POLITECNICO DI TORINO

Master's Degree Course in Mathematical Engineering
Academic year 2023/2024
Graduation session July 2024

Neural Network-Based Calibration of 6D Force-Torque Sensors in Humanoid Robots

Supervisors
Prof. Luigi Preziosi (PoliTo)
Prof. Marta Zoppello (PoliTo)
Ph.D. Daniele Pucci (IIT)
Ph.D. Giulio Romualdi (IIT)

Candidate
Filippo Passerini

Abstract

Precise force-torque sensing plays a fundamental role in humanoid robot control, impacting tasks from manipulation to movement. A conventional technique for calibrating 6D force-torque sensors linear regression, which in some cases could not give good results, especially in the field of robotics. The goal of this thesis is to improve the accuracy and efficiency of 6D force-torque sensor calibration in humanoid robotics by introducing a neural network-based method for *in-situ* calibration. We employ machine learning methods, particularly neural networks, to simulate the cross-axis coupling effects and nonlinearities present in the sensors. As part of the process, data are collected during in-situ experiments, meaning that the robot has the sensor installed already. Such data are used to train a neural network to predict the real *wrench* (i.e. a $6D$ vector containing the components of force and torque) from the raw sensor readings and also other quantities.

The performance of the proposed neural network calibration method is evaluated against traditional calibration techniques. Results indicate a significant improvement in sensor accuracy.

This thesis contributes to the advancement of humanoid robotics by providing an efficient and accurate method for force-torque sensor calibration. The integration of neural networks not only simplifies the calibration process but also paves the way for more intelligent and autonomous robotic systems capable of complex interactions with their surroundings.

Contents

List of Figures iii

List of Tables vi

Prologue 1

I Background 4

1 Force-torque sensors 5
- 1.1 Force Sensing Technologies 6
- 1.2 Piezoresistive Strain Gauges 8
- 1.3 Wheatstone bridge 11
- 1.4 Conclusions 12

2 Neural Networks 15
- 2.1 Introduction to Neural Networks 15
- 2.2 Training of a NN through supervised learning 19
  - 2.2.1 Forward pass 19
  - 2.2.2 Loss computation 20
  - 2.2.3 Backward Pass 21
  - 2.2.4 Weight Update 22
- 2.3 Theoretical Foundations 25
  - 2.3.1 Universal Approximation Theorems 25
- 2.4 Conclusions 26

3 Basics of rigid multibody dynamics 27
- 3.1 Some mathematical definitions 27
- 3.2 Points and frames 30

3.3 Velocities 30
3.4 Accelerations 33
3.5 Forces and Torques 33
3.6 Equations of dynamics 34
3.7 Multibody equations of dynamics 35
3.7.1 Introduction to multibody systems 35
3.7.2 Position and velocity of the joints 37
3.7.3 Link pose 37
3.7.4 Link velocity 38
3.7.5 Dynamics 39
3.8 Conclusions 40

II Contribution **42**

4 Data collection and postprocessing **43**
4.1 Collection of the data in-situ 43
4.2 Postprocessing of the data 46
4.3 Conclusions 47

5 Neural Networks for FT modelling **48**
5.1 Datasets used 49
5.2 Framework and libraries used for development of NN models 49
5.3 Inputs of the NNs 49
5.4 Definition of the loss functions 50
5.4.1 Loss function for the FTs on the legs 51
5.4.2 Loss function for the FTs on the feet 51
5.5 Models for FT on right leg 54
5.5.1 Architecture of the model 54
5.5.2 Results for model with 13 inputs 55
5.5.3 Results for model with 7 inputs 55
5.5.4 Comparison with the results obtained by using polynomial models 61
5.6 Models for FT on left leg 63
5.6.1 Architecture of the model 63
5.6.2 Results for model with 13 inputs 64
5.6.3 Results for model with 7 inputs 64
5.6.4 Comparison with the results obtained by using polynomial models 68

5.7 Models for FTs on right foot 72
5.7.1 Architecture of the models 72
5.7.2 Results for model with 13 inputs 73
5.7.3 Results for model with 7 inputs 73
5.8 Models for FTs on left foot 82
5.8.1 Architecture of the models 83
5.8.2 Results for model with 13 inputs 85
5.8.3 Results for model with 7 inputs 85
5.9 Conclusions 94

6 Deployment and tests of the models on real robot **96**
6.1 Test A 97
6.2 Test B 97
6.3 Test C 102
6.4 Conclusions 103

7 Conclusions and future perspectives **107**

List of Figures

1.1 Components of $6D$ strain gauge force-torque sensor 6
1.2 Block diagram of sensing module. Credits to [28] 7
1.3 Wire strain gauge (credits to [3]) 8
1.4 Foil strain gauge 9
1.5 Base configuration of Wheatstone bridge. Credits to [5] 11
1.6 From left to right: full bridge, half bridge, quarter bridge. Credits to [5] 12
1.7 Design of the strain gauge $6D$ FT sensor discussed in this work . . . 13

2.1 The synaptic connections between neurons. Credits to [9] 16
2.2 Example of Neural Network. Credits to [27] 16
2.3 Sigmoid and Tanh activation functions 17
2.4 ReLU and Leaky ReLU activation functions 18
2.5 Step and Sign activation functions 18

4.1 The ErgoCub robot 44
4.2 Feet of ErgoCub robot. 46

5.1 Visualization of frames `r_foot_front_ft`, `r_foot_rear_ft`, `r_sole` . 53
5.2 Loss function during training of NN for right leg FT 55
5.3 Comparison of wrenches predicted by NN with expected wrenches and raw values for right leg FT test phase (13 inputs) 56
5.4 Residual obtained in test phase for right leg FT (13 inputs) 57
5.5 Loss function during training of NN for right leg FT 59
5.6 Comparison of wrenches predicted by NN (with 7 inputs) with expected wrenches and raw values for right leg FT test phase 60
5.7 Residual obtained in test phase for right leg FT (7 inputs) 61
5.8 Test phase for right leg FT with polynomial models 63
5.9 Loss function during training of NN for left leg FT 65

5.10 Comparison of wrenches predicted by NN with expected wrenches and raw values for left leg FT test phase (13 inputs) 66
5.11 Residual obtained in test phase for left leg FT (13 inputs) 67
5.12 Loss function during training of NN for left leg FT 67
5.13 Comparison of wrenches predicted by NN (with 7 inputs) with expected wrenches and raw values for left leg FT test phase 69
5.14 Residual obtained in test phase for left leg FT (7 inputs) 70
5.15 Test phase for left leg FT with polynomial models 70
5.16 Train and validation loss functions during training of NN for right foot FTs (13 inputs) 74
5.17 Wrenches on right foot on part of dataset related to pole experiment (13 inputs) 75
5.18 Comparison of the sum of wrenches predicted by the neural network (NN) with the expected and raw values during the right foot FTs test phase (13 inputs) 76
5.19 Residual obtained in test phase for right foot FT (13 inputs) 77
5.20 Train and validation loss functions during training of NN for right foot FTs 78
5.21 Wrenches on right foot on part of dataset related to pole experiment (7 inputs) 80
5.22 Comparison of the sum of wrenches predicted by the neural network (NN) with the expected and raw values during the right foot FTs test phase (7 inputs) 81
5.23 Residual obtained in test phase for right foot FTs (7 inputs) 83
5.24 Train and validation loss functions during training of NN for left foot FTs (13 inputs) 86
5.25 Wrenches on left foot on part of dataset related to pole experiment (13 inputs) 87
5.26 Comparison of the sum of wrenches predicted by the neural network (NN) with the expected and raw values during the left foot FTs test phase (13 inputs) 88
5.27 Residual obtained in test phase for left foot FTs (13 inputs) 89
5.28 Train and validation loss functions during training of NN for left foot FTs (7 inputs) 90
5.29 Wrenches on left foot on part of dataset related to pole experiment (7 inputs) 92

5.30 Comparison of the sum of wrenches predicted by the neural network (NN) with the expected and raw values during the left foot FTs test phase (7 inputs) 93
5.31 Residual obtained in test phase for left foot FTs (7 inputs) 94

6.1 Comparison between f_z on left leg FT (without secondary calibration) and right leg FT (with secondary NN calibration) 98
6.2 Angular velocity and linear acceleration of `r_leg_ft along x axis` . 99
6.3 Wrench on legs FTs predicted by NN vs labels (walking experiment) . 100
6.4 Tracking of torque and position on `l_hip_pitch` 105
6.5 Tracking of torque and position on `r_hip_pitch` 106

List of Tables

5.1 Right leg FT: Mean Squared Error (MSE) for Neural Network (NN) and raw data (13 inputs) 57
5.2 Right leg FT: Mean Absolute Error (MAE) for Neural Network (NN) and raw data (13 inputs) 58
5.3 Right leg FT: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data (13 inputs) 58
5.4 Right leg FT with 7 inputs: Mean Squared Error (MSE) for Neural Network (NN) and raw data 59
5.5 loss functionRight leg FT with 7 inputss: Mean Absolute Error (MAE) for Neural Network (NN) and raw data 61
5.6 loss functionRight leg FT with 7 inputss: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data 62
5.7 Left leg FT: Mean Squared Error (MSE) for Neural Network (NN) and raw data (13 inputs) 65
5.8 Left leg FT: Mean Absolute Error (MAE) for Neural Network (NN) and raw data (13 inputs) 68
5.9 Left leg FT: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data (13 inputs) 68
5.10 Left leg FT 7 inputs: Mean Squared Error (MSE) for Neural Network (NN) and raw data 71
5.11 Left leg FT 7 inputs: Mean Absolute Error (MAE) for Neural Network (NN) and raw data 71
5.12 Left leg FT 7 inputs: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data 72
5.13 Sum of wrenches on right foot: Mean Squared Error (MSE) for Neural Network (NN) and raw data (13 inputs) 79
5.14 Sum of wrenches on right foot: Mean Absolute Error (MAE) for Neural Network (NN) and raw data (13 inputs) 79

5.15 Sum of wrenches on right foot: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data (13 inputs) 82
5.16 Right foot FTs 7 inputs: Mean Squared Error (MSE) for Neural Network (NN) and raw data 82
5.17 Right foot FTs 7 inputs: Mean Absolute Error (MAE) for Neural Network (NN) and raw data 83
5.18 Right foot FTs 7 inputs: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data 84
5.19 Sum of wrenches on left foot: Mean Squared Error (MSE) for Neural Network (NN) and raw data (13 inputs) 85
5.20 Sum of wrenches on left foot: Mean Absolute Error (MAE) for Neural Network (NN) and raw data (13 inputs) 89
5.21 Sum of wrenches on left foot: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data (13 inputs) 91
5.22 Left foot FTs 7 inputs: Mean Squared Error (MSE) for Neural Network (NN) and raw data 91
5.23 Left foot FTs 7 inputs: Mean Absolute Error (MAE) for Neural Network (NN) and raw data 94
5.24 Left foot FTs 7 inputs: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data 95

6.1 Mean Squared Error (MSE) for right leg FT (with secondary NN calibration, 7 inputs) and left leg FT (without secondary calibration) . . 101
6.2 Mean Absolute Error (MAE) for right leg FT (with secondary NN calibration, 7 inputs) and left leg FT (without secondary calibration) 101
6.3 Root Mean Squared Error (RMSE) for right leg FT (with secondary NN calibration, 7 inputs) and left leg FT (without secondary calibration) 101

Prologue

The calibration of a force-moment strain gauge sensor plays a crucial role in humanoid robotics. As will be illustrated later, this type of sensors gives as output six electrical potential differences. The calibration process of this sensor consists in finding a relation $f : \mathbb{R}^6 \to \mathbb{R}^6$ which relates the potential differences at the output of a sensor and the *wrench* applied to it. To find this relationship, two steps are necessary: data collection and search for the calibration function f.
As regards the first step, one of the most used methods consists in applying certain known weights to the sensor and recording the output potential differences (collection of data *ex-situ*, i.e. with the sensor not mounted on the robot).
As regards the second step, one of the most used techniques is the linear regression through least squares method. However, such linear models are sometimes too simple, leading to poor results. For this reason, polynomial models of generic order [1] have also been proposed. Furthermore, with the advancement of artificial intelligence, calibration models based on neural networks [15], [16] have also been proposed. However, the output of these sensors is also influenced by other factors, such as temperature. For this reason, it may be useful to also consider temperature in the sensor calibration process [1]. Furthermore, if the data collection and calibration of the sensor takes place ex-situ, there is the risk of a deterioration in performance when it is then mounted on the robot, due for example to the tightening force of the screws [5].
For the reasons stated above, it might be convenient to develop a calibration process in which the data is collected *in-situ* (i.e. with the sensor mounted on the robot), and considering variables such as linear acceleration in the calibration model and angular velocity of the sensor. This thesis develops in this context. The objective of the present work is to develop neural network models for the in-situ calibration of $6D$ force-moment sensors. This will be a secondary calibration, i.e. the network will not take as input the electrical potential differences directly output from the sensor, but the raw wrench output from a linear calibration already implemented on the ErgoCub robot. Furthermore, as mentioned above, these neural network models will

have other variables as input: temperature, angular velocity and linear acceleration of the sensor.

This thesis is divided into two main parts.

In Part I, all the theoretical knowledge necessary to carry out the work will be presented:

- In Chapter 1, the physical and technological principles on which $6D$ force-torque (FT) sensors are based will be presented.
- In Chapter 2, the basics of artificial neural networks will be illustrated; this knowledge will allow us to develop the necessary understanding to build calibration models for FTs.
- Finally, in Chapter 3, some basic knowledge of multibody dynamics will be exposed; these theoretical bases will be useful especially when neural network models are developed, in particular to define the *loss function*.

Part II will contain the development of the original contribution of this work:

- Chapter 4 will deal with the collection and postprocessing of the data necessary for the *training* of neural networks (NN).
- Chapter 5 will discuss the development of the NN models and the results obtained, comparing them with the performance of the models currently present on the robot.
- In Chapter 6, the deployment of the models developed in the previous chapter on the real robot will be carried out. Some tests will be conducted to certify the performance improvement of the FT sensors through the NN models.
- Finally, in Chapter 7, the results obtained and possible future developments will be discussed.

Part I

Background

Chapter 1

Force-torque sensors

A $6D$ force-torque (FT) sensor is a device designed to measure the forces and torques acting on an object in a three-dimensional space. Specifically, it can measure:

- F_x: Force along the x-axis.
- F_y: Force along the y-axis.
- F_z: Force along the z-axis.
- τ_x: Torque around the x-axis.
- τ_y: Torque around the y-axis.
- τ_z: Torque around the z-axis.

$6D$ force-torque sensors are used in various applications, like for example robotics, automotive, aerospace, and manufacturing, where precise force and torque measurements are critical for control, testing, and analysis purposes.

One of the most common types of force-torque sensors is the one based on strain gauges: this particular type of sensor is solicited by a *wrench* (i.e. the $6D$ vector containing the components of force and torque), and gives an electrical signal as an output. Of great importance is the calibration of the sensor, i.e. finding the map between the output potential difference and the applied wrench.

This chapter is structured in the following way. In section 1.1 we will briefly present several approaches to measure force and torques. In section 1.2 we will focus on piezoresistive strain gauges, which are the founding technology of many modern sensors. In section 1.3 we will discuss about *Wheatstone bridge*, an electrical circuits that transforms a variation of electrical resistance in a difference of potential. Finally, in section 1.4 we will resume what discussed in this chapter.

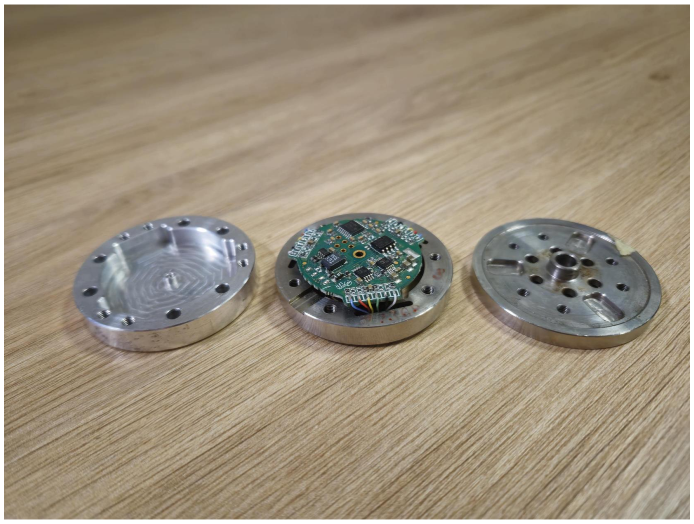


Figure 1.1: Components of $6D$ strain gauge force-torque sensor

1.1 Force Sensing Technologies

According to [2], one of the following methods can be used to measure an unknown force:

1. Directly or via a system of levers, balancing it against the known gravitational force on a standard mass.
2. Calculating the acceleration caused by an unknown force applied to a body with known mass.
3. Weighing it against the magnetic force that results from a magnet and a current-carrying coil interacting.
4. Converting the force to a fluid pressure, which is then measured.
5. Applying force to an elastic component and calculating the deflection that results.
6. Determining the variation in gyroscope precession brought on by an applied torque connected to the force being measured.
7. Determining how much a force-tensioned wire's inherent frequency changes.

