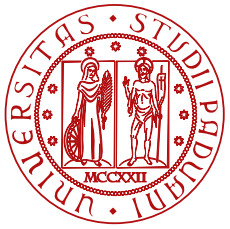

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering

Vision based robot manipulator error compensation

Master Candidate

Angelo Zaccarin

Student ID 2052414

Supervisor

Prof. Emanuele Menegatti

University of Padova

Company tutor

Dott. Luca Bizzotto

Robo Ware

Academic Year
2022/2023

Date
23/10/2023

Abstract

Currently one of the most important obstacles to a wider utilization of robots in industry is their accuracy. This is a critical issue for applications that allow no margin of error such as medical applications or precision machining. A possibility is to solve the problem at its root by manufacturing better robot, but that would require substantial economical and time investments. A viable alternative is to improve the robot accuracy through a calibration process, either by creating a model of the robot that closely represents the real robot or by studying the error model of the robot to compensate it directly. The aim of this study is to develop a correction algorithm to improve the robot accuracy by remapping the robot workspace through the use of computer vision and machine leaning techniques. Different correction techniques were tested against each other to find the one that would bring the robot accuracy closer to its repeatability, together with some important additions to seamlessly include the correction algorithm into an existing pipeline.

Contents

1 Introduction

The purpose of this thesis is to develop an algorithm to improve the accuracy of a robot manipulator. In order to do so the robot workspace will be remapped through the use of computer vision and machine leaning techniques.

1.1 Robot calibration

Currently one of the most important obstacles to a wider utilization of robots in industry is their accuracy [1]. This is a critical issue for industrial applications that allow no margin of error and considerable impediment to a more extensive use of advanced robot programming techniques. The identified parameters relating to robot positional performance are accuracy, repeatability, and resolution. Each of these depends on the various components used in constructing the robot (links, motors, encoders, etc.), the construction procedure, and the capabilities of the driving actuators and the controller. Resolution is defined as the smallest incremental move that the robot can physically produce. It is defined through the control system used to power the manipulator, but is also affected by the construction procedure, manipulator stiffness, (structural flexibility) and encoders. Repeatability is a measure of the ability of the robot to move back to the same position and orientation over and over again [2]. Accuracy is defined

as the ability of the robot to precisely move to a desired position in 3D space [3].

For majority of today industrial robots manipulators the repeatability is on the order of 0.1 mm but the absolute positioning accuracy is on the order of 1 mm [4]. There is no need to explain that it is desirable that the robot reaches both high accuracy and repeatability. However, due to the fact that the absolute positioning accuracy error is frequently greater than the repeatability by an order of magnitude, low accuracy of a robot is often regarded as a more pressing problem, because it practically restricts the applications to the ones that can be satisfactorily programmed by *teaching by showing* methods where a human operator manually guides the robot through a series of movements to teach it a task.

This is a valuable tool in industrial robotics but it has its limitations: the accuracy of the taught path is limited by the operator's ability to guide the robot, can be time consuming as the operator must physically guide the robot through the entire task and the final path will probably not be optimized [5].

Robot accuracy is influenced by a number of factors, which can be classified into five categories [6]:

1. environmental: temperature or humidity changes;
2. parametric: variation of kinematic parameters, joint zero-reference; offsets, influence of dynamic parameters, drive-train compliance, friction and other nonlinearities, including hysteresis and backlash;
3. measurement: resolution and nonlinearity of joint position sensors;
4. computational: computer round-off and steady-state servo errors;
5. application: installation errors and workpiece position and geometry errors.

Analysis of their influence and its elimination is a subject of intensive research aimed at the improvement of both accuracy and repeatability through robot calibration processes. Over the course of time multiple research papers or robot calibration have been published but one of particular interest is the one written by Roth, Mooring and Ravani where they proposed a hierarchical classification in which the three calibration levels are separated [7]:

1. joint level calibration: calibrating the robot by establishing the correct relationship between the joint sensor signal and the actual joint displacement;

2. kinematic model calibration: calibrating the robot by improving the kinematic model, with the assumption that the robot is composed of ideally rigid links and nonelastic joints [8];

3. nongeometric (nonkinematic) calibration: at this level, deviations from the ideal kinematic model due to effects, such as joint compliance, backlash and link compliance are considered.

The aim of this thesis is to develop an algorithm to improve the robot accuracy through the use of computer vision and machine leaning techniques to be able to remap the robot workspace. In doing this it won't be necessary to create an accurate model of the robot. The position error distributed in its workspace will instead be studied and compensated directly.

1.2 Sources of error

The major error contributions can be subdivided into structural, kinematic, and dynamic. An illustration of the hierarchy of the factors that contribute to the positional accuracy and repeatability of a robot is presented in Figure 1.1 [9]. Solutions such as building a better robot, building more rigid fixtures, and improving manufacturing processes that could help in improving this problem are often not feasible due to the required or unavailable resources. Compensation for this error through an in-process feedback mechanism is a much more attractive alternative. As in any control system, the process parameters will define the level of required sophistication. The joint zero position error is often responsible for 90% [9] of the robot positional error.

A mathematical model within each robot controller assumes that the links on one robot are the same length as the links on another robot of the same model and type. Additionally, the same model also assumes that the relative orientations of the joints on one robot are the same as on another robot of the same type. Unfortunately, this assumption is not true due to manufacturing imperfections and assembly variations [10]. Therefore, the controller will incorrectly estimate the robot pose given a set of joint angles. The next most significant factor in

the robot positional error is joint compliance. This may be thought of as a factor representing the elasticity of each joint caused by the effects of gravity, payload, and inertia. Each of the following robot characteristics: accuracy, repeatability, and resolution depends upon many factors that include, but are not limited to, friction, temperature, loading, and manufacturing tolerances. Of the three robot characteristics, high accuracy is the one that this thesis aims to improve. Differences between the modelled, designed components and the actual, built components will affect the accuracy. The controller software can be designed to account for discrepancies between the mathematical controller model and the actual built part. However, this approach is not cost-effective for mass-produced robots because considerable effort must be expended to identify the characteristics of the various robot components within the desired accuracy using complex, expensive, and time-consuming techniques [11].

1.3 State of the art

The main objective of robot calibration is to establish an accurate relationship between the nominal and actual end-effector pose. To achieve this result two main approaches were developed: model-based or parametric methods [12] and modeless or non-parametric methods [13]. The model-based methods involve defining a kinematic model for the robot, measuring positions and orientations of the robot end-effector, identify the actual kinematic parameters to minimise the errors between the locations predicted by the model and the actual measured ones and compensating for its pose errors by modifying its joint angles [14].

The advantage of a model based calibration method is that a large workspace can be calibrated accurately and all pose errors within the calibrated workspace can be compensated for by joint angles [15]. But model based calibration has also some very important limitations: the kinematic model takes only geometric factors into account, but non-geometric factors such as backlash, joint compliance, etc., may be significant in affecting robot accuracy, depending on

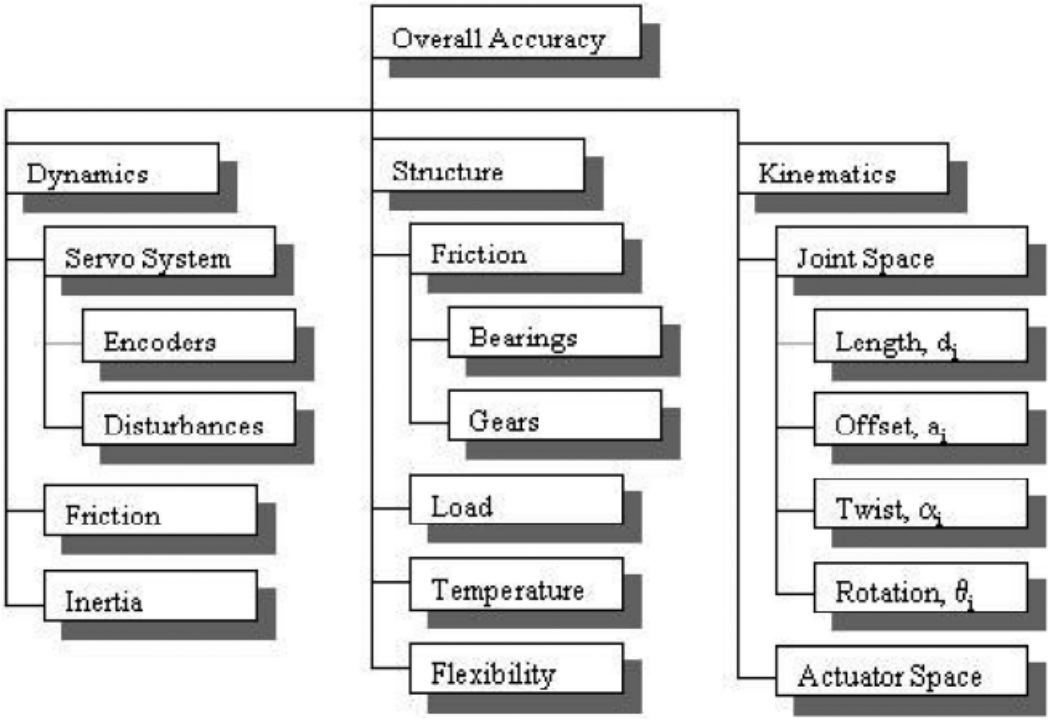

Figure 1.1: Factors contributing in the robot accuracy and repeatability. Sourced from "On the Accuracy, Repeatability, and Degree of Influence of Kinematics Parameters for Industrial Robots" [9].

the application. Including those factors into the kinematic model will increase its complexity exponentially. Furthermore the identification of the kinematic parameters is a complex procedure [16] which can suffer from the numerical problem of ill-conditioning resulting in inaccuracy and inefficiency. The results of identification can hardly be near-perfect due to the numerical inaccuracy and limitations of the model used.

Finally the implementation of the identified model can be problematic due to the difficulty of modifying the controller parameters which are not always accessible, depending on the manufacturer, and the additional difficulty in solving the inversion problem of the modified kinematic model. The alternative proposed in this thesis belongs to the modeless category, in the following paragraphs the motivation of this choice will be explained in detail and the advantages of modeless calibration methods will be highlighted. A modeless method does not go through any kinematic modelling and identification steps instead it is divided into two stages. The first stage comprehend the data gathering process

during which the robot workspace is divided into a sequence of small squares in a two-dimensional (2D) case, or cubes in a three-dimensional (3D) case with the relative distance between the cells assumed known. A camera or other measurement device is attached on the robots end-effector to find the position errors of the end-effector and a calibration board is installed on the robots workspace such that the pattern of the board matches with the grid division of the workspace. During this process, all position errors on the grid points are measured and recorded by moving the robot through all the grid points. These position errors are stored in memory to be used later in the second phase where the information gathered about the robot position errors are used to actually correct the robot position in an application. If the manipulator only has to move in a limited number of points during a specific application then having the grid overlap with the needed points and using a simple lookup function will be enough to correct the errors. But often more flexibility is needed. For common tasks such as picking or placing objects in pallets the robot manipulator only needs high accuracy on a small portion of the workspace. But we can't expect that the objects to be picked or placed will always be in the same exact positions, a small misalignment of the pallets and the lookup function will be unable to locate the correct points. So a way to extract additional information from the saved points is needed. This is done by interpolating errors from its neighbouring grid points.

Different techniques to generalize the information acquired in the data gathering step have been explored, such as: bilinear interpolation and neural networks since they can easily work either in Cartesian space [17] or joint space [18]. this thesis proposes a direct comparison between such techniques, together with some important additions to help transition from the controlled environment of the data gathering step to performing tests in a real application.

1.4 Thesis outline

This thesis is divided into six main chapters. This chapter provided the motivation that lead to the development of this thesis and explain the problem of robot calibration more in depth. The following chapter "Framework" will briefly

illustrate all the components used in this thesis, both hardware and software. The third chapter "Proposed approach" will introduce the reader to the mathematical tools used to correct the robot position such as bilinear interpolation and neural networks. Chapter four "Implementation" will dive deeper on the techniques used to actually implement the correction algorithms, starting with the physical setup with a camera mounted on the end effector of the robot and a calibration board and following with the pipeline to gather information about the robot current repeatability and accuracy. The last sections of the chapter illustrate how to integrate the theoretical correction models already introduced in chapter three into a real pipeline that allows us to objectively quantify the improvements on the robot accuracy provided by the correction algorithms and will also expose some potential problems and solutions that may emerge when moving from the controlled environment of the data gathering step to performing tests in a real application. Chapter five "Results and discussion" will illustrate the results achieved by this thesis and the different approaches presented to improve the robot accuracy will have their performance compared against one another. In the final chapter some possibilities to continue the work begun in this thesis will be considered.

2 Framework

This chapter illustrates all the components used in this thesis, both hardware and software. All the hardware components and the software licenses used for the experiments have been provided by *Robo Ware* [19].

2.1 Dobot Mg400

The robot used to perform the experiments is the Mg400 provided by Dobot, it can be seen in Figure 2.1 [20]. It is a small collaborative desktop robotic arm with four actuated joints and a closed kinematic chain to keep the flange always orthogonal to the ground plane.

Since it supports drag-to-teach and collision detection features the Mg400 is usually applied to automate small batch flexible productions, it can carry a payload of up to 750 g and can extend up to 440 mm.

The Mg400 has a very high repeatability, so it can return to a saved point with a precision of up to +- 0,05 mm but as it is common for industrial robots its accuracy is much worse [21], so when asked to reach a new point the error will be not negligible. This creates complications when it is used for applications that require high precision. An example of such applications where this problem emerges are palletizing operations where the robot has to pick or place objects

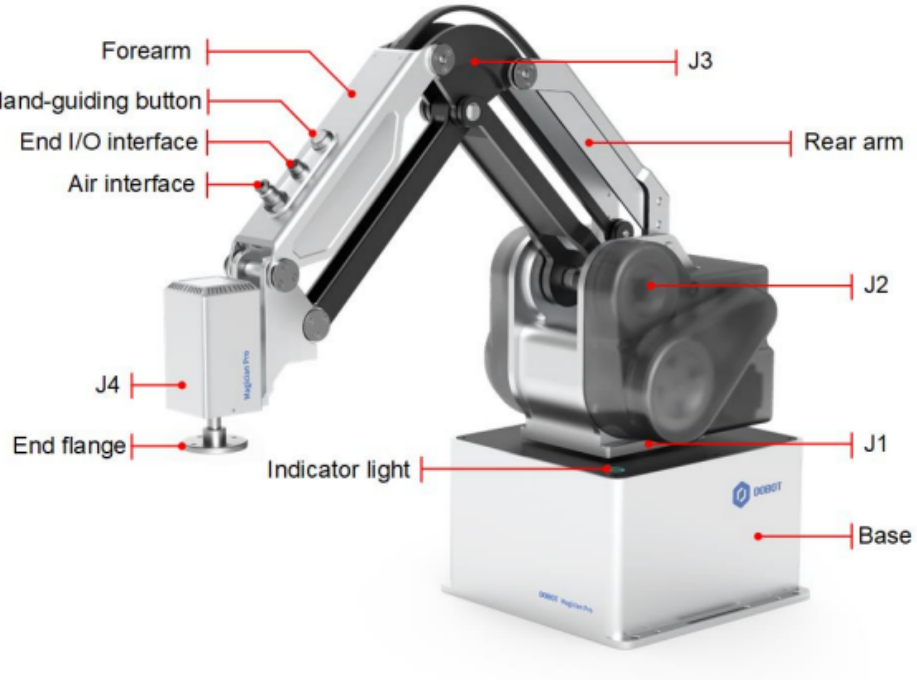

Figure 2.1: Dobot Mg400. Sourced from *Dobot Robotics* [20].

that are stacked in a pallet. In these applications the human operator teaches the robot only the positions of the extremes of the pallet and the robot has to compute all the intermediate positions. In this instance instead of moving in a straight line the robot motion presents a slight drift and the end effector moves on a curved line. Teaching by showing the robot new points is a time intensive process for the human operators so registering all the positions on the pallet is not a viable option.

The aim of this thesis is to improve the accuracy of the Mg400 through the means of a camera mounted on the end effector and computer vision techniques.

2.2 Camera

The camera used to perform the experiments was a Hikrobot MV-CE050-30UC which was included in the vision kit provided by Dobot. The camera can

be seen in Figure 2.2 [22], has a resolution of 5 Mp and can be connected to a computer by means of an USB 3.0 cable. Some other useful specifications are the pixel size 2.2 µm Œ 2.2 µm, the image resolution 2592 x 1944 and the focal length of the lens of 12 mm.

Figure 2.2: Hikrobot MV-CE050-30UC. Sourced from *Hikrobot* [22].

2.3 Calibration board

The calibration board used in this thesis, seen in Figure 2.3 [23] comes from Calib.io. It is composed by aluminium/LDPE (Low-Density Polyethylene) composite sheets. It measures 400x300 mm and is covered in a circle grid pattern of 26 columns and 19 rows of 6 mm diameter circles equispaced by 15 mm. Although only the first 11 rows were used as the whole board would not fit inside the robot workspace at the height it needed to operate.

At the beginning of the study some cheaper options were considered such as printing the calibration grid on a sheet of paper and fixing it in place inside the robot workspace. After some tests it emerged that the measurement of the vertical position error of the robot is very sensitive to printing defects on the

calibration grid that may alter the dimension of the single dots and to the paper not laying completely flat on a plane as it will be discussed more in the "Results and discussion" chapter.

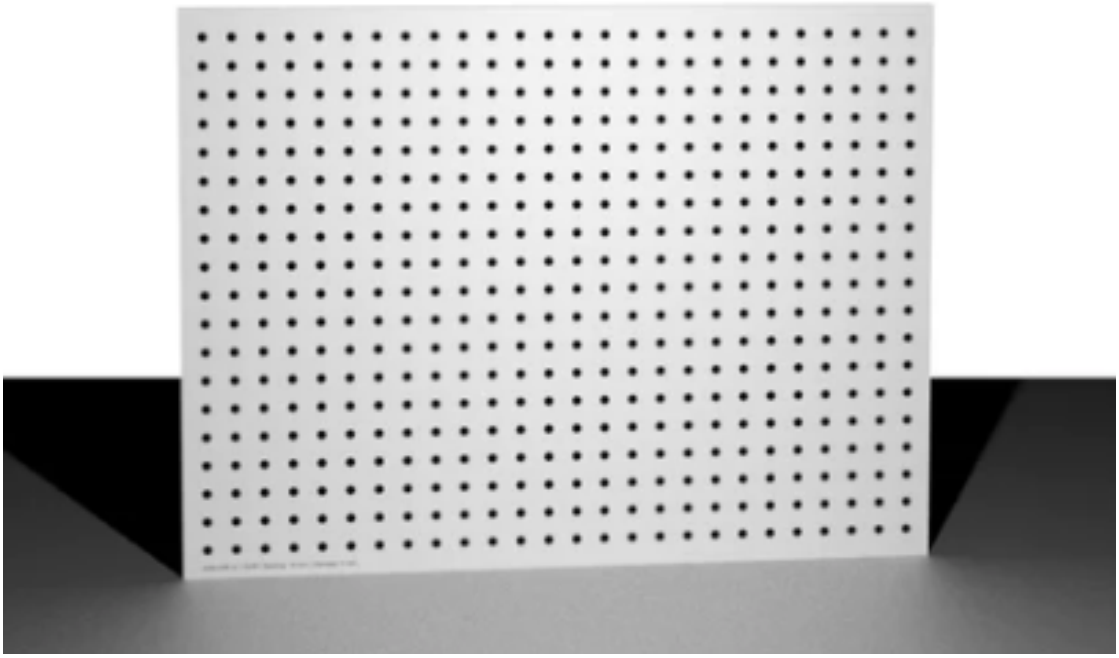

Figure 2.3: Calibration board. Sourced from *Calib.io* [23].

2.4 Comparator and aluminum table

In order to have a mechanical feedback on the efficacy of the correction algorithms in the final tests the camera and calibration board were switched for an mechanical comparator and a milled aluminum plate, (Figure 2.4) to simulate a real application. The mechanical comparator has a sensibility of 0.01 mm while the milled aluminum plate has a tolerance of 0.1 mm. This allows both to test how the correction algorithms would behave when the tool and the board would change between the data gathering step and the test phase as it would happen when they are deployed in a real application.

To measure the position errors in this situation the robot first tries to align the comparator's pointer to the four sides of each square on the plate, using all the four sides instead of just two allows to measure both positive and negative errors. Then the comparator's pointer is raised by 90 degrees and the robot has to align it to the top of each square. In the case considered in this thesis the vertical error was always negative but if the same procedure has to be executed on a robot with a positive vertical error a fixed offset can be used to allow the comparator to measure it.

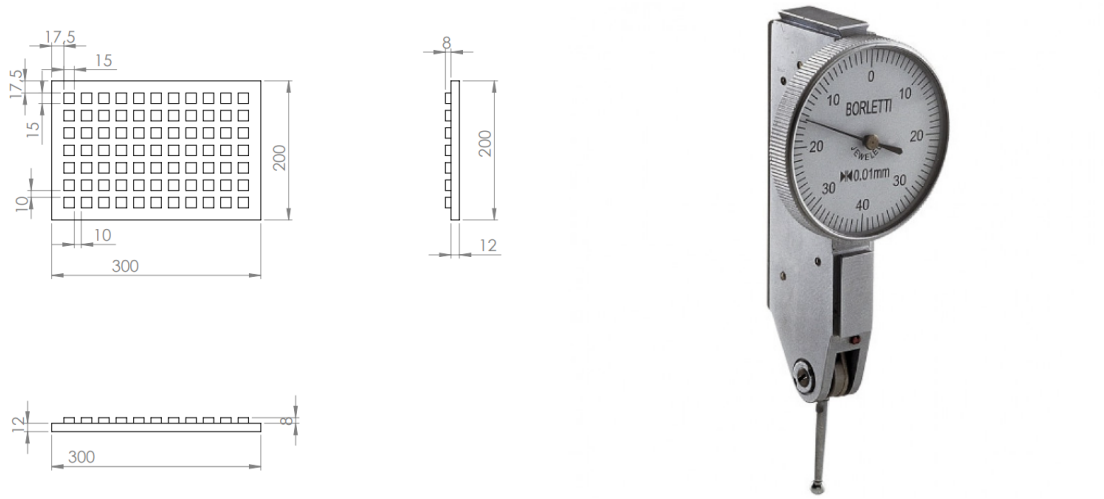

Figure 2.4: Specifications of the milled aluminum plate and comparator used to obtain a mechanical feedback.

2.5 Software

- DobotStudio Pro: is the Dobot proprietary software used to control all Dobot robots by programming them in Lua. It was used in this thesis in the 2.7.1 version to control the Mg400 during the experiments. It includes some useful features such as saving the current robot position and creating custom reference frames both in the space and attached to the current tool being used;

- DobotVisionStudio: is the Dobot proprietary computer vision software and it's the result of a joint effort between Dobot and Hikrobot. It allows to disjoint the code controlling the robot and the one controlling the camera and allowing the two to communicate via TCP/IP protocol. It includes some useful features such as recognizing patterns or shapes. It was used in the 4.1.2 version;
- Google Colab: is an online platform for writing and running Python code in a collaborative environment. It offers Jupyter Notebook-like functionality and integration with Google Drive. In this thesis it was used to analyze the data coming from the camera, create graphs and to train the neural networks model before importing them in DobotStudio Pro.

2.6 Experimental setup

2.6.1 Camera and calibration board

Figure 2.5 is a photo of the experimental setup for the data gathering step using the camera and calibration board. It is possible to see the how the Hikrobot MV-CE050-30UC camera is attached to the robot flange and can be moved by the robot over the calibration board. More details such as the 3D printed attachment to connect the camera to the robot flange, the camera calibration procedure and how it is possible to compute the position errors of the robot using the camera will be discussed in the next chapters.

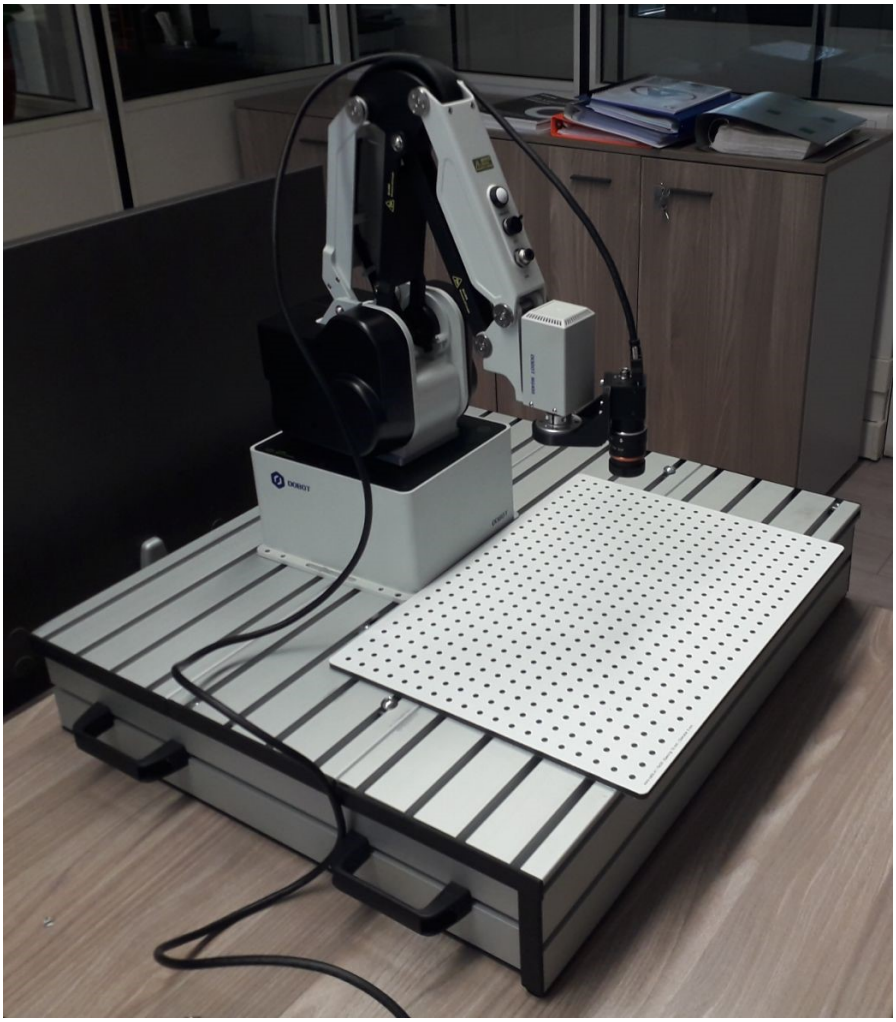

Figure 2.5: Photo of the experimental setup for the data gathering step using the camera and calibration board.

2.6.2 Comparator and aluminum table

Figure 2.6 is a photo of the experimental setup for the test phase using the mechanical comparator and the milled aluminum plate. Using two different setups to study the position errors of the robot offers two important advantages: first of all it can provide a physical feedback on the improvement of the robot accuracy achieved by the correction algorithms presented in this thesis and additionally it simulates a real application where the camera would be swapped with the actual tool so it allows to test how well the correction algorithms would perform in this case.

Figure 2.6: Photo of the experimental setup for the test phase using the comparator and milled aluminum plate.

3 Proposed approach

In the following sections different interpolation techniques proposed in the literature will be taken into consideration and evaluated against each other.

3.1 Interpolation techniques

Figure 3.1 [24] illustrates the setup for the data gathering process. A camera is mounted on the end effector of the manipulator which is moved over a calibration board at a constant height. The calibration board is composed of a series of black circles placed at constant distance between each other on the corners of an imaginary grid. All movements of the robot are in a 2D plane at the same height relative to the calibration board. In the figure it is possible to see four points of the calibration board: point P_{00}, point P_{10}, point P_{01} and point P_{11} with coordinates (X_0, Y_0), (X_1, Y_0), (X_0, Y_1) and (X_1, Y_1) respectively. In addition a fifth point is visible in the figure: point P'_{00}, with coordinates (X'_0, Y'_0).

After the data gathering process the corrections to apply to reach points P_{00}, P_{10}, P_{01} and P_{11} will be known, the goal of interpolation is to extrapolate the correction to reach point P'_{00} given these information. There are many interpolation techniques such as: linear interpolation, polynomial interpolation, fuzzy interpolation, spline interpolation [25] and so on. In this thesis we will take

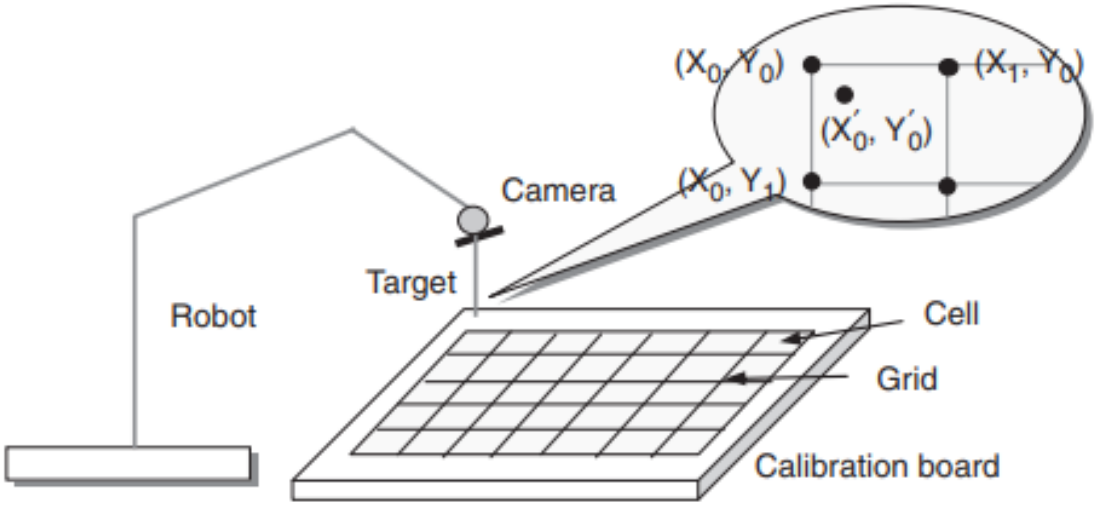

Figure 3.1: Set up for local calibration. Sourced from "Improving Position Accuracy of Robot Manipulators Using Neural Networks" [24].

into consideration linear interpolation, or in this case bilinear interpolation since the camera moves on a plane parallel to the ground the function to interpolate has two free variables (x and y) and neural networks as universal function approximators [24]. Both methods can be implemented to work in Cartesian space, both on the position of the tool center point (TCP) or on the position of the flange. This is relevant because if the position error is caused by a mismatch between the nominal robot model used by the controllers and the actual robot model then both corrections will give the same results. This is because in this case each point in the workspace will have a nominal position and actual position and the calibration process will consist in finding the function that expresses the relationship between the two [26].

On the other side if the errors are caused by a series of mechanical slackness's or a missing correspondence between the nominal and actual values of the joints then the correction function will express the relationship between the joint positions either directly in joint space or indirectly through the position of the flange. So even changing the tool the correction to apply will be the same as long as the joint position is the same and not connected to the Cartesian position of the TCP that is dependant on the tool used. In the next sections the theory behind bilinear interpolation and neural networks will be explored, together

with some possible advantages and disadvantages of each method.

3.2 Bilinear interpolation

Bilinear interpolation is a method for interpolating functions of two variables by using linear interpolation repentantly [27]. Bilinear interpolation is performed using linear interpolation first in one direction, and then again in another direction. Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location. Looking at Figure 3.2 [17] for a visual reference, it is possible to see how the correction for a point P'(x', y') is computed. To simplify the notation in this section the correction along one axis for a generic point P will be referred as f(P). To compute f(P') we will have to look at the four points closest to it of which the correction is already known during the data gathering step. In the

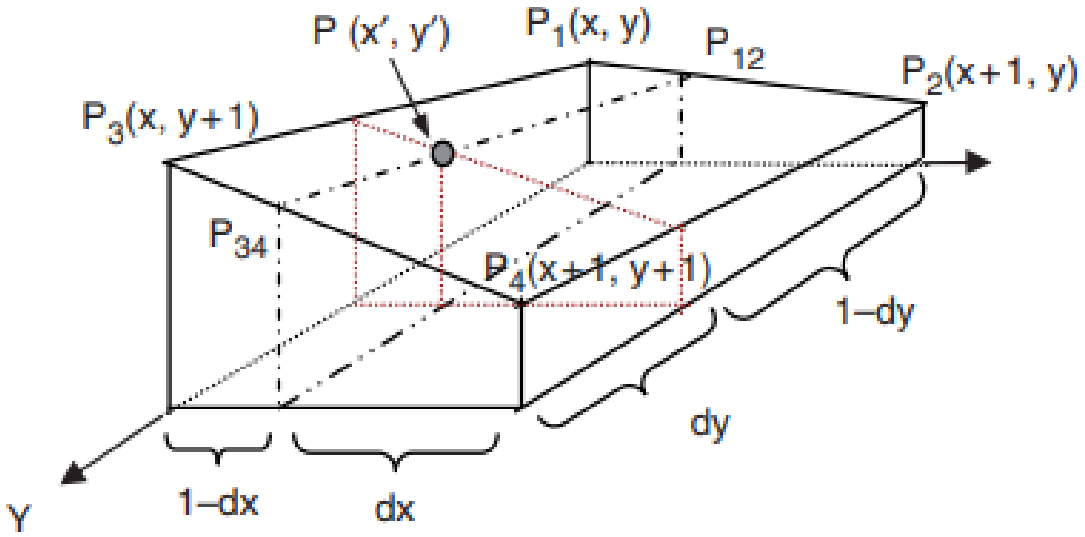

Figure 3.2: Graphical representation of bilinear interpolation. Sourced from "Robot manipulator calibration using neural network and a camera-based measurement system" [17].

case illustrated by the figure the four closes points are: $P_1(x, y)$, $P_2(x + 1, y)$,

$P_3(x, y+1)$ and $P_4(x+1, y+1)$, where 1 represents the unit distance between two points on the grid. The first step is to linearly interpolate P_1 with P_2 and P_3 with P_4 along the x axis obtaining P_{12} and P_{34}.

$$f(P_{12}) = \frac{dx}{dx + (1 - dx)} * f(P_1) + \frac{1 - dx}{dx + (1 - dx)} * f(P_2) \tag{3.1}$$

$$f(P_{34}) = \frac{dx}{dx + (1 - dx)} * f(P_3) + \frac{1 - dx}{dx + (1 - dx)} * f(P_4) \tag{3.2}$$

The equations above show how to obtain $f(P_{12})$ and $f(P_{34})$. Then the two are linearly interpolated along the y axis to finally obtain $f(P')$

$$f(P') = \frac{dy}{dy + (1 - dy)} * f(P_{12}) + \frac{1 - dy}{dy + (1 - dy)} * f(P_{34}) \tag{3.3}$$

Since the correction has to be applied along all three axis the same procedure will be repeated for each one. By applying the formulas seen above some important assumptions are made, the first is that each square on the grid is perfectly square and not slightly a parallelogram but also that the lines of the grid are parallel with those of the unit axis. The first assumption is addressed in the "Framework" chapter when discussing the calibration grids used while the second assumption will be discussed in further details in the implementation section.
Bilinear interpolation is the simplest interpolation technique presented in this thesis but nevertheless it can be quite effective. If the points on the grid are close enough, in fact, not much will differ between an optimal interpolation and a bilinear interpolation since the error is expected to change in a regular way without sudden jumps or shifts. In addition bilinear interpolation does not require training so as soon as the data gathering step is over it can be deployed immediately into the robotic application to improve the accuracy of the manipulator. Despite its many advantages bilinear interpolation has some drawbacks too: first of all

a lot of data has to be stored in memory, if the area of the workspace to map is large and the grid is very dense the amount of points to save will increase consistently. Modern computer can store huge amount of data without problems but it can become cumbersome in terms of time to find the four closest points to interpolate to make corrections. Additionally increasing the number of dimensions such as by switching from 2D to 3D or from Cartesian space to joint space [28] will increase the complexity of computations significantly. Finally bilinear interpolation lacks reversibility due to its inherent information loss during the process. The interpolation method computes weighted averages of nearby grid points, leading to multiple possible original configurations for a given interpolated value. This absence of one-to-one correspondence prevents exact recovery of the original data from the interpolated values.

3.3 Neural network

An Artificial Neural Network is a computing system based on a collection of connected units or nodes called artificial neurons, the name is derivative from the loose similarity with the neurons in a biological brain. An artificial neuron, just like its biological counterpart, receives signals from other neurons then processes them and can send signals to neurons connected to it [29]. The connections mirror the synapses in a biological brain, can transmit a signal to other neurons. Each neuron receives a number of inputs "signals" as real numbers from other neurons, the output of each neuron is computed by some non-linear function of the sum of its inputs and is transmitted to other neurons. The connections are called edges. Edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing multiple layers.

Figure 3.3 [30] illustrates more in details the functioning of a single artificial

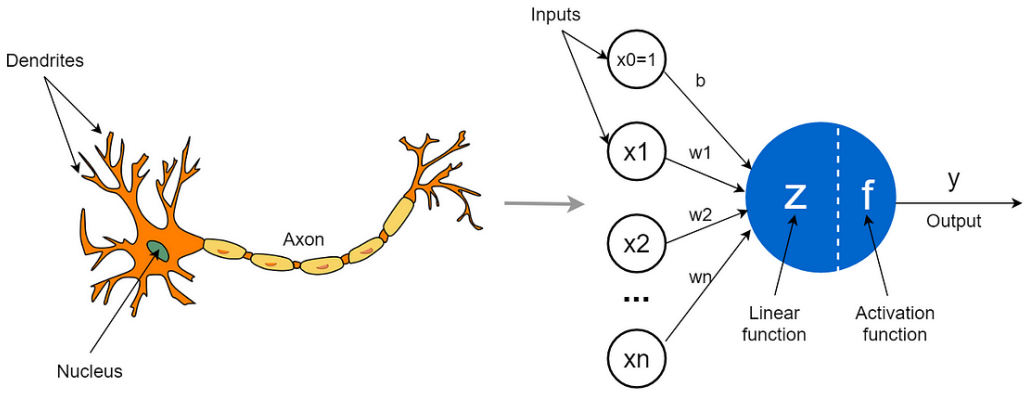

Figure 3.3: Comparison between a biological and artificial neuron. Sourced from *Towards Data Science* [30].

neurons and the comparison with its biological counterpart. The first artificial neural networks attempted to exploit the architecture of the human brain to perform tasks that conventional algorithms had little success with. They soon reoriented towards improving empirical results, abandoning attempts to remain true to their biological counterpart. Each simulated neuron is connected to other nodes via links, it has inputs and produces a single output which can be sent to multiple other neurons, like a biological axon-synapse-dendrite connection. All the nodes connected by links take in some data and use it to perform specific operations and tasks on the data. Each link has a weight, determining the strength of one node's influence on another, allowing weights to regulate the strength of the signal between neurons.

The initial inputs are external data, the outputs of the final output neurons of the neural net instead accomplish the task, such as classifying a sample, producing a real value or recognizing an object in an image. To find the output of the neuron it is necessary to take the weighted sum of all the inputs, weighted by the weights of the connections from the inputs to the neuron. A bias term is added to this sum. This weighted sum is called the activation. The weighted sum is then passed through an activation function to produce the output. It is important to use a nonlinear function as activation for the neurons that compose the hidden layers, otherwise the resulting network will be equivalent to a network with a single layer. The neurons are typically organized into multiple layers, neurons of one layer connect only to neurons of the immediately preceding and

immediately following layers.

Between two layers, multiple connection patterns are possible. The networks considered in this thesis are "fully connected", with every neuron in one layer connecting to every neuron in the next layer. Common activation functions include the rectified linear unit, sigmoid and hyperbolic tangent. Neurons with structure that forms a directed acyclic graph are known as feedforward networks. Alternatively, networks that allow connections between neurons in the same or previous layers are known as recurrent networks.

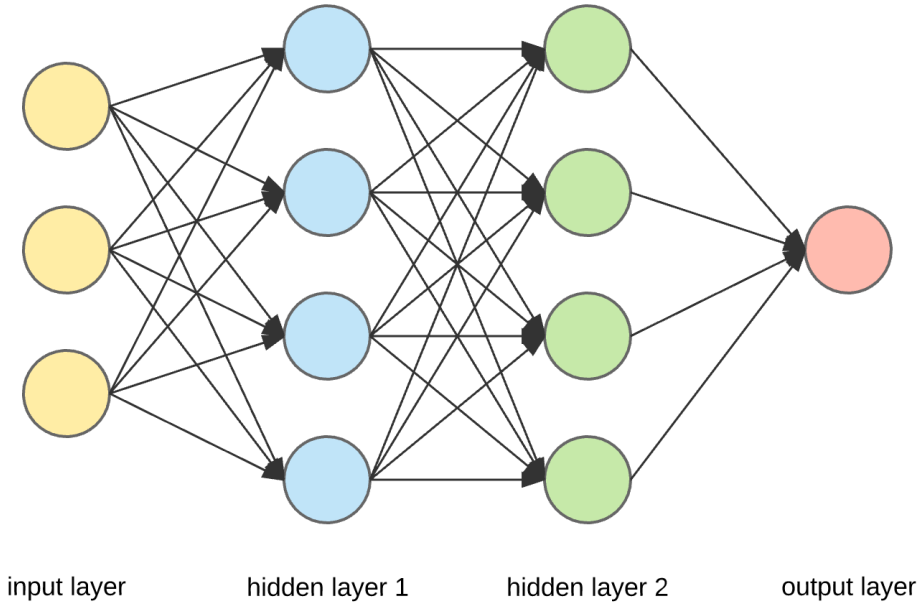

Figure 3.4: Illustration of the structure of a simple artificial neural network. Sourced from *UpGrad* [31].

Figure 3.4 [31] illustrates the structure of a simple feedforward artificial neural network. Each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another. This particular network has the input layer composed of three neurons, this means that the network can process data belonging to $\mathbb{R}^3$. Then there are two fully connected hidden layers of dimension four and an output layer composed of a single neuron, so the output of the network will belong to $\mathbb{R}$.
To use an artificial neural network in a real application a learning process is necessary to set the weights of each connection. Learning involves adjusting the weights (and optional thresholds) of the network to improve the accuracy of the result. This is done by minimizing the observed errors on sample observations

called train set. Learning is complete when examining additional observations does not usefully reduce the cost function, which is a function defined to represents the average distance between the output and the correct answer. Learning attempts to reduce the total of the differences across the observations.

To implement learning in a feedforward neural network algorithms such as backpropagation are used. Backpropagation computes the gradient of a loss function with respect to the weights of the network for a single inputoutput example, and does so efficiently, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule; this can be derived through dynamic programming. Variants of gradient descent, such as stochastic gradient descent, are commonly used [32]. The term backpropagation strictly refers only to the algorithm for computing the gradient, not how the gradient is used to update the weights. However, the term is often used loosely to refer to the entire learning algorithm, including how the gradient is used to update the weights, such as by gradient descent or stochastic gradient descent. The learning rate defines the size of the corrective steps that the model takes to adjust for errors in each observation. A high learning rate shortens the training time, but with lower ultimate accuracy, while a lower learning rate takes longer, but with the potential for greater accuracy. In order to avoid oscillation inside the network such as alternating connection weights, and to improve the rate of convergence, refinements use an adaptive learning rate that increases or decreases as appropriate and the concept of momentum allows the balance between the gradient and the previous change to be weighted such that the weight adjustment depends to some degree on the previous change.

