers to the motion of these rigid objects without considering the forces and torques that cause the motion. Normally, we can distinguish two spaces in which we carry out the motion: joint space (in which we define the robot configuration), and operational space (in which we define the end-effector location). Therefore, we require some transformation models between these two spaces, since we control the robot in the joint space, while we define the tasks in the operational space. For that, we consider two classes of models: forward kinematics (end-effector location as a function of the joint variables), and inverse Kinematics (joint variables as a function of the end-effector location). For the UR3e robot, the joint variable are the joint angles. In this section, we will see in more detail these models. Likewise, a method for describe the robot structure. [62][63]

8.1 Denavit-Hartenberg representation

The forward and inverse kinematics analysis of an n-link robot manipulator can be complex. For this reason, some conventions for selecting the reference frame to each link provide a systematic procedure for performing this analysis. The most popular is the Denavit-Hartenberg (DH) convention, which assigns the coordinate frames for a robot manipulator, as shown in Figure 8.1[64]

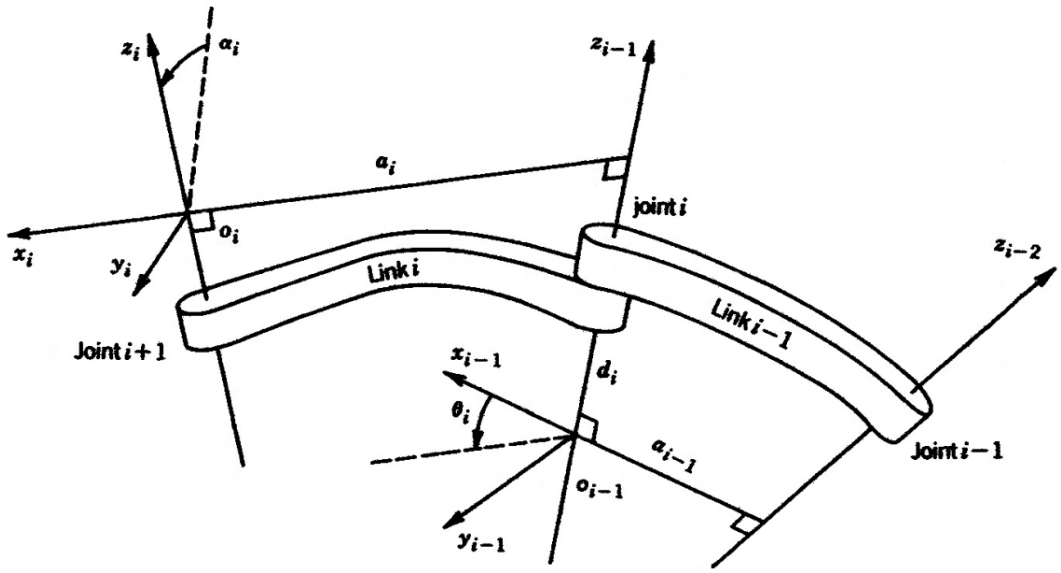

Figure 8.1: Denavit-Hartenberg frame assignment [64]

Where θ_i (joint angle), a_i (link offset), d_i (link length) and α_i (link twist) are called DH parameters, and we can give each the following physical interpretations:

- a_i: The distance between the axes z_{i-1} and z_i, measured along the axis x_i.
- α_i: The angle between the axes z_{i-1} and z_i, measured in a plane normal to x_i.
- d_i: The distance between the origin o_{i-1} and the intersection of the x_i axis with z_{i-1}, measured along the z_{i-1} axis.
- θ_i: The angle between x_{i-1} and x_i measured in a plane normal to z_{i-1}.

By this convention, a robot manipulator with n joints (numbered from 1 to n) will have $n+1$ links (numbered from 0 to n, starting from the base). The axes z_i $(i : 0, 1, \ldots, n-1)$ will be the actuation axes for the joints $i+1$ in such a way that z_0 is the actuation axis for joint 1, z_1 for joint 2, etc. Besides, we fix the joint i location relating to the link $i-1$, therefore when we actuate on the joint i, the link i will move, while the link $i-1$ does not move. Finally, in order to use this convention, the coordinate frames axes i and $i-1$ have to fill the conditions below [64]:

- The axis x_i is perpendicular to the z_{i-1} and z_i axes.
- The axis x_i intersects to the z_{i-1} and z_i axes.
- The origin of the joint n is at the intersection of the x_i and z_i axes.
- The axis y_i completes a right-handed reference frame based on the x_i and z_i axes.

Under the conditions above, the DH convention state that we can represent the homogeneous transformation ${}^{i-1}\mathrm{A}_i$, which represents the location of the frame i relating to the coordinate frame $i-1$, as the product of four basic transformations, as shown below [64]:

$$ {}^{i-1}\mathrm{A}_i = \mathrm{Rot}_{z,\theta_i}\mathrm{Trans}_{z,d}\mathrm{Trans}_{x,a_i}\mathrm{Rot}_{x,\alpha_i} $$

$$ {}^{i-1}\mathrm{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Therefore, we can get the homogeneous transformation matrix ${}^{i-1}\mathrm{A}_i$ as:

$$ {}^{i-1}\mathrm{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.1} $$

The parameters α, d, and a are always constant and they will depend on the robot. Therefore, the computation will depend only on the joints angle. For the UR3e robot, we show the reference frames assignment based on DH convention in Figure 8.2, and its DH parameters in Table 8.1 [65]. From this point on, we will focus on the UR robots.

Joint	θ_i**[rad]**	a_i**[m]**	d_i**[m]**	α_i**[rad]**
1	θ_1	0	0.15185	$\frac{\pi}{2}$
2	θ_2	-0.24355	0	0
3	θ_3	-0.2132	0	0
4	θ_4	0	0.13105	$\frac{\pi}{2}$
5	θ_5	0	0.08535	-$\frac{\pi}{2}$
6	θ_6	0	0.0921	0

Table 8.1: Denavit–Hartenberg parameters - UR3e

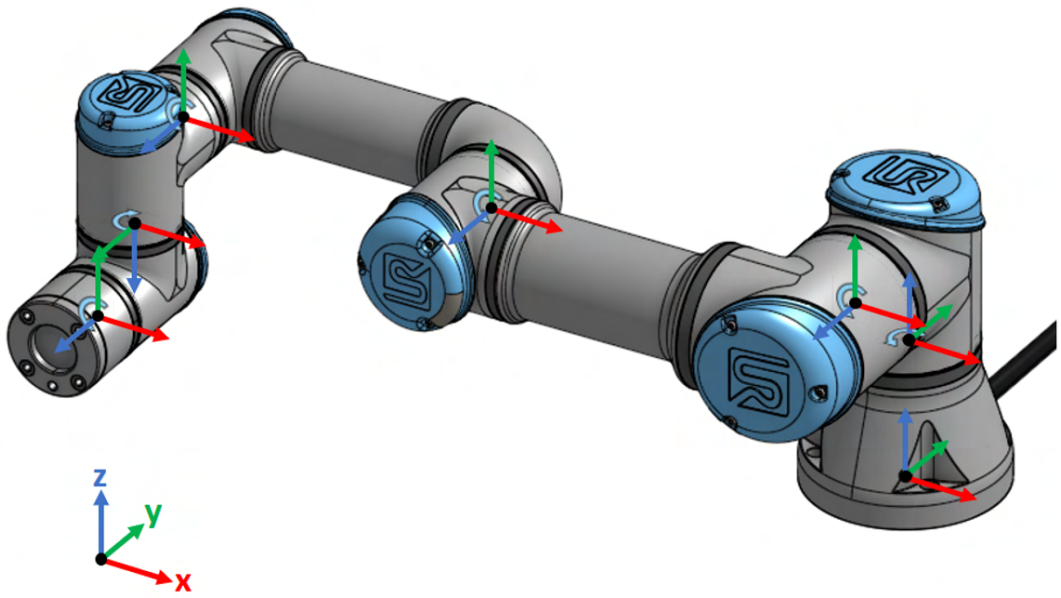

Figure 8.2: Reference frames assignment for the UR3e robot based on DH convention

8.2 Forward kinematics

The forward kinematics (FK) aim for a UR robot is to find the end-effector location (tool frame) relating to the base (station frame) given the joints angles, thus a transformation from the joint space to the operational space. So, we need to compute the homogeneous transformation matrix between the tool and station frames, i.e. the matrix ${}^{0}\mathrm{T}_{n}$. For a robot manipulator, we can describe this transformation by concatenating transformations between frames fixed in adjacent links of the robot, as shown below [62]:

$$ {}^{0}\mathrm{T}_{n} = {}^{0}\mathrm{A}_{1}\,{}^{1}\mathrm{A}_{2}\,{}^{2}\mathrm{A}_{3}\cdots{}^{n-3}\mathrm{A}_{n-2}\,{}^{n-2}\mathrm{A}_{n-1}\,{}^{n-1}\mathrm{A}_{n} \tag{8.2} $$

In this sense, the DH convection offers us a means to simplify this computation. For the UR robot manipulator, the requested matrix will have the form below:

$$ {}^{0}\mathrm{T}_{6} = \begin{bmatrix} \mathrm{SO}(3) & {}^{0}p_{6} \\ \vec{0} & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & {}^{0}p_{6}^{x} \\ n_y & o_y & a_y & {}^{0}p_{6}^{y} \\ n_z & o_z & a_z & {}^{0}p_{6}^{z} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Where $\mathrm{SO}(3)$ and ${}^{0}p_{6}$ represents the orientation and position of the end-effector frame, respectively, relating to the station frame, as shown in Figure 8.3.

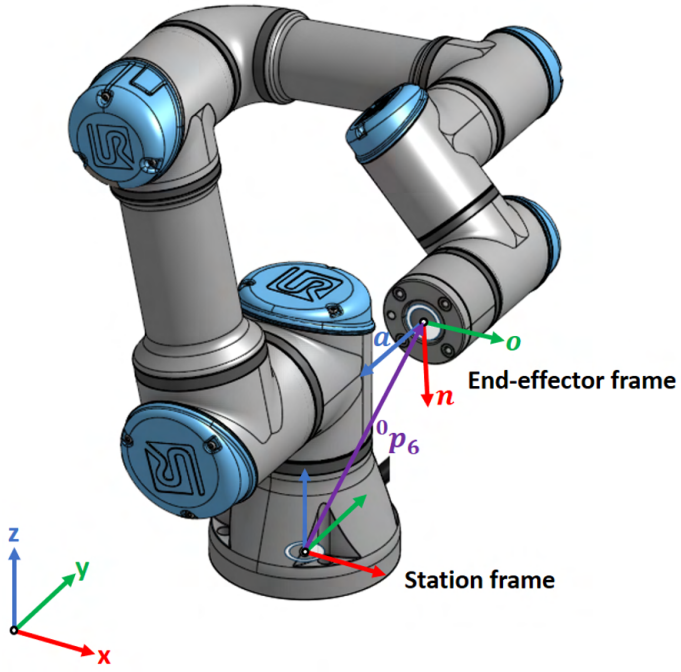

Figure 8.3: End-effector location relating to the station frame

For the UR3e robot, using the values in Table 8.1 and combining (8.1) and (8.2), we can state that the FK for this robot is described by:

$$
\begin{aligned}
n_x &= (s_1 s_5 + c_1 c_5 c_{234}) c_6 - s_6 s_{234} c_1 \\
o_x &= -(s_1 s_5 + c_1 c_5 c_{234}) s_6 - c_6 s_{234} c_1 \\
a_x &= s_1 c_5 - s_5 c_{234} c_1 \\
n_y &= (s_1 c_5 c_{234} - s_5 c_1) c_6 - s_1 s_{234} s_6 \\
o_y &= -(s_1 c_5 c_{234} - s_5 c_1) s_6 - s_1 s_{234} c_6 \\
a_y &= -s_1 s_5 c_{234} - c_1 c_5 \\
n_z &= s_6 c_{234} + s_{234} c_5 c_6 \\
o_z &= -s_6 s_{234} c_5 + c_6 c_{234} \\
a_z &= -s_5 s_{234} \\
{}^0p_6^x &= d_6 s_1 c_5 + d_4 s_1 - d_6 s_5 c_1 c_{234} + d_5 s_{234} c_1 + a_2 c_1 c_2 + a_3 c_1 c_{23} \\
{}^0p_6^y &= -d_6 s_1 s_5 c_{234} + d_5 s_1 s_{234} + a_2 s_1 c_2 + a_3 s_1 c_{23} - d_6 c_1 c_5 - d_4 c_1 \\
{}^0p_6^z &= a_2 s_2 - d_6 s_5 s_{234} + a_3 s_{23} - d_5 c_{234} + d_1
\end{aligned}
$$

From this section, we will use the short-hand above, where:

- $c_i := \cos\theta_i, s_i := \sin\theta_i$
- $c_{ij} := \cos(\theta_i + \theta_j), s_{ij} := \sin(\theta_i + \theta_j)$
- $c_{i-j} := \cos(\theta_i - \theta_j), s_{i-j} := \sin(\theta_i - \theta_j)$

8.3 Inverse kinematics

The inverse kinematics (IK) aim for a UR robot is to find the joints angles given the end-effector location (tool frame) relating to the base (station frame), thus a transformation from the operational space to the joint space. From the previous section, we know both spaces find a relation through to the matrix ${}^0\mathrm{T}_n$, thus the idea is, given a desired end-effector location, represented by the homogeneous transformation H, to find one or all the solution $(\theta_1, \theta_2, \ldots, \theta_n)$ that satisfy the equation below [64]:

$$ {}^0\mathrm{T}_n(\theta_1, \theta_2, \ldots, \theta_n) = \mathrm{H} \tag{8.3} $$

Where:

$$ {}^0\mathrm{T}_n(\theta_1, \theta_2, \ldots, \theta_n) = {}^0\mathrm{A}_1(\theta_1) \cdots {}^{n-1}\mathrm{A}_n(\theta_n) $$

If we analyze the equation (8.3), we can state that the IK problem implies twelve nonlinear equations with n unknown variables, which we can write as:

$$ \mathrm{T}_{ij}(q_1, q_2, \ldots, q_n) = \mathrm{h}_{ij} \quad i = 1, 2, 3 \quad j = 1, 2, 3, 4 $$

Where T_{ij} and h_{ij} refer to the twelve nontrivial entries for ${}^0\mathrm{T}_n$ and H, respectively. Regarding to the IK solutions, whereas the FK problem always has a unique solution, the IK problem might or might not have a solution depending on if the desired end-effector location lie or not in the workspace of the robot manipulator, however, even when the desired location is within the workspace (solution exists), the solution might or might not be unique. It is clear the IK problem is more difficult that the FK problem [62][64]. For most robot manipulators, the obtained equations are much too difficult to solve directly in closed-form (geometric and algebraic methods), thus sometimes we require numerical methods (symbolic elimination, continuation and iterative methods). We will analyze the closed-form methods applied to the UR3 robot in this section [66], and the numerical methods in Section 4.

8.3.1 Geometric IK solution for UR3e robot (Closed-form)

This geometric perspective solution is workable because we can solve the IK problem with pure constructive straightedge and compass geometry, i.e. constructing lengths, angles, and other geometric figures using an idealized ruler and pair of compasses. To develop this solution, we will use the kinematic skeleton and nomenclature shown in Figure 8.4. We will use this nomenclature in the rest of the document.

We assume that J_0 (station frame) is in the world origin $(0, 0, 0)$; and that J_7 (end-effector frame) is in the front face centre point of the flange. Since the joint 0 and joint 7 do not present rotation, we will consider them as pseudo-joints. $\mathrm{J}_1, \mathrm{J}_2, \ldots, \mathrm{J}_6$ represent the intermediate coordinate frames between J_0 and J_7 frames. A, B, ..., G represent the interme-

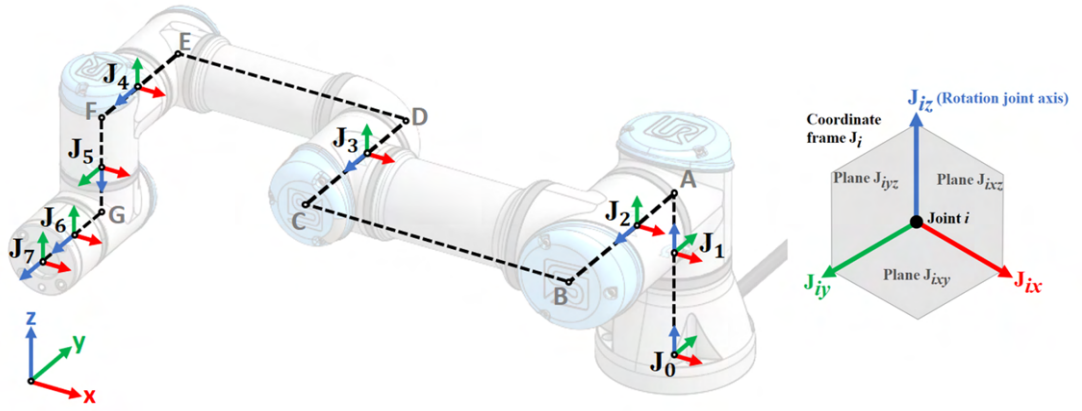

Figure 8.4: Kinematic skeleton and nomenclature for UR3e robot

diate points between axes. Finally, we fully define the robot geometry defining the axial offset between consecutive joints. We show these values in Table 8.2 in such way that, for example, the axial offset $\mathrm{J}_{0y} - \mathrm{J}_{5y}$ would be the sum of $\mathrm{J}_{(0-1)y}, \mathrm{J}_{(1-2)y}, \mathrm{J}_{(2-3)y}, \mathrm{J}_{(3-4)y}$ and $\mathrm{J}_{(4-5)y}$.

Joint	ΔX[mm]	ΔY[mm]	ΔZ[mm]
0 - 1	0.000	0.000	94.750
1 - 2	0.000	-62.950	56.900
2 - 3	-243.550	-10.600	0.000
3 - 4	-213.200	3.450	0.000
4 - 5	0.000	-60.950	-43.100
5 - 6	0.000	-43.100	-42.250
6 - 7	0.000	-49.000	0.000

Table 8.2: Axial Offsets - UR3e

According to the robot geometry, we can state that rotation axes J_{2z}, J_{3z} and J_{4z}, highlighted in Figure 8.5, will be parallel in any configuration, therefore the projection of their origins onto the plane J_{1xz} will form a triangular frame with fixed lengths BC and DE which we will use to compute the joint angle θ_3, once we have the projected distance between the joints 2 and 4. It is important also to note that these consecutive parallel rotation axes introduce a constraint on the origin position of the frame J_5 (and points F, G) relating to the plane J_{1xz}, since the distance of these points onto this plane will be always 131.050 mm (axial offset $\mathrm{J}_{0y} - \mathrm{J}_{5y}$). In Figure 8.5, we point out the projection of the point G onto the plane J_{1xz} as the point X, which we will use to determine the $\mathrm{J}_{1y}/\mathrm{J}_{2z}$ axes orientation. In addition, we also can state that the position of the points B, C, D and E are irrelevant to the kinematic model of the robot since if we offset B/C and D/E by a factor along the plane J_{1xz} will not have any change. However, from a physical point of view, in term of how the robot folds together, this could be create potential for self-collision.

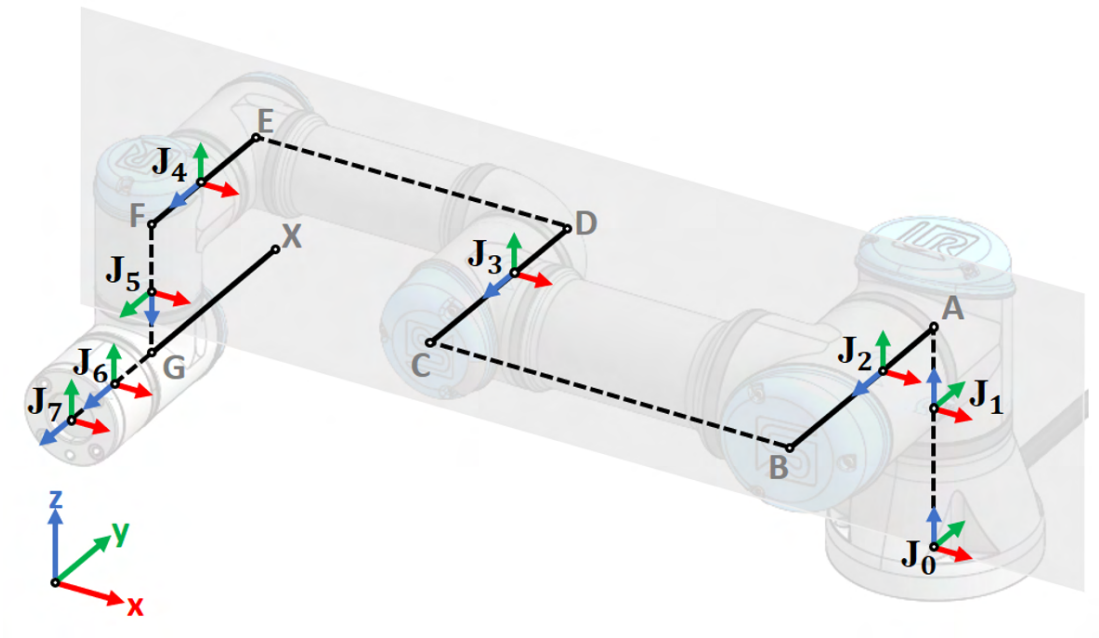

Figure 8.5: UR3e - Geometric observations

We have two ways to find out the joint positions, as shown in Figure 8.6: starting for the base and move upwards (Bottom-Up solution) and the other way around (Top-Down solution). In both scenarios, the idea is to find the J_{2z} and J_{5z} axes orientation. We show both solutions below.