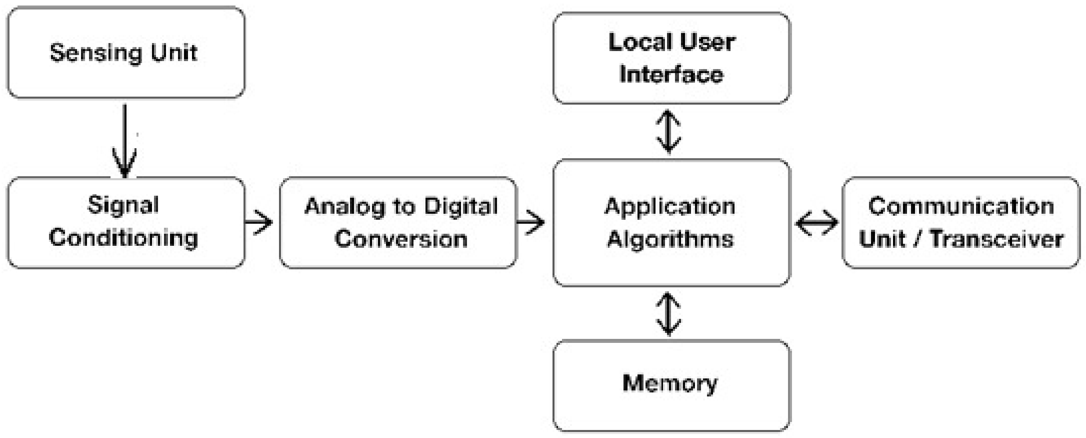


Figure 1.2: Block diagram of sensing module. Credits to [28]

The most popular method is the fifth one [3]; often, by using this method, the sensor transduces the wrench in an electrical signal. By the way, before this transduction, more actions are typically needed. A displacement sensor that transforms displacement into an electrical output paired with a force-to-displacement transducer makes up a standard force sensor. Stated otherwise, a standard force sensor comprises an elastic element (such as a silicon cantilever, polymer lattice, spring, etc.) and a gauge that measures the element's degree of compression or strain in order to transform it into an electrical output signal. In contemporary sensor design, integrating sensing elements with signal conditioning, conversion, and transmission circuits is a growing trend [3]. This combination is referred to as a sensing module. The output signals of a typical sensing element are low-level analog signals; before they can be digitalized, they must be amplified, filtered, impedance matched, and perhaps level shifted. Signal conditioners handle all of these duties. An analog-to-digital converter (ADC) is used to transform the signal from digital data after conditioning. The elastic element and the displacement sensor are given more attention in this thesis, even though every element of the sensing module is significant. With the addition of an elastic element, strain gauges become the primary means of force-torque sensing in robotics due to its capacity to measure dynamic loads and the potential of a high measurement bandwidth.

1.2 Piezoresistive Strain Gauges

The predominant technique for force measurement employs resistive sensing. Resistive sensors are favored due to their dependability, straightforward design, customizable resolution, and maintenance-free nature [4]. Additionally, electrical resistance is the most straightforward electrical property to measure accurately across a broad spectrum and at a reasonable cost. These significant attributes frequently position resistive sensors as the preferred option in sensor design. One of the most important technologies based on electrical resistances is strain gauges. Strain gauges use the piezoresistive capabilities of elastic materials, such as metals, alloys, semiconductors, or cermets, to measure resistance changes under strain. The operating principle is rather simple: a metallic material subjected to stress varies its electrical resistance.

There are two main types of strain gauges: *wire strain gauges* and *foil strain gauge*.

Wire strain gauges

A wire strain gauge comprises a resistor affixed to a flexible backing, which is then attached to the object under stress or force measurement. To achieve optimal sensitivity, the sensor design emphasizes long longitudinal and short transverse segments, minimizing transverse sensitivity to only a few percent of the longitudinal sensitivity. The main advantage of the architecture in figure 1.3 is its robustness. Conversely, the main drawbacks are:

- Limited variety in strain gauge design options.
- Reduced ability to dissipate heat produced by the Joule effect, which decreases sensitivity.

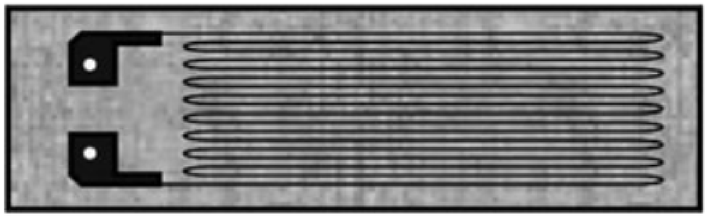


Figure 1.3: Wire strain gauge (credits to [3])

Foil strain gauges

Foil strain gauges are characterized by high versatility as the grid can be designed more freely. Furthermore, heat is dissipated more easily than with wire strain gauges. Another advantage is the ability to allow the passage of a greater current, and therefore greater sensitivity is obtained.

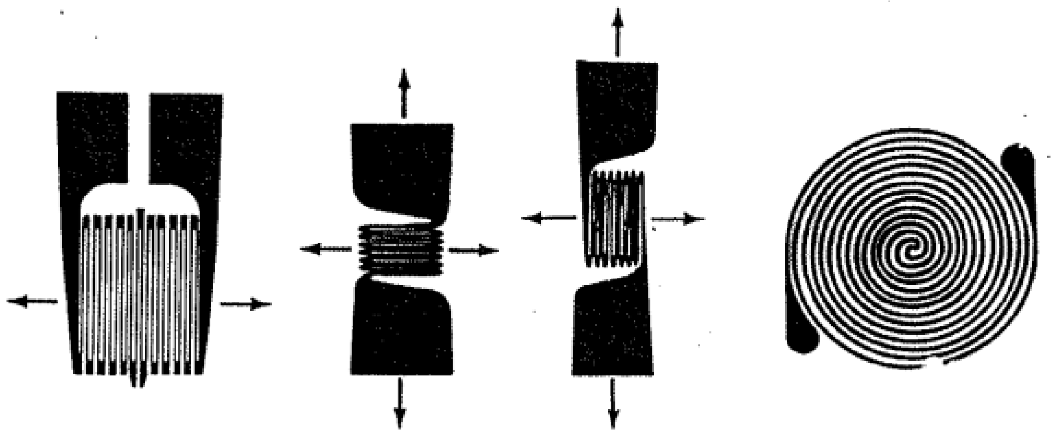


Figure 1.4: Foil strain gauge

The variation of the electrical resistance can be produced by modifying any of the three parameters from the relation:

$$R = \rho \frac{L}{A}, \tag{1.1}$$

where ρ is resistivity, L the conductor length, and A the cross section area. For wire strain gauges, we have $A = \frac{\pi D^2}{4}$ (with D diameter of the wire). In that case, differentiating 1.1, we have

$$\begin{aligned} dR &= \frac{\partial R}{\partial \rho} d\rho + \frac{\partial R}{\partial A} dA + \frac{\partial R}{\partial L} dL \\ &= \frac{L}{A} d\rho - \frac{\rho L}{A^2} dA + \frac{\rho}{A} dL \\ &= \frac{R}{\rho} d\rho - \frac{R}{\frac{\pi D^2}{4}} dD \frac{\pi D}{2} + \frac{R}{L} dL, \end{aligned}$$

which leads to

$$\frac{dR}{R} = \frac{d\rho}{\rho} - 2\frac{dD}{D} + \frac{dL}{L}. \tag{1.2}$$

Considering $\frac{d\rho}{\rho}$ negligible and moving on to finite differences, we obtain

$$\frac{\frac{\Delta R}{R}}{\frac{\Delta L}{L}} = 1 - \frac{\frac{2\Delta D}{D}}{\frac{\Delta L}{L}}.$$

We can notice that $\epsilon_L = \frac{\Delta L}{L}$ is the longitudinal unitary deformation and $\epsilon_T = -\frac{\Delta D}{D}$ is the transversal unitary deformation. Moreover, knowing that $\nu = -\frac{\epsilon_T}{\epsilon_L}$, it follows

$$\frac{\frac{\Delta R}{R}}{\frac{\Delta L}{L}} = 1 - 2\nu = F = const,$$

$$\frac{\Delta R}{R} = F\epsilon_L = F\epsilon. \tag{1.3}$$

The constant F is the so called gauge factor, which depends on the material of the strain gauge and temperature. Usually, F assumes values between 1.9 and 4.

Observation 1.2.1. *The relation* (1.3) *does not depend on the section A.*

The ideal material for a strain gauge should have the following characteristics:

- same $\frac{\Delta R}{R}$ in traction and compression
- high gauge factor F
- high Young Modulus E
- high ρ for having small grid dimensions
- high fatigue resistance
- dilatation coefficient similar to the material on which it is applied
- not ferromagnetic material (to avoid spurious EMFs (Electromotive Forces))

Some good materials are for example constantan, isoelastic, karma alloys.

Strain gauges can be configured in different arrangements to measure strains along various axes, often integrated into Wheatstone bridge circuits. Notably, semiconductive strain gauges exhibit high sensitivity to temperature changes, necessitating temperature-compensating networks in interface circuits or within the gauges themselves.

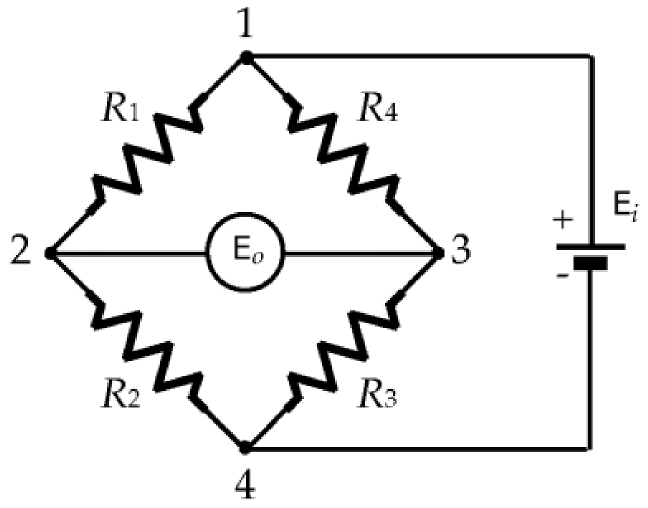


Figure 1.5: Base configuration of Wheatstone bridge. Credits to [5]

1.3 Wheatstone bridge

The Wheatstone Bridge aims to measure the variation of the electrical resistance.

The Wheatstone Bridge (figure 1.5) is an electrical circuit given by the combination of four resistances. The input tension E_i can be either direct or alternating. If $\frac{R_1}{R_2} = \frac{R_3}{R_4}$, we have an output voltage $E_0 = 0$. When one of the resistances changes value, the bridge becomes unbalanced, and $E_0 \neq 0$. Such output tension can be measured and used to determine the variation of the resistance. We can express the ratio between E_0 and E_i as:

$$\frac{E_O}{E_I} = \frac{R_1}{R_1 + R_2} - \frac{R_4}{R_3 + R_4} = \frac{R_1R_3 - R_2R_4}{(R_1 + R_2)(R_3 + R_4)}, \tag{1.4}$$

where $R_i(\Omega)$ is the value of resistance at the i-th position. Based on the number of variable resistances or strain gauges, it can be classified as a full bridge (all variable resistances), half bridge, or quarter bridge (1.6). In our particular case, the force-torque (FT) sensor used by the Artificial and Mechanical Intelligence group belongs to the second type.

This arrangement of resistances is well-suited for measuring small changes in resistance, making it ideal for detecting resistance variations in a strain gauge. Additionally, the Wheatstone Bridge can be configured to compensate for interference effects such as temperature, pressure, humidity, magnetic fields, radiation, etc. [6].

Accurate temperature compensation in the Wheatstone Bridge is achieved only if certain conditions are strictly met:

- Symmetry of the bridge.
- Identical temperature coefficients for all materials used.
- Identical resistances of all parts in the bridge arms that are combined for compensation.
- Identical temperatures on all compensating elements in the bridge circuit.
- Identical active grid areas.

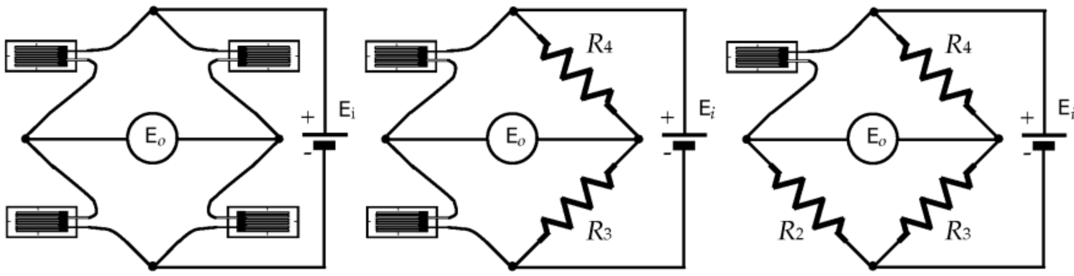


Figure 1.6: From left to right: full bridge, half bridge, quarter bridge. Credits to [5]

1.4 Conclusions

In this chapter we presented $6D$ force-torque sensors. Among the different sensing technologies, piezoresistive strain gauges are widely used due to their reliability and sensitivity. These sensors, integrated into Wheatstone bridge circuits, effectively convert mechanical strain into electrical signals. Key to their operation is the gauge factor, which links resistance changes to mechanical strain. Ensuring accurate measurements involves addressing factors like temperature compensation and circuit symmetry. As technology advances, the integration of sensing elements with signal conditioning and digital conversion is becoming more prevalent, enhancing the overall performance and applicability of these sensors. The insights presented in this chapter highlight the fundamental principles and operational intricacies of

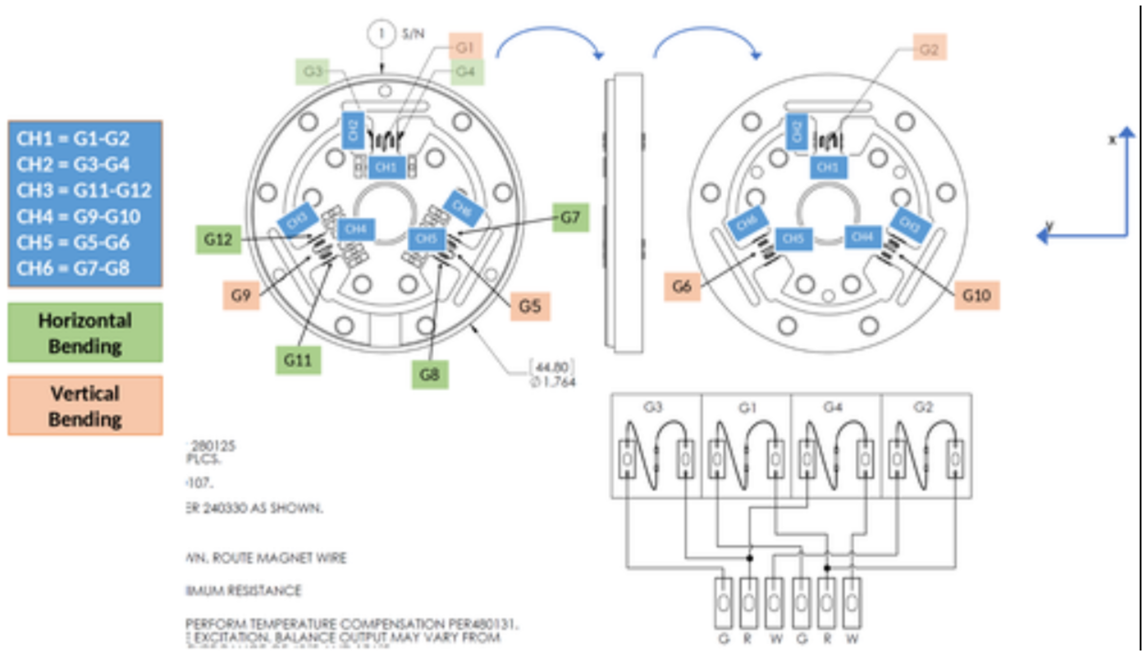


Figure 1.7: Design of the strain gauge $6D$ FT sensor discussed in this work

force-torque sensing, laying the groundwork for their effective application in robotics and other fields.

The sensor on which we will work in this thesis arises in this context. In particular, in our sensor there are 3 cantilevers (in the case of a sensor produced by AMI) in which strain gauges are glued. Totally, there are 12 strain gauges, which combine to form a total of 6 Wheatstone half-bridges; each of these half bridges therefore contains two constant resistances and two variable resistances (the strain gauges). Each of these half bridges produces a potential difference in output: in total, therefore, a vector $v \in \mathbb{R}^6$ of potential differences is obtained.

For this reason, it is necessary to calibrate the sensor, i.e. identify a map $f : \mathbb{R}^6 \to \mathbb{R}^6$ which, taking as input the vector potential difference v, returns the vector wrench f. However, the identification of this function is far from trivial. Models typically used for this purpose can be, for example, linear or, more generally, polynomial, capable of interpolating the available data. Unfortunately, the calibration of an FT-sensor is very sensitive to the conditions under which it takes place, such as sensor orientation and temperature. It is therefore difficult to find an analytical model capable of taking into account all these non-linearities. In light of this, in recent years the literature has proposed the use of neural networks to calibrate FT-sensors

[15], [16]; in this way, one avoids having to define a closed-form model describing the relationship between potential difference v and wrench f. In this work, we would like to calibrate the force-torque sensor with a neural network approach, considering not only the raw values of the FT, but also other inputs, like for example the temperature of the sensor. Moreover, unlike [15] and [16], in this thesis we will perform an *in-situ* calibration, meaning that we will collect data with the sensor already mounted on the robot.

Chapter 2

Neural Networks

In this chapter we will present the theoretical basis of *artificial neural networks* (ANNs); this will be useful for the implementation of neural networks (NNs) for the calibration of the force-torque (FT) sensors introduced in the chapter 1.

2.1 Introduction to Neural Networks

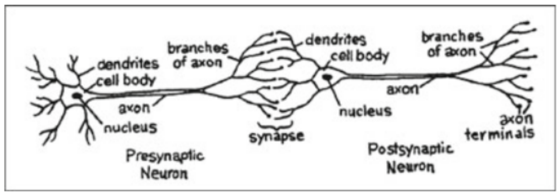
Artificial neural networks are a subset of machine learning which try to simulate the human cell behavior. This is made through cells (neurons) connected to each other with axons and dendrites. Such connections areas are called synapses. Through these connections, electrical stimuli are propagated from one neuron to another.

Similar to biological neural networks, ANNs are composed of units (artificial neurons) connected to other units. Again, each neuron processes information and transmits the output to other neurons.

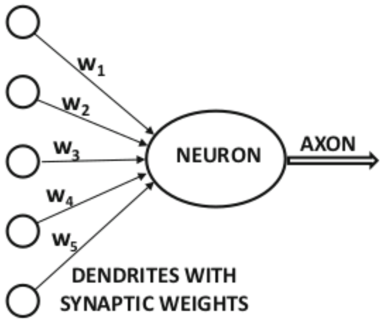
A Neural Network is mathematically described as a graph $G = (\mathcal{V}, \mathcal{E}, W)$, a structure defined by

- A set of nodes $\mathcal{V}$, which represents the units participating in the network;
- A set of links $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$;
- Weights W_{ij} associated to each link. The matrix W is usually called adjacency matrix.

