POLITECNICO DI TORINO

Dipartimento di Ingegneria Meccanica ed Aereospaziale
Master's Degree in Mechanical Engineering

Master's Thesis

Absolute Calibration of an Industrial Manipulator through a Compensation Method based on a Neural Network

Supervisor:
Prof. Terenziano Raparelli

Candidate:
Stefano Floridan

July 2019

Alla mia famiglia

Acknowledgments

I would like to express my gratitude to all people of COMAU Italia for kindly allowing me to work with them on this final Thesis.
Thank you all for your collaboration and patience while guiding me through all your technologies and knowhow. This project couldn't have been developed without the precious information that I was allowed to gather during my stay in your facilities.

Furthermore, I would like to thank my supervisor Prof. Terenziano Raparelli for believing in this project and for being always eager to help during the entire process.

Table of Contents

INTRODUCTION **8**

1) CALIBRATION OF A ROBOTIC ARM **10**

1.1 GEOMETRICAL ERRORS 10
1.2 NON-GEOMETRICAL ERRORS: 10
1.3 MODEL BASED ERROR COMPENSATION 11
1.4 MODEL-FREE ERROR COMPENSATION 12

2) ARTIFICIAL NEURAL NETWORKS (A.N.N.) **15**

2.1 NEURAL NETWORK STRUCTURES 17
2.2 ACTIVATION FUNCTIONS 18
2.3 CONNECTIONS 18
2.4 TRAINING 19
2.4.1 Univariate Linear Regression *19*
2.4.2 Non Linear Models *21*
2.4.3 Multivariate Linear Regression *22*

3) ERROR COMPENSATION PRINCIPLE WITH EXTREME MACHINE LEARNING **23**

3.1 GENERAL ELM COMPENSATION 3 INPUTS 3 OUTPUTS 24
3.2 ELM TRAINING 24
3.3 COMPENSATION PROCESS 26

4) COMAU'S CALIBRATION STATE OF THE ART **28**

4.1 KRYPTON 28
4.2 MEASUREMENT SYSTEM 28
4.2.1 K600 Camera *29*
4.2.2 LEDs layout *30*
4.3 CALIBRATION PROCESS WITH KRYPTON 31

5) SIMULATION **32**

5.1 THE ROBOT NS 12-185 32
5.1.1 Technical Specifics NS12-185 Robotic Arm *33*
5.1.2 The C5G Control Unit *35*
5.1.3 Cartesian Coordinates *36*
5.1.4 Joint Coordinates *36*

..... **36**

5.1.5 Teach Pendant "TP5" *37*
5.2 DATABASE GATHERING 38
5.2.1 Input creation: *38*
5.2.2 Reachability Verification of the Theoretical points *40*
5.2.3 SIMULATED SPATIAL ERROR CREATION 42
5.3 ANALYSIS OF THE ALGORITHM 43
5.3.1 Training Phase: *43*
5.3.2 Testing Phase: *44*
5.4 DATASETS CREATION 45
5.4.1 Layout Dataset X coordinate *45*
5.4.2 Layout Dataset Y coordinate *46*
5.4.3 Layout Dataset Z coordinate *47*
5.4.4 Layout Dataset Absolute Error *48*

6) RESULTS AND DISCUSSION **49**

6.1 SIMULATION 1: ΔX PREVISION 50
6.2 SIMULATION 2: ΔY PREVISION 51
6.3 SIMULATION 3: ΔZ PREVISION 52
6.4 SIMULATION 4: ABSOLUTE ERROR PREVISION 53

7) CONCLUSIONS AND FUTURE WORK 54
APPENDIX A 55
APPENDIX B 57
REFERENCES 62

Table of Figures

FIGURE 1, ACCURACY VS REPEATABILITY 8
FIGURE 2: COMPARISON BETWEEN NEURAL NETWORKS AND HUMAN BRAIN CELLS 16
FIGURE 3: MATHEMATICAL STRUCTURE OF A NODE 17
FIGURE 4: PERCEPTRON THRESHOLD (A) AND A SIGMOID THRESHOLD (B) 18
FIGURE 5: EXAMPLE OF A LINEAR REGRESSION 20
FIGURE 6: 3D PLOT OF THE LOSS FUNCTION 21
FIGURE 7: SINGLE FEEDFORWARD MULTI INPUT- MULTI OUTPUT NEURAL NETWORK SCHEME 23
FIGURE 8: NEURAL NETWORK TRAINING FLOW CHART 26
FIGURE 9: CALIBRATION FLOW CHART WITH ERROR COMPENSATION THROUGH THE NEURAL NETWORK 27
FIGURE 10: PHOTO OF THE REAL K600 CAMERA 29
FIGURE 11: PYRAMIDAL VOLUJME OF THE CAMERA 29
FIGURE 12: FIELD OF VIEW OF THE CAMERA WITH ALL THE THREE DIFFERENT ACCURACIES ZONES 29
FIGURE 13: REAL PHOTO OF A ROBOTIC ARM SET UP FOR KRYPTON 30
FIGURE 14: LED SETUP ON ROBOTIC FLANGE 30
FIGURE 15: C5G CONTROL UNIT 35
FIGURE 16: TEACH PENDANT TP5 37
FIGURE 17: ROBOSIM INTERFACE WITH 3D BLENDER ON THE LEFT AND THE VIRTUAL TP5 ON THE RIGHT 40
FIGURE 18: ROBOSIM PRINT OF THE CARTESIAN AND JOINT COORDINATES OF EACH TESTED SPATIAL POINT 41
FIGURE 19: ROBOSIM TRAJECTORY TRACE OF ALL THE 1000 POSITIONS UNDERGONE IN THE DATABASE CREATION 41
FIGURE 20: X COORDINATE REALATIVE ERROR PREVISION 50
FIGURE 21: Y COORDINATE RELATIVE ERROR PREVISION 51
FIGURE 22: Z COORDINATE REALTIVE ERROR PREVISION 52
FIGURE 23: ABSOLUTE ERROR PREVISION 53

Abstract

Now days increasing the accurate positioning of a robotic arm's end-effector is a crucial point in the field of Industrial Robotics.

A more accurate manipulator allows the business to effectively enhance all aspects of the product quality, increasing its competitiveness in the markets.

This project goal is to verify, through an 'ad hoc' simulation, if a spatial error compensation method based on a Neural Network may improve the state of the art in robotics calibration.
The Network will be trained with a novel learning pattern defined as 'Extreme Machine Learning' that should guarantee a successful output without pursuing an iterative procedure. The lack of iterations diminishes computational complexity and time.

The Network's aim is to predict with 4 different simulations the error that a specific 6 degrees of freedom open kinematic robotic arm displays during its operation. Once these errors are known, you may improve the spatial positioning of the arm through a simple compensation of the controller's inputs.

Introduction

A robot is a mechanical system in which geometrical inaccuracies and non-geometrical inaccuracies work together to create an inaccuracy of the end effector's absolute position and orientation.

The effectiveness of an industrial manipulator during its operations is dictated by both metrological and measurable parameters. These parameters will have a direct impact on the quality of the task performed.
The two main measurable characteristics are accuracy and repeatability. The precision of a robot measures the ability of the industrial manipulator to repeat the same task over time. Accuracy, on the other hand, measures the difference (i.e. the error) between the task theoretically required and that actually achieved by the robot.
Therefore, repeatability always does the same task over and over again, while accuracy strikes your target every time.

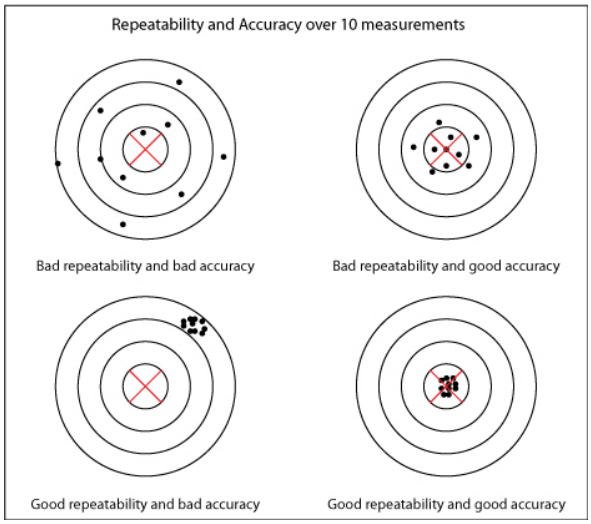


figure 1, Accuracy vs Repeatability

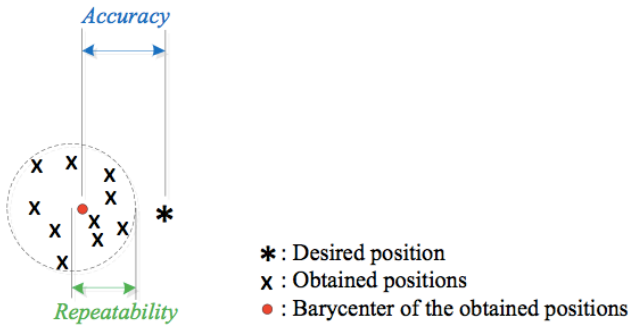
Absolute position accuracy is the robot's ability to reach a specific programmed position with minimal error.
The term 'absolute' defines that position accuracy is evaluated against the single work reference frame (or the universal one). The framework used to evaluate static accuracy of robot's operation requires that position measurements are made at the end of the end effector's movement (regardless of the path followed to reach the programmed position.

Robot accuracy may be defined in 2 different ways:

- Geometrically it may be defined as the distance between the average position of all points reached in spatial coordinates, defined as 'centroid', and the position theoretically desired for that task.

- Mathematically, absolute accuracy is the compilation of compound errors for each of the Cartesian position errors x, y, z.

Finally, the accuracy of the robot's position for a specific workspace can be described as the maximum compound error available for several uniformly distributed positions within the predetermined workspace or reference frame.



Repeatability is the measure of its precision in repositioning itself in a previously reached point (with the same initial conditions).
Geometrically it can be defined as the radius of the smallest sphere encompassing all the different positions reached by the end effector for a single desired position. The norm afferent to the repeatability is the NF EN ISO9283.

It is therefore of crucial importance to enhance as much as you possibly can the repeatability and accuracy in a robot. In this way the manipulator will be able to repeat effectively the programmed movement always hitting the target.

Generally speaking Industrial robots present a good repeatability while their accuracy is usually worse.

1) Calibration of a Robotic Arm

Calibration is a methodology to improve the robot accuracy without mechanical means working exclusively on its controller.

It has been proven that a proper calibration most surely improves the robot accuracy up to a value close to the robot repeatability.

It is of crucial importance to understand and stress out the causes of spatial inaccuracy. The sources of poor absolute accuracy are made up by two macro categories: The “Geometrical Errors” and the “Non-Geometrical Errors”.

1.1 Geometrical Errors

Positioning errors caused by Manufacturing mistakes, base misalignment, poor maintenance and poor assembly, mounting procedure, non-parallelism between axis are classified as the Geometrical errors and accountable for the 90% of the Absolute accuracy loss.

1.2 Non-Geometrical Errors:

On the other hand, errors produced by geometrically non related factors as gear backlash, thermal dilatation, inertial forces, payload, vibrations, servo errors and so on, are grouped under the macro-set of NON-Geometrical errors and account for the remaining 10% of the absolute accuracy loss.

A procedure to improve the robot accuracy (which for industrial manipulators it some-times gets up to a couple of [mm]) is made up of 2 main steps:

1. Measurement of the end effector position and orientation error for a predefined set of gripper poses in the workspace;

2. the development of a mathematical model or technique to firstly predict and secondly compensate for the measured errors.

During the past 30 years 2 main compensation methods have been developed:

- model-based compensation
- model-free compensation

1.3 Model Based Error Compensation

A widely used approach for kinematic calibration is parametric calibration which is based on the development of a parametric model of the robot and the identification of the parameters of the machine to reproduce the kinematic behavior of the real robot in the most reliable way possible. To reduce the complexity of the model, errors due to play, elasticity, thermal dilatations are often overlooked; on the other hand, for most industrial manipulating robots geometric errors are the main cause of inaccuracy and parametric calibration allows to obtain sufficient results, improving the accuracy of the robot to bring it to values close to repeatability

In the model-based compensation framework is made up as follows :

1. Construct a kinematic model
2. Errors associated with the kinematic parameters are identified based on this model.

The most suitable model to find the errors of the kinematic parameters is the Denavit-Hartenberg (DH) model of the robot.

A Denavit–Hartenberg (DH) model is a convenient method for determining the deviation of kinematic parameters. When adjacent axes are parallel, the solution will be singular. To solve this problem, a modified Denavit–Hartenberg (MDH) model is proposed. Compared with a DH model, a MDH model adds an additional parameter representing the rotation around the y axis.

In model-based compensation a further complexity is introduced by the parameter identification. To solve this issue some algorithms for parameters identification are exploited. The most popular ones are the Least squares Method, Non-Linear Optimization procedure, Iterative Linearization, extended Kalman Filter and Levenberg-Marquardt.

It is crucial to stress out that a model-based method generally ignores all the non-geometrical errors. This choice is made with a very specific goal: obtaining a solvable model that presents a bearable computational complexity. Therefore only 90% of the total error causes are accounted for, leading inevitably to a non-ideal compensation accuracy.
A further issue is that you will need to build a kinematic model for each Machine thus lacking Universality traits that are desirable for a calibration method.

In this paper this kind of approach has not been pursued, therefore you may easily deepen your knowledge in this matter consulting the scientific literature.

1.4 Model-Free Error Compensation

In the past 10 years a new approach to robotic arm absolute calibration was pursued, a model-free based method.
This kind of approach assumes that position errors are linked to robot joint values or robot end effector's positions.
Based on this fresh assumption, many different numerical approaches have been tried out. For instance: Fourier polynomial, kriging, inverse distance weighting, and artificial neural network (ANN), have been utilized to predict position errors.
All of these numerical methods may be exploited to solve a linear regression problem.

Fourier polynomial presents two undesirable traits: a high computational complexity that leads to a poor absolute accuracy, therefore it is limited for our application.

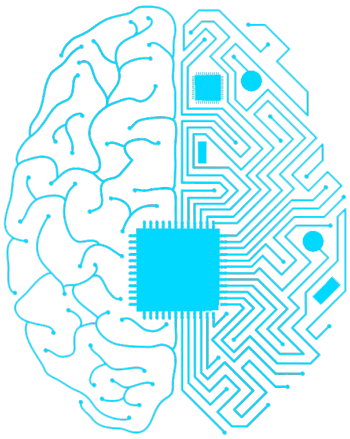
Inverse distance weighting presents a good final accuracy and bearable computational complexity but its application is limited to a small range of robot movement in its work space.

Finally, the kriging method is really complex, and it is challenging for field engineers working in a firm to master.

In comparison with the previous methods, the ANN (Artificial Neural Network) method, which presents a high learning ability and high adaptability, can constantly adjust the weight of the associated node so that the output can approach the desired results.
Therefore, the ANN method has been chosen as the state of art technology to provide higher precision for industrial robots.