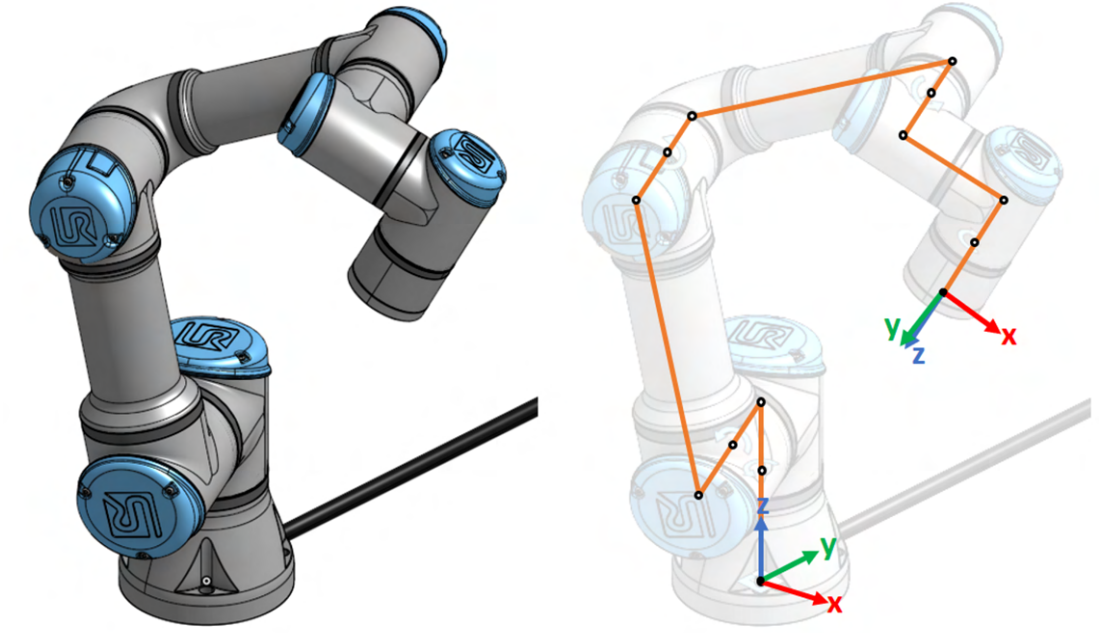

Figure 8.6: Location of intermediate joint positions

8.3.1.1 Top-Down solution

We start with the origin of the frame J_7, and we locate the point G which is 92.100 mm back from this frame (axial offset $\mathrm{J}_{5y} - \mathrm{J}_{7y}$). First, we draw a cylinder in the plane J_{0xy} along the J_{0z} axis with radius 131.050 mm (axial offset $\mathrm{J}_{0y} - \mathrm{J}_{5y}$), and we find its intersection with the plane P, which have origin in G and is normal to the J_{7z} axis. The intersection curve K will be an ellipse, or a circle, when the J_{7z} axis is parallel to J_{0z} axis. Then, we draw the tangent lines from G to K in such a way that if we project in the plane J_{0xy} the directions from the K centre to the tangent T_1 and T_2 points; we define the two solutions for the J_{2z} axis orientation. Now, we intersect the tangent lines of K with a circle with centre G and radius 43.100 mm (axial offset $\mathrm{J}_{4z} - \mathrm{J}_{5z}$), and that belongs to the plane P. The directions from G to the intersections points $\mathrm{W}_{\mathrm{a1,a2}}$ and $\mathrm{W}_{\mathrm{b1,b2}}$ define the solutions for the J_{5z} axis orientation. We show the procedure above in Figure 8.7.

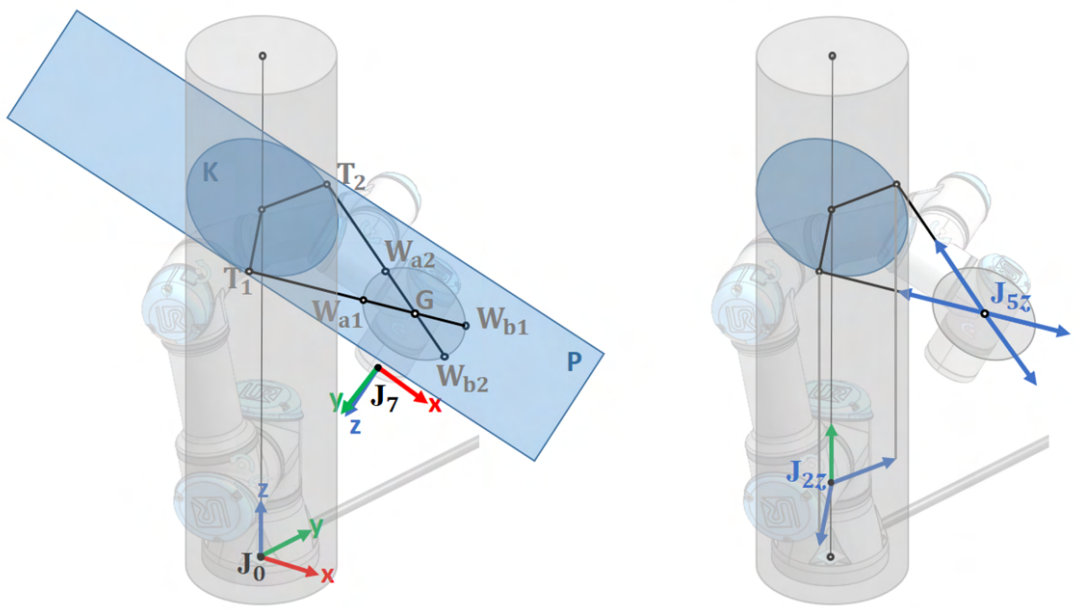

Figure 8.7: Top-Down solution

If there is no intersection, means that the J_{0z} axis is parallel to the J_{5z} axis, thus $\mathrm{J}_{5z} = \pm\mathrm{J}_{0z}$. Finally, the intersection of the two vertical tangent planes from G onto the cylinder and the horizontal plane J_{0xy} define the J_{2z} axis orientation. According to the solutions for the J_{2z} and J_{5z} axes, we can state that in total we will have eight solutions (robot configurations) for the IK problem. For example, in Figure 8.8, we show all the solutions for a certain desired end-effector location [66].

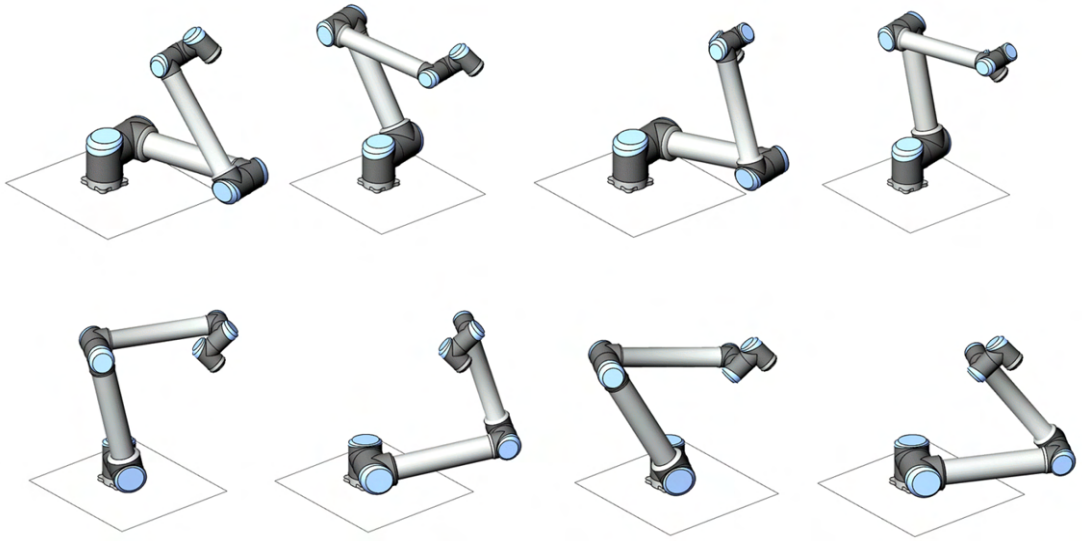

Figure 8.8: Eight solutions for the IK problem [66]

8.3.1.2 Bottom-Up solution

Recalling the constraints of the parallel axes J_{2z}, J_{3z} and J_{4z}, we define a circle with centre in the origin of the frame J_0 and with radius 131.050 mm (axial offset $\mathrm{J}_{0y} - \mathrm{J}_{5y}$), where the tangent lines from G to the circle, projected in the plane J_{0xy}, determine the two directions for the J_{2z} axis, as shown in Figure 8.9. It is important to note that if G lies in the circle, there is no solution.

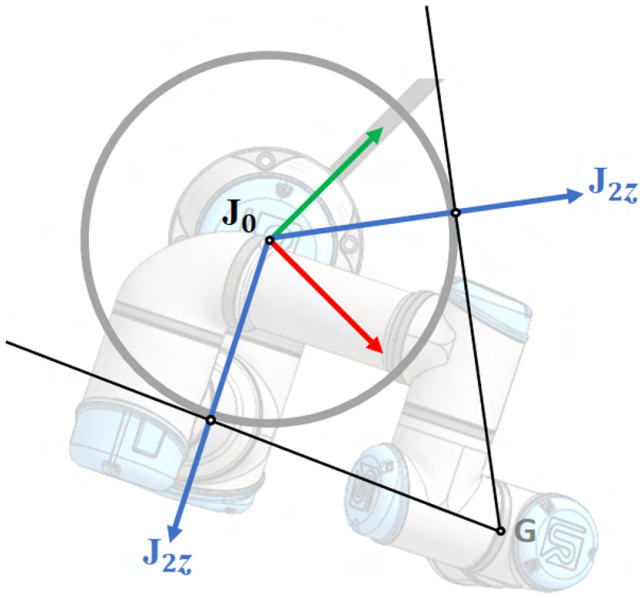

Figure 8.9: Bottom-up solution

We already know the J_{2z} axis direction, so we also know the directions of the axes J_{3z} and J_{4z}. Now, we locate the point G in such a way that the J_{5z} axis is the radial direction of a circle with centre in G and radius 43.100 mm (axial offset $J_{4z} - J_{5z}$). This circle represents all the possible positions for the point F, but because of the coordinate frames J_4/J_5 and J_5/J_6 are square, the J_{5z} axis direction must be perpendicular to the axes J_{4z} and J_{7z}, i.e. the J_{4z} axis is tangent to this circle. Therefore, we already have the J_{5z} axis direction and its inverse, as shown in Figure 8.10.

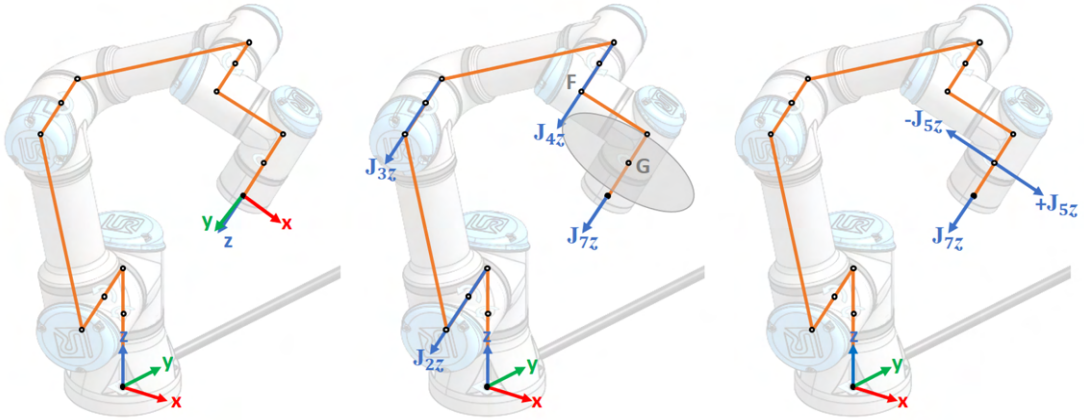

Figure 8.10: Determination of the J_{5z} direction

Finally, if we project the origin of the frame J_4 in the plane J_{2xy}, we will have a triangle J_{234} of which we know its three lengths (J_{23}, J_{34} and J_{24}) and two of its points (origins of the frames J_2 and J_4), but we do not know the positions or orientations of the remaining frame J_3. So, we can find it from the intersection of two circles with centre in the origins of the frames J_2 and J_4 with radius J_{23} and J_{34}, respectively, as shown in Figure 8.11. It is important to note that if the frame J_7 is far enough from the frame J_0 such that J_{23} + $J_{34} < J_{24}$, then there is no solution because of the robot will be overstretched. Another important case is when the internal angle of the frame J_3 is zero, which generates a self-collision problem instead of a kinematic one.

8.3.2 Analytical IK solution for UR3e robot (Closed-form)

For this solution, based on [67][68][69], we will use the UR3e kinematic structure shown in Figure 8.12. The distances $-a_2, -a_3, d_1, d_4, d_5, d_6$ are the DH parameters. Unlike the kinematic skeleton used above, the frame J_6 location represent the end-effector location. Recalling the equation (8.3), we can state that the IK problem for the UR3e robot is described by the equation below:

$$^0T_6(\theta_{1...6}) = H$$

Where $(\theta_1, \theta_2, \ldots, \theta_6) \in [0; 2\pi[$, and H is the desired end-effector location.

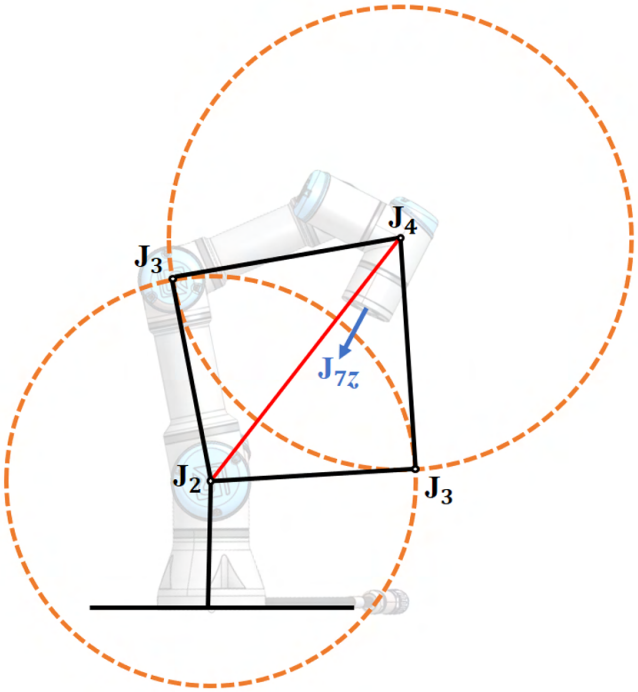

Figure 8.11: Determination of the J_3 position

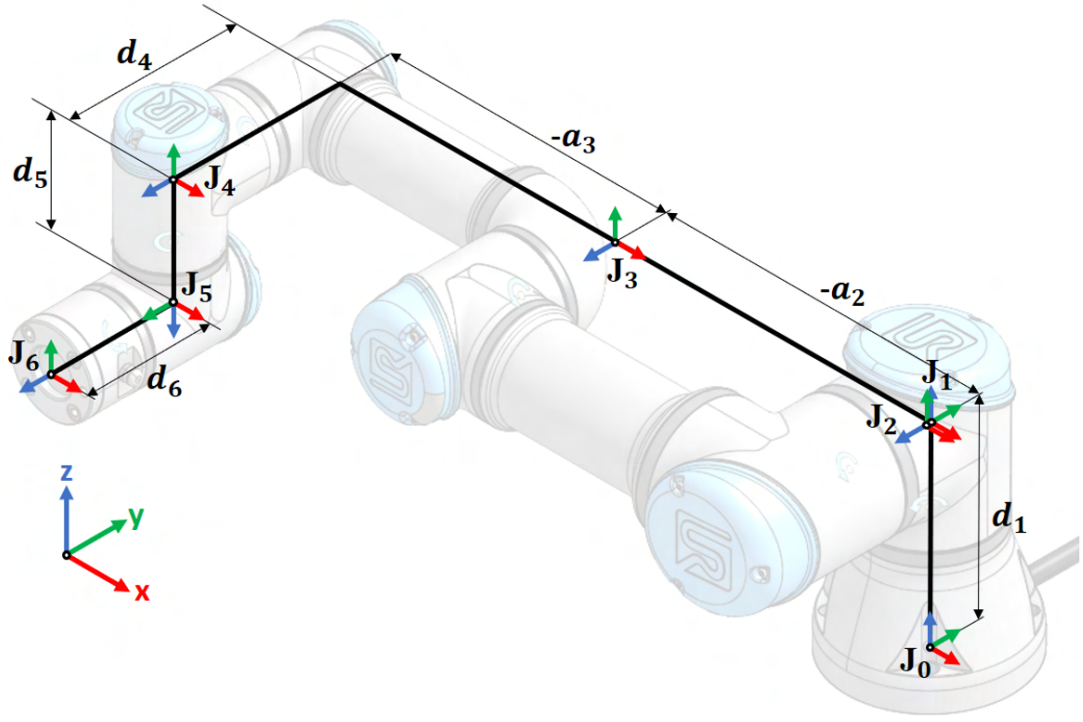

Figure 8.12: UR3e robot kinematic structure in zero position

8.3.2.1 Computation of the angle θ_1

First, we compute the frame J_5 position relating to the frame J_0 ($^0p_5$). As shown in Figure 8.13, we can find $^0p_5$ translating backwards a distance d_6 from the frame J_6 to the frame J_5, along the $\mathrm{J}_{6\mathrm{z}}$ axis.

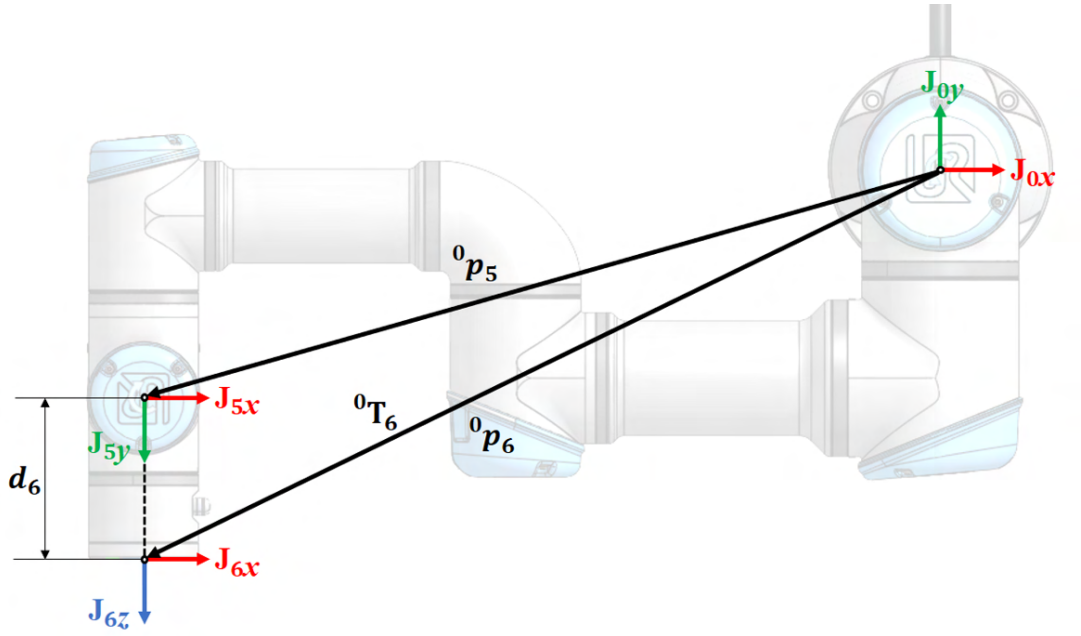

Figure 8.13: Computation of the frame J_5 position

Therefore, we can compute $^0p_5$ from the equation below:

$$^0p_5 = {}^0p_6 - d_6\mathrm{SO}(3)^z = {}^0\mathrm{T}_6 \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} \tag{8.4}$$

To find the joint angle θ_1, we simulate the robot movement around the joint 1 and we analyze the joint 5 position relating to the frames J_1 and J_0, viewed from the plane J_{0xy}, as shown in Figure 8.14. For the rest of the document, $^{i-1}\theta_i := \theta_i$.