It consists in at least an input layer and an output layer. Moreover, there could be also one or more hidden layers. Every layer consists in a certain number of nodes



(a) Biological neural network



(b) Artificial neural network

Figure 2.1: The synaptic connections between neurons. Credits to [9]

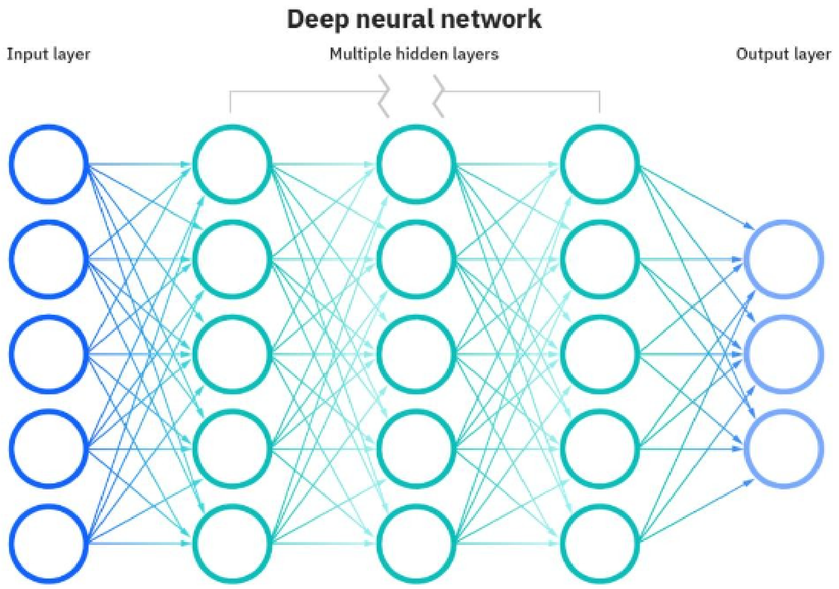


Figure 2.2: Example of Neural Network. Credits to [27]

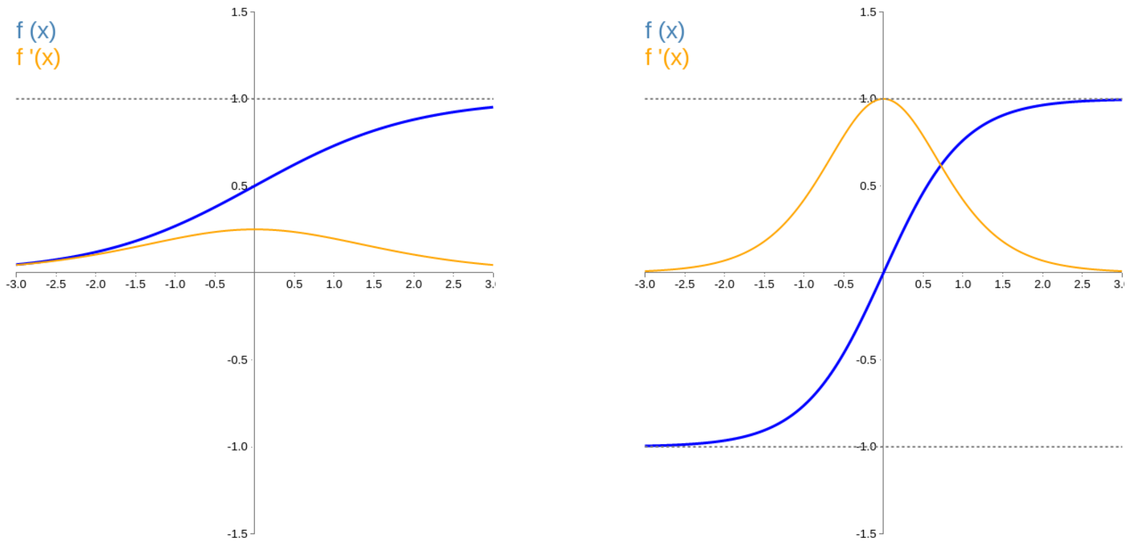
(neurons). Every node contains a variable y, which is the result of the following operation:

$$y = \Phi(w^T x + b), \tag{2.1}$$

where x are the values contained in the neurons of the previous layer, b the *bias* and w is a vector of weights. w is nothing but a column of the adjacency matrix

W mentioned above. Φ is called *activation function.* Activation functions are really important because can introduce non-linearity into the network, enabling it to learn complex patterns. Some example of common activation functions are:

- **Sigmoid**:
$$\sigma(x) = \frac{1}{1+e^{-x}}.$$
Outputs values between 0 and 1, making it suitable for binary classification tasks.



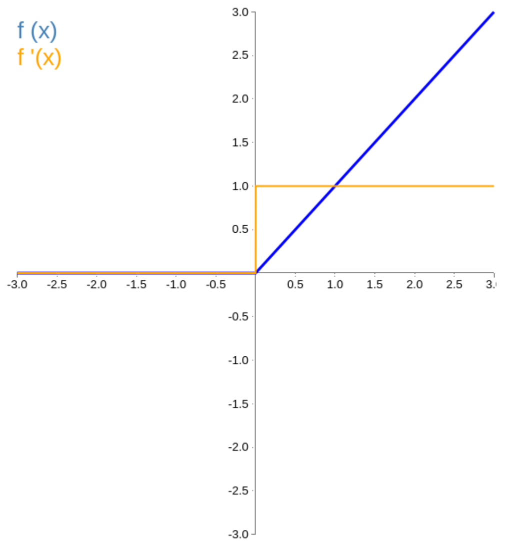
(a) Sigmoid **(b)** Tanh

Figure 2.3: Sigmoid and Tanh activation functions

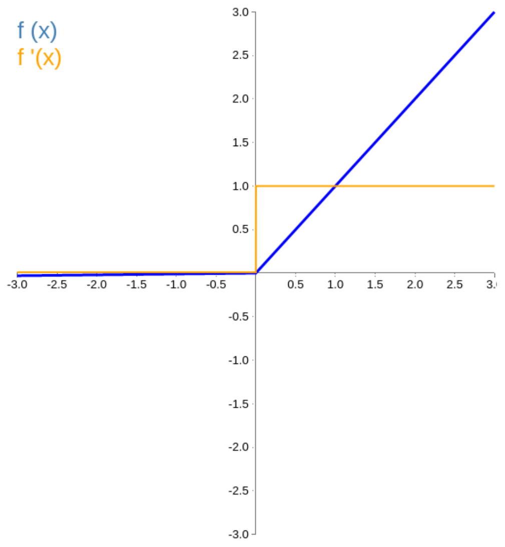
- **Rectified Linear Unit (ReLU)**:
$$\mathrm{ReLU}(x) = \max(0, x).$$
Widely used in deep networks.

- **Step**:
$$\mathrm{step}(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0. \end{cases}$$

As we will see later, the use of non-linear functions within neural networks gives the model great capabilities to approximate functions.

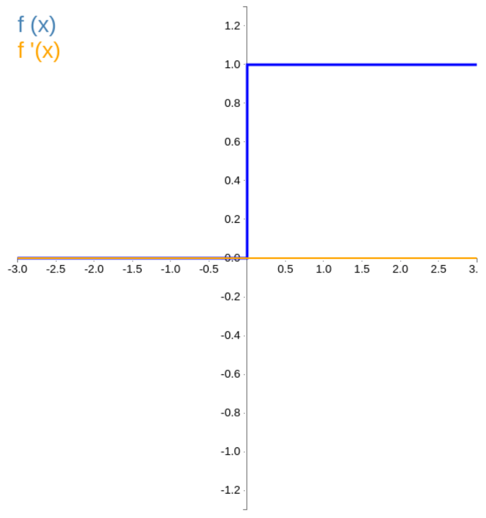


(a) ReLU

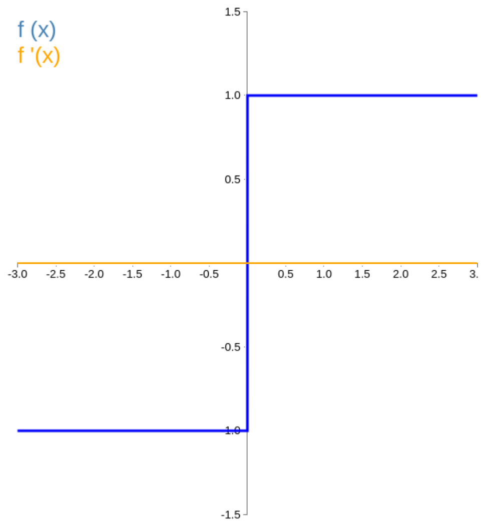


(b) Leaky ReLU

Figure 2.4: ReLU and Leaky ReLU activation functions



(a) Step



(b) Sign

Figure 2.5: Step and Sign activation functions

2.2 Training of a NN through supervised learning

Training neural networks consists of optimizing their parameters (weights and biases) to minimize a loss function that quantifies the network's prediction error. One of the most common approacheas is supervised learning, where the network learns from labeled examples provided in a training dataset. Basically, calling x the input, y the label, the NN tries to learn a function f such that $y = f(x)$. In order to do that, we have to solve an optimization problem

$$\min_{w \in \mathbb{R}^B} L(w), \tag{2.2}$$

where w is a vector containing all the trainable weights of the NN and $L : \mathbb{R}^B \to \mathbb{R}^+$ is a function that could have many local minima. To solve the problem (2.2), is often used a gradient-based method. The idea of this kind of methods is to move repeatadly in the opposite direction of the gradient of the cost function with respect to the optimization variables. The gradient tells us the direction of the steepest ascent, and by moving in the opposite direction, we can find the direction of the steepest descent. We'll see further details in 2.2.4. Usually, the resolution of the problem (2.2) involves four main steps:

1. **Forward Pass**: Compute the output of the network.
2. **Loss Computation**: Evaluate discrepancy between output of the Neural Network y and the label $\hat{y}$ using a loss function.
3. **Backward Pass**: Compute gradients of the loss with respect to each weight (and bias) of the NN.
4. **Weight Update**: Adjust weights using gradient descent.

This procedure is repeated until a good solution for the (2.2) problem is reached.

2.2.1 Forward pass

In section 2.1 we already discussed the main idea of the computations in a NN. For simplicity, let's consider a NN with the input layer, the output layer and an hidden layer. Let input vector be $x \in \mathbb{R}^n$, weights between input layer and hidden layer be $W^{(1)} \in \mathbb{R}^{m \times n}$, weights between hidden layer and output layer be $W^{(2)} \in \mathbb{R}^{k \times m}$, biases for hidden layer be $b^{(1)} \in \mathbb{R}^m$, and biases for the output layer be $b^{(2)} \in \mathbb{R}^k$.

Activation functions for the hidden layer and the output layer are denoted as $\Phi^{(1)}$ and $\Phi^{(2)}$ respectively.

The forward pass is computed as follows:

$$z^{(1)} = W^{(1)}x + b^{(1)},$$

$$a^{(1)} = \Phi^{(1)}(z^{(1)}),$$

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)},$$

$$y = \Phi^{(2)}(z^{(2)}).$$

Here, $z^{(1)}$ and $z^{(2)}$ are the linear combinations of inputs and weights for the hidden and output layers respectively, while $a^{(1)}$ is the vector of variables saved in the hidden layer, and y is the final output of the network.

2.2.2 Loss computation

After the forward computation of the Neural Network, we need an error metric to quantify the discrepancy between the exact $\hat{y}$ and the output of the NN y. In order to do that, we need to choose a *loss function*. Below, some of the most used loss functions:

- **Mean Squared Error (MSE)**

  $$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2.$$

  Used commonly for regression problems where $\hat{y}_i$ represents the label and y_i represents the predicted value by the model.

- **Binary Cross-Entropy Loss (Log Loss)**

  $$\text{Binary Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}\left[\hat{y}_i \log(y_i) + (1-\hat{y}_i)\log(1-y_i)\right].$$

  Used for binary classification tasks where the output is a probability between 0 and 1.

- **Categorical Cross-Entropy Loss**

  $$\text{Categorical Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C}\hat{y}_{ij}\log(y_{ij}).$$

Used for multi-class classification tasks where C is the number of classes, $\hat{y}_{ij}$ is 1 if the observation i is in class j and 0 otherwise, and y_{ij} is the predicted probability of observation i belonging to class j.

- **Hinge Loss**

  $$\text{Hinge Loss} = \max(0, 1 - \hat{y}_i \cdot y_i).$$

  Commonly used for training classifiers in Support Vector Machines (SVMs) and also for margin-based classifiers in neural networks.

- **Huber Loss**

  $$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta) & \text{for } |a| > \delta, \end{cases}$$

  where $a = \hat{y}_i - y_i$. It is used in regression tasks and is less sensitive to outliers than the squared error loss.

- **Kullback-Leibler Divergence**

  $$\text{KL Divergence} = \sum_i \hat{y}_i \log\left(\frac{\hat{y}_i}{y_i}\right).$$

  Measures how one probability distribution diverges from a second, expected probability distribution. Often used in probabilistic models and for training variational autoencoders.

2.2.3 Backward Pass

Now, we need to compute the gradients of the function with respect to the weights of the Neural Network. The algorithm to do that is called *backpropagation*. Such algorithm was mostly used in the field of control theory, and has begun to be used for NN trainings in [12].

The backward pass involves computing the gradients of the loss function with respect to each weight in the network. This is done using the chain rule for differentiation, propagating the error backward through the network from the output layer to the input layer.

Gradients for Output Layer

First, we compute the gradient of the loss with respect to the output of the network $\frac{\partial L}{\partial y}$.

Next, we compute the gradient of the loss with respect to the pre-activation output $z^{(2)}$ using the activation function derivative $\Phi^{(2)\prime}$:

$$\frac{\partial L}{\partial z^{(2)}} = \frac{\partial L}{\partial y} \odot \Phi^{(2)\prime}(z^{(2)}),$$

where $\odot$ denotes the element-wise product.

The gradients with respect to the weights $W^{(2)}$ and biases $b^{(2)}$ are then:

$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z^{(2)}} a^{(1)T},$$

$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial z^{(2)}}.$$

Gradients for Hidden Layer

For the hidden layer, we first propagate the error backward from the output layer:

$$\frac{\partial L}{\partial a^{(1)}} = W^{(2)T} \frac{\partial L}{\partial z^{(2)}}.$$

Next, we compute the gradient of the loss with respect to the pre-activation input $z^{(1)}$ using the activation function derivative $\Phi^{(1)\prime}$:

$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial a^{(1)}} \odot \Phi^{(1)\prime}(z^{(1)}).$$

The gradients with respect to the weights $W^{(1)}$ and biases $b^{(1)}$ are:

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial z^{(1)}} x^T,$$

$$\frac{\partial L}{\partial b^{(1)}} = \frac{\partial L}{\partial z^{(1)}}.$$

2.2.4 Weight Update

Once the gradients are computed, the weights and biases are updated using gradient descent. The update rule for gradient descent is:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}},$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}}.$$

where η is the learning rate and l denotes the layer index. This process is repeated for each layer in the network. It is worth underlining the importance of η: it is one of the fundamental parameters in the construction of a neural network, as it quantifies the tendency of the network weights to be updated during the training phase.

This classical gradient descent method is affected by several issues, like for example:

- vanishing or exploding gradients, especially in Recurrent Neural Networks [10];
- presence of saddle points, which slows down the training [11];
- possibility of getting trapped in local minima.

To remedy these problems, some modifications have been introduced to this algorithm:

- Stochastic Gradient Descent. While in the classic gradient descent, all the samples of the dataset are used at a single time, in this variation only a sample is stochastically chosen in each iteration. Main advantages are convergence speed and efficiency from a memory point of view. The main disadvantage is related to the single sample approach. In fact, a single sample could be an outlier or, more generally, contain an error;
- Mini-Batch Stochastic Gradient Descent. Similiar to the method discussed above, but considering a group of samples (batch) rather than a single sample during each iteration; in this way, it fixes the disadvantage of the classic stochastic gradient descent method.
- Adagrad (Adaptive Gradient Algorithm). Throughout the optimization process, this technique records the total squared magnitude of the partial derivative with respect to each parameter. In order to do that, we consider the aggregate value for the i-th parameter, A_i; during each iteration, we update such quantity in the following way:

  $$A_i \leftarrow A_i + \left(\frac{\partial L}{\partial w_i}\right)^2 \quad \forall i \quad . \tag{2.3}$$

  The update for the i-th parameter w_i is as follows:

  $$w_i \leftarrow w_i - \frac{\eta}{\sqrt{A_i}} \left(\frac{\partial L}{\partial w_i}\right) \quad \forall i \quad . \tag{2.4}$$

  The main problem of this method is the slowness.

- RMSprop (Root Mean Square Propagation). This method is similiar to the previous one, but moreover we have also an exponential averaging in the updating of A_i. For the averaged value of the i-th parameter w_i, we have the following way of updating A_i:

$$A_i \leftarrow \rho A_i + (1-\rho)\left(\frac{\partial L}{\partial w_i}\right)^2 \quad \forall i \quad , \tag{2.5}$$

  with $\rho \in (0,1)$. Then, we update η as follows:

$$w_i \leftarrow w_i - \frac{\eta}{\sqrt{A_i}}\left(\frac{\partial L}{\partial w_i}\right) \quad \forall i \quad . \tag{2.6}$$

  Advantages with respect to Adagrad: faster and the importance of old gradients decays exponentially in time.

- Adam (Adaptive Moment Estimation). Similiarly to RMSProp, we have for the updating of A_i:

$$A_i \leftarrow \rho A_i + (1-\rho)\left(\frac{\partial L}{\partial w_i}\right)^2 \quad \forall i \quad , \tag{2.7}$$

  with $\rho \in (0,1)$. Then we compute F_i, a smoothed version of the gradient:

$$F_i \leftarrow \rho_f F_i + (1-\rho_f)\left(\frac{\partial L}{\partial w_i}\right) \quad \forall i \quad . \tag{2.8}$$

  where $\rho_f \in (0,1)$ is another decay parameter. Finally, we have for the weights:

$$w_i \leftarrow w_i - \frac{\eta_t}{\sqrt{A_i}} F_i \quad \forall i \quad . \tag{2.9}$$

  η_t is a modification of η, computed in the following way:

$$\eta_t = \eta\left(\frac{\sqrt{1-\rho^t}}{1-\rho_f^t}\right) \quad , \tag{2.10}$$

  where t is the iteration index. Adam combines the advantages of other algorithms [13]; for that reason, it is one of the most popular algorithm in NN field.

2.3 Theoretical Foundations

2.3.1 Universal Approximation Theorems

In this section we are going to enunciate the main theorems about the capability of representation of Neural Networks. Such theorems state that, for a sufficiently large network with non-linear activation functions, neural networks can approximate any continuous function to an arbitrary degree of accuracy [7], [8]. In the following theorems, we will indicate with $\mathcal{C}^0$ the space of continuous functions and with $\|f\|_{\infty,K} = \max_{f\in K}|f|$ the infinity norm of the function f.