The main advantages of artificial neural networks are their ability to reproduce and model nonlinear processes, this is proven by the universal approximation theorem that illustrates how the multilayer perceptron is a universal function approximator. In addition, as it will be shown in the following subsections neural networks offer the flexibility to easily change the dimensions of input and output and to be able to train with additional data without changing the dimensions of the calibration grid. Artificial neural networks also have important drawbacks too, fist of all the number of training samples needed to optimize the model increases exponentially. This is a particular harsh problem in the application considered in this thesis since the calibration grid can only contain so many points. Furthermore not all the points can be used on the training

set, some random points will be left out for the validation set. This is necessary to prevent overfitting but it may create some "holes" in the coverage of the remapped workspace.

A lesser problem but still worth considering is the training time, while the correction model based on bilinear interpolation can be deployed immediately after the data gathering process the artificial neural network will need a few minutes to train on the dataset before being able to make new predictions but in exchange once trained the predictions can be made in constant time with respect to the size of the mapped workspace and the number of points on the calibration grid saved. An additional advantage of the artificial neural network over the bilinear interpolation is that while the bilinear interpolation can only predict the correction form points that are completely inside the mapped workspace (unless a way of extending it artificially is implemented) the neural network can also process points outside the mapped workspace, even though its accuracy is expected to degrade quickly as the distance to the closest point in the mapped workspace increases. In the next subsections the architectures of the networks used will be considered with additional details.

3.3.1 Neural network in Cartesian space

Figure 3.5 [33] illustrates the architecture of the artificial neural network used to predict the corrections to apply in Cartesian space. Since the camera moves on a plane parallel to the ground the requested z coordinate is always constant so the only relevant input data are the x and y coordinates of the desired point that we want to reach, this is why the dimension of the input layer is two. On the other hand the correction has to be applied along all three axis so the dimension of the output layer is three.

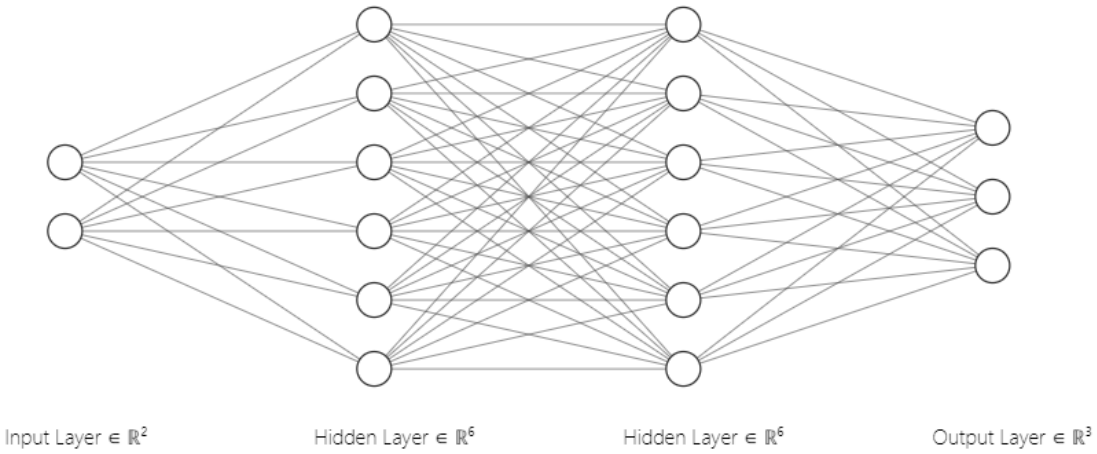

Figure 3.5: Architecture of the artificial neural network for predictions in Cartesian space. Drawn with $NN - SVG$ [33].

3.3.2 Neural network in joint space

Figure 3.6 [33] illustrates the architecture of the artificial neural network used to predict the corrections to apply in joint space. Even though the camera moves on a plane parallel to the ground the values of all four joints are constantly changing. But since the fourth joint does not contribute to the position of the flange and we are interested in the correction to apply at that specific point only the values of the first three joints are going to be used as input for the network. This explains why both the input and output layer are composed by three neurons.

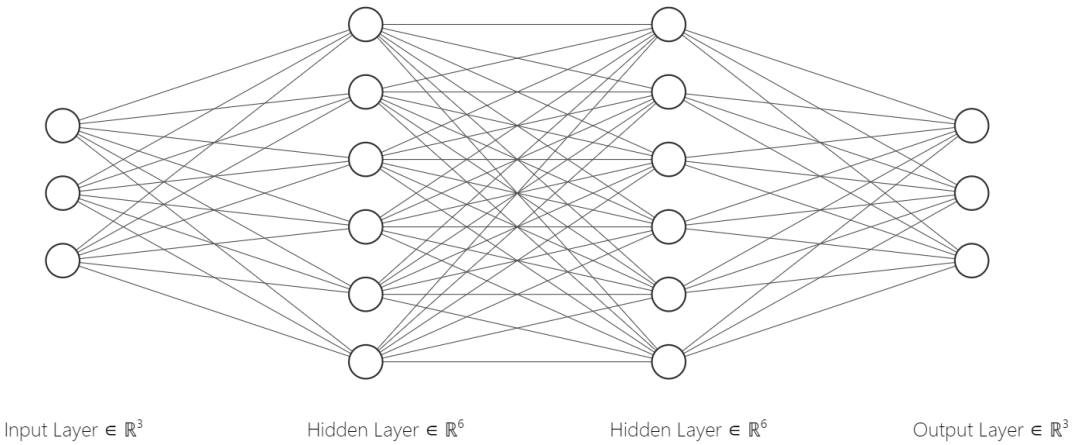

Figure 3.6: Architecture of the artificial neural network for predictions in Cartesian space. Drawn with $NN - SVG$ [33].

4 Implementation

This chapter details the steps requires to implement the calibration algorithms, the first sections will dive into the setup of all the equipment including mounting the camera on the robot arm and determining the new tool center point (TCP), then the data gathering process will be explored in depth together with the methods implemented to compute the error positions and lastly will come the correction tecniqes that can be applied to improve the accuracy with some important considerations to facilitate the transition between train and test.

4.1 Camera calibration

There are two possible types of setups for vision-based robot pose measurement. One is to fix cameras in the robot environment so that the cameras can see a calibration tool mounted on the robot end effector while the robot changes its pose [34]. The locations of the calibration tool in the world coordinate system can be computed by the cameras in various robot poses. This type of setups has two main advantages. First, it is non-invasive. The cameras are normally installed outside of the robot workspace, and need not be removed after robot calibration. Second, there is no need to identify the transformation from the camera to the end effector, although this transformation is easy to compute in

the case taken into consideration by this thesis. The major problem existing in all stationary camera setups is system accuracy. In order to have a large field of view for the stationary cameras, one has to sacrifice measurement accuracy. The second type of setups is to mount a camera on the end effector of the robot manipulator, this method is invasive, which may prevent it from being used in certain applications and it requires to find the transformation from the camera system to the tool system, but it resolves the conflict between high accuracy and large field-of-view of cameras as the cameras only need to perform local measurement so it was the one chosen for this thesis [35].

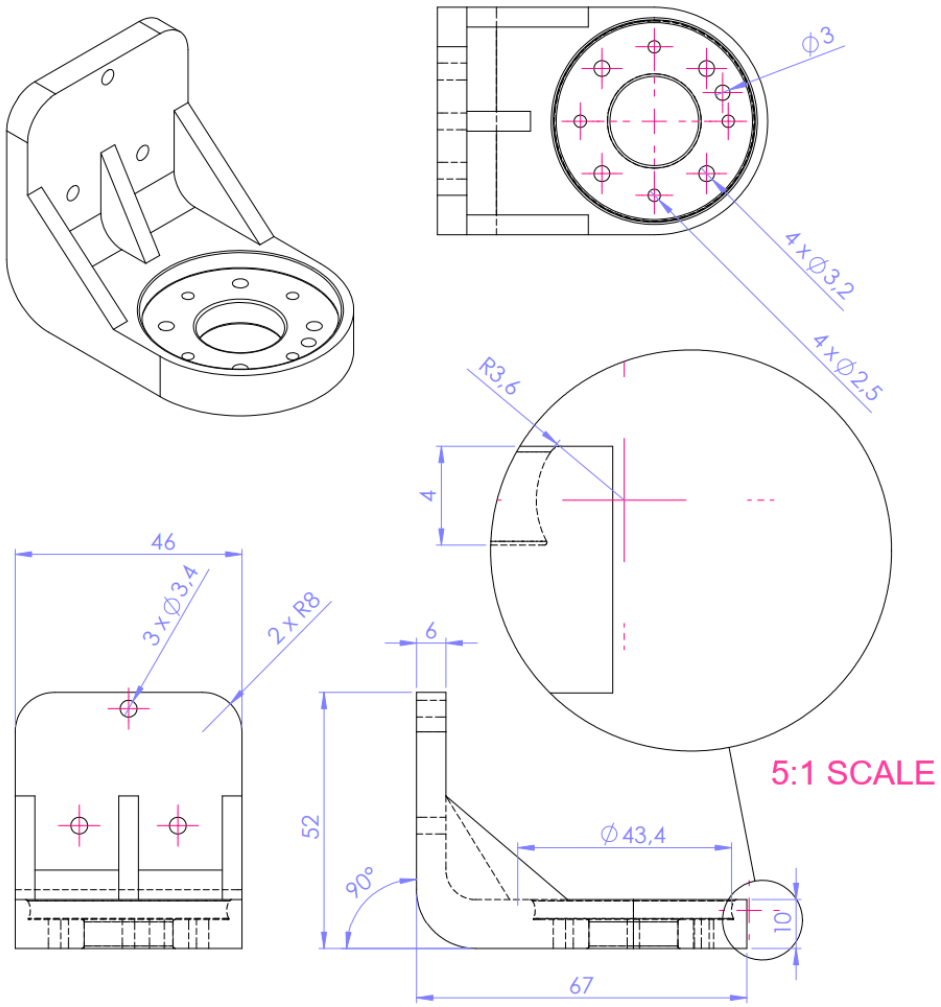

Figure 4.1: Perspective and section view of the 3D model for the camera attachment.

To mount the camera on the end effector of the robot a custom attachment

was modeled and 3D printed in PLA (Polylactic Acid). Figure 4.1 shows some perspective views and a section view. The view on the bottom left shows the face where the camera is attached so that it can face the ground and the view on the top right shows the slot where the flange can be fixed through the use of four screws. In the section view on the bottom right it is possible to notice how the slot for the flange is not flush but curved. This is a feature implemented to allow to partially regulate the orientation of the camera in case in which it is not perfectly orthogonal to the ground plane. During the data gathering process in fact, the camera has to be moved over the calibration grid keeping its orientation constant and to do the robot has to act on the values of the first and fourth joints.

In the unfortunate case in which the camera is not perfectly orthogonal to the ground changing the values of the first and fourth joints will also change the field of view of the camera in an undesirable way. To avoid this problem a simple procedure is iterated.

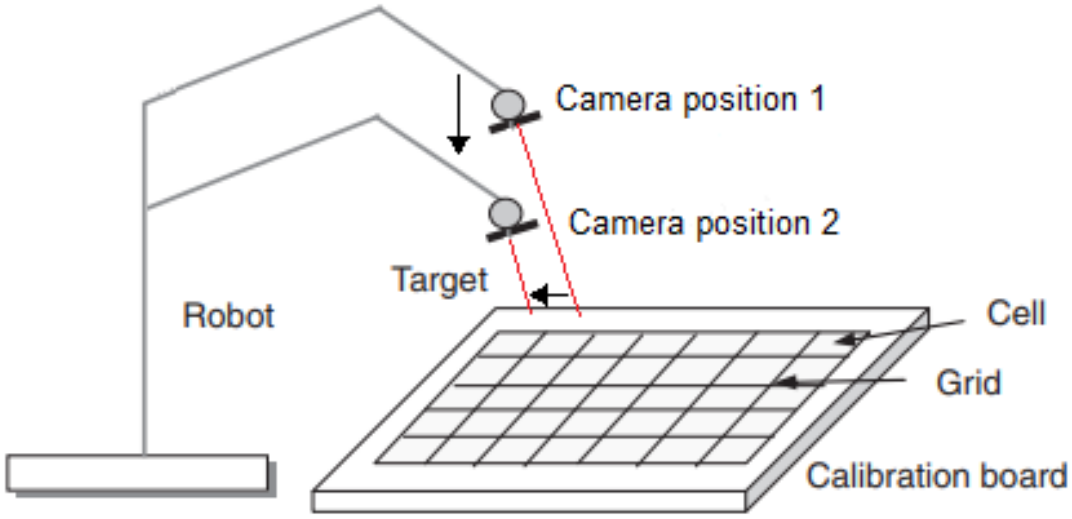

Figure 4.2: Set up for calibrating camera orientation. Sourced from "Improving Position Accuracy of Robot Manipulators Using Neural Networks" [24] and edited.

The core concept to calibrate the orientation of the camera is illustrated in Figure 4.2 [24], if the camera was perfectly orthogonal to the ground then by changing its height its field of view would remain centered on the same point, if instead the camera is oriented in a slightly different way then the center of the

field of view will shift according to Equation 4.1

$$dx = dz * \frac{sin(\alpha)}{cos(\alpha)} \qquad \alpha = tan^{-1}(\frac{dx}{dz}) \tag{4.1}$$

Where dz is the shift of the height of the camera which can be easily set through the manipulator parameters and dx is the shift of the center of the field of view of the camera. There are multiple ways to measure the horizontal displacement of the field of view, the simplest method is to mark on a sheet of paper positioned under the camera the center of the field of view before and after moving the camera and then measure the distance between the two with a ruler.

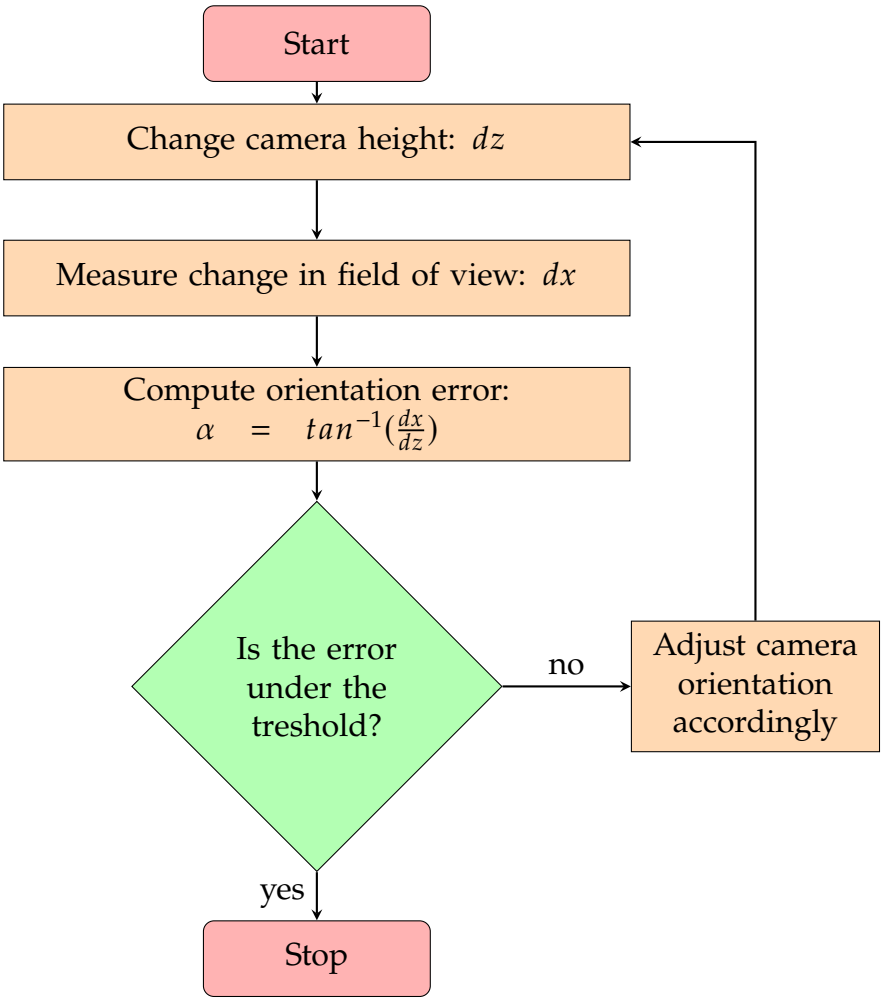

Figure 4.3: Iterative procedure to calibrate camera orientation

Alternatively if the software to compute the distance between the projection

on the ground of center of the field of view and a point on the grid that will be explained in the following sections has already been implemented then it is possible to align the center of the field of view of the camera with one of the points on the calibration grid before introducing the height displacement and run the program to compute the horizontal error afterwards. Independently by the method used, it is not essential to have an exact estimate of the error, also because the error may be influenced not only by the orientation of the camera but by the robot miscalibration too. An accurate measurement of the direction and an idea of the magnitude is enough to proceed with the iterative procedure shown in Figure 4.3 until the results are satisfactory.

At this point it is worth addressing why a camera calibration procedure as intended in computer vision terms [36] wasn't performed to discover the internal parameters of the camera with exact precision. First of all the camera used is intended for guiding the robot manipulator to pick object detected through a matching system and in order to do so it is already using high quality lenses with a distortion lower than 0.28%, additionally in the data gathering sections an in depth explanation will be provided of the methodologies used to avoid relying on the camera to directly compute the exact values of the errors. To do so the vision system will be used to guide the robot to the correct pose through an iterative procedure that aims to minimize the error between the requested position on the plane and the one provided by the robot manipulator when asked to reach said position.

4.2 Tool Center Point determination

When requesting the robot manipulator to move the end effector to a certain position we are usually asking to align the tool reference frame of the robot to the specific pose in the space. During the data gathering step the tool mounted on the robot flange is a camera so it is necessary to create a new tool reference frame to communicate to the robot that its end effector does not coincide with the flange anymore but an additional rototranslation is needed to align the two. The process of discovering the parameters of the rototranslation needed to align

the flange with the camera will be detailed in this section and it takes the name of tool center point determination. In the field of robotics, the Tool Center Point (TCP) plays a pivotal role in defining and controlling a robot's actions and movements. Referred to as the Tool Center Frame (TCF) or Tool Center Position (TCP), the TCP represents the specific location on a robotic end-effector or tool where the robot interacts with the environment or performs tasks, it serves as a reference point that determines the position and orientation of the tool with respect to the robot's base coordinate system. The TCP is typically located at the tip or end of the robot's tool and acts as the primary point of interaction with the surrounding environment. Accurately determining the TCP is essential because it defines the position and orientation of the tool in relation to the robot's base coordinate system. By knowing the TCP's location, the robot can accurately plan and execute its movements, including tasks such as grasping objects, manipulating tools, or performing precise operations.

In the case considered in this thesis the tool center point will be represented by just a translation along the plane perpendicular to the ground while its height will be the same as to the one of the flange. With this consideration aligning the tool center point with one of the calibration point will mean to move the camera directly over it, at a specific height, such that the center of the calibration point corresponds to the center of the image provided by the camera. It is very important to determine the tool center point correctly for this application, as the camera moves over the calibration grid it will keep a constant orientation with respect to the robot reference frame. To be able to do it consistently with the position of the tool center point the transformation needs to be as precise as possible.

DobotStudio Pro provides an easy to use interface to determine the tool center point transformation by aligning it over the same point but thought two different manipulator poses. This has the advantage of being a quick, ready to use method but it also has a very important drawback that makes it unfit for this application: if the robot has not been calibrated correctly and there is a mismatch between the nominal and actual positions of the end effector then performing the necessary alignments in one area of the workspace or another or using different orientations to achieve the two necessary poses to compute the tool center point will lead to different transformations as a result. In the case considered in this thesis considering just one alignment with just two different poses is not enough. It is necessary to use multiple alignments with multiple

different poses for each of them and to average them in some way to compensate the errors introduced by the missed calibration. Unfortunately DobotStudio Pro does not contemplate this possibility so a different more laborious method has to be implemented. Following, in this section the method used will be explained in depth but in order to do so a brief subsection on the theory behind rotation matrices, translation vectors and rototranslation matrices is necessary.

4.2.1 Rototranslation matrices

A rotation matrix is a mathematical construct used to describe the orientation of an object or coordinate system in three-dimensional space. It represents a pure rotation around a specified axis or a combination of rotations around multiple axes. A rotation matrix is denoted as R and is a 3x3 matrix of the form:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

where r_{ij} represents the elements of the matrix. To be considered a rotation matrix, the matrix R must satisfy the following properties:

1. orthogonality: the rotation matrix is orthogonal, meaning its transpose is equal to its inverse: $R^T = R^{-1}$. This property ensures that the length and orthogonality of vectors are preserved under rotation and allow us to quickly invert a rotation matrix by simply computing its transpose.

2. determinant: the determinant of the rotation matrix is either +1 or -1, depending on whether it represents a proper (rotation) or improper (reflection) rotation.

The rotation matrix is used to transform points or vectors in a coordinate system from one orientation to another. By multiplying a vector v_0 by the rotation matrix R, the transformed vector v' is obtained:

$$v' = Rv_0$$

Figure 4.4 [37] illustrates a simple example of this in 2D space.

Of particular interest for this application are elementary rotation matrices. The elementary rotation matrix, also known as a basic rotation matrix or elemental rotation matrix, is a specific type of rotation matrix that represents a rotation around a single axis in three-dimensional space. It is used to perform simple rotations around the x, y, or z axes. The elementary rotation matrices are denoted as $R_x(\theta)$, $R_y(\theta)$ and $R_z(\theta)$ for rotations around the x, y, and z axes, re-

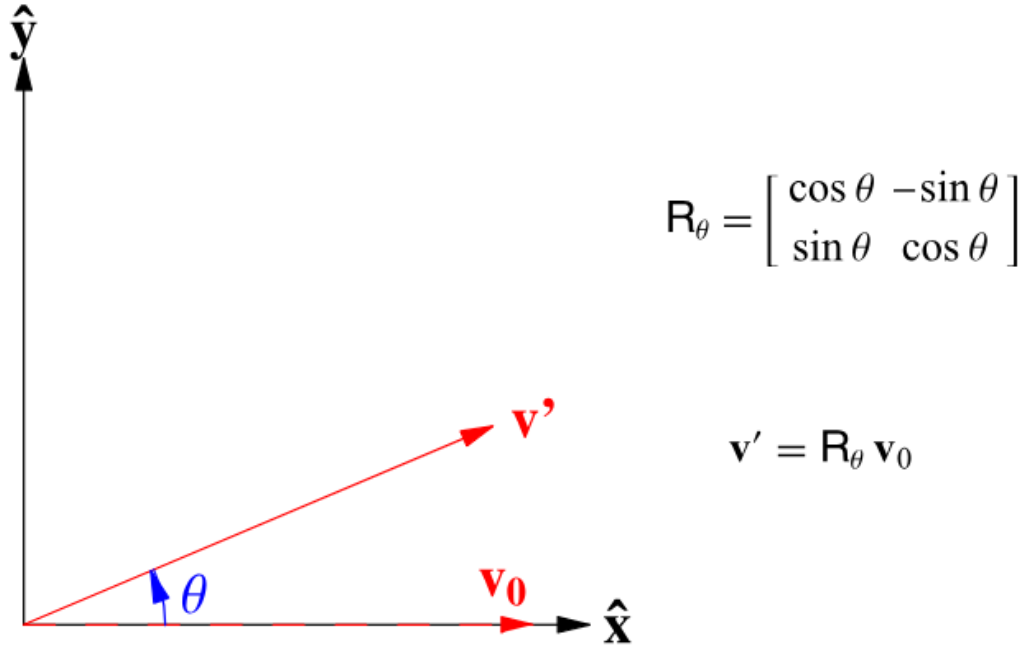

Figure 4.4: Illustration of the application of a rotation matrix on a vector in 2D space. Sourced form *Wolfram MathWorld* [37]

spectively. Each rotation matrix is a 3x3 matrix that depends on a single rotation angle θ. By combining these elementary rotation matrices, complex rotations and transformations can be achieved in three-dimensional space.

Since the Mg400 used for this thesis has a closed kinematic chain meant to keep the flange always parallel to the z axis, every rotation of the end effector can be expressed as an elementary rotation matrix $R_z(\theta)$ Where $\cos(\theta)$ and

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\sin(\theta)$ represent the cosine and sine of the rotation angle θ, respectively. This will greatly simplify a lot of computations in the following sections when the computation of the direct kinematic formulas of the manipulator will come into play.

Rotation matrices can also be applied directly to references frame as it can be seen from Figure 4.5 [38] where an elementary rotation matrix $R_z(\theta)$ is applied

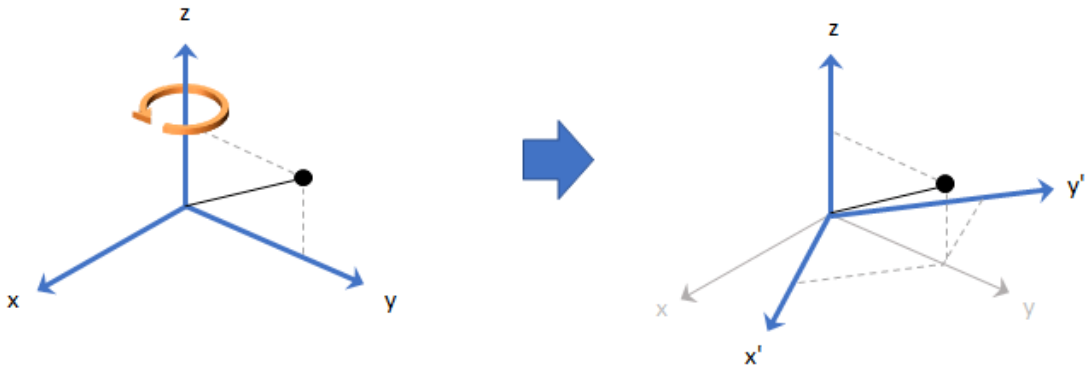

Figure 4.5: Illustration of the application of a elementary rotation matrix on a reference frame in 3D space along the z axis. Sourced from *MathWorks* [38].

to a reference frame in 3D space. It is possible to notice how the points in the space are not moving just like the black point represented in the figure but the same point will have different coordinates after the frame is rotated according to the formulas seen above.

Another important mathematical construct needed to compute the tool center point are the translation vectors. A translation vector represents the displacement or shift of an object or coordinate system in three-dimensional space. It describes the magnitude and direction of translation along the x, y, and z axes. A translation vector is denoted as t and is a 3x1 vector of the form:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

where t_x, t_y, and t_z represent the translation components along the x, y, and z axes, respectively. The translation vector is used to move points or vectors in a coordinate system by adding the translation vector to their coordinates. Given a vector v, the transformed vector v' incorporating the translation is obtained as:

$$v' = v + t$$

The translation vector represents a shift without any rotation, and it does not affect the orientation of the object or coordinate system. It is typically used in combination with rotation matrices to represent complete transformations

involving both rotation and translation.

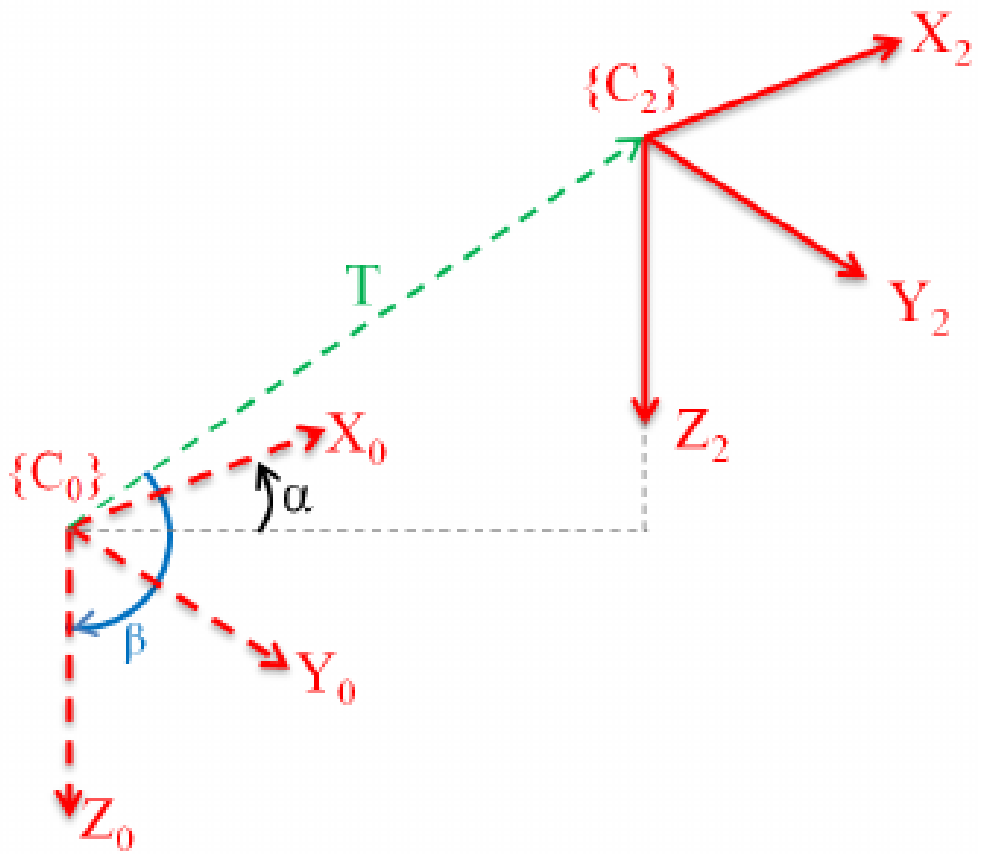

Figure 4.6: Illustration of the application of a translation vector on a reference frame in 3D space. Sourced from "2-Point-based Outlier Rejection for Camera-IMU Systems with applications to Micro Aerial Vehicles" [39].

Figure 4.6 [39] illustrates how a translation vector can be applied to a reference frame. In the figure reference frame 1 is translated along its y axis by a translation vector of length O_2O_1.

In general to align two reference frames both a rotation R and a translation t will be needed, as can be seen in Figure 4.7 [40], so a point P^b in the coordinates of the first reference frame can be rewritten as

$$P^a = R^a_b P^b + Ob^a$$

in the coordinates of the second reference frame, where in the notation the superscript represent the reference frame with respect to which the point, vector or matrix was written.

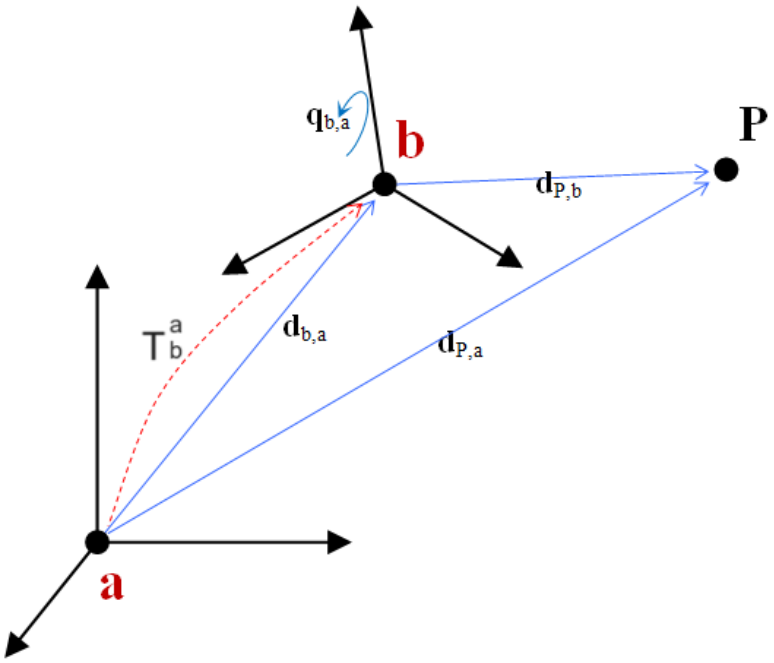

Figure 4.7: Illustration of the combined application of a rotation matrix and a translation vector on a reference frame in 3D space. Sourced from *Project Chrono* [40].

To simplify the notation it is possible to combine rotation and translation in a single matrix product by using homogeneous coordinates. Homogeneous coordinates provide a powerful framework for representing points and vectors in projective geometry, facilitating the efficient manipulation of transformations, including rototranslation. In homogeneous coordinates, a point in three-dimensional space is represented as a four-dimensional vector $[x, y, z, w]$, where w is a non-zero scalar usually set equal to one. This representation allows for the seamless integration of translations and rotations as part of the transformation matrix, enabling concise and efficient computations. Using homogeneous coordinates it is possible to combine a rotation matrix and a translation vector into a single matrix called a rototranslation matrix. The rototranslation matrix, denoted as T, is a 4x4 matrix that combines a rotation matrix R and a translation vector v to represent the complete transformation between two coordinate systems. It is defined as:

$$T = \begin{bmatrix} R & v \\ 0 & 1 \end{bmatrix}$$

or in extended form:

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & v_x \\ r_{21} & r_{22} & r_{23} & v_y \\ r_{31} & r_{32} & r_{33} & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where R represents a 3x3 rotation matrix, v represents a 3x1 translation vector, and 0 and 1 are placeholders to maintain homogeneity. Using this notation a point P^b in the coordinates of the first reference frame can be rewritten as

$$P^a = T_b^a P^b$$

in the coordinates of the second reference frame.

The rototranslation matrix provides a comprehensive representation of combined rotational and translational transformations in three-dimensional space. In robotics, it enables the precise control of robot end-effectors, facilitating tasks such as grasping, manipulation, and trajectory planning.

4.2.2 TCP calibration

Now that the topic of rototranslation matrices has been explored it is possible to apply this mathematical tool to the determination of the tool center point. For this application it is of crucial importance to calibrate the tool center point accurately [41], as the camera moves over the calibration grid it will keep a constant orientation with respect to the robot reference frame. To be able to do it consistently with the position of the tool center point the transformation needs to be as precise as possible.

The robotic visual inspection system mainly consists of an industrial robot and a non-contact visual sensor, which is represented by a camera in this thesis. Figure 4.8 [41] illustrates the reference frames in play during the robot calibration procedure. The coordinate systems of the robotic visual inspection system consist of robot base frame (BF), end-effector frame (EF), camera frame (CF) and world frame (WF) [42]. The measured result of the visual sensor is usually transformed to the workspace frame. For a visual point P on the work piece, the

mapping relationship between the coordinate P_w in the world frame and P_c in the camera frame is expressed as follows:

$$P_w = T_b^w T_e^b T_c^e P_c$$

where T_c^e denotes the transform matrix between the camera frame and the robot end-effector frame, which is also called hand-to-eye relationship [43]. T_e^b denotes the transform relationship between the robot end-effector frame and the robot base frame. It can be obtained from the robot forward kinematic model and it will change as the robot moves. T_b^w is the transformation matrix between the robot base frame and the workspace frame.

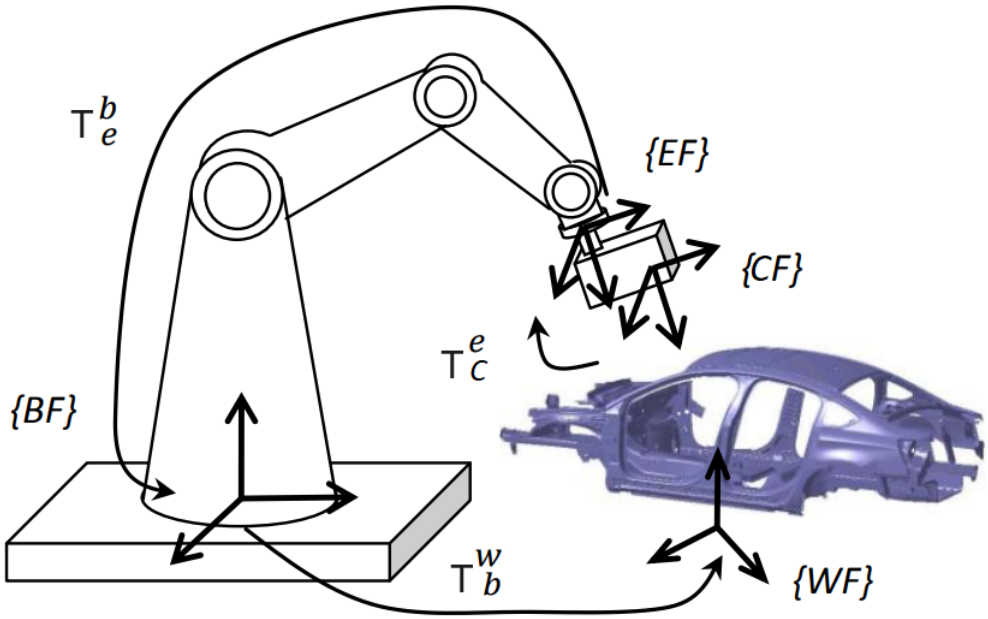

Figure 4.8: Illustration of the most important reference systems in play during vision based robot calibration. Sourced from "A Vision-Based Self-Calibration Method for Robotic Visual Inspection Systems " [41].

To determine the robot tool center point it is equivalent to determine T_c^e. Luckily the transformation between the camera reference frame and the end effector reference frame is always constant as the camera and end effector are locked together so it will be sufficient to perform the following procedure only once offline and then the results can be imported directly into DobotStudio Pro where the new tool reference frame can be set and used from now on. In this

thesis, the TCP position is defined as the intersection point of the camera optical axis and the plane of movement of the robot end effector. When the camera is mounted on the robot end-effector, the TCP is a fixed point with respect to the robot end-effector.

As mentioned before, DobotStudio Pro provides an easy to use interface to determine the tool center point transformation by aligning it over the same point thought two different manipulator poses. This is optimal to set up the robot quickly but if the robot has not been calibrated correctly and there is a mismatch between the nominal and actual positions of the end effector then performing the necessary alignments in one area of the workspace or another or using different orientations to achieve the two necessary poses to compute the tool center point will lead to different transformations as a result. To avoid this problem in this method, the robot is controlled to align the TCP to multiple fixed points at several different robot poses, that is to say, making the robot TCP position coincident with the fixed points. Figure 4.9 [44] illustrates an example of this, viewed from above the Mg400 can be mistaken from a two degrees of freedom manipulator since only two of its revolute joints are parallel to the z axis. A two degrees of freedom manipulator like the one in the figure could only reach the same point through two different poses but thanks to the two additional joints that the Mg400 is provided with every point in its dexterous workspace could theoretically be reached with any orientation along the z axis. Due to mechanical limitations this is not possible so a limited number of orientations was chosen to compute the tool center point of the camera.

For the purpose of this thesis five points were chosen in the workspace of the Mg400, each point was reached through three different poses. Assuming that X_b is the coordinate of the fixed point P in the robot base frame, R_{ij} and T_{ij} are the orientation and position of robot flange obtained for point i and pose j. X_t is the TCP position relative to the robot flange and the unknown that we are going to determine. Aligning the tool center point on the same point through three different poses allow us to write the following equations:

$$\begin{aligned} X_b &= R_{11}X_t + T_{11} \\ X_b &= R_{12}X_t + T_{12} \\ X_b &= R_{13}X_t + T_{13} \end{aligned} \tag{4.2}$$

Since X_b does not change it is possible to subtract the equations two by two and

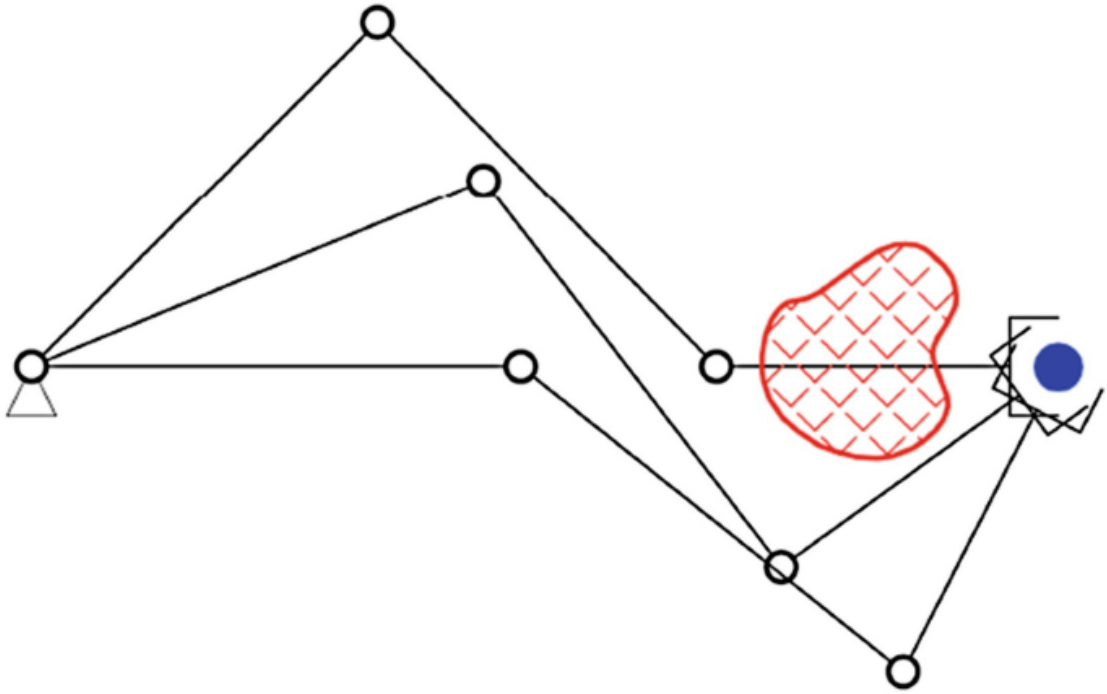

Figure 4.9: Illustration of how the robot manipulator can align the tool center point on the same point but with different poses. Sourced from "Encyclopedia of Systems and Control" [44].

rearrange them into a system:

$$\begin{aligned}(R_{11} - R_{12})X_t + &= T_{12} - T_{11} \\ (R_{12} - R_{13})X_t + &= T_{13} - T_{12}\end{aligned} \tag{4.3}$$

Since this operation was repeated across five point the ten resulting equations can be rewritten in matrix form:

$$\begin{bmatrix} R_{11} - R_{12} \\ R_{12} - R_{13} \\ R_{21} - R_{22} \\ R_{22} - R_{23} \\ \vdots \end{bmatrix} X_t = \begin{bmatrix} T_{12} - T_{11} \\ T_{13} - T_{12} \\ T_{22} - T_{21} \\ T_{23} - T_{22} \\ \vdots \end{bmatrix} \tag{4.4}$$

X_t can then be found by solving the redundant linear system with the form

Ax = b where

$$A = \begin{bmatrix} R_{11} - R_{12} \\ R_{12} - R_{13} \\ R_{21} - R_{22} \\ R_{22} - R_{23} \\ \vdots \end{bmatrix}, x = X_t, b = \begin{bmatrix} T_{12} - T_{11} \\ T_{13} - T_{12} \\ T_{22} - T_{21} \\ T_{23} - T_{22} \\ \vdots \end{bmatrix} \tag{4.5}$$

Which can be solved easily by a computer by means of singular value decomposition and least square method. It is also possible to compare the results obtained with this method with the ones provided by the two points calibration method of DobotStudio Pro and notice how they differ. To appreciate the improvement it is also possible to center the camera over a point on the grid and, once the tool reference frame is set, modify the orientation of the tool along the z axis. If the procedure above has been followed correctly it is possible to see how during the rotation the point in the center of the image captured with the camera stays aligned with the point on the grid. Otherwise the center of the image captured by the camera will slowly drift away from the point on the calibration grid and then align itself again once a full rotation along the z axis is completed. Of course since the robot is not calibrated correctly the procedure discussed above will only mitigate the effects and since the calibration error manifest itself differently in different regions of the workspace some drift will always be present even after the tool center point determination procedure.