From the graph, we can state that θ_1 can be computed from $^0\theta_5$ and $^5\theta_1$ as:

$$\theta_1 = {}^0\theta_5 + {}^5\theta_1 \tag{8.5}$$

Where $^0\theta_5$ and $^5\theta_1$ are equal to:

$$^0\theta_5 = \phi_1 \quad {}^5\theta_1 = \left(\phi_2 + \frac{\pi}{2}\right) \tag{8.6}$$

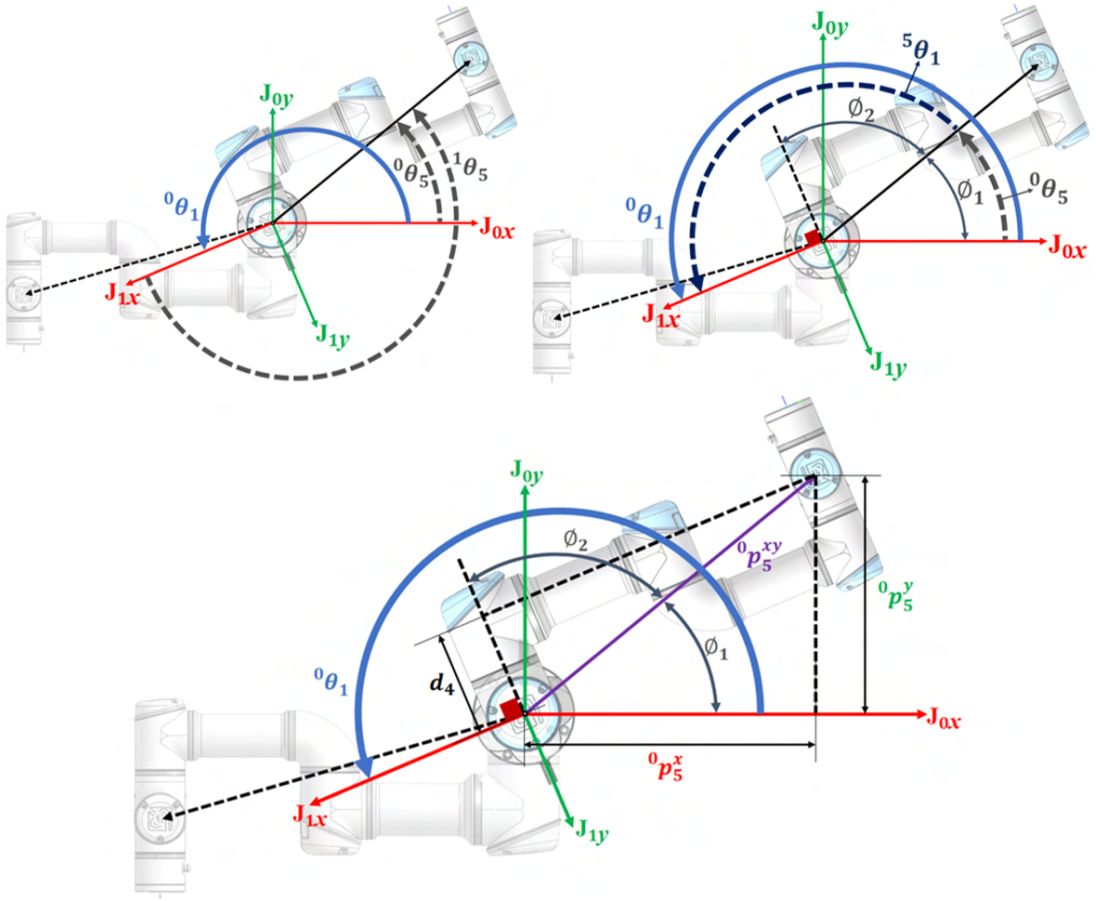

Figure 8.14: Computation of the joint variable θ_1

From the graph, ϕ_1 and ϕ_2 can be computed as:

$$\phi_1 = \text{atan2}({}^0p_5^y, {}^0p_5^x) \quad \phi_2 = \pm \arccos\left(\frac{d_4}{\sqrt{({}^0p_5^x)^2 + ({}^0p_5^{y2})}}\right) \tag{8.7}$$

Finally, combining (8.4), (8.6) and (8.7) in (8.5) we can compute the joint angle θ_1 from the equation below:

$$\theta_1 = \text{atan2}({}^0p_5^y, {}^0_xp_5) \pm \arccos\left(\frac{d_4}{\sqrt{{}^0p_5^{x2} + {}^0p_5^{y2}}}\right) + \frac{\pi}{2} \tag{8.8}$$

In analogous to the geometry solution, the equation above represents the two J_{2z} axis directions, which correspond to the shoulder is to the “left” or to the “right”, as shown in Figure 8.15.

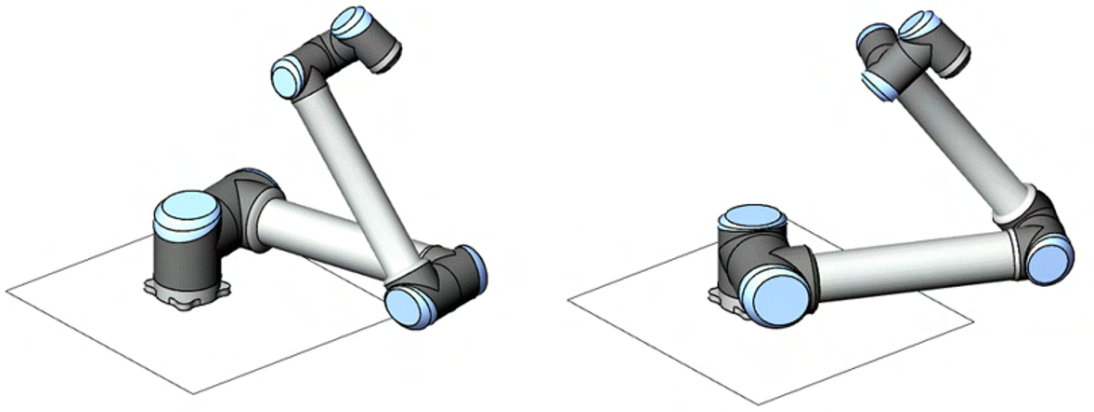

Figure 8.15: Solutions for the joint angle θ_1 [66]

8.3.2.2 Computation of the angle θ_5

To find the joint angle θ_5, we now simulate the robot movement around the joints 1 and 5, and we analyze the joint 6 position relating to the frame J_1 ($^1p_6$) and the frame J_0 ($^0p_6$), viewed from the plane J_{0xy}, as shown in Figure 8.16.

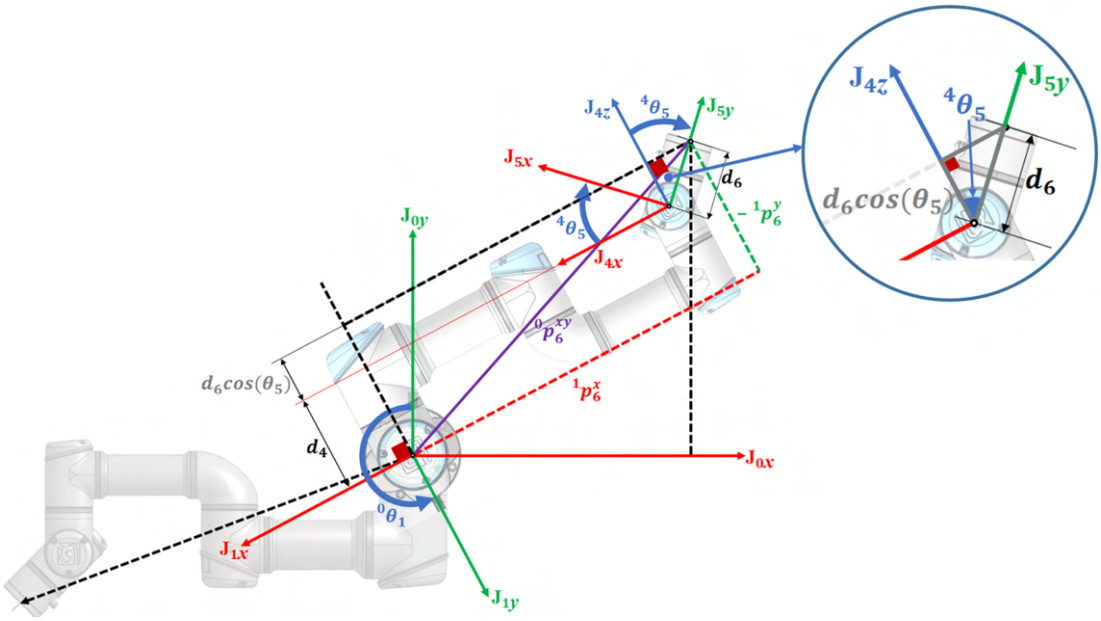

Figure 8.16: Computation for the joint angle θ_5

From the graph, we can state that the y-component of $^1p_6$ ($^1p_6^y$) only depends on the joint angle θ_5, and we can compute it as:

$$-\,{}^1p_6^y = d_4 + d_6 \cos\theta_5 \tag{8.9}$$

Likewise, we can also express ${}^{1}p_{6}^{y}$ as a rotation of ${}^{0}p_{6}$ around the J_{z1} axis:

$$ {}^{0}p_{6} = {}^{0}\mathrm{R}_{1}\,{}^{1}p_{6} $$

$$ {}^{1}p_{6} = \left({}^{0}\mathrm{R}_{1}\right)^{\mathrm{T}}\,{}^{0}p_{6} $$

$$ \begin{bmatrix} {}^{1}p_{6}^{x} \\ {}^{1}p_{6}^{y} \\ {}^{1}p_{6}^{z} \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{0}p_{6}^{x} \\ {}^{0}p_{6}^{y} \\ {}^{0}p_{6}^{z} \end{bmatrix} $$

$$ {}^{1}p_{6}^{y} = {}^{0}p_{6}^{x}(-\sin\theta_1) + {}^{0}p_{6}^{y}\cos\theta_1 \tag{8.10} $$

Finally, equalling (8.9) and (8.10) we can compute the joint angle θ_5 from the equation below:

$$ -d_4 - d_6\cos\theta_5 = {}^{0}p_{6}^{x}(-\sin\theta_1) + {}^{0}p_{6}^{y}\cos\theta_1 $$

$$ \cos\theta_5 = \frac{{}^{0}p_{6}^{x}\sin\theta_1 - {}^{0}p_{6}^{y}\cos\theta_1 - d_4}{d_6} $$

$$ \theta_5 = \pm\arccos\left(\frac{{}^{0}p_{6}^{x}\sin\theta_1 - {}^{0}p_{6}^{y}\cos\theta_1 - d_4}{d_6}\right) \tag{8.11} $$

It is important to note that the solution exists as long as $|{}^{1}p_{6}^{y} - d_4| \leq |d_6|$. In analogous to the geometry solution, the equation above represents the J_{5z} axis directions, which correspond to the wrist is being “up” or “down”, as shown in Figure 8.17.

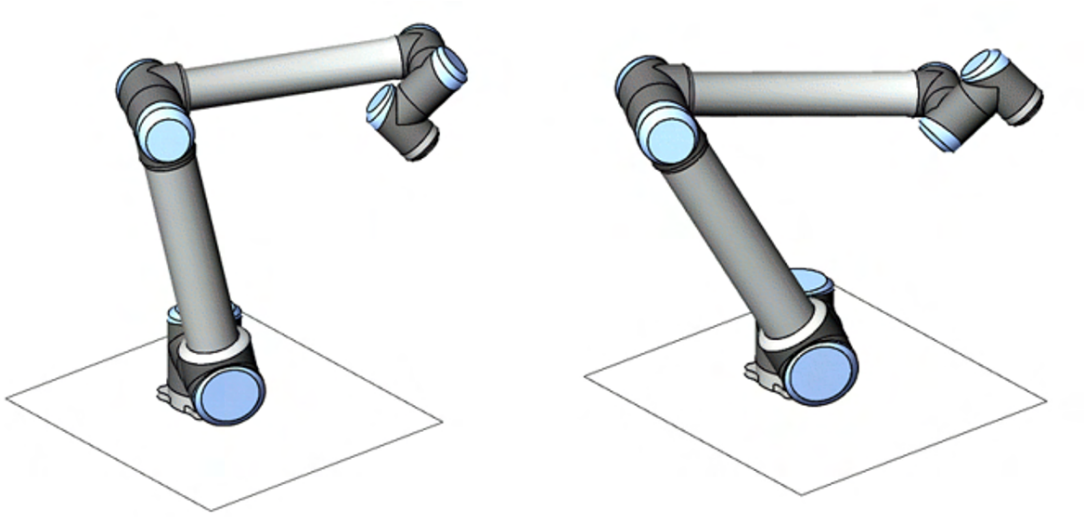

Figure 8.17: Solutions for the joint angle θ_5 [66]

8.3.2.3 Computation of the angle θ_6

To find the joint angle θ_6, we simulate the robot movement around the joints 1, 5 and 6; and we analyze the J_{1y} axis relating to the frame J_6 ($^6\mathrm{J}_{1y}$). Recalling the robot kinematic structure shown in Figure 8.12, we can state that the J_{1y} axis always will be parallel to the J_{2z}, J_{3z} and J_{4z} axes, thus it will only depend on θ_5 and θ_6. In figure 8.18, we describe $^6\mathrm{J}_{1y}$ from its spherical coordinates where the azimuth is -θ_6, the polar angle is θ_5, and -$^6\mathrm{J}_{1y}$ is its inverse.

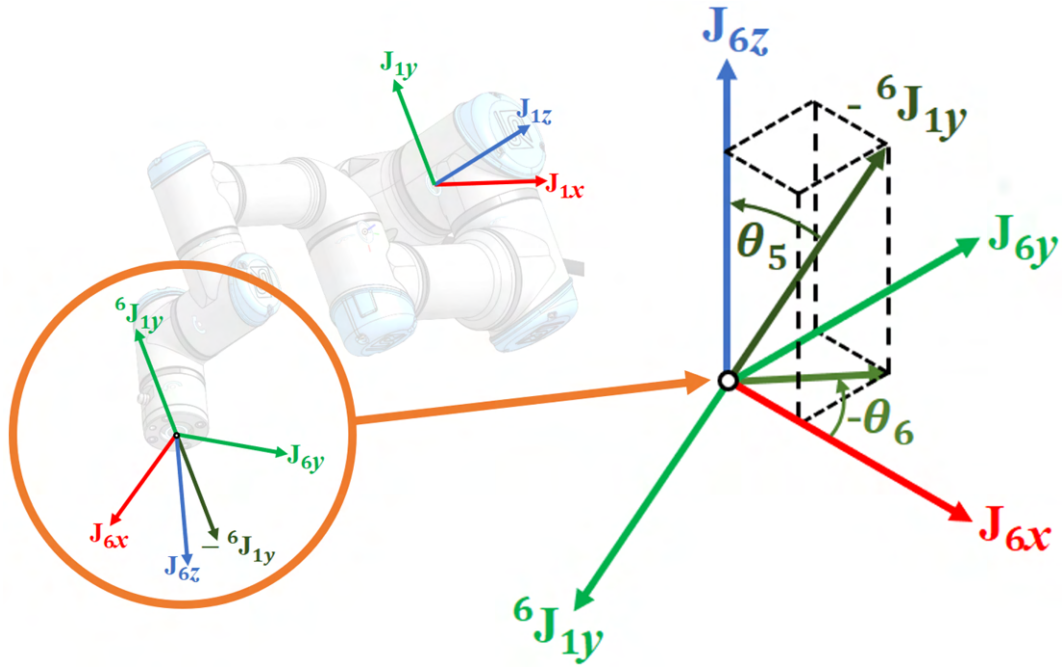

Figure 8.18: Spherical coordinates - $^6\mathrm{J}_{1y}$

Given the spherical coordinates, we can express the -$^6\mathrm{J}_{1y}$ axis as:

$$-{}^6\mathrm{J}_{1y} = \begin{bmatrix} \sin\theta_5 \cos(-\theta_6) \\ \sin\theta_5 \sin(-\theta_6) \\ \cos\theta_5 \end{bmatrix}$$

Therefore, we can express the $^6\mathrm{J}_{1y}$ axis as:

$$^6\mathrm{J}_{1y} = \begin{bmatrix} -\sin\theta_5 \cos\theta_6 \\ \sin\theta_5 \sin\theta_6 \\ -\cos\theta_5 \end{bmatrix} \tag{8.12}$$

By other hand, we can express the frame J_1 rotation relating to the frame J_6 as:

$$^6\mathrm{R}_1 = \begin{bmatrix} ^6\mathrm{J}_{1x} & ^6\mathrm{J}_{1y} & ^6\mathrm{J}_{1z} \end{bmatrix} = (({}^0\mathrm{R}_1)^{\mathrm{T}} \cdot {}^0\mathrm{R}_6)^{\mathrm{T}} \tag{8.13}$$

Where ${}^0\mathrm{R}_1$ is the frame J_1 rotation relating to the frame J_0, and ${}^0\mathrm{R}_6$ is the desired end-effector orientation:

$$ {}^0\mathrm{R}_6 = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \quad {}^0\mathrm{R}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.14} $$

By combining (8.14) in (8.13), we can express ${}^6\mathrm{R}_1$ as:

$$ {}^6\mathrm{R}_1 = \begin{bmatrix} n_x\cos\theta_1 + n_y\sin\theta_1 & -n_x\sin\theta_1 + n_y\cos\theta_1 & n_z \\ o_x\cos\theta_1 + o_y\sin\theta_1 & -o_x\sin\theta_1 + o_y\cos\theta_1 & o_z \\ a_x\cos\theta_1 + a_y\sin\theta_1 & -a_x\sin\theta_1 + a_y\cos\theta_1 & a_z \end{bmatrix} \tag{8.15} $$

And we equal the y-component of (8.15) with (8.12) to find the desired joint angle:

$$ \begin{bmatrix} -n_x\sin\theta_1 + n_y\cos\theta_1 \\ -o_x\sin\theta_1 + o_y\cos\theta_1 \\ -a_x\sin\theta_1 + a_y\cos\theta_1 \end{bmatrix} = \begin{bmatrix} -\sin\theta_5\cos\theta_6 \\ \sin\theta_5\sin\theta_6 \\ -\cos\theta_5 \end{bmatrix} $$

Finally, by using the first and second row of the equality above, we can compute the joint angle θ_6 from the equation below:

$$ \frac{\sin\theta_6}{\cos\theta_6} = \frac{\dfrac{-o_x\sin\theta_1 + o_y\cos\theta_1}{\sin\theta_5}}{\dfrac{n_x\sin\theta_1 - n_y\cos\theta_1}{\sin\theta_5}} $$

$$ \tan\theta_6 = \frac{\dfrac{-o_x\sin\theta_1 + o_y\cos\theta_1}{\sin\theta_5}}{\dfrac{n_x\sin\theta_1 - n_y\cos\theta_1}{\sin\theta_5}} $$

$$ \theta_6 = \mathrm{atan2}\left(\frac{-o_x\sin\theta_1 + o_y\cos\theta_1}{\sin\theta_5}, \frac{n_x\sin\theta_1 - n_y\cos\theta_1}{\sin\theta_5} \right) \tag{8.16} $$

It is important to note that if the J_{2z}, J_{3z}, J_{4z} and J_{6z} axes are aligned ($\sin\theta_5 = 0$), the solution is undetermined (too many degrees of freedom), and in the same way if both of the numerators in equation (8.16) are zero. In this situation, it is possible to rotate the end-effector (frame J_6) around the J_{6z} axis without moving it, by only rotating the J_{2z}, J_{3z} and J_{4z} axes. Here, the joint 6 becomes redundant, so that we can set it to an arbitrary value.