Theorem 2.3.1 (Fixed depth, variable witdth)**.** *Let's condider a function* $\bar{f}$

$$\bar{f} : K \subset \mathbb{R}^n \to \mathbb{R}^m \quad K\,compact, \bar{f} \in \mathcal{C}^0(K, \mathbb{R}^m),$$

the NN function f

$$f : K \to \mathbb{R}^m,$$

and the function $\Phi : \mathbb{R} \to \mathbb{R}$.
If and only if Φ *is not a polynomial, then*

$$\forall n \in \mathbb{N}, \forall m \in \mathbb{N}, \forall K\,compact \subset \mathbb{R}^n, \forall \bar{f} \in \mathcal{C}^0(K, \mathbb{R}^m), \exists k \in \mathbb{N}, W^1 \in \mathbb{R}^{k\times n}, w_0 \in \mathbb{R}^k, W^2 \in \mathbb{R}^{m\times k}$$

such that, setting

$$y = f(x) = W^2(\Phi(W^1x + w_0)),$$

one has

$$\|\bar{f} - f\|_{\infty,K} < \epsilon.$$

Theorem 2.3.2 (Fixed width, variable depth)**.** *Assume* Φ *is continuous, not affine, continuously differentiable on at least one* $\bar{x} \in \mathbb{R}$*, where* $\Phi'(\bar{x}) \neq 0$*. Then*

$$\forall n \in \mathbb{N}, \forall m \in \mathbb{N}, \forall K\,compact \subset \mathbb{R}^n, \forall \bar{f} \in \mathcal{C}^0(L, \mathbb{R}^m), \forall \epsilon > 0, \exists L > 0$$

and exists a NN with L *layers, each one of width* $= n + m + 2$*, such that*

$$\|\bar{f} - f\|_{\infty,K} < \epsilon.$$

Theorem 2.3.3 (Fixed depth, fixed width). $\forall [a,b] \in \mathbb{R}, \forall n \in \mathbb{N}, \forall \bar{f} \in \mathcal{C}^0([a,b]^n, \mathbb{R})$ $\exists \Phi : \mathbb{R} \to \mathbb{R}$ *continuous and computable (for example, sigmoid function) and exists a NN with* 2 *hidden layers* ($L = 3$)*, with* n *neurons in the first layer,* $2n+2$ *neurons in the second layer such that*

$$\|\bar{f} - f\|_{\infty,[a,b]^n} < \epsilon.$$

The theorems just exposed lay the theoretical foundations for the NNs that we will use to find a sufficiently accurate calibration map for force-torque sensors.

2.4 Conclusions

This chapter covered neural networks, a key component of contemporary machine learning that draws inspiration from the neurons found in the human brain. We began by understanding how weights and associated nodes in artificial neural networks imitate organic neurons. These networks are graph-structured, with nodes standing in for units and edges for connections that have weights assigned to them. The activation functions, such as the sigmoid or ReLU, are essential to neural networks because they introduce non-linearities, which are necessary for learning intricate patterns. We covered the training of neural networks using supervised learning, wherein methods such as gradient descent are used to modify weights in order to decrease prediction errors. Different optimization techniques were described, each with a special advantage for improving training efficiency, like for example stochastic gradient descent and Adam. We also talked about the Universal Approximation Theorems, which state that neural networks can approximate any continuous function with sufficient complexity. Overall, this chapter covers fundamental understanding about neural networks, their training methods, and theoretical possibilities. Such basis will be useful for the construction of Neural Networks for force-torque sensors calibration.

Chapter 3

Basics of rigid multibody dynamics

Rigid body dynamics is a fundamental aspect of mechanics that deals with the motion of solid bodies without deformation. This chapter delves into the mathematical foundation necessary to understand and analyze the dynamics of rigid bodies, particularly within the context of humanoid robotics. The study of rigid body dynamics enables us to model, simulate, and control the motion of robots, providing critical insights into their behavior and interactions with the environment. We will explore essential mathematical definitions and concepts, frame kinematics, velocity, accelerations, and the forces and torques acting on rigid bodies.

In the subsequent sections, we will introduce the mathematical structures and operations fundamental to rigid body dynamics, such as vectors, matrices, and groups like SO(3) and SE(3) (section 3.1). These tools allow us to describe and manipulate the orientation and position of rigid bodies (section 3.2). Afterwards, rigid body velocities (section 3.3) and accelerations (section 3.4) will be presented. Then, after discussed force-torques (section 3.5), we will finally write the dynamics equations for a rigid body (section 3.6). After doing that, we will finally present the basis of dynamics of rigid multibody systems (section 3.7).

In the present work, the theory developed in this chapter will be useful to define the loss function of neural networks for the calibration of $6D$ force-torque sensors.

The main references for this chapter are [17], [18], [19] and [20].

3.1 Some mathematical definitions

- The set of real numbers is denoted by $\mathbb{R}$. Let u and v be two n-dimensional column vectors of real numbers, i.e. $u, v \in \mathbb{R}^n$. Their inner product is represented as $u^T v$, where T indicates the transpose operator.

- The identity matrix of size n is denoted as $I_n \in \mathbb{R}^{n\times n}$; the zero column vector of size n is denoted as $0_n \in \mathbb{R}^n$; the zero matrix of size $n \times m$ is denoted as $0_{n\times m} \in \mathbb{R}^{n\times m}$.

- The special orthogonal group SO(3) is the set of 3×3 orthogonal matrices with a determinant equal to one, defined as

$$\mathrm{SO}(3) := \{R \in \mathbb{R}^{3\times 3} \mid R^T R = I_3, \det(R) = 1\}. \tag{3.1}$$

  Such set is called group under the matrix multiplication operator because, given $A, B \in \mathrm{SO}(3)$ we have the following properties:

  - closure: $AB \in \mathrm{SO}(3)$
  - associativity: (AB)C = A(BC)
  - identity element existence: there exists an element $I \in \mathrm{SO}(3)$ (the identity matrix for SO(3)) such that $AI = IA = A$
  - inverse element existence: there exists an element A^{-1} in the group such that $AA^{-1} = A^{-1}A = I$.

- The set $\mathfrak{so}(3)$, consists of 3×3 skew-symmetric matrices,

$$\mathfrak{so}(3) := \{S \in \mathbb{R}^{3\times 3} \mid S^T = -S\}. \tag{3.2}$$

- The special Euclidean group SE(3) is defined as

$$\mathrm{SE}(3) := \left\{ \begin{bmatrix} R & p \\ 0_{1\times 3} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \mid R \in \mathrm{SO}(3), p \in \mathbb{R}^3 \right\}. \tag{3.3}$$

  This group is also known as group of rigid-body motions or homogeneous transformation matrices in $\mathbb{R}^3$.

- The set $\mathfrak{se}(3)$ is defined as

$$\mathfrak{se}(3) := \left\{ \begin{bmatrix} \Omega & v \\ 0_{1\times 3} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4} \mid \Omega \in \mathfrak{so}(3), v \in \mathbb{R}^3 \right\}. \tag{3.4}$$

- For the vector $w = (x, y, z) \in \mathbb{R}^3$, we define $w^\wedge$ (read "w hat") as the 3×3 skew-symmetric matrix

$$w^\wedge = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^\wedge := \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \in \mathfrak{so}(3). \tag{3.5}$$

- For the skew-symmetric matrix $W = w^{\wedge}$, we define $W^{\vee} \in \mathbb{R}^3$ (read "W vee") as

$$W^{\vee} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}^{\vee} := \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3. \tag{3.6}$$

  Clearly, the vee operator is the inverse of the hat operator.

- For a vector $\mathrm{v} = (v, \omega) \in \mathbb{R}^6$, where v and ω are in $\mathbb{R}^3$, we define

$$\mathrm{v}^{\wedge} = \begin{bmatrix} v \\ \omega \end{bmatrix}^{\wedge} := \begin{bmatrix} \omega^{\wedge} & v \\ 0_{1\times 3} & 0 \end{bmatrix} \in \mathfrak{se}(3). \tag{3.7}$$

- For a vector $\mathrm{v} = (v, \omega) \in \mathbb{R}^6$, where v and ω are in $\mathbb{R}^3$, we define

$$\mathrm{v}\times := \begin{bmatrix} \omega^{\wedge} & v^{\wedge} \\ 0_{3\times 3} & \omega^{\wedge} \end{bmatrix} \in \mathbb{R}^{6\times 6}. \tag{3.8}$$

- For a vector $\mathrm{v} = (v, \omega) \in \mathbb{R}^6$, where v and ω are in $\mathbb{R}^3$, we define

$$\mathrm{v}\bar{\times}^{*} := \begin{bmatrix} \omega^{\wedge} & 0_{3\times 3} \\ v^{\wedge} & \omega^{\wedge} \end{bmatrix} \in \mathbb{R}^{6\times 6}. \tag{3.9}$$

- The vee operator is defined as the inverse of the hat operator, such that

$$\begin{bmatrix} \omega^{\wedge} & v \\ 0_{1\times 3} & 0 \end{bmatrix}^{\vee} := \begin{bmatrix} v \\ \omega \end{bmatrix} = \mathrm{v} \in \mathbb{R}^6. \tag{3.10}$$

- For $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{p\times q}$, we denote the Kronecker product as $A \otimes B \in \mathbb{R}^{np\times mq}$.

- Given $X \in \mathbb{R}^{m\times p}$, $\mathrm{vec}(X) \in \mathbb{R}^{mp}$ represents the column vector obtained by stacking the columns of the matrix X. According to this definition of $\mathrm{vec}(\cdot)$, we have

$$\mathrm{vec}(AXB) = (B^T \otimes A)\mathrm{vec}(X). \tag{3.11}$$

3.2 Points and frames

In the field of humanoid robotics, the rigid body assumption is really useful in order to study the dynamics of the system. With the aim of describing the kinematics of a rigid body, the concept of *frame* is fundamental, which is identified by a point called *origin* (a vector in $\mathbb{R}^3$) and an *orientation frame* (a matrix in SO(3)). We will express the frame $A = (o_A, [A])$, where o_A is the origin of the frame and $[A]$ is its orientation. In this way, given a point p, we can express its coordinates with respect to the frame A as

$$^{A}p := \begin{bmatrix} \vec{r}_{o_A,p} \cdot \vec{x}_A \\ \vec{r}_{o_A,p} \cdot \vec{y}_A \\ \vec{r}_{o_A,p} \cdot \vec{z}_A \end{bmatrix} \in \mathbb{R}^3, \tag{3.12}$$

where $\cdot$ is the scalar product and $\vec{x}_A$, $\vec{y}_A$, $\vec{z}_A$ are the versors which define $[A]$. Given two frames A, B, we can define the rotation between such frames as

$$^{A}R_B \in \mathrm{SO}(3).$$

This transformation depends only on the relative orientation of the frames, not on the relative position of the origins. In order to consider also the distances between the origin of the frames A, B, we can use the homogeneous transformation:

$$^{A}H_B := \begin{bmatrix} ^{A}R_B & ^{A}o_B \\ 0_{1\times 3} & 1 \end{bmatrix}. \tag{3.13}$$

Moreover, given a point p, $^{A}H_B$ can be also useful to map the representation ^{B}p in ^{A}p. In order to do that, we have to consider a modified version of ^{B}p in ^{A}p, namely $^{A}\bar{p} := (^{A}p; 1)$, $^{B}\bar{p} := (^{B}p; 1) \in \mathbb{R}^4$. In this way, we can easily write

$$^{A}\bar{p} = {^{A}H_B}\, {^{B}\bar{p}}. \tag{3.14}$$

3.3 Velocities

In the following, given a point p and a frame A, we define

$$^{A}\dot{p} := \frac{d}{dt}\left(^{A}p\right). \tag{3.15}$$

In particular, when p is the origin of a frame, e.g., $p = o_B$, we have

$$^{A}\dot{o}_B = \frac{d}{dt}\left(^{A}o_B\right).$$

Similarly to (3.15), we also define

$$^{A}\dot{R}_B := \frac{d}{dt}\left(\,^{A}R_B\right), \tag{3.16}$$

and

$$^{A}\dot{H}_B := \frac{d}{dt}\left(^{A}H_B\right) = \begin{bmatrix} ^{A}\dot{R}_B & ^{A}\dot{o}_B \\ 0_{1\times 3} & 0 \end{bmatrix}. \tag{3.17}$$

The relative velocity between a frame B with respect to a frame A can be represented by the time derivative of the homogenous matrix $^{A}H_B \in \mathrm{SE}(3)$. We can obtain also a more compact form of $^{A}\dot{H}_B$, by multiplying it with the inverse of $^{A}H_B$ on the left or on the right. By multiplying on the left, we obtain

$$\begin{aligned} ^{A}H_B^{-1}\,^{A}\dot{H}_B &= \begin{bmatrix} ^{A}R_B^T & -^{A}R_B^{T}\,^{A}o_B \\ 0_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} ^{A}\dot{R}_B & ^{A}\dot{o}_B \\ 0_{1\times 3} & 0 \end{bmatrix} \\ &= \begin{bmatrix} ^{A}R_B^{T}\,^{A}\dot{R}_B & ^{A}R_B^{T}\,^{A}\dot{o}_B \\ 0_{1\times 3} & 0 \end{bmatrix} \in \mathfrak{se}(3). \end{aligned} \tag{3.18}$$

By introducing

$$^{B}v_{A,B} := \,^{A}R_B^{T}\,^{A}\dot{o}_B, \tag{3.19}$$

$$^{B}\omega^{\wedge}_{A,B} := \,^{A}R_B^{T}\,^{A}\dot{R}_B, \tag{3.20}$$

we can define the *left trivialized* velocity of frame B with respect to frame A:

$$^{B}\mathrm{v}_{A,B} := \begin{bmatrix} ^{B}v_{A,B} \\ ^{B}\omega_{A,B} \end{bmatrix} \in \mathbb{R}^6. \tag{3.21}$$

By construction (equation (3.18)), we have:

$$^{B}\mathrm{v}^{\wedge}_{A,B} = \,^{A}H_B^{-1}\,^{A}\dot{H}_B. \tag{3.22}$$

Similarly to what was done in equation (3.18), if instead of multiplying by H_B^{-1} on the left, we multiply on the right, we obtain

$$
{}^A\dot{H}_B {}^A H_B^{-1} = \begin{bmatrix} {}^A\dot{R}_B & {}^A\dot{o}_B \\ 0_{1\times 3} & 0 \end{bmatrix} \begin{bmatrix} {}^A R_B^T & -{}^A R_B^T {}^A o_B \\ 0_{1\times 3} & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} {}^A\dot{R}_B {}^A R_B^T & {}^A\dot{o}_B - {}^A\dot{R}_B {}^A R_B^T {}^A o_B \\ 0_{1\times 3} & 0 \end{bmatrix} \in \mathfrak{se}(3). \tag{3.23}
$$

Define ${}^A v_{A,B}$ and ${}^A\omega_{A,B} \in \mathbb{R}^3$ as

$$
{}^A v_{A,B}, := {}^A\dot{o}_B - {}^A\dot{R}_B {}^A R_B^T {}^A o_B, \tag{3.24}
$$

$$
{}^A\omega_{A,B}^\wedge := {}^A\dot{R}_B {}^A R_B^T. \tag{3.25}
$$

The *right trivialized* velocity of B with respect to A is then defined as

$$
{}^A\mathrm{v}_{A,B} := \begin{bmatrix} {}^A v_{A,B} \\ {}^A\omega_{A,B} \end{bmatrix} \in \mathbb{R}^6. \tag{3.26}
$$

By construction,

$$
{}^A\mathrm{v}_{A,B}^\wedge = {}^A\dot{H}_B {}^A H_B^{-1}. \tag{3.27}
$$

By combining the definitions (3.22) and (3.27), we obtain the mapping between right and left trivialized velocities:

$$
{}^A\mathrm{v}_{A,B}^\wedge = {}^A H_B {}^B\mathrm{v}_{A,B}^\wedge {}^A H_B^{-1}.
$$

We can also write the $6D$ vector shape of the relation above as

$$
{}^A\mathrm{v}_{A,B} = {}^A X_B {}^B\mathrm{v}_{A,B}.
$$

where

$$
{}^A X_B = \begin{bmatrix} {}^A R_B & {}^A o_B^\wedge {}^A R_B \\ 0_{3\times 3} & {}^A R_B. \end{bmatrix} \in \mathbb{R}^{6\times 6}. \tag{3.28}
$$

${}^A X_B$ is usually called *adjoint matrix*. By the way, in some situations we would like to express the $6D$ velocity of a frame with just the time derivatives ${}^A\dot{o}_B$ and ${}^A\omega_{A,B}$. In order to do that, it's convenient to introduce the frame $B[A] = (o_B, [A])$, which has the same origin of B and the orientation of A. We have then:

$$
{}^{B[A]}H_B = \begin{bmatrix} {}^A R_B & 0_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}, \tag{3.29}
$$

and by expressing the velocity $\mathrm{v}_{A,B}$ in $B[A]$, we get:

$$
{}^{B[A]}\mathrm{v}_{A,B} = {}^{B[A]}X_B {}^B\mathrm{v}_{A,B} = \begin{bmatrix} {}^A R_B & 0 \\ 0 & {}^A R_B \end{bmatrix} \begin{bmatrix} {}^B R_A {}^A\dot{o}_B \\ {}^B\omega_{A,B} \end{bmatrix} = \begin{bmatrix} {}^A\dot{o}_B \\ {}^A\omega_{A,B} \end{bmatrix}. \tag{3.30}
$$

3.4 Accelerations

Equation (3.22) can be rewritten as

$$ {}^{A}\dot{H}_{B} = {}^{A}H_{B}\,{}^{B}\mathrm{v}^{\wedge}_{A,B}. \tag{3.31}$$

Differentiating (3.28) with respect to time, we have

$$ {}^{A}\dot{X}_{B} = \begin{bmatrix} {}^{A}\dot{R}_{B} & {}^{A}\dot{o}^{\wedge}_{B}\,{}^{A}R_{B} + {}^{A}o^{\wedge}_{B}\,{}^{A}\dot{R}_{B} \\ 0_{3\times 3} & {}^{A}\dot{R}_{B} \end{bmatrix}. \tag{3.32}$$

As $v^{\wedge} \in \mathfrak{so}(3)$, we have:

$$ {}^{A}\dot{o}^{\wedge}_{B}\,{}^{A}R_{B} = {}^{A}R_{B}\,{}^{B}v^{\wedge}_{A,B}. \tag{3.33}$$

So, remembering the relation (3.20), we can rewrite (3.34) as

$$ {}^{A}\dot{X}_{B} = {}^{A}X_{B}\,{}^{B}\mathrm{v}_{A,B}\times\,. \tag{3.34}$$

3.5 Forces and Torques

Let's consider a $6D$ vector in which the first three elements are the sum of all the contact forces, while the last three components are the sum of the moments with respect to a given point in space. Similarly to the $6D$ velocity case, also the $6D$ force-torque can be represented in different frames, in which the *origin* of the frame is the point with respect to which the moment is taken and the *orientation* is the one in which the forces and moments are expressed. In particular, we indicate the coordinates of a $6D$ force f with respect to frame B with

$$ {}_{B}\mathrm{f} := \begin{bmatrix} {}_{B}f \\ {}_{B}\tau \end{bmatrix} \in \mathbb{R}^{6}. \tag{3.35}$$

Similarly to what we did for a $6D$ velocities, we can define a linear map to change the coordinates of a $6D$ force from a frame B to another frame A. This coordinate transformation is indicated with ${}_{A}X^{B}$ and written as

$$ {}_{A}\mathrm{f} = {}_{A}X^{B}\,{}_{B}\mathrm{f}. \tag{3.36}$$

The mapping ${}_{A}X^{B}$ is strictly related to the transformation of $6D$ velocities; more precisely, we have

$$ {}_{A}X^{B} := {}^{B}X_{A}^{T} = \begin{bmatrix} {}^{A}R_{B} & 0_{3\times 3} \\ {}^{A}o^{\wedge}_{B}\,{}^{A}R_{B} & {}^{A}R_{B}. \end{bmatrix} \in \mathbb{R}^{6\times 6}. \tag{3.37}$$

Observation 3.5.1. *Given the definition (3.37), the following identity of power holds:*

$$\left\langle {}_B\mathrm{f}, {}^B\mathrm{v}_{A,B} \right\rangle = \left\langle {}_A\mathrm{f}, {}^A\mathrm{v}_{A,B} \right\rangle, \tag{3.38}$$

where the symbol $\langle \cdot, \cdot \rangle$ denotes the scalar product.