Now that the setup is in place it is possible to begin discussing the techniques applied to measure the calibration errors. In the next section the error computation and data gathering steps will be analyzed in detail, discussing this and more.

4.3 Identification of the position errors

After performing a movement with the robot to be able to measure the displacement between the desired position and the actual position with the camera it is necessary to have a fixed point of reference [45]. In this thesis this role will be covered by the calibration grid. The particular grid used for this thesis is a white slate composed of aluminium/LDPE composite sheets covered by a grid of black circles with a diameter 4 mm and a constant distance between each center of 15 mm. Since it is not possible to easily obtain the position of a point on the grid with respect to the robot reference frame, other than by using the robot itself (which cannot be trusted since it has calibration issues), at the beginning of the data gathering step the tool center point previously determined is positioned over a dot on the corner of the grid so that the center of the dots coincides with the center of the image captured by the camera. Then a Lua script controls the robot to move the camera over each subsequent dot by using relative motions and the displacement error in Cartesian space can be determined by observing the error occurring in pixel space. Figure 4.10 illustrates this process, the white rectangle represent the field of view of the camera and how it slides form dot to dot.

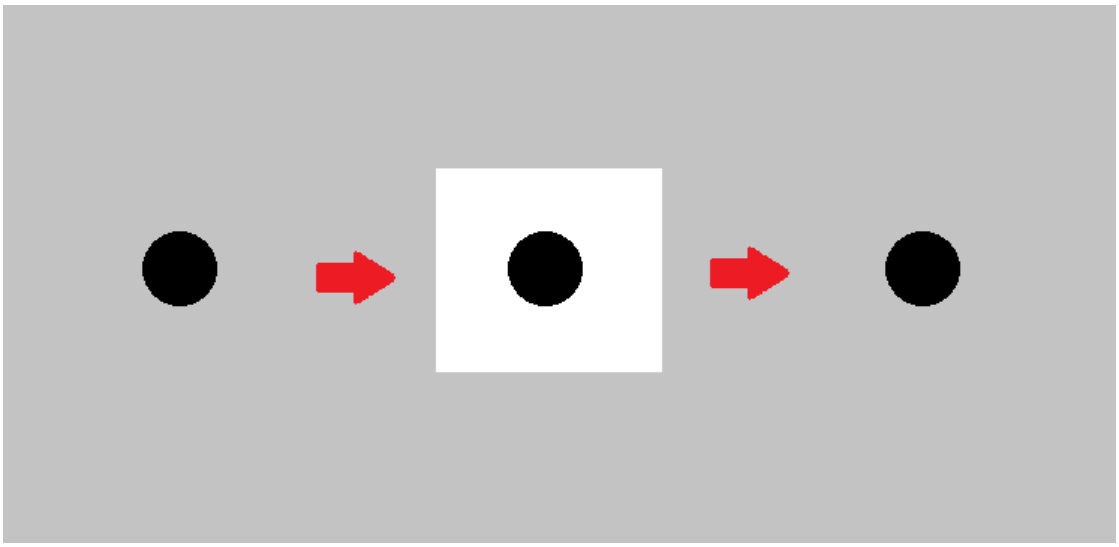

Figure 4.10: Illustration of how the view of the camera moves as the robot moves the camera across the calibration grid

For the purposes of this application it is necessary that the calibration grid is very precise. A small difference between the stated distance between the center of each dot and the actual one and every step of the robot this error will be added incrementally. It is easy to see how with a grid with rows of 26 dots like the one used for this thesis even a small error could then get enlarged very quickly and become a problem. Another relevant observation is that by aligning the tool center point with the first dot on the grid and then measuring the position error on each dot as the robot moves, at the end of the data gathering process the error measured on the first dot will always be zero. This will be discussed in more details in the Counter Correction sections of this thesis.

A similar problem could occur in the case in which the robot is not moving perfectly parallel to the rows of the grid. However it is not possible to position the grid in the workspace such that it is perfectly parallel to the robot reference frame. DobotStudio Pro gives us the possibility to create custom references frames with just two points as input. So by aligning the tool center point of the robot to a dot in a corner of the calibration grid and then to a dot on the corner on the same row but on the opposite side the new origin will be placed on the first saved point with the height set as the height of the end effector. The z axis will be parallel to the z axis of the robot reference frame, the x axis will be placed orthogonal to the z axis and pointing towards the second saved point and the y axis will be placed such that it is orthogonal to both the x and z axis to create a right handed reference frame. By creating a new custom reference frame in this way it it possible to move across the columns and rows of the calibration grids using fixed increments of x and y respectively.

This method of generating custom reference frames allows for a quick development of robot applications but there are some important limitations. Firstly the generated frames can be aligned with the robot reference frame with just a translation and an elementary rotation along the z axis, so no rotations along the x or y axis are allowed. This should not be a big problem for the Mg400 since rotations along the x or y axis are not part of its degrees of freedom. But consider the case in which the z axis of the robot reference frame is not orthogonal to the ground plane, in this scenario also all the z axes of the generated custom reference frames wont be orthogonal to the ground plane. Additionally no integrated function is present to transform the coordinates of a point between different coordinate systems and neither between flange and tool reference frames. Both these problems will be relevant later in this thesis so a different method to

move along the calibration grid will be used and to transform points between coordinate systems.

The following subsections will detail how the position errors are computed both on the x, y plane parallel to the ground and along the z axis orthogonal to the ground.

4.3.1 Circle detection

The Dobot environment already allows for the integration of a vision system where cameras are usually fixed and makes available a software to process the images, DobotVisionStudio. DobotVisionStudio allows to create a script by dragging and connecting together command blocks. Then a TCP server can be set in place, whenever the server receives a TCP message containing a chosen string it will run the script. So from the robot prospective a TCP client can be created in the Lua script and when the robot placed the camera over a dot on the calibration grid a TCP message is sent to DobotVisionStudio to trigger the script.

Figure 4.11 is a screenshot taken from DobotVisionStudio which shows the script used for this thesis. Figure 4.12 details the purpose of each block.

The first block is the image acquisition block, it allows to connect to a camera, an Hikrobot-MV-CE050-30UC in the case of this thesis, and to import the images obtained by the camera to the next block. It also allows to modify some important parameters such as the exposure time.

The second block is the most interesting one for the purposes of this thesis since it is the one dedicated to the circle detection. Unfortunately in the DobotVisionStudio manual there is no mention about the details of the detection algorithm but by looking at the output image in Figure 4.13 and the parameters that can be modified in the block some guesses can be made. A set of lines generate from the center of the image, represented in blue in the figure, the first time these lines encounter a sudden change in gradient the point in which it happened is saved, the green crosses, and then the resulting points are interpolated into a circle, highlighted in green. It is also possible to specify if the

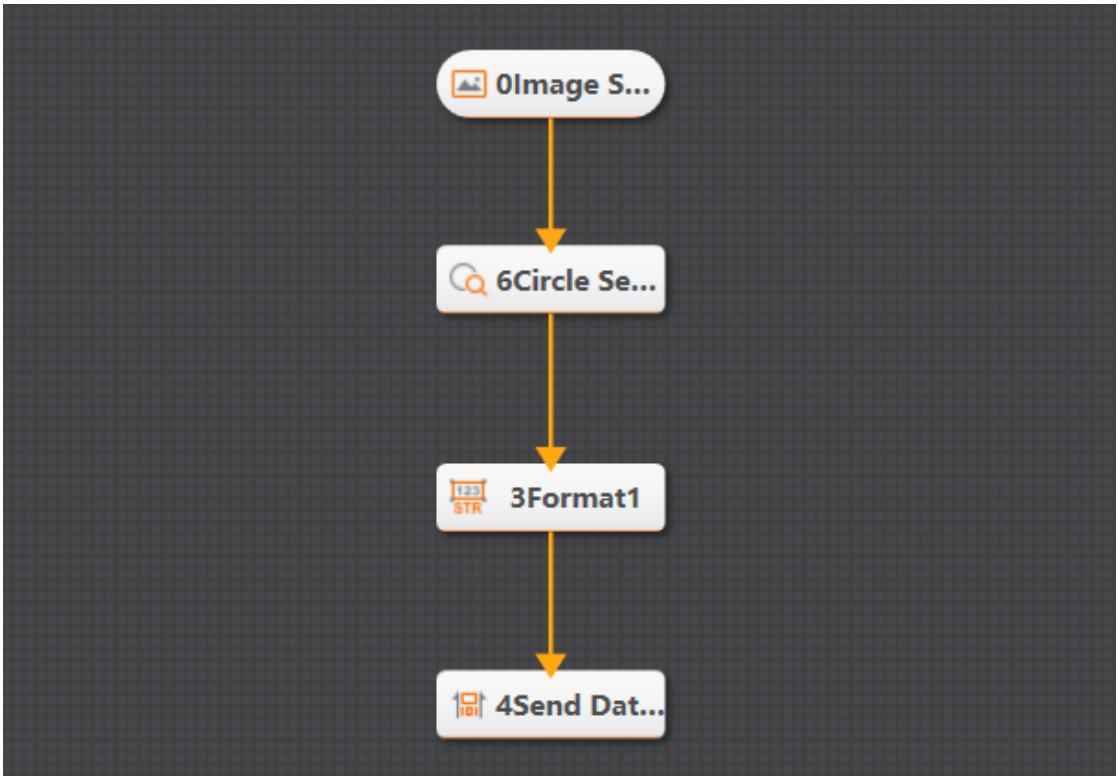

Figure 4.11: Screenshot from DobotVisionStudio of the blocky script to detect a circle on the image and to transmit the data back to the Lua script running on the robot

algorithm should focus on sudden rises or falls in the gradient to better identify the circles if they are dark dots on a bright background or bright dots on a dark background. The limitation of this algorithm is that it can only find a circle if the center of the image belongs to it or if it is very close to its border. This can become a serious limitation in the case in which the position error of the robot may be big enough that the center of the image falls outside the circle and the detection block won't be able to detect it. This means that having a starting guess about the order of magnitude of the error so that the calibration grid used can have dots big enough and with enough space between them such that this problem will not occur.

The third block takes the output information of the second block and concatenates the data into into a single string adding a ";" between each value to allow the TCP client to separate them once again. The second block has the possibility to output a variety of different information but for the purpose of this thesis only the radius and x, y postition of the center of the circle in pixel

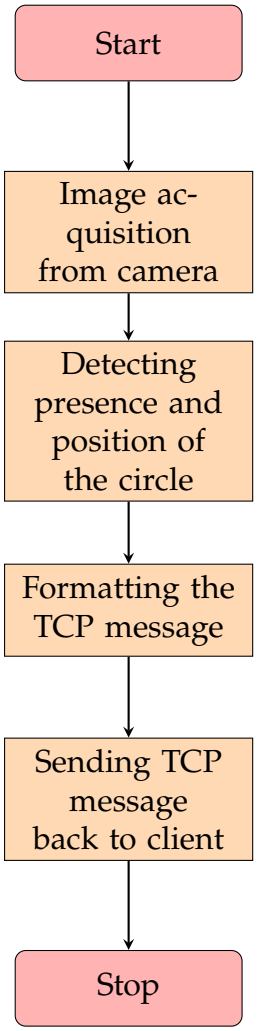

Figure 4.12: Flowchart of the DobotVisionStudio script used for detecting the presence and position of the calibration dots in an image

coordinates were used. The length of the radius, also expressed in pixels, can also be used as a control parameter too, if no circle is detected the returned radius will be zero and setting a threshold on the maximum length of the radius can prevent false positive that tend to happen when the center of the image falls slightly outside of the calibration dot.

Finally the last block is the one that actively sends the response string back to the TCP client once the script is over. The TCP client on the robot can then read the response string from the communication stream and parse it to extract all the necessary informations. In the following sections the steps necessary to turn the informations about the calibration dot detected by the camera into informations

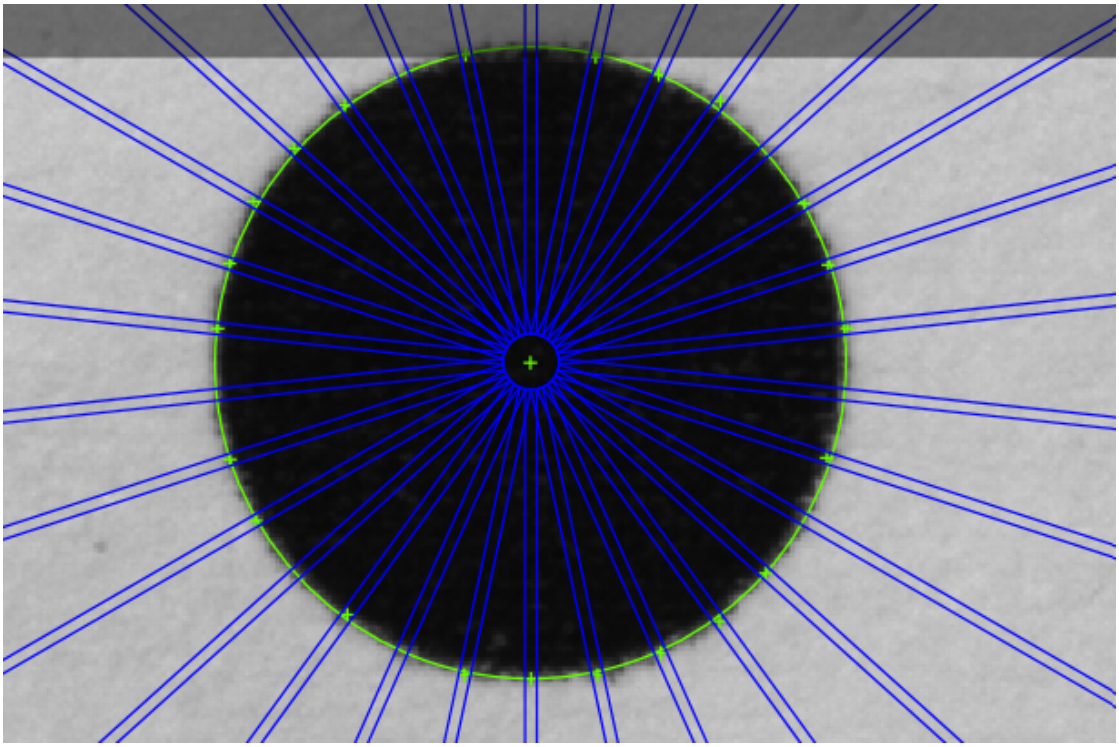

Figure 4.13: Screenshot from DobotVisionStudio of the Circle Search block and the result of it detecting a circle in the image

about the position error of the robot will be explored in depth, starting with the computation of the planar error along the x, y plane parallel to the ground plane and following with the error along the z axis orthogonal to the ground plane.

4.3.2 Planar Error computation

Figure 4.14 illustrates the output of the DobotVisionStudio in the case in which there is an error in the xy plane when the robot is asked to position the camera over one of the dots on the calibration grid. The dot on the calibration board will still be detected but its center will not coincide with the center of the image. The blue circle represents the center of the image and the blue cross the center of the detected circle. If the position of the end effector was correct the two would overlap but in the illustarted case they don't and the red arrow

highlights the displacement.

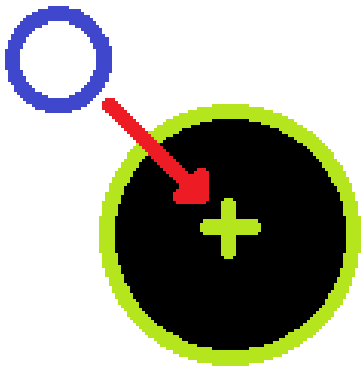

Figure 4.14: Illustration of the planar displacement as displayed in DobotVisionStudio, the blue circle represents the center of the image and the blue cross the center of the detected circle. If the position of the end effector was correct the two would overlap but in the illustarted case they don't and the red arrow highlights the displacement.

To convert this information into the error the robot manipulator committed when asked to move on the expected position it is necessary to follow a process composed of two steps.

First we need to compute the error in image coordinates an then to convert it in robot coordinates. Since the requirement for the robot was to align the tool center point over the dot on the calibration board to accomplish the first task it is sufficient to find the distance in pixel between the center of the image and the center of the circle. Since the image reference frame is positioned on the top left corner of the image as seen in Figure 4.15 [46] to compute the pixel coordinates of the center of the image we can halve the width and height of the image and thus obtaining the x and y coordinates of the center. The dimensions of the image provided by the camera are 2592 pixels in width and 1944 pixels in height so the coordinates of the image center in the image coordinate reference frame are going to be 1296 x and 972 y. To obtain the distance between the center

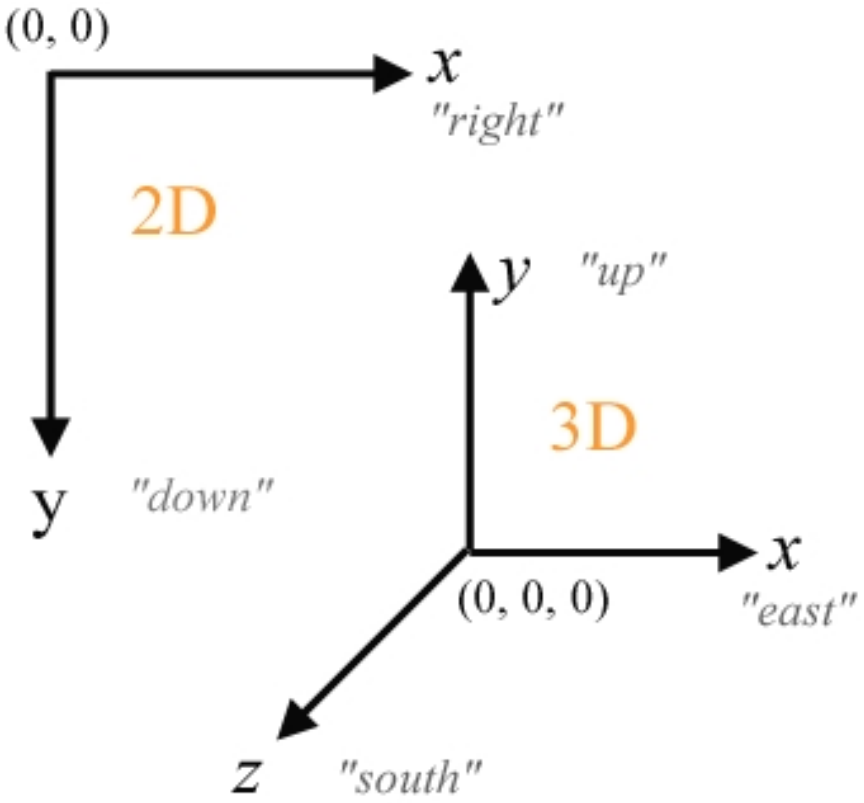

Figure 4.15: Coordinate reference frame of a 2D image compared to a coordinate reference frame in 3D space. Sourced from "Unreal Engine Physics Essentials" [46].

of the dot on the calibration board and the center along the axis of the image coordinate reference frame it is sufficient to subtract the two measures so:

$$d\_x = m\_x - c\_x \tag{4.6}$$

$$d\_y = m\_y - c\_y \tag{4.7}$$

$$d\_tot = \sqrt{d\_x^2 + d\_y^2} \tag{4.8}$$

Where m_x and m_y are the pixel coordinates of the center of the dot on

the calibration board returned by the DobotVisionStudio script and c_x and c_y are the pixel coordinates of the center of the image computed above. It is important to notice that c_x and c_y are always constant. d_x and d_y are the displacements computed along the x and y axis respectively and d_tot is the total displacement parallel to the horizontal plane.

It is possible to apply Pythagoras's theorem to also compute the total distance, this may be useful for statistical reasoning but to better understand how the position error is manifesting across the robot workspace it is also important to store separately the displacements along the axis of the image reference frame (and convert them in the robot reference frame too).

The second step is to find a relation that can allow us to convert the displacements form pixel coordinates into robot coordinates. This is where the radius of the dots of the calibration board comes into play and also one of the reasons why a circle pattern was used instead of a square pattern. By knowing the radius dimension both in mm and in pixels it is possible to set up a proportional relation of the type:

$$\frac{d\_mm}{r\_mm} = \frac{d\_pix}{r\_pix} \tag{4.9}$$

$$d\_mm = d\_pix * \frac{r\_mm}{r\_pix} \tag{4.10}$$

Where r_mm is the dimension of the radius of one of the dots on the calibration board measured in mm. For the specific calibration board used in this thesis 2.5 mm. r_mm represents the dimension of the radius of the dot on the calibration board over which the camera is positioned, expressed in pixels. This value is provided by the DobotVisionStudio script. d_pix represents the displacement measured in pixels, obtained as discussed above and d_mm is the displacement now converted in mm.

To convert the displacement from the image coordinate frame to the robot reference frame it is not sufficient, however to convert the measured distances from pixels to mm. The image coordinate frame is in fact a left handed reference frame. Meaning that if we imagine the z axis exiting from the image and parallel

to the robot z axis the two frames are not compatibles and cannot be aligned with a rotation.

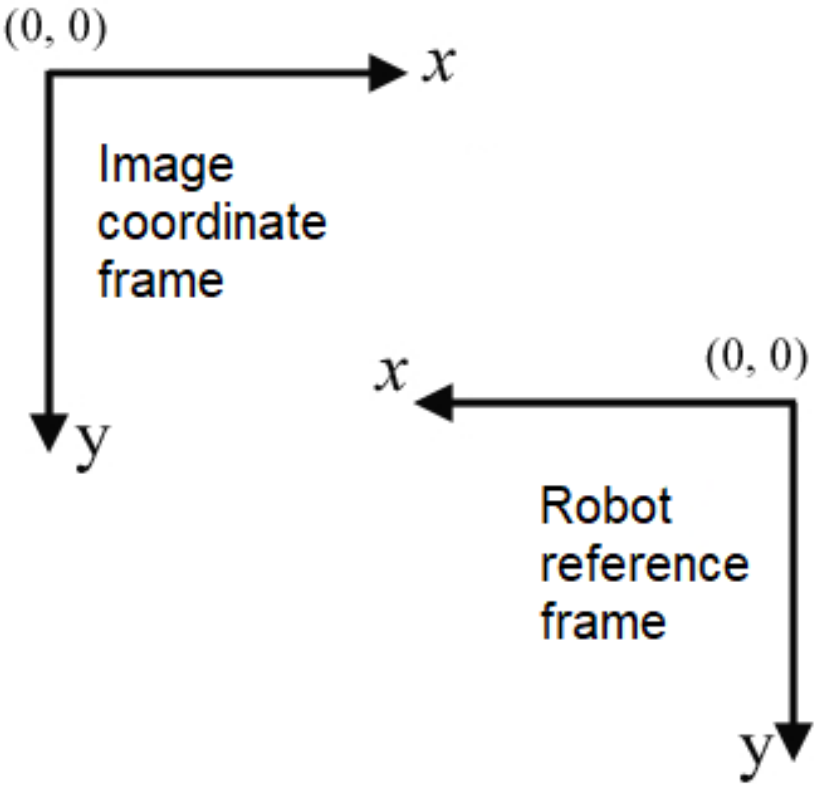

Figure 4.16: Illustration of the alignment between the image coordinate frame (on the left) and robot reference frame (on the right).

To simplify the conversion as much as possible it is possible to observe some precautions. When choosing the orientation of the camera and the position of the calibration board in the robot workspace it will be done is such a way that when the custom robot reference frame will be create the axis of the two coordinate systems will be parallel two by two. Figure 4.16 shows an example of this, in this case the y axis of the image coordinate frame and robot reference frame (the robot reference frame is aright handed reference frame so the z axis is exiting the image) coincide so no additional computations are necessary, the x axes on the other hand have the same orientation but opposite directions so to complete the conversion between the x displacement in image coordinates and the x displacement in robot coordinates it will be sufficient to swap its sign. To summarise everything said so far the following equations apply:

$$r\_d\_x = -(m\_x - c\_x) * \frac{r\_mm}{r\_pix} \tag{4.11}$$

$$r\_d\_y = (m\_y - c\_y) * \frac{r\_mm}{r\_pix} \tag{4.12}$$

Where r_d_x is the displacement of the robot along the x axis in mm, r_d_y is the displacement of the robot along the y axis in mm, m_y is the y coordinate in pixels of the match point, radius_ mm is the radius of the dot on the calibration grid expressed in mm, r_pix is the radius of the dot on the calibration grid expressed in pixels, c_x and c_y represent the coordinates of the image center expressed in pixels.

The following sections will contain detailed explanations on how to use the error just computed to correct the robot position but before it is necessary to compute the displacement along the vertical axis too. As it is possible to imagine, since the camera has been positioned in such a way that its optical axis is orthogonal to the xy plane, a vertical displacement of the robot will not translate into a displacement in the image obtained by the camera. So in the next section the methodology applied to estimate the distance between the camera and the calibration board using only the 2D image captured by the camera will be discussed in depth.

4.3.3 Vertical Error computation

In this section we will discuss the methodologies applied to estimate the vertical displacement of the robot. This is particularly problematic since the robot has not been calibrated properly which means its internal information about its position is not to be trusted so the only source of reliable data at our disposal is the 2D camera which does not provide any explicit information about

the depth of what it is seeing. The main idea to extract depth information from a 2D image is to use the knowledge we already have about the objects that appear in the image captured by the camera and the fact that objects further away from the camera appear smaller in the image than objects that are closer to the camera. Since we can use the informations about the planar displacement (along the x and y axes) to move the robot is such a way to align the optical axis of the camera with the center of the dot on the calibration grid it is then possible to relate the size of the dot in the image in pixels with the size of the dot in mm. Knowing the camera intrinsic parameters it is possible to estimate the distance between the calibration board and the camera sensor with a quite high level of accuracy. To properly understand the procedure it is necessary to introduce the mathematical camera model used in this thesis and what intrinsic parameters were employed.

Pinhole Camera model

The simplest way to model a camera is the pinhole camera model which describes the mathematical relationship between the coordinates of a point in three-dimensional space and its projection onto the image plane of an ideal pinhole camera, as can be seen from Figure 4.17 [47], where the camera aperture is described as a point and no lenses are used to focus light [48]. The pinhole camera model applies a lot of simplifications with respect to a real camera. The model does not include, for example, geometric distortions or blurring of unfocused objects caused by lenses and finite sized apertures. It also does not take into account that real cameras have only discrete image coordinates. This means that the pinhole camera model can only be used as an approximation of the mapping from a 3D scene to a 2D image. Its validity depends on the quality of the camera and, in general, decreases from the center of the image to the edges as lens distortion effects increase [49]. But this is fine for the application taken into account is this thesis since we are already trying to position the camera directly over the dot on the calibration board that is going to be used as a reference to compute the distance, so the influence caused by distortion should be minimal.

As can be seen from Figure 4.18 [50] in this model, we assume that light travels in straight lines and that the camera's lens can be reduced to a single point called the "pinhole". The key elements of the pinhole camera model are

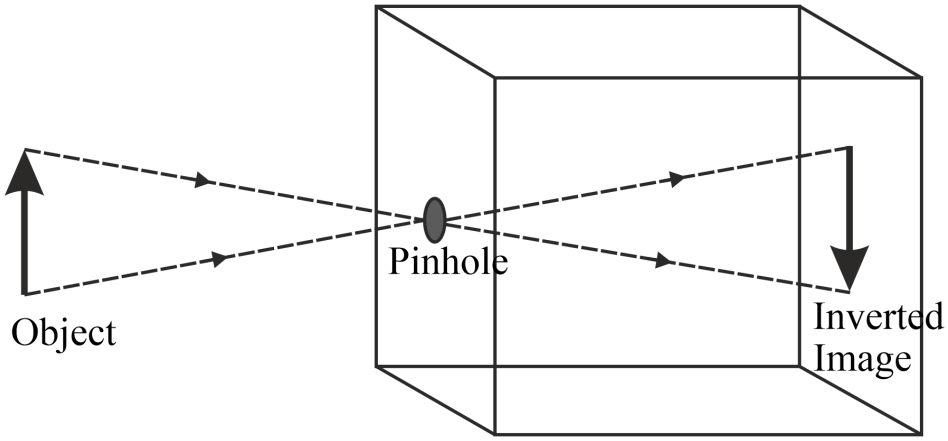

Figure 4.17: Illustration of the basic principle of the pinhole camera model. Sourced from *Embibe* [47].

as follows:

- pinhole (aperture): the pinhole is a small hole on the front surface of the camera, opposite to the image plane. It is through this hole that light enters the camera and forms the image. The size of the pinhole directly affects the sharpness and clarity of the resulting image. A smaller pinhole will produce a sharper image, but it will also reduce the amount of light reaching the image sensor or film. A larger pinhole on the other hand will allow more light to enter the camera but at the same time since more light rays will bounce on the object and enter the camera, multiple projections of the object will overlap on the image plane resulting in a blurred image. It is denoted as "C" in Figure 4.18 [50];

- image plane: the image plane is the surface inside the camera where the final image is formed. This is usually the camera's sensor in digital cameras or the film in traditional film cameras. When light passes through the pinhole, it projects an inverted image of the scene onto the image plane. The distance between the pinhole and the image plane is known as the "focal length" of the camera, which will be denoted as "f" in Figure 4.18

[50]. In the figure the image plane is moved by symmetry with respect to the camera center. This can be done to avoid including the inversion in the computations of the perspective projection;

- camera center (principal point): the camera center is the point at which the optical axis (the imaginary line passing through the pinhole perpendicular to the image plane) intersects the image plane. It is denoted as "P" in Figure 4.18 [50];

- optical axis: the optical axis is the straight line that connects the camera center (p in Figure 4.18 [50]) with the pinhole. This axis is crucial for understanding the perspective in the image;

- scene: the scene refers to the three-dimensional objects and points in the real world that the camera is capturing. These objects are projected onto the image plane to form a two-dimensional representation of the scene.

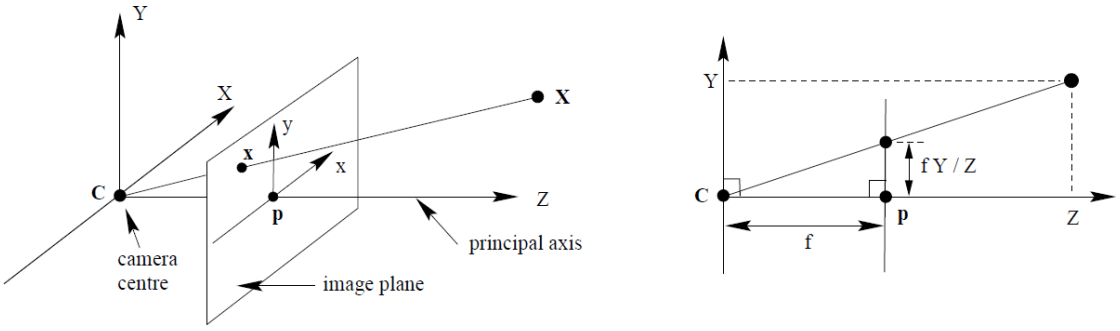

Figure 4.18: Detailed illustration of the basic basic geometry involved in the pinhole camera model. Sourced from "Multi-View Geometry in Computer Vision" [50].

The pinhole camera model can be mathematically described using perspective projection. If a point in 3D space is considered with X, Y, Z coordinates with respect to the world reference frame then its projection x, y on the image plane can be obtained by using the following equations:

$$x = -f * \frac{X}{Z} \tag{4.13}$$

$$y = -f * \frac{Y}{Z} \tag{4.14}$$

Where X, Y, and Z are the coordinates of the point in the world coordinate system, x and y are the image coordinates of the point on the image plane and f is the focal length of the camera. These equations show that the image coordinates are directly proportional to the world coordinates of the point and inversely proportional to its distance from the camera (Z). It is important to note that the negative signs indicate that the image formed is inverted. Additionally, the field of view (FOV) of the camera can be determined based on the sensor size and the focal length. The FOV is the angular extent of the scene that is captured by the camera and is typically measured in degrees or radians.

Thin lens equation

The thin lens equation is a formula that relates the distances of an object and its image formed by a thin lens. It provides a simple way to predict where the image will be located based on the object distance and the focal length of the lens. In this equation, we assume that the lens is thin, meaning its thickness is negligible compared to its other dimensions [51]. This is an idealization but we will discuss how in the application considered in this thesis it is possible to make these assumptions without loss of generality.

The thin lens equation can be stated as follows:

$$\frac{1}{f} = \frac{1}{d_o} * \frac{1}{d_i} \tag{4.15}$$

Where f is the focal length of the lens, d_o is the is the object distance or the

distance between the object and the lens, called u in the illustration shown in Figure 4.19 [52] and d_i is the image distance or the distance between the lens and the image formed, called v in the illustration.

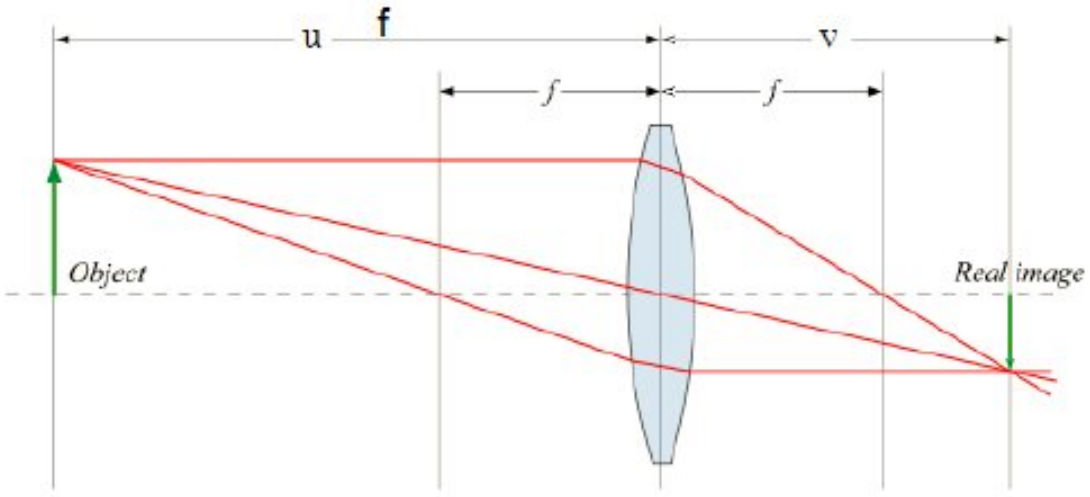

Figure 4.19: Illustration of the geometry involved in the thin lens equation. Sourced from "Copper for particle accelerators: electron stimulated desorption and study of hydrogen content measurement by laser ablation" [52].

The equation indicates that the reciprocals of the focal length and the object and image distances are related. The focal length is a property of the lens, while the object and image distances can vary depending on the position of the object with respect to the lens. This equation allows us to determine the image distance or object distance given the other two values. For example, if we know the focal length and object distance, we can find the image distance using the equation. Similarly, if we know the focal length and image distance, we can find the object distance, just like we aim to do in this thesis.

When the image distance d_i is positive, the image is formed on the opposite side of the lens from the object, meaning it is a real image. A positive d_o indicates that the object is on the same side as the incident light (the side from which the light is coming). Conversely, when the image distance d_i is negative, the image is formed on the same side as the object, meaning it is a virtual image. In this case, d_o is also negative. The thin lens equation is a powerful tool used in optics for understanding and predicting the behavior of light passing through lenses in various optical systems, including cameras, telescopes, and eyeglasses.

Computing the vertical displacement

Now that we have all the theoretical knowledge needed at our disposal it is time to actually compute the vertical displacement.

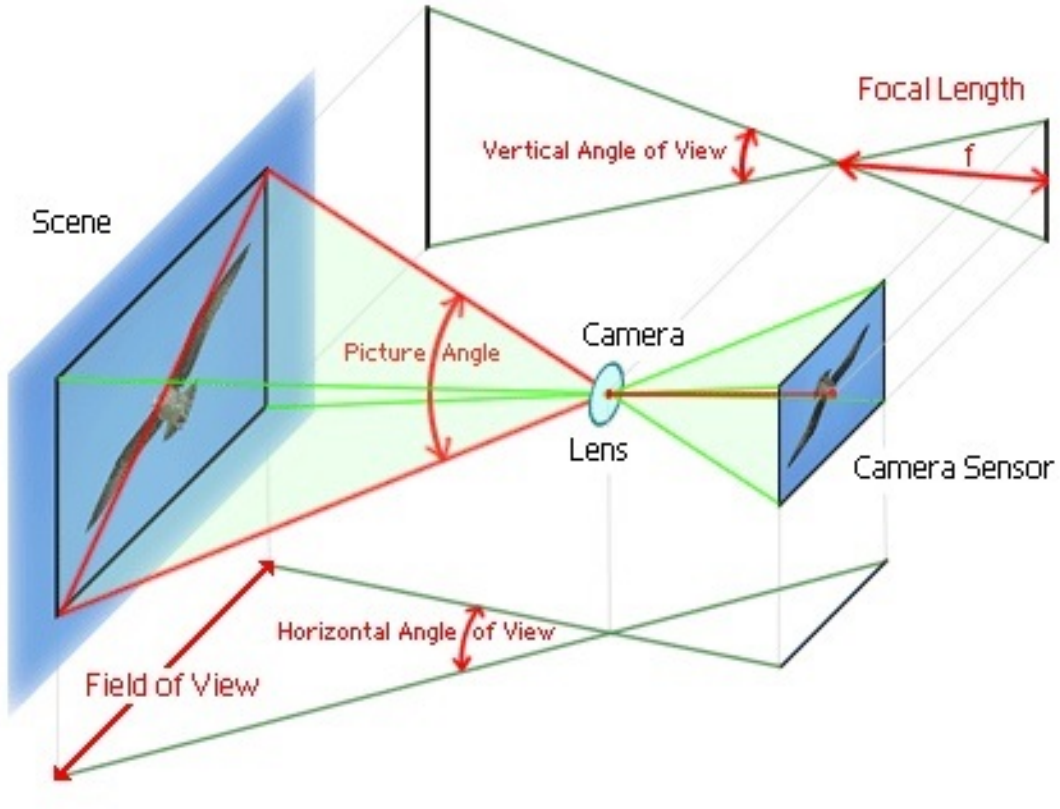

Figure 4.20: Illustration of the how the objects seen by the camera are projected through the lenses onto the surface of the sensor. Sourced from "Multiple sensors' lenslets for secure document scanners" [53].

Figure 4.20 [53] can help to visualize the main concept that allows us to compute the vertical displacement using only the 2D image and the prior knowledge we have about the object and the camera. As seen in the figure the light bounces off on the surface of the object. Then the light rays reflected by it pass through the lenses of the camera and are projected on the sensor. So by knowing the dimension of the object, its orientation and the camera intrinsic parameters we can estimate the distance between the object and the camera sensor.

Figure 4.21 provides a more stylized representation of what was just described but it is more helpful to relate the following computations to their physical meaning. In the figure on the left is represented the dot on the calibra-

tion board with it's radius expressed in mm. The horizontal line represents the camera optical axis which should be centered on the dot except for a hopefully small horizontal displacement caused by the robot manipulator. On the right is represented the dot on the calibration board projected on the camera sensor with it's radius expressed in pixels. The distance between the camera lens and the object is represented by d_o and the distance between the camera lens and the sensor is represented by d_i.

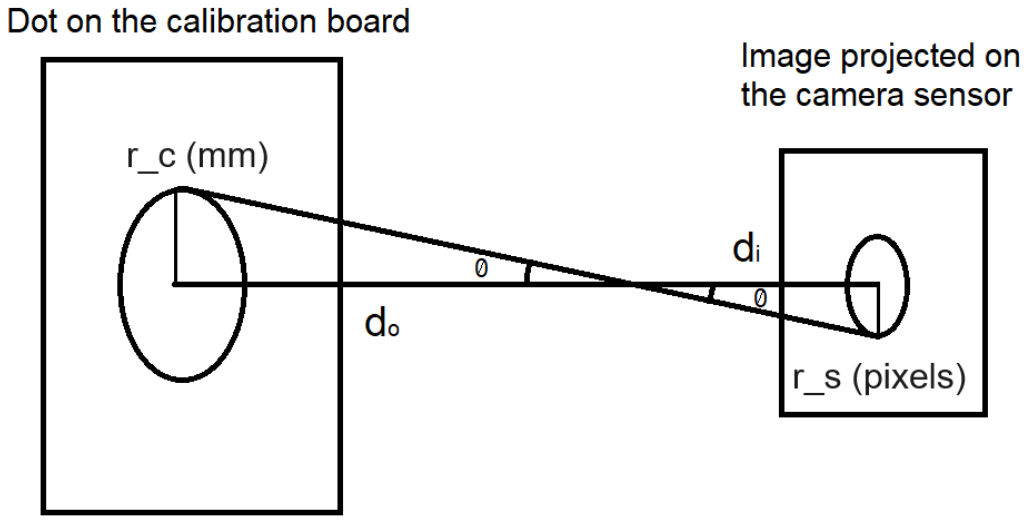

Figure 4.21: Illustration of the geometry involved in computing the vertical displacement of the camera.

It is possible to notice the two triangles formed by the radius of the dot on the calibration board, the distance between the center of the dot and the segment formed by the center of the camera lens and the intersection between the radius of the dot on the calibration board and the outer border of the dot itself. Since both of these triangles are sharing two angles: the 90° one and θ they are similar triangles. This allows us to write the equation:

$$\frac{d_o}{r\_c} = \frac{d_i}{r\_s} \tag{4.16}$$

Where d_o is the distance between the camera lens and the object, d_i is the distance between the camera lens and the sensor and r_c is the radius of the dot on the calibration board expressed in millimeters. Of particular interest is r_s which represents the radius of the dot projected on the camera sensor. To relate the two radiuses in the formula they need to be expressed in the same unit of measurement. Right now we have available the radius of the dot on the calibration board in millimeters and the radius of the dot in the image in pixels. In order to express the radius of the dot in the image in millimeters too we need the dimension of a single pixel on the camera sensor expressed in millimeters. Luckily the datasheet of the camera includes the physical dimensions of the sensors which are 5.76 x 4.29 mm. By knowing the image dimensions in pixels it is possible to compute the dimensions of a single pixel:

$$p\_h = \frac{s\_h}{i\_h} = \frac{4.29\ mm}{1944\ px} = 0,0022\ \frac{mm}{px} \tag{4.17}$$

$$p\_w = \frac{s\_w}{i\_w} = \frac{5.76\ mm}{2592\ px} = 0,0022\ \frac{mm}{px} \tag{4.18}$$

Where p_h is the height of a single physical pixel in mm, p_w is the width of a single physical pixel in mm, s_h is the height of the sensor in millimeters, s_w is the width of the sensor in millimeters, i_h is the height of the image captured by the camera in pixels and i_w is the width of the image captured by the camera in pixels. It is possible to notice how the single pixels on the sensors are squares as most common in modern cameras, this will help to simplify the following calculations.

Thanks to this new information it is possible to rewrite r_s as:

$$r\_s = r\_s\_pix * pix\_d \tag{4.19}$$

Where pix_d is one of the dimensions of a physical pixel just computed,

r_s_pix is the radius of the calibration dot on the image expressed in pixels and r_s is the radius of the calibration dot on the image expressed in millimeters.

The only unknown left from Equation 4.16 is d_i, which is a constant and it is possible to extract it from Equation 4.15, the thin lens equation. After some computations we can write:

$$d_i = \frac{f * d_o}{d_o - f} \tag{4.20}$$

Equation 4.20 is the rewritten thin lens equation where d_i is the distance between the camera lens and the sensor, d_o is the distance between the camera lens and the object and f is the focal length of the camera.