In this paper we will use a SLFN (Single Layer Feedforward Network) to forecast the spatial absolute positional error of a 6 DOF COMAU robotic arm

ARTIFICIAL INTELLIGENCE (A.I.)



Now days a hot topic in scientific society is Artificial Intelligence (A.I.).
There is not a unique definition of A.I. Throughout the last 3 decades several definitions have been formulated. We collected the 8 major definitions in 4 boxes. These definitions are laid out along 2 dimentions, horiziontal and vertical, which define some classification parameters for each one. To be more clear:

- **VERTICAL BLOCKS**

The definitions on the upper blocks concerne the thought process and reasoning, whereas the two lower ones adress behaviour.

- **HORIZONTAL BLOCKS**

The definitions on the two left blocks measure success in terms of fidelity to human performance, whereas the ones on the right measure against an ideal performance measure , called rationality.
A system is adressed as rational if it makes the right choice given what he knows.

Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Hystorically all four approaches to AI have been followed, each by different people with different methods and solutions.

A human centered approach must be more an empirical science, involving observations and formulating hypothesis

A rational approach on the other hand involves a combo of mathematics and engineering.
The various groups have both disparged and helped each other.

As far as this paper is concerned AI will be defined as the study of agents that receive inputs from the environment and perform actions accordingly. Each such agent implements a function that maps input sequences with the corresponding actions/outputs.

An agent is just something that acts (agree in Latin, to do). Of course all computer programs do something, but computer agents are expected to:

- Operate autonomously.
- Perceive their environment.
- Persist over a prolonged time span.
- Adapt to change.
- Create and pursue goals.

A rational agent is one that given the inputs from the environment acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

2) ARTIFICIAL NEURAL NETWORKS (A.N.N.)

Artificial Neural Networks are computer agents inspired by the biological neural networks that we have in our own brain.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, this original goal changed over time as this kind of technology may not resemble all human reasonings and rationality. ANN's employment shifted towards performing efficiently and effectively very specific tasks, leading to an inevitable deviation from the biological networks.

It is important to stress out that an artificial neural network is not a specific algorithm, it is rather a framework for many different machine learning algorithms all of which interact in order to process complex data inputs.

Before training you may think at this system as a newborn brain that is not programmed to perform any specific task. So how do you train this novel network?
We will see it in detail later on, but in brief the system learns how to perform tasks just by adjusting the 'power' of each synapsis in order to match a large data set of examples.

For instance, let's suppose that the network must solve an image classification problem. The network might learn to identify images that contain tigers by analyzing a manually labeled dataset of "Tiger" or "No Tiger" images. Once it is trained on this dataset, the network may identify tigers in other novel images that it sees.
All of this is achieved without any prior knowledge about tigers, for instance, that they have sharp nails, fur and whiskers. Instead, they automatically generate identifying features for tigers from the learning database that they've just processed.

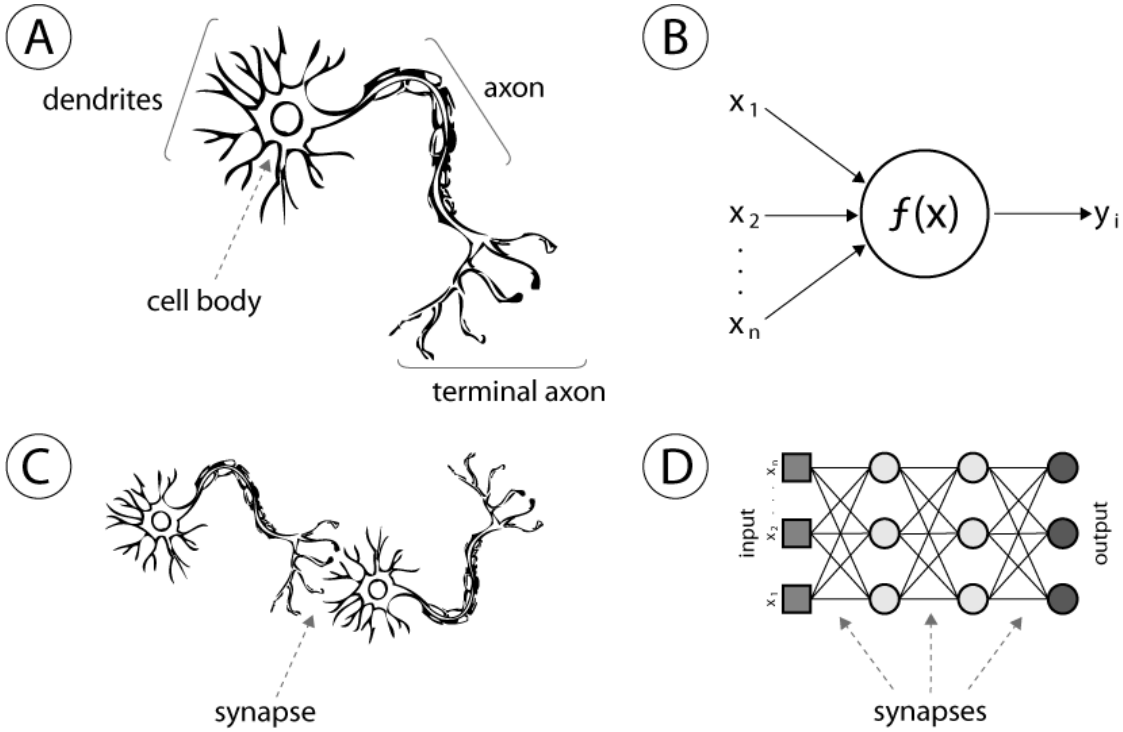


figure 2: Comparison between Neural Networks and Human Brain Cells

In the image above you may appreciate the similarities between the structural principles of biological networks and artificial ones.

Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The incoming signal in the artificial neurons is a real number, one real number for each connection with the neurons of the previous layer. The output of each artificial neuron is computed by feeding the sum of all inputs into a non-linear function *f(x)*.
The connections between neurons in a human brain are called Synapsis, in the ANN we address them as '*edges*'.
At all edges typically another real number is associated that adjusts as learning proceeds, this number is called ‘ *weight*’. The weight increases or decreases the strength of the signal at a connection.
Sometimes in order Artificial neurons may have a threshold called ‘*Bias*’. If a Bias is present the signal is only sent to the next layer if the aggregate sum of the inputs signals crosses that threshold.

In this way the programmer may choose which neurons you want to fire the most (according to the final goal of this network).

2.1 Neural Network Structures

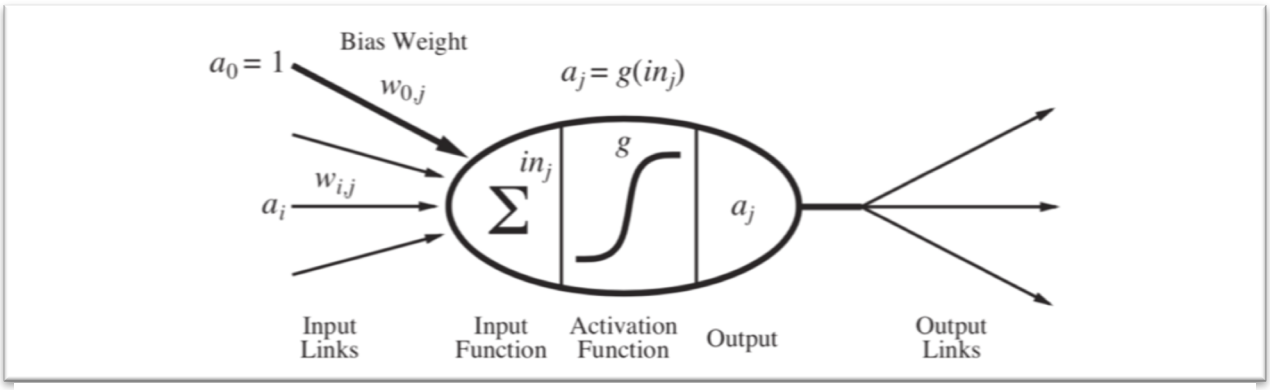


figure 3: Mathematical Structure of a Node

Neural networks are composed of several units called 'nodes'. Each node is connected with other nodes by links. The job of a link that connects unit *i* to unit *j* is to propagate the activation *ai* from *i* to *j*. A fundamental property of a link is to have its own number defined as weight *wi,j* that determines strength and sign of that link.

Each unit j first computes a weighted sum of its inputs:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i$$

An activation function 'g' is applied to the summation in order to derive output:

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$

2.2 Activation Functions

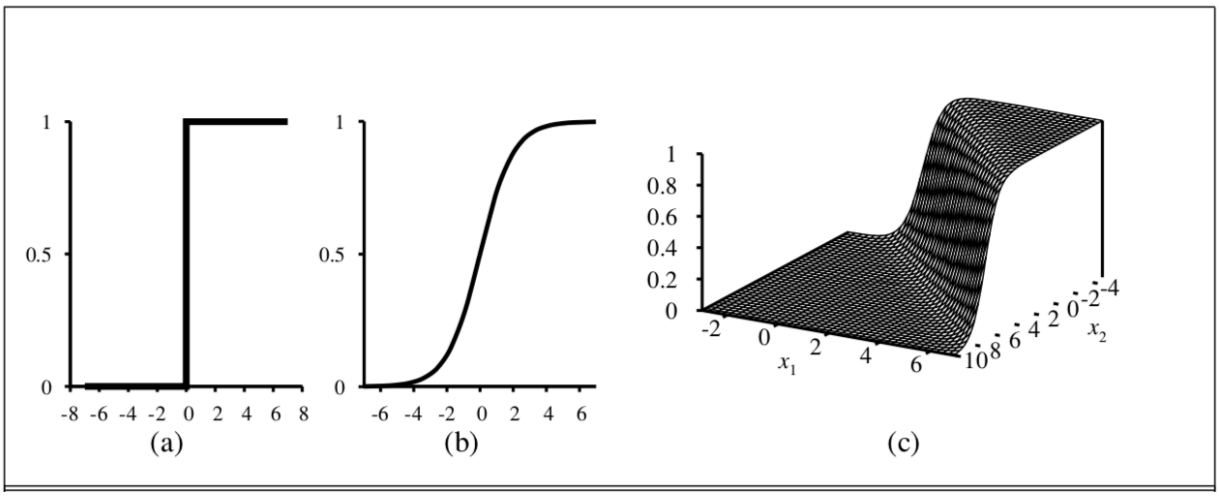


figure 4: Perceptron threshold (a) and a sigmoid threshold (b)

There are 2 main types of activation functions 'g':

- A hard threshold: neuron is referred as '*Perceptron*' (Figure a).
- A sigmoidal function: neuron is referred as '*Sigmoid Perceptron*' (Figure b).

Both these functions have as a main scope to 'squeeze' any large input that the neuron receives into a number between 0 and 1. You may easily observe that the sigmoidal function is differentiable and can represent a non-linear function.

2.3 Connections

After analyzing the mathematical meaning of neuron it will be now seen how these units may be connected to form the global network. Two main layouts exist:

- *Feed-forward network:* this network type has the fundamental property of being an acyclic structure. The links between each neuron present one and only direction that goes from left to right. Every neuron of the network receives inputs from an upstream unit and sends its output to a downstream one. Feed-forward networks are subdivided in layers, such that each neuron receives inputs only from units located in the immediately preceding layer (e.g. in figure….)

- *Recurrent network:* the main difference is that in these networks the response of a given input is directly dependent on its initial state. Therefore these networks display something like a short term memory. It is hence a simplified model of human brain.
  It is made up of a closed loop in which the outputs are delivered back into its inputs. This means that this dynamic structure may tend to a steady state or to a massive chaotic behavior.

The neural network that will be exploited during the calibration process is a feed forward single hidden layer neural network that will undergo a peculiar type of training, called ELM. For the sake of completeness the next subsection will give a general overview on the 'classic' backpropagation training of a single layer feed forward network that aims to solve a linear regression problem.

2.4 Training

As our case of study is a Linear Regression problem. We will focus on the training that best fits linear regression.

2.4.1 Univariate Linear Regression

Given a set of N points, the goal of a univariate linear regression is to find a straight line that best approximates all the data. This line will be posed in the form:

$$y = x w_1 + w_0$$

Where *x* and *y* will be input and output respectively and *w1,w0* be the weights that will be adjusted to achieve the optimum fit of data. Therefore it is convenient to define the **W**=[*w0,w1*] as a vector element. If we rewrite the line equation as follows:

$$h_{\boldsymbol{w}}(x) = w_1 x + w_0$$

The quest will be finding the best couple of weights that will 'minimize the empirical loss'. It is a good idea to exploit the squared loss function, that we will refer as L_2, summed over all the n training examples:

$$Loss(h_{\boldsymbol{w}}) = \sum_{j=1}^{N} L2\left(y_j, h_{\boldsymbol{w}}(x_j)\right) = \sum_{j=1}^{N} \left(y_j - h_{\boldsymbol{w}}(x_j)\right)^2 = \sum_{j=1}^{N} \left(y_j - w_1 x_j + w_0\right)^2$$

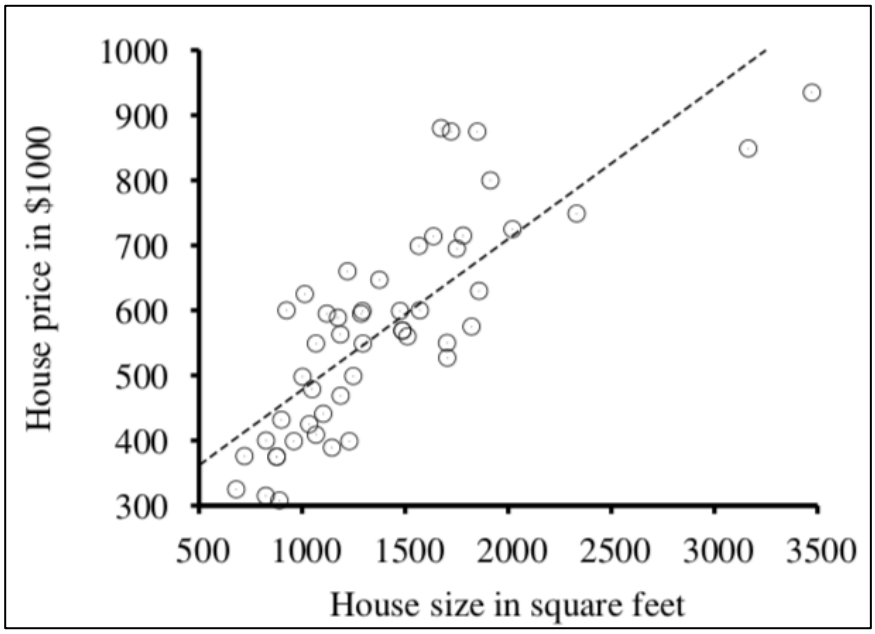


figure 5: Example of a Linear Regression

The goal now is to find the vector: $\boldsymbol{w^*} = argmin_{\boldsymbol{w}} Loss(h_{\boldsymbol{w}})$.