8.3.2.4 Computation of the angle θ_3

As stated above, the J_{2z}, J_{3z} and J_{4z} axes are always parallel. Therefore, the frames J_2, J_3 and J_4, viewed from the plane J_{1xy}, constitute a robot planar manipulator, as shown in Figure 8.19. To find the joint angles θ_2, θ_3 and θ_4; we simulate the robot movement around the joint 1 and we analyze the joint 4 position relating to the frame J_1 ($^1p_4^{xz}$).

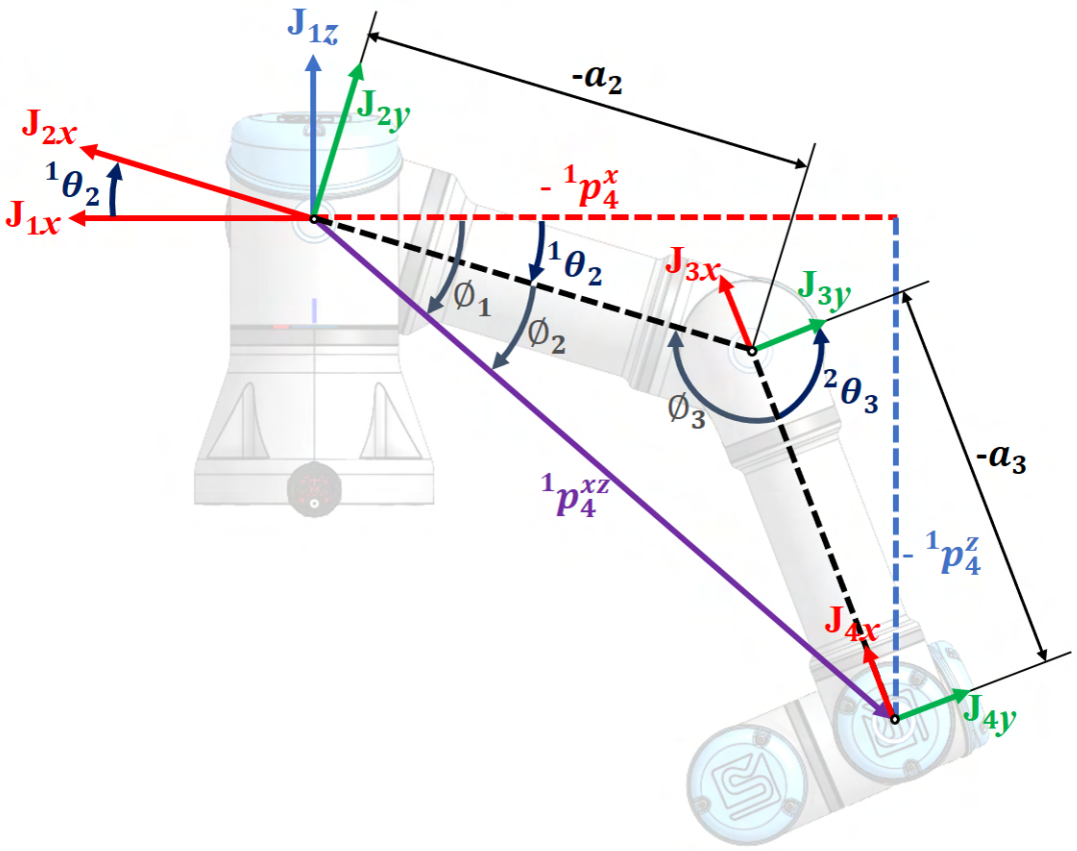

Figure 8.19: Robot planar manipulator - J_{234}

From the graph, we can compute the angle ϕ_3 using the law of cosine as:

$$\cos\phi_3 = \frac{(-a_2)^2 + (-a_3)^2 - |^1p_4^{xz}|^2}{2(-a_2)(-a_3)} = \frac{{a_2}^2 + {a_3}^2 - |^1p_4^{xz}|^2}{2a_2a_3}$$

Since $\theta_3 = (\pi - \phi_3)$, thus $\cos\theta_3 = -\cos\phi_3$; and $|^1p_4^{xz}|^2 = |^1p_4^x|^2 + |^1p_4^z|^2$, we can find the joint angle θ_3 from the equation below:

$$\cos\theta_3 = \frac{|^1p_4^x|^2 + |^1p_4^z|^2 - {a_2}^2 - {a_3}^2}{2a_2a_3}$$

$$\theta_3 = \pm\arccos\left(\frac{|^1p_4^x|^2 + |^1p_4^z|^2 - {a_2}^2 - {a_3}^2}{2a_2a_3}\right) \tag{8.17}$$

It is important to point out that $|^1p_{4x}|^2$ and $|^1p_{4z}|^2$ can be found calculating the homogeneous transformation ${}^1\mathrm{T}_4$, which we can find based on the joint angles already calculated θ_1, θ_5 and θ_6 as:

$$ {}^1\mathrm{T}_4 = ({}^0\mathrm{T}_1)^{-1}({}^0\mathrm{T}_6)({}^5\mathrm{T}_6)^{-1}({}^4\mathrm{T}_5)^{-1} \tag{8.18} $$

Where ${}^0\mathrm{T}_6$ is the desired end-effector location; and ${}^0\mathrm{T}_1, {}^5\mathrm{T}_6$ and ${}^4\mathrm{T}_5$ can be expressed as:

$$ {}^0\mathrm{T}_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^5\mathrm{T}_6 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ -\sin\theta_6 & -\cos\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^4\mathrm{T}_5 = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & 0 & 0 \\ 0 & 0 & -1 & -d_5 \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

It is important to note that the solution exists as long as $|^1p_4^{xz}| \in [|a_2 - a_3|; |a_2 + a_3|]$. The equation (8.17) represents the two solutions for the joint 3, which correspond to the elbow is “up” or “down”, as shown in Figure 8.20.

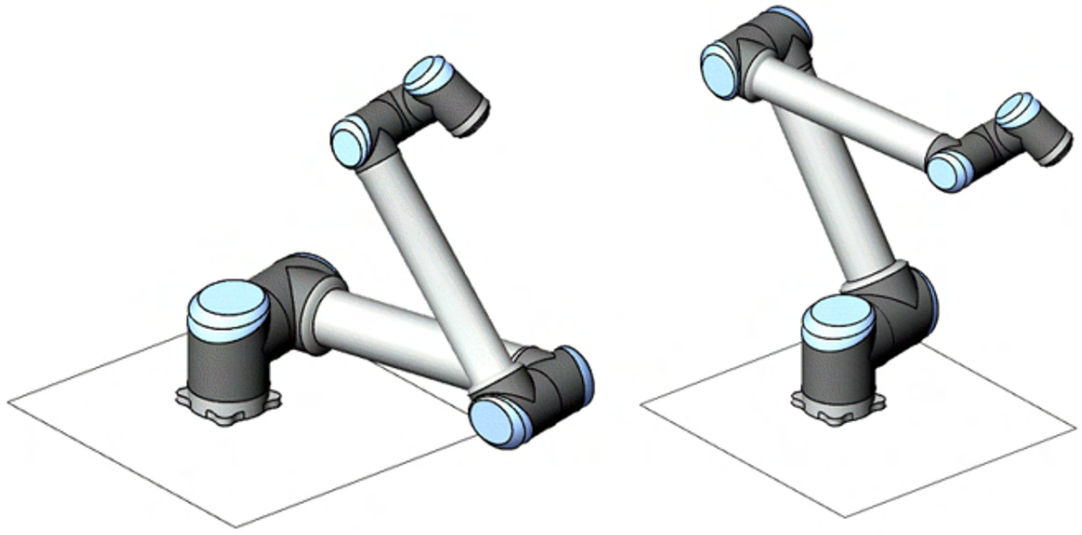

Figure 8.20: Solutions for the joint angle θ_3 [66]

8.3.2.5 Computation of the angle θ_2

According to Figure 8.19, we can find the joint angle θ_2 from:

$$\theta_2 = \phi_1 - \phi_2 \tag{8.19}$$

Where we can find ϕ_2 using the law of sine as:

$$\frac{\sin\phi_2}{-a_3} = \frac{\sin\phi_3}{|{}^1p_4^{xz}|}$$

$$\phi_2 = \arcsin\left(\frac{-a_3\sin\phi_3}{|{}^1p_4^{xz}|}\right) \tag{8.20}$$

And ϕ_1 using a tangent relation as:

$$\phi_1 = \text{atan2}(-{}^1p_4^z, -{}^1p_4^x) \tag{8.21}$$

By replacing (8.20) and (8.21) in (8.19), and considering that $\sin\phi_3 = \sin\theta_3$, we can compute the joint angle θ_2 from the equation below:

$$\theta_2 = \text{atan2}(-{}^1p_4^z, -{}^1p_4^x) - \arcsin\left(\frac{-a_3\sin\theta_3}{|{}^1p_4^{xz}|}\right) \tag{8.22}$$

8.3.2.6 Computation of the angle θ_4

We can find the joint angle θ_4 by means the homogeneous transformation ${}^3T_4$, which we can compute as:

$${}^3\mathrm{T}_4 = ({}^2\mathrm{T}_3)^{-1}({}^1\mathrm{T}_2)^{-1}({}^0\mathrm{T}_1)^{-1}({}^0\mathrm{T}_6)({}^5\mathrm{T}_6)^{-1}({}^4\mathrm{T}_5)^{-1} = \begin{bmatrix} {}^3n_4^x & {}^3o_4^x & {}^3a_4^x & {}^3p_4^x \\ {}^3n_4^y & {}^3o_4^y & {}^3a_4^y & {}^3p_4^y \\ {}^3n_4^z & {}^3o_4^z & {}^3a_4^z & {}^3p_4^z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.23}$$

Where ${}^2\mathrm{T}_3$ and ${}^1\mathrm{T}_2$ can be expressed as:

$${}^1\mathrm{T}_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^2\mathrm{T}_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & a_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, we can compute the joint angle θ_4 from the equation below:

$$\theta_4 = \text{atan2}({}^3n_4^y, {}^3n_4^x) \tag{8.24}$$

In accordance with the geometric IK solution, we have 8 robot configurations for the analytical IK solution for the UR3e robot: $2_{\theta_1} \times 2_{\theta_5} \times 1_{\theta_6} \times 2_{\theta_3} \times 1_{\theta_2} \times 1_{\theta_4}$. We summarize all the solutions found for each joint below:

$$\theta_1 = \text{atan2}({}^0p_{5y}, {}^0p_{5x}) \pm \arccos\left(\frac{d_4}{\sqrt{{}^0p_5^{y2} + {}^0p_5^{x2}}}\right) + \frac{\pi}{2}$$

$$\theta_2 = \text{atan2}(-{}^1p_4^z, -{}^1p_4^x) - \arcsin\left(\frac{-a_3 \sin\theta_3}{|{}^1p_4^{xz}|}\right)$$

$$\theta_3 = \pm \arccos\left(\frac{|{}^1p_4^x|^2 + |{}^1p_4^z|^2 - {a_2}^2 - {a_3}^2}{2a_2a_3}\right)$$

$$\theta_4 = \text{atan2}({}^3n_4^y, -{}^3n_4^x)$$

$$\theta_5 = \pm \arccos\left(\frac{{}^0p_6^x \sin\theta_1 - {}^0p_6^y \cos\theta_1 - d_4}{d_6}\right)$$

$$\theta_6 = \text{atan2}\left(\frac{-o_x \sin\theta_1 + o_y \cos\theta_1}{\sin\theta_5}, \frac{n_x \sin\theta_1 - n_y \cos\theta_1}{\sin\theta_5}\right)$$

8.4 Velocity Kinematics - The manipulator Jacobian

In Sections 8.2 and 8.3, we found the forward and inverse transformation models which allow us to establish a relation between the joints angles and the end-effector location. In this section, we seek to find a transformation model relating the linear and angular velocities of the end-effector and the joints velocities, where the angular velocity refers to the angular velocity of the end-effector frame, and the linear velocity refers to the velocity of its origin. Mathematically, the forward kinematics equations define a function between the operational space and the joint space. So, the required transformation model becomes the Jacobian of this function. This Jacobian is a matrix-valued function, and is one of the most important quantities in the analysis and control of robot motion, applied in the planning and execution of smooth trajectories, determination of singular configurations, execution of coordinated anthropomorphic motion, derivation of the dynamics equation of motion, and the transformation of forces and torques from the end-effector to the manipulator joints [64].
As said above, the Jacobian matrix implies a relationship between velocities, therefore we will begin this section with a description about velocity concepts and its representations. First, we will analyze the angular velocity around a fixed axis, and then we will generalize this case to a general case using the skew symmetric matrices. With this general representation of angular velocities, we can derive the equations for the angular and linear velocity of a moving frame. Finally, we can derive the Jacobian ($6 \times n$ matrix), which represents the instantaneous transformations between the joint velocities n-vector and 6-vector comprising the linear and angular velocities of the end-effector [64].

8.4.1 Angular velocity: The fixed axis case

If we apply a pure rotation to a rigid body around a fixed axis, every point belonging to the body will move in a circle which centers lie on the axis rotation. Here, a vector r from the origin (assumed to lie on the axis rotation) to every point in the body, sweeps out and angle θ, as shown in Figure 8.21.
In this case, if k is a unit vector in the direction of the rotation axis, we can describe the angular velocity as:

$$\omega = \dot{\theta} k \tag{8.25}$$

Where $\dot{\theta}$ is the time derivative of θ. Likewise, given the angular velocity ω, we can compute the linear velocity of any point in the body as:

$$v = \omega \times r \tag{8.26}$$

If we analyze the equation (8.26), we can state that the major role of the angular velocity is to induce a linear velocity in every point of the rigid body, however our aim is to describe the motion of a moving frame, including the linear motion of the frame origin and the rotational

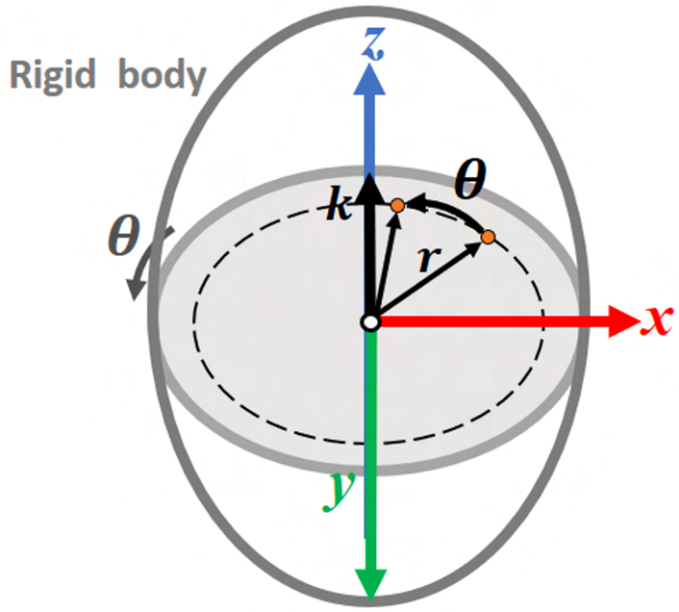

Figure 8.21: Pure rotation about a fixed axis

motion of the frame's axes, thus linear and angular velocities will have same importance.
As seen in the previous sections, we define the rigid body orientation from the orientation of a coordinate frame attach rigidly to the object. Then, since that every point in the body is in a fixed geometric relationship to this body-attached frame, and that every point experiments the same angular velocity, we can state that the angular velocity of the body only depends on the angular velocity of the attached coordinate frame itself, while the individual points in the body only experiments a linear velocity induced by this angular velocity. Therefore, in the equation (8.26), υ refers to the linear velocity of a point, while ω refers to the angular velocity of a rotating coordinate frame.
So far, we have made a description about the angular velocity of a rigid body around a fixed axis. However, in this case, we are specifying the angular displacement in a plane (planar problem), so that it is not possible to consider using theta to represent the angular velocity in the three-dimensional case, either when the rotation axis is not fixed, or the angular velocity results from multiple rotations around distinct axes. Therefore, we need to develop a general representation for angular velocities, analogous to the development of rotation matrices to represent orientation in all three dimensions. And the key tool for this is the skew-symmetric matrix, thus we will detail these matrices below.

8.4.1.1 Skew symmetric matrices

The computation of relative velocity transformations between coordinate frames implies the use of derivatives of rotation matrices. However, by introducing the skew-symmetric matrix concept, we can simplify it. From this section, we will consider a matrix S as skew-symmetric matrix if and only if it satisfies the equation below:

$$\mathrm{S} + \mathrm{S}^{\mathrm{T}} = 0 \tag{8.27}$$

We denote all the 3×3 skew symmetric matrices by SS(3). So, if S $\in$ SS(3), S has the components s_{ij} $(i, j = 1, 2, 3)$. Then the equation (8.27) is equivalent to the nine equations below:

$$\mathrm{s}_{ij} + \mathrm{s}_{ji} = 0 \quad i, j = 1, 2, 3 \tag{8.28}$$

From the equation (8.28), we can to note that the diagonal terms of S are zero $(\mathrm{s}_{ii} = 0)$, while the off-diagonal terms $(\mathrm{s}_{ij}, i \neq j)$ satisfy $\mathrm{s}_{ij} = -\mathrm{s}_{ij}$. Then, S has the form below:

$$\mathrm{S} = \begin{bmatrix} 0 & -\mathrm{s}_3 & \mathrm{s}_2 \\ \mathrm{s}_3 & 0 & -\mathrm{s}_1 \\ -\mathrm{s}_2 & \mathrm{s}_1 & 0 \end{bmatrix}$$

Finally, if $\vec{a} = (a_x, a_y, a_z)^{\mathrm{T}}$, we define the skew-symmetric matrix $\mathrm{S}(a)$ as:

$$\mathrm{S} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \tag{8.29}$$

For example, let i, j and k be the standard basis vectors:

$$i = (1, 0, 0)^{\mathrm{T}} \quad j = (0, 1, 0)^{\mathrm{T}} \quad k = (0, 0, 1)^{\mathrm{T}}$$

By using the definition (8.29), the skew symmetric matrices $\mathrm{S}(i), \mathrm{S}(j)$ and $\mathrm{S}(k)$ will be the below:

$$\mathrm{S}(i) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathrm{S}(j) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad \mathrm{S}(k) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{8.30}$$

8.4.1.2 Properties of Skew symmetric matrices

The purpose of using skew symmetric matrices is to take advantage of its properties, which will facilitate subsequent calculations. These properties are [64]:

1. S($\alpha a + \beta b$) = αS(a) + βS(b); where the vectors $a, b \in \mathrm{R}^3$ and α, β are scalars.
2. S(a)b = $a \times b$
3. R($a \times b$) = R$a\times$ Rb; where the matrix R $\in$ SO(3), and must be orthogonal.
4. RS(a)R^{T} = S(Ra)b

We suppose R is a rotation matrix in function of the angle θ (R(θ) $\in$ SO(3)). Since R is orthogonal, then:

$$\mathrm{R}(\theta)\mathrm{R}(\theta)^{\mathrm{T}} = \mathrm{I} \tag{8.31}$$

Now, by differentiating (8.31) relating to θ using the product rules as follows:

$$\frac{d\mathrm{R}}{d\theta}\mathrm{R}(\theta)^{\mathrm{T}} + \mathrm{R}(\theta)\frac{d\mathrm{R}(\theta)^{\mathrm{T}}}{d\theta} = 0$$

And defining the matrix S and its transpose S^{T} as:

$$\mathrm{S} = \frac{d\mathrm{R}}{d\theta}\mathrm{R}(\theta)^{\mathrm{T}} \qquad \mathrm{S}^{\mathrm{T}} = \left(\frac{d\mathrm{R}}{d\theta}\mathrm{R}(\theta)^{\mathrm{T}}\right)^{\mathrm{T}} = \mathrm{R}(\theta)\frac{d\mathrm{R}(\theta)^{\mathrm{T}}}{d\theta} \tag{8.32}$$

We can note that the matrix S satisfies the equation (8.27), $\mathrm{S} + \mathrm{S}^{\mathrm{T}} = 0$. Therefore, the matrix S defined in (8.32) is a skew-symmetric matrix. Finally, by multiplying both sides of the matrix S on the right by $\mathrm{R}(\theta)$, we have the equation below:

$$\frac{d\mathrm{R}}{d\theta} = \mathrm{SR}(\theta) \tag{8.33}$$

The equation (8.33) states that the derivative of the rotation matrix R is equivalent to its matrix multiplication by a skew-symmetric matrix S, where R commonly is a basic rotation matrix or a product of basis rotation matrices. Regarding the matrix S, if R is the basis rotation matrix about the x-axis $(\mathrm{R}_{x,\theta})$, we can compute its corresponding matrix S using the definition in (8.32) as:

$$\mathrm{S} = \frac{d\mathrm{R}_{x,\theta}}{d\theta} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\sin\theta & -\cos\theta \\ 0 & \cos\theta & -\sin\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{8.34}$$