By putting together everything seen so far we obtain:

$$d_o = (\frac{r\_c}{r\_s\_pix * pix\_d} + 1) * f \tag{4.21}$$

Where d_o is the distance between the camera lens and the object which is the value we were looking for, r_s_pix is the radius of the calibration dot on the image expressed in pixels pix_d is the physical dimension of a single pixel on the sensor expressed in millimeters, f is the focal length of the camera and r_c is the radius of the dot on the calibration board expressed in millimeters. By substituting all the constants we obtain:

$$d_o = (\frac{2\ mm}{r\_s\_pix * 0,0022\ \frac{mm}{px}} + 1) * 12\ mm \tag{4.22}$$

Of course using the thin lens equation for a real camera is an oversimplification, as it is possible to notice from Figure 4.22 [54] the lens of a camera is actually composed by multiple lenses and lens groups to compensate for distortion and to provide a better final picture. This means that the measure we are going to

find won't be accurate but there will be a constant offset between the real height of the camera and the estimated one. In this application this won't be a problem: since the robot is supposed to move the camera over each dot on the calibration board keeping the height of the camera constant, without loss of generality, we can assume that when the robot is placed at the starting point with the camera over the first dot on the calibration board, it is also placed at the "correct" height. So when the robot will move the camera over the calibration board instead of computing the height of the camera as an absolute value we can just compute the vertical displacement with respect with the first point as can be seen from Equation 4.23.

$$r\_d\_z = d'_o - d_o \tag{4.23}$$

Where r_d_z is the estimated vertical displacement of the camera, d'_o is the estimated height of the camera over the first calibration point and d_o is the estimated height of the camera over the current calibration point. It is possible to notice how by computing the vertical displacement using this formula the estimated vertical displacement of the first point will always be 0. In the following sections we will address how this can become a problem when switching from the data gathering step to actually applying the corrections in an actual example where the initial point is in a different position.

Some additional complications need to be addressed when computing the vertical displacement, first of all the computations seen above are very reliant on having a precise matching of the dot on the calibration board and a precise measurement of its radius. This may become tricky in some situations, such as if there are some printing artifacts on the calibration board that prevent the dots to be all perfectly equal. If the circle is not exactly in the center of the image some distortion effects may come into play. Or if, due to the vertical displacement, the image is out of focus then the detection algorithm will have a hard time measuring the radius of the dots accurately. If the image is only slightly out of focus then having the calibration board illuminated not uniformly may lead to see some dots slightly bigger than the others.

The last complication was solved by mounting a ring light around the camera

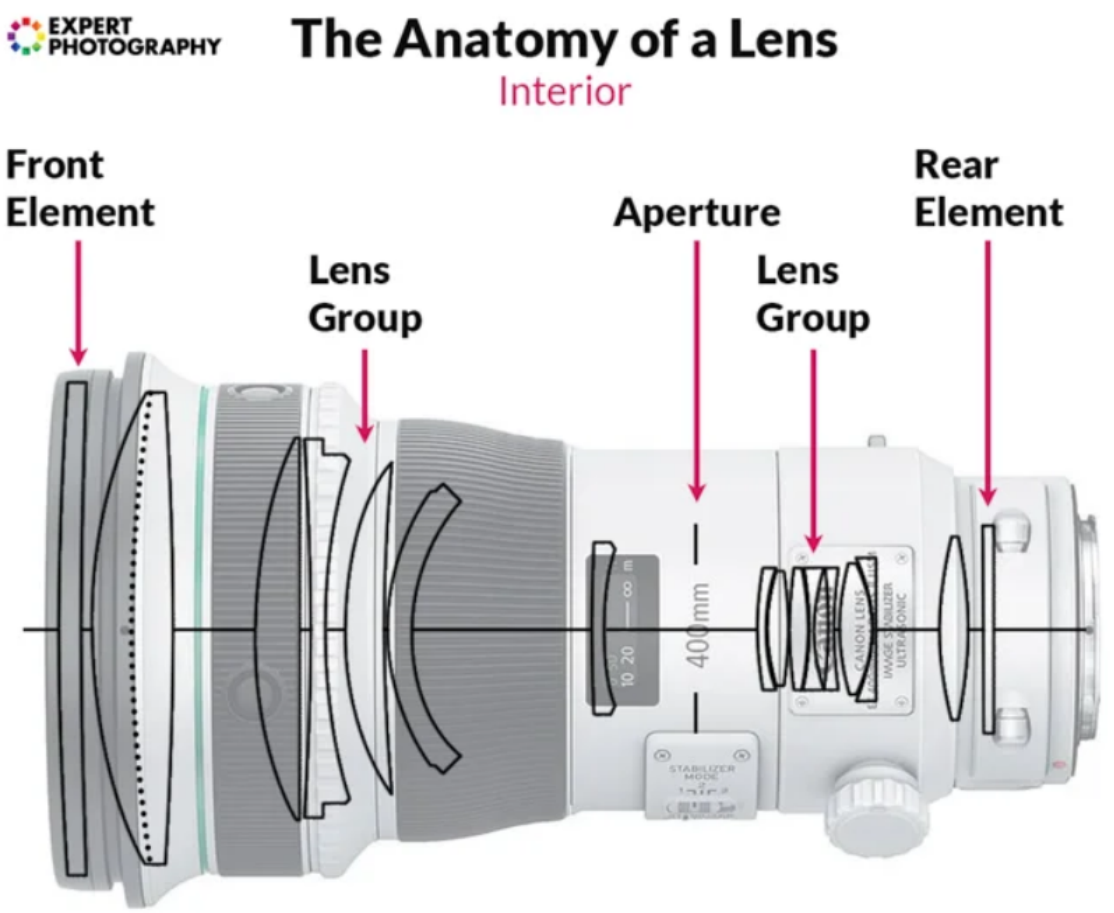

Figure 4.22: Illustration of a section of a real camera. It is possible to notice how the lens is actually composed by a series of lenses. Sourced from *Expert Photography* [54].

to illuminate the area captured in the images always in the same way but in the next section some countermeasures will be considered for all the other possible complications too. The main idea is to not just measure the error on each position but to actually correct the robot pose, this way we will instantly know if the error was computed correctly and eventually correct the course.

Once both the vertical and horizontal displacements are available the total displacement of the robot can be computed using Pythagoras's theorem a seen in Equation 4.24:

$$d\_tot = \sqrt{d\_x^2 + d\_y^2 + d\_z^2} \tag{4.24}$$

Where d_tot represents the total displacement of the robot, d_x represent the robot displacement along its x axis, d_y represent the robot displacement along its y axis and d_z represent the robot displacement along its z axis. The total robot displacement so computed only contains informations about the magnitude of the robot displacement is that particular point but it is also important to store the singular displacements ans they add informations about the direction of the displacement which is essential to correct it.

4.3.4 Repositioning

In the previous sections it was mentioned how some important assumptions were made to compute the horizontal and vertical displacements of the robot. First of all about the quality of the image captured by the camera, both in terms of resolution and absence of distortions. But also about the accuracy of the estimation of the displacements. As it was discussed beforehand the conversion between pixels and millimeters in based entirely on knowing the dimension of the dot on the calibration board in the image in pixels so we are heavily relying on having a match that outlines perfectly the dot. All of this means that it's not simply possible to correct the robot position by just reverting the estimated displacement. Even if the displacement was estimated with 100% accuracy we cannot assume that the error of the robot is constant in its workspace. So by modifying the requested coordinates beforehand by the know displacement that will be obtained by moving the robot to the desired coordinates a different displacement will be generated and the robot will not move to the correct pose. This concept will be made more clear after illustrating the iterative correction procedure applied in this thesis, but the key concept is that it is not sufficient to know the displacement in advance to prevent it we actually need to know the coordinates to provide to the robot to reach the desired point.

Figure 4.23 illustrates the iterative procedure used to correct the robot pose. After the first movement where the robot moves to the expected position the

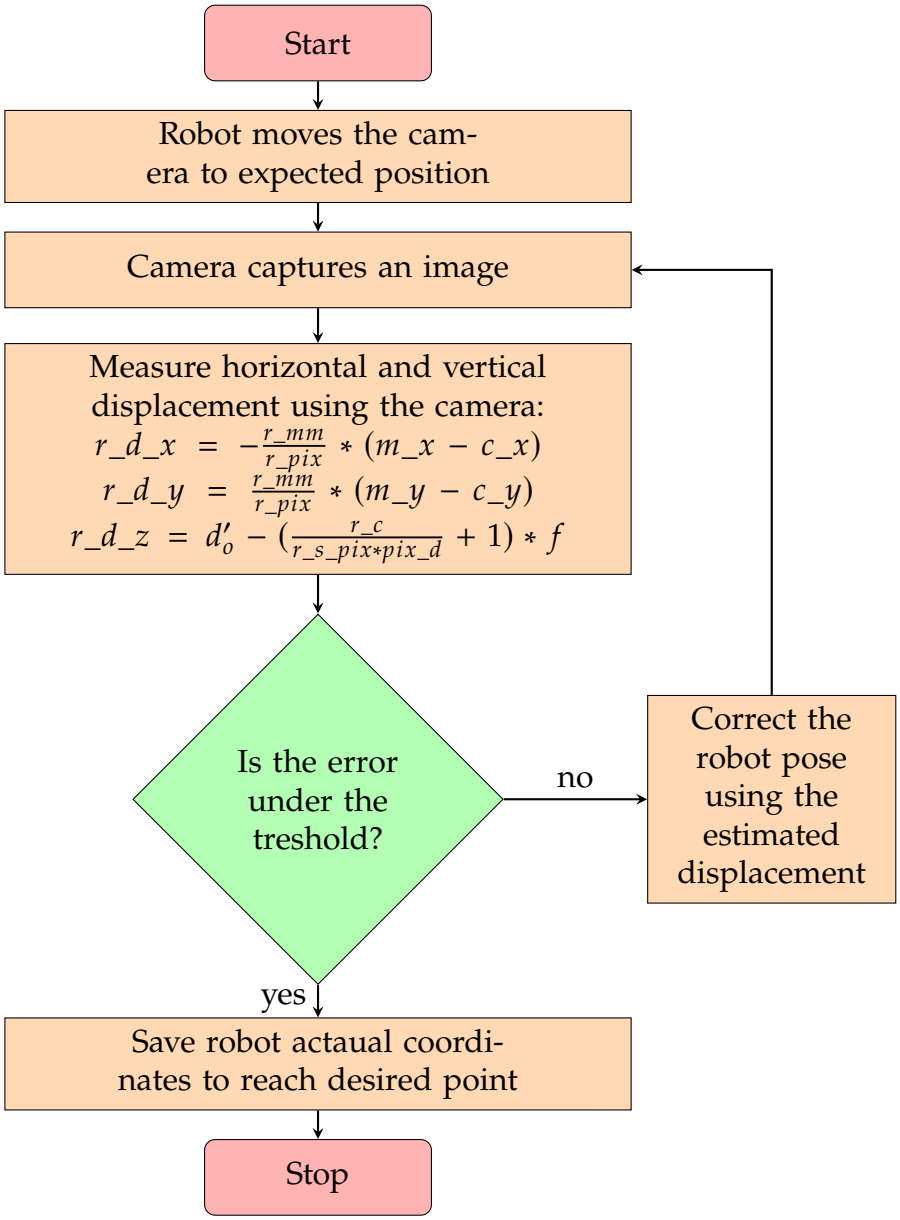

Figure 4.23: Iterative procedure to correct the robot pose.

camera captures an image and the horizontal and vertical displacements are computed. As mentioned before we cannot assume that the displacement is constant through the robot workspace so just moving in the opposite direction of the computed displacement is not enough to correct it. But we can assume that the robot displacement are continuous functions so by moving the camera in the opposite direction of the displacements the discrepancy between the actual robot pose and expected one will decrease. This process can be repeated until

the total robot displacement is under a selected threshold. For the purposes of this thesis the threshold was set at 0,02 mm which is under the expected robot repeatability of 0,05 mm. During the experiments it emerged how the robot was almost never able to satisfy the threshold on the first attempt but almost always on the second. This fact underlines the importance of not limiting the data gathering process on just estimating the displacements but to actually correct the robot pose. This allows us to have available the actual robot coordinates that we need to provide to the robot to reach the desired pose for each dot on the calibration board.

Before seeing how to generalize this informations to be able to correct the robot pose in any point in its workspace it is necessary to establish a baseline. Juts measuring the displacements of the robot in reaching each dot on the calibration board is meaningless if we don't have an accurate estimation of the robot repeatability to to compare them against. This is why in the next section we will discuss the procedure used to recompute the robot repeatability

4.4 Determining the repeatability

An essential step when trying to improve any process is to provide a baseline. This allows to compare the obtained improvements in an objective way and to draw meaningful conclusions. In this thesis the baseline to improve upon is represented by the displacements computed in the data gathering step and the objective of this thesis is to decrease them as much as possible. It wont be possible to reduce them to zero however because even when returning to a previously shown point the robot will still commit an error. This is referred to as the robot repeatability and in this thesis represents the lower bound that we cannot improve further upon.

4.4.1 Robot repeatability

Before going over the procedure applied to investigate the robot repeatability it is important to define it in more details. Robot repeatability is a critical performance metric in robotics, referring to the ability of a robot to consistently return to a specified position or follow a predetermined path with high precision. It is a measure of the robot's ability to reproduce the same motion or action multiple times under similar conditions, and it plays a fundamental role in various industrial and research applications [55]. Repeatability is usually expressed as a maximum deviation value from the desired position. In manufacturing processes, robots frequently perform repetitive tasks and high repeatability ensures that the robot executes these tasks with reliability. When a robot consistently performs a task with a high level of precision, it becomes easier to control the process and achieve consistent results, ultimately enhancing overall productivity.

Several factors influence the repeatability of a robot, including [56]:

- mechanical design: the mechanical structure of a robot plays a crucial role in its repeatability. The robot's structural rigidity, material quality, and manufacturing tolerances determine how well it can resist deformations and vibrations during operation. A stiffer and more precise mechanical design leads to reduced flexing and bending, resulting in improved accuracy and repeatability;

- control system: the control system governs the robot's movements and responses to commands. Advanced control algorithms can compensate for disturbances and uncertainties, enabling the robot to achieve more accurate and repeatable motion. Properly tuned control loops enhance the robot's ability to reach desired positions with high precision;

- environmental conditions: variations in environmental conditions, such as temperature and humidity, can affect the robot's components and lead to changes in its behavior. To maintain consistent repeatability, it is essential

to control and monitor these environmental factors during robot operation;

- quality of sensors: sensors play a crucial role in providing feedback to the robot's control system. Encoders, for instance, measure the joint positions accurately. High-quality sensors with better resolution and accuracy contribute to improved repeatability by providing more reliable feedback to the control system.

To achieve high repeatability, all these factors must be carefully considered and optimized during the robot's design, manufacturing, and operation. A well-engineered robot with a stiff and precise mechanical design, advanced control algorithms, accurate calibration, reliable sensors, and proper environmental control is more likely to exhibit higher repeatability, making it suitable for applications where precision and consistency are essential.

4.4.2 Repeatability estimation procedure

The procedure to estimate the robot repeatability is divided into two phases. The first phase consist in moving the robot over some known points and memorizing the robot coordinates associated with that robot pose. Once this is done it is sufficient to go back to these poses and see if the robot is still centered on the same reference point and eventually measure the displacement. In the previous sections the procedures applied to align the camera to a point on the calibration board and to measure the displacement were described in detail, so now it is possible to combine them to estimate the robot repeatability.

The flowchart in Figure 4.24 illustrates the combined procedures to obtain the robot repeatability. The first phase is the same as the data gathering step used to collect the displacements of the robot to reach each point. So the robot through a series of relative movements will try to position the camera over each dot on the calibration board and using the informations provided by the camera it can center itself over it. The coordinates of each correct position are memorized by the robot. In the second phase then the robot will attempt to return to the memorized positions. The displacements measured during the second phase

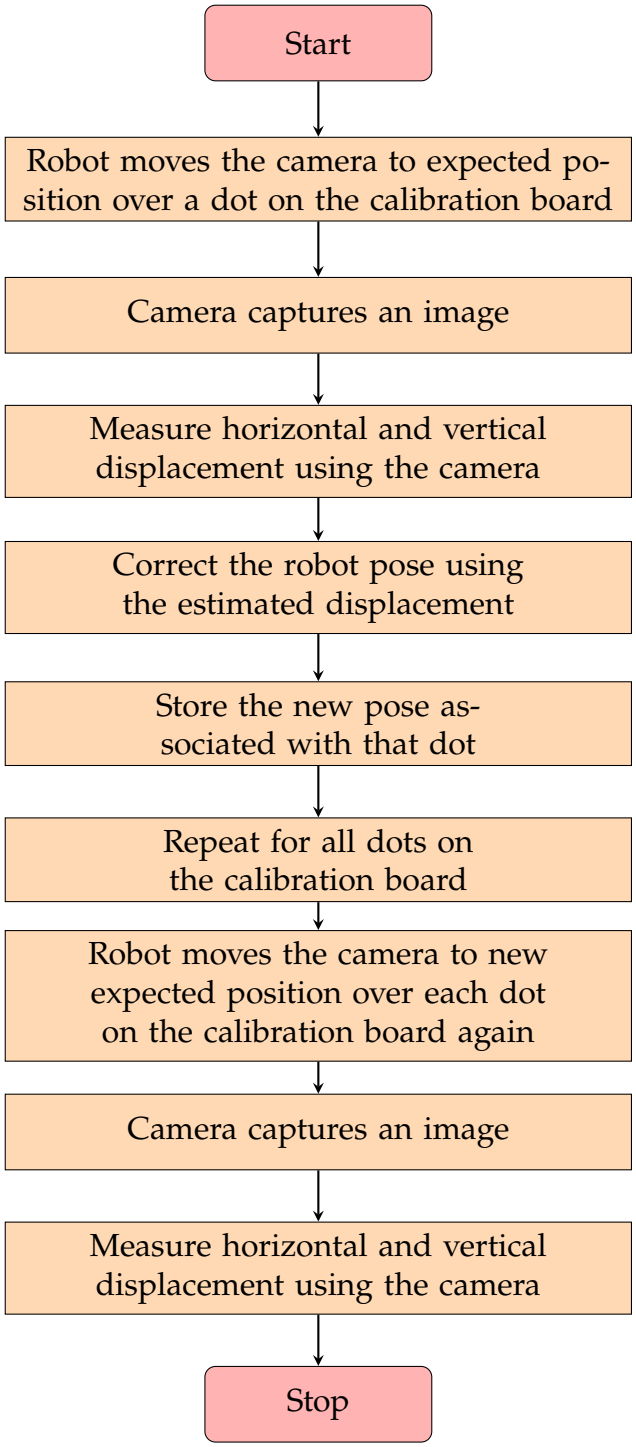

Figure 4.24: Procedure to estimate robot repeatability trough its workspace.

are not due to robot miscalibration because the robot is returning to points already memorized so by averaging the obtained displacements we can estimate

the robot repeatability. As mentioned previously this step is essential in the calibration process, it gives us a starting point thought the initial displacements and a final theoretical goal through the robot repeatability and most important it assures that once the robot accuracy gets close to its repeatability no resources will be wasted trying to improve it further.

4.5 Applying the correction

Now that the data about the robot displacements has been gathered it is time to generalize it by finding a correction function capable of mapping between the robot coordinates before and after the correction. There are many interpolation techniques but the ones we will focus on in this thesis are bilinear interpolation and neural networks used as *"universal function approximators"*.

One thing worth mentioning now is how the reference frames were managed between the data gathering step and the application of the correction. For the test to be fair we cannot assume that the target points to measure the displacements are in the same position between train and test set, otherwise interpolation would be useless as the whole process would be equivalent to the repeatability estimation procedure. To ensure that the data of the validation and test sets is meaningful the calibration board is translated and rotated after the training data has been gathered. This ensures that no two points will coincide thus creating a fresh dataset. Having a new calibration board with a completely random pattern of points would be a better solution but having only one calibration board available this is a valid compromise.

To easily shift the robot from point to point with a relative movement a reference frame is created on the calibration board with its axis aligned with the calibration pattern. This is useful to move the robot without performing additional computations but it means that the data from the train and test set live on two different reference frames. To feed the data to the correction function we first need to convert the desired point in the training set frame then the actual point returned by the correction function needs to be converted to the test frame again. DobotStudio Pro allows to easily create new reference frames for the Mg400 by using only two points which are going to specify the origin and direction of the x axis of the reference frame. Unfortunately there is no function already implemented in DobotStudio Pro to use the rototranslation matrix associated with a reference frame to convert points between different reference frames. Furthermore there is no matrix product already implemented in the Lua language, this is solved by the function in Figure 4.25 that shows a code snippet implemented to be able to treat Lua tables as matrices. This will be useful later too when the trained neural network model will have to be imported

in DobotStudio Pro.

```
function dot(m1,m2)
  if n_cols(m1) ~= n_rows(m2) then
    print("Matrix mismatch ".. n_cols(m1).." != "..n_rows(m2))
    return -1
  end
  result = {}
  for i=1,n_rows(m1),1 do
    result[i] = {}
    for j=1,n_cols(m2),1 do
        result[i][j] = 0
    end
  end
  for i=1,n_rows(m1),1 do
    for j=1,n_cols(m2),1 do
      for k=1,n_cols(m1),1 do
        result[i][j] = result[i][j] + m1[i][k]*m2[k][j]
      end
    end
  end
  return result
end
```

Figure 4.25: Code snippet from DobotStudio Pro showing how to execute matrix multiplication in Lua using tables.

Since the Mg400 can only translate its end effector and rotate it along the z axis the rototranslation matrices to jump from one frame to another are going to be composed by a translation and an elementary rotation along the z axis:

$$T_f^0 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & f_x^0 \\ \sin\theta & \cos\theta & 0 & f_y^0 \\ 0 & 0 & 1 & f_z^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.25}$$

T_f^0 is the rototranslation matrix that allows us to write a point from a frame f to frame 0. To write it we need to know the coordinates of the origin of the reference frame f with respect to the reference frame 0 and the rotation θ between the two. Both of those information are provided by DobotStudio Pro.

This matrix will be used to covert both the desired point from the test reference frame to the reference frame 0 and the correction from the train reference frame to the reference frame 0, but we will also need to convert the desired point from the reference frame 0 to the train reference frame and the correction from the reference frame 0 to the test reference frame. To do so we will also need the rototranslation matrix T_0^f which can be computed by knowing the coordinates of the reference frame 0 with respect to the reference frame f and the angle needed to align the reference frame f to the reference frame 0. Since these informations are not easily available to us in DobotStudio Pro we can resort to invert the T_f^0 matrix instead. To invert a rototranslation matrix T_f^0:

$$T_f^0 = \begin{bmatrix} R_f^0 & t_f^0 \\ 0 & 1 \end{bmatrix} \tag{4.26}$$

where R is the rotation matrix between f and 0 and t is the translation vector:

$$R_f^0 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} t_f^0 = \begin{bmatrix} f_x^0 \\ f_y^0 \\ f_z^0 \end{bmatrix} \tag{4.27}$$

To find the inverse of T_f^0, we need to compute the inverse of the rotation matrix R_f^0:

$$R_0^f = (R_f^0)^{-1} = (R_f^0)^T = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.28}$$

and apply the inverse translation t_0^f:

$$t_0^f = (t_f^0)^{-1} = R_0^f * (-t_f^0) = R_0^f * \begin{bmatrix} -f_x^0 \\ -f_y^0 \\ -f_z^0 \end{bmatrix} = \begin{bmatrix} -f_x^0 \cos\theta - f_y^0 \sin\theta \\ f_x^0 \sin\theta - f_y^0 \cos\theta \\ -f_z^0 \end{bmatrix} \tag{4.29}$$

Finally the inverse rototranslation matrix T_0^f is obtained by assembling R_0^f and t_0^f:

$$T_0^f = (T_f^0)^{-1} = \begin{bmatrix} (R_f^0)^{-1} & (t_f^0)^{-1} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & -f_x^0 \cos\theta - f_y^0 \sin\theta \\ -\sin\theta & \cos\theta & 0 & f_x^0 \sin\theta - f_y^0 \cos\theta \\ 0 & 0 & 1 & -f_z^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.30}$$

Having written all of that, the flow used to apply the corrections in Cartesian space using both bilinear interpolation and the neural network will look like the one illustrated in Figure 4.26. The first step is to provide the algorithm with the coordinates of the desired point written with respect to the reference frame currently in use Dp^{test}. The point is then rewritten with respect to frame 0 using the rototranslation matrix T_{test}^0: $Dp^0 = T_{test}^0 * Dp^{test}$. Once Dp^0 is available it can be converted into the train reference frame, which was the one used during the data gathering step, $Dp^{train} = T_0^{train} * Dp^0$. This may seem like an unnecessary passage since the two rototranslation matrices could be easily combined into one $T_{train}^{test} = T_0^{test} * T_{train}^0$, but since the test frame can change based on the application and the train frame remains constant it makes sense both logically and computationally to keep the two steps separate.

Dp^{train} can be used directly by a correction algorithm, either bilinear interpolation or a neural network in Cartesian space. The correction algorithm will output the actual point coordinates, with respect to the train reference frame Ap^{train}. The actual point coordinates represent the coordinates that need to be given as input to the robot to reach the desired point, in the case in which the robot was perfectly calibrated desired point and actual point would coincide. Ap^{train} is then converted back to frame 0 $Ap^0 = T_{train}^0 * Ap^{train}$ and then to the

test reference frame $Ap^{test} = T_0^{test} * Ap^0$. The robot could already reach Ap^{train} correctly but the extra steps allow for the correction algorithm to be inserted seamlessly into other robot programs without messing up further computations that were meant to be applied to the desired point.

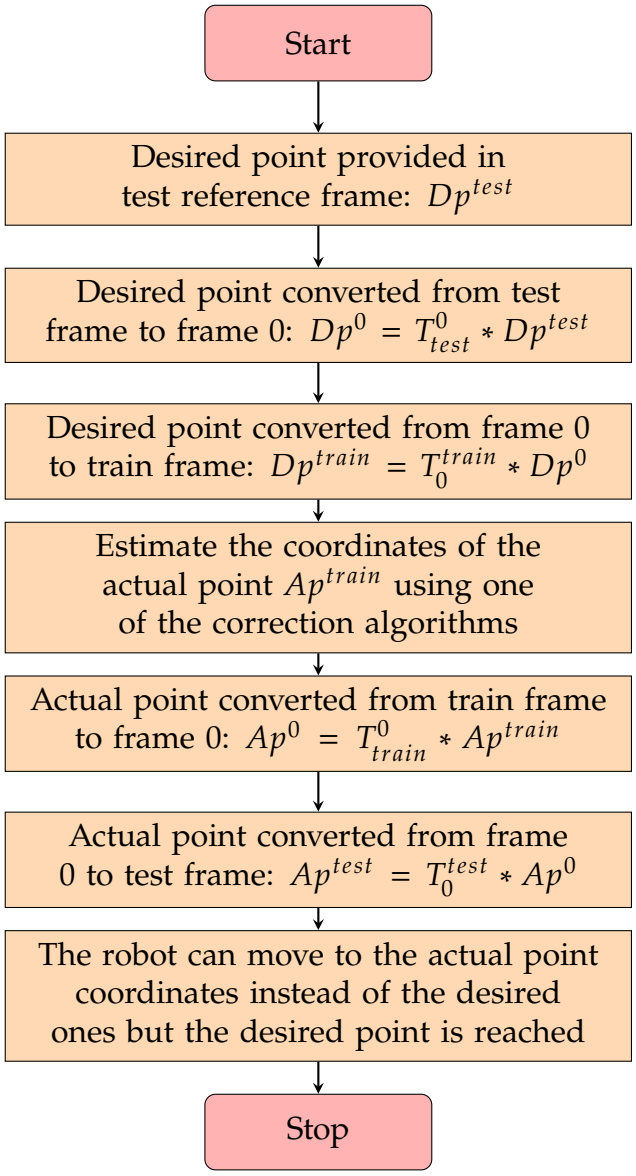

Figure 4.26: Flow to apply a correction to a point in a different frame with respect to the one used in the data gathering step.

The following sections will explore more in depth the specific correction algorithms considered in this thesis and their specific implementation, starting

with bilinear interpolation since its the simplest and most deterministic and moving into neural networks, first in Cartesian space and then in joint space.

4.5.1 Correction with bilinear interpolation

Bilinear interpolation is the simplest interpolation technique presented in this thesis, but at the same time it provide the important advantages of being completely deterministic and not requiring any training unlike neural networks. Bilinear interpolation is a method for interpolating functions of two variables by using linear interpolation repentantly. Bilinear interpolation is performed using linear interpolation first in one direction, and then again in another direction. In the chapters sections the general idea for bilinear interpolation was already presented briefly but it will be discussed now in more detail.

Consider a regular grid consisting of data points arranged in rows and columns. Each data point is associated with a known value. Bilinear interpolation aims to compute an estimated value at a non-grid position, typically within the region bounded by four neighboring grid points. This is achieved by considering the weighted average of the known values of these four points, taking into account their relative distances from the target position.

In the case taken into account by this thesis the grid is represented by the calibration points, identified by their x, y desired coordinates and the known values associated with each of them are their actual coordinates x, y and z. Any desired point that belongs to the area covered by the calibration board will fall in a square belonging to the original calibration board that was set up during the data gathering step as illustarted by Figure 4.27 [57].

To identify the square on the original grid where the desired point would belong it is sufficient to either scroll through all of them which is inefficient but effective or, since the train reference frame x and y axis are parallel to the calibration grid it is also possible to identify in advance the the row where the point belongs (or column) and then only take into account only the cells in the corresponding row (or column). This can help to avoid computational delays in cases in which the calibration grid becomes a lot bigger. Once the original square

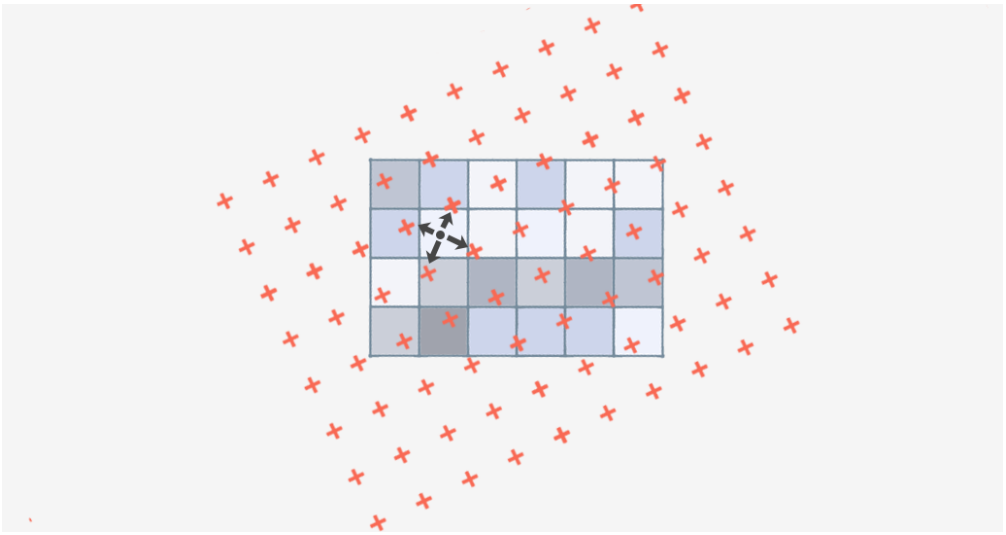

Figure 4.27: illustration of the overlap between the calibration grid used during the data gathering step and the test step. Sourced from *GisGeography* [57].

has been identified it is possible to recover from the data stored in memory the four original points composing its corners as seen in Figure 4.28 [58]:

- Q_{11} at coordinates (xd_1,yd_1) with associated values xa_{11}, ya_{11} and za_{11};
- Q_{12} at coordinates (xd_1,yd_2) with associated values xa_{12}, ya_{12} and za_{12};
- Q_{21} at coordinates (xd_2,yd_1) with associated values xa_{21}, ya_{21} and za_{21};
- Q_{22} at coordinates (xd_2,yd_2) with associated values xa_{22}, ya_{22} and za_{22}.

These points are used to create a weighted average that estimates the value at the target position. Given v and u as:

$$u = \frac{xd - xd_1}{xd_2 - xd_1}, v = \frac{yd - yd_1}{yd_2 - yd_1} \tag{4.31}$$

Where u represents the normalized horizontal distance from Q_{11} to the target position (xd,yd) and v represents the normalized vertical distance from Q_{11} to the target position (xd,yd). u varies from 0 to 1 as xd goes from xd_1 to xd_2 and u varies from 0 to 1 as yd goes from yd_1 to yd_2.

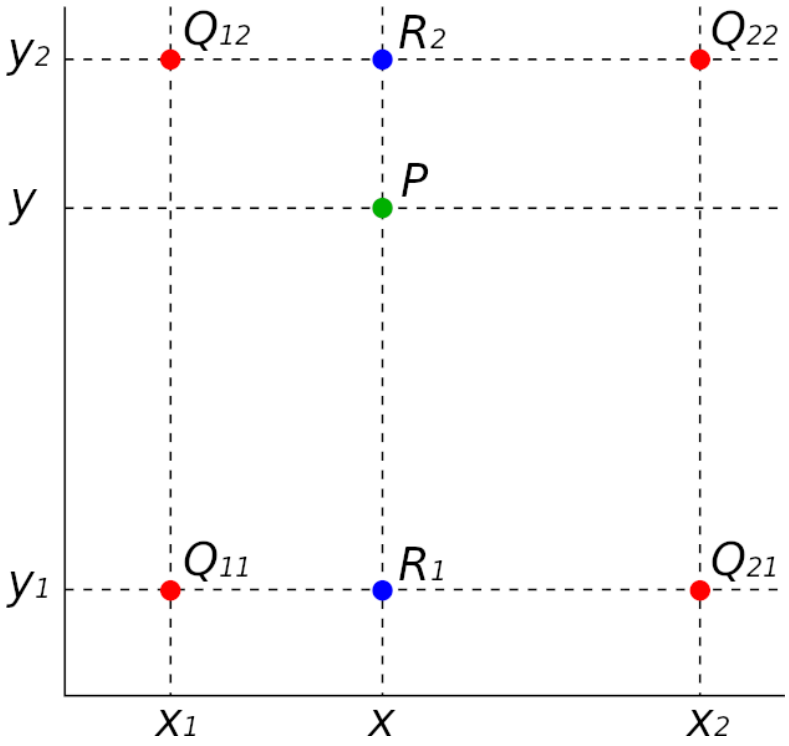

Figure 4.28: 2D illustration of the bilinear interpolation principle. Sourced from "Model risk and techniques for controlling market parameters. The experience in Banco Popolare" [58].

To compute the interpolation we can write:

$$Rax_1 = (1 - u) * xa_{11} + u * xa_{21} \tag{4.32}$$

Equation 4.32 computes a linear interpolation between the values xa_{11} and xa_{21} along the horizontal direction. $(1 - u) * xa_{11}$ represents the contribution of xa_{11} scaled by $1 - u$ which decreases as xd moves from xd_1 to xd_2. $u * xa_{21}$ represents the contribution of xa_{21} scaled by u which increases as xd moves from xd_1 to xd_2. Together, this part computes the weighted average of xa_{11} and xa_{21} along the horizontal line at the target vertical position.

$$Rax_2 = (1 - u) * xa_{12} + u * xa_{22} \tag{4.33}$$

Equation 4.33 computes a linear interpolation between the values xa_{12} and xa_{22} along the horizontal direction. $(1-u)*xa_{11}$ represents the contribution of xa_{11} scaled by $1-u$ which decreases as xd moves from xd_1 to xd_2. $u*xa_{21}$ represents the contribution of xa_{21} scaled by u which increases as xd moves from xd_1 to xd_2. Together, this part computes the weighted average of xa_{11} and xa_{21} along the horizontal line at the target vertical position.

The two equations can be put together to compute the final value

$$Pax = (1-v)*Rax_1 + v*Rax_2 \tag{4.34}$$

Equation 4.34 computes a linear interpolation between the values Rax_1 and Rax_2 along the vertical direction. $(1-v)*Rax_1$ represents the contribution of Rax_1 scaled by $1-v$ which decreases as yd moves from yd_1 to yd_2. $v*Rax_2$ represents the contribution of Rax_2 scaled by v which increases as yd moves from yd_1 to yd_2. Together, this part computes the weighted average of Rax_1 and Rax_2 along the vertical line at the target horizontal position.

It is possible to notice how the points stored in memory are only identified by their original x and y desired coordinates. This is because since the camera is moving on a plane adding a value equal for all the points wouldn't add any information, also bilinear interpolation only accepts two values as input. In the case in which the plane where the camera movements belong to is the same between train and test then everything is fine. If instead the distance between the camera and the calibration grid changes between train and test then a different solution would have to be considered, maybe mapping the displacements at different heights and then performing cubic interpolation including the height information too [59]. This can be done but it would add to the complexity of the algorithm so this case is going to be contemplated only for neural networks.

To compensate for small variation in height between train and test a simple countermeasure is put into place: the vertical correction that would be applied to the first point in the test set isn't applied and instead it is saved as an offset to compensate on all the other points too.

One last consideration about bilinear interpolation is that its corrections limited for points that fall inside the original calibration grid used during the data gathering step. For points that fall just outside it is possible to extend the correction by linearity but its accuracy will diminish quickly as the desired points moves further away from the mapped area.

4.5.2 Correction with neural network in Cartesian space

Neural networks are a class of machine learning algorithms inspired by the structure and functioning of the brain. These models are composed of interconnected computational units, referred to as neurons or nodes, organized into layers that process and transform input data to produce desired output predictions or classifications. In the case considered in this thesis, more specifically they are used as function approximator, aiming to map input data to output predictions. The network's operation involves two fundamental components: the weighted sum of inputs, often called activations, and an activation function that introduces non-linearity. Neurons are organized into layers, with input data provided to the input layer, intermediate computations performed in hidden layers, and final predictions obtained from the output layer. The number of hidden layers and the number of neurons in each layer, known as the network's architecture, significantly impact its capacity to learn and generalize. The learning process in neural networks revolves around adjusting the weights and biases to minimize a predefined loss function when making predictions on sample data. This optimization is typically achieved using the backpropagation algorithm coupled with gradient descent optimization methods. Backpropagation involves computing the gradient of the loss with respect to the model parameters and updating the parameters in the opposite direction of the gradient to minimize the loss [60].

Figure 4.29 shows a simplified view of the architecture of the neural network used to make predictions in Cartesian space. Since the available dataset have a limited dimension it wouldn't make sense to use a big network or not enough samples would be available to train it properly.

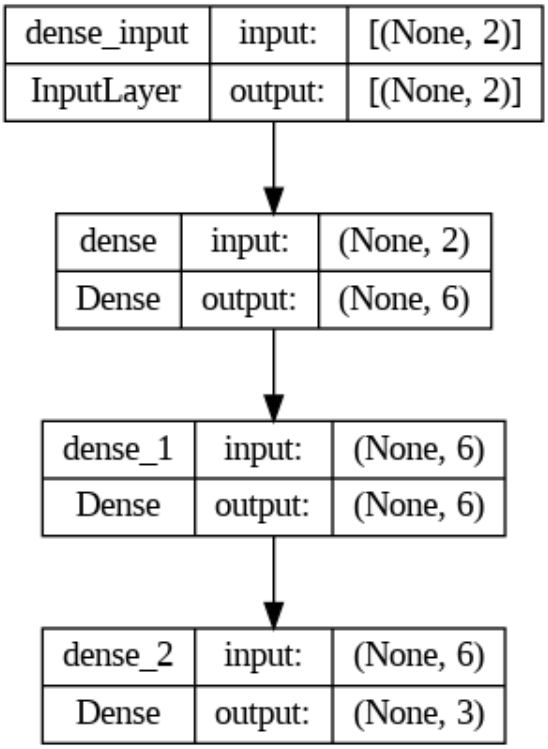

Figure 4.29: Illustration of the structure of the neural network used to make predictions in Cartesian space.

To avoid complications with different reference frames a validation set is generated by randomly sampling some points from the train set, instead of moving the grid and perform another data gathering process. This allows us to perform a grid search looking for the best architecture and hyper-parameters for the neural network. Different architectures were put to test and the results showed very similar results for networks with one or two hidden layers with at least four neurons each. The activation function that performed better in the hidden layers is the hyperbolic tangent while the output neurons use the linear function to not distort the results. Since the camera moves on a plane parallel to the calibration board the desired z coordinate is always constant so the only relevant input data are the x and y coordinates of the desired point that we want to reach, this is why the dimension of the input layer is two. On the other hand the correction has to be applied along all three axis so the dimension of the output layer is three. The two hidden layers are composed of six neurons each since this was the architecture that performed better when put to test on the validation set. Changing the batch size does not seem to have an influence

on the result but a small learning rate ($1e^{-5}$) allows the model to keep slowly improving after the first few epochs reaching an overall better accuracy on the validation set as seen in Figure 4.30.

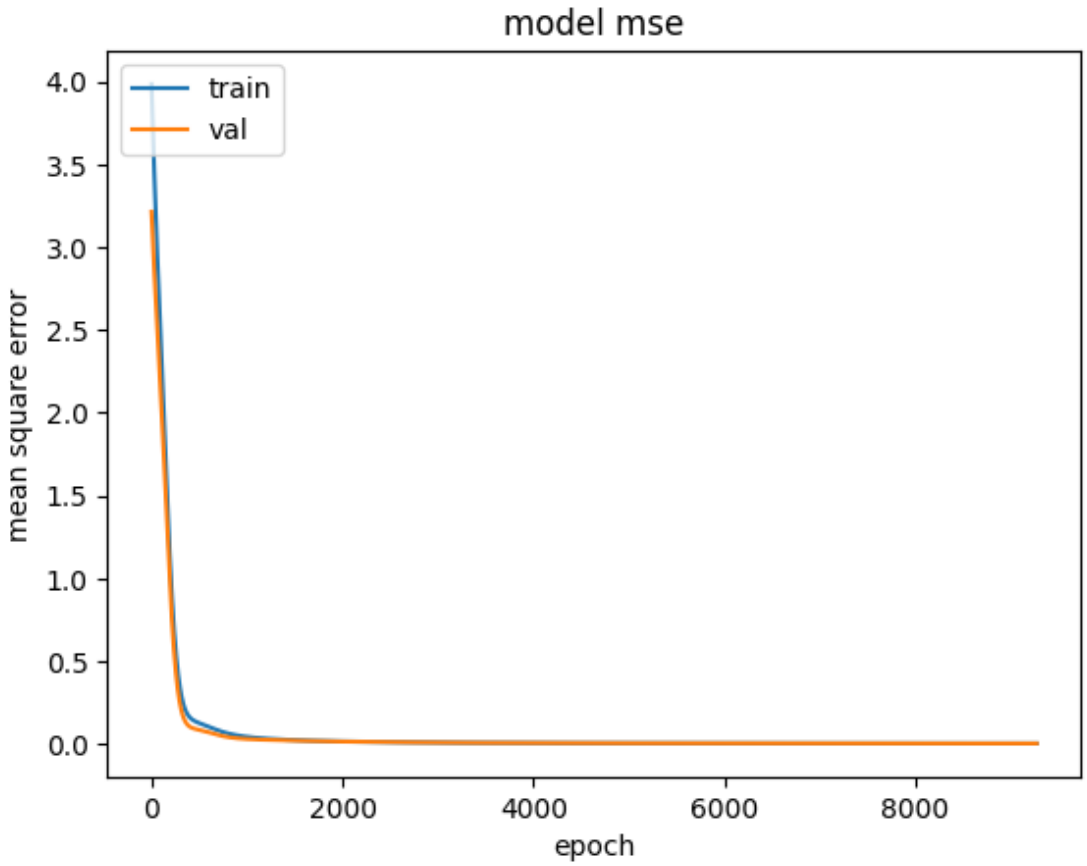

Figure 4.30: Plot of the training and validation loss of the neural network during training, it is possible to notice how after the first few epochs of fast improvement the model keeps learning slowly until it eventually stops.