The sum $\sum_{j=1}^{N}\left(y_j - w_1 x_j + w_0\right)^2$ is minimized if its partial derivatives with respect to the weights are null:

$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}\left(y_j - w_1 x_j + w_0\right)^2 = 0$ and $\frac{\partial}{\partial w_1}\sum_{j=1}^{N}\left(y_j - w_1 x_j + w_0\right)^2 = 0$

The solution of these 2 equations is unique:

$$w_1 = \frac{N\left(\sum x_j y_j\right) - \left(\sum x_j\right)\left(\sum y_j\right)}{N\left(\sum {x_j}^2\right) - \left(\sum x_j\right)^2} \; ; \qquad w_0 = \frac{\left(y_j - w_1\left(\sum x_j\right)\right)}{N} \qquad (18.3)$$

Several different learning algorithms base their acquisition of knowledge tweeking the values of all the elements of a 'weight space'. This space in the previous example will be 2D, therefore we may plot the function *hw(X):*

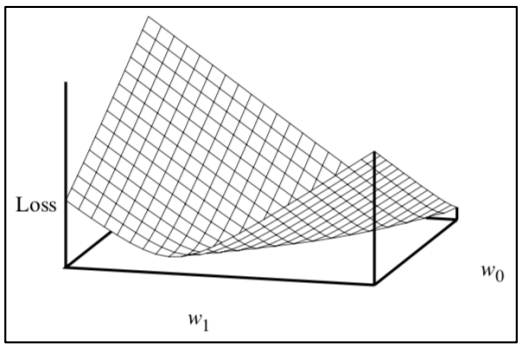


figure 6: 3D Plot of the Loss Function

You may easily perceive that the loss function has a convex shape. This will be the same case of all functions L2, they will always present a global minimum point (but not local minimums). Therefore for univariate linear regressions that will be the end of it.

2.4.2 Non Linear Models

In case of non-linear regression problems, the problem will be that the minimum empirical loss solution will not have a closed shape. Therefore the deal now will be to search for a minimum in a continuous weight space, hence seeking for a general optimum point.

Usually the best algorithm that solves this issue is the gradient descent hill climbing one. This kind of search follows the path of the 'maximum descending gradient' in the neighborhood of your point in the weight space. On the long run this path will lead to the minimum point of the function.

Here under you may find the logic of the search:

$$\begin{array}{l}\mathbf{w} \leftarrow \text{any point in the parameter space} \\ \textbf{loop until convergence } \textbf{do} \\ \quad \textbf{for each } w_i \textbf{ in w do} \\ \quad\quad w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})\end{array}$$

α is named 'step size' and it is a parameter that may also be addressed as the 'learning rate'.
You may decide if keep alpha constant or change its value over time, according to your search problem.
Thanks to the gradient descent approach convergence to the unique global minimum is guaranteed (as long as we pick α small enough) but may be very slow.

2.4.3 Multivariate Linear Regression

It is not difficult to generalize the univariate regression to a multivariate regression problem. The main difference between the two stands in the dimension of the weight vector. In the previous case it was bi dimensional, now it will be n-dimensional. Let's call ***xj*** the n dimensional vector**.**
The hypothesis space is made up of these functions:

$$h_{sw}(\boldsymbol{x}_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}$$

You may notice how the first term *w0* seems to lack of its own variable, this is easily adjusted by inserting a 'dummy input' *xj,0* equals to 1. Hence the function *h* will be defined by the 'dot product':

$$h_{sw}(\boldsymbol{x}_j) = \boldsymbol{w} \cdot \boldsymbol{x}_j = \boldsymbol{w}^T \cdot \boldsymbol{x}_j = \sum_i w_i \, x_{j,i}$$

Now the optimal weight vector W* will minimize, as we did previously, the quadratic error:

$$\mathbf{w}^* = \; argmin_{\boldsymbol{w}} \sum_j L_2(y_j, \boldsymbol{w} \cdot \boldsymbol{x}_j)$$

3) Error Compensation Principle with Extreme Machine Learning

The absolute spatial calibration method proposed in this paper exploits the Extreme Learning Machine (ELM) principle, which ,in other words, is defined as the error minimization method that has been analyzed in the previous the multivariate linear regression paragraph. This specific training method avoids the time-consuming necessity to develop a kinematic model of the machine by identifying its kinematic parameters. Moreover the calculation time is significantly reduced compared to a 'classic' back-propagation learning process. Finally, both geometrical and non-geometrical errors of the industrial manipulator NS 12-185 are taken into account with the ELM.

The built model therefore aims to predict the positioning error using the ELM.
We will see later how the input and output vectors of the neural net have been created using a simulation in RoboSim and Matlab enviorments.

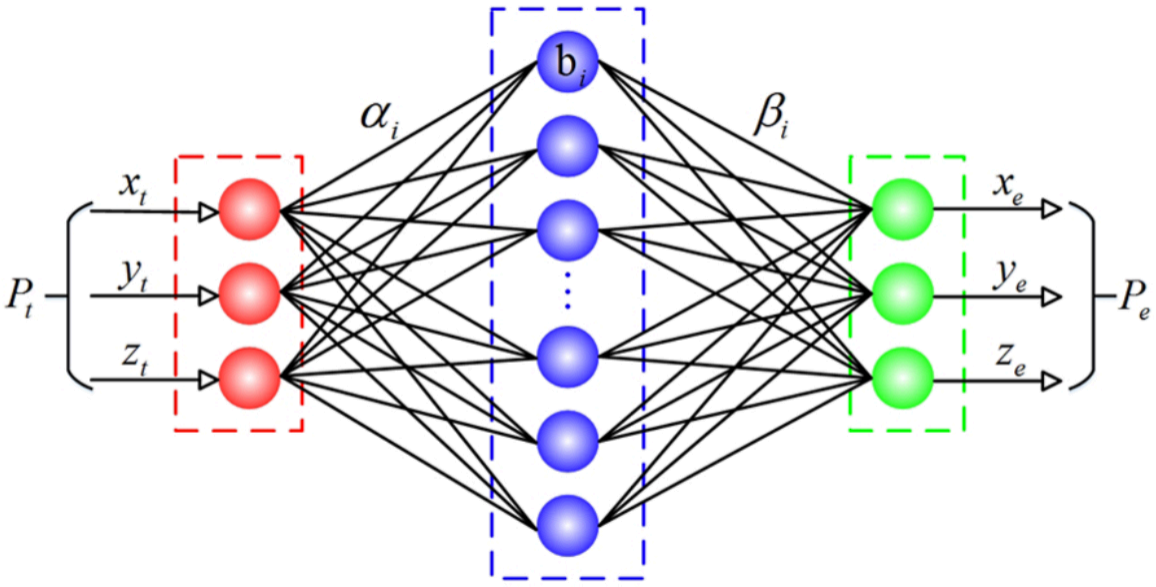


figure 7: Single Feedforward Multi Input- Multi Output Neural Network Scheme

3.1 General ELM compensation 3 inputs 3 outputs

Compensation will be made by referring to the tool center point of the robotic arm's end-effector. It has been decided to maintain a constant orientation of the tool during the whole simulation, in order to decrease the number of variables in game.

It is meaningful , in order to be able to fully appreciate the objective of the successive simulations carried out during this project, to firstly analyze the generic method of compensation obtainable using a network to 3 neurons of input and 3 neurons of output (in figure..)
The three nodes in the input layer correspond to the 3 elements of the vector *Pt= [Xth,Yth,Zth].* Let's suppose that the intermediate layer is composed by an integer number K of neurons. The output of the network will be the vector of the errors corresponding to each coordinate *Pe=[Xe,Ye,Ze].*

The absolute error is calculated as follows:

$$\boldsymbol{e} = \sqrt{x_e^2 + y_e^2 + z_e^2}$$

3.2 ELM Training

Given N learning sample pairs *Ptj* and *Pej,* with both the vector belonging to the vectorial space R^3
The output of the neural network can be defined as :

$$f(\boldsymbol{P}_{tj}) = \sum_{i=1}^{k} \boldsymbol{\beta}_i G_i(\boldsymbol{\alpha}_i, b_i, \boldsymbol{P}_{tj}) \quad , \qquad j = 1,2,\dots,N$$

where $\boldsymbol{\alpha i}$ is the input weights of the *ith* hidden neuron, *bi* is the bias of the i*th* hidden neuron (*i*=1,2….K) and $\boldsymbol{\beta i}$ is the output weights connecting the *ith* hidden neuron to the output layer.

With :

$$\boldsymbol{P}_{tj} = \lceil x_{tj}, y_{tj}, z_{tj} \rceil^T$$

$$\boldsymbol{P}_{ej} = \lceil x_{ej}, y_{ej}, z_{ej} \rceil^T$$

$$\boldsymbol{\alpha}_i = \lceil \alpha_{i1}, \alpha_{i2}, \alpha_{i3} \rceil^T$$

$$\boldsymbol{\beta}_i = \lceil \beta_{i1}, \beta_{i2}, \beta_{i3} \rceil^T$$

G(...) is a sum function that in our case will be made up of the activation function *g(....)*of the neurons belonging to the hidden layer. The latter will be a sigmoidal function in our simulation.

$$G(\boldsymbol{\alpha_i}, b_i, \boldsymbol{P_{tj}}) = g(\boldsymbol{\alpha_i P_{tj}} + b_i)$$

The ELM looks for the optimum values $\boldsymbol{W}$ (αi, βi, and b*i*) that minimize the error *E(**W**)*:

$$E(\boldsymbol{W}) = \sum_{j=1}^{N} \left(f(\boldsymbol{P_{tj}}) - \boldsymbol{P_{ej}}\right)^2$$

Now from the equation (2) we can write the following matrix $\boldsymbol{H}$*:*

$$\boldsymbol{H\beta} = \boldsymbol{T}$$

H represents the output matrix of the intermediate layer. Its *ith* column is made up of the outputs of all the N input instances corresponding to the *ith* hidden neuron. Moreover *H* is defined as shown in equation (10), and matrices β and T are presented in the order of equations (11) and (12)

$$\boldsymbol{H} = \begin{bmatrix} G(\boldsymbol{\alpha_1}, b_1, \boldsymbol{P_{t1}}) & \cdots & G(\boldsymbol{\alpha_k}, b_k, \boldsymbol{P_{t1}}) \\ G(\boldsymbol{\alpha_1}, b_1, \boldsymbol{P_{t2}}) & \cdots & G(\boldsymbol{\alpha_k}, b_k, \boldsymbol{P_{t2}}) \\ \vdots & \cdots & \vdots \\ G(\boldsymbol{\alpha_1}, b_1, \boldsymbol{P_{tN}}) & \cdots & G(\boldsymbol{\alpha_k}, b_k, \boldsymbol{P_{tN}}) \end{bmatrix}_{N \times k}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1^T & \boldsymbol{\beta}_2^T & \cdots & \boldsymbol{\beta}_k^T \end{bmatrix}^T_{k \times 3}$$

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{P_{e1}^T} & \boldsymbol{P_{e2}^T} & \cdots & \boldsymbol{P_{eN}^T} \end{bmatrix}^T_{N \times 3}$$

The Extreme Learning Method randomly sets the parameters $\boldsymbol{\alpha i}$ and *bi* of the *K* hidden neurons. By doing so the βi weights that minimize *E(W)* may be directly calculated with no need for any iterative process:

$$\boldsymbol{B} = \boldsymbol{H}^{+}\boldsymbol{T}$$

H+ is a Moore–Penrose generalized inverse matrix. This specific matrix is the product of the following:

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

After the matrix β is obtained, the neural network for the prediction compensation model is trained.

Here below in Figure (…) you may appreciate a graphical representation of the training procedure. Vector *Pt* will as before be the theoretical position of the tool center point. It is is used as the input data. The output of the industrial manipulator is the real position *Pr*.
Hence the positional error *Pe*:

$$\boldsymbol{P_e = P_t - P_r}$$

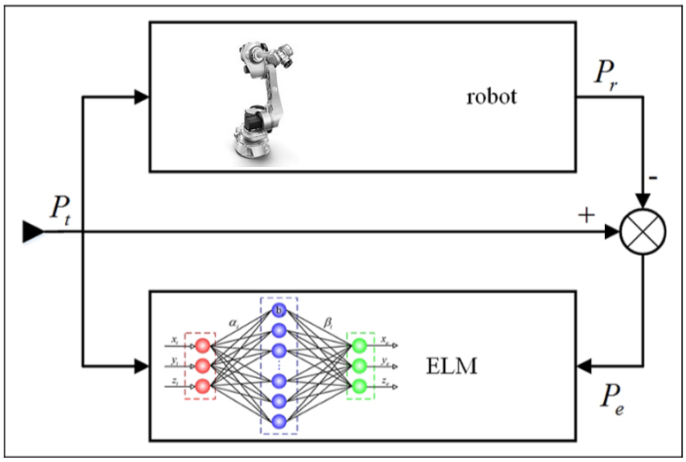


figure 8: Neural Network Training Flow Chart

3.3 Compensation Process

Finally the trained network can be used for the prediction of the positional error *Pe* given as input the vector *Pt* of the initial theoretical spatial positions that the robotic arm wants to reach during its operations. This calibration method does not aim to vary the kinematic parameters in the controller. It will solely calculate adjusted *P't* input coordinates that take into account the forecasted error (outputted by the trained network).

$$\boldsymbol{P'_t = P_t - P'_e}$$

To reach the specified location, the controller would give the drilling robot the position coordinates P't and not the initial theorical desired position coordinates.

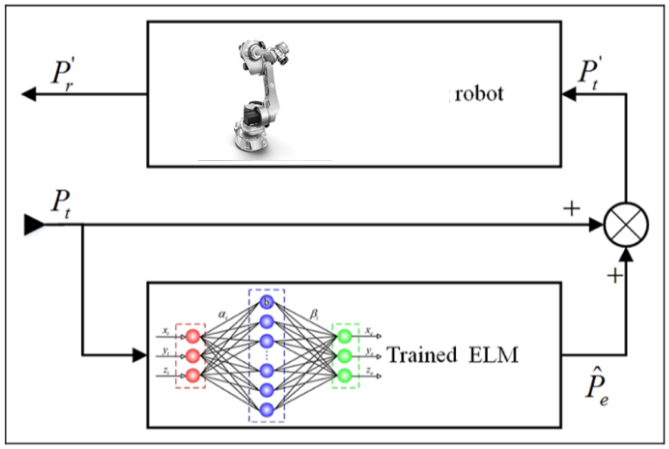


figure 9: Calibration Flow chart with Error Compensation through the Neural Network

4) COMAU's Calibration State of the Art

It is crucial to stress out that the calibration of one of COMAU's products is an extra service that the customer must specifically require while purchasing a Robot.

This calibration is necessary according to the application of the machine purchased by the client. If this robotic arm is exploited in high precision applications, then a calibration process is compulsory and it is performed by COMAU itself.

The technology used by COMAU for this absolute calibration is referred as 'KRYPTON' and it is provided by an external company named METRIS.

It is important to highlight that this process usually performs a Static Calibration. You may also try to improve dynamic accuracy of the robotic arm (by increasing the LEDs frequency pulse) but the outcome, most of the times, isn't entirely satisfying.

4.1 Krypton

Metris designs, develops and markets world-class solutions for optical metrology in the automotive and aerospace manufacturing business.