Similiarly to what we have done in (3.34), we can compute the time derivative of the transformation of $6D$ wrenches between frames:

$${}_A\dot{X}^B = {}_A X^{B B}\mathrm{v}_{A,B} \bar{\times}^* \tag{3.39}$$

where the dual cross product $\bar{\times}^*$ is defined by

$${}^B\mathrm{v}_{A,B}\bar{\times}^* = \begin{bmatrix} {}^B\omega^\wedge_{A,B} & 0_{3\times 3} \\ {}^B v^\wedge_{A,B} & {}^B\omega^\wedge_{A,B} \end{bmatrix}. \tag{3.40}$$

3.6 Equations of dynamics

In this section, we are going to use a simplified notation: $r := {}^B p$, In order to obtain the equations of the dynamics of a rigid body, one could use the Lagrangian dynamics approach. In order to do that, let's introduce the following functions

$$L(H, \dot{H}) = K(H, \dot{H}) - U(H), \tag{3.41}$$

$$K(H, \dot{H}) = \frac{1}{2} \iiint_{\mathbb{R}^3} \rho(r) \left| \dot{R} r + \dot{o} \right|^2 dr, \tag{3.42}$$

$$U(H) = \iiint_{\mathbb{R}^3} \rho(r) g^T \left(R r + o \right) dr, \tag{3.43}$$

where ρ is the density of the body. Such functions are called, respectively, *Lagrangian*, *kinetic energy* and *potential energy*. We can obtain a simplified expression if we write the left-trivialized Lagrangian:

$$l(H, \mathrm{v}) = k(\mathrm{v}) - U(H), \tag{3.44}$$

$$k(\mathrm{v}) = \frac{1}{2} \mathrm{v}^T \mathbb{M} \mathrm{v}, \tag{3.45}$$

$$U(H) = \begin{bmatrix} g^T & 0 \end{bmatrix} H \begin{bmatrix} mc \\ m \end{bmatrix}, \tag{3.46}$$

where $m \in \mathbb{R}$ is the total mass of the body, $c \in \mathbb{R}^3$ is the center of mass of the body, $I \in \mathbb{R}^{3\times 3}$ is the $3D$ inertia matrix of the body, and $\mathbb{M} \in \mathbb{R}^{6\times 6}$ is the $6D$ inertia matrix of the body, defined as:

$$\mathbb{M} = \begin{bmatrix} \iiint_{\mathbb{R}^3} \rho(r) dr 1_3 & -\left(\iiint_{\mathbb{R}^3} r\rho(r)dr\right)^\wedge \\ \left(\iiint_{\mathbb{R}^3} r\rho(r)dr\right)^\wedge & -\iiint_{\mathbb{R}^3} \rho(r)(r^\wedge)^2 dr \end{bmatrix} = \begin{bmatrix} m1_3 & -(mc)^\wedge \\ (mc)^\wedge & I \end{bmatrix}. \tag{3.47}$$

In (3.47), 1_3 is the 3×3 identity matrix.

Now, let's consider a time interval $[0, T]$ and write the functional *action*:

$$S[H(\cdot)] = \int_0^T L(H(t), \dot{H}(t))dt. \tag{3.48}$$

Theorem 3.6.1 (Principle of Least Action)**.** *The Principle of Least Action states that the trajectory of the system in the interval $t \in [0, T]$ is the stationary point that minimises the system's action functional $[H(\cdot)]$.*

Thanks to theorem 3.6.1, we can obtain the left-trivialized equations of motion (see [17] for details):

$$\dot{H} = H\mathrm{v}^\wedge, \tag{3.49}$$

$$\mathbb{M}\dot{\mathrm{v}} + \mathrm{v}\bar{\times}^*\mathbb{M}\mathrm{v} = \mathbb{M}\begin{bmatrix} R^T g \\ 0_{3\times 1} \end{bmatrix}. \tag{3.50}$$

Moreover, if we consider also the sum of the external wrenches ${}_B\mathrm{f}^x$, we can get:

$$\mathbb{M}\dot{\mathrm{v}} + \mathrm{v}\bar{\times}^*\mathbb{M}\mathrm{v} = \mathbb{M}\begin{bmatrix} R^T g \\ 0_{3\times 1} \end{bmatrix} + {}_B\mathrm{f}^x. \tag{3.51}$$

3.7 Multibody equations of dynamics

3.7.1 Introduction to multibody systems

A multibody system is made by:

- n_j joints, which describe the kinematics of a robot. They connect two links, namely a parent link and a child link. Joints can vary in type, affecting their number of degrees of freedom (dof). In this chapter we will consider the case in which each joint has only 1 degree of freedom; therefore, we will have $n_j = n_{dof} = n$.

- n_L links, which describe the mass properties of a robot through an inertia matrix.

Links and joints are described in the *Universal Robot Description Format* (URDF), an XML file which describes the kinematics, inertial properties and geometry of the robot.
A multibody system can be mathematically described as a graph $G = (\mathcal{L}, \mathcal{J}, \mathcal{W})$; such graph has the following properties:

- each node of the graph is a link; we assume that at least one frame is attached to each link;
- each arc represents a joint;
- there exists one node which is called *base* of the system; such link has a frame called B;
- the graph is undirected, so the adjacency matrix of the graph is symmetric;
- the graph is connected, i.e., there exists at least one path between any two nodes.

If the base of the multi-body system does not have a predetermined fixed position relative to an inertial frame W it is referred to as a floating base.

Definition 3.7.1 (Path). *The path $\pi_B(E) = \{B, \ldots, E\}$ between link B and link E is the ordered sequence of links part of the kinematic graph that connects B to E.*

Definition 3.7.2 (Parent link). *For each link $L \in \mathcal{L}$, if B is the base link, the parent function $\lambda_B : \mathcal{L}/B \rightarrow \mathcal{L}$ maps each link to its parent, with the exclusion of the base link since it is the graph's root. In contexts where B is clearly specified, we omit the subscript.*

Given two links P, C connected by a joint characterized by it position $\theta \in \mathbb{R}$ and velocity $\dot{\theta} \in \mathbb{R}$, we can express the relative velocity as:

$$ {}^{X}\mathrm{v}_{P,C} = {}^{X}S_{P,C}(\theta)\dot{\theta} \in \mathbb{R}^6, \tag{3.52}$$

where we introduced the *joint motion subspace vector* ${}^{X}S_{P,C}(\theta) \in \mathbb{R}^6$. X is a placeholder that selects any of the velocity representations; such quantity is defined as:

$$
\begin{aligned}
{}^{C}S_{P,C}(\theta) &= \left[{}^{C}H_P(\theta)\frac{d\,{}^{P}H_C(\theta)}{d\theta}\right]^{\vee}, \\
{}^{P}S_{P,C}(\theta) &= \left[\frac{d\,{}^{P}H_C(\theta)}{d\theta}{}^{C}H_P(\theta)\right]^{\vee}, \\
{}^{C[P]}S_{P,C}(\theta) &= \begin{bmatrix} \frac{d\,{}^{P}o_C(\theta)}{d\theta} \\ \left[\frac{d\,{}^{P}R_C(\theta)}{d\theta}{}^{C}R_P(\theta)\right]^{\vee} \end{bmatrix}^{\vee}.
\end{aligned}
\tag{3.53}
$$

3.7.2 Position and velocity of the joints

We denote respecitvely with $s, \dot{s} \in \mathbb{R}^n$ the position (also called *shape*) and velocity of all the joints of the system. The configuration of a free-floating mechanical system can be described by the pose of a base link and the generalised joints positions s. In this way, we can describe the kinematics of the floating-base multibody system as:

$$
\begin{cases}
q = \left({}^{W}H_B, s\right) \in \mathcal{Q} = \mathrm{SE}(3) \times \mathbb{R}^n, \\
\dot{q} = \left({}^{W}\dot{H}_B, \dot{s}\right) \in \mathcal{V} = \mathrm{SE}(3) \times \mathbb{R}^n.
\end{cases}
\tag{3.54}
$$

As in the previous sections, we can express the velocity of a rigid body (in this case, a link) as column vector, thanks to left/right/mixed representations:

$$
{}^{X}\nu = \begin{bmatrix} {}^{X}\mathrm{v}_{W,B} \\ \dot{s} \end{bmatrix} \in \mathbb{R}^{6+n},
\tag{3.55}
$$

where ${}^{X}\mathrm{v}_{W,B}$ is the velocity of the base link, $\dot{s} \in \mathbb{R}^n$ are the *joint velocities*, and the generic frame X is a placeholder to select one among the *body representation* $X = B$, *inertial representation* $X = W$, or *mixed* $X = B[W]$ representations.

3.7.3 Link pose

Now, let's write the pose of a link E with respect to the world frame:

$$
{}^{W}H_E(q) = {}^{W}H_B{}^{B}H_E(s) = \begin{bmatrix} {}^{W}R_B & {}^{W}o_B \\ 0_{1\times3} & 1 \end{bmatrix} {}^{B}H_E(s).
\tag{3.56}
$$

3.7.4 Link velocity

Now we want to write the velocity $ls^{E}\mathrm{v}_{W,B}$ as the sum of the base velocity and the velocity between the base B and the link E:

$$^{E}\mathrm{v}_{W,E} = {}^{E}\mathrm{v}_{W,B} + {}^{E}\mathrm{v}_{B,E}.$$

We can express $\mathrm{v}_{B,E}$ as the sum of the velocities between adjacent links in the link path $\pi_B(E)$ between link B and E:

$$\begin{aligned}
^{E}\mathrm{v}_{W,E} &= {}^{E}\mathrm{v}_{W,B} + \sum_{L_i \in \{\pi_B(E)/B\}} {}^{E}\mathrm{v}_{\lambda(L_i),L_i} \\
&= {}^{E}\mathrm{v}_{W,B} + \sum_{L_i \in \{\pi_B(E)/B\}} {}^{E}X_{L_i} {}^{L_i}\mathrm{v}_{\lambda(L_i),L_i} \\
&= {}^{E}\mathrm{v}_{W,B} + \sum_{L_i \in \{\pi_B(E)/B\}} {}^{E}X_{L_i} {}^{L_i}S_{\lambda(L_i),L_i}(s_i)\dot{s}_i,
\end{aligned}$$

where we used the expression of the relative velocity between two adjacent links $^{L}\mathrm{v}_{\lambda(L),L}$ introduced in Equation (3.52).

Expressing the obtained relation in matrix form, we obtain:

$$^{E}\mathrm{v}_{W,E} = \begin{bmatrix} {}^{E}X_X & {}^{E}S_{B,E}(s) \end{bmatrix} \begin{bmatrix} {}^{X}\mathrm{v}_{W,B} \\ \dot{s} \end{bmatrix} = {}^{E}J_{W,E/X} {}^{X}\nu, \tag{3.57}$$

where X is a placeholder that depends on the representation of the system's velocity. Moreover, we introduced the matrix $S_{B,E}(s) \in \mathbb{R}^{6\times n}$ for the joint part, where its i-th column is defined as:

$$^{E}S_{B,E}(:,i) = \begin{cases} {}^{E}X_L {}^{L}S_{\lambda(L),L}(s) & \text{if } L \in \{\pi_B(E)/B\}, \\ 0_{6\times 1} & \text{otherwise.} \end{cases}$$

Moreover, introducing the floating-base Jacobian of link E

$$^{E}J_{W,E/X} = \begin{bmatrix} {}^{E}X_X & {}^{E}S_{B,E}(s) \end{bmatrix}, \tag{3.58}$$

we can rewrite the equation (3.59) as follows:

$$^{E}\mathrm{v}_{W,E} = {}^{E}J_{W,E/X} {}^{X}\nu. \tag{3.59}$$

3.7.5 Dynamics

Similiarly to what we have done in section 3.6, we can write the Lagrangian function for a multibody system:

$$l(q,\nu) = k(q,\nu) - U(q), \tag{3.60}$$

$$k(q,\nu) = \sum_{L\in\mathbb{L}} {}^{L}\mathrm{v}_{A,L}^{T} \mathbb{M}_{L} {}^{L}\mathrm{v}_{A,L}, \tag{3.61}$$

$$U(q) = -\sum_{L\in\mathbb{L}} \begin{bmatrix} {}^{A}g^{T} & 0 \end{bmatrix} {}^{A}H_{L}(q) \begin{bmatrix} m_{L} {}^{L}c_{L} \\ m_{L} \end{bmatrix}. \tag{3.62}$$

We can express the functions above in a more compact way with the left-trivialization:

$$k(q,\nu) = \frac{1}{2}\nu^{T} M(s)\nu, \tag{3.63}$$

$$U(q) = -\begin{bmatrix} {}^{A}g^{T} & 0 \end{bmatrix} {}^{A}H_{B} \begin{bmatrix} m {}^{B}c(s) \\ m \end{bmatrix}, \tag{3.64}$$

where $M(s) \in \mathbb{R}^{n+6\times n+6}$ is the system's mass matrix, defined as:

$$M(s) = \sum_{L\in} J_{L}^{T}(s)_{L}\mathbb{M}_{L} J_{L}(s), \tag{3.65}$$

m is the total mass of the multibody system

$$m := \sum_{L} m_{L}, \tag{3.66}$$

and ${}^{B}c(s)$ is the total center of mass of the multibody system, defined as:

$$\begin{bmatrix} {}^{B}c(s) \\ 1 \end{bmatrix} := \frac{1}{m}\sum_{L} {}^{B}H_{L}(s) \begin{bmatrix} m_{L} {}^{L}c_{L} \\ m_{L} \end{bmatrix}. \tag{3.67}$$

Using again the theorem 3.6.1, we obtain the equations of motion of a multibody system (see [17] for details):

$$M(s)\dot{\nu} + C(q,\nu)\nu + G(q) = \begin{bmatrix} 0_{6\times 1} \\ \tau \end{bmatrix} + \sum_{L\in\mathbb{L}} J_{L}^{T} {}_{L}\mathrm{f}^{x}, \tag{3.68}$$

where we have:

$$
\begin{aligned}
M(s) &= \sum_{L} J_L^T \, {}_L\mathbb{M}_L J_L, && (3.69) \\
C(q,\nu) &= \sum_{L} J_L^T \left[\left(\mathrm{v}_L \bar{\times}^* \, {}_L\mathbb{M}_L + {}_L\mathbb{M}_L \mathrm{v}_L \times \right) J_L + {}_L\mathbb{M}_L \dot{J}_L \right], && (3.70) \\
G(q) &= -\, M(s) \begin{bmatrix} {}^A R_B^T \, {}^A g \\ 0_{3\times 1} \\ 0_{n\times 1} \end{bmatrix}. && (3.71)
\end{aligned}
$$

The matrix $C(q,\nu)$ is called *Coriolis* matrix.

3.8 Conclusions

In this chapter, we have established the mathematical groundwork necessary for studying rigid body dynamics, focusing on key concepts such as vectors, matrices, and transformations. By understanding these fundamentals, we can accurately describe the position and orientation of rigid bodies, crucial for analyzing and controlling their motion.

We began with essential mathematical definitions, including vector spaces, inner products, and special orthogonal groups. These concepts laid the foundation for exploring frame kinematics, where we discussed the representation of points and frames, and how transformations between different frames are performed using homogeneous matrices.

We then introduced the study of velocities and accelerations in the context of rigid body dynamics. We introduced the notions of left and right trivialized velocities, and demonstrated how to compute these velocities using the time derivatives of transformation matrices. Additionally, we explored the relationship between different representations of velocities and their transformations.

Finally, we examined the concepts of forces and torques, and their representation in various frames. This understanding is critical for analyzing the interaction of rigid bodies with their environment, particularly in applications like humanoid robotics, where precise control of motion is essential.

As already said at the beginning of the chapter, the knowledge acquired will be important in defining the loss function of the Neural Networks for the calibration of the force-torque sensors.

Part II

Contribution

Chapter 4

Data collection and postprocessing

In the previous chapters, we have laid the theoretical foundations necessary for the development of this thesis, i.e. the fundamental notions regarding $6D$ strain gauge force torque sensors, artificial neural networks and dynamics of rigid multibody systems. Now that we have all the necessary tools, we will delve into the core of the present work.

This chapter is dedicated to the crucial step in the development of a neural network model: the construction of the dataset. We will first discuss the process of data collection in-situ, presenting the methodologies and some details about the data in section 4.1. Instead, in section 4.2 the post processing necessary to make the data usable for training the Neural Networks (NN) will be explained.