We can notice that the matrix got in (8.34) corresponds to the skew-symmetric matrix S(i)

in (8.30), therefore we can state that:

$$\frac{d\mathrm{R}_{x,\theta}}{d\theta} = \mathrm{S}(i)\mathrm{R}_{x,\theta}$$

Similar computations with the basis rotation matrices about the y and z axes show the same relation:

$$\frac{d\mathrm{R}_{y,\theta}}{d\theta} = \mathrm{S}(j)\mathrm{R}_{y,\theta} \qquad \frac{d\mathrm{R}_{z,\theta}}{d\theta} = \mathrm{S}(k)\mathrm{R}_{z,\theta}$$

8.4.2 Angular velocity: The general case

In this section, we now will consider a general case in which the rigid body presents a rotation about an arbitrary and moving frame. Let R be a time varying matrix continuously differentiable as a function of t (R = R$(t) \in$ SO(3) for every $t \in \mathrm{R}$), we can compute its time derivative $\dot{\mathrm{R}}(t)$, recalling the equation (8.33) as:

$$\dot{\mathrm{R}}(t) = \mathrm{S}(t)\mathrm{R}(t) \tag{8.35}$$

The matrix S(t) is skew symmetric, thus we can represent (8.35) for a unique vector $\omega(t)$ as:

$$\dot{\mathrm{R}}(t) = \mathrm{S}(\omega(t))\mathrm{R}(t) \tag{8.36}$$

Where the vector $\omega(t)$ is the angular velocity of a rotating frame relating to a fixed frame at time t. Now analogous to the rotation matrices, we want to compute the resultant angular velocity because of the relative rotation of several coordinate frames. Then, we will derive the expression for the composition of angular velocities of two moving frames $o_1x_1y_1z_1$ and $o_2x_2y_2z_2$ relating to the fixed frame $o_0x_0y_0z_0$, considering that these frames share the same origin. Let the rotation matrices ${}^0\mathrm{R}_1(t)$ and ${}^1\mathrm{R}_2(t)$ be the relative orientations of the frames $o_1x_1y_1z_1$ and $o_2x_2y_2z_2$, then:

$${}^0\mathrm{R}_2(t) = {}^0\mathrm{R}_1(t){}^1\mathrm{R}_2(t) \tag{8.37}$$

Differentiating both sides of (8.37) relating to the time gives:

$${}^0\dot{\mathrm{R}}_2 = {}^0\dot{\mathrm{R}}_1{}^1\mathrm{R}_2 + {}^0\mathrm{R}_1{}^1\dot{\mathrm{R}}_2 \tag{8.38}$$

Now using (8.36), we can write the term ${}^0\dot{\mathrm{R}}_2$ on the left-hand side of (8.38) as:

$${}^0\dot{\mathrm{R}}_2 = \mathrm{S}({}^0\omega_2){}^0\mathrm{R}_2 \tag{8.39}$$

Where ${}^0\omega_2$ is the total angular velocity of the frame $o_2x_2y_2z_2$, which results from to combine the rotation ${}^0\mathrm{R}_1$ and ${}^1\mathrm{R}_2$. Regarding to the first term on the right-hand side of (8.38), we can compute it as:

$${}^0\dot{\mathrm{R}}_1{}^1\mathrm{R}_2 = \mathrm{S}({}^0\omega_a){}^0\mathrm{R}_1{}^1\mathrm{R}_2 = \mathrm{S}({}^0\omega_a){}^0\mathrm{R}_2 \tag{8.40}$$

Where ${}^0\omega_a$ is the angular velocity of the frame frame $o_1x_1y_1z_1$ that results from the rotation ${}^0\mathrm{R}_1$, and is expressed relating to the coordinate frame $o_0x_0y_0z_0$. Regarding to the second term on the right-hand side of (8.38), using the skew property (d) we can compute it as:

$$\begin{aligned}
&{}^0\mathrm{R}_1\,{}^1\dot{\mathrm{R}}_2 = {}^0\mathrm{R}_1\mathrm{S}({}^1\omega_b){}^1\mathrm{R}_2 \\
&{}^0\mathrm{R}_1\,{}^1\dot{\mathrm{R}}_2 = {}^0\mathrm{R}_1\mathrm{S}({}^1\omega_b)({}^0\mathrm{R}_1)^{\mathrm{T}}{}^0\mathrm{R}_1\,{}^1\mathrm{R}_2 \rightarrow \text{Using (d)} \\
&{}^0\mathrm{R}_1\,{}^1\dot{\mathrm{R}}_2 = \mathrm{S}({}^0\mathrm{R}_1\,{}^1\omega_b){}^0\mathrm{R}_1\,{}^1\mathrm{R}_2 \\
&{}^0\mathrm{R}_1\,{}^1\dot{\mathrm{R}}_2 = \mathrm{S}({}^0\mathrm{R}_1\,{}^1\omega_b){}^0\mathrm{R}_2
\end{aligned} \tag{8.41}$$

Where ${}^1\omega_b$ is the angular velocity of the frame $o_2x_2y_2z_2$ that results from the rotation ${}^1\mathrm{R}_2$, and is expressed relating to the coordinate frame $o_1x_1y_1z_1$. Therefore, the product ${}^0\mathrm{R}_1\,{}^1\omega_b$ represents the angular velocity of the frame $o_2x_2y_2z_2$ but relating to the coordinate frame $o_0x_0y_0z_0$. Now, by combining (8.41), (8.40) and (8.39) in (8.38), we can state that:

$$\mathrm{S}({}^0\omega_2){}^0\mathrm{R}_2 = \{\mathrm{S}({}^0\omega_a) + \mathrm{S}({}^0\mathrm{R}_1\,{}^1\omega_b){}^0\mathrm{R}_2\} \tag{8.42}$$

By using the skew property (a), we can see that:

$$\mathrm{S}({}^0\omega_2){}^0\mathrm{R}_2 = \{\mathrm{S}({}^0\omega_a + {}^0\mathrm{R}_1\,{}^1\omega_b){}^0\mathrm{R}_2\}$$

Therefore, we can state that:

$${}^0\omega_2 = {}^0\omega_a + {}^0\mathrm{R}_1\,{}^1\omega_b$$

That means that we can add angular velocities as long as they are expressed relating to the same coordinate frame, in this case $o_0x_0y_0z_0$. Now, we can extend the procedure above to many coordinate frames in such a way that if we have the concatenated rotation matrices below:

$${}^0\mathrm{R}_n = {}^0\mathrm{R}_1\,{}^1\mathrm{R}_2 \cdots {}^{n-1}\mathrm{R}_n$$

Where ${}^0\mathrm{R}_n$ represents the rotation of the n-coordinate frame relating to the station frame. Finally, let ${}^{n-1}\omega_n$ be the angular velocity related to the rotation matrix ${}^{n-1}\omega_n$, we can extend the equation (8.36) as:

$${}^0\dot{\mathrm{R}}_n = \mathrm{S}({}^0\omega_{\mathrm{n}}){}^0\mathrm{R}_n$$

Where:

$${}^0\omega_n = {}^0\omega_1 + {}^0\mathrm{R}_1\,{}^1\omega_2 + {}^0\mathrm{R}_2\,{}^2\omega_3 + \cdots + {}^0\mathrm{R}_{n-1}\,{}^{n-1}\omega_n \tag{8.43}$$

8.4.3 Linear velocity of a point attached to a moving frame

Let p be a point rigidly attached to the moving frame $o_1x_1y_1z_1$, which rotates relating to the frame $o_0x_0y_0z_0$. We can express its coordinates relating to the frame $o_0x_0y_0z_0$, considering that they share the same origin, as:

$$^0p = {}^0\mathrm{R}_1(t)\,{}^1p \tag{8.44}$$

Now, we can compute the velocity $^0\dot{p}$ differentiating (8.44), and using the equation (8.36) as:

$$\begin{aligned} {}^0\dot{p} &= {}^0\dot{\mathrm{R}}_1(t)\,{}^1p + {}^0\mathrm{R}_1(t)\,{}^1\dot{p} \rightarrow {}^1\dot{p} = 0 \\ {}^0\dot{p} &= \mathrm{S}({}^0\omega)\,{}^0\mathrm{R}_1(t)\,{}^1p \\ {}^0\dot{p} &= \mathrm{S}({}^0\omega)\,{}^0p \end{aligned} \tag{8.45}$$

Where $^1\dot{p} = 0$ since p is rigidly attached to the frame $o_1x_1y_1z_1$, thus its coordinates relating to the frame $o_1x_1y_1z_1$ do not change. Using the skew property (b), we can reduce (8.45) to:

$$^0\dot{p} = {}^0\omega \times {}^0p \tag{8.46}$$

Which is familiar to the expression (8.26) in terms of the vector cross product. Now, for a general description of the linear velocity, we consider that there is a time varying displacement of frame $o_1x_1y_1z_1$ origin relating to the frame $o_0x_0y_0z_0$ origin $(^0o_1(t))$. Then the equation (8.44) becomes:

$$^0p = {}^0\mathrm{R}_1(t)\,{}^1p + {}^0o_1(t) \tag{8.47}$$

By differentiating the equation (8.47), using the product rule, gives:

$$\begin{aligned} {}^0\dot{p} &= {}^0\mathrm{R}_1(t)\,{}^1p + {}^0o_1(t) \\ {}^0\dot{p} &= \mathrm{S}({}^0\omega)\,{}^0\mathrm{R}_1(t)\,{}^1p + {}^0\dot{o}_1(t) \\ {}^0\dot{p} &= {}^0\omega \times {}^0p + {}^0v_1 \end{aligned} \tag{8.48}$$

Where $^0p = {}^0\mathrm{R}_1(t)\,{}^1p$ is the vector position from the origin o_1 to p, expressed relating to the frame $o_0x_0y_0z_0$; and $^0v_1$ is the rate at which the origin o_1 is moving relating to the frame $o_0x_0y_0z_0$. If p also has a movement relating to the frame $o_1x_1y_1z_1$ $(^1\dot{p} \neq 0)$, then we must consider the term $^1\dot{p}$ in (8.48), which is the rate of change of the p coordinates relating to the frame $o_0x_0y_0z_0$.

8.4.4 Jacobian derivation

For an n-link robot manipulator, the homogeneous transformation, which denotes the transformation from the end-effector frame to the station frame, has the form below:

$$ {}^{0}\mathrm{T}_{n}(\vec{q}) = \begin{bmatrix} {}^{0}\mathrm{R}_{n}(\vec{q}) & {}^{0}p_{n}(\vec{q}) \\ \vec{0} & 1 \end{bmatrix} $$

Where $\vec{q} = (q_1, q_2, \cdots q_n)^{\mathrm{T}}$ is the joint variables vector. Since the robot movement varies over time; $\vec{q}$, ${}^{0}p_{n}$ and ${}^{0}\mathrm{R}_{n}$ will be functions of time. Then, we can define the equations below:

$$ \mathrm{S}({}^{0}\omega_{n}) = {}^{0}\dot{\mathrm{R}}_{n}({}^{0}\mathrm{R}_{n})^{\mathrm{T}} \tag{8.49} $$

$$ {}^{0}\upsilon_{n} = {}^{0}\dot{p}_{n} \tag{8.50} $$

Where the equation (8.49) defines the angular velocity vector of the end-effector $({}^{0}\omega_{n})$ relating to the station frame, and (8.50) denotes the end-effector linear velocity. However, since we seek a model transformation model relating the linear and angular velocities of the end-effector and the joint velocities, we seek expressions of the form below:

$$ {}^{0}\upsilon_{n} = \mathrm{J}_{\upsilon}\dot{\vec{q}} \tag{8.51} $$

$$ {}^{0}\omega_{n} = \mathrm{J}_{\omega}\dot{\vec{q}} \tag{8.52} $$

Where J_{υ} and J_{ω} are $3 \times n$ matrices. We can express (8.51) and (8.52) together as:

$$ \begin{bmatrix} {}^{0}\upsilon_{n} \\ {}^{0}\omega_{n} \end{bmatrix} = {}^{0}\mathrm{J}_{n}\dot{\vec{q}} \tag{8.53} $$

Where ${}^{0}\mathrm{J}_{n}$ is given by:

$$ {}^{0}\mathrm{J}_{n} = \begin{bmatrix} \mathrm{J}_{\upsilon} \\ \mathrm{J}_{\omega} \end{bmatrix} \tag{8.54} $$

Finally, ${}^{0}\mathrm{J}_{n}$ is the well-known manipulator Jacobian or Jacobian matrix. Therefore, we seek to find this matrix, which is a $6 \times n$ matrix where n is the number of links. We will define the procedure to compute J_{ω} and J_{υ} below, where, since we focus the analysis in the UR robot family, the join variable q_i is the joint angle θ_i.

8.4.4.1 Computation of J_ω

Following the DH convention, we can define the angular velocity of the link i relating to the frame $o_{i-1}x_{i-1}y_{i-1}z_{i-1}$, which is imparted by the rotation of the join i, as:

$$^{i-1}\omega_i = \dot{\theta}_i {}^{i-1}z_{i-1} = \dot{\theta}_i k$$

Where k is the unit vector $(0,0,1)^{\mathrm{T}}$. Then, recalling the equation (8.43), we can state that is possible to determine the angular velocity of the end-effector relating to the station frame $(^0\omega_n)$ by computing the angular velocity contributed by each joint relating to the station frame and then adding them. Therefore, we can determine $^0\omega_n$ from the equation below:

$$^0\omega_n = \dot{\theta}_1 k + \dot{\theta}_2 {}^0\mathrm{R}_1 k + \cdots + \dot{\theta}_n {}^0\mathrm{R}_{n-1} k \tag{8.55}$$

Now, since:

$$^0z_{i-1} = {}^0\mathrm{R}_{i-1} k$$

Then, we can define the equation (8.55) as:

$$^0\omega_n = \dot{\theta}_1 {}^0z_0 + \dot{\theta}_2 {}^0z_1 + \cdots + \dot{\theta}_n {}^0z_{n-i} = \begin{bmatrix} {}^0z_0 & {}^0z_1 & \cdots & {}^0z_{n-1} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \vdots \\ \dot{\theta}_n \end{bmatrix}$$

Therefore, we can compute J_ω from:

$$\mathrm{J}_\omega = \begin{bmatrix} {}^0z_0 & {}^0z_1 & \cdots & {}^0z_{n-1} \end{bmatrix} \tag{8.56}$$

8.4.4.2 Computation of J_υ

We know we can compute the homogeneous transformation ^0T_n by concatenating the intermediate transformations between the end-effector and the station frames. Therefore, we can write $^0\mathrm{T}_n$ as:

$$^0\mathrm{T}_n = {}^0\mathrm{T}_{i-1} {}^{i-1}\mathrm{T}_i {}^i\mathrm{T}_n \tag{8.57}$$

By resolving (8.57), we get that:

$$\begin{bmatrix} {}^0\mathrm{R}_n & {}^0o_n \\ \vec{0} & 1 \end{bmatrix} = \begin{bmatrix} {}^0\mathrm{R}_{i-1} & {}^0o_{i-1} \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} {}^{i-1}\mathrm{R}_i & {}^{i-1}o_i \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} {}^i\mathrm{R}_n & {}^io_n \\ \vec{0} & 1 \end{bmatrix}$$

$$\begin{bmatrix} {}^0\mathrm{R}_n & {}^0o_n \\ \vec{0} & 1 \end{bmatrix} = \begin{bmatrix} {}^0\mathrm{R}_n & {}^0\mathrm{R}_i {}^io_n + {}^0\mathrm{R}_{i-1} {}^{i-1}o_i + {}^0o_{i-1} \\ \vec{0} & 1 \end{bmatrix}$$

Therefore, we can define ${}^0o_n$ from the equation below:

$$ {}^0o_n = {}^0\mathrm{R}_i{}^i o_n + {}^0\mathrm{R}_{i-1}{}^{i-1} o_i + {}^0o_{i-1} \tag{8.58} $$

Then, we can compute the linear velocity of the end-effector $({}^0\dot{o}_n)$, using the chain rule for differentiation in (8.58), as:

$$ {}^0\dot{o}_n = \sum_{i=1}^{n} \frac{\partial^0 o_n}{\partial \theta_i} \dot{q}_i \tag{8.59} $$

From (8.59), we can define the i-th column of J_v as:

$$ \mathrm{J}_{vi} = \frac{\partial^0 o_n}{\partial \theta_i} \tag{8.60} $$

Now, by computing (8.60) using the equation (8.58), we get:

$$ \begin{aligned}
\frac{\partial^0 o_n}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i}[{}^0\mathrm{R}_i{}^i o_n + {}^0\mathrm{R}_{i-1}{}^{i-1} o_i] \\
\frac{\partial^0 o_n}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i}{}^0\mathrm{R}_i{}^i o_n + {}^0\mathrm{R}_{i-1}\frac{\partial}{\partial \theta_i}{}^{i-1} o_i \\
\frac{\partial^0 o_n}{\partial \theta_i} &= \dot{\theta}_i \mathrm{S}({}^0 z_{i-1}){}^0\mathrm{R}_i{}^i o_n + \dot{\theta}_i \mathrm{S}({}^0 z_{i-1}){}^0\mathrm{R}_{i-1}{}^{i-1} o_i \\
\frac{\partial^0 o_n}{\partial \theta_i} &= \dot{\theta}_i \mathrm{S}({}^0 z_{i-1})[{}^0\mathrm{R}_i{}^i o_n + {}^0\mathrm{R}_{i-1}{}^{i-1} o_i] \\
\frac{\partial^0 o_n}{\partial \theta_i} &= \dot{\theta}_i \mathrm{S}({}^0 z_{i-1})({}^0 o_n - {}^0 o_{i-1}) \\
\frac{\partial^0 o_n}{\partial \theta_i} &= \dot{\theta}_i {}^0 z_{i-1} \times ({}^0 o_n - {}^0 o_{i-1})
\end{aligned} $$

Then, we can compute the i-th column J_{vi} as:

$$ \mathrm{J}_{vi} = {}^0 z_{i-1} \times ({}^0 o_n - {}^0 o_{i-1}) $$

Therefore, we can compute J_v from:

$$ \mathrm{J}_v = \begin{bmatrix} \mathrm{J}_{v1} & \mathrm{J}_{v2} & \cdots & \mathrm{J}_{vn} \end{bmatrix} \tag{8.61} $$

Finally, by replacing (8.61) and (8.56) in (8.54), we can compute the Jacobian matrix from the equation below:

$$ {}^0\mathrm{J}_n = \begin{bmatrix} {}^0 z_0 \times ({}^0 o_n - {}^0 o_0) & {}^0 z_1 \times ({}^0 o_n - {}^0 o_1) & \cdots & {}^0 z_{i-1} \times ({}^0 o_n - {}^0 o_{i-1}) \\ z_0 & z_1 & \cdots & z_{i-1} \end{bmatrix} \tag{8.62} $$

For a 6-DoF robot manipulator like the UR robots, we can define the Jacobian as:

$$\mathrm{J} = \begin{bmatrix} \mathrm{J}_{1v} & \mathrm{J}_{2v} & \mathrm{J}_{3v} & \mathrm{J}_{4v} & \mathrm{J}_{5v} & \mathrm{J}_{6v} \\ \mathrm{J}_{1\omega} & \mathrm{J}_{2\omega} & \mathrm{J}_{3\omega} & \mathrm{J}_{4\omega} & \mathrm{J}_{5\omega} & \mathrm{J}_{6\omega} \end{bmatrix} \tag{8.63}$$

Where:

$$\mathrm{J}_i = \begin{bmatrix} {}^0z_{i-1} \times ({}^0o_6 - {}^0o_{i-1}) \\ {}^0z_{i-1} \end{bmatrix} \tag{8.64}$$

8.4.5 Jacobian computation for UR robot family

For the UR robot family, the Jacobian will have the form defined in the equation (8.63), and we can compute each component of this matrix using (8.64), where ${}^0o_{i-1}$ can be obtained as:

$$ {}^0o_{i-1} = {}^0\mathrm{A}_1 {}^1\mathrm{A}_2 \cdots {}^{i-2}\mathrm{A}_{i-1} {}^0o_0 \tag{8.65} $$

And ${}^0z_{i-1}$ as:

$$ {}^0z_{i-1} = {}^0\mathrm{R}_1 {}^1\mathrm{R}_2 \cdots {}^{i-2}\mathrm{R}_{i-1} {}^0z_0 \tag{8.66} $$