Metris integrates with its metrology technology with specific software solutions. In our case the latter developed an algorithm (covered by industrial secrecy) that compensates the spatial error of COMAU's machine in order to increase its spatial absolute accuracy.

4.2 Measurement System

The entire system is made up of the following parts:

- K600 CCD Camera
- Camera control unit
- Measurement probe (for manual acquisitions).
- Multiplexer boxes and infrared LED's
- PC

4.2.1 K600 Camera

The K600 camera system is a 3D measurement system based on three linear CCD cameras. By triangulation the position of an infrared LED in space is calculated. This can be a static or a dynamic measurement (in our case it will be a STATIC measurement).

The field-of-view of the camera is determined by the overlap area of the three linear CCD-camera's in the camera unit, resulting in a pyramidal volume.

figure 10: Photo of the Real K600 Camera

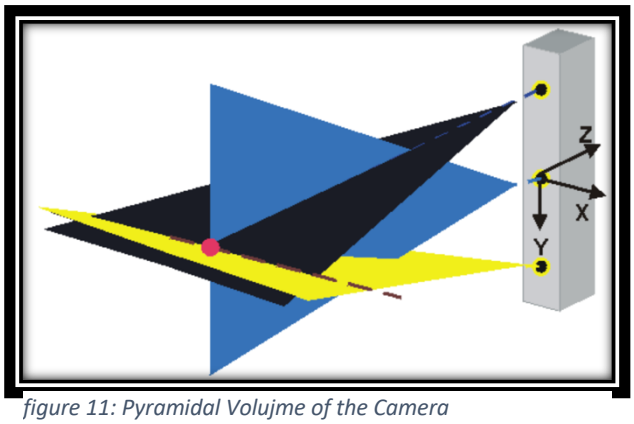


figure 11: Pyramidal Volujme of the Camera

The technical specifications of the K600 camera are provided in the Manual. Regarding the Field of view of the K600 camera system, you may observe that there are 3 zones of accuracy (proportional to the distance from the camera).

To obtain a good accuracy it is ought to plan the robot path inside the zone I and II .

K600 CAMERA UNIT

Field-of-view: 17 m³, distributed into three accuracy zones as follows:

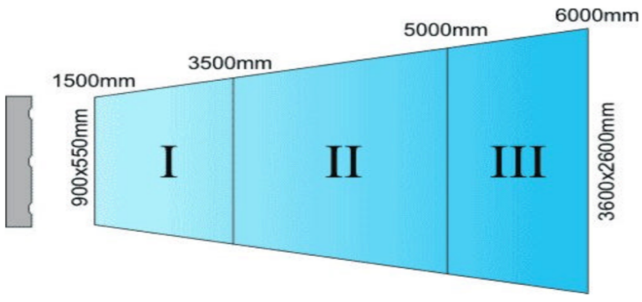


Figure 10: camera field-of-view

Resolution : 0,002 mm at 2,5 mm

Noise (1σ) : 0,010 mm

Accuracy :

Zone	Volumetric Accuracy (±2σ)	Single Point Accuracy (±2σ)
I	90µm + 10µm/m	60µm + 7µm/m
II	90µm + 25µm/m	60µm + 17µm/m
III	190µm + 25µm/m	130µm + 17µm/m

figure 12: Field of View of the Camera with all the Three different Accuracies Zones

4.2.2 LEDs layout

On COMAU's robotic arms 5 LEDs are disposed on a square alluminium plate. The latter is mounted on a 'test mass' (yellow) attached on the robot Flange. The test mass is usually equal to the maximum payload of the robotic arm, in order to maximize de spatial error produced due to inertial forces.

The K600 camera will scan all 5 LEDs. If one of these LEDs is out of the camera field that specific position is not valid and it is discarded from the calibration process.
This 5 LED feedback system allows Krypton to reduce drastically errors during the measurement process.

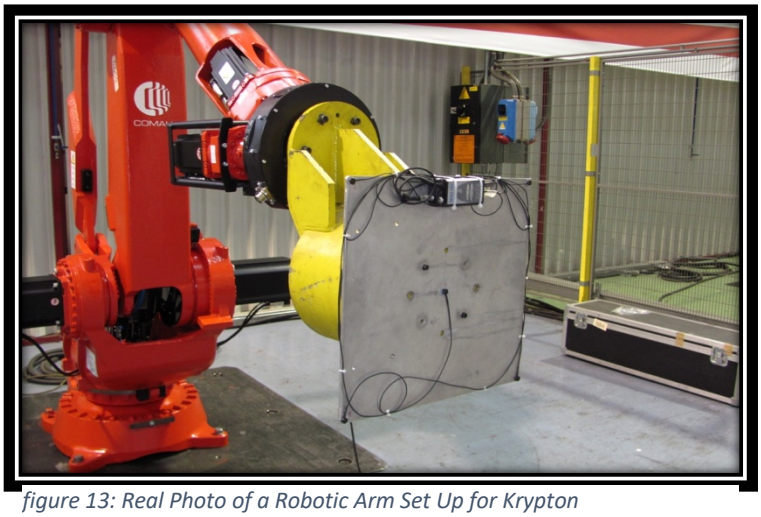


figure 13: Real Photo of a Robotic Arm Set Up for Krypton

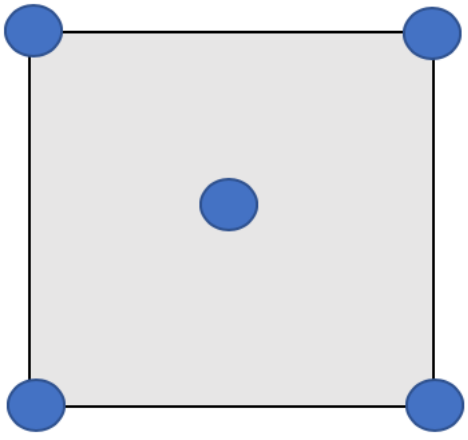


figure 14: Led Setup on Robotic Flange

4.3 Calibration Process with Krypton

1. Program a standard path of 150 THEORICAL STATIC poses (*Xth, Yth, Zth*) of the robotic arm. This path must necessarily take into account the vision area of the camera.

2. Identification: Once the movement program is fed to the robot controller, Krypton through the K600 camera acquires 110 real static poses (***Xreal, Yreal, Zreal***) of the robot (this number may sink down to 90 as some points are eliminated due to measurement errors that we mentioned before).

3. Krypton elaborates the

   Pre-Calibration Average error = (Real poses coordinate - Theoretical poses coordinates)

4. Krypton runs its internal error compensation algorithm for that specific machine.

5. The Validation process is executed on 50 different poses, again calculating *pre* and *post* calibration average error

All of the described movements are performed at about 25% of the max speed (in order to minimize dynamic measurement inaccuracies).

The desirable outcome will always be:

Pre-Calibration Error > Post Calibration Error

If this condition is verified then the static calibration process is successful (the success rate is close to 100 %).

Usually a Large Robot will cope with initial inaccuracy of 3-5 [mm].
After Krypton's calibration process the error will sink down of 1 order of magnitude: 0,3-0,5 [mm].

Therefore this specific machine will feature a higher spatial absolute accuracy that is of crucial importance for client's high precision applications.

5) SIMULATION

5.1 THE ROBOT NS 12-185

Amongst the broad catalogue of COMAU's robotic products, for our simulation we chose to work with the industrial manipulator NS 12-185. This name is a code specifically chosen by COMAU to sum up the main features of this machine:

N is the name chosen by COMAU for this family of robots.
S refers at the type of wrist Spherical wrist.
12 stands for the maximum wrist payload [Kg].
185 defines machine's maximum reach in its working area. [cm]

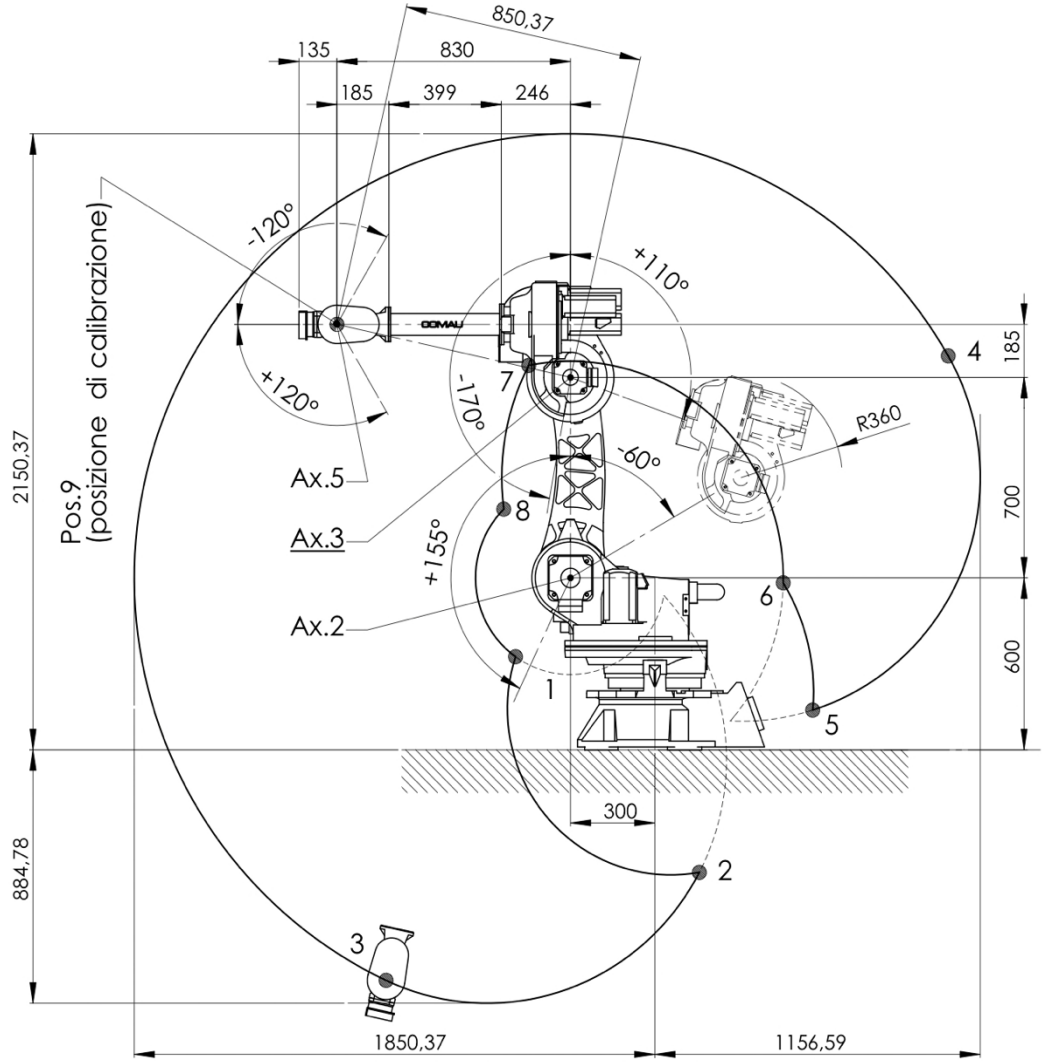
The NS 12-185 system is made out of:

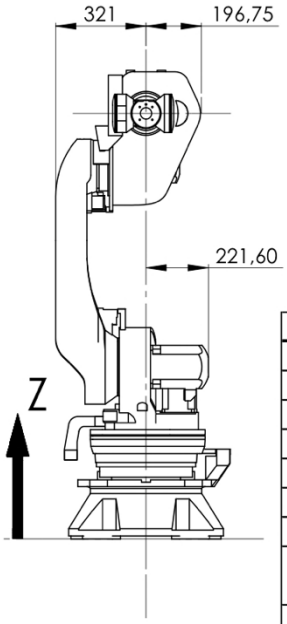
- A 6 axis robotic manipulator. The latter are assembled in an open kinematic configuration.
- Control unit: C5G
- Men Machine Interface: TP5

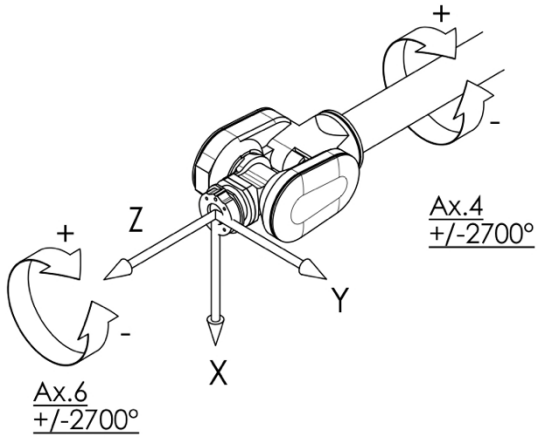
It is specifically designed for rapid and accurate applications. This robot suits perfectly a wide range of tasks in machining, handling, assembly and arc welding processes.
In the next page you may find an accurate list of the technical specifics of the manipulator.