4.1 Collection of the data in-situ

In order to collect the datasets for the NNs' trainings, we mainly had two possibilities:

- Collect datasets with the sensor removed from the robot, i.e. *ex-situ*, applying known weights.
- Collect datasets with the sensor mounted on the robot, i.e. *in-situ*. How the *ground truth* will be obtained will be discussed later.

In our case, we opted for the second method. This choice is due to the fact that, as noted in past years, calibrating Force-Torque (FT) sensors with datasets collected ex-situ is not the best choice, as the real output of such sensors is influenced by many hidden variables, even including the tightening force of the screws [5].

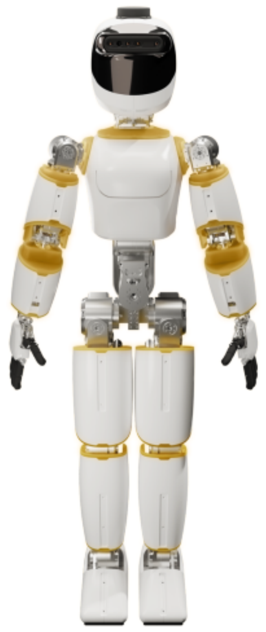


Figure 4.1: The ErgoCub robot

The experiments were conducted on the ErgoCub robot, developed by the Italian Institute of Technology (figure 4.1).

4 types of experiments were conducted:

- Walking;
- Balancing on one foot;
- Balancing on half foot;
- Moving a leg while the robot was on the pole.

This robot has different sensors to collect various types of data, including:

- $6D$ force-torque sensors, to measure the wrenchs $\mathrm{f} \in \mathbb{R}^6$; the first 3 components (forces) are measured in N, while the other 3 (torques) are measured in Nm;
- accelerometers on the FTs, to measure accelerations $a \in \mathbb{R}^3$; the unit of measurement is $\frac{m}{s^2}$;

- gyroscopes on the FTs, to measure angular velocities $\omega \in \mathbb{R}^3$; the unit of measurement is $\frac{rad}{s}$;
- thermometers on the FTs, to measure temperatures $T \in \mathbb{R}$; the unit of measurement is $°C$;
- encoders on the FTs, to measure the joints' positions $s \in \mathbb{R}^{n_{dof}}$; the unit of measurement is rad;

In particular, the robot ErgoCub has 8 force-torque sensors:

- 2 per foot; the foot of the robot is composed by two parts, front part and rear part. On each part, there is an FT sensor;
- 1 per upper leg;
- 1 per arm.

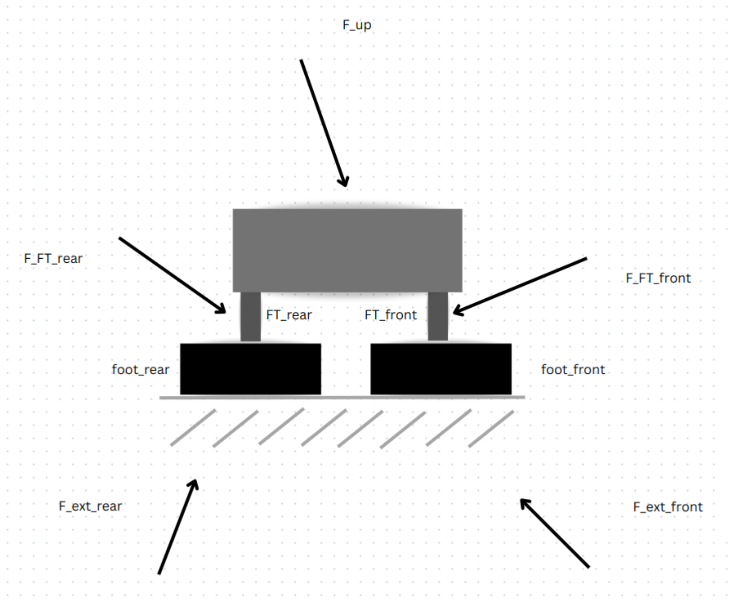
The presence of two FTs per foot must be taken into consideration when calculating the expected wrenches. The reason is that, when the entire foot, for example the right one, is in contact with the ground, the individual expected wrenches on `r_foot_front_ft` and `r_foot_rear_ft` cannot be calculated exactly because the structure is hyperstatic; let's see it with an example. In figure 4.2a we can see a scheme of the foot of the robot, with the links `foot_front` and `foot_rear` in contact with the ground. The small vertical rectangles represent the two force-torque sensors, each of which is positioned on a fixed joint. Above the FTs, there is the remaining part of the robot.

The statics equations of this subsystem can be written as:

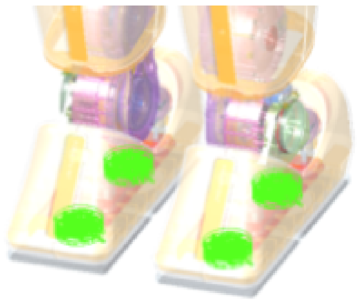
$$\begin{cases} F^{\texttt{FT\_rear}} + F^{\texttt{FT\_front}} + F^{up} = 0, \\ F^{\texttt{ext\_front}} + F^{\texttt{FT\_front}} = 0, \\ F^{\texttt{ext\_rear}} + F^{\texttt{FT\_rear}} = 0. \end{cases}$$

Assuming known F^{up}, the system has four unknowns and three equations, so it is undetermined. At most, only the sum of $F^{\texttt{FT\_front}}$ and $F^{\texttt{FT\_rear}}$ can be precisely determined. The concept extends to FTs on the upper legs when both feet are in contact with the ground. For this reason, it's important to understand in which *timestamps* the robot is in contact with the ground through:

- a link (for example `r_foot_front` or `root_link`);



(a) Simplified scheme of ErgoCub foot



(b) The feet of the robot in transparency with FT sensors highlighted in green. Credits to [23].

Figure 4.2: Feet of ErgoCub robot.

- two links (an entire foot);
- more than two links (both feet);

The timestamps of the last type will be thrown away, while those of the first two types will be treated with switches (as we will see in section 5.4).

4.2 Postprocessing of the data

As anticipated in section 4.1, with the collection of data in-situ, one does not have the ground truth for the wrenches that should be measured on the force-torque sensors. Therefore, to obtain the expected values of such wrenches, we needed to use a library for multibody dynamics, iDynTree [21]. Such library is written in C++, but it has also Python and Matlab bindings. One of the algorithms in this library allows to calculate, after knowing the kinematics and the URDF model of the robot, the expected wrenches on the FTs. In order to do that, basically iDynTree divides the robot model in smaller submodels and, using the equation (3.68) for each submodel, it calculates the forces transmitted between one submodel and another. The

kinematics is basically given by the knowledge of position (s), velocity ($\dot{s}$) and acceleration ($\ddot{s}$) of the joints. Such data can be collected during the experiments discussed above.
However, the data for $\dot{s}$ and $\ddot{s}$ is quite noisy, which could adversely affect the calculation of expected wrenches using iDynTree. For this reason, it was chosen not to use the $\dot{s}$ and $\ddot{s}$ data collected in the experiment, but rather to calculate these quantities starting from the s data collected. For this purpose, a Savitzky-Golay filter has been used [22].

After having done this, iDynTree was used to calculate the expected wrenches, i.e. the values that will be the labels for training the NNs. Furthermore, with iDynTree other quantities useful for training purposes were also calculated, i.e. the homogeneous transformations needed to express wrenches in different frames.

4.3 Conclusions

In this chapter, we covered the essential steps of building the dataset for our neural network models. We chose to collect data in situ using the ErgoCub robot, conducting various types of experiments. We also explained the techniques used for postprocessing, i.e. the techniques for smoothing data with a Savitzky–Golay filter and computation of expected wrenches using iDynTree.

In the next chapter, we will use the datasets obtained in this chapter to train the Neural Network models for the calibration of the force-torque sensors.

Chapter 5

Neural Networks for FT modelling

While in the last chapter the procedure for collecting and processing data was presented, this chapter is dedicated to the presentation of the Neural Network (NN) models to calibrate the force-torque (FT) sensors of the upper legs and feet and the results obtained with such models.

First of all, in section 5.1, the training and testing datasets necessary for the development of NN models will be discussed. In section 5.2 the frameworks used for the development of the NN models will be examined. In section 5.3 the topic will be the inputs of the Neural Network models. In section 5.4, the loss functions used for the training of the models will be presented. Then, we will enter in the main contribution of this work: in sections 5.5, 5.6, 5.7, 5.8 will be presented the architectures and results of the models for the calibration of FTs for right leg, left leg, right foot and left foot respectively. More specifically, for each force-torque sensors will be developed two models with the same architecture, but with different inputs:

- raw wrench (i.e. the $6D$ vector containing force and torque measured by the sensor), linear acceleration of the sensor (a $3D$ vector), angular velocity of the sensor (a $3D$ vector) and temperature of the sensor (a scalar); so, totally, this model takes as input a $13D$ vector;

- raw wrench and temperature of the sensor; so, totally, this model takes as input a $7D$ vector.

Moreover, in sections 5.5, 5.6 a comparison of the results with polynomial models [1] will be carried out.

5.1 Datasets used

The collection and postprocessing of datasets have been presented in chapter 4. Many experiments on the robot have been carried out to collect datasets. We can basically distinguish four types of experiments: balancing on a whole foot, balancing on half a foot, robot on the pole while moving a leg with weights attached to the foot, walking. In section 4.1 we have already showed that the label for the supervised training are not always available, according to the number of robot links in contact with ground. In order to consider that, some switches will be used (as we will see in section 5.4).

For each model we develop in this chapter, we will need a training dataset and a testing dataset. Both datasets will contain data related to balancing, pole and walking experiments.

5.2 Framework and libraries used for development of NN models

In this work, Neural Network models are developed in the Pytorch framework [24]. The architecture of a NN is described by some hyperparameters, such as the number of layers, the number of neurons, the batch size for training, etc. For this reason, it is of fundamental importance to try different possible configurations to have better results in the testing phase. To accelerate this hyperparameter search, an automatic hyperparameter optimization software framework has been used, Optuna [25].

5.3 Inputs of the NNs

Currently, linear models are used in the real robot to calibrate the force-torque sensors. These models essentially represent functions of the type:

$$f : \mathbb{R}^6 \rightarrow \mathbb{R}^6 \tag{5.1}$$

In practice, these models take as input the potential differences returned by the Wheatstone bridge circuit (after amplification etc., see chapter 1 for details) and return as output the wrench applied on the sensor. In this project, we will perform a secondary calibration using the wrenches obtained from the primary linear calibration already applied to the real robot. Furthermore, other inputs will also be considered, to try to exploit all possible information that could influence the calibration output. Below, the complete list of neural network inputs that we will develop for each sensor:

- the wrench f $\in \mathbb{R}^6$ on the FT, output of the primary linear calibration; it will be often called *raw wrench* in this work;
- the linear acceleration $a \in \mathbb{R}^3$ of the FT;
- the angular velocity $\omega \in \mathbb{R}^3$ of the FT;
- the temperature $T \in \mathbb{R}$ of the FT.

So, totally, we are looking for functions of the type:

$$f : \mathbb{R}^{13} \to \mathbb{R}^6 \tag{5.2}$$

Furthermore, we will also train other models using fewer inputs, specifically:

- the wrench f $\in \mathbb{R}^6$ on the FT, output of the primary linear calibration;
- the temperature $T \in \mathbb{R}$ of the FT.

This is done mainly for two reasons:

- while the influence of temperature on the sensor is physically clear (temperature alters deformation of straing gauge), theoretically there shouldn't be a physical influence of linear acceleration and angular velocity on the measurement of FT;
- the data of linear acceleration and angular velocity could be noisy.

So, in this second case, we will be looking for a function f such that:

$$f : \mathbb{R}^{7} \to \mathbb{R}^6 \tag{5.3}$$

The functions (5.2) and (5.3) will be identified through the training process discussed in section 2.2. In order to do that, we have to decide the loss functions to minimize during the training of each model; this will be the topic of next section.

5.4 Definition of the loss functions

As already discussed in chapter 2, a fundamental component for training a NN is the definition of its loss function.

Depending on which sensor we will consider (for example the one on the right leg), we will use different loss function components.

5.4.1 Loss function for the FTs on the legs

Below, the loss function used for the trainings of the NN for the FTs on the legs:

$$\mathcal{L}_{Leg} = \alpha_{ExpectedFootFront}\lambda_{expected}\mathcal{L}_{ExpectedWrenchLeg}, \tag{5.4}$$

where

- $\mathcal{L}_{ExpectedWrenchLeg} = mse(f^{expected} - f^{NN})$;
  here, mse stands for Mean Squared error, the loss function discussed in subsection 2.2.2, while $f^{expected}$ and f^{NN} are respectively the ground truth and the output of the Neural Network;

- $\alpha_{ExpectedLeg}$ is a switch such that, denoting the timestamp as t:

  $$\alpha_{ExpectedLeg} = \begin{cases} 1 & \text{if exact groundtruth is available,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

  This switch allows to discriminate the timestamps as discussed in section 4.1. Basically, in the case of FTs on the legs, the groundtruth is always available except when the robot is in contact with the ground through both feet.

- $\lambda_{expected}$ is a weight that can be freely chosen.

5.4.2 Loss function for the FTs on the feet

As said before, on each foot there are two sensors: one on the front part of the foot, one on the rear. So, totally, there are 2 FTs on the right foot and 2 on the left foot. Below, the loss function used for the trainings of the NN for the FTs on the feet:

$$\begin{aligned} \mathcal{L}_{FootFront} = {} & \alpha_{ExpectedFootFront}\lambda_{ExpectedFootFront}\mathcal{L}_{ExpectedWrenchFootFront} \\ & + \alpha_{ExpectedFootSum}\lambda_{ExpectedFootSum}\mathcal{L}_{ExpectedWrenchFootSum} \end{aligned} \tag{5.6}$$

$$\begin{aligned} \mathcal{L}_{FootRear} = {} & \alpha_{ExpectedFootRear}\lambda_{ExpectedFootRear}\mathcal{L}_{ExpectedWrenchFootRear} \\ & + \alpha_{ExpectedFootSum}\lambda_{ExpectedFootSum}\mathcal{L}_{ExpectedWrenchFootSum} \end{aligned} \tag{5.7}$$

where

- $\mathcal{L}_{FootFront}$ and $\mathcal{L}_{FootRear}$ are analogous to the loss function component discussed in (5.4) for the legs FTs;

- $\mathcal{L}_{ExpectedWrenchFootSum}$ is the component of loss function related to the discrepancy between the sum of wrenches expected vs the sum of wrenches output of the NNs; it's worth noticing that, in order to compute the sum of two wrenches, they have to be expressed in the same frame. The single wrenches, both the expected and the output of the NNs, are expressed in the proper sensor frame: for example, the wrench on the right foot front FT sensor is expressed with respect to the frame `l_foot_front_ft`. As seen in section 3.5, we need the homogeneous transformation discussed in equation (3.36) to express a wrench in a different frame (see figure 5.1 to visualize the frames `r_foot_front_ft`, `r_foot_rear_ft`, `r_sole`). In this case, we want to express the wrenches on the front and of the rear parts with respect to the frame `r_sole` (in the case of right foot) or the frame `l_sole` (in the case of left foot). For that reason, this loss function component is expressed as follows:

$$\mathcal{L}_{ExpectedWrenchFootSum} = mse({}_{sole}f^{expected}_{foot} - {}_{sole}f^{N}N_{foot}), \tag{5.8}$$
$$f^{expected}_{foot} = {}_{sole}X^{front}{}_{front}f^{expected}_{front} + {}_{sole}X^{rear}{}_{rear}f^{expected}_{rear}, \tag{5.9}$$
$$f^{NN}_{foot} = {}_{sole}X^{front}{}_{front}f^{NN}_{front} + {}_{sole}X^{rear}{}_{rear}f^{NN}_{rear}. \tag{5.10}$$

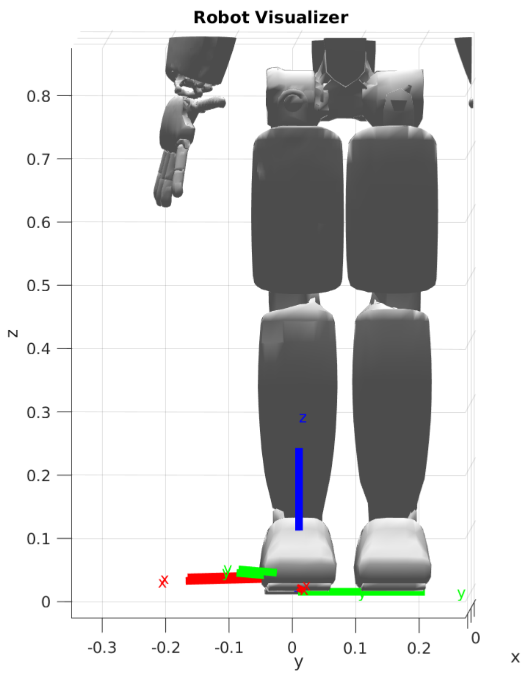
- $\alpha_{ExpectedFootFront}$ and $\alpha_{ExpectedFootRear}$ are switches that could be equal to 0 or 1; namely:

$$\alpha_{ExpectedFootPart} = \begin{cases} 1 & \text{if exact groundtruth is available for the single wrench,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.11}$$

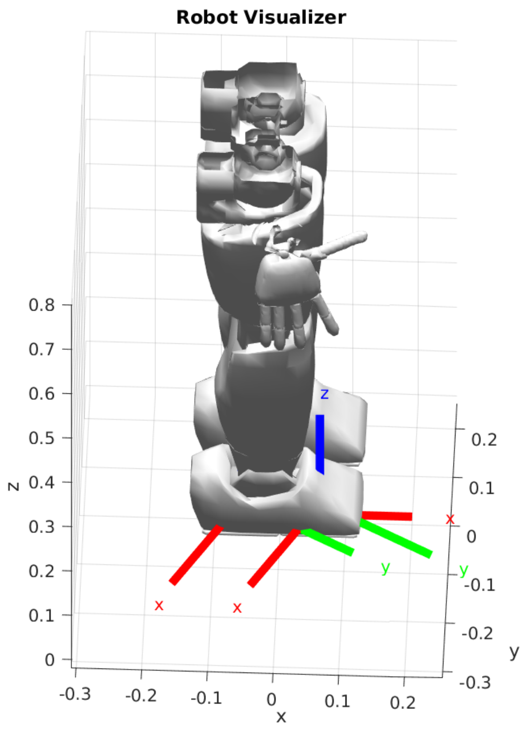
These switches allow to discriminate the timestamps as discussed in section 4.1. Basically, in the case of FTs on feet, the groundtruth for individual wrenches is only available in experiments where the robot performs balancing on half a foot or is on the pole (on which it is fixed via the root link).