Since the mapped workspace is relatively small the outputs of the neural network are all very close in value this creates a problem during training: the network easily falls into the local minima of always giving as output the average of the training set instead of actually learning. To avoid this problem the datasets are reformatted in such a way that the target values to predict are not the actual point coordinates but the difference between the actual point coordinates and the desired point coordinates. This allows to easily reconstruct the target value after the prediction.

Differently from the bilinear interpolation no special extension is needed for the neural network model to generate prediction for points that fall outside of the original calibration board used during the data gathering step, but just like for the bilinear interpolation the accuracy of the predictions will quickly deteriorate as soon as the requested point moves further away from the original calibration board the network was trained on.

The complete set of steps to predict a correction using the neural network in Cartesian space, without taking into account the frames conversions already discusses, are:

1. normalize the input: the training set was normalized so any new input needs to also be normalized according to the same normalization function;
2. apply the feedforward algorithm of the neural network to get the predicted difference between actual and desired coordinates;
3. add the predicted difference to the original desired coordinates to obtain the predicted desired coordinates.

One last important point to mention in how the neural network model was imported into DobotStudio Pro in the Lua language. Google Colab and Python offer a series of libraries that greatly simplify the creation and training of machine learning models such as neural networks, they also offer the possibility to save a trained model and upload it somewhere else. Unfortunately Lua does not support any of this. To use the neural network model in DobotStudio Pro a workaround had to be set up. The network was first trained on Google Colab, then its weights were imported as tables in Lua. Once the network has been trained only the feedforward algorithm needs to be applied which can be reduced to a series of matrix multiplications and activation functions. Luckily the same function shown previously that allows us to treat Lua tables as matrices can be used again here.

Figure 4.31 shows some code snippets of the other functions created to use a neural network model in Lua. The first is the $tanh()$ function which is used as activation function on a single value, the second allows us to apply the activation function on a mono dimensional Lua table treating it as a vector and the third

```
function tanh(x)
  return ((math.exp(x) - math.exp(-x))/(math.exp(x) + math.exp(-x)))
end

function activation_function(h)
  result = h
  for i=1,n_rows(h)-1,1 do
    result[i][1] = tanh(h[i][1])
  end
  return result
end

function feed_forward(j1d,j2d,j3d,j4d)
  input = normalize_input(j1d,j2d,j3d,j4d)
  h1 = dot(W1,input)
  o1 = activation_function(h1)

  h2 = dot(W2,o1)
  o2 = h2
  o2 = activation_function(h2)

  o3 = dot(W3,o2)

  return j1d+o3[1][1],j2d+o3[2][1], j3d+o3[3][1], j4d+o3[4][1]
end
```

Figure 4.31: Code snippets of the functions created to use a neural network model in Lua.

shows how the feedforward algorithm can be written in a compact way thanks to the functions previously implemented. W1,W2 and W3 are Lua tables containing the weights of the trained neural network model which are outputted already with the correct formatting by the Python script and are saved separately in a different file.

4.5.3 Correction with neural network in joint space

An alternative to estimating the corrections in Cartesian space is to do it in joint space. The procedure is less intuitive but by using a neural network the added complexity is masked by the network. There are many reasons to try to correct the robot pose in joint space, the main one is that the forward and inverse kinematics which allows us to move from joint to Cartesian space and vice versa are non linear transformations so having them already included in the data presented to the network can help it guide it to find patterns in the error model that were harder to find in Cartesian space given the limited number of samples available. In addition working in joint space saves us a lot of troubles with reference frames conversions since the joint values for a point in space are the same independently from which reference frame the point is being associated to, the counterpoint of this is that when creating a point starting from Cartesian coordinates its joint coordinates are generated automatically by DobotStudio Pro but when creating a point in joint coordinates the Cartesian coordinates are not generated automatically so a forward kinematics function needs to be implemented in order for the algorithm to output the corrected point in Cartesian coordinates.

Figure 4.32 illustrates a simplified schematic of the architecture of the artificial neural network used to predict the corrections to apply in joint space. Even though the camera moves on a plane parallel to the ground the values of all four joints are constantly changing. But since the fourth joint does not contribute to the position of the flange and we are interested in the correction to apply at that specific point only the values of the first three joints are going to be used as input for the network, this is going to simplify the computations when changing tool between train and test. This explains why both the input and output layer are composed by three neurons. Additionally both during train and test the orientation of the tool with respect to the zero reference frame was kept constant so it would not make sense to include the fourth joint in the dataset to feed to the neural network. This is because in the case in which the tool would need a different orientation during test, as was the case when using the mechanical comparator in the "Results and discussion" chapter, the correction on the fourth joint would not only be useless but detrimental to the accuracy of the robot. As it wouldn't be possible to gather a dataset for each possible orientation of the

tool the fourth joint is left out, furthermore since the tool link is almost always short a small error in the fourth joint would result in a small error in the position of the end effector while a small error in the first joint would result in a big error in the position of the end effector due to its longer reach.

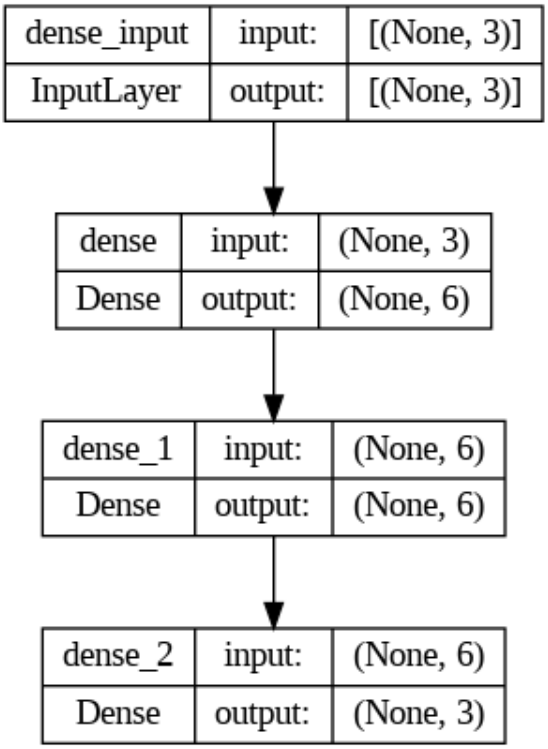

Figure 4.32: Illustration of the structure of the neural network used to make predictions in joint space.

As for its Cartesian counterpart, using a validation set, different architectures were put to test and the results showed very similar results for networks with one or two hidden layers with at least four neurons each. The activation function that performed better in the hidden layers is the hyperbolic tangent while the output neurons use the linear function to not distort the results. One additional consideration when training the neural network in joint space is that having three inputs means that the network has more parameters to tune so it would need more data for training, this can be achieved by performing the data gathering step at slightly different heights from the calibration board. Operating in joint space means that the additional data can be integrated into the training set without worrying about frames transformations.

Mg400 Direct Kinematics

As mentioned above in order to integrate the joint space based neural network into other algorithms we need to be able to output he actual point coordinates in Cartesian space and to do that it is necessary to implement a direct kinematic function for the Mg400. Luckily having only four motorized joints and only three of them contribute to the position of the flange the forward kinematics transformation can be easily computed in closed form without the need for creating Denavit Hartemberg frames.

Looking at Figure 4.33 [20] it is possible to see a schematic of the Mg400 joints and links. To compute the x and y position of the flange it is easier to compute first the length of the projection of links two and three on the ground plane:

$$L_{23} = L_2 * sin(J_2) + L_3 * cos(J_3) + L_{1x} \tag{4.35}$$

Where L_2 is the length of link two, J_2 is the value of joint two, L_3 is the length of link three, J_3 is the value of joint three and L_{1x} is the horizontal component of link one or the horizontal displacement between the axis of joint one and two. L_{1x} also includes the small horizontal displacement between the non actuated joint at the end of link three and the flange.

Once L_{23} is available it is easy to decompose in its x and y components:

$$P_x = cos(J_1) * L_{23} = cos(J_1) * (L_2 * sin(J_2) + L_3 * cos(J_3) + L_{1x}) \tag{4.36}$$

$$P_y = sin(J_1) * L_{23} = sin(J_1) * (L_2 * sin(J_2) + L_3 * cos(J_3) + L_{1x}) \tag{4.37}$$

Where P_x and P_y are the x and y Cartesian coordinates of the point. P_z can then be computed as:

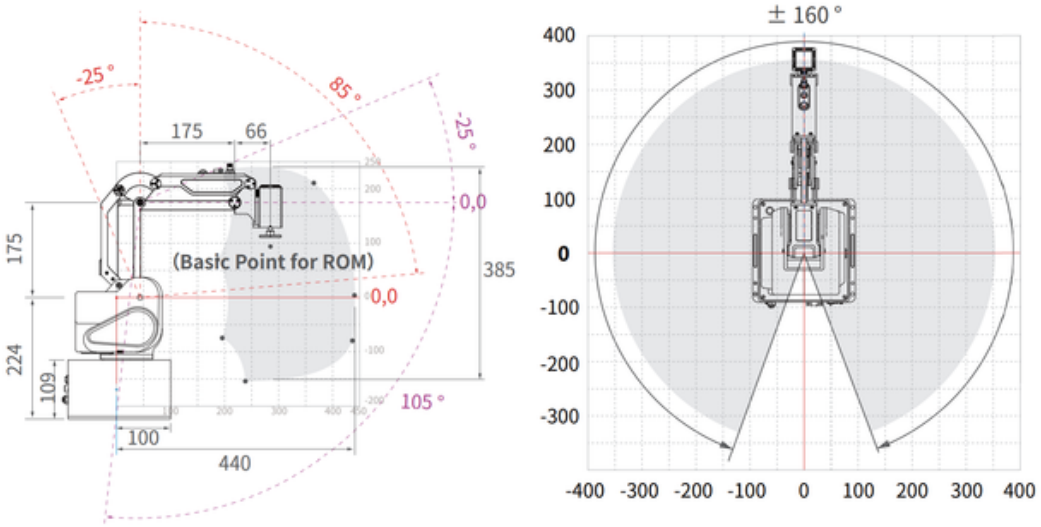

Figure 4.33: Illustration of the Mg400, useful to compute the direct kinematics transformation. Sourced from *Dobot Robotics* [20].

$$P_z = -L_{1z} + L_2 * cos(J_2) - L_3 * sin(J_3) \tag{4.38}$$

Where L_{1z} is the horizontal displacement between the axis of joint one and two. L_{1z} is the vertical component of link one or the vertical displacement between the axis of joint one and two. L_{1z} also includes the small vertical displacement between the non actuated joint at the end of link three and the flange.

The direct kinematic transformation can be easily extended in case in which the tool center point does not correspond with the flange:

$$P_x = cos(J_1)*(L_2*sin(J_2)+L_3*cos(J_3)+L_{1x})+cos(J_1+J_4)*L_{5x}-sin(J_1+J_4)*L_{5y} \tag{4.39}$$

$$P_y = sin(J_1)*(L_2*sin(J_2)+L_3*cos(J_3)+L_{1x})+sin(J_1+J_4)*L_{5x}+cos(J_1+J_4)*L_{5y} \tag{4.40}$$

$$P_z = -L_{1z} + L_2 * cos(J_2) - L_3 * sin(J_3) \tag{4.41}$$

In this case L_5 represents the link attaching the tool to the flange and it can be decomposed into its x and y components L_{5x} and L_{5y} while L_{5z} is already included in L_{1z}. Just like L_4 the link connecting the non actuated joint at the end of link three and the flange is included in L_{1x} since the two are always parallel.

```
L1_x = 109.5
L1_z = 53.0
L2 = 174.2076
L3 = 175.0707
L5_x = 59.45
L5_y = 4.905

function forward_k(j1,j2,j3,j4)
  x = cos(j1)*(sin(j2)*L2+cos(j3)*L3 + L1_x) + cos(j1+j4)*L5_x -
    sin(j1+j4)*L5_y
  y = sin(j1)*(sin(j2)*L2+cos(j3)*L3 + L1_x) + sin(j1+j4)*L5_x +
    cos(j1+j4)*L5_y
  z = -L1_z + cos(j2)*L2 - sin(j3)*L3
  r = -0.5
  return x,y,z,r
end
```

Figure 4.34: Lua function for the forward kinematics transformation.

Figure 4.34 shows the Lua function to compute the forward kinematics of the manipulator. The $sin()$ and $cos()$ functions are actually wrapper function to access the math Lua library and convert the joint values from degrees to radians. It is worth noting how the length of each link is different from their nominal length, link two should be 175 mm long but it differs for almost 0.8 mm. This is due to the robot calibration that was already performed in the factory to compensate for manufacturing defects. The actual length of the links are extracted from the robot internal files.

Figure 4.35 illustrates the complete procedure to estimate the actual point

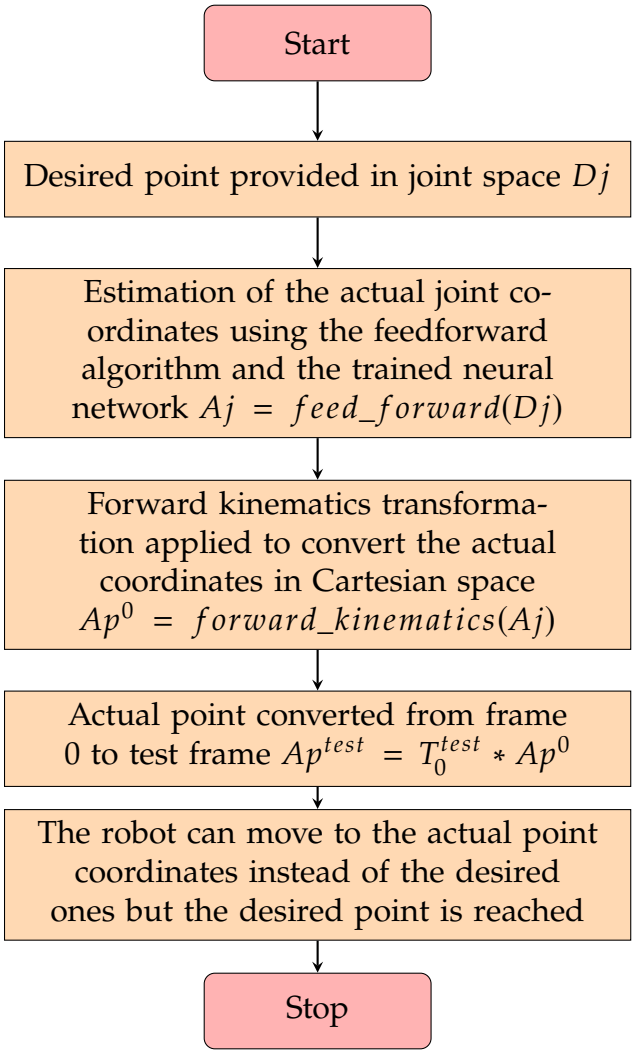

Figure 4.35: Flow to apply a correction to a point in joint space.

coordinates using the neural network in joint space. The first step is to provide a desired point in joint coordinates, this is easy since when a point is generated in DobotStudio Pro in Cartesian space the corresponding joint value are assigned to it immediately. Then the feedforward algorithm can be applied to estimate the correction in joint space. These two steps could already be sufficient if the goal was to just move the robot to the desired position but to integrate the correction algorithm seamlessly in other pipelines it is necessary to output a point in Cartesian coordinates written according to the current reference frame. This is why the following steps include applying the direct kinematics transformation to convert the actual coordinates in joint space into actual coordinates in Cartesian space. Since the forward kinematics function is set up to output a Cartesian

point according to the zero reference frame, the T_0^{test} rototranslation matrix is needed to convert the actual point from the zero reference frame to the test reference frame currently in use.

The concepts seen in this section illustrate how to gather the necessary data, train a neural network model, import it in Lua and use it to estimate predictions and correct the robot pose. But an essential detail was left out: at the beginning of the data gathering phase the robot is moved so that the camera is centered over the first dot on the calibration grid, then when switching to the test phase the calibration grid is moved in the robot workspace and the robot is moved so that the camera is centered over the first dot on the calibration grid again. In the next section this problem will be analyzed in depth together with some possible solutions to adapt the error model generated during training to perform correctly during test too.

4.6 Changing frames between train and test

As mentioned above it is not sufficient to convert a point from the reference frame used in the training step to the reference frame used in the test step to be able to successfully estimate the correction between different reference frames. At the beginning of the data gathering phase the robot is moved so that the camera is centered over the first dot on the calibration grid, then when switching to the test phase the calibration grid is moved in the robot workspace and the robot is moved again so that the camera is centered over the first dot on the calibration grid. This means that the first point both in training and test will have zero error, this is fine during training since the error model still has to be generated but during test there will be a point with zero error in an area of the workspace where the error is supposed to be different from zero. And the same happens for the points closer to the first point, their displacement is going to be lower since they are close to the point that was centered manually but if they happen to be in an area of the workspace where the displacement was big during the training step then the algorithm will try to apply an equally big correction to their coordinate with the end result being worse than having no correction

at all. This problem generates from the fact that moving the robot so that the camera is centered over the first dot on the calibration grid is already applying a correction. This creates a discrepancy between the error model generated during training and the one that corresponds to the current situation during test. In the following sections different cases in which this happens will be taken into considerations together with some possible solutions.

4.6.1 Translating frames

The first simpler case to take into consideration is when the calibration board has been translated at the end of the data gathering step but not rotated before starting the test step. This means that the origin of the reference frame associated with the calibration board has been translated, together with all the calibration dots but the axis of the reference frames associated with the calibration board are still parallel before and after the translation.

This situation is similar to the case discussed earlier when the test is performed in at a different height with respect to the training. The solution that was proposed back then was to memorize the vertical correction that the algorithm wanted to impose on the first point of the test set and use it as a negative offset for all the points, this way the first point in the calibration board will have zero vertical error in the test phase too. A similar solution can be implemented for the case of translating frames. By memorizing the first correction that the algorithm proposes for the first point on the calibration board and using it as negative offset for all the points in the test set it is possible to translate the error model and have it match the error model generated during training. This way the first point in the calibration board will have zero error in the test phase too and the points neighboring it will have a smaller error since it is reduced by the same amount that was computed for the first point.

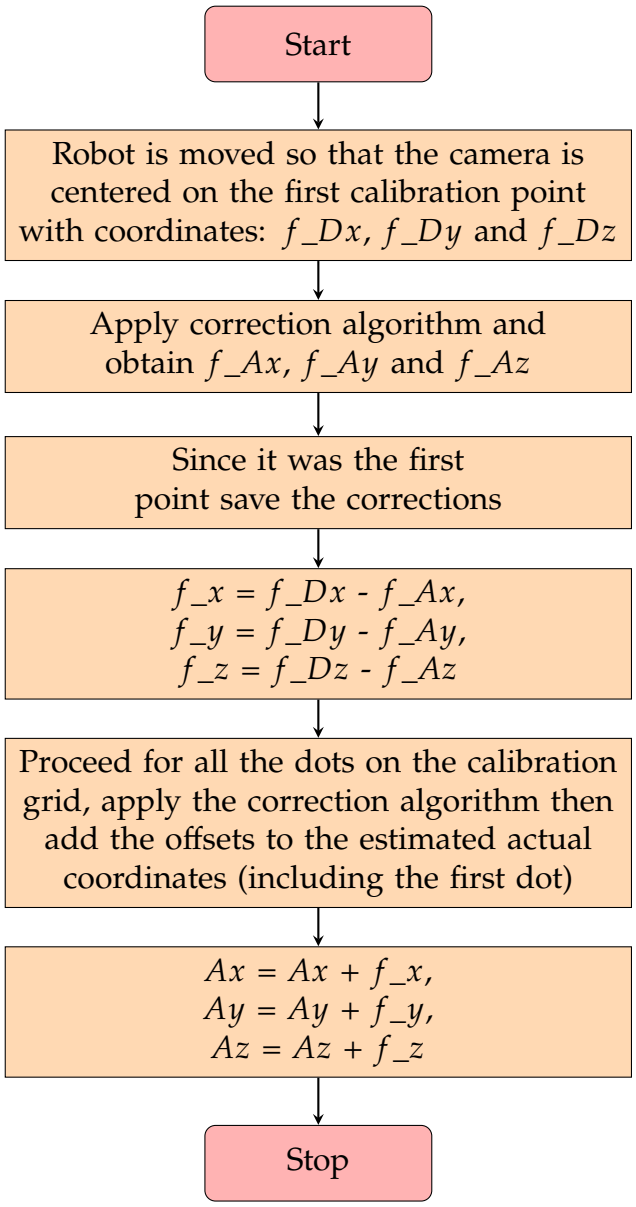

Figure 4.36: Flow to apply a counter correction when the calibration board was just translated between the data gathering step and the test phase.

Figure 4.36 illustrates the flow of the counter correction procedure when the calibration board was just translated between the data gathering step and the test phase. When the first point with desired coordinates f_Dx, f_Dy and f_Dz is being processed the correction algorithm will output some actual coordinates f_Ax, f_Ay and f_Az. Since the camera was centered manually over the first point the desired coordinates and actual coordinates of the first point should be the same, but since the correction algorithm does not know that this point was centered manually it will output a correction based just on the position of the point in the workspace. To counter correct its position the robot needs to stay still on the first point so the displacement between the correct desired coordinates and the wrongly estimated actual coordinates is saved in memory:

$$f\_x = f\_Dx - f\_Ax \tag{4.42}$$

$$f\_y = f\_Dy - f\_Ay \tag{4.43}$$

$$f\_y = f\_Dy - f\_Ay \tag{4.44}$$

Where f_x, f_y and f_z are the correction offsets for the first point on the calibration board. Then the test phase may proceed with the robot attempting to move the camera over each dot on the calibration board, but this time after applying the correction algorithm the initial offsets are being kept into account:

$$Ax = Ax + f\_x \tag{4.45}$$

$$Ay = Ay + f\_y \tag{4.46}$$

$$Ay = Ay + f\_y \tag{4.47}$$

This way for the first point on the calibration board the desired and actual coordinates will coincide and for all the other points the error model will be translated back so that the overlap with the calibration board will coincide with the error model generated during the training phase.

This case may seem like an oversimplification but it is important to keep in mind that if the calibration is performed with a specific application in mind for the robot so the calibration board can be positioned so that its reference frame can be aligned with the one used in the application by means of just a translation. This would greatly simplify the computations and save the need for more complex procedures.

4.6.2 Rototranslating frames

A more complex problem is presented when the calibration board is not just translated between the data gathering step and the test phase. In the case in which a rotation is also applied to the calibration board the error model changes drastically [61] and a simple translation is not sufficient anymore to realign it with the one generated during the training phase. Applying the same rototranslation to the error model would also not suffice since only some components of the error model are changing. The solution proposed in this thesis is to take the two calibration points that were used to create the reference frame associated with the calibration board in the test phase, with one being the origin of the reference frame and the other giving the direction of the x axis and applying the inverse of the correction on both of them before recomputing the reference frame. Applying this procedure on the origin of the frame means that when applying the correction algorithm the output point will coincide with it as we want. Furthermore applying the counter-correction procedure on the second point composing the reference frame attached to the calibration board means to realign the component of the error model that rotated back to its original position during the data gathering step.

The problem with this solution is that both the bilinear interpolation and

the neural networks are not invertible functions. But making some assumptions on the error model of the robot it is possible through an iterative procedure to find a point close to the actual point such that applying the correction algorithm on the first point the output is very close to the actual point. Assuming that the corrections to apply are small and that the error model is continuous it is possible to come close to the inverted point by computing the correction of the original point and then subtracting it to the original point. This can be done in a single jump or through multiple small jumps to achieve a better accuracy as will be shown later. The advantage of completing the jump in a single step is that it saves computational time but the outcome is less accurate, this is because even if we assume that the error model of the robot is continuous we cannot assume that it is linear. As in the data gathering step we would need multiple computations of the error to actually center the camera precisely over a dot on the calibration board, multiple jumps reduced by a decaying factor provide an overall better result in a similar fashion to the gradient descend algorithm.

Figure 4.37 [62] can give a better idea of this last concept. The blue gradient map represents the error model of the robot generated during the training step, x_0 represent the original point to counter correct and the green arrow shows the point obtained feeding x_0 to the correction algorithm. Since x_0 may correspond to the first point on the calibration board (or the other point giving the direction of the x axis of the reference frame) having it in a different position with respect to its corrected version means that the error on the first point wouldn't be zero. The objective is to find a point such that applying the correction algorithm to it we would obtain x_0 again.

Figure 4.38 can help provide a better understanding of the iterative process proposed to find said inverted point such that applying the correction algorithm the output is as close as possible to the input point. The first step, given an input point with coordinates Dx, Dy and Dz is to apply one of the correction algorithms seen before to obtain the estimated actual coordinates Ax, Ay and Az. In this case since the correction algorithms only work with the x and y coordinates as input the z coordinate is ignored. Then to find the correction offset the difference between the desired and actual coordinates is computed:

$$Diff\_x = Dx - Ax \tag{4.48}$$

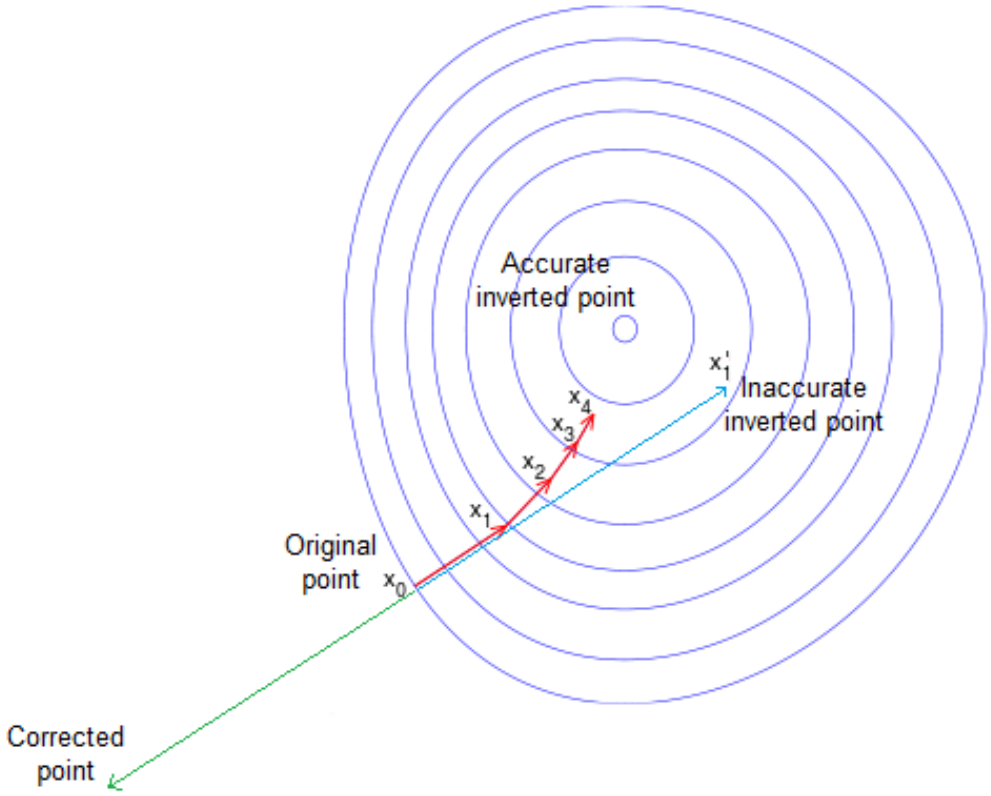

Figure 4.37: Comparison between the single jump and the multiple jumps counter correction procedure. Sourced from "Metodo di Edge-Detection basato sul calcolo di aree parziali" (*EdgeDetection method based on computation of partial areas*) [62] and edited.

$$Diff\_y = Dy - Ay \tag{4.49}$$

$Diff\_x$ and $Diff\_y$ gives us the correction vector to apply in the opposite direction to try to reach the inverted point. So a new point is obtained by adding the correction offset to the input point but scaled by a decaying factor which is needed for the algorithm to converge since the magnitude of the correction does not tend to zero as it would in the gradient descend algorithm:

$$Dx' = Dx + decay * Diff\_x \tag{4.50}$$

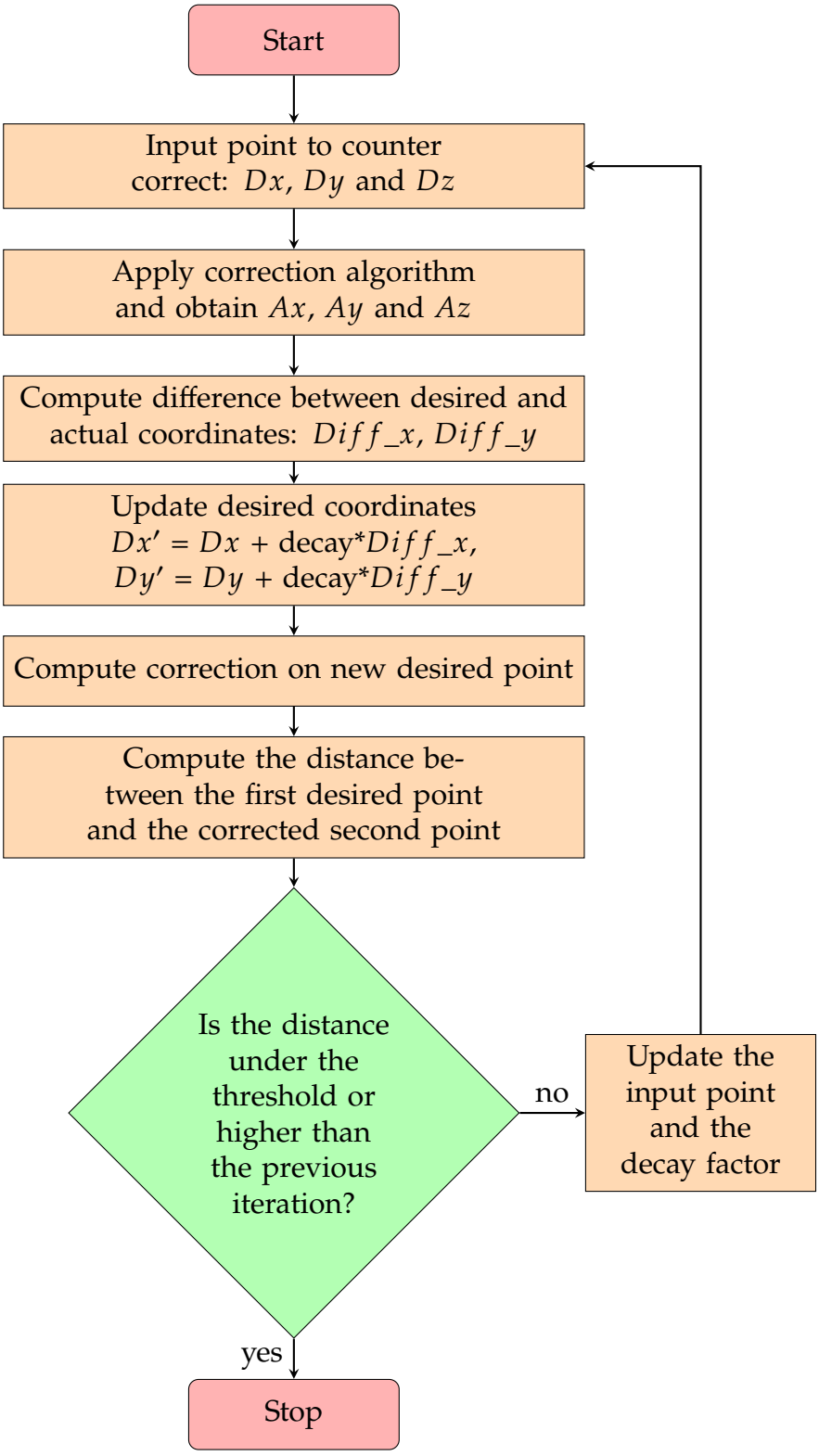

Figure 4.38: Flow to apply the iterative counter correction algorithm.

$$Dy' = Dy + decay * Diff\_y \tag{4.51}$$

Where Dx' and Dy' are candidate coordinates of a possible inverted point. To verify the quality of the inversion we need to apply the correction algorithm on D' and and compute the distance between the obtained point A' and the original point D:

$$d = \sqrt{(Ax' - Dx)^2 + (Ay' - Dy)^2} \tag{4.52}$$

To see if the algorithm should continue or not it is possible to compare the distance (d in the formula above) with a fixed threshold but it is usually better to compare it with the distance obtained in the previous iteration to see if the algorithm started diverging. At that point the algorithm can be stopped and the output is set to the results of the previous iteration. In the case in which the algorithm should continue the input point is set equal to D' and the decay factor is updated.

Different heuristics were put to test to update the decay factor such as keeping it a constant value or having it decay exponentially but the one that worked better in practise and allowed to reach the inverted point in the least number of jumps was to start with a high value for the decay, 0.9 and after each iteration set it equal to half the distance between D and A' so that it can auto regulate its magnitude depending on how close the algorithm is to converging.

In order to have a reality check on the efficacy of the iterative counter correction algorithm another algorithm is used to make a comparison. Figure 4.40 shows its functioning. The main idea is the same but instead of executing multiple jumps in an iterative fashion the correction offset are added to the input coordinates only once without any decaying factor. Even if the error model is not linear the correction offsets are small enough that this algorithm can still achieve a good approximation. However testing both algorithms in practice the iterative procedure always outperformed the single jump algorithm, Figure 4.39 [62] can give an idea of why is that. The red arrow shows the point obtained after apply-

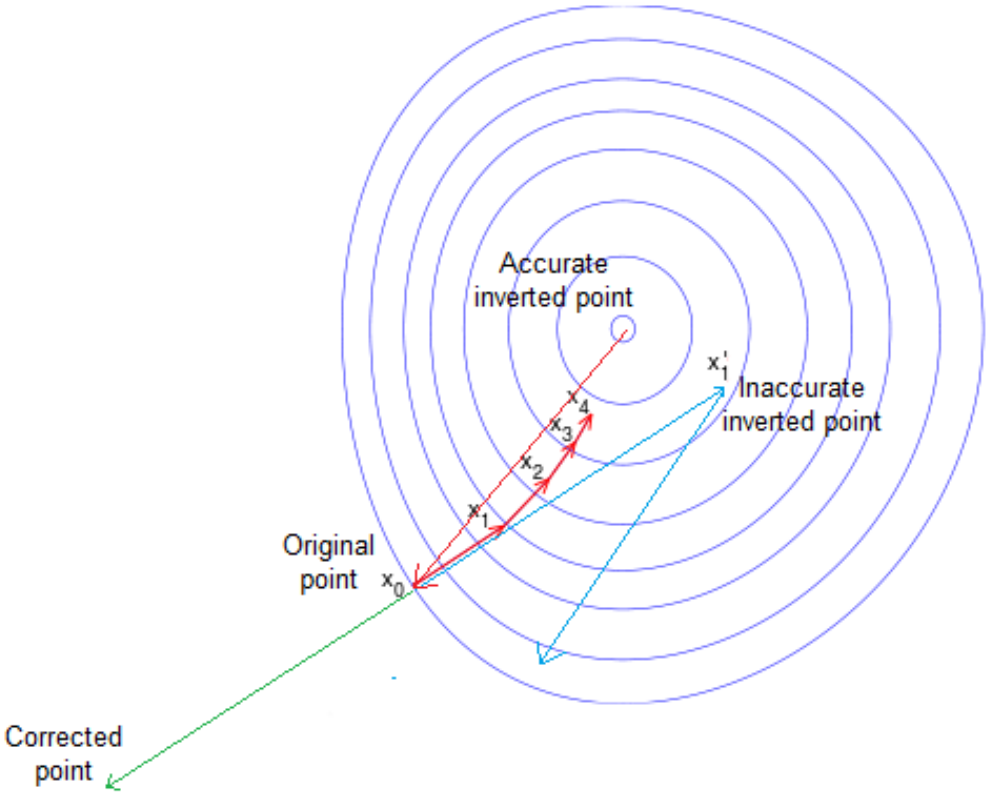

Figure 4.39: Comparison between the single jump and the multiple jumps counter correction procedure results. Sourced from "Metodo di Edge-Detection basato sul calcolo di aree parziali" (*EdgeDetection method based on computation of partial areas*) [62] and edited.

ing the correction algorithm on the inverse point computed with the multi jump algorithm while the blue arrow shows the point obtained after applying the correction algorithm on the inverse point computed with the single jump algorithm.

The complete procedure to apply the correction algorithm when the calibration board is rotated and translated between training and test is then to:

1. take the two points composing the test reference frame and apply the counter correction algorithm to them;
2. recompute the test reference frame;
3. proceed with the test as normal.

In this case it would still be necessary to save the first z offset and use it as a fixed correction to compensate fro height variations between train and test since the iterative procedure only considers x and y.

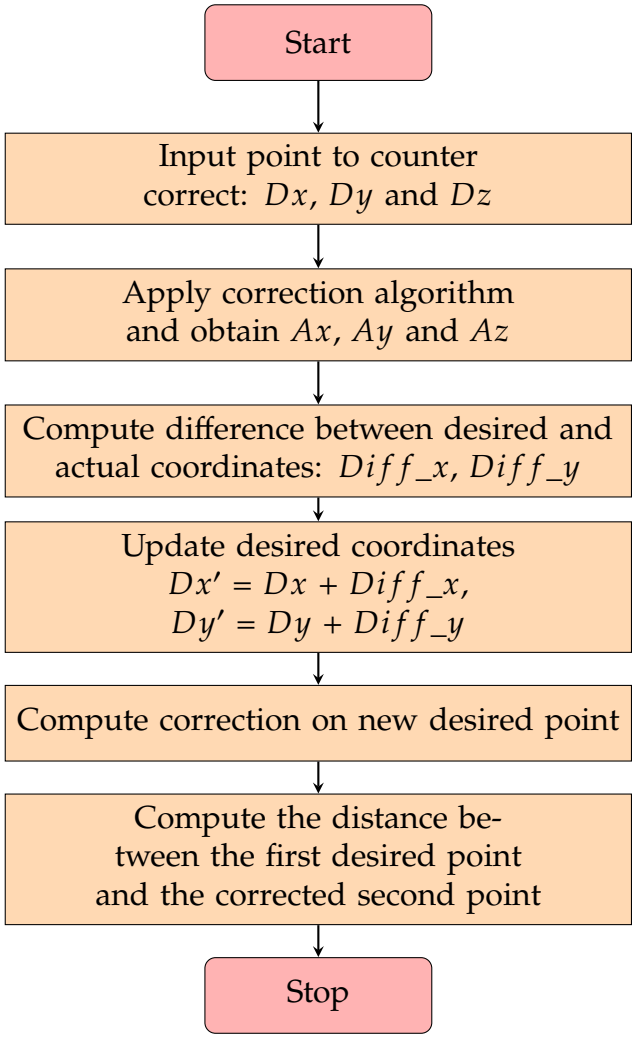

Figure 4.40: Flow to apply the single jump counter correction algorithm.

5 Results and discussion

In this chapter the results of all the experiments performed in this thesis will be taken into consideration, examined and discussed. Starting form the repeatability tests and following with the position errors measured both in Cartesian space and joint space. We will then discuss the error compensation obtained with the camera and calibration board (Figure 2.5) with all the algorithms considered in this thesis including bilinear interpolation and neural networks both in Cartesian and joint space, followed by some edge cases when the calibration board is translated and rotated between training and test. At the end for the test phase the camera and calibration board will be switched for an mechanical comparator and a milled aluminum plate to simulate a real application and have a mechanical feedback on the effectiveness of the whole pipeline (Figure 2.6).

5.1 Repeatability

The first results that will be taken into consideration are the repeatability test since they can give an idea of the best possible outcome we can expect from the correction algorithm in the sense that they set a higher limit on the improvement we can achieve.

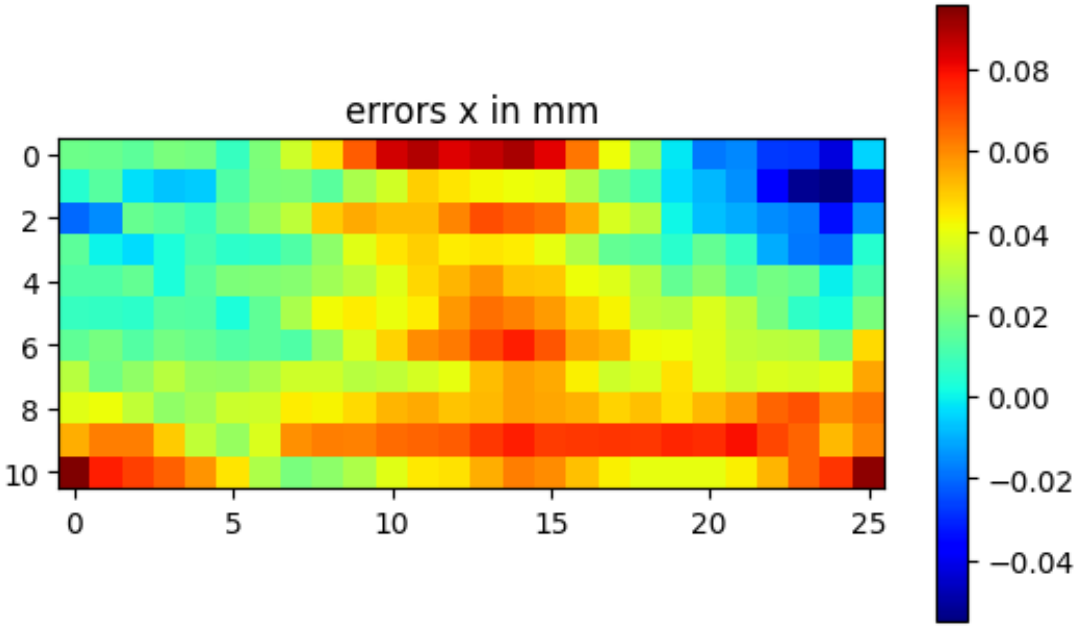

Figure 5.1: Robot repeatability error along the x axis, in the image the robot is positioned over the top at the center.

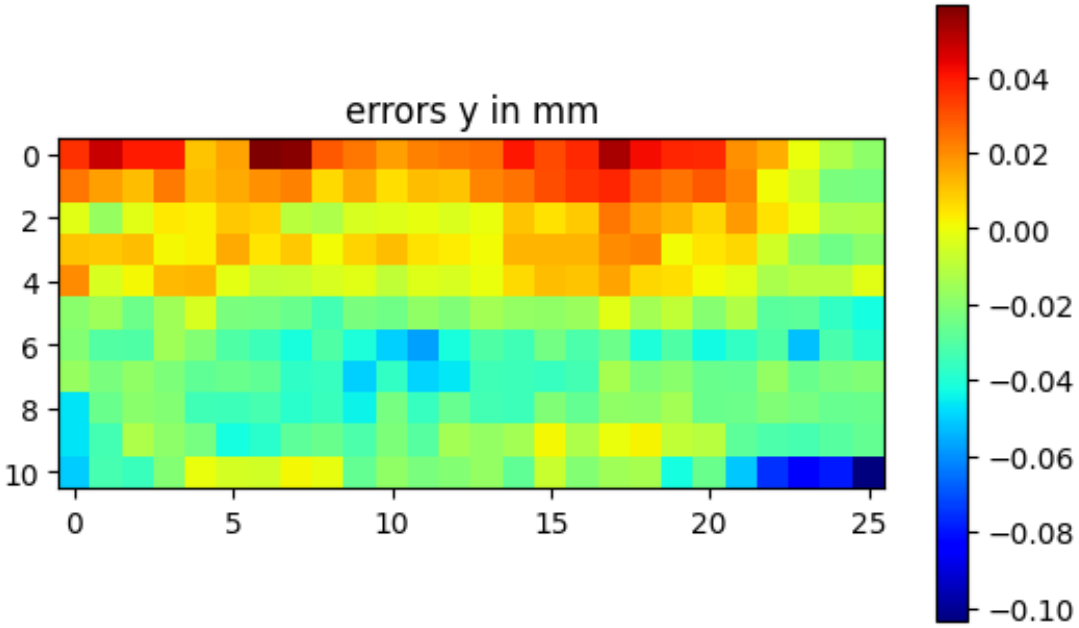

Figure 5.2: Robot repeatability error along the y axis, in the image the robot is positioned over the top at the center.