5.1.1 Technical Specifics NS12-185 Robotic Arm

Number of axes	6
Maximum wrist payload (Kg)	12
Additional load on forearm (Kg)	10
Maximum horizontal reach (mm)	1850
Torque on axis 4 (Nm)	39
Torque on axis 5 (Nm)	39
Torque on axis 6 (Nm)	20
Stroke (Speed) on Axis 1	+/- 180° (155 °/s)
Stroke (Speed) on Axis 2	-60°/ +155° (155 °/s)
Stroke (Speed) on Axis 3	-170°/ +110° (170 °/s)
Stroke (Speed) on Axis 4	+/- 2700° (360 °/s)
Stroke (Speed) on Axis 5	+/- 120° (350° /s)
Stroke (Speed) on Axis 6	+/- 2700° (550° /s)
Repeatability (mm)	0.05
Tool coupling flange	ISO 9409 - 1 - A63
Robot weight (Kg)	335
Protection class	IP65 / IP67
Mounting position	Floor / Ceiling / Sloping (45° max)
Operating Areas A (mm)	2150
Operating Areas B (mm)	1850
Operating Areas C (mm)	950
Operating Areas D (mm)	1157
Operating Areas E (mm)	885

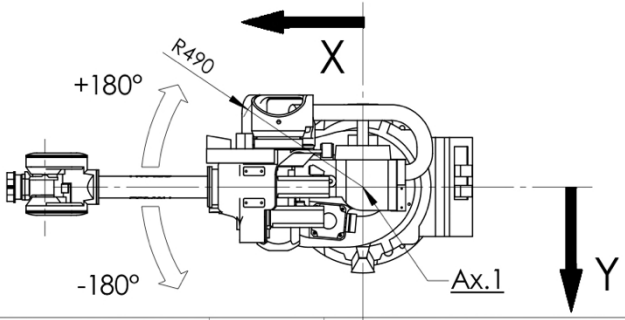






Pos	X	Z	Ax.2	Ax.3
	[mm]	[mm]	[deg]	[deg]
1	495,17	324,93	+40°	-170°
2	-158	427,93	+155°	-100°
3	955,21	-805,11	+155°	-12,57°
4	-1042,66	1375,18	-60°	-12,57°
5	-561	138,7	-60°	+90°
6	-456,32	582,55	-20°	+110°
7	448,51	1341,8	+82,64°	+110°
			-60°	-135,13°
8	537	839,97	-60°	-170°
9	1130	1485	0°	-90°

Giunti in posizione di calibrazione (pos.9)					
Ax 1	Ax 2	Ax 3	Ax 4	Ax 5	Ax 6
0°	0°	-90°	0°	0°	0°



N° Complessivo - Assembly Dwgs | Quantita' per Complessivo - Q.ty Assemby | N° Part. - Part. N°

N° - No. | Modifiche - Revisions | Data - Date | Visto - Approv

Inizio N° - Sheet first N° | Fine N° - Sheet last N° | Ultimo N° - Last used

Numerazione conduttori - Wires numbering

Tipo - Type | Codice - Code

Materiale - Material

Superficie - Surface | Termico - Heat

Trattamento - Treatment

No. | Kg. | HRC / HB

Modello - Model | Peso - Weight | Durezza - Hardness

Quote senza indicazioni di tolleranza grado precisione medio: UNI ISO 2768

For dimensions with no tolerance indication average accuracy: UNI ISO 2768

Tolleranze generali - General tolerances

Commessa - Job | 11.09.06 Data - Date | Scala - Scale | A 2 Formato - Size

Foglio - Sheet 1/2 | Disegn. - Drawn Marcon | Visto - Checked

Titolo - Title: ROBOT SMART NS 12 -1.85-HAND/FOUNDRY AREA OPERATIVA

N° Disegno - Drawing No. CR82280916 | COMAU

5.1.2 The C5G Control Unit

figure 15: C5G Control Unit

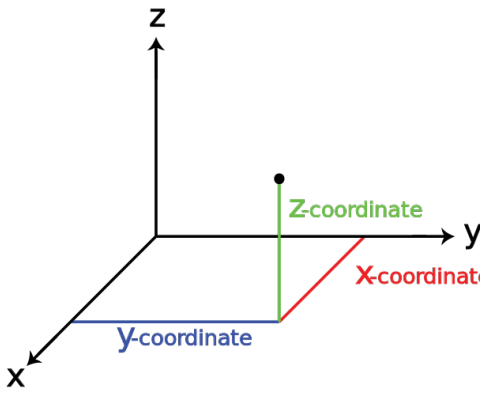
Every Robot needs a brain that controls the entire set of operation it is commanded to perform. The NS12-185 follows the commands of the C5G control unit.

Inside this unit you find all the drive units that singularly communicate with each motor driving each axis of the robot. These units are modular, therefore you may add more drive units, in the cabinet, up to 13 total axes.

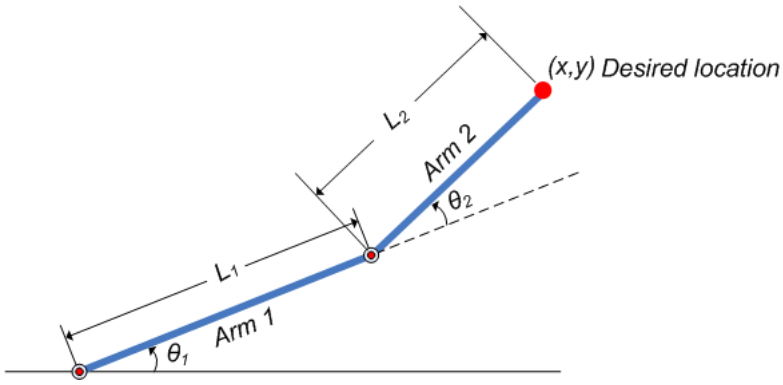
The C5G is driven by a state of art industrial PC APC820 with Core2 Duo technology CPU which guarantees high performances while maintaining low energetic consumption.

Through this device you are able to upload the instructions to the servo motors according to the specific application you are pursuing. You can feed the motion instructions both in cartesian or joint coordinates.

5.1.3 Cartesian Coordinates



5.1.4 Joint Coordinates



5.1.5 Teach Pendant “TP5”

figure 16: Teach Pendant TP5

The interface between the Robot and a human operator is the Teach Pendant (TP5).

This Man Machine Interface has been developed by COMAU and it is part of the Control unit C5G. Thanks to this ergonomic device an operator may interact with the robot.

It is provided with a 7” wide touch-screen and a Simplified keyboard designed to locate keys more easily during the programming phase thanks special tactile references on the membrane with which you may:

- Monitor Robot’s status and movements
- Upload/modify motion programs that the robot is following
- Regulate the speed/accelerations of the entire system or of a single axis
- In case of emergencies stop the entire machine

To correctly use this controller, all the operators and engineers, from both customers and COMAU side, must follow a specific training.

5.2 Database Gathering

Based on extensive literature research, an 'excellent' compromise was found between the number of neurons in the middle layer and the number of poses of the Dataset.

The number of neurons in the middle layer of the neural network is 20. The Dataset, on the other hand, will be composed of 1000 total poses, subdivided in turn into 800 poses used for the training of the net and another 200 poses for the verification of the qality of the linear regression carried out by the latter.

The database has been created using two softwares: Matlab and Robosim.
The latter is a Java based portable simulator to visualize and understand the Robot Localization, Path planning, Path Smoothing and PID controller concepts. It is very flexible and easy to use. has been developed and is the property of COMAU itself and allows to create a digital twin of any Robot in their portfolio.

5.2.1 Input creation:

The 1000 spatial points where the robot will be positioned to detect 1000 static poses have been created in a Matlab environment, starting from a motion program already used by COMAU engineers in carrying out ISO tests. During these specific tests, the robot follows a movement program that combines 150 poses included within a parallelogram.

Starting from this program we have simply modulated some parameters in order to:

- Increase the number of random spatial points (x.y.z) from 150 to 1000
- Hire all of the newly created 1000 points within a 1200x600x1350 [mm] cube, which densifies the I zone of the Krypton camera (maximum accuracy).

Here below you will find the print of the Matlab program used:

```
a% limiti in cartesiano
  X_1 = transpose([800  : 1 : 1400]);
  X_2 = transpose([800  : 1 : 1400]);
  X = [X_1; X_2];
  Y = transpose([-600 : 1 :  600]);
  Z = transpose([312  : 1 : 1687]);

% creazione posizioni cartesiane random
  KI_MAX_SIZE = 1000;
  Position    =
[X(randperm(KI_MAX_SIZE)),Y(randperm(KI_MAX_SIZE)),Z(randperm(KI_MAX_SIZE))];
  Orientation = [0,90,0];

% Creazione Posizioni
  for vi_Idx = 1 : length(Position)
    TotalPosition(vi_Idx, :) = [Position(vi_Idx,:) , Orientation];
  end % for vi_Idx = 1 : length(Position)

% creazione position
  for vi_Idx = 1 : length(TotalPosition)
    cr_TotalPosition{vi_Idx} = ['pnt',num2str(vi_Idx),'p :=
POS(',num2str(TotalPosition(vi_Idx,1)),', ',num2str(TotalPosition(vi_Idx,2)),',
',num2str(TotalPosition(vi_Idx,3)),', ',num2str(TotalPosition(vi_Idx,4)),',
',num2str(TotalPosition(vi_Idx,5)),', ',num2str(TotalPosition(vi_Idx,6)),',
''W'')'];
  end % for vi_Idx = 1 : length(TotalPosition)
```

It should be noted that part [0,90,0] refers to the orientation of the end-effector. In our simulation we want the end-effector to maintain a constant orientation. This approach is more suitable for a first analysis of this new technology, decreasing the number of variables involved.

5.2.2 Reachability Verification of the Theoretical points

After creating the desired motion program, we translate the .m into a PDL file.This is because the Robosim software reads the programs in .PDL.
At the end of the PDL script we insert two crucial functions:

- POS-TO-JOINT': This function developed by Comau performs inverse kinematics (from Cartesian coordinates (x,y,z,0,90,0) to joint coordinates (theta1,theta2,theta3,theta4,theta5,theta6). We are obliged to carry out this transformation because Robosim only reads motion commands in joint coordinates.

- MOVE TO': as the name suggests, this function checks in simulation the reachability of the robot NS185...of the theoretical spatial point. (there are spatial points that the real robot cannot reach because of physical limits created by the physical configuration of the machine itself) .

If a point is not reachable, RoboSim blocks the simulation on that point. This point will then be deleted from the database.

Among the various features, Robosim presents a graphic simulator called 3D Blender, in which you can load and display the digital twin of our NS robot during all the movements of the program.

RoboSim also has a virtualization of the TP5 with which you can control the vital robot. To all intents and purposes, therefore, it is like having the real robot in front of you.

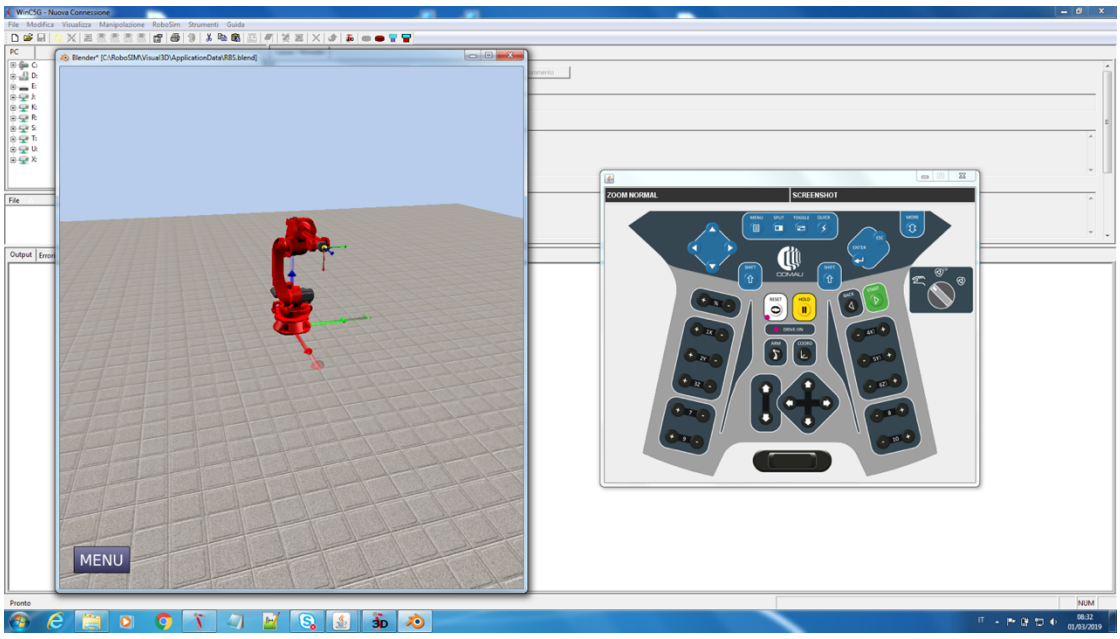


figure 17: RoboSim interface with 3D Blender on the Left and the Virtual TP5 on the Right

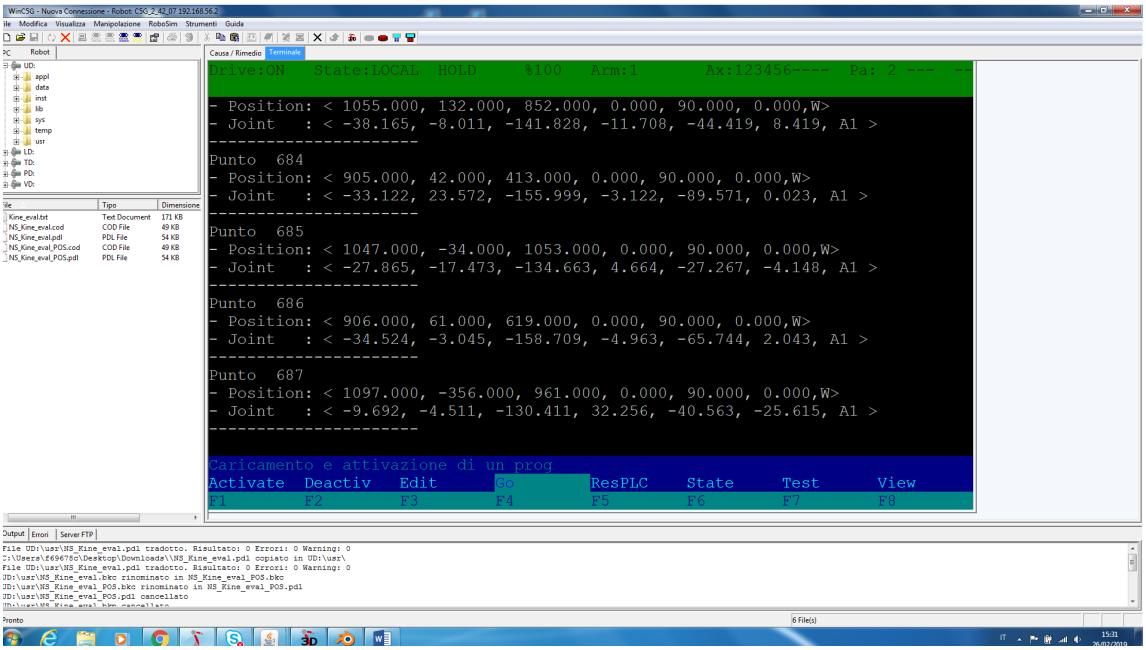


figure 18: RoboSim print of the Cartesian and Joint Coordinates of each Tested Spatial Point

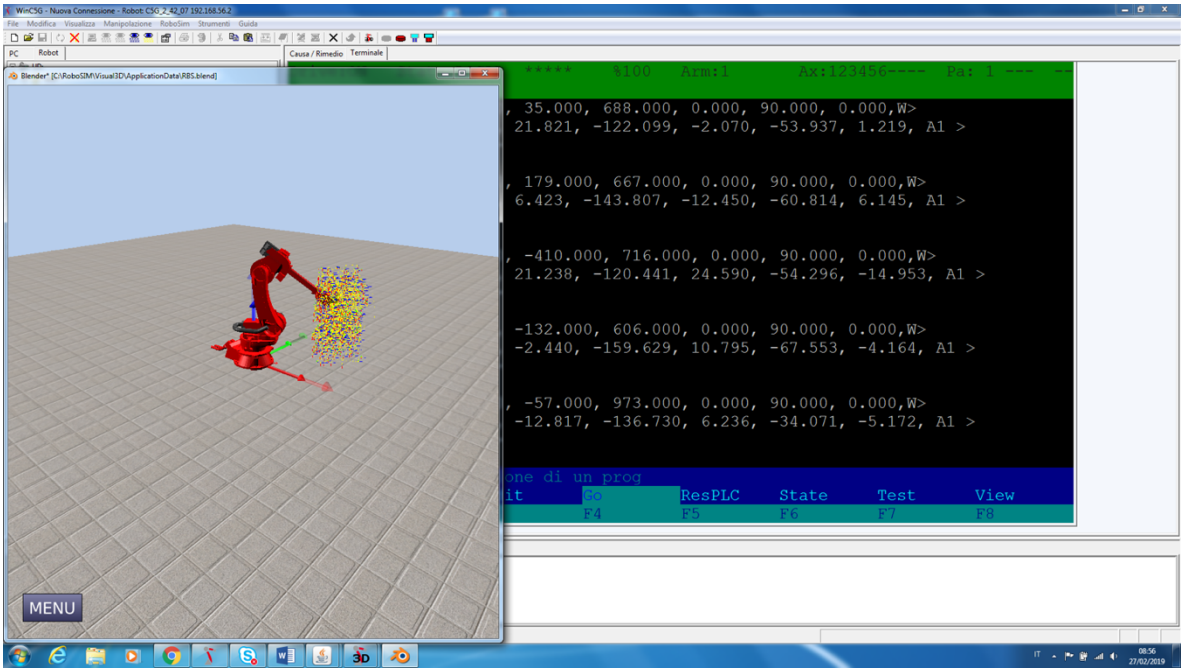


figure 19: RoboSim Trajectory Trace of all the 1000 positions undergone in the Database Creation

After the first run of the PDL program All 1000 points are reachable, Therefore ***P_input*** is successfully created

5.2.3 Simulated Spatial Error Creation

In the real working area, as already specified in chapter 1, the end effector will undergo an the absolute spatial error due to the combination of geometric and non-geometric errors.
In order to simulate this error, we chose to change the length of two of the six total axes of the NS robot as follows:

- Axis 2, lengthened by 2 [mm]
- Axis 4, elongated by 3 [mm]

These modifications respect the order of magnitude of the error actually manifested by an industrial manipulator during its operations.