- $\alpha_{ExpectedFootSum}$ is a switch such that:

$$\alpha_{ExpectedFootSum} = \begin{cases} 0 & \text{if exact groundtruth is available for the single wrench,} \\ 1 & \text{otherwise.} \end{cases} \tag{5.12}$$



(a) Front view



(b) Side view

Figure 5.1: Visualization of frames `r_foot_front_ft`, `r_foot_rear_ft`, `r_sole`

This switch allows to discriminate the timestamps as discussed in section 4.1. Instead, in the moments in which the robot is in contact with the ground through a single full foot, the groundtruth for the sum of the wrenches is available; however it should be noted that this sum must be made by expressing the two wrenches with respect to the same frame. In this work, we will express such wrenches with respect to the frame `r_sole` in the case of right foot, compared to the frame `l_sole` in the case of left foot.

- $\lambda_{ExpectedFootFront}$, $\lambda_{ExpectedFootRear}$ and $\lambda_{ExpectedFootSum}$ are weights that can be freely chosen. For training models on feet, such fixed weights can be important to ensure that the various components of the loss function have approximately the same orders of magnitude.

5.5 Models for FT on right leg

In this section, we will first present the architecture of the NN models (with both 13 and 7 inputs) for the FT on the right leg. Subsequently, we will present the obtained results and compare them with the current calibration on the robot and with the polynomial models proposed by [1].

5.5.1 Architecture of the model

In this section, the main characteristics of the model will be presented:

- **Batch size:** 1000
- **Number of epochs:** 8
- **Normalization:** max normalization (i.e., each component of the input is divided by its maximum value in the training dataset)
- **Number of layers:** 2 (plus the output layer)
- **Number of neurons per layer:** 100
- **Dropout probabilities:** [0.0, 0.0]
- **Activation functions:** [LeakyReLU, ReLU]
- **Optimizer:** Adam (weight_decay=0)
- **Learning rate:** 0.0094023971181947206
- **Learning rate scheduler:** ExponentialLR, $\gamma = 0.99$, applied only when training loss reaches the threshold 0.005 (this threshold is not reached during this training)
- **Initialization of the weights of the neural network:** `xavier_uniform` [26]
- $\lambda_{\text{expected}} = 407.3938302752665$

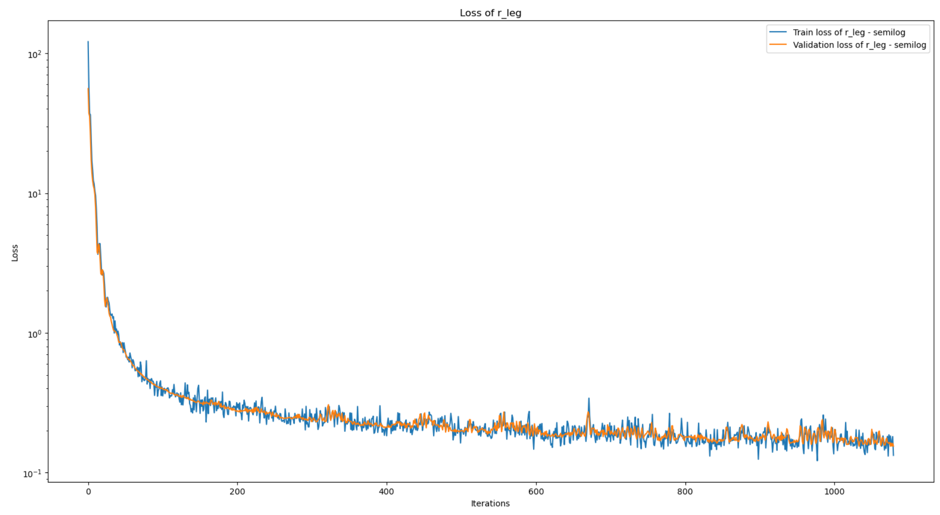


Figure 5.2: Loss function during training of NN for right leg FT

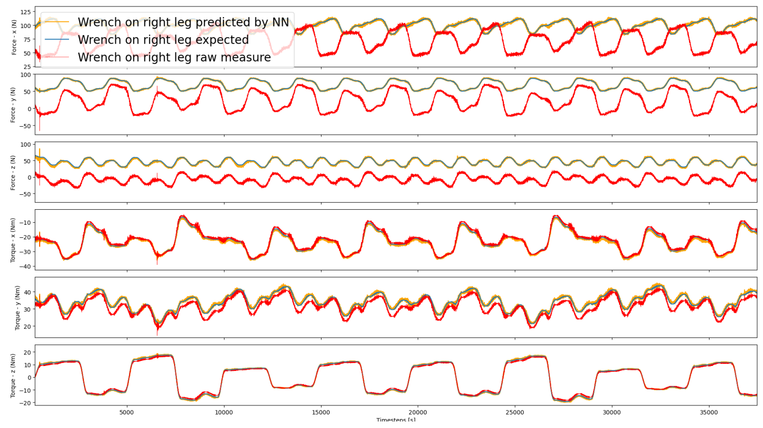
5.5.2 Results for model with 13 inputs

In figure 5.2 the loss function during training is reported. Instead, in figure 5.3 is reported the wrench predicted by the Neural Network in the test phase. In such plots, the prediction is compared with the raw wrench (i.e. the output of the actual calibration on the robot) and the expected wrench (i.e. the label). In figure 5.4 is reported the discrepancy between the NN prediction and the label, and the discrepancy between the raw wrench and the label. In tables 5.1, 5.2 and 5.3 are reported respectively the MSE, MAE and RMSE committed by the NN and the actual calibration (raw value) with respect to the labels.

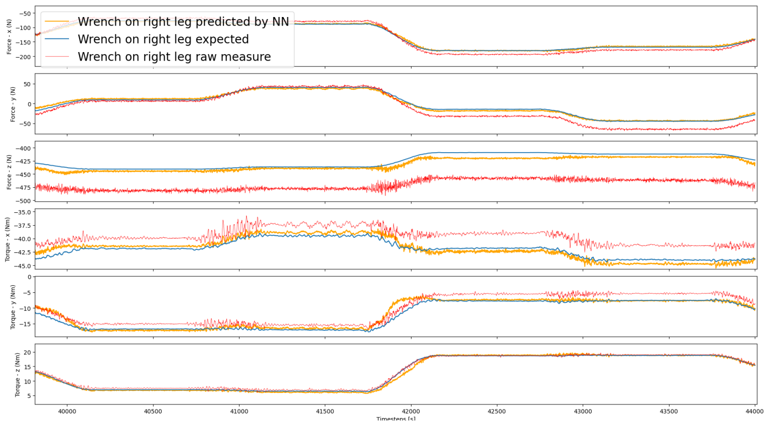
5.5.3 Results for model with 7 inputs

The architecture is the same of the model with 13 inputs presented in section 5.5.1. For that reason, we will present directly the results obtained.

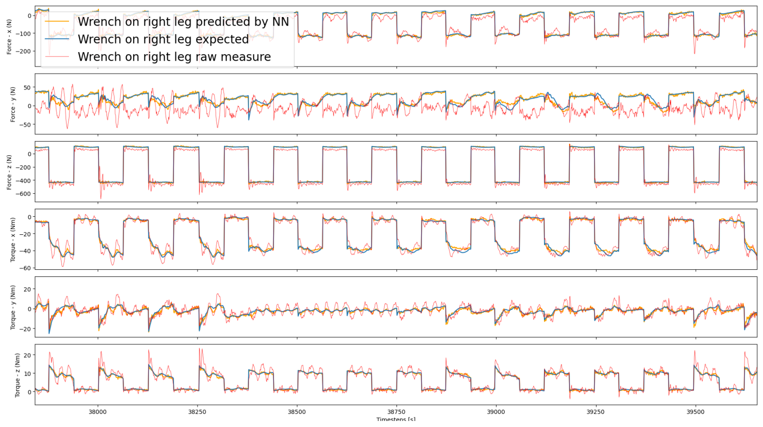
In figure 5.2 the loss function during training is reported. Instead, in figure 5.6 is reported the wrench predicted by the Neural Network in the test phase. In such plots, the prediction is compared with the raw wrench (i.e. the output of the actual calibration on the robot) and the expected wrench (i.e. the label). In figure 5.7 is reported the discrepancy between the NN prediction and the label, and the



(a) Zoom on part of dataset related to pole experiment



(b) Zoom on part of dataset related to balancing experiment



(c) Zoom on part of dataset related to walking experiment

Figure 5.3: Comparison of wrenches predicted by NN with expected wrenches and raw values for right leg FT test phase (13 inputs)

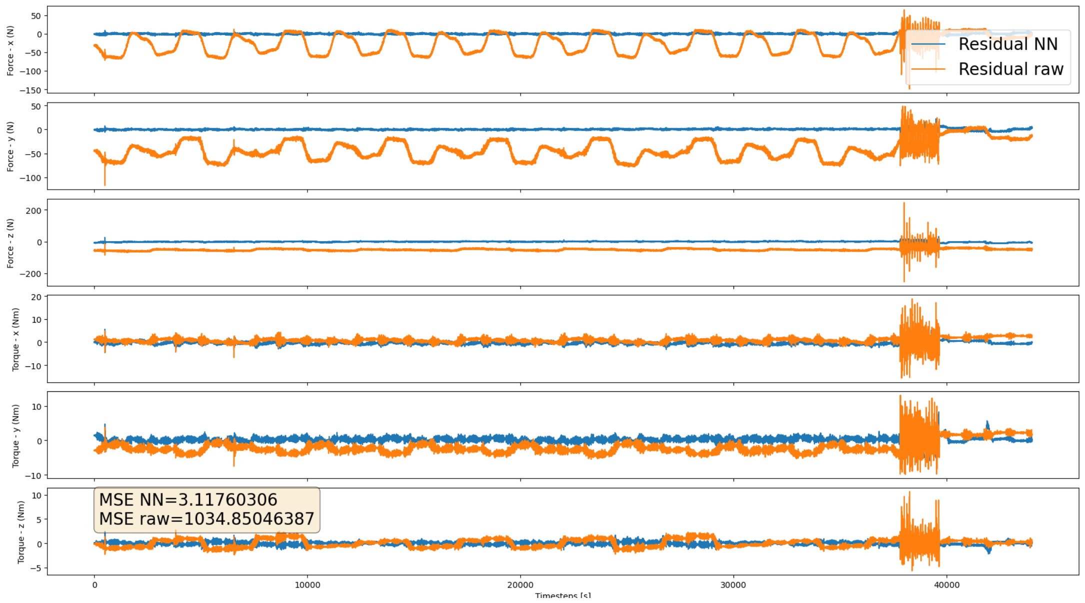


Figure 5.4: Residual obtained in test phase for right leg FT (13 inputs)

MSE	**NN**	**raw**
MSE global	3.12	1034.85
MSE force	5.84	2066.15
MSE torque	0.39	3.55
MSE fx	3.37	1334.77
MSE fy	3.26	2198.65
MSE fz	10.90	2665.04
MSE tx	0.56	2.28
MSE ty	0.52	7.60
MSE tz	0.10	0.77

Table 5.1: Right leg FT: Mean Squared Error (MSE) for Neural Network (NN) and raw data (13 inputs)

MAE	**NN**	**raw**
MAE global	1.01	21.13
MAE force	1.58	40.82
MAE torque	0.44	1.45
MAE fx	1.30	28.68
MAE fy	1.17	42.68
MAE fz	2.25	51.09
MAE tx	0.58	1.07
MAE ty	0.52	2.54
MAE tz	0.23	0.73

Table 5.2: Right leg FT: Mean Absolute Error (MAE) for Neural Network (NN) and raw data (13 inputs)

RMSE	**NN**	**raw**
RMSE global	1.77	32.17
RMSE force	2.42	45.45
RMSE torque	0.63	1.88
RMSE fx	1.83	36.53
RMSE fy	1.80	46.89
RMSE fz	3.30	51.62
RMSE tx	0.75	1.51
RMSE ty	0.72	2.76
RMSE tz	0.31	0.88

Table 5.3: Right leg FT: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data (13 inputs)

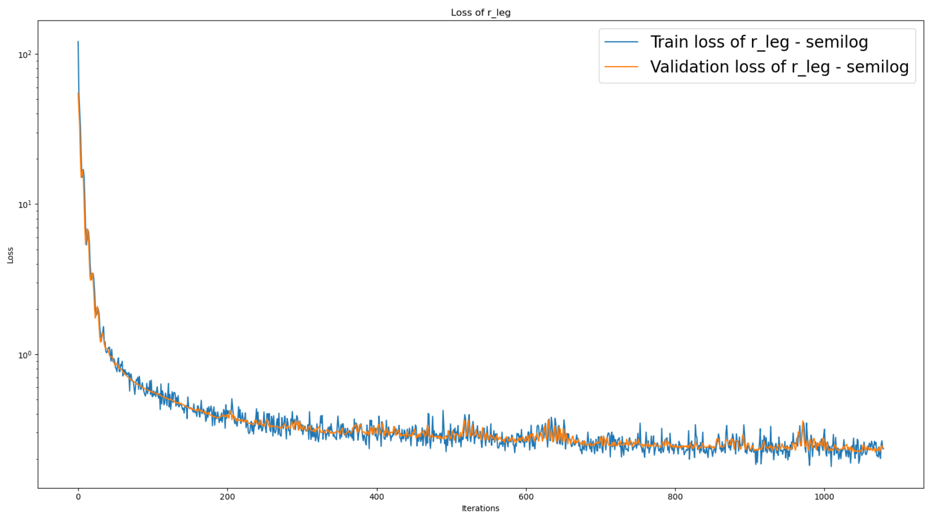
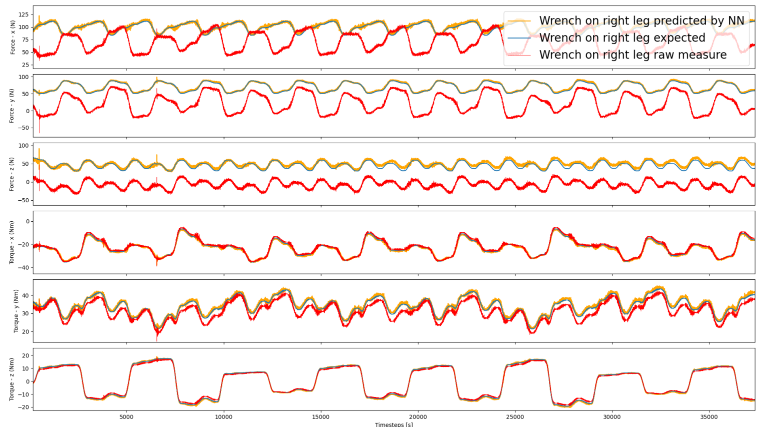


Figure 5.5: Loss function during training of NN for right leg FT

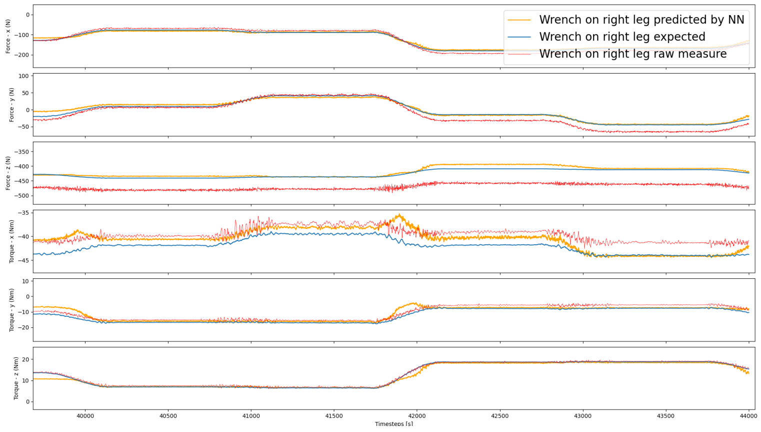
discrepancy between the raw wrench and the label. In tables 5.4, 5.5 and 5.6 are reported respectively the MSE, MAE and RMSE committed by the Neural Network and the actual calibration (raw value) with respect to the labels.