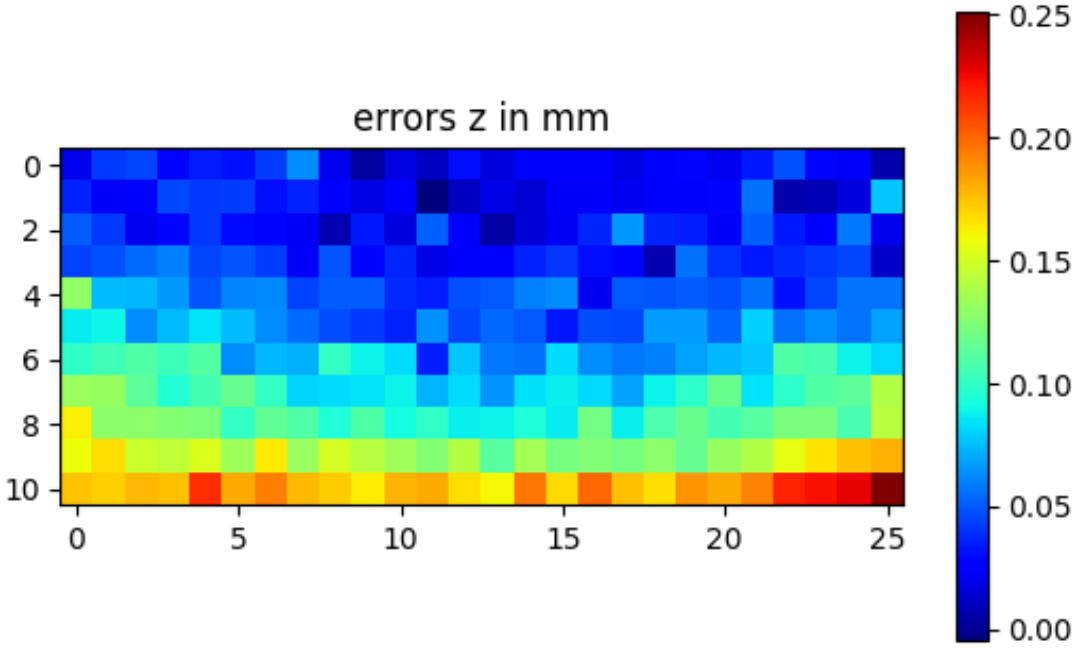

Figure 5.3: Robot repeatability error along the z axis, in the image the robot is positioned over the top at the center.

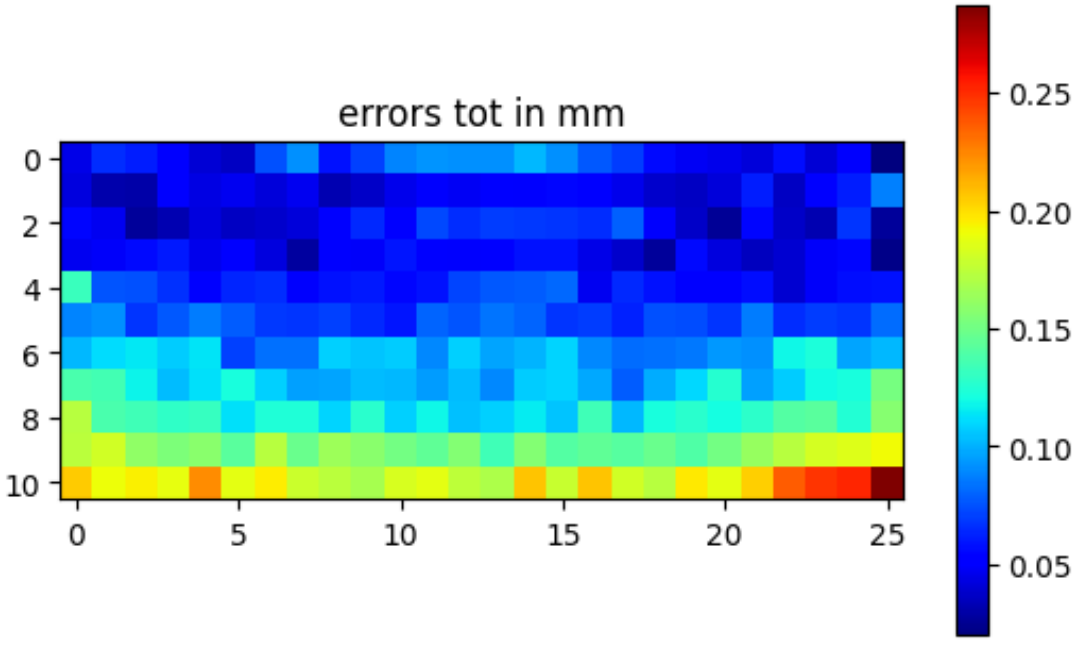

Figure 5.4: Robot total repeatability error, in the image the robot is positioned over the top at the center.

Figure 5.1 shows the repeatability errors of the robot measured with the camera along the x axis. Figure 5.2 shows the repeatability errors of the robot measured with the camera along the y axis and Figure 5.3 shows the repeatability errors of the robot measured with the camera along the z axis. Figure 5.4 instead shows the total repeatability errors of the robot. The points used to create the reference frame for these and all future images are the one on the bottom right constituting the origin of the frame and the one on the bottom left giving information about the orientation of the x axis. The robot is positioned over the top at the center and was moving row by row from right to left and from bottom to the top.

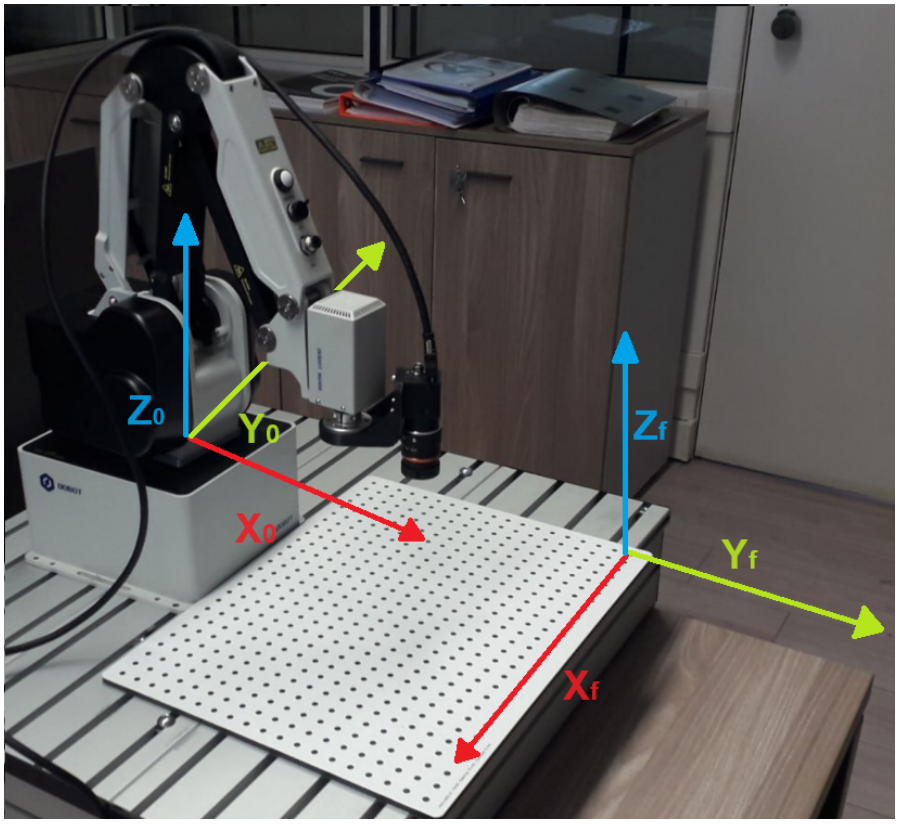

Figure 5.5: Reference frame zero and the reference frame solidal with the calibration board drawn on the experimental setup.

It is also worth noting that all the positions were converted back to frame zero before computing the errors along each axis, Figure 5.5 shows how the reference frame zero and the reference frame solidal with the calibration board are positioned relative to each other in the experimental setup. This way all errors are referred to the same reference frame. It is interesting to notice how the errors are smaller closer to the robot and get progressively bigger the more the robot arm has to stretch to reach the desired point. The following table reports the averages, standard deviations and maximum errors obtained during the experiment:

Error (mm)	**Total**	**X**	**Y**	**Z**
Average	0.066	0.037	0.022	0.043
Std	0.033	0.022	0.015	0.032
Max	0.228	0.095	0.104	0.181

The official repeatability for the Mg400 is of 0.05 mm and the average total error computed here is very close if it wasn't for the points at the edge of the robot workspace that threw off the value.

5.2 Position errors

For the position error maps the setup is the same as for the repeatability ones. Since the scale of the errors varies wildly between each axis a different scale is used for each one, but from now on the same scale will be applied to the corrections too. The setup is the same that was used for the repeatability tests: the points used to create the reference frame are the one on the bottom right constituting the origin of the frame and the one on the bottom left giving information about the orientation of the x axis. The robot is positioned over the top at the center (Figure 2.5) and was moving row by row from right to left and from bottom to the top. As for the repeatability errors all the positions were converted back to frame zero before computing the errors along each axis. This way all errors are referred to the same reference frame

5.3 Position errors in Cartesian space

The following graphs show the position error of the robot in Cartesian space.

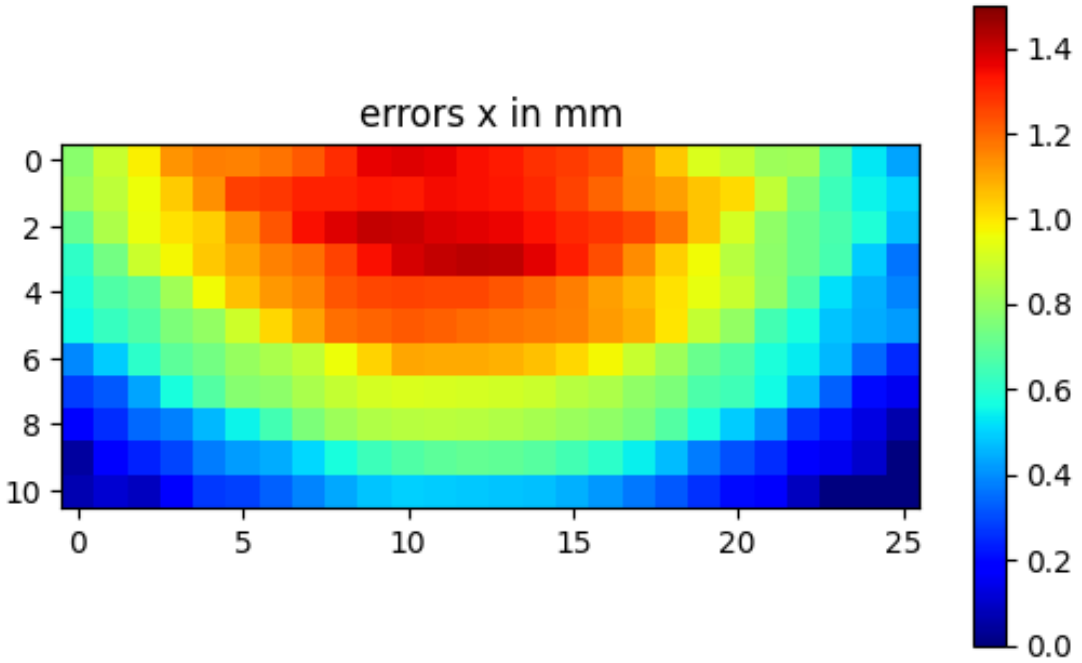

Figure 5.6: Robot position error along the x axis, in the image the robot is positioned over the top at the center.

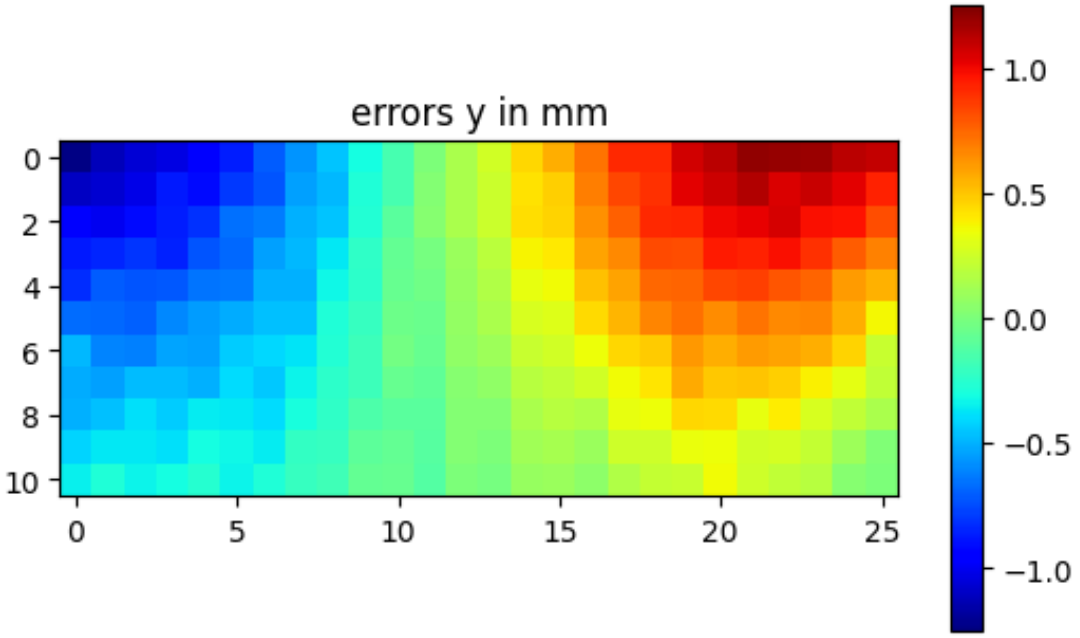

Figure 5.7: Robot position error along the y axis, in the image the robot is positioned over the top at the center.

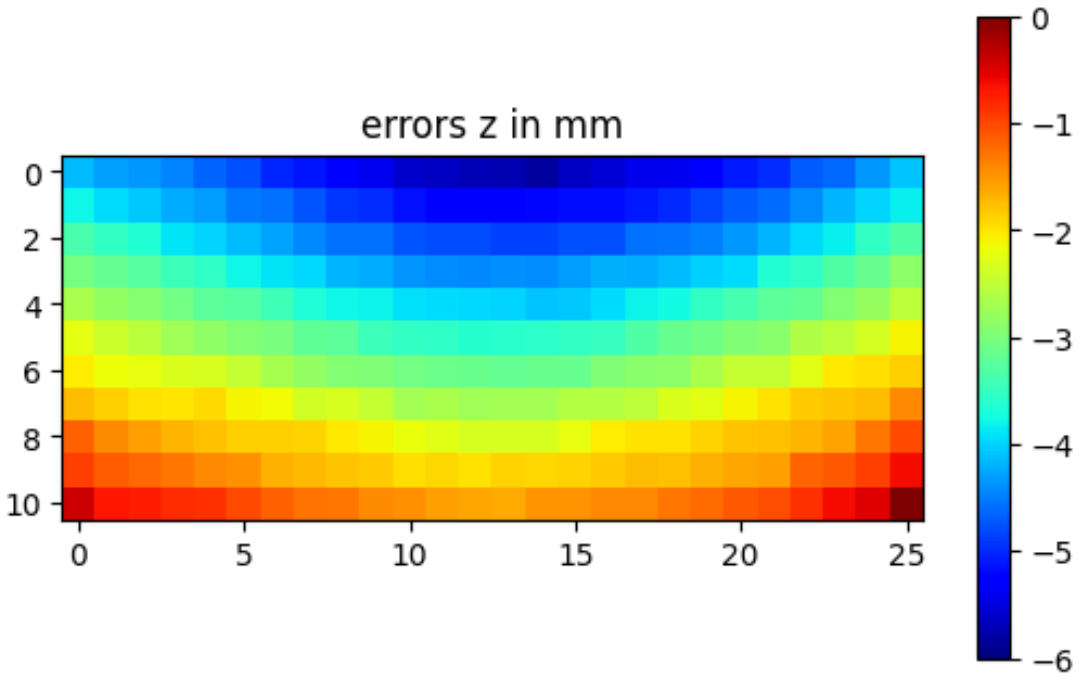

Figure 5.8: Robot position error along the z axis, in the image the robot is positioned over the top at the center.

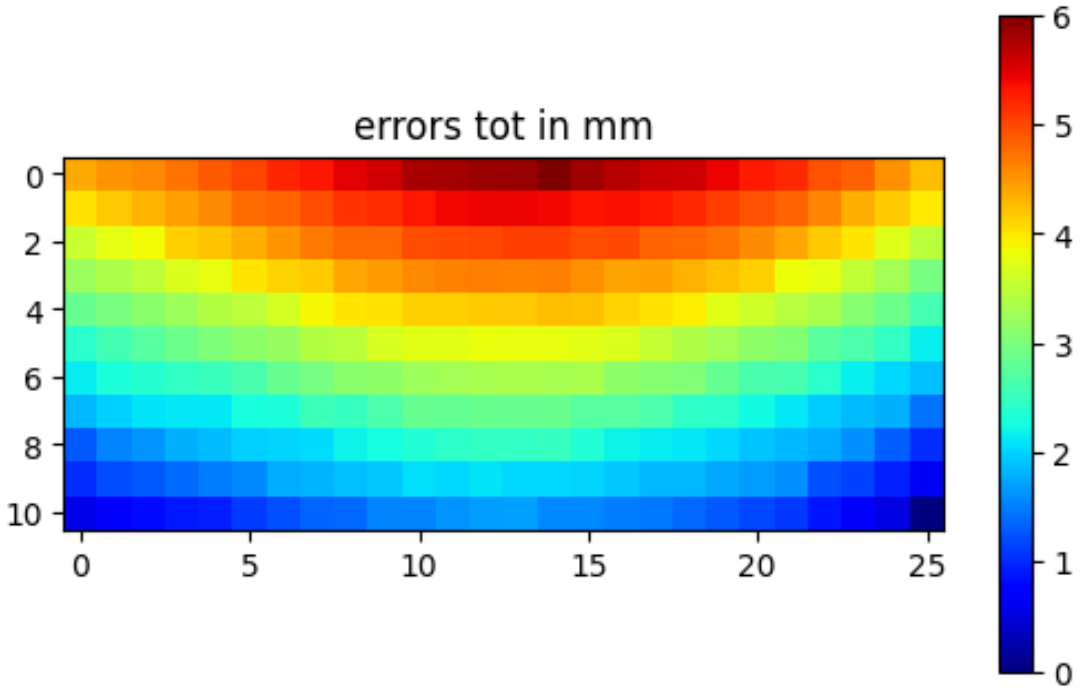

Figure 5.9: Robot total position error, in the image the robot is positioned over the top at the center.

Figure 5.6 shows the position errors of the robot measured with the camera along the x axis, Figure 5.7 shows the position errors of the robot measured with the camera along the y axis and Figure 5.8 shows the position errors of the robot measured with the camera along the z axis. Figure 5.9 instead shows the total position errors of the robot. The errors along the x and y axis show that the robot is not moving the camera in a straight line but at an arc. While the z errors shows that the robot is not moving on a plane parallel to the calibration board but on a concave imaginary surface. All of the graphs show an inverted phenomenon compared to the repeatability test where the error grew the more the robot arm stretched, here the position error seems to be bigger closer to the robot. This is an illusion given by the way the robot is moving from further away to closer to itself and that the starting points are manually placed. As mentioned in the translating frames section if the robot would start from the top left point for example the whole error model would shift for a fixed offset. The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error (mm)	Total	X	Y	Z
Average	3.211	0.799	0.472	3.055
Std	1.374	0.371	0.316	1.328
Max	5.986	1.423	1.230	5.829

From the data it looks like the position errors of the robot are quite large, especially along the z axis, but this leaves a lot of margin to improve upon. It is important to keep in mind the repeatability errors and these errors in the following sections too as they set a lower and upper bound on the error and will give an idea of the improvement reached by the correction algorithms.

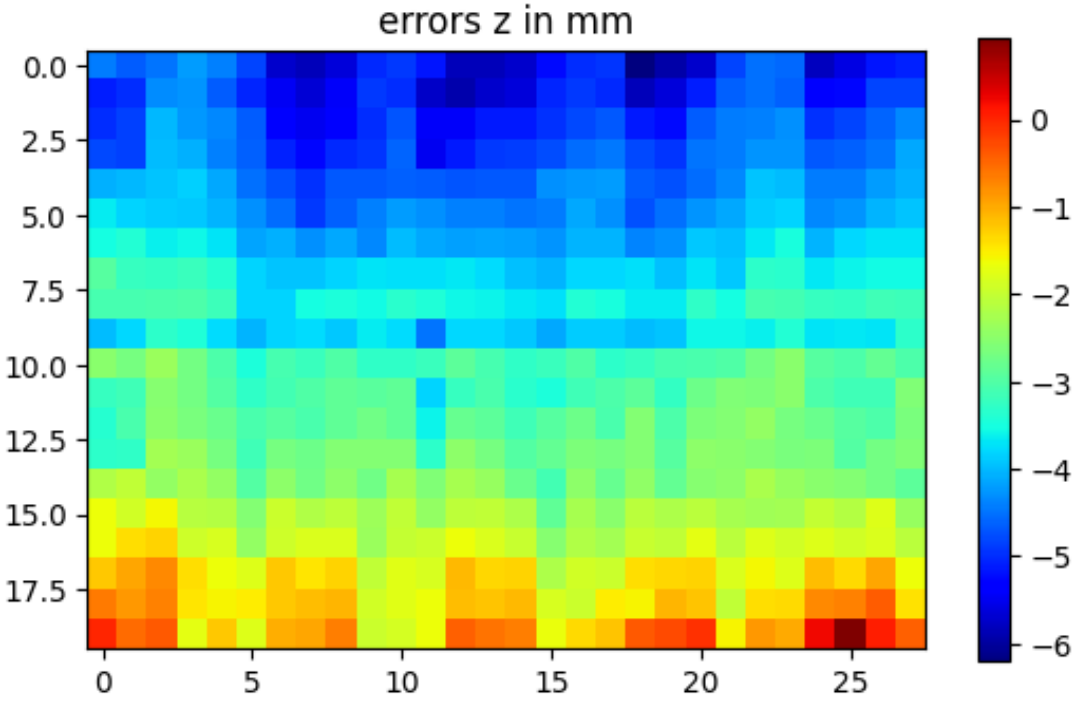

Figure 5.10: Robot position error along the z axis using a calibration board printed on a sheet of paper.

At this point it is also important to underline the importance of using a rigid calibration board for the experiments, Figure 5.10 shows the robot position error along the z axis using a calibration board printed on a sheet of paper and it is possible to see a weird undulating pattern happening along the horizontal axis. The pattern is due to a combination of the folds of the piece of paper that wasn't secured properly on a plane parallel to the ground and imperfections in the print. The effect disappeared once the LDPE calibration board came into play.

5.4 Position errors in joint space

The following graphs show the position error of the robot in Cartesian space. It is interesting to compare the position error in Cartesian space with the ones in joint space and find patters and similarities between the two.

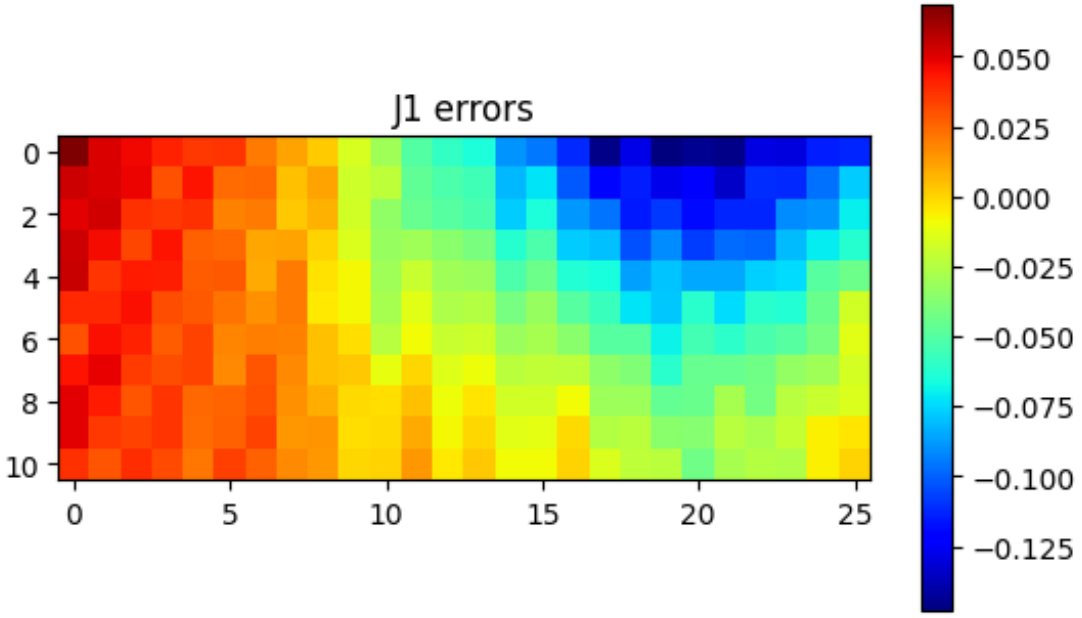

Figure 5.11: Robot position error in joint space along the first joint.

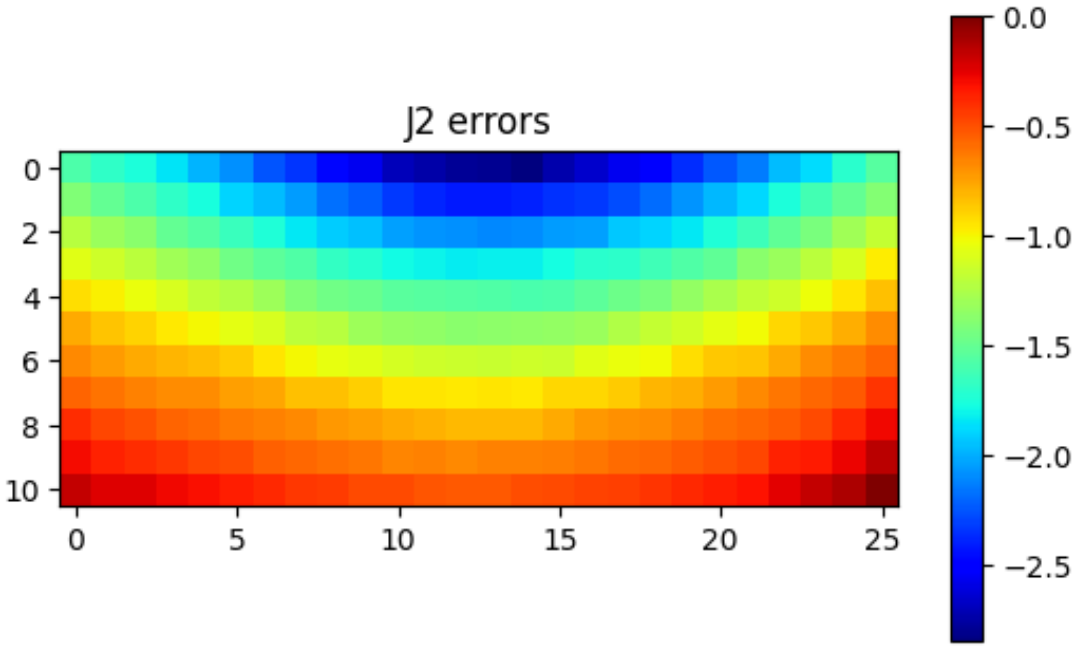

Figure 5.12: Robot position error in joint space along the second joint.

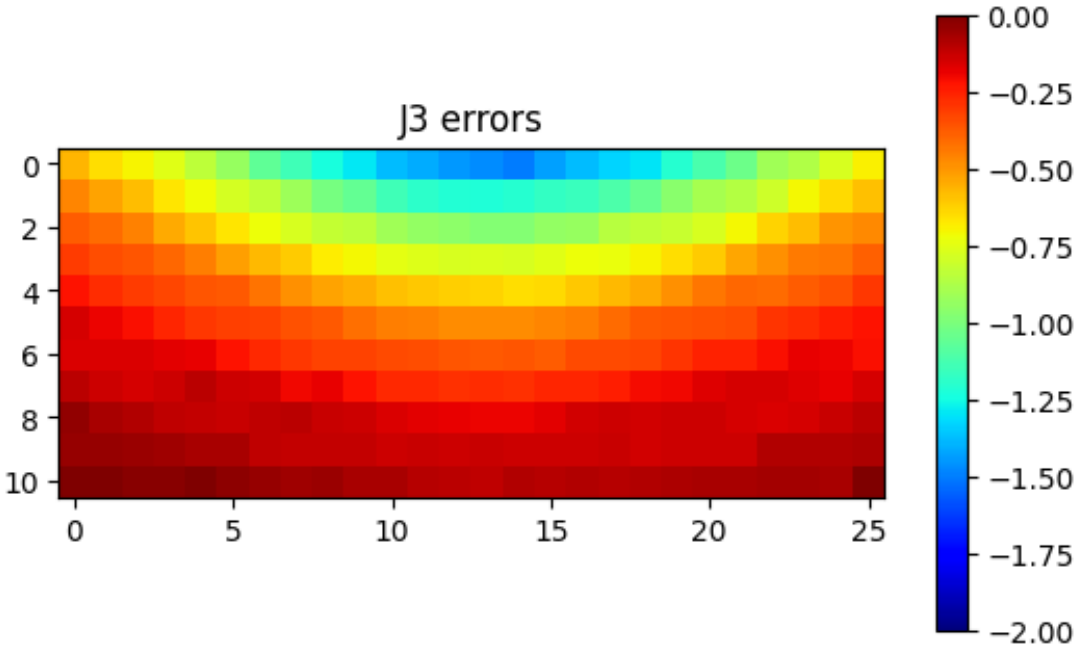

Figure 5.13: Robot position error in joint space along the third joint.

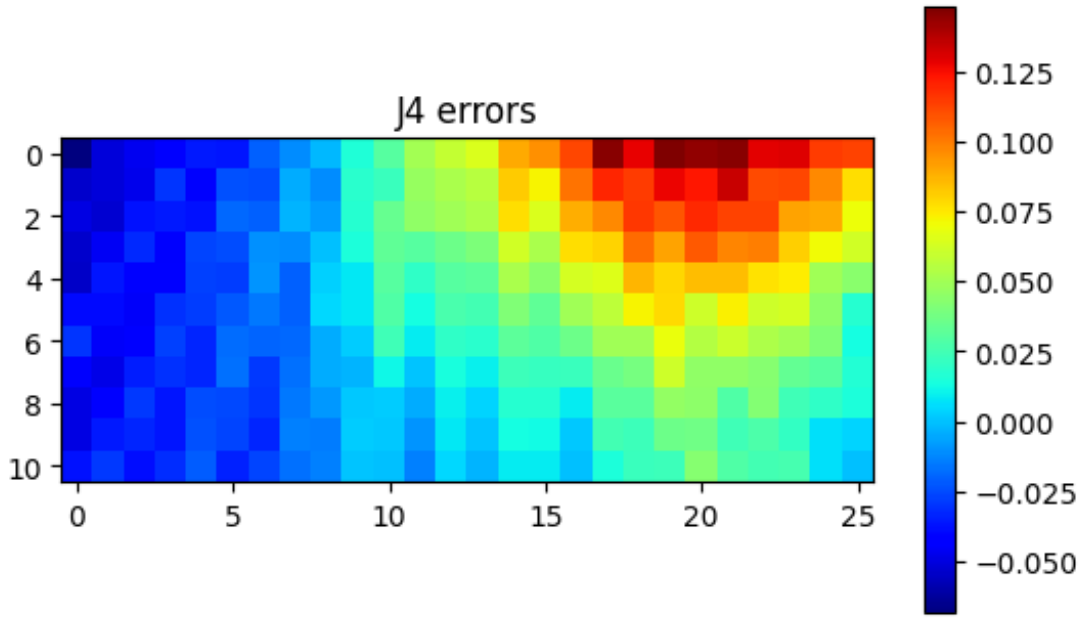

Figure 5.14: Robot position error in joint space along the fourth joint.

Figure 5.11 shows the position errors of the robot in joint space measured with the camera along the first joint, Figure 5.12 shows the position errors of the robot in joint space measured with the camera along the second joint, Figure 5.13 shows the position errors of the robot in joint space measured with the camera along the third joint and Figure 5.14 shows the position errors of the robot in joint space measured with the camera along the fourth joint. It is possible to notice form the images how much more regular and smooth the position error appear in joint space rather than in Cartesian space. The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error (deg)	**J1**	**J2**	**J3**	**J4**
Average	0.042	1.186	0.445	0.042
Std	0.033	0.643	0.360	0.033
Max	0.147	2.847	1.505	0.147

It is worth noticing how most of the error resides between J2 and J3 since their errors are at least an order of magnitude above the others. The remaining error is split perfectly even between J1 and J4 and influences more the orientation of the tool rather than its position. This reinforces the hypothesis of giving as

input to the neural network in joint space only the values of the first three joints to have the correction independent of the tool used by the robot.

5.5 Camera Correction by bilinear interpolation

The following graphs show the residual position error of the robot after applying the correction algorithm using bilinear interpolation. Between train and test the calibration board was moved so that no two points would match.

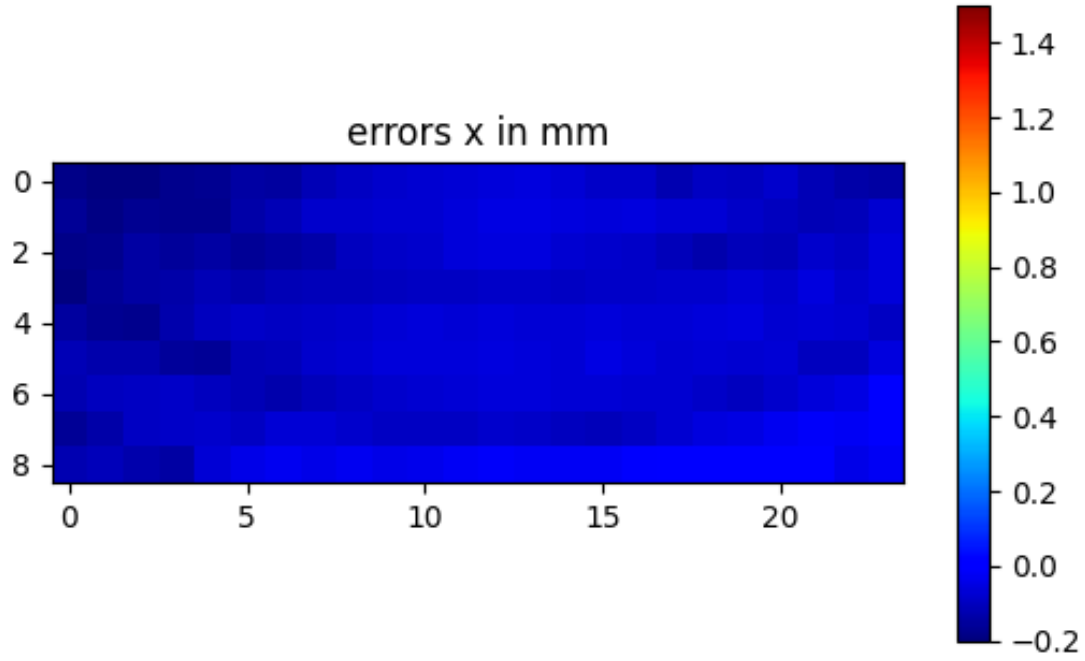

Figure 5.15: Robot residual position error along the x axis after correction with bilinear interpolation.

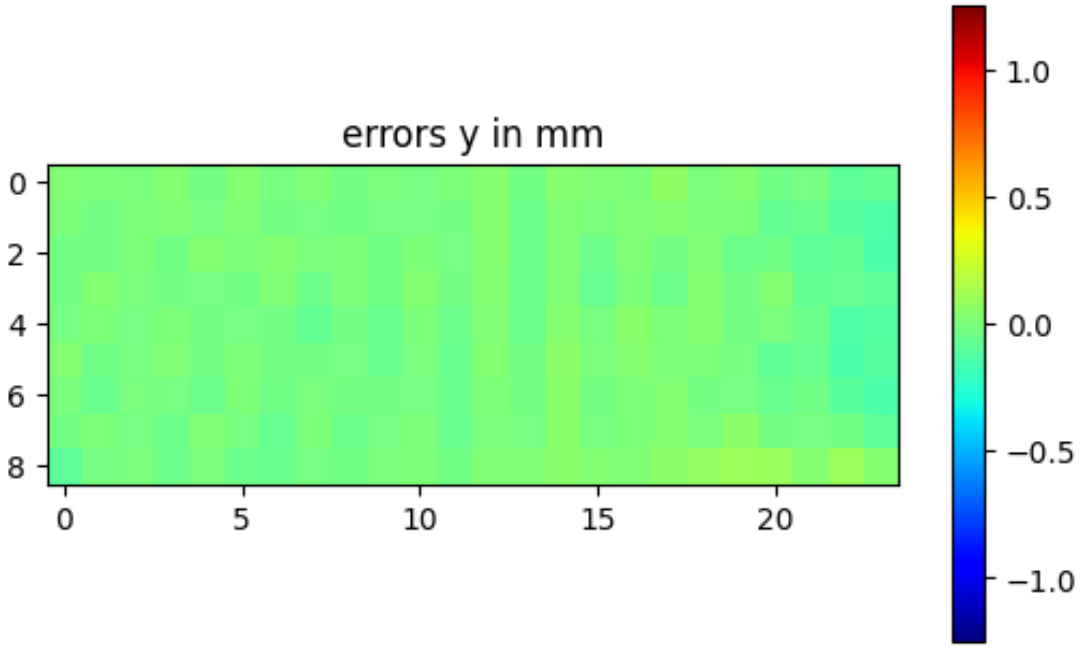

Figure 5.16: Robot residual position error along the y axis after correction with bilinear interpolation.

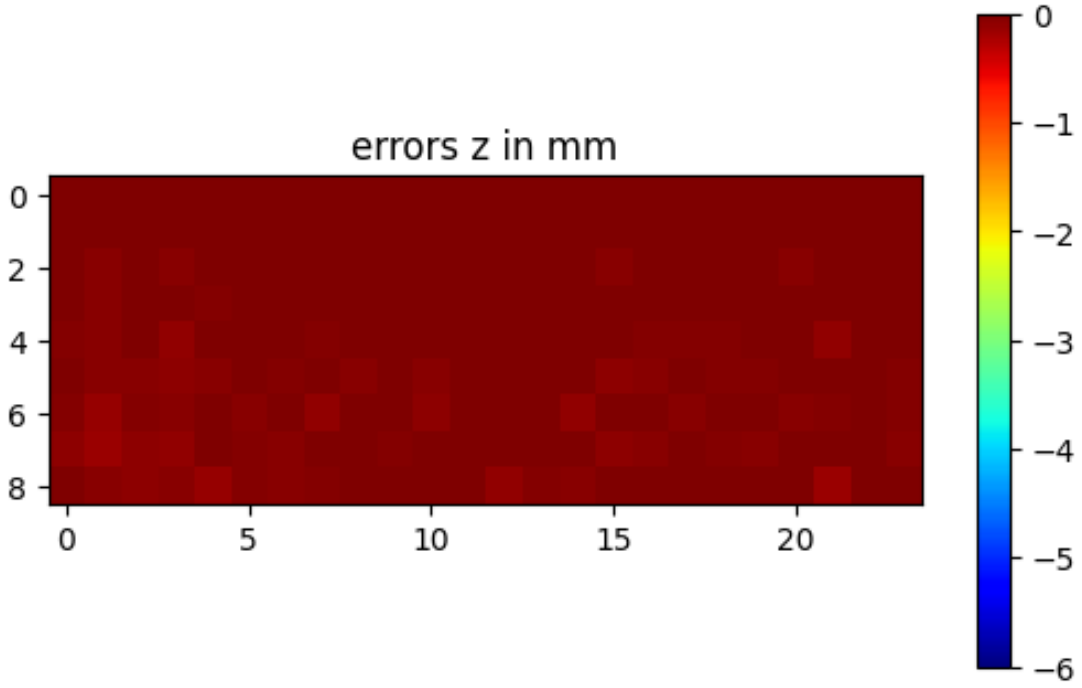

Figure 5.17: Robot residual position error along the z axis after correction with bilinear interpolation.

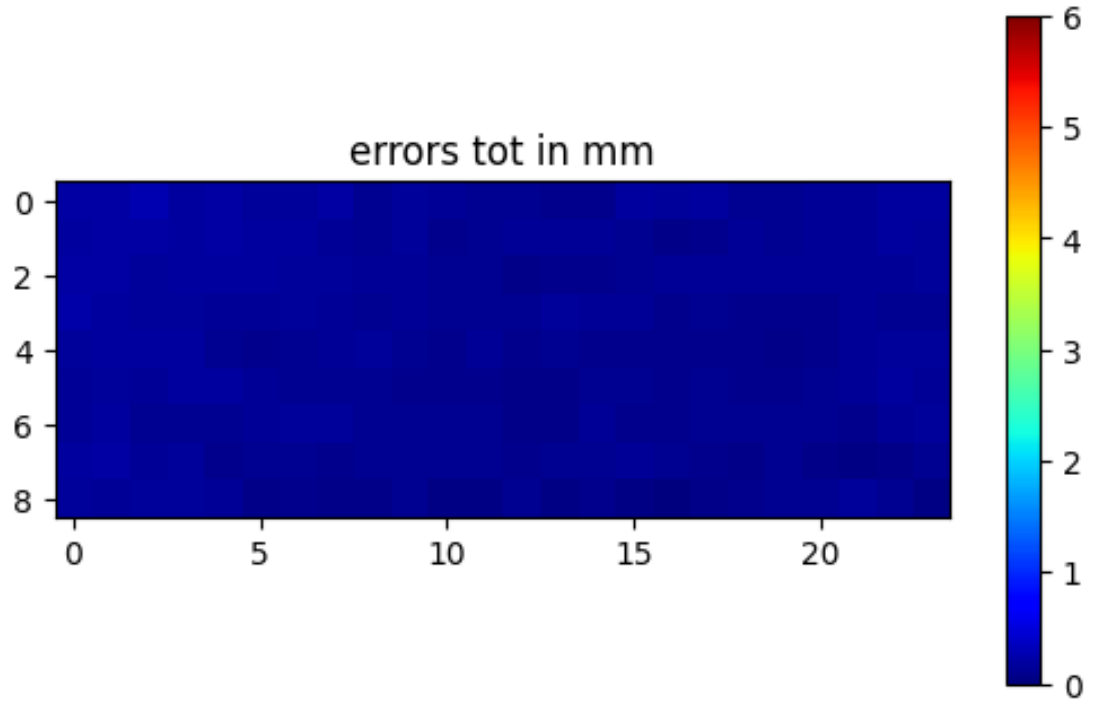

Figure 5.18: Robot total residual position error after correction with bilinear interpolation.