Where ${}^0o_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}}$ and ${}^0z_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}}$. Following the above procedure, we show the general Jacobian matrix for the UR robot family below:

$$ \mathrm{J} = \begin{bmatrix} \mathrm{J}_{11} & \mathrm{J}_{12} & \mathrm{J}_{13} & \mathrm{J}_{14} & \mathrm{J}_{15} & \mathrm{J}_{16} \\ \mathrm{J}_{21} & \mathrm{J}_{22} & \mathrm{J}_{23} & \mathrm{J}_{24} & \mathrm{J}_{25} & \mathrm{J}_{26} \\ \mathrm{J}_{31} & \mathrm{J}_{32} & \mathrm{J}_{33} & \mathrm{J}_{34} & \mathrm{J}_{35} & \mathrm{J}_{36} \\ \mathrm{J}_{41} & \mathrm{J}_{42} & \mathrm{J}_{43} & \mathrm{J}_{44} & \mathrm{J}_{45} & \mathrm{J}_{46} \\ \mathrm{J}_{51} & \mathrm{J}_{52} & \mathrm{J}_{53} & \mathrm{J}_{54} & \mathrm{J}_{55} & \mathrm{J}_{56} \\ \mathrm{J}_{61} & \mathrm{J}_{62} & \mathrm{J}_{63} & \mathrm{J}_{64} & \mathrm{J}_{65} & \mathrm{J}_{66} \end{bmatrix} \tag{8.67} $$

Where:

$$ \begin{aligned} \mathrm{J}_{11} &= d_6 s_1 s_5 c_{234} - d_5 s_1 s_{234} - a_2 s_1 c_2 - a_3 s_1 c_{23} + d_6 c_1 c_5 + d_4 c_1 \\ \mathrm{J}_{12} &= (-a_2 s_2 + d_6 s_5 s_{234} - a_3 s_{23} + d_5 c_{234}) c_1 \\ \mathrm{J}_{13} &= (d_6 s_5 s_{234} - a_3 s_{23} + d_5 c_{234}) c_1 \\ \mathrm{J}_{14} &= (d_6 s_5 s_{234} + d_5 c_{234}) c_1 \\ \mathrm{J}_{15} &= -d_6 s_1 s_5 - d_6 c_1 c_5 c_{234} \\ \mathrm{J}_{16} &= 0 \\ \mathrm{J}_{21} &= d_6 s_1 c_5 + d_4 s_1 - d_6 s_5 c_1 c_{234} + d_5 s_{234} c_1 + a_2 c_1 c_2 - a_3 c_1 c_{23} \\ \mathrm{J}_{22} &= (-a_2 s_2 d_6 s_5 s_{234} - a_3 s_{23} + d_5 c_{234}) s_1 \\ \mathrm{J}_{23} &= (d_6 s_5 s_{234} - a_3 s_{23} + d_5 c_{234}) s_1 \\ \mathrm{J}_{24} &= (d_6 s_5 s_{234} + d_5 C_{234}) s_1 \\ \mathrm{J}_{25} &= -d_6 s_1 c_5 c_{234} + d_6 s_5 c_1 \\ \mathrm{J}_{26} &= 0 \\ \mathrm{J}_{31} &= 0 \end{aligned} $$

$$
\begin{aligned}
\mathrm{J}_{32} &= -d_6 s_5 c_{234} + d_5 s_{234} + a_2 c_2 + a_3 c_{23} \\
\mathrm{J}_{33} &= -d_6 s_5 c_{234} + d_5 s_{234} - a_3 c_{23} \\
\mathrm{J}_{34} &= -d_6 s_5 c_{234} + d_5 s_{234} \\
\mathrm{J}_{35} &= -d_6 s_{234} c_5 \\
\mathrm{J}_{36} &= 0 \\
\mathrm{J}_{41} &= 0 \\
\mathrm{J}_{42} &= s_1 \\
\mathrm{J}_{43} &= s_1 \\
\mathrm{J}_{44} &= s_1 \\
\mathrm{J}_{45} &= s_{234} c_1 \\
\mathrm{J}_{46} &= s_1 c_5 - s_5 c_1 c_{234} \\
\mathrm{J}_{51} &= 0 \\
\mathrm{J}_{52} &= -c_1 \\
\mathrm{J}_{53} &= -c_1 \\
\mathrm{J}_{54} &= -c_1 \\
\mathrm{J}_{55} &= s_{234} s_1 \\
\mathrm{J}_{56} &= -c_1 c_5 - s_5 s_1 c_{234} - s_5 s_{234} \\
\mathrm{J}_{61} &= 1 \\
\mathrm{J}_{62} &= 0 \\
\mathrm{J}_{63} &= 0 \\
\mathrm{J}_{64} &= 0 \\
\mathrm{J}_{65} &= -c_{234} \\
\mathrm{J}_{66} &= -s_5 s_{234}
\end{aligned}
$$

It is important to point that the computed matrix in (8.67) is valid for the entire UR robot family; where $a_2, a_3, d_1, d_4, d_5, d_6$ are parameters that depend on the robot type.

8.5 Singularities

In Section 8.4.4, we defined the Jacobian matrix as the transformation model between the end-effector velocities and the joint velocities, and according to the equations in Section 8.4.5, we can state that this matrix depends on the robot configuration. However, there are certain robot configurations that lead us to scenarios known as singularities, in which the robot movement presents some restrictions. According to the ISO 10218 [70], a singularity scenario is a condition caused by the collinear alignment of two o more robot axes resulting in unpredictable motion and velocities. Mathematically, these singularity scenarios are described as those in which the rank of the Jacobian is less than the full rank, i.e. in which the determinant is zero, leading to the joint velocity to be infinite in order to keep the end-effector velocity. In the real world, end-effector motions passing near to the singularities can produce higher axis speed that can be unexpected to the operator. We can conclude that analyzing singularities in a robot is basically analyzing the range (or determinant) of the Jacobian matrix [64]. We will analyze the singularities for the UR robot family below.

8.5.1 Singularities computation

As said above, the singularities are configurations where the rank is less than the full rank, thus the condition for the manipulator being in a singularity is $\det(\mathrm{J}) = 0$. Then, first we have to compute the determinant of the Jacobian. By using the Jacobian computed in Section 8.4.5, we can compute its determinant from the equation below:

$$\det(\mathrm{J}) = -a_2 a_3 s_5 (0.5 a_2 s_{2-3} - 0.5 a_3 s_{23} - 0.5 d_5 c_{24} - 0.5 a_2 s_{23} + 0.5 d_5 c_{234} + 0.5 a_3 s_2) \quad (8.68)$$

The factorization of the equation (8.68) yields:

$$\det(\mathrm{J}) = (\sin\theta_5)(-a_2 \sin\theta_3)[a_3 \cos(\theta_2 + \theta_3) + a_2 \cos\theta_2 + d_5 \sin(\theta_2 + \theta_3 + \theta_4)] \quad (8.69)$$

Finally, by equaling the equation (8.69) to zero, we have mathematically expressed the three singularity scenarios [71].

8.5.1.1 Wrist singularity

The wrist singularity occurs when the axes joints 4 and 6 are parallel, as shown in Figure 8.22. Mathematically defined when $\sin\theta_5 = 0$. Therefore, there will be a wrist singularity when $\theta_5 = 0, \pm\pi$ or $\pm 2\pi$.

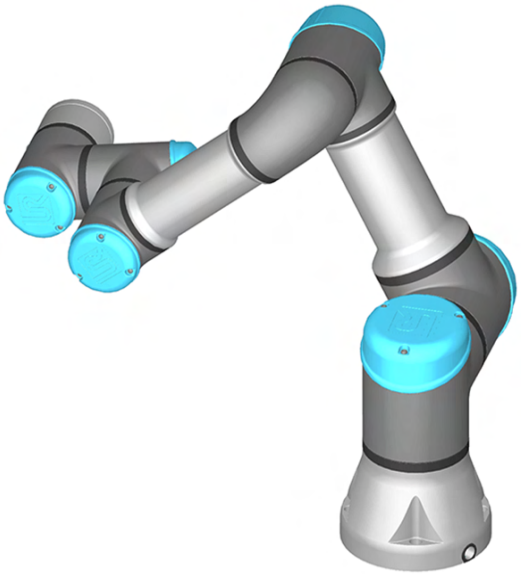

Figure 8.22: Wrist singularity for UR robot family [71]

8.5.1.2 Elbow singularity

The elbow singularity occurs when the arm is fully stretched, i.e. when the axes joint 2, 3 and 4 are coplanar, as shown in Figure 8.23. Mathematically defined when $\sin\theta_3 = 0$, i.e. when $\theta_3 = 0, \pm\pi$ or $\pm 2\pi$. However, since physically it is not possible to bring the joint 3 angle to $\pm\pi$ or $\pm 2\pi$, the only condition to the elbow singularity is $\theta_3 = 0$.

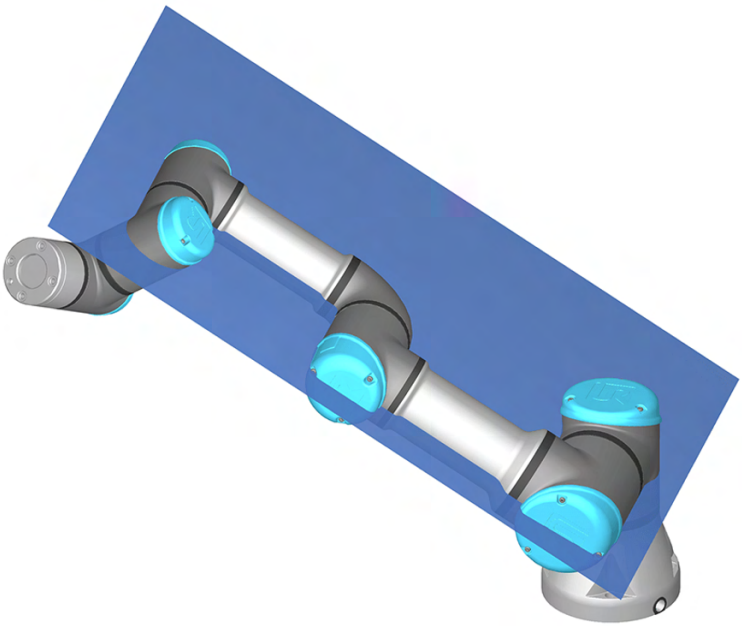

Figure 8.23: Elbow singularity for UR robot family [71]

8.5.1.3 Shoulder singularity

The shoulder singularity occurs when the intersection point of the joint axes 5 and 6 lie in one plane with the joint axes 1 and 2, as shown in Figure 8.24. Mathematically defined when $a_3 \cos(\theta_2 + \theta_3) + a_2 \cos\theta_2 + d_5 \sin(\theta_2 + \theta_3 + \theta_4) = 0$.

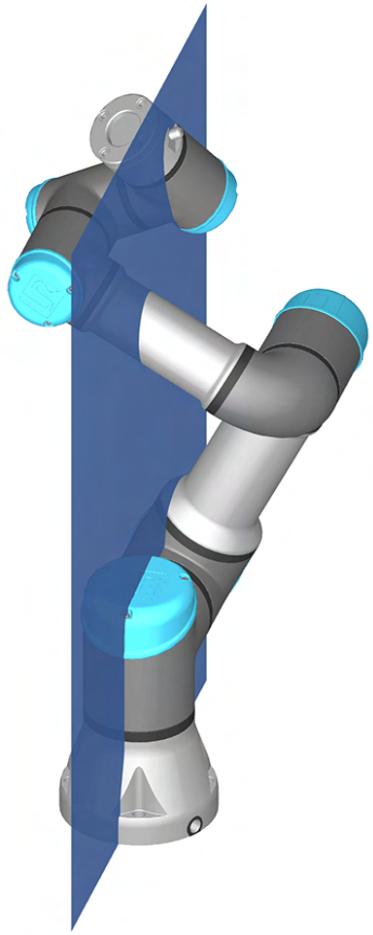

Figure 8.24: Shoulder singularity for UR robot family [71]

8.6 Singularities analysis

As mentioned in Section 8.5, motions near to the singularities produce higher axis speed unexpected to the operator. In this section, we will analyze mathematically the velocities behaviour of the joints when the robot passes through a singularity point.

8.6.1 Wrist singularity analysis

From Section 8.5.1 we already know that the wrist singularity occurs when the joints axes 4 and 6 are parallel. Therefore, viewed around axis alignment, we can force the wrist singularity passing the end-effector through a straight line keeping the same orientation, as shown in Figure 8.25.

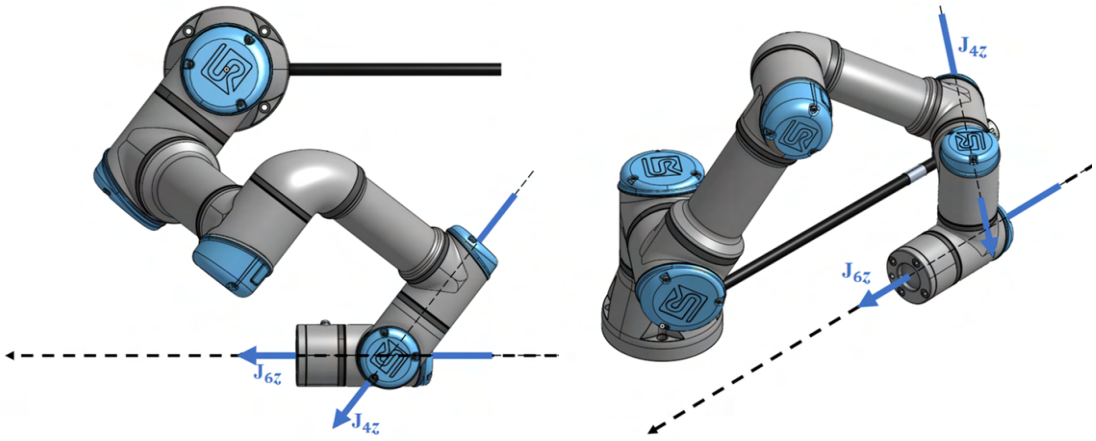

Figure 8.25: Motion bringing to the wrist singularity

To analyze the joints angles and velocities, we define a straight line composed of 1000 points (simulation steps) that goes from A $(0.340, 0.098, 0.330)$ to B $(0.340, -0.098, 0.330)$. If we associate each step with a target end-effector location required to follow the trajectory, then, recalling the IK formulas from Section 8.3.2, we will compute for each target location the required angle configuration. However, since we want to analyze the wrist singularity, first we will focus on the joint 5, since recalling the section 8.5.1.1, there will be a wrist singularity when $\theta_5 = 0, \pm\pi$ or $\pm 2\pi$. In Figure 8.26, we show the joint 5 angles.
From the graph above, we can to note that the dashed black line point out the simulation step (301) in which the target location requires that θ_5 approaches one of the singularity angles ($\theta_5 = \pm\pi$) in four of the solutions (from the 1 to the 4 solution). Therefore, we state that there is a singularity point. Now we will compute the remaining joints angles in order to observe the angles behaviour when the robot pass for the singularity point. In Figure 8.27, we show the joints angles.

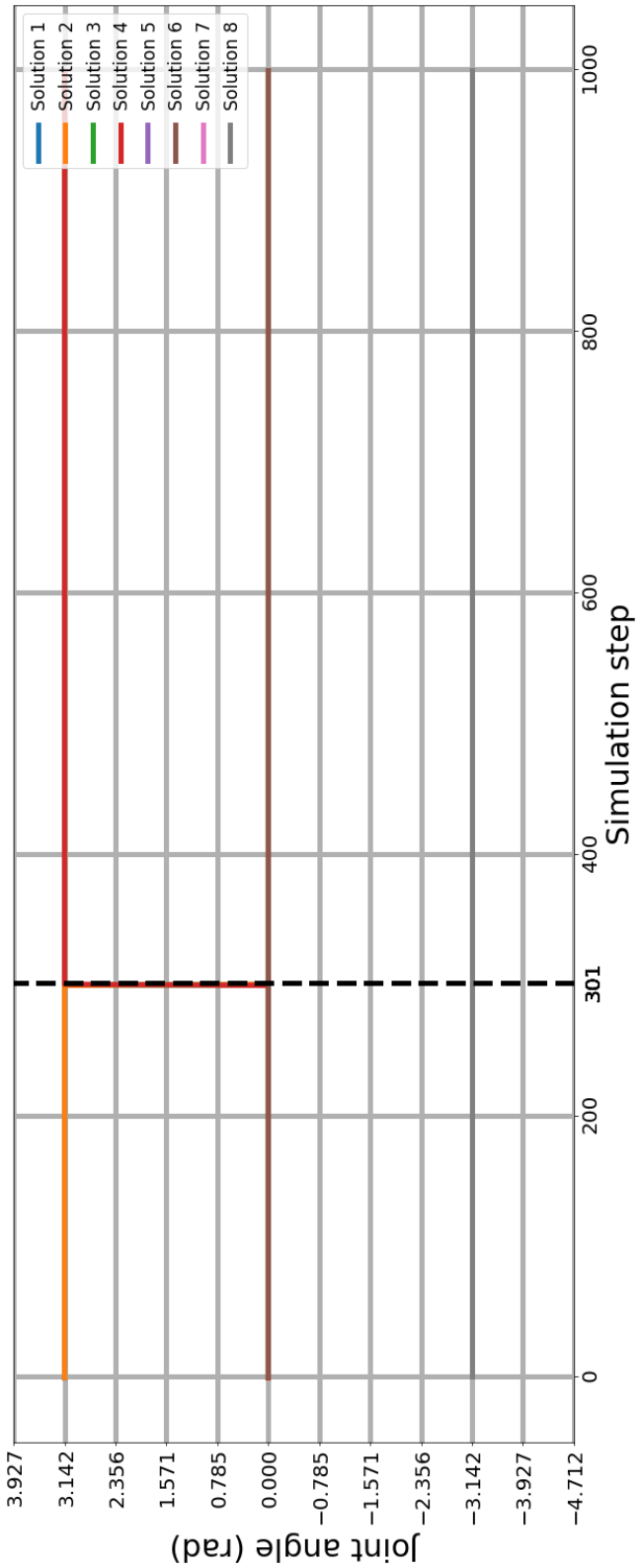

Figure 8.26: Joint 5 angles

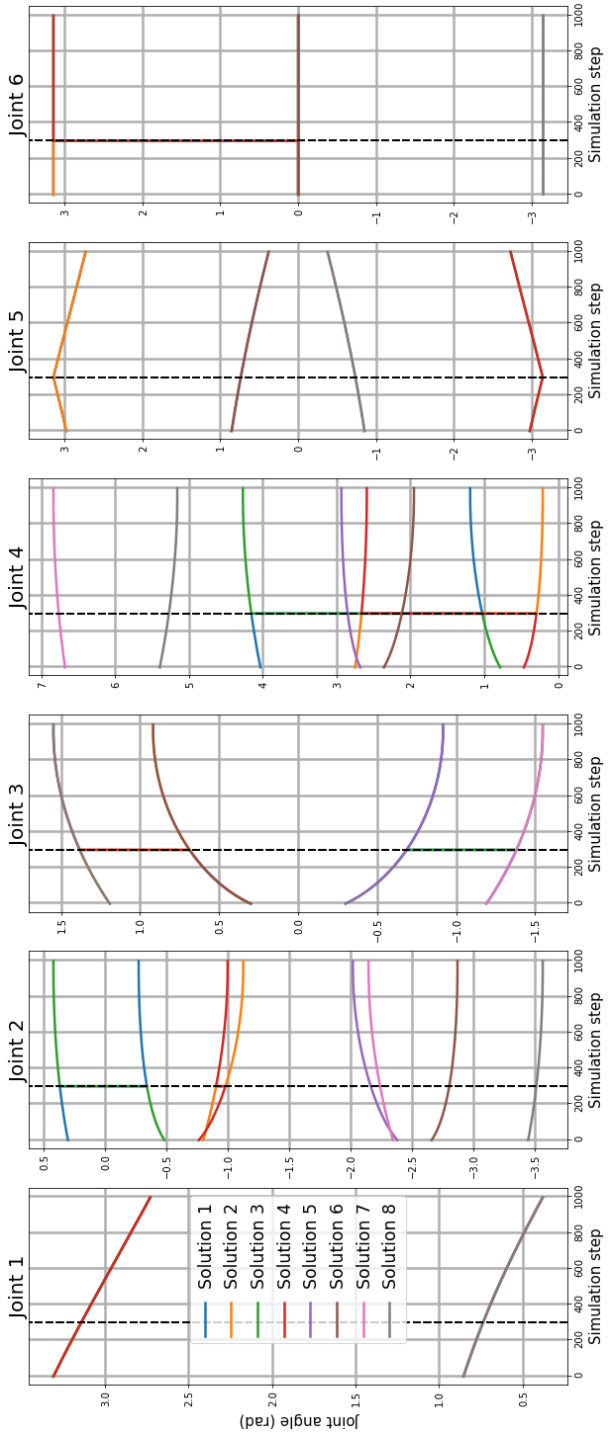

Figure 8.27: Joints angles - Wrist singularity case

In agreement with [72], we can observe that in the singularity point (dashed black line) the joints 4 and 6 spin 180 degrees ($\pm\pi$ rad) instantaneously. In addition, although not the same spin angle, since we compute the joints 2 and 3 angles from the joint 6 angle, we observe the same behaviour for these.