These changes were made by adjusting the two length parameters within the script programmed for the creation of the digital twin of the NS robot.

As before, from the .m file we moved to the translation in PDL language.

Now we have to make sure that this 'new' machine does the same 1000 moves as the previously launched movement program.

The same PDL that was written previously was then loaded on RoboSim, with only one difference:

- Rewritten in the PDL all the 1000 moves in joint coordinates obtained from the function POS TO JOINT carried out previously

The following function was inserted into the PDL:

- JOINT TO POS: thanks to this function, the joint coordinates (Theta1, Theta2, Theta3, Theta4, Theta5, Theta6) are transformed into Cartesian coordinates (X, Y, Z, 0, 90, 0).

In this way a second vector ***P_output*** was created containing the REAL spatial points (with therefore a discrepancy, called error, with respect to the theoretical points given in the program). In other words, it is as if these thousand points had been acquired using Krypton's camera in reality.

As this paper’s goal is to develop a preliminary analysis for this new technology, creating this error in simulation has led to significant cost savings compared to using Krypton for the acquisition of 1000 poses.

5.3 Analysis of the Algorithm

The Matlab code that has been used was created by mr Quin-Yu Zhu and Dr. Guang-Bin Huang of the Nanyang Technological University of Singapore. You may find the entire script in Appendix B.

There will be 2 main phases: the training phase and the testing phase. For each phase there is a specific Matlab function. Here under the 2 steps are broke down and analyzed.

5.3.1 Training Phase:

During this phase the network sets its Bias and weights in order to perform a linear regression as accurate as possible between the output data with the corresponding inputs. The Matlab function will require:

Input:

- *TrainingData_File* : Filename of training data set in .txt format.
- *Elm_Type* : insert 0 for regression; 1 for (both binary and multi-classes) classification.
- *NumberofHiddenNeurons* : Number of hidden neurons assigned to the ELM.
- *ActivationFunction* : Type of activation function choose 'sig' for Sigmoidal function.

Output:

- *TrainingTime* : Time (seconds) spent on training ELM.
- *TrainingAccuracy* : Training accuracy calculated as the RMSE for the regression case.

Here under you may find a screenshot of the particular lines of code of the training accuracy calculation.

```
%%%%%%%%%% Calculate the training accuracy
Y=(H' * OutputWeight)';
if Elm_Type == REGRESSION
    TrainingAccuracy=sqrt(mse(T - Y))
    output=Y;
end
```

Where H'= Hessian matrix of weights?
Y= Output of the ELM training
T= Train data

At the end of the training phase the function will save the trained neural network as a 'elm_model.mat' file in the main directory. You may hence use it with different testing datasets.

5.3.2 Testing Phase:

During this phase we use the network already trained to predict the 200 output instances. In our case each instance will be the error (e.g. deltaX) associated with each spatial coordinate given the 200 known inputs (e.g. Xth)

Input:

- *TestingData_File* : Filename of testing data set in .txt format.

Output:

- *TestingTime* : Time (seconds) spent on predicting all testing data.
- *TestingAccuracy* : Testing accuracy again calculated with the RMSE algorithm
- *Output.m:* row vector with all the forecasted instances

```
TY=(H_test' * OutputWeight)';
end_time_test=cputime;
TestingTime=end_time_test-start_time_test

if Elm_Type == REGRESSION
    TestingAccuracy=sqrt(mse(TV.T - TY))
    output=TY;
```

Where TV.T= testing data

5.4 Datasets creation

Once the dataset of 1000 theoretical positions (Xth,Yth,Zth) and the dataset of the corresponding real positions (X,Y,Z) have been created (both datasets are reported in Appendix1) we can proceed to the creation of the Training dataset (800 poses) and of the Testing dataset (200 poses) for each single simulation.

Therefore We will create 8 different datasets: two for the X coordinate, two for the Y coordinate two for the Z coordinate and finally two for the Absolute Error forecast.

Each dataset will be composed as follows.

As you may appreciate form the charts, the absolute error simulation will be the only one with multiple inputs and mono output. It is important to evaluate this regression problem too in order to verify the goodness and broadness of this calibration method.

5.4.1 Layout Dataset X coordinate

Training Dataset X coordinate	
Target [mm]	Input Attribute [m]
ΔX_1	X_{1th}
ΔX_2	X_{2th}
ΔX_3	X_{3th}
ΔX_4	X_{4th}
ΔX_5	X_{5th}
ΔX_6	X_{6th}
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
ΔX_{798}	X_{798th}
ΔX_{799}	X_{799th}
ΔX_{800}	X_{800th}

Testing Dataset X coordinate	
Target [mm]	Input Attribute [m]
ΔX_{801}	X_{801th}
ΔX_{802}	X_{802th}
ΔX_{803}	X_{803th}
ΔX_{804}	X_{804th}
ΔX_{805}	X_{805th}
ΔX_{806}	X_{806th}
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
ΔX_{898}	X_{898th}
ΔX_{899}	X_{899th}
ΔX_{1000}	X_{1000th}

5.4.2 Layout Dataset Y coordinate

Training Dataset Y coordinate	
Target [mm]	Input Attribute [m]
ΔY_1	Y_{1th}
ΔY_2	Y_{2th}
ΔY_3	Y_{3th}
ΔY_4	Y_{4th}
ΔY_5	Y_{5th}
ΔY_6	Y_{6th}
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
ΔY_{798}	Y_{798th}
ΔY_{799}	Y_{799th}
ΔY_{800}	Y_{800th}

Testing Dataset Y coordinate	
Target [mm]	Input Attribute [m]
ΔY_{801}	Y_{801th}
ΔY_{802}	Y_{802th}
ΔY_{803}	Y_{803th}
ΔY_{804}	Y_{804th}
ΔY_{805}	Y_{805th}
ΔY_{806}	Y_{806th}
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
ΔY_{998}	Y_{998th}
ΔY_{999}	Y_{999th}
ΔY_{1000}	Y_{1000th}

5.4.3 Layout Dataset Z coordinate

Training Dataset Z coordinate	
Target [mm]	Input Attribute [m]
ΔZ_1	Z_{1th}
ΔZ_2	Z_{2th}
ΔZ_3	Z_{3th}
ΔZ_4	Z_{4th}
ΔZ_5	Z_{5th}
ΔZ_6	Z_{6th}
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
ΔZ_{798}	Z_{798th}
ΔZ_{799}	Z_{799th}
ΔZ_{800}	Z_{800th}

Testing Dataset Z coordinate	
Target [mm]	Input Attribute [m]
ΔZ_{801}	Z_{801th}
ΔZ_{802}	Z_{802th}
ΔZ_{803}	Z_{803th}
ΔZ_{804}	Z_{804th}
ΔZ_{805}	Z_{805th}
ΔZ_{806}	Z_{806th}
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
ΔZ_{998}	Z_{998th}
ΔZ_{999}	Z_{999th}
ΔZ_{1000}	Z_{1000th}

5.4.4 Layout Dataset Absolute Error

Training Dataset Absolute Error E			
Target [mm]	Input Attribute [m]	Input Attribute [m]	Input Attribute [m]
E_1	X_{1th}	Y_{1th}	Z_{1th}
E_2	X_{2th}	Y_{2th}	Z_{2th}
E_3	X_{3th}	Y_{3th}	Z_{3th}
E_4	X_{4th}	Y_{4th}	Z_{4th}
E_5	X_{5th}	Y_{5th}	Z_{5th}
E_6	X_{6th}	Y_{6th}	Z_{6th}
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
E_{798}	X_{798th}	Y_{798th}	Z_{798th}
E_{799}	X_{799th}	Y_{799th}	Z_{799th}
E_{800}	X_{800th}	Y_{800th}	Z_{800th}

Testing Dataset Absolute Error E			
Target [mm]	Input Attribute [m]	Input Attribute [m]	Input Attribute [m]
E_{801}	X_{801th}	Y_{801th}	Z_{801th}
E_{802}	X_{802th}	Y_{802th}	Z_{802th}
E_{803}	X_{803th}	Y_{803th}	Z_{803th}
E_{804}	X_{804th}	Y_{804th}	Z_{804th}
E_{805}	X_{805th}	Y_{805th}	Z_{805th}
E_{806}	X_{806th}	Y_{806th}	Z_{806th}
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
E_{998}	X_{998th}	Y_{998th}	Z_{998th}
E_{999}	X_{999th}	Y_{999th}	Z_{999th}
E_{1000}	X_{1000th}	Y_{1000th}	Z_{1000th}

6) Results and Discussion

The results for each simulation will be displayed as follows:

- Training Accuracy= Root Mean Square Error
- Training Time [s]
- Testing Accuracy = Root Mean Square Error
- Testing Time [s]
- Plot of the relative (or absolute) error of prediction that the neural network displayed, calculated as follows:

$$(NNerror) = abs(Validation\ Instance_i - NNoutput_i)\ ,\quad i=1.2.3\ldots,200$$

The closest the error is to 0 the better we may compensate the machine.
Another important aspect is the computational time: optimum result will be to obtain the most accurate prediction as possible minimizing the computational time.