Figure 5.15 shows the residual position errors of the robot measured with the camera along the x axis after the correction with bilinear interpolation, Figure 5.16 shows the residual position errors of the robot measured with the camera along the y axis after the correction with bilinear interpolation and Figure 5.17 shows the residual position errors of the robot measured with the camera along the z axis after the correction with bilinear interpolation. Figure 5.18 instead shows the total residual position errors of the robot measured with the camera after the correction with bilinear interpolation. The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error (mm)	**Total**	**X**	**Y**	**Z**
Average	0.119	0.091	0.031	0.049
Std	0.039	0.042	0.029	0.036
Max	0.269	0.219	0.145	0.156

Although there is still some residual error the final results shows that the correction algorithm improved the accuracy of the robot in the range of its

repeatability. This was in the ideal case in which the calibration bard was just shifted between the data gathering step and the test phase but it is also the most common occurrence as the calibration should be performed with an application already in mind so there should not be big variation between the two phases.

5.6 Camera Correction by Neural network in Cartesian space

The following graphs show the residual position error of the robot after applying the correction algorithm using the neural network in Cartesian space. Between train and test the calibration board was moved so that no two points would match.

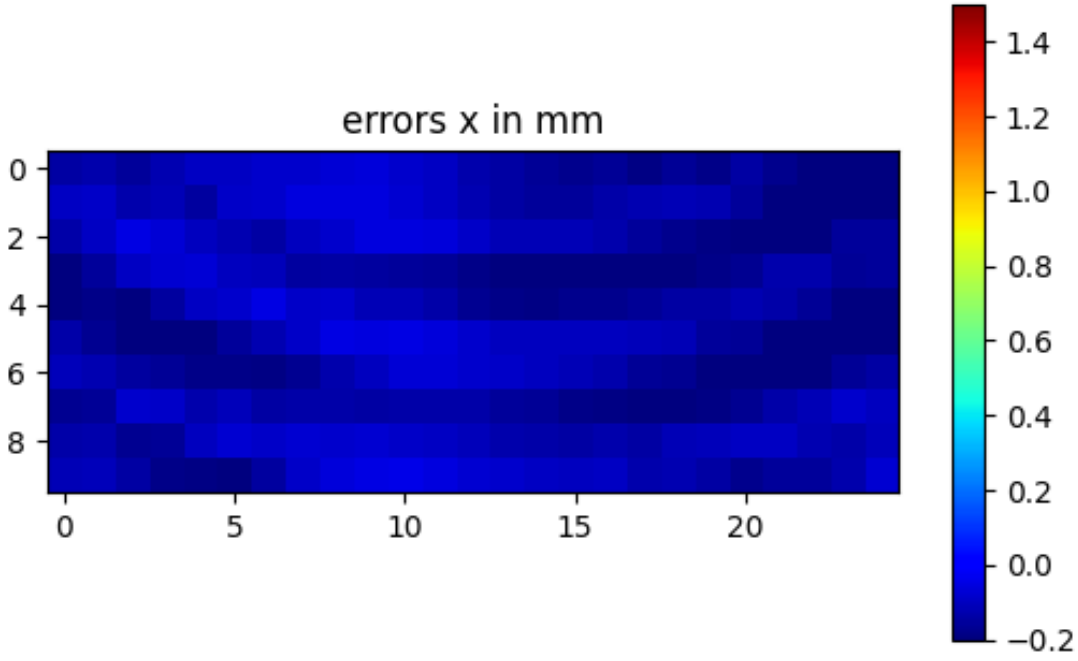

Figure 5.19: Robot residual position error along the x axis after correction with the neural network in Cartesian space.

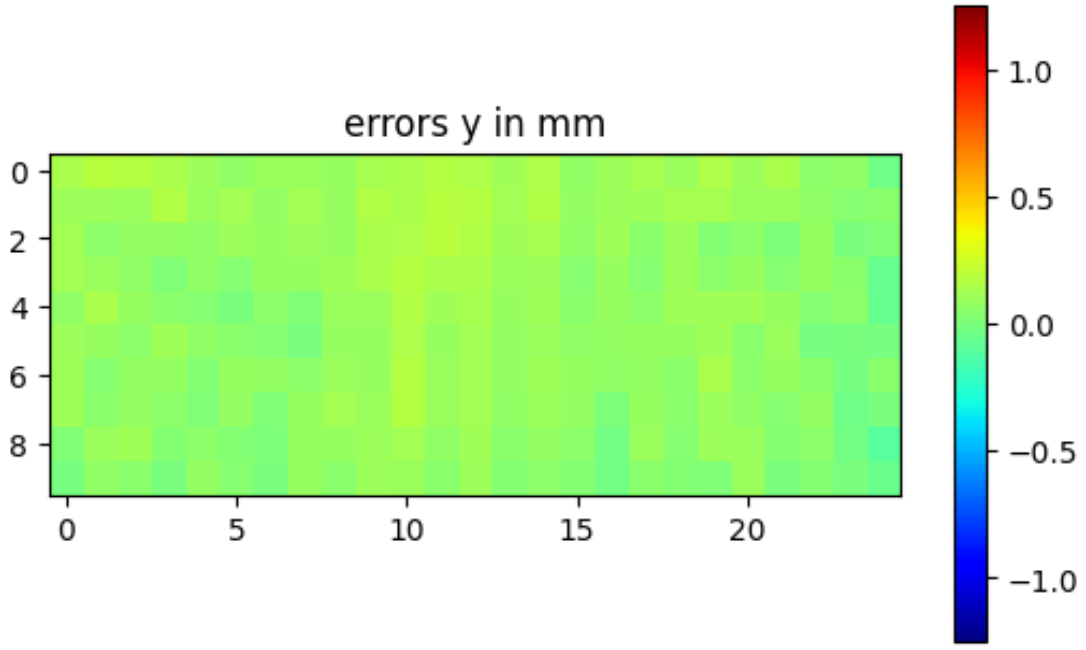

Figure 5.20: Robot residual position error along the y axis after correction with the neural network in Cartesian space.

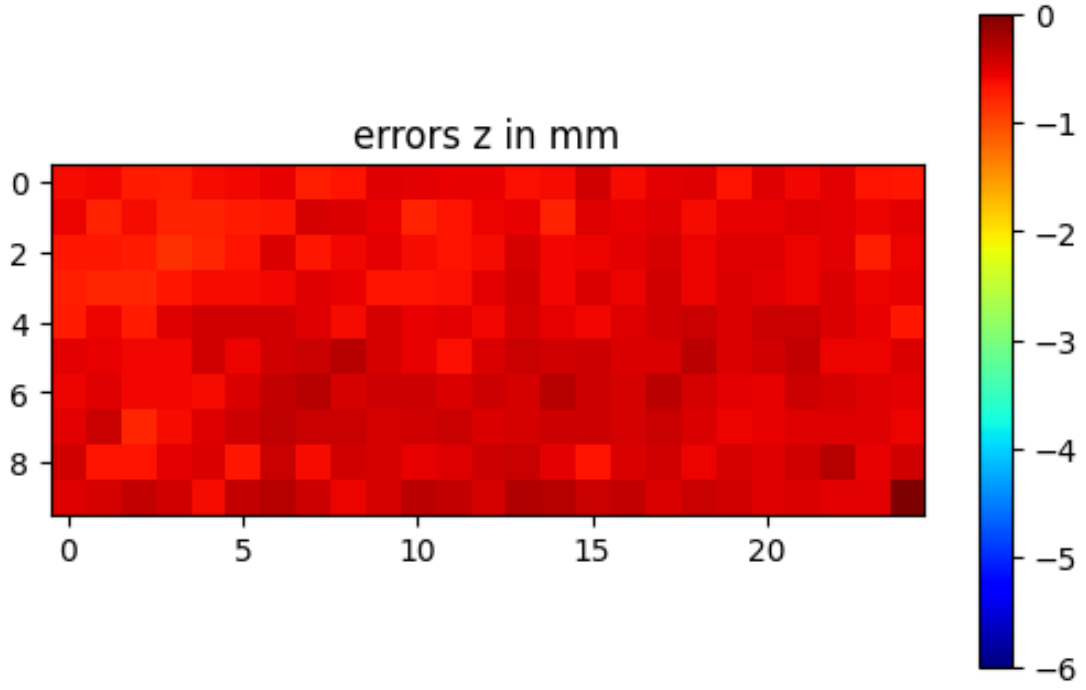

Figure 5.21: Robot residual position error along the z axis after correction with the neural network in Cartesian space.

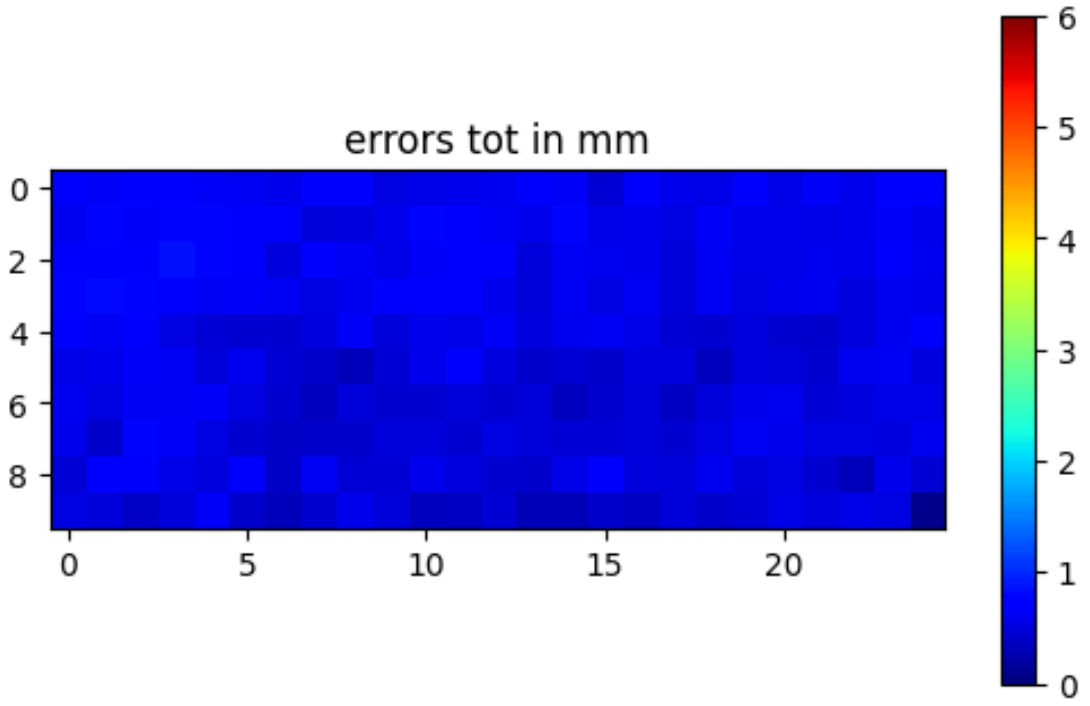

Figure 5.22: Robot total residual position error after correction with the neural network in Cartesian space.

Figure 5.19, 5.20 and 5.21 show the residual position errors of the robot measured with the camera along the x, y and z axis respectively after the correction with the neural network in Cartesian space. Figure 5.18 instead shows the total residual position errors of the robot measured with the camera after the correction with the neural network in Cartesian space. The waving pattern that Figure 5.19 shows is probably due to the vibrations of the robotic arm or some fixed steps on the motors. The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error (mm)	**Total**	**X**	**Y**	**Z**
Average	0.552	0.136	0.079	0.525
Std	0.114	0.047	0.043	0.117
Max	0.850	0.283	0.183	0.844

The neural network in Cartesian space provides an important improvement on the accuracy of the robot but it isn't as good as the one obtained by the bilinear interpolation. This can be due to the fact that having a very densely populated

calibration board the optimal solution is already very close to the linear solution so adding the uncertainty of the neural network training on top of it does not help.

5.7 Translating frames

The following graphs show the position error of the robot when the data gathering step is performed after translating the origin of the frame associated with the calibration board to the point at the top left and the test phase has been performed with the first point coinciding with the one on the bottom right. The graph on the left shows the error model after the translation, we can notice how different it is with respect to the one seen before when the origin of the frame was on the bottom right. The graph on the right instead shows what happens when the value that would be used in the correction of the first point in the test phase is added, along each axis, to the error model. It is possible to appreciate from the image how the graphs on the right are very similar to the ones seen before when the origin of the frame was on the bottom right (the scale was changed to be able to keep it constant in the before-after process). This shows how easy it is to adapt the error model when the calibration board is translated between training and test and thanks to it the correction algorithm can operate undisturbed.

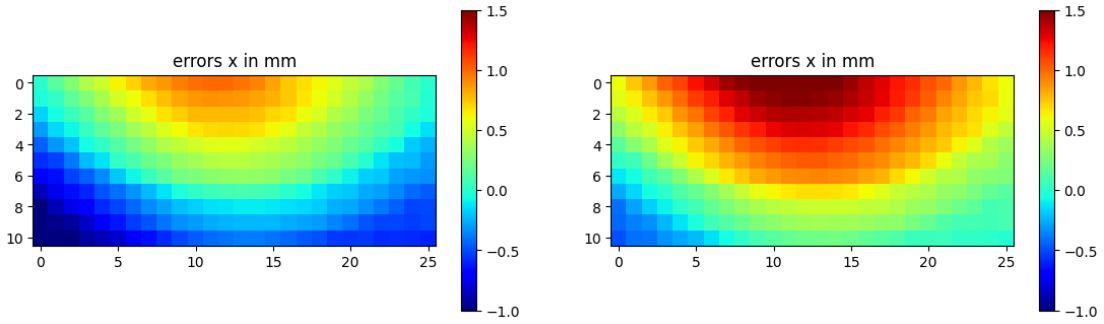

Figure 5.23: Error along the x axis with a translated frame before and after adding the fixed counter correction offset.

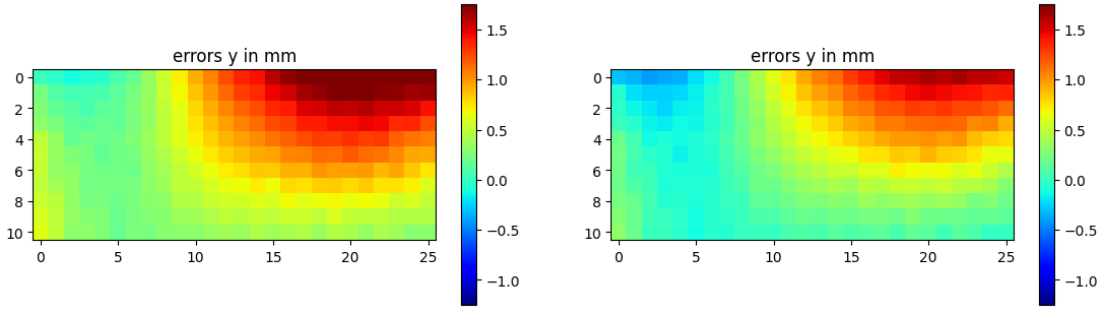

Figure 5.24: Error along the y axis with a translated frame before and after adding the fixed counter correction offset.

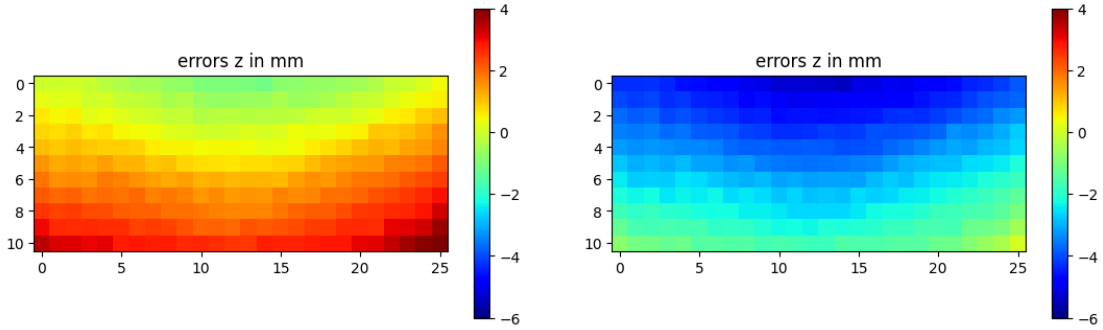

Figure 5.25: Error along the z axis with a translated frame before and after adding the fixed counter correction offset.

Figure 5.23, 5.24 and 5.25 show how the error model changes before (left) and after (right) applying the fixed offset counter correction along the x, y and z axis respectively.

Since it was the method that was performing better in the optimal case only the bilinear interpolation algorithm is tested against this particular edge case. The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error (mm)	**Total**	**X**	**Y**	**Z**
Average	0.153	0.062	0.039	0.124
Std	0.054	0.041	0.031	0.054
Max	0.262	0.173	0.153	0.224

It is possible to notice how the correction algorithm is not performing as good as it was performing in ideal conditions, even though the improvements

are still very good but looking at the graphs of the error model it is easy to see that it would have gone completely off course without the counter correction procedure.

5.8 Rotating frames

The following graphs show the position error of the robot when the data gathering step is performed after rotating the reference frame associated with the calibration board by 90°. The new reference frame ha still the origin on the point at the bottom right but the direction of the x axis is given by the point at the top right instead of the one at the bottom left as it was used until now.

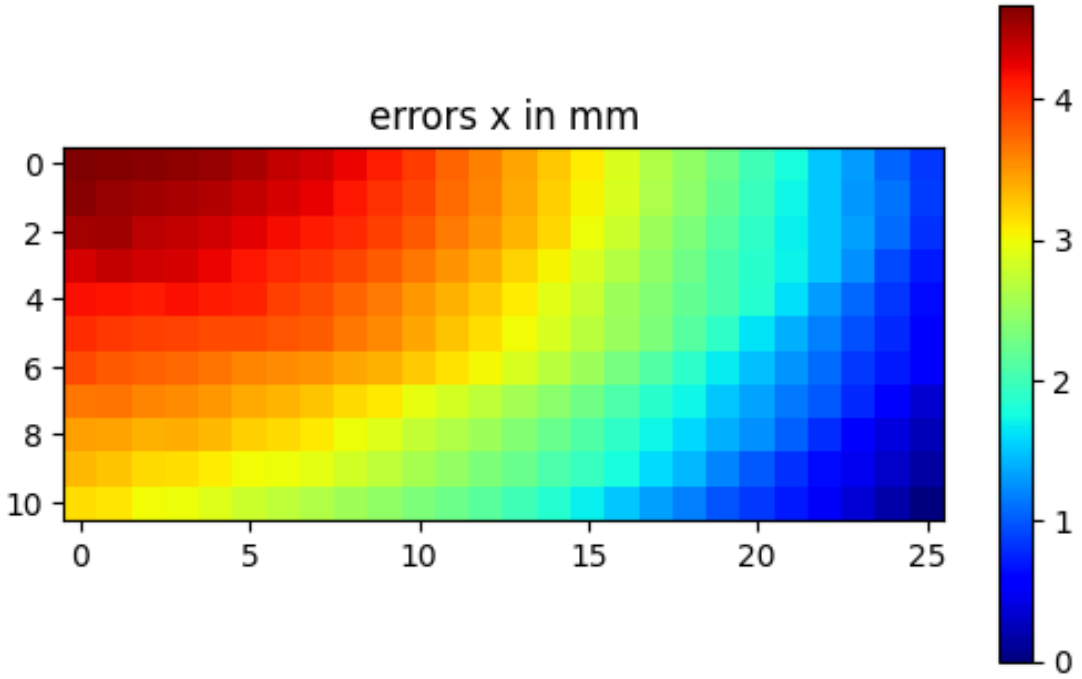

Figure 5.26: Robot position error along the x axis after rotating the calibration board reference frame by 90°.

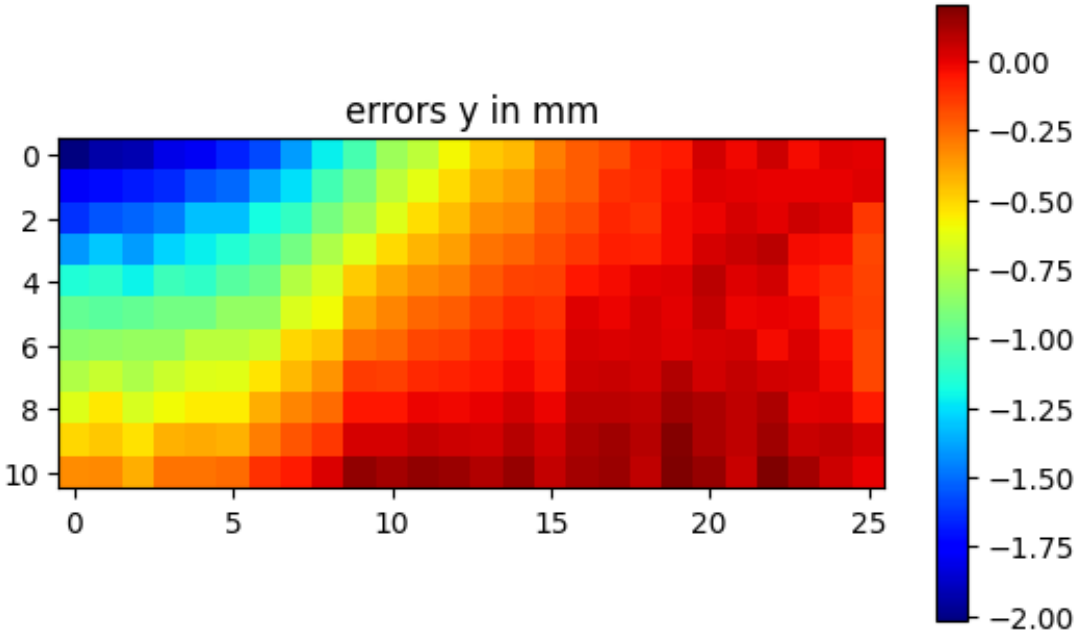

Figure 5.27: Robot position error along the y axis after rotating the calibration board reference frame by 90°.

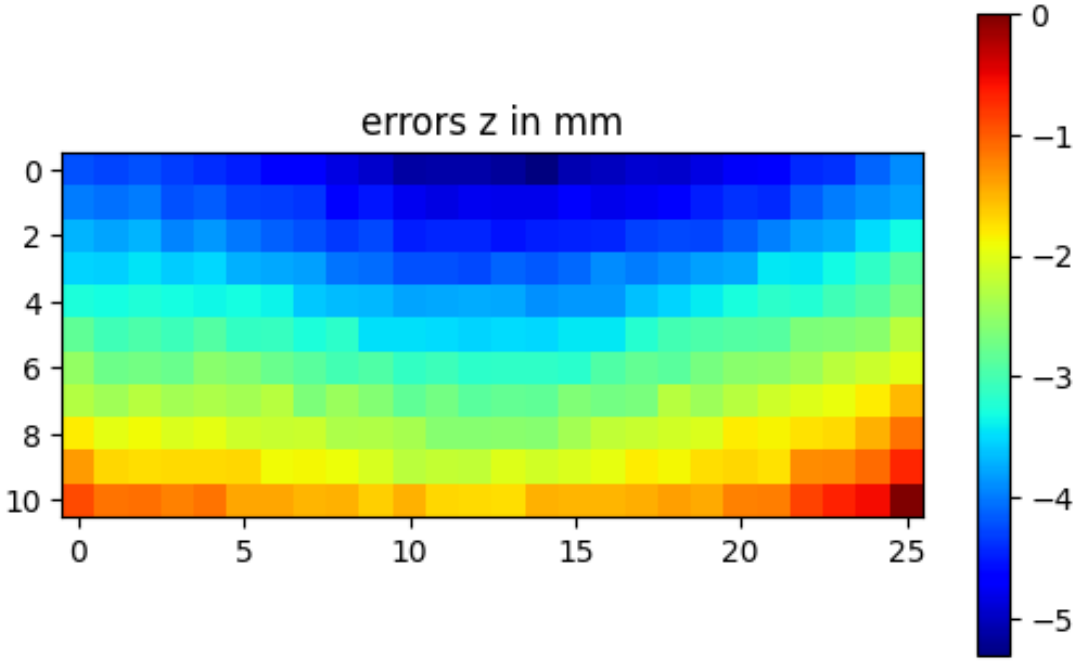

Figure 5.28: Robot position error along the z axis after rotating the calibration board reference frame by 90°.

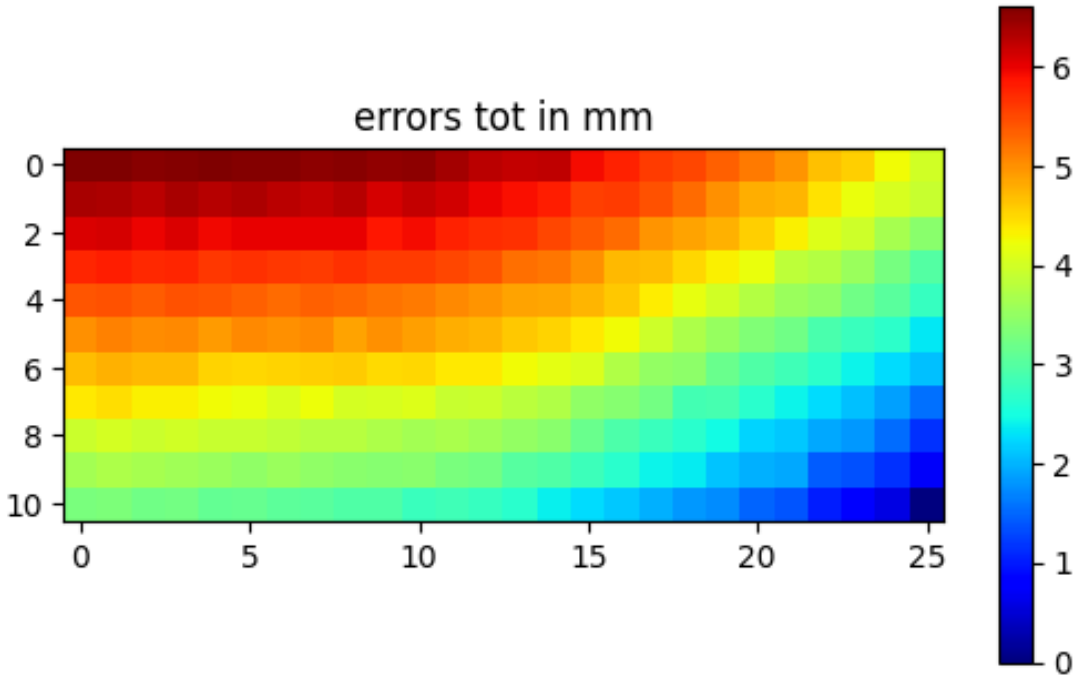

Figure 5.29: Robot total position error after rotating the calibration board reference frame by 90°.

Figure 5.26, 5.27 and 5.28 show the position errors of the robot measured with the camera along the x, y and z axis respectively after rotating the calibration board reference frame by 90°. Figure 5.29 instead shows the total position errors of the robot measured with the camera after rotating the calibration board reference frame by 90°. It is possible to notice how the error pattern is completely different along the x and y axis, no single fixed offset could bring it back it to the original error model like we did in the translating frames section. This is because centering the robot on the point at the top right of the calibration board performed a correction, especially along the y axis that wasn't present before . The error model along the z axis instead is practically unchanged as the rotation of the reference frame did not affect the vertical axis at all since the reference frame is generated by only two points the direction of the z axis is always kept parallel to the z axis of frame zero.

Since it was the method that was performing better in the optimal case only the bilinear interpolation algorithm is tested against this particular edge case. To evaluate the effectiveness of the the correction algorithm it was tested without any counter correction procedure before and after with the counter

correction procedure. In both cases during the test phase the reference frame of the calibration board was shifted back to its original position to simulate a rotation. The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error tot (mm)			
	No correction	**Bilinear interpolation**	**Bilinear counter correction**
Average	4.199	0.510	0.111
Std	1.418	0.100	0.055
Max	6.613	0.757	0.336
Error x (mm)			
	No correction	**Bilinear interpolation**	**Bilinear counter correction**
Average	2.672	0.278	0.051
Std	1.193	0.085	0.048
Max	4.672	0.466	0.179
Error y (mm)			
	No correction	**Bilinear interpolation**	**Bilinear counter correction**
Average	0.415	0.095	0.050
Std	0.486	0.041	0.033
Max	2.019	0.173	0.133
Error z (mm)			
	No correction	**Bilinear interpolation**	**Bilinear counter correction**
Average	3.067	0.405	0.066
Std	1.123	0.094	0.052
Max	5.306	0.743	0.319

From the first column it is apparent how a component of the position error shifted around between the x and y axis while the total error remain almost unchanged, which underlines the difference with the error models shown previously. The second column shows the results obtained with the bilinear interpolation algorithm without any counter correction applied. It is possible to appreciate that the improvement is still very substantial even though it it nowhere near the one obtained in optimal conditions. The last column instead shows he results obtained with the bilinear interpolation algorithm when combined with the counter correction procedure. The improvement with respect

to the previous case is quite impressive and the residual position error of the robot is very close to its repeatability leaving very small margin for further improvements.

5.9 Comparator position errors in Cartesian space

In order to have a mechanical feedback on the efficacy of the correction algorithms the camera and calibration board were switched for an mechanical comparator and a milled aluminum plate to simulate a real application (Figure 2.6).

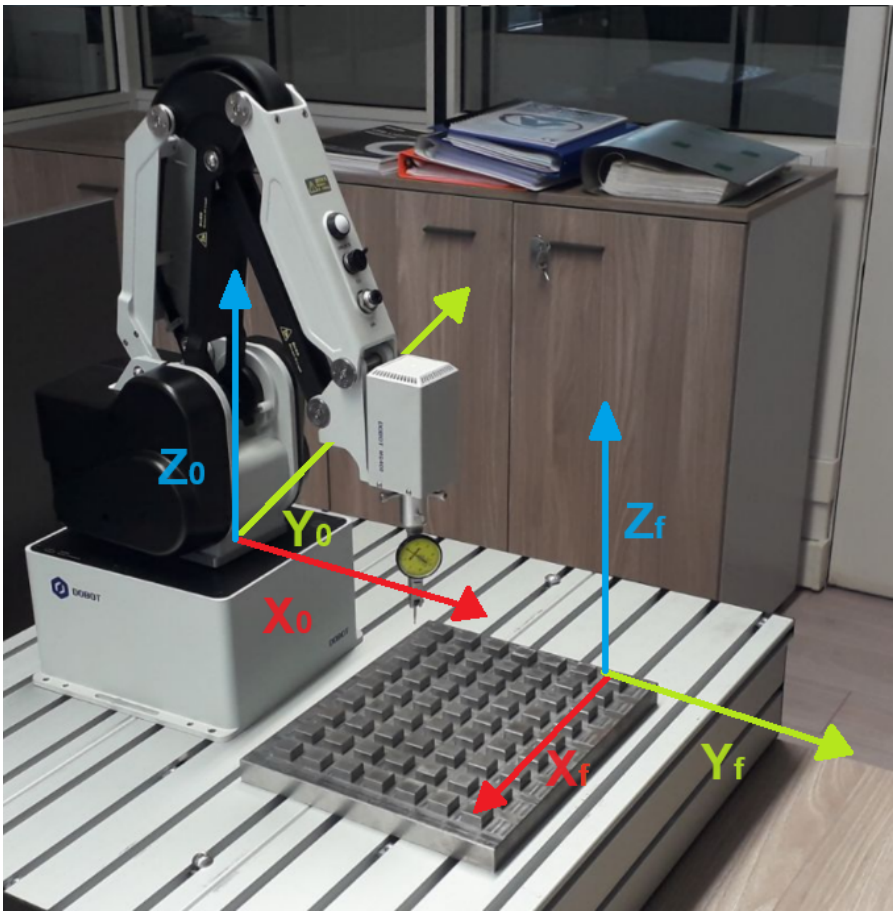

Figure 5.30: Reference frame zero and the reference frame solidal with the aluminum plate drawn on the experimental setup.

Since the comparator can only measure the error in one direction at a time the data gathering procedure has to be run three times, one for each axis with the robot first approaching and then position in the comparator's pointer in contact with the predisposed spot on the plate. For the x and y axis the comparator is moved on both sides of the predisposed spots in order to measure both positive and negative errors. As before all the positions were converted back to frame zero before computing the errors along each axis, Figure 5.30 shows how the reference frame zero and the reference frame solidal with the aluminum plate are positioned relative to each other in the experimental setup.

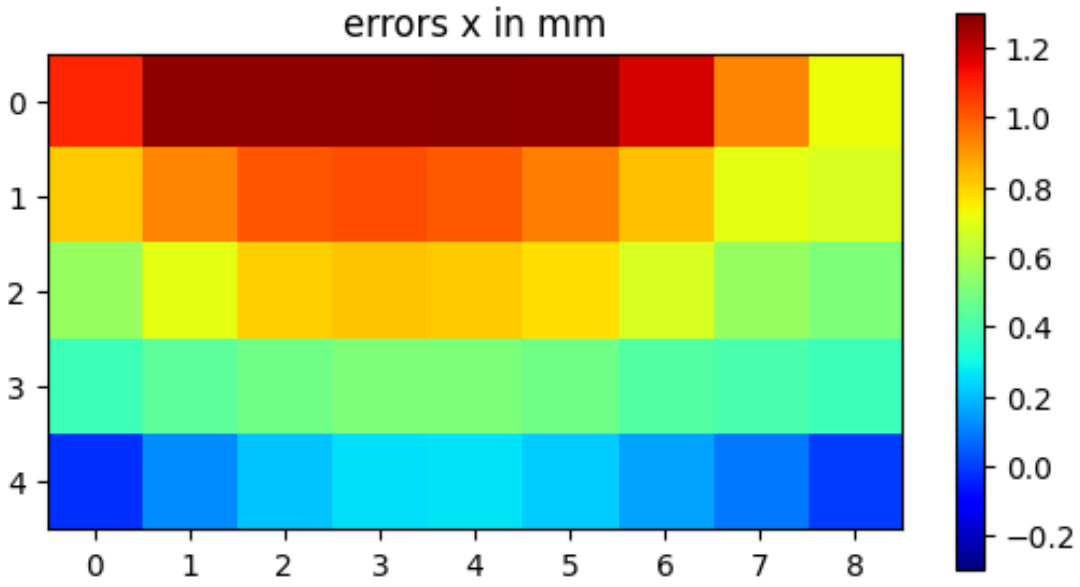

Figure 5.31: Robot position error along the x axis, measured with a mechanical comparator.

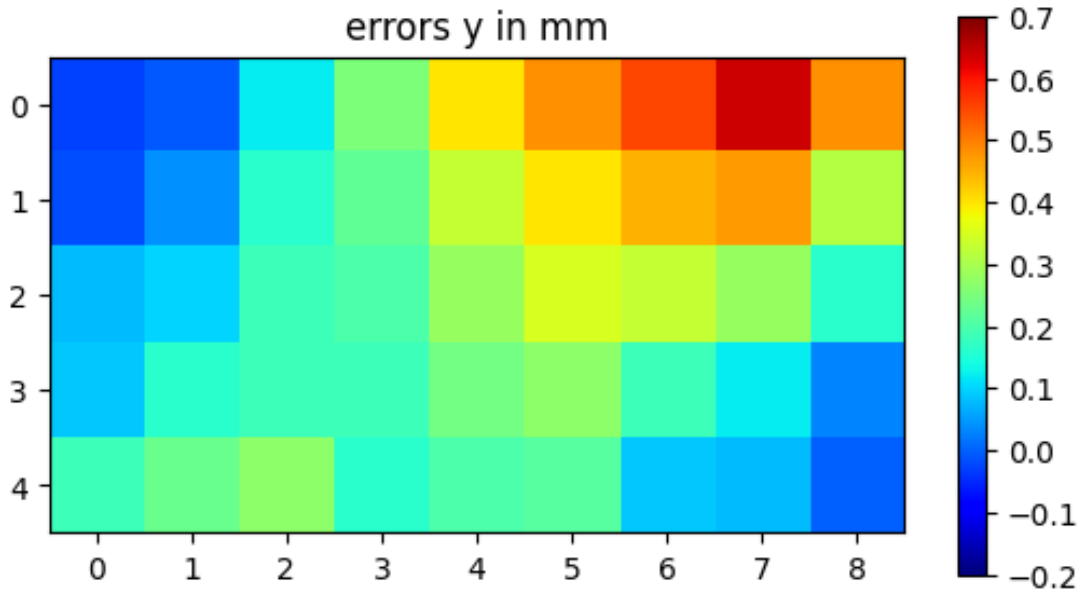

Figure 5.32: Robot position error along the y axis, measured with a mechanical comparator.

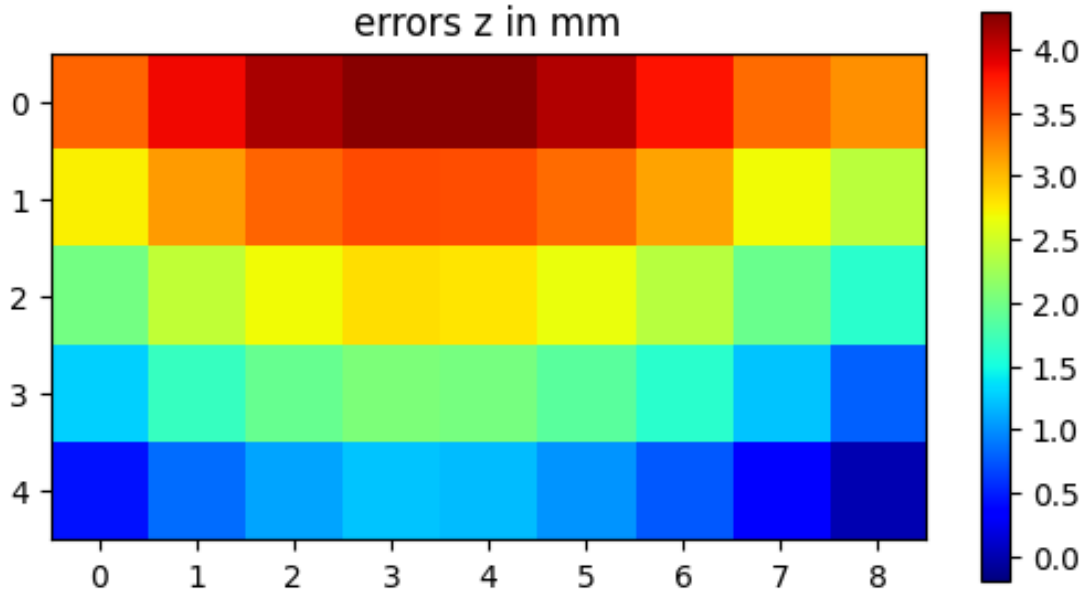

Figure 5.33: Robot position error along the z axis, measured with a mechanical comparator.

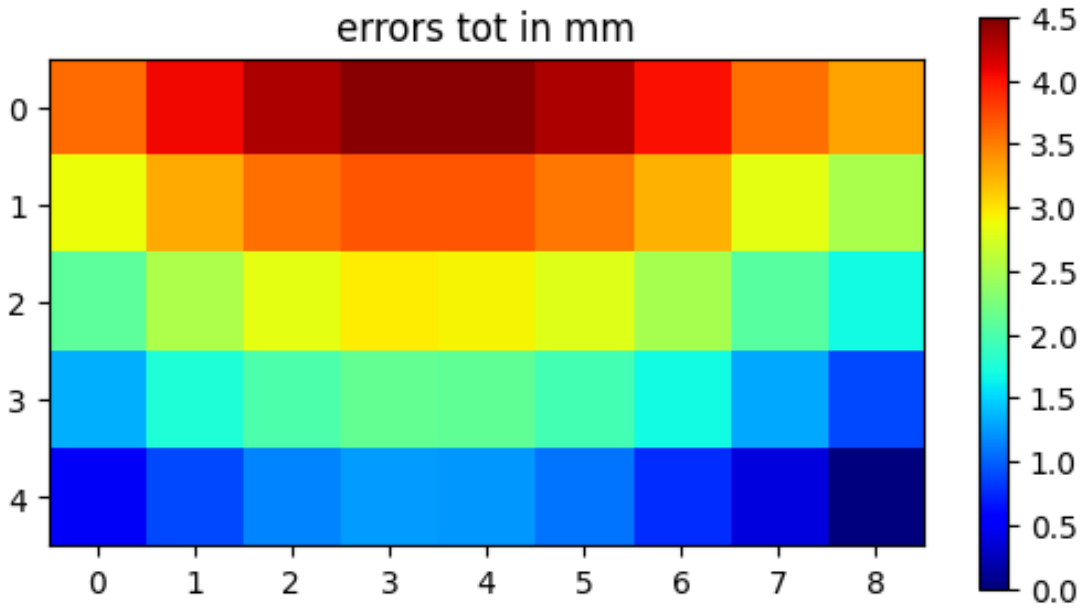

Figure 5.34: Robot total position error, measured with a mechanical comparator.

Figure 5.31, 5.32 and 5.33 show the position errors of the robot measured with the comparator along the x, y and z axis respectively. Figure 5.34 instead shows the total position errors of the robot measured with the comparator. It is possible to notice how even with the plate having a lower point resolution with respect to the calibration board and the plate being smaller the error graphs can be overlapped quite precisely to a portion of their counterparts computed on the LDPE calibration board. The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error (mm)	**Total**	**X**	**Y**	**Z**
Average	2.443	0.660	0.226	2.336
Std	1.197	0.364	0.151	1.142
Max	4.456	1.280	0.640	4.260

The error along the z axis was so big that the comparator would go out of scale so the requested position had to be adjusted to keep track of the error.

5.10 Comparator Correction by bilinear interpolation

Switching the LDPE calibration board and the camera for an mechanical comparator and a milled aluminum plate in the test phase allows to simulate a real application where the data gathering step is still performed with the camera but then the tool is swapped for something else. Since bilinear interpolation was the method that was performing better in the previous cases it is the first one to be put to test in this one too.

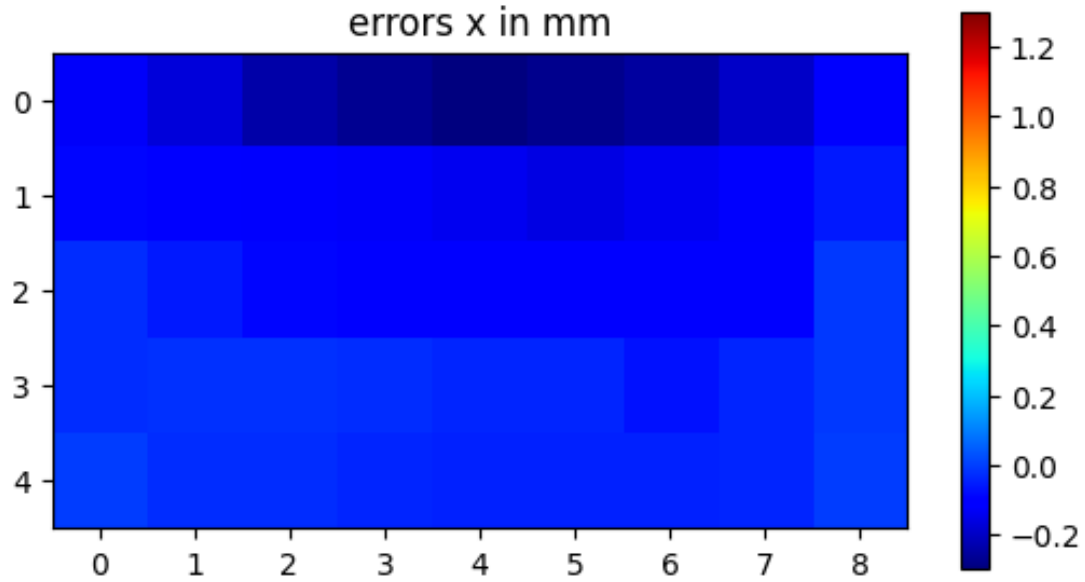

Figure 5.35: Robot residual position error along the x axis, measured with a mechanical comparator. Correction algorithm with bilinear interpolation.