Now, we will analyze how this instantaneous spin affects the joints velocities. For that, from the joints angles, we compute the joints velocities using the equation (8.53) considering a constant linear and angular velocity of the end-effector. In Figure 8.28, we show the joints velocities.

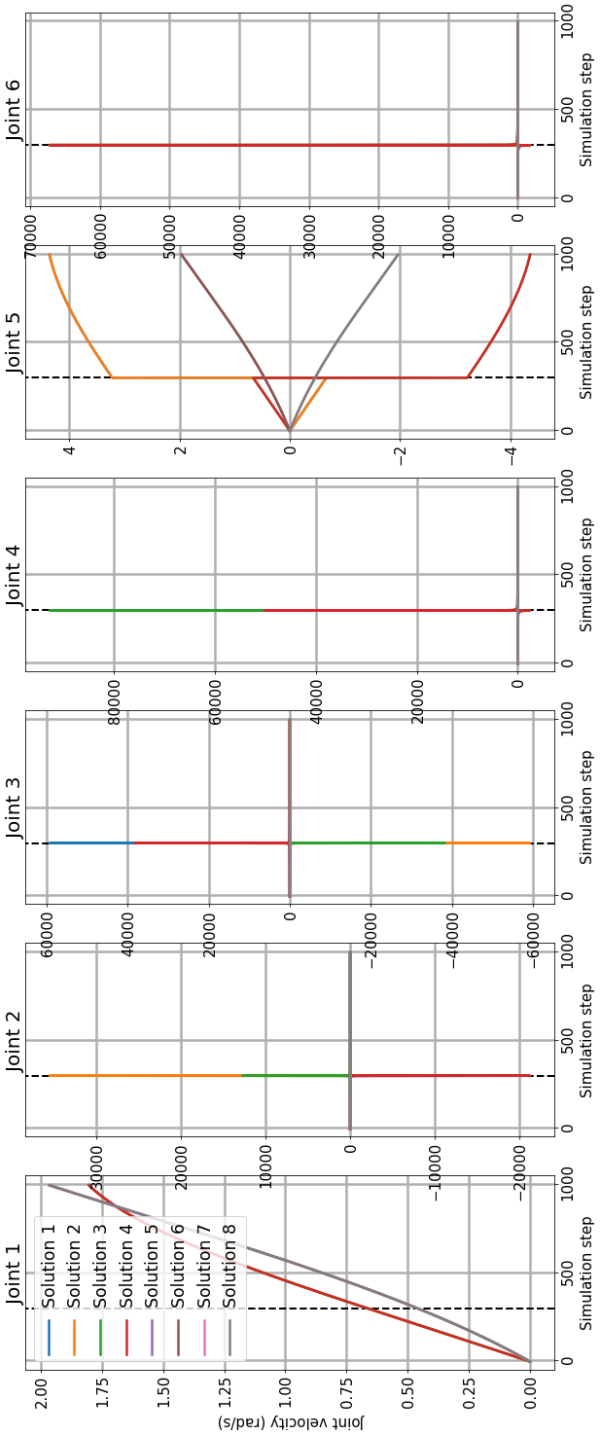

Figure 8.28: Joints velocities

As we can observe in the graph, the velocities of the joints 2, 3, 4 and 6 in the singularity point have very high velocity peaks. Therefore, in agreement with [72], we can state that at the singularity point, these velocities tend to infinity. To corroborate this statement, we will develop the same analysis, but for the elbow singularity.

8.6.2 Elbow singularity analysis

In the elbow's case singularity, we already know that occurs when the axes joint 2, 3 and 4 are coplanar, i.e. when the robot is fully stretched. Therefore, we can force the elbow singularity passing the end-effector through a straight line, keeping the same orientation, in such a way that we bring the end-effector to the full stretch condition as shown in Figure 8.29.

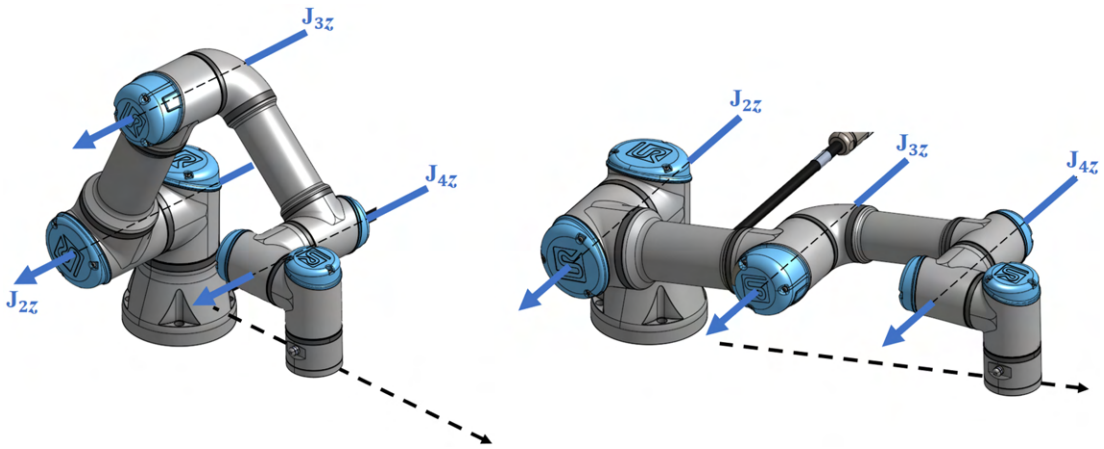

Figure 8.29: Motion bringing to the elbow singularity

Here, we define a straight line composed of 1000 points (simulation steps) that goes from A $(0.240, 0.0, 0.25)$ to B $(0.355, 0.0, 0.25)$. Recalling the Section 8.5.1.2, we know that there will be an elbow singularity when $\theta_3 = 0$. Therefore, following the procedure used in the wrist case, we will focus on this joint. In Figure 8.30, we show the joint 3 angles.

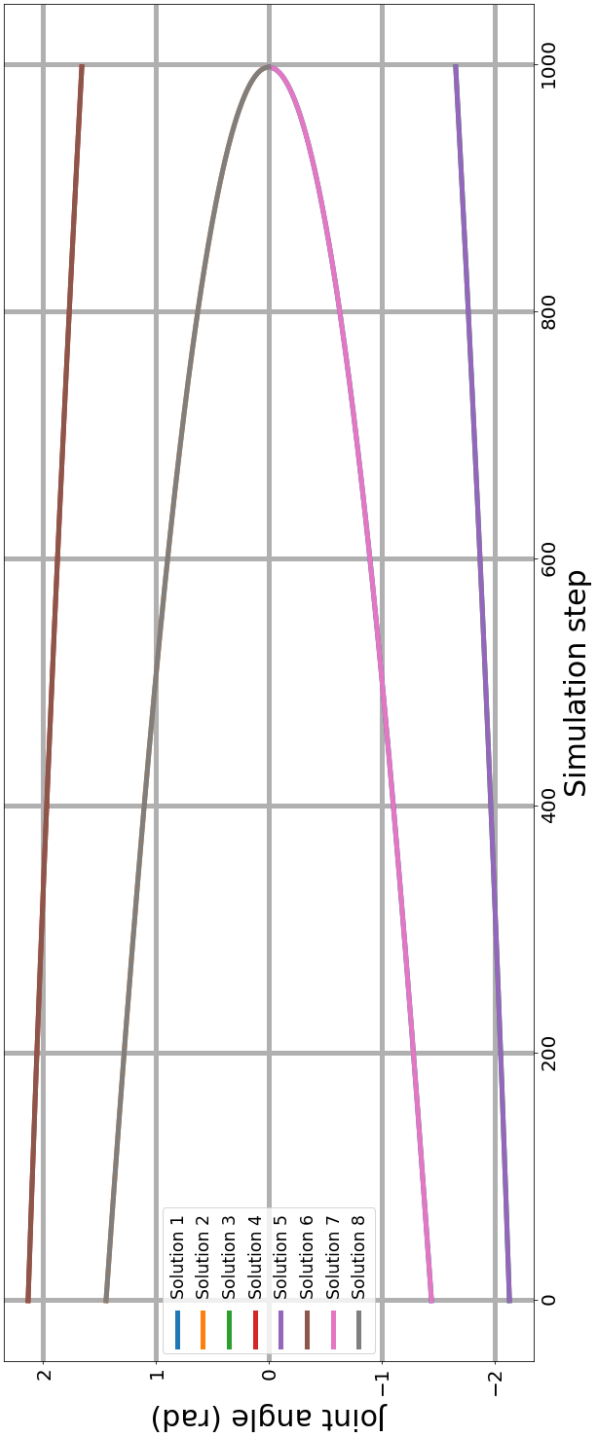

Figure 8.30: Joint 3 angles

From the graph above, we can observe that for some solutions θ_3 reaches to the singularity angle ($\theta_3 = 0$) in the last simulation step. Therefore, we state there is a singularity point. Further analysis reveals that these solutions are the 1, 2, 7 and 8 one, then just like the wrist case, we compute the remaining joint angles.

In Figure 8.31 we show them. In agreement with [72], we can observe that when θ_3 reaches to the singularity angle, the joints 2 and 4 angles drop instantly to this angle in such a way that we can interpret that the robot has gotten stuck.

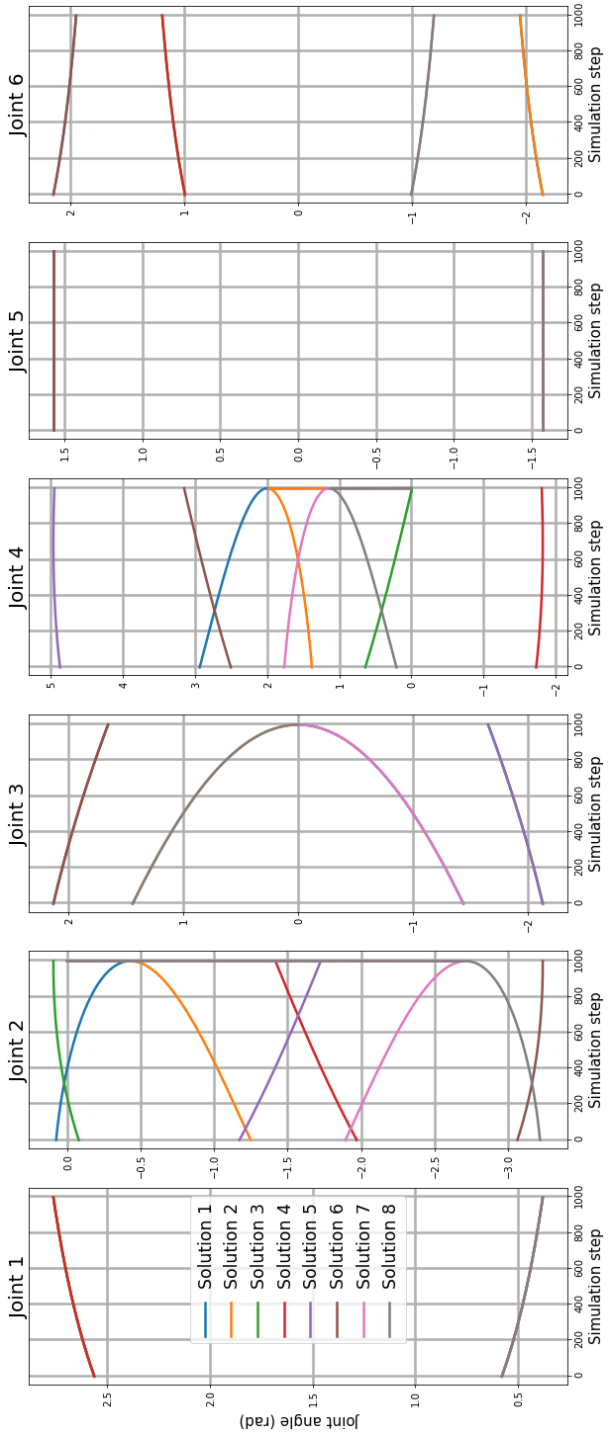

Figure 8.31: Joints angles - Elbow singularity case

Now, in order to observe how this instantaneous drop affects the joints velocities, we will compute them with same considerations of the wrist case. In Figure 8.32, we show the joints velocities.

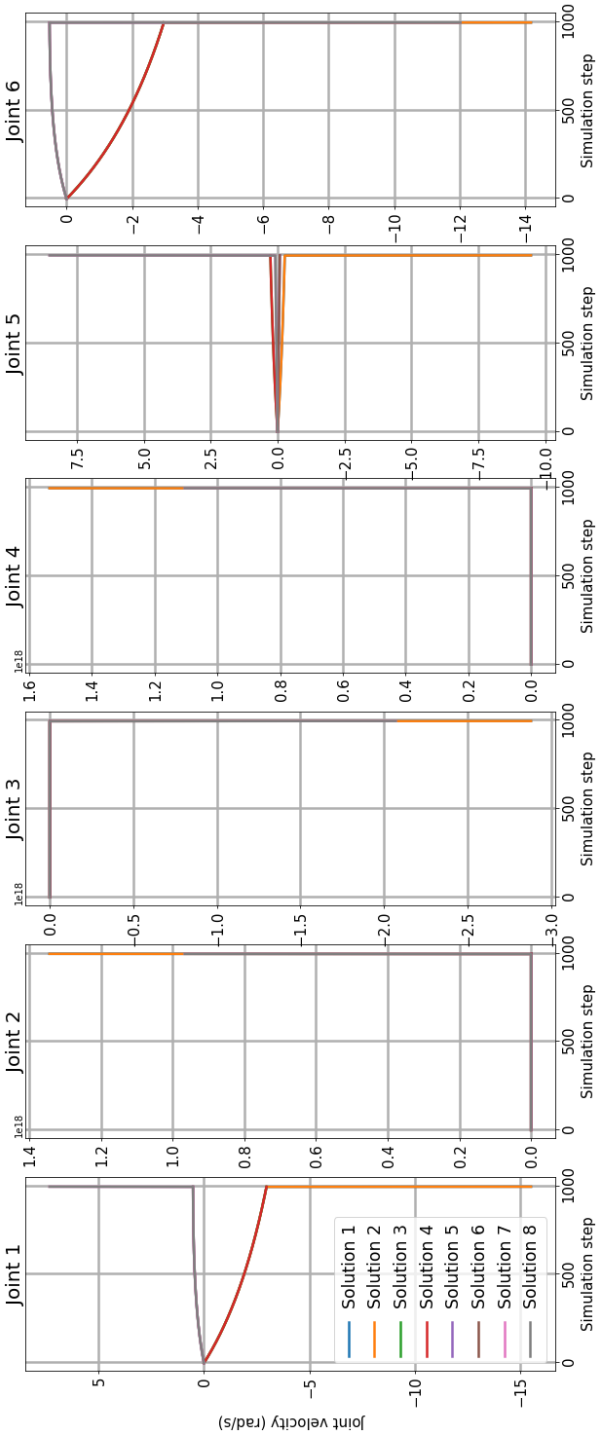

Figure 8.32: Joints velocities - Elbow singularity case

8.7 Jacobian Inverse Kinematic

In Section 8.3, we analyzed closed-form solutions to solve the IK problem for the UR3e robot. While these methods are faster, the equations got are very complex, leading us to have up to 8 different solutions, as well as depending on the structure of the robot. However, the major disadvantage is that they are not robust to situations where the robot arm is close to or inside a singularity scenario. Although that it is difficult to handle these situations, there are some iterative methods such as the Jacobian transpose method, Pseudoinverse method and Damped least-squares method which offer a good performance. In this section, we will describe these methods.

8.7.1 Preliminaries

From the FK, we know that the mapping function from the joint space to the operational space is given by:

$$\vec{x} = f(\vec{\theta}) \tag{8.70}$$

Where f is the mapping function, $\vec{\theta}$ is a current joint angles configuration and $\vec{x}$ is a current generalized end-effector location. Based on the equation (8.70), we can state that it is possible to express $\vec{x}$ in function of the joint angles: $\vec{x}(\vec{\theta})$. In such a way that for a desired end-effector location $\vec{x}_d$, what the IK algorithm seeks to find is a configuration $\vec{\theta}_d$ such that [73][74]:

$$\vec{x_d} = \vec{x}(\vec{\theta_d}) \tag{8.71}$$

Unfortunately, we know that the equation (8.71) not always will have a solution, and if there is, it is not will be unique (For the UR robot family there will be 8 solutions). And, even in a well-behaved situation, there may be no a closed form equation for the solution. However, we can use iterative methods in order to get a suitable solution. For that, we approximate linearly the mapping function using the Jacobian matrix (J) as follows [73][74]:

$$\mathrm{J}(\vec{\theta}) = \frac{\partial f(\vec{\theta})}{\partial \vec{\theta}} \tag{8.72}$$

Differentiating the equation (8.70) and using the definition (8.72), we get:

$$\dot{\vec{X}} = \frac{\partial f(\vec{\theta})}{\partial \vec{\theta}} \dot{\vec{\theta}} = \mathrm{J}(\vec{\theta}) \dot{\vec{\theta}} \tag{8.73}$$

From the equation (8.73), we can estimate the change in the end-effector location caused by the change in the joint angles as [73]:

$$\Delta \vec{X} \approx \mathrm{J} \Delta \vec{\theta} \tag{8.74}$$

Therefore, we can state that the Jacobian leads us to an iterative method for solving the equation (8.70) using instead the equation (8.74) in such a way that we can find a solution from the equation below:

$$\Delta\vec{\theta} = \mathrm{J}^{-1}\Delta\vec{X} \tag{8.75}$$

From equation (8.75) we compute the update value $\Delta\vec{\theta}$ to increase the joint angles $\vec{\theta}$ by $\Delta\vec{\theta}$ in such a way that [73]:

$$\vec{\theta} := \vec{\theta} + \Delta\vec{\theta} \tag{8.76}$$

Therefore, what will differentiate the methods described below is the strategy that they use to calculate $\Delta\vec{\theta}$. However, in most cases, we cannot solve the equation (8.75) uniquely. Furthermore, the Jacobian may not be square or invertible, and even if it is invertible, just using this equation we will not have a suitable performance if J is nearly singular [73].

8.7.2 The Jacobian transpose method

Starting from the equation (8.73) we can approximate it infinitesimally using the finite equivalents of $\dot{\vec{\theta}}$ and $\dot{\vec{x}}$ as follows [74]:

$$\dot{\vec{\theta}} = \frac{(\vec{\theta}_d - \vec{\theta})}{\Delta t} \tag{8.77}$$

$$\dot{\vec{X}} = \alpha\frac{(\vec{x}_d - \vec{x})}{\Delta t} \tag{8.78}$$

Where α is a given positive coefficient [74]. Replacing (8.77) and (8.78) in the equation (8.73), we get:

$$\alpha\frac{(\vec{x}_d - \vec{x})}{\Delta t} = \mathrm{J}(\vec{\theta})\frac{(\vec{\theta}_d - \vec{\theta})}{\Delta t}$$

$$\alpha(\vec{x}_d - \vec{x}) = \mathrm{J}(\vec{\theta})(\vec{\theta}_d - \vec{\theta})$$

$$(\vec{\theta}_d - \vec{\theta}) = \alpha\mathrm{J}(\vec{\theta})^{-1}(\vec{x}_d - \vec{x})$$

$$\vec{\theta}_d = \vec{\theta} + \alpha\mathrm{J}(\vec{\theta})^{-1}(\vec{x}_d - \vec{x}) \tag{8.79}$$

If we compare the equations (8.79) and (8.76), we conclude that:

$$\Delta\vec{\theta} = \alpha\mathrm{J}(\vec{\theta})^{-1}(\vec{x}_d - \vec{x}) \tag{8.80}$$

As mentioned above, the possibility that the Jacobian is non-invertible or that it is nearly singular will make that the algorithm work poorly. Therefore, we need to find some kind of Jacobian inverse matrix, and the most used in practice is the Moore–Penrose inverse matrix [75]. According to the definition of this pseudoinverse matrix, if the Jacobian has linearly independent rows (matrix JJ^{T} is invertible), we can express the Moore-Penrose inverse of J as:

$$\mathrm{J}^{+} = \mathrm{J}^{\mathrm{T}}(\mathrm{JJ}^{\mathrm{T}})^{-1} \tag{8.81}$$

Using the definition (8.81) in the equation (8.80), we get:

$$\Delta\vec{\theta} = \alpha \mathrm{J}(\vec{\theta})^{\mathrm{T}} (\mathrm{M}(\vec{\theta}))^{-1} (\vec{x}_d - \vec{x}) \tag{8.82}$$

Where $\mathrm{M}(\vec{\theta})$ denotes an non-negative definite, symmetric manipulable matrix:

$$\mathrm{M} = \mathrm{J}(\vec{\theta})\mathrm{J}(\vec{\theta})^{\mathrm{T}}$$

Finally, if we omit the manipulable matrix $\mathrm{M}(\vec{\theta})$ from the equation (8.82), we got the well-known Jacobian transpose inverse method as follows:

$$\Delta\vec{\theta} = \alpha \mathrm{J}(\vec{\theta})^{\mathrm{T}} (\vec{x}_d - \vec{x}) \tag{8.83}$$

For a sufficiently small value of α, the algorithm always will move the end-effector closer to the target location. The Jacobian transpose method is stable and has a suitable performance for multi-bodies with a single end-effector. However, it converges slowly with multiple end-effectors [76].