6.1 Simulation 1: ΔX prevision

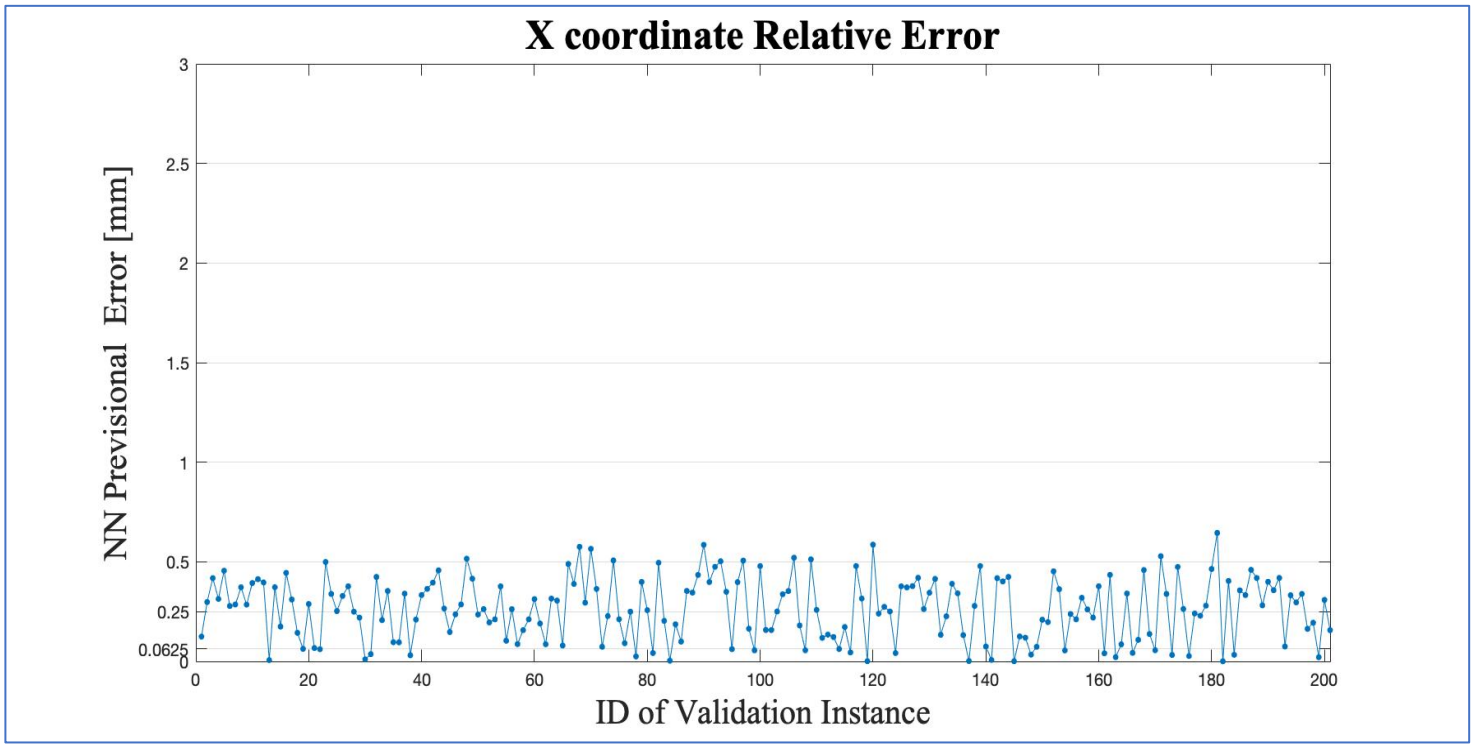


figure 20: X coordinate Realative Error Prevision

- Training Accuracy: 0.3060
- Training time: 0.04

- Testing Accuracy: 0.3095
- Testing Time: 0.04

6.2 Simulation 2: ΔY prevision

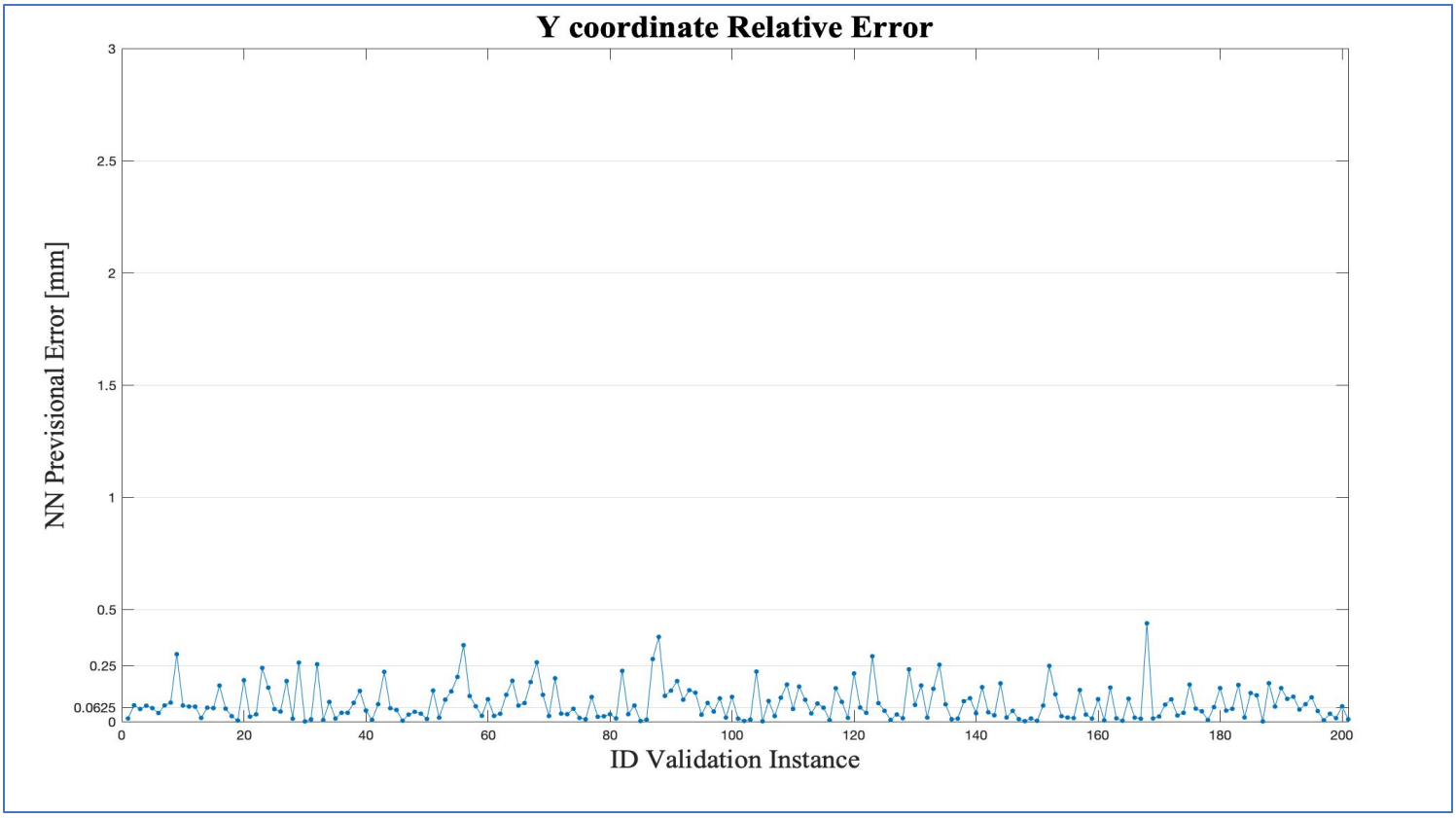


figure 21: Y coordinate Relative Error Prevision

- Training Accuracy:0.1085 [mm]
- Training time: 0.04 [s]

- Testing Accuracy:0.1138 [mm]
- Testing Time: 0.02 [s]

6.3 Simulation 3: ΔZ prevision

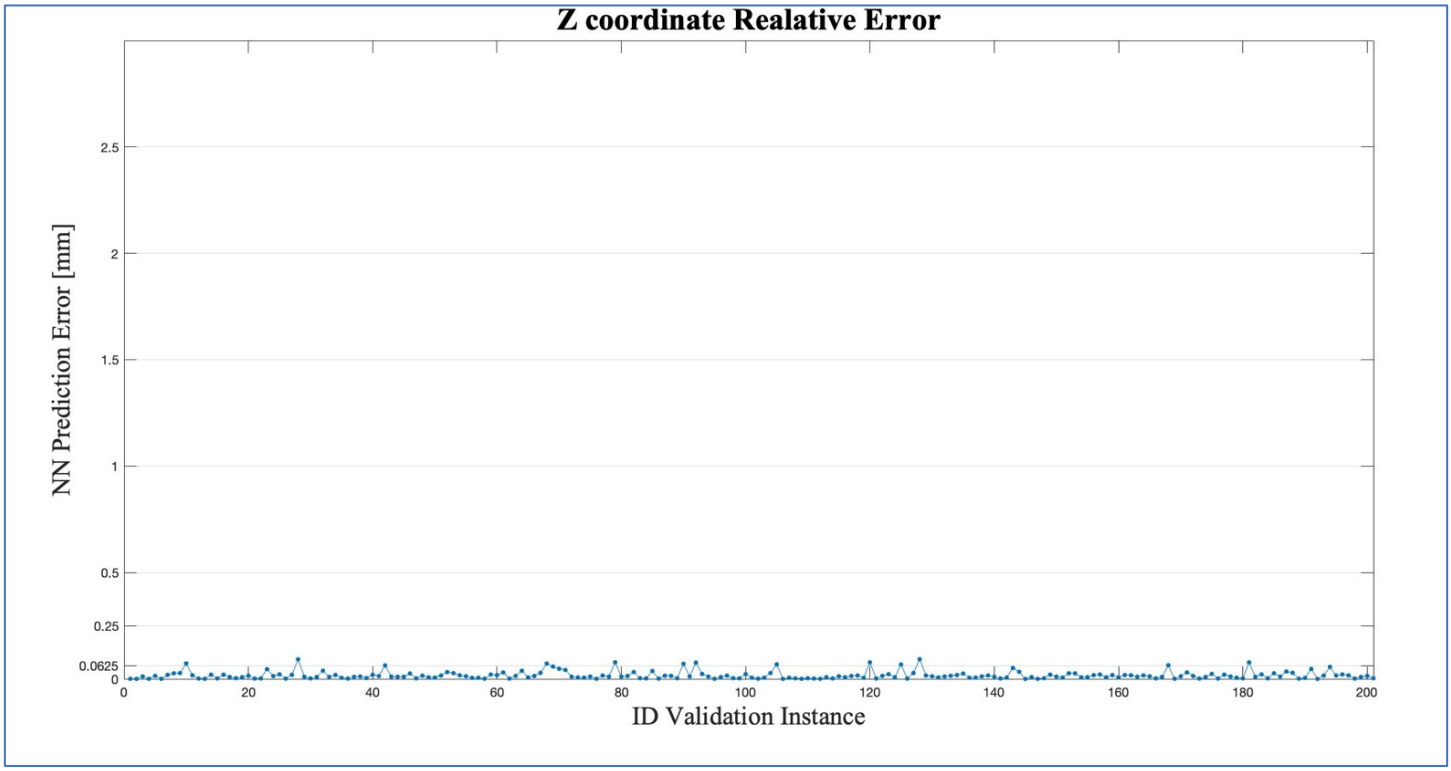


figure 22: Z Coordinate Realtive Error Prevision

- Training Accuracy:0.0296 [mm]
- Training time: 0.04 [s]

- Testing Accuracy: 0.1138 [mm]
- Testing Time: 0.02 [s]

6.4 Simulation 4: Absolute Error prevision

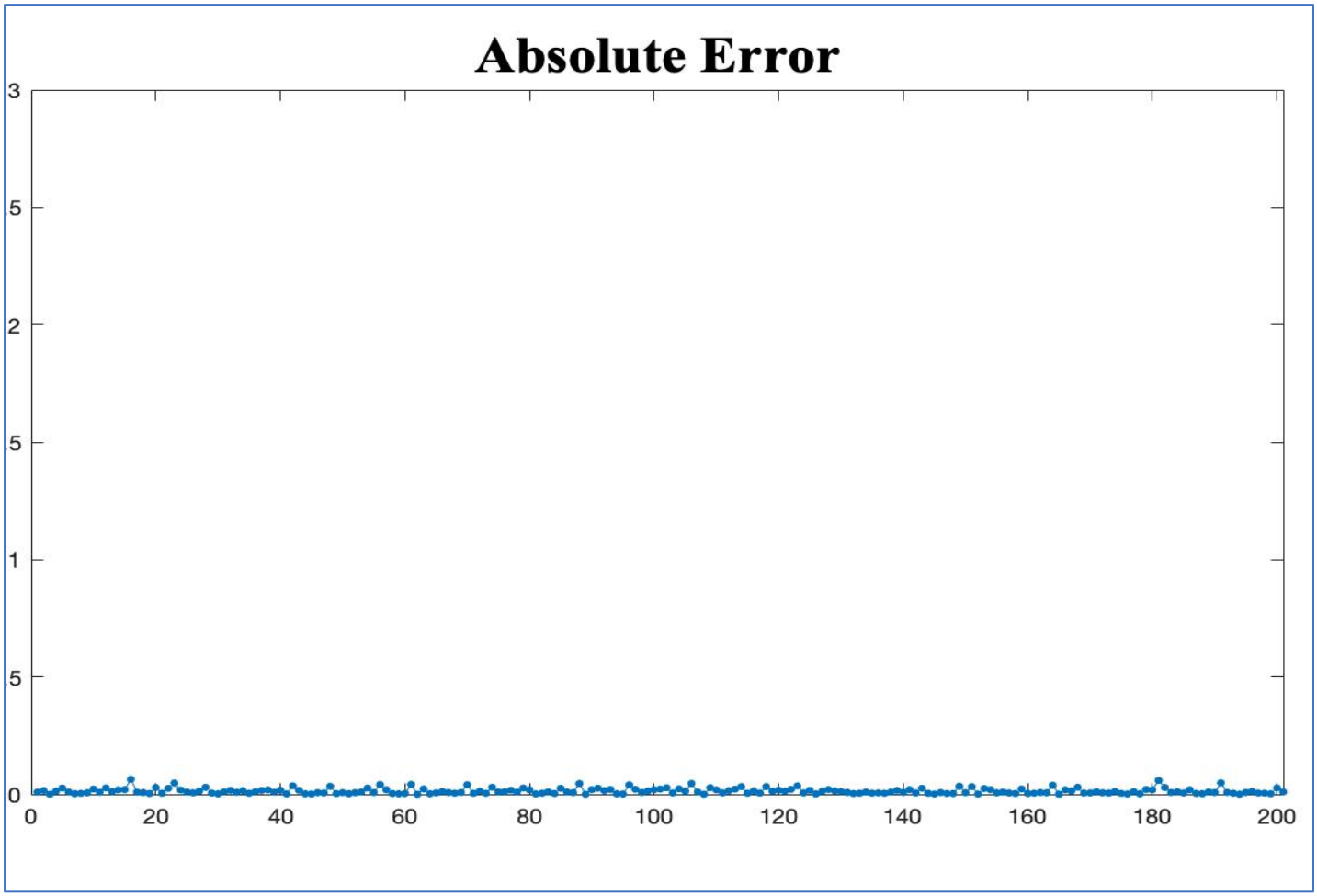


figure 23: Absolute Error Prevision

- Training Accuracy: 0.0117 [mm]
- Training time: 0.01 [s]

- Testing Accuracy: 0.0118 [mm]
- Testing Time: 0.03 [s]

7) Conclusions and Future Work

The purpose of this thesis was to evaluate the efficiency of a model free calibration of a robotic arm carried out through a neural network. For the training of such a network it was chosen to fix to 20 the neurons of the intermediate layer and to use a learning pattern called or ELM in which there are no iterative processes.

The network predicted the error in all 4 simulations with very high accuracy, thus achieving the desired goal. The only oscillation of the results is the one associated to the prediction of the relative error associated to the spatial coordinate x. This type of deviation from the other simulations is physiological of the technology of the neural network. At the moment the scientific world agrees on the goodness and the quality of the neural networks in the resolution of problems of linear regression or classification, but it has not yet understood the 'way of reasoning' of such networks, that is it is not fully understood according to which criterion the weights and the bias are adjusted by the algorithm itself.

Anyway, since the RMSE maintains the same order of magnitude of its variables, with the accuracy values obtained in the simulations, it is possible to compensate the spatial error of the robotic arm with extreme precision. Moreover, the calculation time of both training and testing is close to 0, so that the output of this algorithm can be considered as 'Real Time output'.

The next steps to this thesis work will be the experimental validations of this simulation. You will have to compensate the real robotic arm with the parameters predicted by the neural network and then measure with an infrared camera the real position of the end-effector of the robotic arm again evaluating the gap between theoretical and real position. If this difference is zero, the compensation will be the achievable best.