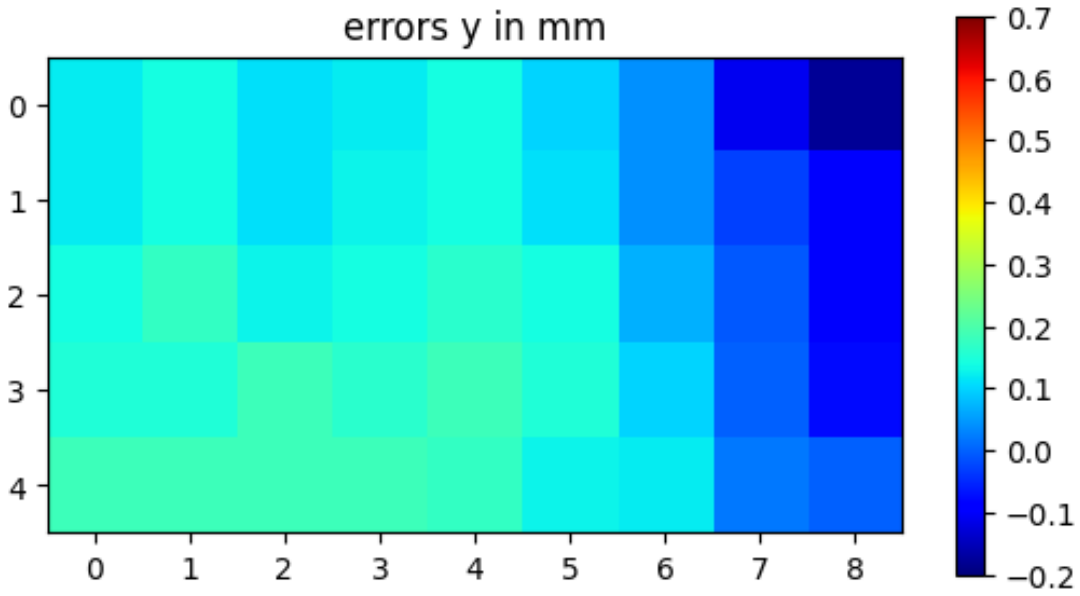

Figure 5.36: Robot residual position error along the y axis, measured with a mechanical comparator. Correction algorithm with bilinear interpolation.

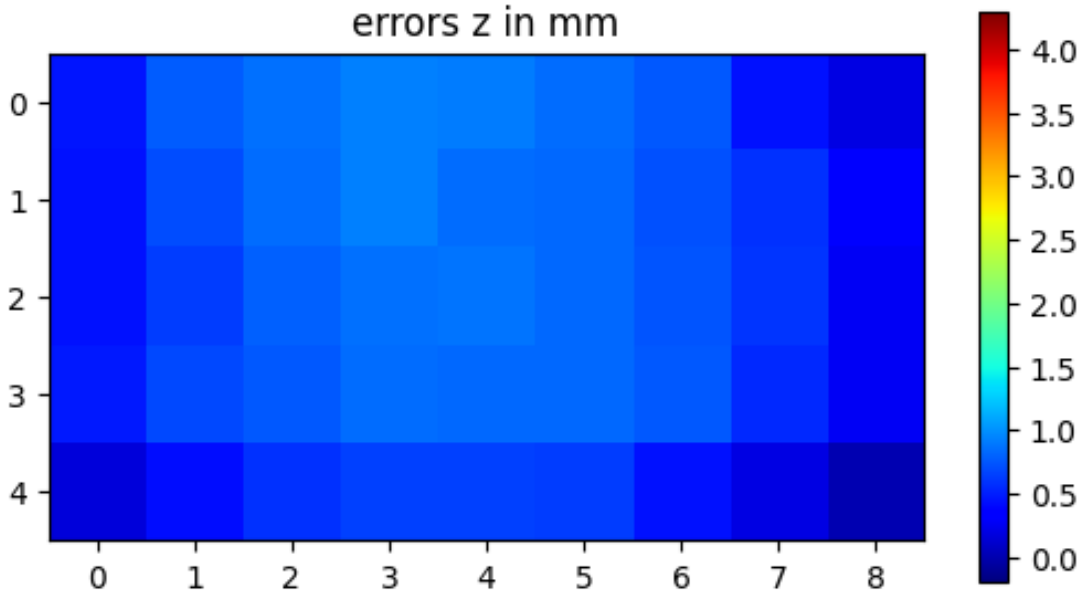

Figure 5.37: Robot residual position error along the z axis, measured with a mechanical comparator. Correction algorithm with bilinear interpolation.

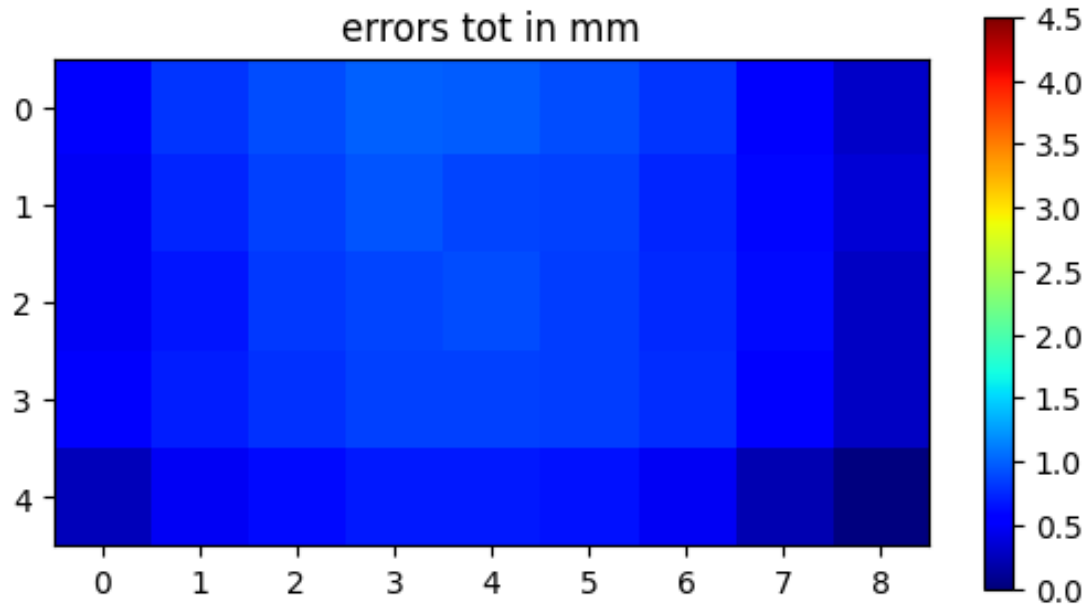

Figure 5.38: Robot total residual position error, measured with a mechanical comparator. Correction algorithm with bilinear interpolation.

The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error (mm)	**Total**	**X**	**Y**	**Z**
Average	0.651	0.096	0.118	0.626
Std	0.234	0.077	0.050	0.235
Max	0.985	0.300	0.180	0.940

The improvements provided by the bilinear interpolation algorithm are noticeable but, especially along the z axis the residual error is still quite high. This can be explained by the fact that due to the shape of the comparator pointer and the one of the plate the z displacement has to be measured on a different plane at a higher height from the ground with respect to the one chosen for the data gathering step with the camera. To be able to account for this situation it is necessary to include more height data into the correction algorithm. But bilinear interpolation cannot work at the same time on multiple planes without using something like cubic interpolation so the next obvious choice was to give neural networks another possibility. Since the neural network in Cartesian space was always outperformed by bilinear interpolation the next neural network used

will work in joint space. The idea is that ignoring the correction on the fourth joint may be of help when changing tool.

5.11 Comparator Correction by Neural network in joint space

First the neural network in joint space is tested against bilinear interpolation using the same data gathered with the camera on a single plane of height.

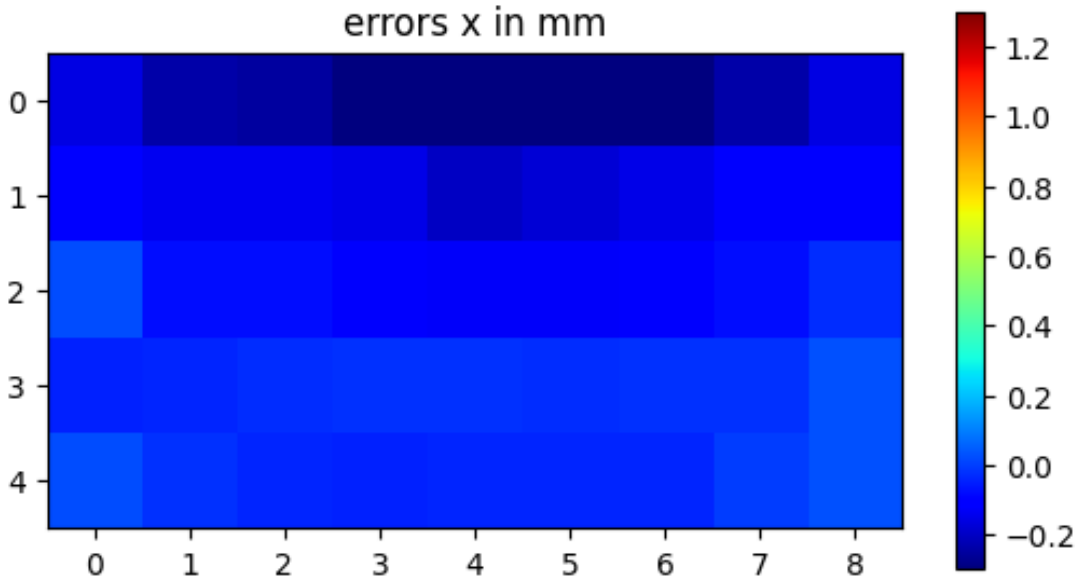

Figure 5.39: Robot residual position error along the x axis, measured with a mechanical comparator. Correction algorithm with neural network in joint space.

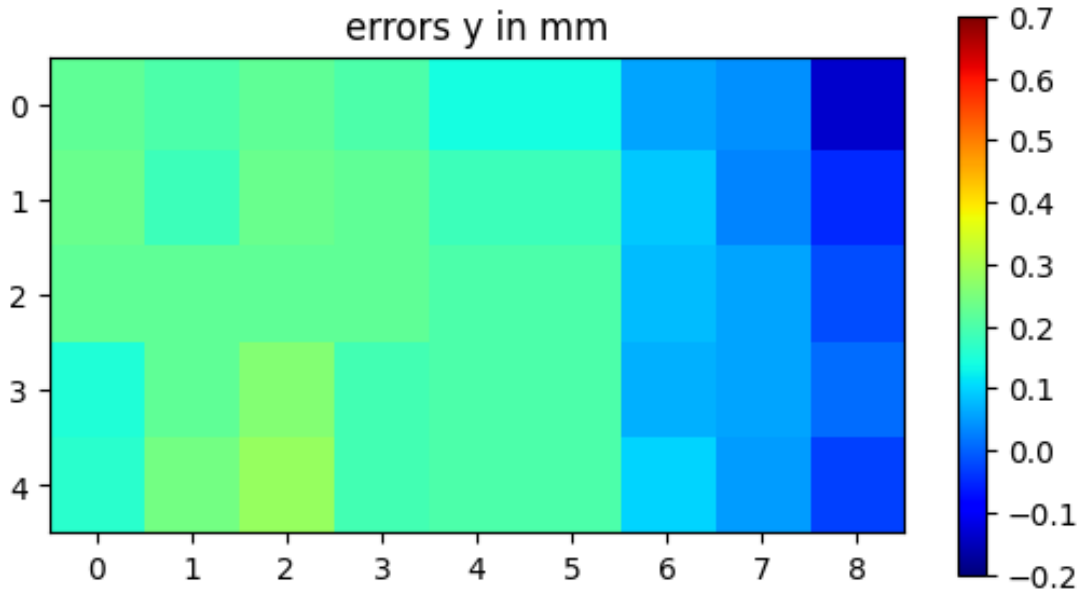

Figure 5.40: Robot residual position error along the y axis, measured with a mechanical comparator. Correction algorithm with neural network in joint space.

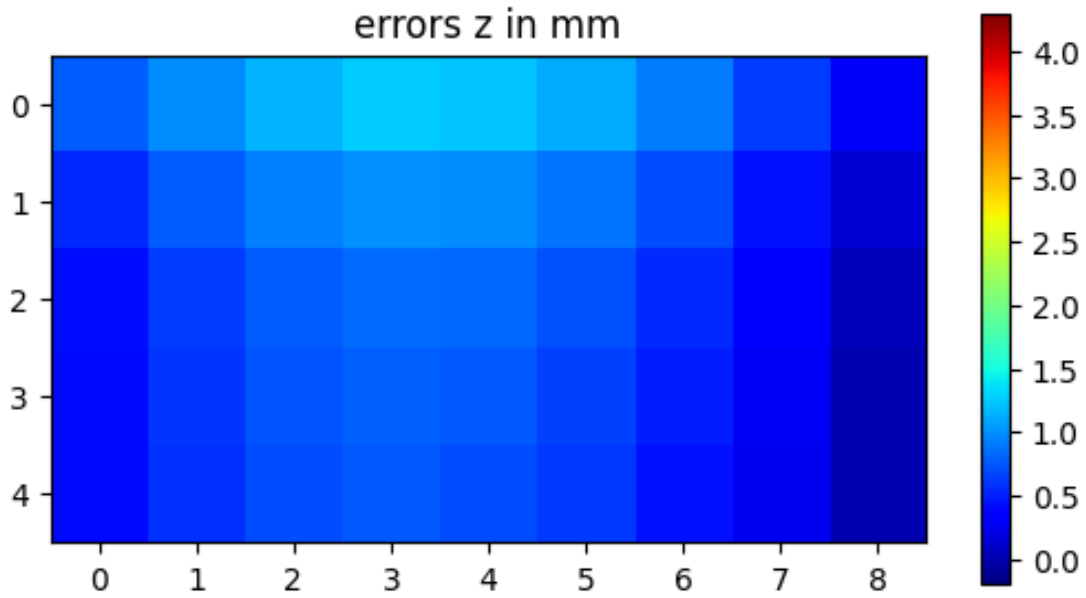

Figure 5.41: Robot residual position error along the z axis, measured with a mechanical comparator. Correction algorithm with neural network in joint space.

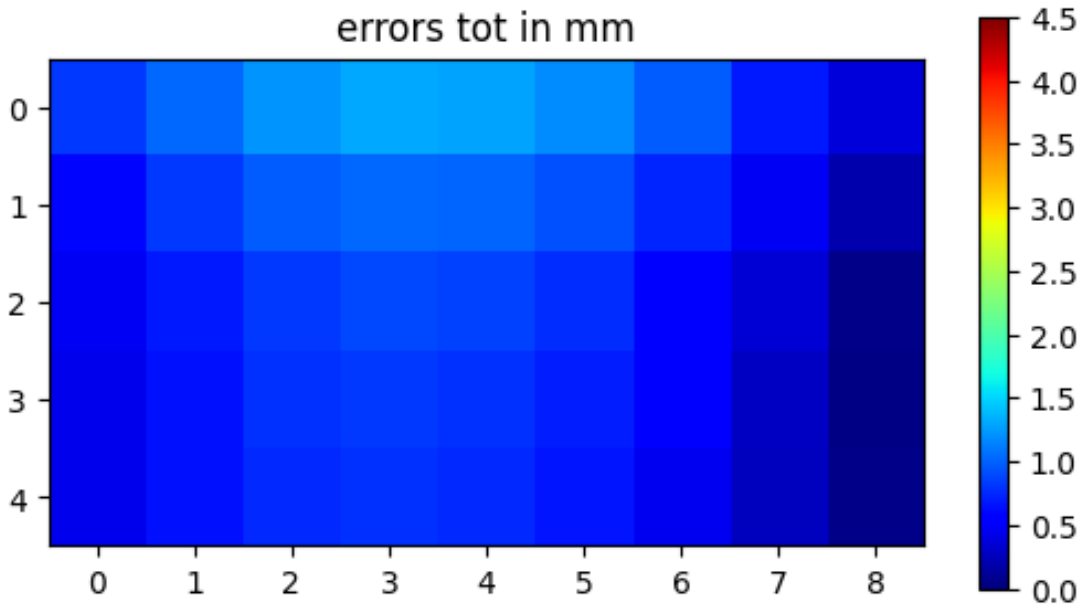

Figure 5.42: Robot total residual position, measured with a mechanical comparator. Correction algorithm with neural network in joint space.

Figure 5.39, 5.40 and 5.41 show the residual position errors of the robot measured with the comparator along the x, y and z axis respectively after being corrected with the neural network working in joint space. Figure 5.42 instead shows the total position errors of the robot measured with the comparator after being corrected with the neural network working in joint space.

The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error (mm)	**Total**	**X**	**Y**	**Z**
Average	0.682	0.108	0.155	0.647
Std	0.314	0.090	0.074	0.308
Max	1.310	0.320	0.280	1.260

With the dataset coming from just one height plane the neural network results are just slightly worse than the bilinear interpolation ones. Neural networks in joint space seem to perform considerably better than in Cartesian space when compared to the bilinear interpolation algorithm so it is worth investigating what happens when the network is trained on more data.

5.12 Comparator Correction by Neural network in joint space increased dataset

The first experiment is performed using a dataset with just one value of height, the same on which the bilinear interpolation is using. The second time another dataset is added to the train set of the network, this one is about one centimeter below the previous one. Finally a third dataset is used which also include data from one centimeter above the original dataset.

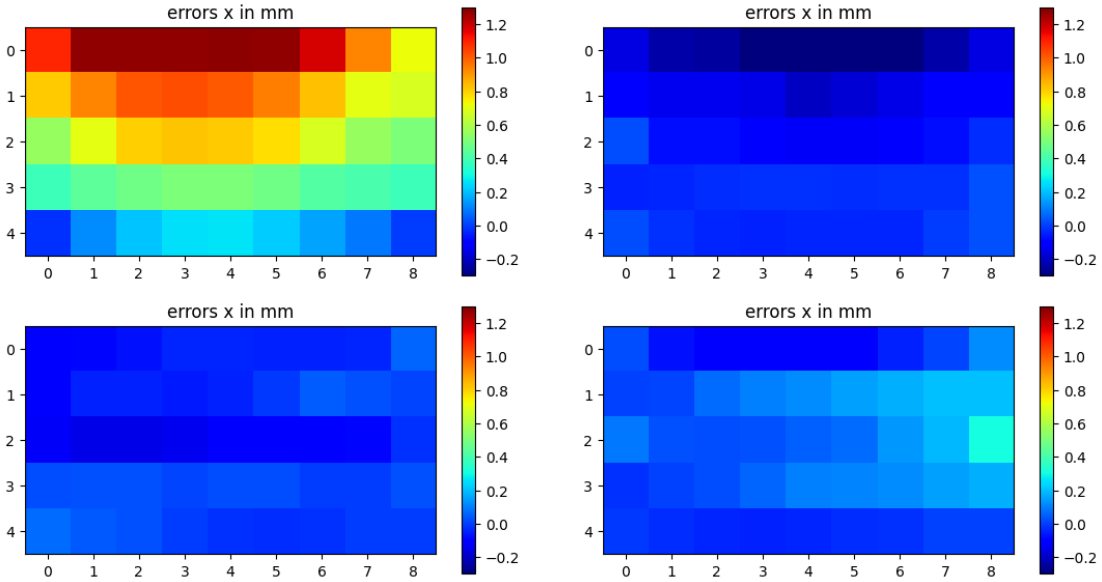

Figure 5.43: Improvement of error along the x axis with no correction and having a neural network with one, two or three sets of data during training.

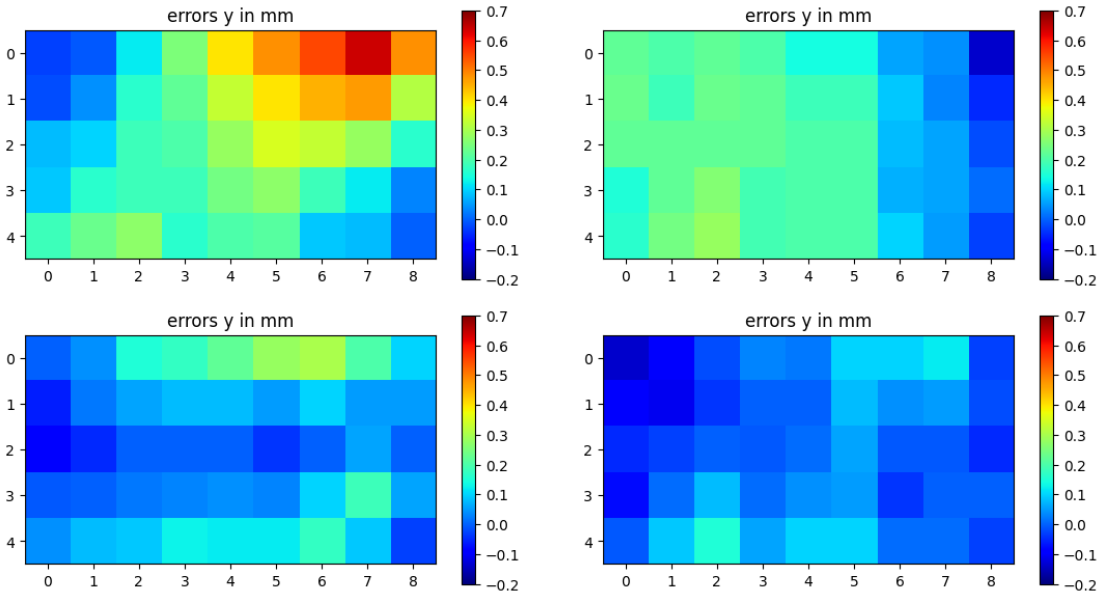

Figure 5.44: Improvement of error along the y axis with no correction and having a neural network with one, two or three sets of data during training.

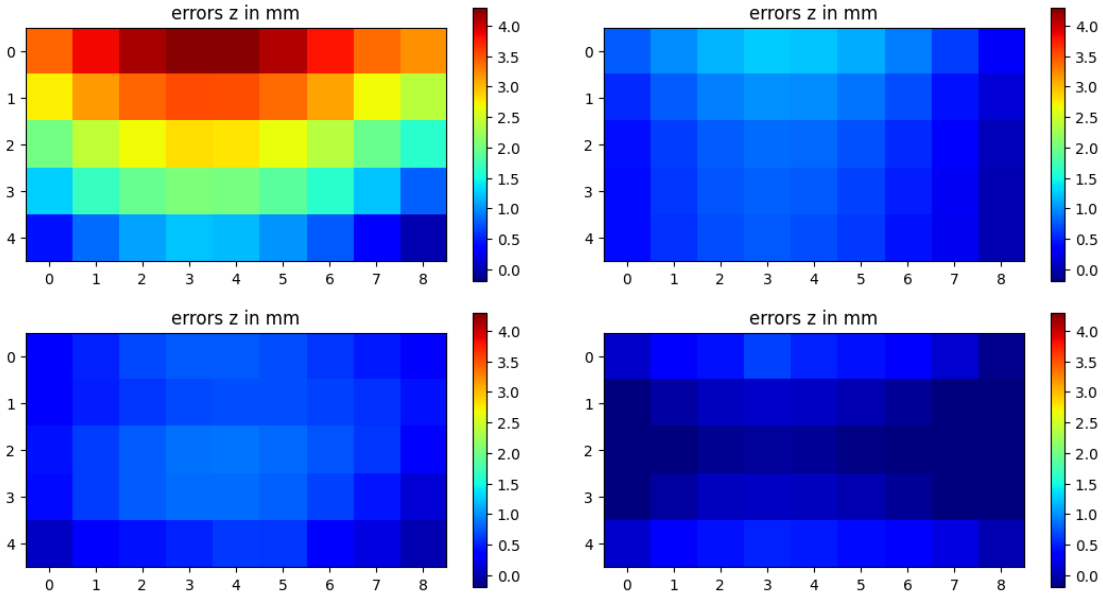

Figure 5.45: Improvement of error along the z axis with no correction and having a neural network with one, two or three sets of data during training.

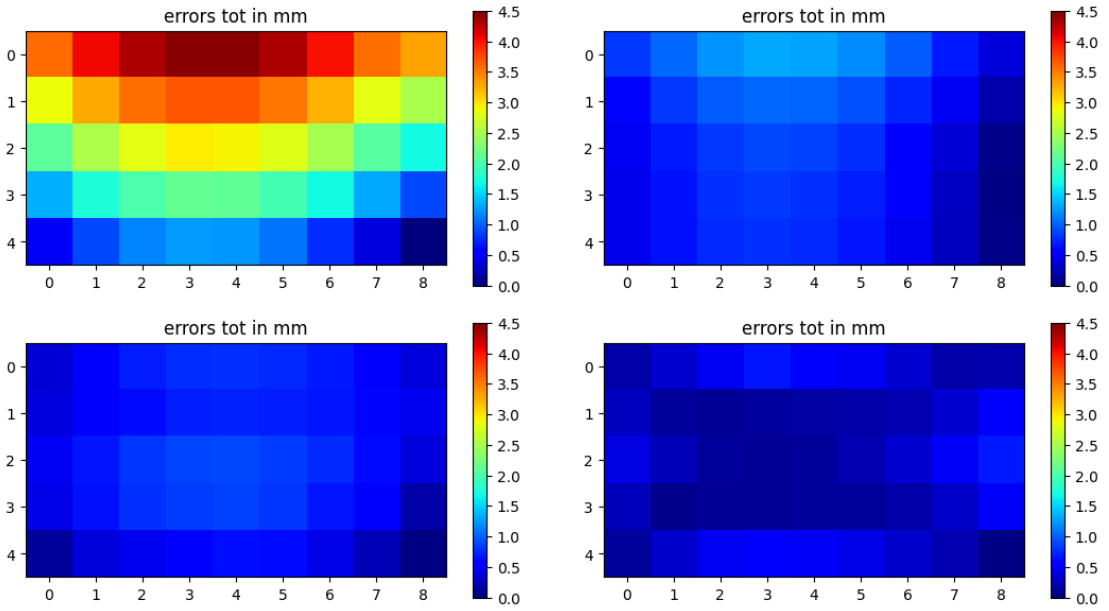

Figure 5.46: Total improvement of error with no correction and having a neural network with one, two or three sets of data during training.

Figure 5.43, 5.44 and 5.45 show a comparison of the position errors of the robot measured with the comparator along the x, y and z axis respectively. Figure 5.46 instead shows the total position errors of the robot measured with the comparator. In each figure the top left graph shows the error situation before applying any kind of correction, the graph at the top right shows the error situation after applying a correction to the robot position using a neural network in joint space with a dataset composed of a single pass over the calibration board. The graphs at the bottom left and right show the same but now the neural network has been trained with a dataset composed of two and three passes respectively over the calibration board at different heights. The following table reports the absolute values of the averages, standard deviations and maximum errors obtained during the experiment:

Error tot (mm)					
	No corr.	**Bilinear**	**Network 1**	**Network 2**	**Network 3**
Average	2.443	0.652	0.682	0.576	0.275
Std	1.197	0.234	0.314	0.208	0.168
Max	4.456	0.985	1.131	0.888	0.673
Error x (mm)					
	No corr.	**Bilinear**	**Network 1**	**Network 2**	**Network 3**
Average	0.660	0.096	0.109	0.052	0.081
Std	0.364	0.078	0.090	0.043	0.068
Max	1.280	0.300	0.320	0.150	0.300
Error y (mm)					
	No corr.	**Bilinear**	**Network 1**	**Network 2**	**Network 3**
Average	0.226	0.119	0.156	0.079	0.049
Std	0.151	0.051	0.075	0.073	0.042
Max	0.640	0.180	0.280	0.300	0.150
Error z (mm)					
	No corr.	**Bilinear**	**Network 1**	**Network 2**	**Network 3**
Average	2.336	0.626	0.647	0.561	0.240
Std	1.143	0.236	0.309	0.211	0.178
Max	4.260	0.940	1.260	0.880	0.650

From the results exposed on the table it is clear to see how much the correction algorithm using the neural network in joint space can improve using more data. Figure 5.47 shows how the average error of the robot decreases as the neural network working in joint space, in green, is trained on more data. In blue is the average robot repeatability to keep as a reference of the lower bound we can obtain. In red is shown the average error obtained with the bilinear interpolation algorithm which is still close to the repeatability line but definitely worse once the neural network gets enough data. This graph shows that once the accuracy needed for a specific application is know is up to the user to decide how much time to dedicate to the data gathering step to improve the performance of the algorithm.

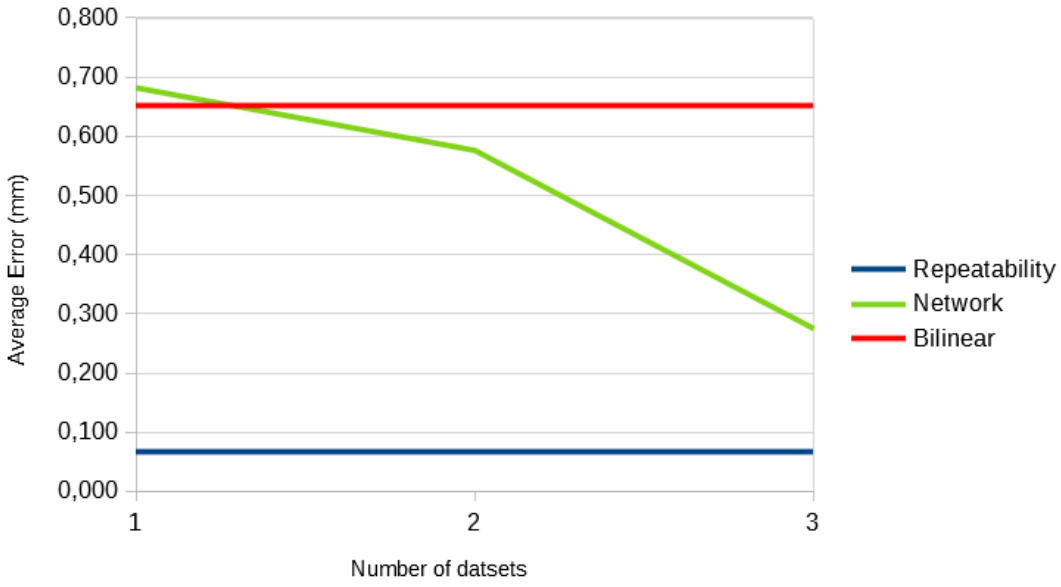

Figure 5.47: Comparison between error correction of bilinear interpolation and neural network in joint space. Repeatability added for reference.

6

Conclusions and future works

The aim of this thesis has been to develop an algorithm to improve robot accuracy by remapping the robot workspace through the use of computer vision and machine leaning techniques. The previous chapters walked through the steps taken to get there. The "Introduction" chapter presented the problem of robot calibration, underlined the causes of position errors that make it necessary and discussed the current state of the art in this matter. The following chapter "Framework" illustrated all the components used in this thesis, both hardware and software. The third chapter "Proposed approach" introduced the mathematical tools used to correct the robot position such as bilinear interpolation and neural networks. At this point the idea of using neural networks both in Cartesian and joint space was already introduced. Chapter four "Implementation" presented in a more practical way the techniques used to actually implement the correction algorithms, starting with the physical setup with a camera mounted on the end effector of the robot and a calibration board and following with the pipeline to gather information about the robot current repeatability and accuracy. The last sections of the chapter illustrates how to integrate the theoretical correction models already introduced in chapter three into a real pipeline that allows us to objectively quantify the improvements on the robot accuracy provided by the correction algorithms. It also discussed how to deal effectively with problems that emerged when moving from the controlled environment of the data gathering step to performing tests into a simulation of a real application. Chapter five "Results and discussion" presented the results

achieved by this thesis the performance of each approach considered to increase the robot accuracy was evaluated. It was interesting to see how while using the camera and calibration board the simpler, more deterministic method of bilinear interpolation based correction algorithm was performing slightly better than its neural network based counterpart. But when switching to the mechanical comparator and the milled aluminum plate additional uncertainties were introduced, such as a slight height change between the data gathering step and the test phase and the need to modify the reference frame associated with the board. In this situation the neural network based correction algorithm working in joint space was able to adapt more easily by integrating more data into the training set. After all the needed fine tuning the correction algorithm was able to successfully reduce the error and bring the robot accuracy closer to its repeatability.

In the future it would be interesting to see how much the bilinear interpolation method could be improved too. Since it does not require time for training it would be a more attractive solution to deploy in a real application but some way of integrating the height information should be included, cubic interpolation could be a solution worth exploring. Furthermore if the correction algorithm needs to be deployed quickly into an application it could be interesting to see how many points on the calibration board are really necessary to obtain satisfactory results. Less points would allow to save time in the data gathering step but at the price of having less data to use to predict the corrections, more complex interpolation techniques such as spline interpolation could be employed to compensate for that.

References

[1] Tianyan Chen et al. “Research of Calibration Method for Industrial Robot Based on Error Model of Position”. In: *Applied Sciences* 11 (2021).

[2] Xiangpeng Zhang et al. “Evaluation and prediction method of robot pose repeatability based on statistical distance”. In: *Mechanism and Machine Theory* 179 (2023).

[3] Vytautas Bucinskas et al. “Improving Industrial Robot Positioning Accuracy to the Microscale Using Machine Learning Method”. In: *Machines* 10.10 (2022).

[4] Branko Karan and Miomir Vukobratovi. “Calibration and accuracy of manipulation robot modelsAn overview”. In: vol. 29. 3. 1994, pp. 479–500.

[5] Sang Choi et al. “Lead-through robot teaching”. In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. 2013, pp. 1–4.

[6] H. Kochekali et al. “Factors affecting robot performance”. In: *Industrial Robot vol. 18 no. 1*. 1991.

[7] Zvi Roth, Benjamin Mooring, and Bahram Ravani. “An Overview of Robot Calibration”. In: vol. 3. Nov. 1987, pp. 377–385.

[8] Guanglong Du and Ping Zhang. “Online robot calibration based on vision measurement”. In: vol. 29. 6. 2013, pp. 484–492.

[9] P.S. Shiakolas, K.L. Conrad, and T.C. Yih. “On the Accuracy, Repeatability, and Degree of Influence of Kinematics Parameters for Industrial Robots”. In: vol. 22. 4. Taylor and Francis, 2002, pp. 245–254.

[10] Chandra Sekhar Gatla et al. “An Automated Method to Calibrate Industrial Robots Using a Virtual Closed Kinematic Chain”. In: vol. 23. 6. 2007, pp. 1105–1116.

[11] Yan Meng and Hanqi Zhuang. “Autonomous robot calibration using vision technology”. In: vol. 23. 4. 2007, pp. 436–446.

[12] B. Mooring, Zvi Roth, and M. Driels. “Fundamentals of Manipulator Calibration”. In: Jan. 1991.

[13] Hanqi Zhuang and Zvi S. Roth. “Camera-Aided Robot Calibration”. In: 1996.

[14] Ming Z. Huang and Oren Masory. “A Simple Method of Accuracy Enhancement for Industrial Manipulators”. In: vol. 8. 1993, pp. 114–122.

[15] D. H. Kim, K. H. Cook, and J. H. Oh. “Identification and Compensation of a Robot Kinematic Parameter for Positioning Accuracy Improvement”. In: vol. 9. 1. Cambridge University Press, 1991, pp. 99–105.

[16] Hamid Majidi Balanji, Ali Emre Turgut, and Lutfi Taner Tunc. “A novel vision-based calibration framework for industrial robotic manipulators”. In: vol. 73. 2022, p. 102248.

[17] Dali Wang, Ying Bai, and Jiying Zhao. “Robot manipulator calibration using neural network and a camera-based measurement system”. In: vol. 34. 1. 2012, pp. 105–121.

[18] Xiaolin Zhong, John Lewis, and Francis L. N-Nagy. “Inverse robot calibration using artificial neural networks”. In: vol. 9. 1. 1996, pp. 83–93.

[19] *Robo Ware*. `https://roboware.it/`.

[20] *Dobot Robotics*. `https://www.dobot-robots.com/`.

[21] Marek Paczek and ukasz Piszczek. “Testing of an industrial robots accuracy and repeatability in off and online environment”. In: *Eksploatacja i Niezawodnosc - Maintenance and Reliability* 20 (June 2018), pp. 455–464.

[22] *Hikrobot*. `https://www.hikrobotics.com/en/machinevision/visionproduct?typeId=40&id=49&pageNumber=1&pageSize=50`.

[23] *Calib.io Camera Calibration*. `https://calib.io/`.

[24] Dali Wang and Ying Bai. “Improving Position Accuracy of Robot Manipulators Using Neural Networks”. In: *2005 IEEE Instrumentationand Measurement Technology Conference Proceedings*. Vol. 2. 2005, pp. 1524–1526.

[25] A. Doria, F. Angrili, and S. De Marchi. “Inverse kinematics robot calibration by spline functions”. In: vol. 17. 9. 1993, pp. 492–498.

[26] Edward Red, Xuguang Wang, and Ed Turner. "Calibration control of robot vertical assembly". In: vol. 6. 3. 1989, pp. 269–299.

[27] Earl J. Kirkland. "Bilinear Interpolation". In: *Advanced Computing in Electron Microscopy*. Boston, MA: Springer US, 2010, pp. 261–263.

[28] Tien-fu Lu and Grier C.I. Lin. "An on-line relative position and orientation error calibration methodology for workcell robot operations". In: vol. 13. 2. 1997, pp. 89–99.

[29] Yu-chen Wu and Jun-wen Feng. "Development and Application of Artificial Neural Network". In: (Sept. 2018).

[30] *Towards Data Science, The Concept of Artificial Neurons (Perceptrons) in Neural Networks*. https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc.

[31] *UpGrad, Neural Network: Architecture, Components and Top Algorithms*. https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/.

[32] Iqbal Sarker. "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions". In: *SN Computer Science* 2 (Aug. 2021).

[33] *NN-SVG*. https://alexlenail.me/NN-SVG/.

[34] Alberto Omodei, Giovanni Legnani, and Riccardo Adamini. "Three methodologies for the calibration of industrial manipulators: Experimental results on a SCARA robot". In: vol. 17. 6. 2000, pp. 291–307.

[35] Hanqi Huang and Z.S. Roth. "On vision-based robot calibration". In: *Conference Record Southcon*. 1994, pp. 104–109.

[36] T. A. Clarke and J. G. Fryer. "The Development of Camera Calibration Methods and Models". In: *The Photogrammetric Record* 16.91 (1998), pp. 51–66.

[37] *Wolfram MathWorld, Rotation Matrix*. https://mathworld.wolfram.com/RotationMatrix.html.

[38] *MathWorks, Quaternion frame rotation*. https://it.mathworks.com/help/nav/ref/quaternion.rotateframe.html.

[39] Chiara Troiani et al. “2-Point-based Outlier Rejection for Camera-IMU Systems with applications to Micro Aerial Vehicles”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (May 2014).

[40] *Project Chrono, Coordinate transformations*. `https://api.projectchrono.org/coordinate_transformations.html`.

[41] Shibin Yin et al. “A Vision-Based Self-Calibration Method for Robotic Visual Inspection Systems”. In: vol. 13. 12. 2013, pp. 16565–16582.

[42] H. Zhuang, L. Wang, and Z.S. Roth. “Simultaneous calibration of a robot and a hand-mounted camera”. In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. 1993, 149–154 vol.2.

[43] R.K. Lenz and R.Y. Tsai. “Calibrating a Cartesian robot with eye-on-hand configuration independent of eye-to-hand relationship”. In: *Proceedings CVPR '88: The Computer Society Conference on Computer Vision and Pattern Recognition*. 1988, pp. 67–75.

[44] Stefano Chiaverini. “Redundant Robots”. In: *Encyclopedia of Systems and Control*. Ed. by John Baillieul and Tariq Samad. London: Springer London, 2019, pp. 1–10.

[45] Alberto Traslosheros et al. “An Inexpensive Method for Kinematic Calibration of a Parallel Robot by Using One Hand-Held Camera as Main Sensor”. In: vol. 13. 8. 2013, pp. 9941–9965.

[46] Stefano Chiaverini. “The 2D and 3D coordinate systems”. In: *Unreal Engine Physics Essentials*. Ed. by Emperore Katax and Sherry Devin. Packt Publishing, 2015.

[47] *Embibe, Draw a diagram of pinhole camera and labelled it properly*. `https://www.embibe.com/questions/Draw-a-diagram-of-pinhole-camera-and-labelled-it-properly.-/EM2429180`.

[48] Boris Martin. In: *Computer vision based robot calibration and control*. 1990.

[49] M. Young. “Pinhole Optics”. In: *Appl. Opt.* 10.12 (1971), pp. 2763–2767.

[50] R.I. Hartley and A. Zisserman. *Multi-View Geometry in Computer Vision*. Vol. 2. Jan. 2003.

[51] Haiyin Sun. “Thin lens equation for a real laser beam with weak lens aperture truncation”. In: *Optical Engineering* 37.11 (1998), pp. 2906–2913.

[52] Irene Martini. "Copper for particle accelerators: electron stimulated desorption and study of hydrogen content measurement by laser ablation (MSc Thesis)". PhD thesis. Apr. 2012.

[53] Jelena Kocic, Aleksej Makarov, and Sasa Vujic. "Multiple sensors' lenslets for secure document scanners". In: (Mar. 2011).

[54] *Expert Photography, Camera Lens Guide (Parts, Functions and Types Explained)*. `https://expertphotography.com/camera-lenses-guide/`.

[55] B. W. Mooring and T. J. Pack. "Aspects of robot repeatability". In: *Robotica* 5.3 (1987), pp. 223–230.

[56] Michal Vocetka et al. "Influence of Drift on Robot Repeatability and Its Compensation". In: *Applied Sciences* 11.22 (2021).

[57] *GisGeography, What is Bilinear Interpolation?* `https://gisgeography.com/bilinear-interpolation-resampling/`.

[58] Michele Bonollo et al. "Model risk and techniques for controlling market parameters. The experience in Banco Popolare". In: *Universita di Modena e Reggio Emilia, Facoltà di Economia "Marco Biagi", Centro Studi di Banca e Finanza (CEFIN) (Center for Studies in Banking and Finance)* (Jan. 2007).

[59] Yin Bai and Hanqi Zhuang. "On the comparison of model-based and modeless robotic calibration based on the fuzzy interpolation technique". In: *IEEE Conference on Robotics, Automation and Mechatronics, 2004.* Vol. 2. 2004, 892–897 vol.2.

[60] Maad Mijwil, Adam Esen, and Aysar Alsaadi. "Overview of Neural Networks". In: 1 (Apr. 2019), p. 2.

[61] C.-H. Menq and J.-H. Borm. "Statistical measure and characterization of robot errors". In: *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. 1988, 926–931 vol.2.

[62] De Rosa Andrea. "Metodo di Edge-Detection basato sul calcolo di aree parziali". PhD thesis. University of Bologna, 2018.