8.7.3 The pseudoinverse method

From the equation (8.74), multiplying both sides by J^{T}, we can derive the normal equation below [73]:

$$\mathrm{J}^{\mathrm{T}}\mathrm{J}\Delta\vec{\theta} = \mathrm{J}^{\mathrm{T}}\Delta\vec{X} \tag{8.84}$$

Then, we let $\mathrm{J}^{\mathrm{T}}\Delta\vec{X} = \vec{Z}$ and solve the equation below:

$$\mathrm{J}^{\mathrm{T}}\mathrm{J}\Delta\vec{\theta} = \vec{Z} \tag{8.85}$$

According to [73], $\vec{Z}$ is always in the range of the matrix $\mathrm{J}^{\mathrm{T}}\mathrm{J}$, thus exists a solution for the equation (8.85). In this way, we can find a solution with row operations from the equation below [77]:

$$\Delta\vec{\theta} = \left(\mathrm{J}^{\mathrm{T}}\mathrm{J}\right)^{-1} \mathrm{J}^{\mathrm{T}}\Delta\vec{X} \tag{8.86}$$

We notice that, if $\mathrm{J}^{\mathrm{T}}\mathrm{J}$ is invertible, the matrix $\left(\mathrm{J}^{\mathrm{T}}\mathrm{J}\right)^{-1}\mathrm{J}^{\mathrm{T}}$ is the Moore-Penrose inverse of J. Therefore, in this way, the pseudoinverse method sets $\Delta\vec{\theta}$ equal to:

$$\Delta\vec{\theta} = \mathrm{J}^{+}\Delta\vec{X}$$

Now, if the Jacobian has linearly independent rows (full row rank), then we can express the Moore-Penrose inverse of J as:

$$\mathrm{J}^{+} = \mathrm{J}^{\mathrm{T}}\left(\mathrm{J}^{\mathrm{T}}\mathrm{J}\right)^{-1}$$

Therefore, in this case, we can rewrite the equation (8.86) as:

$$\Delta\vec{\theta} = \mathrm{J}^{\mathrm{T}} \left(\mathrm{J}^{\mathrm{T}}\mathrm{J}\right)^{-1} \Delta\vec{X} \tag{8.87}$$

The equation (8.87) gives the best possible solution in the sense of least squares. If the joint angle configuration is exactly at a singularity, then the pseudoinverse method will be well-behaved. However, if the configuration is close to a singularity, the method will lead to very large changes in the joint angles, even for slight movements in the target position. Therefore, we can state that the pseudoinverse method has stability problems in the singularity neighbourhood [76][73].

8.7.4 Damped least squares

The damped least squares method, also called the Levenberg-Marquardt method, avoids the problems with singularities described in the pseudoinverse method. It is a numerically stable method of selecting $\Delta\theta$. The aim of the algorithm, instead of to find the minimum vector $\Delta\theta$, is to find the value $\Delta\theta$ that minimizes the quantity below [73]:

$$\|\mathrm{J}\Delta\theta - \Delta\vec{X}\|^2 + \lambda\|\Delta\theta\|^2 \tag{8.88}$$

Where $\lambda \in \mathrm{R}$ is a non-zero damping constant. Minimizing the quantity (8.88) is equivalent to minimizes the quantity:

$$\left\| \begin{pmatrix} \mathrm{J} \\ \lambda I \end{pmatrix} \Delta\theta - \begin{pmatrix} \Delta\vec{X} \\ \vec{0} \end{pmatrix} \right\| \tag{8.89}$$

From (8.89), we can define the corresponding normal equation as:

$$\begin{pmatrix} \mathrm{J} \\ \lambda I \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \mathrm{J} \\ \lambda I \end{pmatrix} \Delta\theta = \begin{pmatrix} \mathrm{J} \\ \lambda I \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \Delta\vec{X} \\ \vec{0} \end{pmatrix}$$

Which we can rewrite as:

$$\left(\mathrm{J}^{\mathrm{T}}\mathrm{J} + \lambda^2 I\right) \Delta\vec{\theta} = \mathrm{J}^{\mathrm{T}}\Delta\vec{X}$$

Therefore, the damped least squares solution is:

$$\Delta\vec{\theta} = \left(\mathrm{J}^{\mathrm{T}}\mathrm{J} + \lambda^2 I\right)^{-1} \mathrm{J}^{\mathrm{T}}\Delta\vec{X}$$

Finally, since $\left(\mathrm{J}^{\mathrm{T}}\mathrm{J} + \lambda^2 I\right)^{-1} \mathrm{J}^{\mathrm{T}} = \mathrm{J}^{\mathrm{T}} \left(\mathrm{J}\mathrm{J}^{\mathrm{T}} + \lambda^2 I\right)^{-1}$, we can compute the update value $\Delta\vec{\theta}$ as:

$$\Delta\vec{\theta} = \mathrm{J}^{\mathrm{T}} \left(\mathrm{J}\mathrm{J}^{\mathrm{T}} + \lambda^2 I\right)^{-1} \Delta\vec{X}$$

To illustrate how the damped least-squares method deals with a singularity situation (wrist singularity), in Figure 8.33, we show two trajectories calculated using the analytical IK and the damped method.

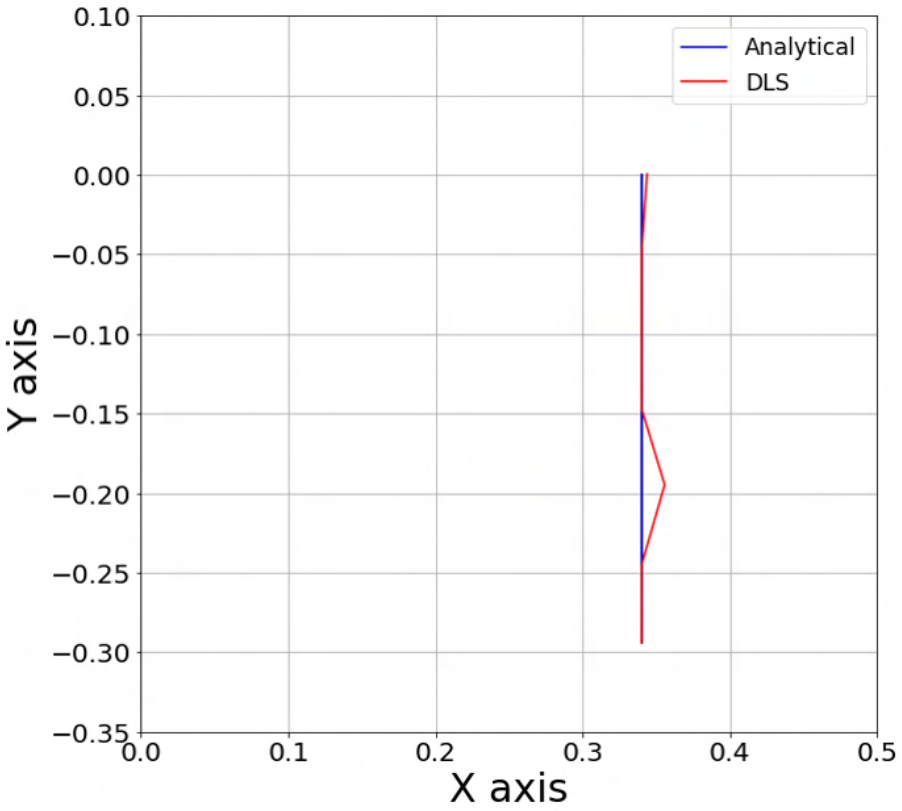

Figure 8.33: Singularity avoiding in wrist singularity situation

As we can see, at the singularity point, the trajectory calculated using the damped method deviates in such a way that it avoids reaching the singularity.

Chapter 9

Conclusions

- In the development of the systems, we have made use of both strategies based on image processing techniques and strategies based on deep learning models. From both strategies, we can highlight the speed of the image processing techniques, but we can also highlight their problem in adapting to a variable environment, such as variable lighting conditions. These drawbacks are solved by deep learning model-based strategies, which can adapt to variable environments by learning patterns from a dataset. However, it is this last detail, the bottleneck of this type of approach, the dataset. Without a good representation of the data, we cannot expect the model to meet our objectives.

Bibliography

[1] Ronja Möller, Antonino Furnari, Sebastiano Battiato, Aki Härmä, and Giovanni Maria Farinella. A survey on human-aware robot navigation. *Robotics and Autonomous Systems*, 145:103837, 2021.

[2] Zhihao Liu, Quan Liu, Wenjun Xu, Lihui Wang, and Zude Zhou. Robot learning towards smart robotic manufacturing: A review. *Robotics and Computer-Integrated Manufacturing*, 77:102360, 2022.

[3] Yueyue Liu, Zhijun Li, Huaping Liu, and Zhen Kan. Skill transfer learning for autonomous robots and human–robot cooperation: A survey. *Robotics and Autonomous Systems*, 128:103515, 2020.

[4] Tshilidzi Marwala. *Rational Machines and Artificial Intelligence*. Academic Press, 2021.

[5] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[6] D. Partridge. Artificial intelligence. In *Reference Module in Neuroscience and Biobehavioral Psychology*. Elsevier, 2017.

[7] Neeraj Karnani, PM Tiwari, and JK Rai. Design and implementation of sudoku solving robot using microcontroller. In *2013 Students Conference on Engineering and Systems (SCES)*, pages 1–5. IEEE, 2013.

[8] Sonal Jain and Neeraj Khera. An intelligent method forsolving tic-tac-toe problem. In *International Conference on Computing, Communication & Automation*, pages 181–184. IEEE, 2015.

[9] Sai Cheong Fok and EK Ong. A high school project on artificial intelligence in robotics. *Artificial intelligence in engineering*, 10(1):61–70, 1996.

[10] Aviva Rutkin. Tic-tac-toebot. *New Scientist*, 232(3103):22, 2016.

[11] Ekaterina Karmanova, Valerii Serpiva, Stepan Perminov, Aleksey Fedoseev, and Dzmitry Tsetserukou. Swarmplay: Interactive tic-tac-toe board game with swarm of

nano-uavs driven by reinforcement learning. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, pages 1269–1274. IEEE, 2021.

[12] Sebastian Adi Nugroho, Ary Setijadi Prihatmanto, and Arief Syaichu Rohman. Design and implementation of kinematics model and trajectory planning for nao humanoid robot in a tic-tac-toe board game. In *2014 IEEE 4th International Conference on System Engineering and Technology (ICSET)*, volume 4, pages 1–7. IEEE, 2014.

[13] Canny edge detector. `https://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html#canny-edge-detector`. Accessed: 2022-07-26.

[14] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[15] Canny edge detection. `https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html`. Accessed: 2022-07-26.

[16] Canny edge detector. `https://www.digimizer.com/manual/m-image-filtercanny.php#:~:text=Sigma%3A%20Standard%20deviation%20of%20the,time%20and%20causes%20more%20blurring`. Accessed: 2022-07-26.

[17] Smoothing images. `https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html#:~:text=This%20Gaussian%20filter%20is%20a,don't%20want%20to%20do`. Accessed: 2022-07-26.

[18] Label image regions. `https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_label.html`. Accessed: 2022-07-27.

[19] Measure region properties. `https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_regionprops.html`. Accessed: 2022-07-27.

[20] Allison Zhang and Don Gourley. *Creating digital collections: A practical guide*. Elsevier, 2014.

[21] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[22] Hough line transform. `https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html`. Accessed: 2022-07-27.

[23] Straight line hough transform. https://scikit-image.org/docs/stable/auto_examples/edges/plot_line_hough_transform.html. Accessed: 2022-07-27.

[24] Thresholding. https://scikit-image.org/docs/stable/auto_examples/applications/plot_thresholding.html. Accessed: 2022-07-27.

[25] Skeletonize. https://scikit-image.org/docs/stable/auto_examples/edges/plot_skeleton.html. Accessed: 2022-07-27.

[26] Image thresholding. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html. Accessed: 2022-07-28.

[27] Convex hull. https://scikit-image.org/docs/stable/auto_examples/edges/plot_convex_hull.html. Accessed: 2022-07-28.

[28] Aiswarya Raj Munappy, Jan Bosch, Helena Holmström Olsson, Anders Arpteg, and Björn Brinne. Data management for production quality deep learning models: Challenges and solutions. *Journal of Systems and Software*, page 111359, 2022.

[29] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[30] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[32] A comprehensive guide to convolutional neural networks — the eli5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed: 2022-08-08.

[33] Convolutional neural networks (cnns / convnets). https://cs231n.github.io/convolutional-networks/. Accessed: 2022-08-08.

[34] Cnn convolution on rgb images. https://datahacker.rs/convolution-rgb-image/. Accessed: 2022-08-14.

[35] Maxpool2d. https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d. Accessed: 2022-08-14.

[36] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74, 2021.

[37] Dropout layer. https://keras.io/api/layers/regularization_layers/dropout/#:~:text=The%20Dropout%20layer%20randomly%20sets,time%2C%20which%20helps%20prevent%20overfitting. Accessed: 2022-08-14.

[38] How the backpropagation algorithm works. http://neuralnetworksanddeeplearning.com/chap2.html. Accessed: 2022-08-03.

[39] Gradient descent. https://www.ibm.com/cloud/learn/gradient-descent#:~:text=Gradient%20descent%20is%20an%20optimization,each%20iteration%20of%20parameter%20updates. Accessed: 2022-08-04.

[40] Adam. https://keras.io/api/optimizers/adam/. Accessed: 2022-08-04.

[41] F Gaillard and DJ Bell. Epoch (machine learning): Radiology reference article. *Radiopaedia Blog RSS*, 2020.

[42] A Murphy and F Gaillard. Batch size (machine learning). *Radiopaedia Blog RSS*, 2019.

[43] A Murphy and F Gaillard. Iteration (machine learning). *Radiopaedia Blog RSS*, 2019.

[44] Robert L. Kissell. Chapter 19 - machine learning and trade schedule optimization. In Robert L. Kissell, editor, *Algorithmic Trading Methods (Second Edition)*, pages 519–542. Academic Press, 2021.

[45] Fatih Demir. Deep autoencoder-based automated brain tumor detection from mri data. In *Artificial Intelligence-Based Brain-Computer Interface*, pages 317–351. Elsevier, 2022.

[46] Deepak Kumar Sharma, Mayukh Chatterjee, Gurmehak Kaur, and Suchitra Vavilala. Deep learning applications for disease diagnosis. In *Deep Learning for Medical Applications with Unique Data*, pages 31–51. Elsevier, 2022.

[47] Ajay Kulkarni, Deri Chong, and Feras A Batarseh. Foundations of data imbalance and solutions for a data democracy. In *data democracy*, pages 83–106. Elsevier, 2020.

[48] Coordinates of a flat scene. https://towardsdatascience.com/coordinates-of-a-flat-scene-b37487df63ca. Accessed: 2022-08-01.

[49] Burak Benligiray and Cihan Topal. Lens distortion rectification using triangulation based interpolation. In *International Symposium on Visual Computing*, pages 35–44. Springer, 2015.

[50] Olgierd Stankiewicz, Gauthier Lafruit, and Marek Domański. Multiview video: Acquisition, processing, compression, and virtual view rendering. In *Academic Press Library in Signal Processing, Volume 6*, pages 3–74. Elsevier, 2018.

[51] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[52] I Kurniastuti, ENI Yuliati, F Yudianto, and TD Wulan. Determination of hue saturation value (hsv) color feature in kidney histology image. In *Journal of Physics: Conference Series*, page 012020. IOP Publishing, 2022.

[53] Hui Liu. *Robot systems for rail transit applications*. Elsevier, 2020.

[54] Mariusz Specht. Statistical distribution analysis of navigation positioning system errors—issue of the empirical sample size. *Sensors*, 20(24):7144, 2020.

[55] Yi Nam Suen and Ester Cerin. *Measurement Error*, pages 3907–3909. Springer Netherlands, Dordrecht, 2014.

[56] Brent J. Lewis, E. Nihan Onder, and Andrew A. Prudil. Chapter 8 - treatment of experimental results. In Brent J. Lewis, E. Nihan Onder, and Andrew A. Prudil, editors, *Advanced Mathematics for Engineering Students*, pages 233–263. Butterworth-Heinemann, 2022.

[57] Francisco Dallmeier, Robert C. Szaro, Alfonso Alonso, James Comiskey, and Ann Henderson. Framework for assessment and monitoring of biodiversity. In Simon A Levin, editor, *Encyclopedia of Biodiversity (Second Edition)*, pages 545–559. Academic Press, Waltham, 2013.

[58] J. Ferré. 3.02 - regression diagnostics. In Steven D. Brown, Romá Tauler, and Beata Walczak, editors, *Comprehensive Chemometrics*, pages 33–89. Elsevier, Oxford, 2009.

[59] Matthew Hartley and Tjelvar SG Olsson. dtoolai: Reproducibility for deep learning. *Patterns*, 1(5):100073, 2020.

[60] Saeed S Alahmari, Dmitry B Goldgof, Peter R Mouton, and Lawrence O Hall. Challenges for the repeatability of deep learning models. *IEEE Access*, 8:211860–211868, 2020.

[61] Multi-otsu thresholding. `https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_multiotsu.html#:~:text=The%20multi%2DOtsu%20threshold%201,the%20number%20of%20desired%20classes.` Accessed: 2022-08-07.

[62] Bruno Siciliano, Oussama Khatib, and Torsten Kröger. *Springer handbook of robotics*, volume 200. Springer, 2008.

[63] Wisama Khalil and Etienne Dombre. *Modeling identification and control of robots*. CRC Press, 2002.

[64] Mark W Spong and Mathukumalli Vidyasagar. *Robot dynamics and control*. John Wiley & Sons, 2008.

[65] Dh parameters for calculations of kinematics and dynamics. `https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/`. Accessed: 2022-08-08.

[66] A geometric inverse kinematics solution for the universal robot. `https://asd.sutd.edu.sg/dfab/a-geometric-inverse-kinematics-solution-for-the-universal-robot/`. Accessed: 2022-08-08.

[67] Kelsey P Hawkins. Analytic inverse kinematics for the universal robots ur-5/ur-10 arms. Technical report, Georgia Institute of Technology, 2013.

[68] Rasmus Skovgaard Andersen. Kinematics of a ur5. *Aalborg University*, 2018.

[69] Qiang Liu, Daoguo Yang, Weidong Hao, and Yao Wei. Research on kinematic modeling and analysis methods of ur robot. In *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pages 159–164. IEEE, 2018.

[70] EN ISO. 10218: Robots and robotic devices—safety requirements for industrial robots—part 1: Robots. *ISO: Geneve, Switzerland*, 2011.

[71] What are singularities in a six-axis robot arm? `https://www.mecademic.com/en/what-are-singularities-in-a-six-axis-robot-arm`. Accessed: 2022-08-08.

[72] Why singularities can ruin your day. `https://blog.robotiq.com/why-singularities-can-ruin-your-day?hsLang=en-us`. Accessed: 2022-08-08.

[73] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1-19):16, 2004.

[74] Ignacy Dulkeba and Michal Opalka. A comparison of jacobian-based methods of inverse kinematics for serial robot manipulators. *International Journal of Applied Mathematics and Computer Science*, 23(2):373–382, 2013.

[75] Yoshihiko Nakamura. *Advanced robotics: redundancy and optimization*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[76] Samuel R Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. *Journal of Graphics tools*, 10(3):37–49, 2005.

[77] Rickard Nilsson. Inverse kinematics, 2009.