Appendix A

```
PROGRAM NS_Kine_eval_POS &PA

VAR vi_Idx : INTEGER
VAR wr_Pos : ARRAY[1000] OF POSITION
VAR wr_JointPos : ARRAY[1000] OF JOINTPOS FOR ARM[1]
VAR lun_file : INTEGER NOSAVE

BEGIN

  -- Riferimenti
  $TOOL := POS(0, 0, 0, 0, 0, 0, '')
  $BASE := POS(0, 0, 0, 0, 0, 0, '')
  $UFRAME := POS(0, 0, 0, 0, 0, 30, '')

  -- Apertura del file
  OPEN FILE lun_file ('Kine_eval.txt', 'w')

  -- punti cartesiani
  wr_Pos[1] := POS(1354, -18, 764, 0, 90, 0, 'W')
  wr_Pos[2] := POS(1095, -397, 1011, 0, 90, 0, 'W')
  wr_Pos[3] := POS(1114, 209, 1268, 0, 90, 0, 'W')
  wr_Pos[4] := POS(1180, -76, 748, 0, 90, 0, 'W')
  wr_Pos[5] := POS(1010, 333, 1125, 0, 90, 0, 'W')


 wr_Pos[987] := POS(1267, -360, 1226, 0, 90, 0, 'W')
 wr_Pos[988] := POS(905, 327, 1001, 0, 90, 0, 'W')
 wr_Pos[989] := POS(877, -489, 1161, 0, 90, 0, 'W')
 wr_Pos[990] := POS(801, 314, 1096, 0, 90, 0, 'W')
 wr_Pos[991] := POS(968, -253, 1149, 0, 90, 0, 'W')
 wr_Pos[992] := POS(1153, -268, 896, 0, 90, 0, 'W')
 wr_Pos[993] := POS(824, -190, 1113, 0, 90, 0, 'W')
 wr_Pos[994] := POS(1085, -371, 1100, 0, 90, 0, 'W')
 wr_Pos[995] := POS(1193, 87, 1175, 0, 90, 0, 'W')
 wr_Pos[996] := POS(1333, 35, 688, 0, 90, 0, 'W')
 wr_Pos[997] := POS(1069, 179, 667, 0, 90, 0, 'W')
 wr_Pos[998] := POS(1277, -410, 716, 0, 90, 0, 'W')
 wr_Pos[999] := POS(886, -132, 606, 0, 90, 0, 'W')
 wr_Pos[1000] := POS(1070, -57, 973, 0, 90, 0, 'W')

 -- punti giunti
 FOR vi_Idx := 1 TO 1000 DO
   POS_TO_JNTP(wr_Pos[vi_Idx], wr_JointPos[vi_Idx], $BASE, $TOOL, $UFRAME)
 ENDFOR


 -- stampe video
 FOR vi_Idx := 1 TO 1000 DO
   -- video
   WRITE LUN_CRT ('Punto ', vi_Idx, NL)
   WRITE LUN_CRT ('- Position: ', wr_Pos[vi_Idx], NL)
   WRITE LUN_CRT ('- Joint   : ', wr_JointPos[vi_Idx], NL)
   WRITE LUN_CRT ('----------------------', NL)
   -- su file
   WRITE lun_file ('Punto ', vi_Idx, NL)
   WRITE lun_file ('- Position: ', wr_Pos[vi_Idx], NL)
   WRITE lun_file ('- Joint   : ', wr_JointPos[vi_Idx], NL)
   WRITE lun_file ('----------------------', NL)
   -- movimento sul punto
   MOVE TO (wr_JointPos[vi_Idx])
   DELAY 500
   -- hold
 ENDFOR

 -- Chiusura del file
 CLOSE FILE lun_file

END NS_Kine_eval_POS
```

```
PROGRAM NS_DirectKine_eval NOHOLD, &PA

VAR vi_Idx      : INTEGER
VAR wr_Pos      : ARRAY[1000] OF POSITION
VAR wr_JointPos : ARRAY[1000] OF JOINTPOS FOR ARM[1]
VAR lun_file    : INTEGER NOSAVE

BEGIN

-- Riferimenti
  $TOOL   := POS(0, 0, 0, 0, 0, 0, '')
  $BASE   := POS(0, 0, 0, 0, 0, 0, '')
  $UFRAME := POS(0, 0, 0, 0, 0, 30, '')

-- Apertura del file
  OPEN FILE lun_file ('DirectKine_eval.txt', 'w')

-- punti joint
   JNT(wr_JointPos[   1],  -29.154, 19.128, -119.231, 1.132, -48.365, -0.752)
   JNT(wr_JointPos[   2], -7.533, -5.072, -127.309, 37.785, -38.588, -31.215)
   JNT(wr_JointPos[   3], -42.051, -12.303, -115.941, -42.157, -18.124, 40.711)
   JNT(wr_JointPos[   4], -25.840, 7.832, -134.633, 5.240, -52.581, -3.190)
   JNT(wr_JointPos[   5], -50.835, -16.741, -129.147, -44.955, -30.225, 40.785)
   JNT(wr_JointPos[   6], -37.567, -4.658, -107.404, -31.051, -14.791, 30.205)


   JNT(wr_JointPos[ 999], -20.031, -2.440, -159.629, 10.795, -67.553, -4.164)
   JNT(wr_JointPos[1000], -26.511, -12.817, -136.730, 6.236, -34.071, -5.172)

-- punti giunti
  FOR vi_Idx := 1 TO 1000 DO
    JNTP_TO_POS(wr_JointPos[vi_Idx], wr_Pos[vi_Idx], $BASE, $TOOL, $UFRAME)
  ENDFOR

-- stampe video
  FOR vi_Idx := 1 TO 1000 DO
  -- video
    WRITE LUN_CRT('Punto ',vi_Idx, NL)
    WRITE LUN_CRT('- Joint   : ', wr_JointPos[vi_Idx], NL)
    WRITE LUN_CRT('- Position: ', wr_Pos[vi_Idx], NL)
    WRITE LUN_CRT('----------------------', NL)
  -- su file
    WRITE lun_file('Punto ',vi_Idx, NL)
    WRITE lun_file('- Joint   : ', wr_JointPos[vi_Idx], NL)
    WRITE lun_file('- Position: ', wr_Pos[vi_Idx], NL)
    WRITE lun_file('----------------------', NL)
  ENDFOR

-- Chiusura del file
  CLOSE FILE lun_file
```

Appendix B

ELM Train

```
function [TrainingTime,TrainingAccuracy] = elm_train(Training_Dataset,0,20,sig)

% Usage: elm_train(TrainingData_File, Elm_Type, NumberofHiddenNeurons,
ActivationFunction)
% OR:    [TrainingTime, TrainingAccuracy] = elm_train(TrainingData_File,
Elm_Type, NumberofHiddenNeurons, ActivationFunction)
%
% Input:
% TrainingData_File     - Filename of training data set
% Elm_Type              - 0 for regression; 1 for (both binary and multi-
classes) classification
% NumberofHiddenNeurons - Number of hidden neurons assigned to the ELM
% ActivationFunction    - Type of activation function:
%                           'sig' for Sigmoidal function
%                           'sin' for Sine function
%                           'hardlim' for Hardlim function
%
% Output:
% TrainingTime          - Time (seconds) spent on training ELM
% TrainingAccuracy      - Training accuracy:
%                           RMSE for regression or correct classification rate
for classification
%
% MULTI-CLASSE CLASSIFICATION: NUMBER OF OUTPUT NEURONS WILL BE AUTOMATICALLY
SET EQUAL TO NUMBER OF CLASSES
% FOR EXAMPLE, if there are 7 classes in all, there will have 7 output
% neurons; neuron 5 has the highest output means input belongs to 5-th class
%
% Sample1 regression: [TrainingTime, TrainingAccuracy, TestingAccuracy] =
elm_train('sinc_train', 0, 20, 'sig')
% Sample2 classification: elm_train('diabetes_train', 1, 20, 'sig')
%
    %%%%    Authors:    MR QIN-YU ZHU AND DR GUANG-BIN HUANG
    %%%%    NANYANG TECHNOLOGICAL UNIVERSITY, SINGAPORE
    %%%%    EMAIL:      EGBHUANG@NTU.EDU.SG; GBHUANG@IEEE.ORG
    %%%%    WEBSITE:    http://www.ntu.edu.sg/eee/icis/cv/egbhuang.htm
    %%%%    DATE:       APRIL 2004

%%%%%%%%%%% Macro definition
REGRESSION=0;
CLASSIFIER=1;

%%%%%%%%%%% Load training dataset
train_data=load(Training_Dataset);
T=train_data(:,1)';
P=train_data(:,2:size(train_data,2))';
clear train_data;                                   %   Release raw training
data array

NumberofTrainingData=size(P,2);
NumberofInputNeurons=size(P,1);

if Elm_Type~=REGRESSION
    %%%%%%%%%%% Preprocessing the data of classification
```

```
    sorted_target=sort(T,2);
    label=zeros(1,1);                               %   Find and save in 'label'
class label from training and testing data sets
    label(1,1)=sorted_target(1,1);
    j=1;
    for i = 2:NumberofTrainingData
        if sorted_target(1,i) ~= label(1,j)
            j=j+1;
            label(1,j) = sorted_target(1,i);
        end
    end
    number_class=j;
    NumberofOutputNeurons=number_class;

    %%%%%%%%%% Processing the targets of training
    temp_T=zeros(NumberofOutputNeurons, NumberofTrainingData);
    for i = 1:NumberofTrainingData
        for j = 1:number_class
            if label(1,j) == T(1,i)
                break;
            end
        end
        temp_T(j,i)=1;
    end
    T=temp_T*2-1;
end                                                 %   end if of Elm_Type

%%%%%%%%%%% Calculate weights & biases
start_time_train=cputime;

%%%%%%%%%%% Random generate input weights InputWeight (w_i) and biases
BiasofHiddenNeurons (b_i) of hidden neurons
InputWeight=rand(NumberofHiddenNeurons,NumberofInputNeurons)*2-1;
BiasofHiddenNeurons=rand(NumberofHiddenNeurons,1);
tempH=InputWeight*P;
clear P;                                            %   Release input of
training data
ind=ones(1,NumberofTrainingData);
BiasMatrix=BiasofHiddenNeurons(:,ind);              %   Extend the bias matrix
BiasofHiddenNeurons to match the demention of H
tempH=tempH+BiasMatrix;

%%%%%%%%%%% Calculate hidden neuron output matrix H
switch lower(ActivationFunction)
    case {'sig','sigmoid'}
        %%%%%%%% Sigmoid
        H = 1 ./ (1 + exp(-tempH));
    case {'sin','sine'}
        %%%%%%%% Sine
        H = sin(tempH);
    case {'hardlim'}
        %%%%%%%% Hard Limit
        H = hardlim(tempH);
        %%%%%%%% More activation functions can be added here
end
clear tempH;                                        %   Release the temparary
array for calculation of hidden neuron output matrix H

%%%%%%%%%%% Calculate output weights OutputWeight (beta_i)
OutputWeight=pinv(H') * T';
end_time_train=cputime;
TrainingTime=end_time_train-start_time_train        %   Calculate CPU time
(seconds) spent for training ELM
```

```
%%%%%%%%%% Calculate the training accuracy
Y=(H' * OutputWeight)';                             %   Y: the actual output of
the training data
if Elm_Type == REGRESSION
    TrainingAccuracy=sqrt(mse(T - Y))               %   Calculate training
accuracy (RMSE) for regression case
    output=Y;
end
clear H;

if Elm_Type == CLASSIFIER
%%%%%%%%%% Calculate training & testing classification accuracy
    MissClassificationRate_Training=0;

    for i = 1 : size(T, 2)
        [x, label_index_expected]=max(T(:,i));
        [x, label_index_actual]=max(Y(:,i));
        output(i)=label(label_index_actual);
        if label_index_actual~=label_index_expected
            MissClassificationRate_Training=MissClassificationRate_Training+1;
        end
    end
    TrainingAccuracy=1-MissClassificationRate_Training/NumberofTrainingData
end

if Elm_Type~=REGRESSION
    save('elm_model', 'NumberofInputNeurons', 'NumberofOutputNeurons',
'InputWeight', 'BiasofHiddenNeurons', 'OutputWeight', 'ActivationFunction',
'label', 'Elm_Type');
else
    save('elm_model', 'InputWeight', 'BiasofHiddenNeurons', 'OutputWeight',
'ActivationFunction', 'Elm_Type');
end
```

ELM Predict

```
function [TestingTime, TestingAccuracy] = elm_predict(TestingData_File)

% Usage: elm_predict(TestingData_File)
% OR:    [TestingTime, TestingAccuracy] = elm_predict(TestingData_File)
%
% Input:
% TestingData_File      - Filename of testing data set
%
% Output:
% TestingTime           - Time (seconds) spent on predicting ALL testing data
% TestingAccuracy       - Testing accuracy:
%                           RMSE for regression or correct classification rate
for classification
%
% MULTI-CLASSE CLASSIFICATION: NUMBER OF OUTPUT NEURONS WILL BE AUTOMATICALLY
SET EQUAL TO NUMBER OF CLASSES
% FOR EXAMPLE, if there are 7 classes in all, there will have 7 output
% neurons; neuron 5 has the highest output means input belongs to 5-th class
%
% Sample1 regression: [TestingTime, TestingAccuracy] = elm_predict('sinc_test')
% Sample2 classification: elm_predict('diabetes_test')
%
    %%%%    Authors:    MR QIN-YU ZHU AND DR GUANG-BIN HUANG
    %%%%    NANYANG TECHNOLOGICAL UNIVERSITY, SINGAPORE
    %%%%    EMAIL:      EGBHUANG@NTU.EDU.SG; GBHUANG@IEEE.ORG
    %%%%    WEBSITE:    http://www.ntu.edu.sg/eee/icis/cv/egbhuang.htm
    %%%%    DATE:       APRIL 2004

%%%%%%%%%%% Macro definition
REGRESSION=0;
CLASSIFIER=1;

%%%%%%%%%%% Load testing dataset
test_data=load(TestingData_File);
TV.T=test_data(:,1)';
TV.P=test_data(:,2:size(test_data,2))';
clear test_data;                                    %   Release raw testing data
array

NumberofTestingData=size(TV.P,2);

load elm_model.mat;

if Elm_Type~=REGRESSION

    %%%%%%%%%% Processing the targets of testing
    temp_TV_T=zeros(NumberofOutputNeurons, NumberofTestingData);
    for i = 1:NumberofTestingData
        for j = 1:size(label,2)
            if label(1,j) == TV.T(1,i)
                break;
            end
        end
        temp_TV_T(j,i)=1;
    end
    TV.T=temp_TV_T*2-1;
```

```
end                                                 %   end if of Elm_Type

%%%%%%%%%% Calculate the output of testing input
start_time_test=cputime;
tempH_test=InputWeight*TV.P;
clear TV.P;             %   Release input of testing data
ind=ones(1,NumberofTestingData);
BiasMatrix=BiasofHiddenNeurons(:,ind);              %   Extend the bias matrix
BiasofHiddenNeurons to match the demention of H
tempH_test=tempH_test + BiasMatrix;
switch lower(ActivationFunction)
    case {'sig','sigmoid'}
        %%%%%%%% Sigmoid
        H_test = 1 ./ (1 + exp(-tempH_test));
    case {'sin','sine'}
        %%%%%%%% Sine
        H_test = sin(tempH_test);
    case {'hardlim'}
        %%%%%%%% Hard Limit
        H_test = hardlim(tempH_test);
        %%%%%%%% More activation functions can be added here
end
TY=(H_test' * OutputWeight)';                       %   TY: the actual output of
the testing data
end_time_test=cputime;
TestingTime=end_time_test-start_time_test           %   Calculate CPU time
(seconds) spent by ELM predicting the whole testing data

if Elm_Type == REGRESSION
    TestingAccuracy=sqrt(mse(TV.T - TY))            %   Calculate testing
accuracy (RMSE) for regression case
    output=TY;
end

if Elm_Type == CLASSIFIER
%%%%%%%%%% Calculate training & testing classification accuracy
    MissClassificationRate_Testing=0;

    for i = 1 : size(TV.T, 2)
        [x, label_index_expected]=max(TV.T(:,i));
        [x, label_index_actual]=max(TY(:,i));
        output(i)=label(label_index_actual);
        if label_index_actual~=label_index_expected
            MissClassificationRate_Testing=MissClassificationRate_Testing+1;
        end
    end
    TestingAccuracy=1-MissClassificationRate_Testing/NumberofTestingData
end

save('elm_output','output');
```

References

1. Peijiang Yuan, Dongdong Chen, Tianmiao Wang, Shuangqian Cao, Ying Cai and Lei Xue, *A compensation method based on extreme learning machine to enhance absolute position accuracy for aviation drilling robot*.

2. Giovanni Legnani, Pierluigi Magnani, Monica Tiboni, *An innovative approach for the Calibration of Industrial Manipulators based on Neural Networks.*

3. Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo, *Robotics Modelling Planning and Control.*

4. Stuart J. Russell and Peter Norvig, *Artificial Intelligence*: *A Modern Approach Third Edition*

5. *Krypton help Pages on K400/K600 Harware & Software Guides*

6. Guang-Bin Huang, *Extreme Learning Machines (ELM), Filling the Gap between Frank Rosenblatt's Dream and John von Neumann's Puzzle?*

7. Marvin L*, Neural Networks with Matlab*

8. COMAU s.p.a, *Comau Robotics Product Range*

9. Giovanni Legnani, Monica Tiboni *Improved Kinematics Calibration of Industrial Robots by Neural Networks.*

10. Praveen Kumar M, Denis Ashok S, *Artificial neural network based geometric error correction model for enhancing positioning accuracy of a robotic sewing manipulator*