MSE	**NN**	**raw**
MSE global	8.34	1034.85
MSE force	16.02	2066.15
MSE torque	0.67	3.55
MSE fx	8.73	1334.77
MSE fy	7.01	2198.65
MSE fz	32.31	2665.04
MSE tx	0.80	2.28
MSE ty	1.00	7.60
MSE tz	0.19	0.77

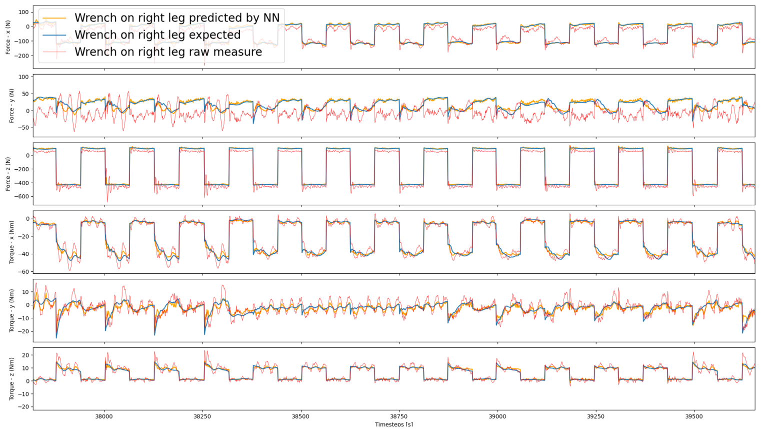
Table 5.4: Right leg FT with 7 inputs: Mean Squared Error (MSE) for Neural Network (NN) and raw data



(a) Zoom on part of dataset related to pole experiment



(b) Zoom on part of dataset related to balancing experiment



(c) Zoom on part of dataset related to walking experiment

Figure 5.6: Comparison of wrenches predicted by NN (with 7 inputs) with expected wrenches and raw values for right leg FT test phase

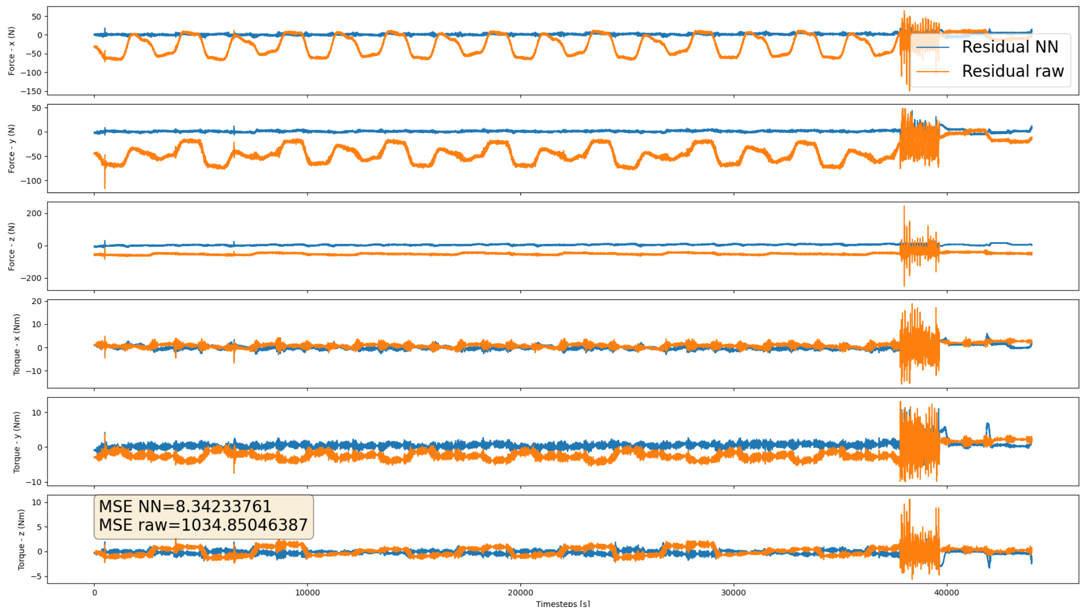


Figure 5.7: Residual obtained in test phase for right leg FT (7 inputs)

MAE	**NN**	**raw**
MAE global	1.73	21.13
MAE force	2.92	40.82
MAE torque	0.54	1.45
MAE fx	2.25	28.68
MAE fy	1.85	42.68
MAE fz	4.66	51.09
MAE tx	0.63	1.07
MAE ty	0.67	2.54
MAE tz	0.31	0.73

Table 5.5: loss functionRight leg FT with 7 inputss: Mean Absolute Error (MAE) for Neural Network (NN) and raw data

5.5.4 Comparison with the results obtained by using polynomial models

In this subsection, we are going to compare the results obtained with our Neural Network model with the one obtained with polynomials models developed by [1]. Of

RMSE	**NN**	**raw**
RMSE global	2.89	32.17
RMSE force	4.00	45.45
RMSE torque	0.82	1.88
RMSE fx	2.95	36.53
RMSE fy	2.65	46.89
RMSE fz	5.68	51.62
RMSE tx	0.90	1.51
RMSE ty	1.00	2.76
RMSE tz	0.44	0.88

Table 5.6: loss functionRight leg FT with 7 inputss: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data

course, the training and testing datasets are the same used for the NN model. Below, the parameters which describe the polynomial model:

- np: the order of the polynomial;
- ny: number of outputs. In this case, it is set to 6, which is the dimension of expected F/T forces and torques y;
- nu: number of inputs. In this work, it is set to 7, which is the dimension of actual F/T measurements, in addition to the internal temperature measurement;
- na: number of delayed samples of the output. It captures the degree of the dynamics in the model;
- nb: number of delayed samples of the input.

Here, the software developed by [1] will be used to calculate the parameters and therefore build the polynomial models from order $np = 1$ up to $np = 4$. Moreover, the case $na = nb = 0$ will be considered.

Below the plot of the wrenches during the test phase. For each component of the wrench, the RMSEs committed by each model (from degree 1 to 4) and by the workbench (i.e. the raw data, the output measurements from the current calibration on the robot) are reported.

Comparing the RMSEs in figure 5.8 with the ones obtained for Neural Network models (model with 13 inputs in table 5.3 and model with 7 inputs in table 5.6), we

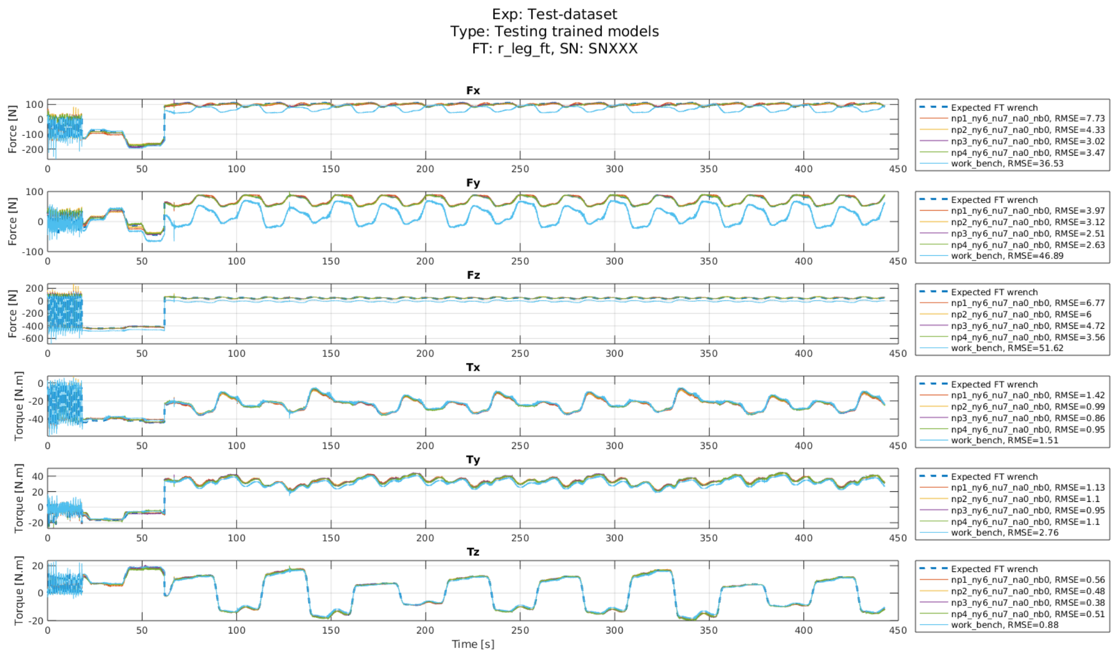


Figure 5.8: Test phase for right leg FT with polynomial models

can see that the NN is able to perform better than the polynomial model in every component of the wrench.

5.6 Models for FT on left leg

In this section, we will first present the architecture of the NN models (with both 13 and 7 inputs) for the FT on the left leg. Subsequently, we will present the obtained results and compare them with the current calibration on the robot and with the polynomial models proposed by [1].

5.6.1 Architecture of the model

In this section, the main characteristics of the model will be presented:

- **Batch size:** 4096
- **Number of epochs:** 100

- **Normalization:** max normalization (i.e., each component is divided by its maximum value in the dataset)
- **Number of layers:** 2 (plus the output layer)
- **Number of neurons per layer:** 64
- **Dropout probabilities:** [0.0, 0.0]
- **Bias in the linear transformations in each layer:** true
- **Activation functions:** [LeakyReLU, ReLU]
- **Optimizer:** Adam (weight_decay=0)
- **Learning rate:** 0.000540239711819472006
- **Learning rate scheduler:** ExponentialLR, $\gamma = 0.99$, applied only when training loss reaches the threshold 0.005 (this threshold is not reached during this training)
- **Initialization of the weights of the neural network:** `xavier_uniform`
- $\lambda_{\text{expected}} = 407.3938302752665$

5.6.2 Results for model with 13 inputs

In figure 5.9 the loss function during training is reported. Instead, in figure 5.10 is reported the wrench predicted by the Neural Network in the test phase. In such plots, the prediction is compared with the raw wrench (i.e. the output of the actual calibration on the robot) and the expected wrench (i.e. the label). In figure 5.11 is reported the discrepancy between the NN prediction and the label, and the discrepancy between the raw wrench and the label. In tables 5.7, 5.8 and 5.9 are reported respectively the MSE, MAE and RMSE committed by the NN and the actual calibration (raw value) with respect to the labels.

5.6.3 Results for model with 7 inputs

The architecture is the same of the model with 13 inputs presented in section 5.6.1. For that reason, we will present directly the results obtained.

In figure 5.12 the loss function during training is reported. Instead, in figure

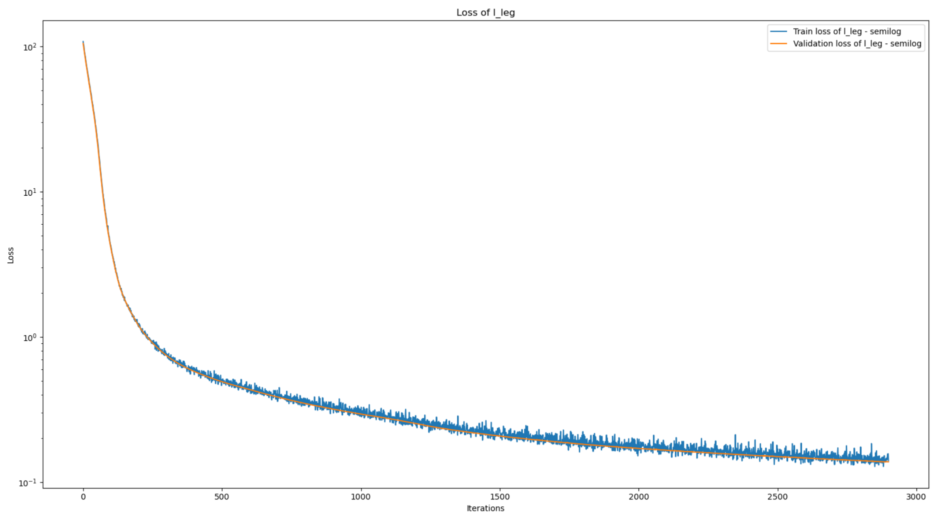
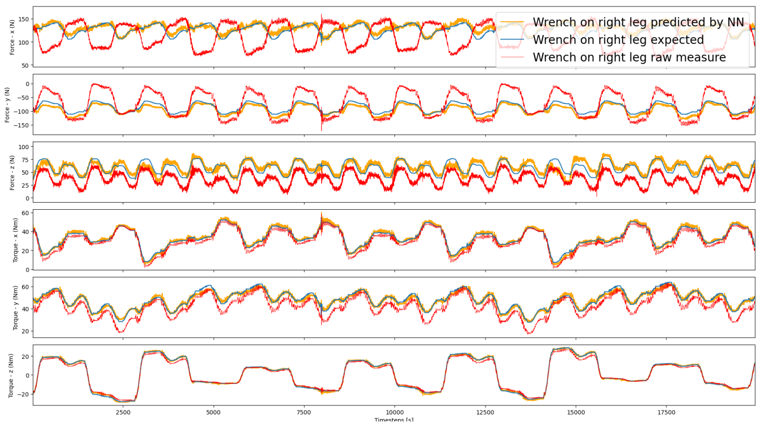


Figure 5.9: Loss function during training of NN for left leg FT

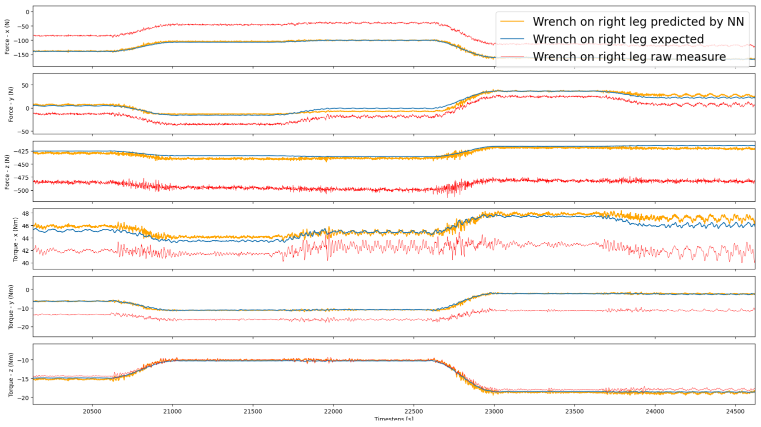
MSE	NN	raw
MSE global	26.81	661.69
MSE force	52.05	1304.93
MSE torque	1.56	18.45
MSE fx	29.25	1768.69
MSE fy	79.31	1014.31
MSE fz	47.60	1131.80
MSE tx	2.05	11.86
MSE ty	2.17	40.96
MSE tz	0.46	2.55

Table 5.7: Left leg FT: Mean Squared Error (MSE) for Neural Network (NN) and raw data (13 inputs)

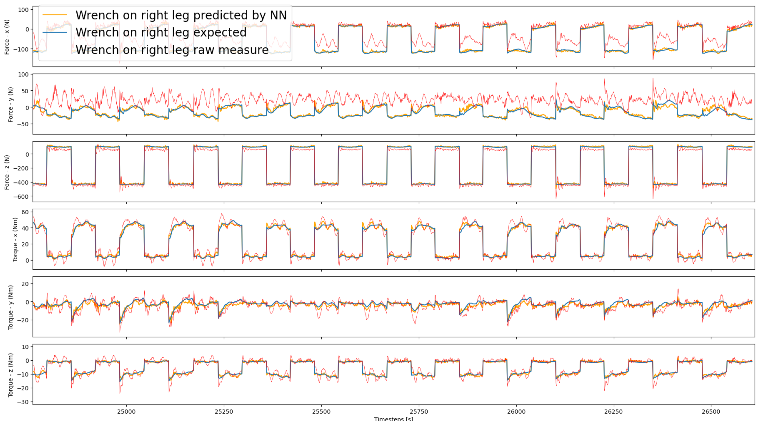
5.13 is reported the wrench predicted by the Neural Network in the test phase. In such plots, the prediction is compared with the raw wrench (i.e. the output of the actual calibration on the robot) and the expected wrench (i.e. the label). In figure 5.14 is reported the discrepancy between the NN prediction and the label, and the discrepancy between the raw wrench and the label. In tables 5.10, 5.11 and 5.12



(a) Zoom on part of dataset related to pole experiment



(b) Zoom on part of dataset related to balancing experiment



(c) Zoom on part of dataset related to walking experiment

Figure 5.10: Comparison of wrenches predicted by NN with expected wrenches and raw values for left leg FT test phase (13 inputs)

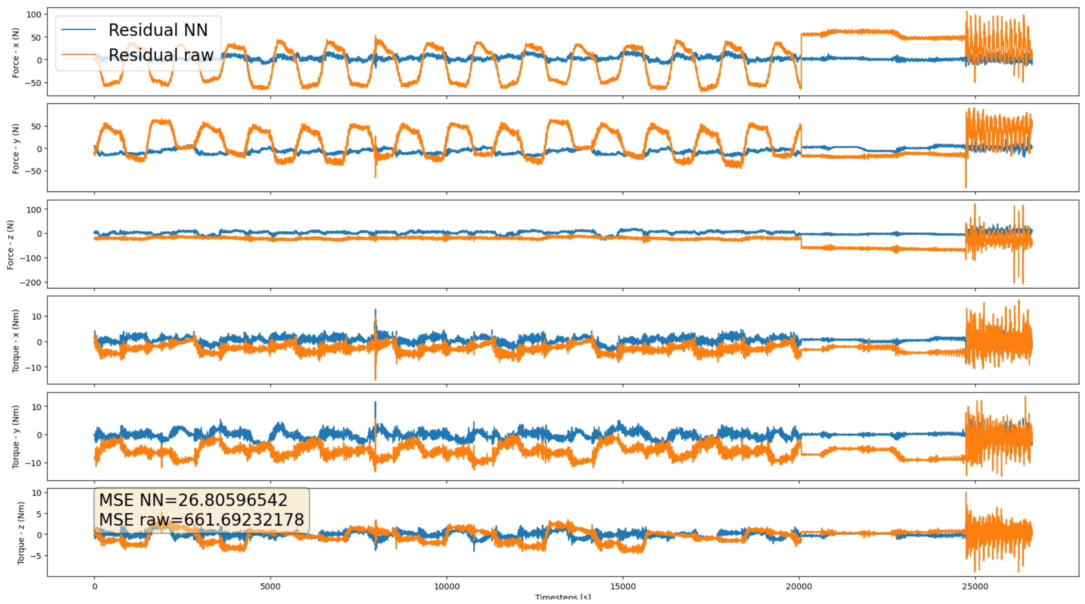


Figure 5.11: Residual obtained in test phase for left leg FT (13 inputs)

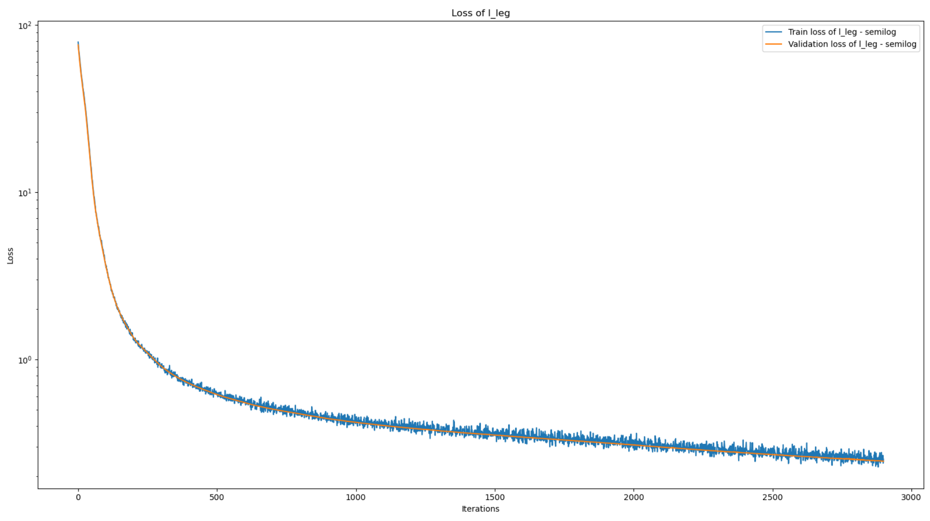


Figure 5.12: Loss function during training of NN for left leg FT

MAE	**NN**	**raw**
MAE global	3.28	17.37
MAE force	5.66	31.32
MAE torque	0.91	3.41
MAE fx	4.07	37.72
MAE fy	7.42	27.46
MAE fz	5.49	28.79
MAE tx	1.12	3.02
MAE ty	1.09	5.95
MAE tz	0.51	1.25

Table 5.8: Left leg FT: Mean Absolute Error (MAE) for Neural Network (NN) and raw data (13 inputs)

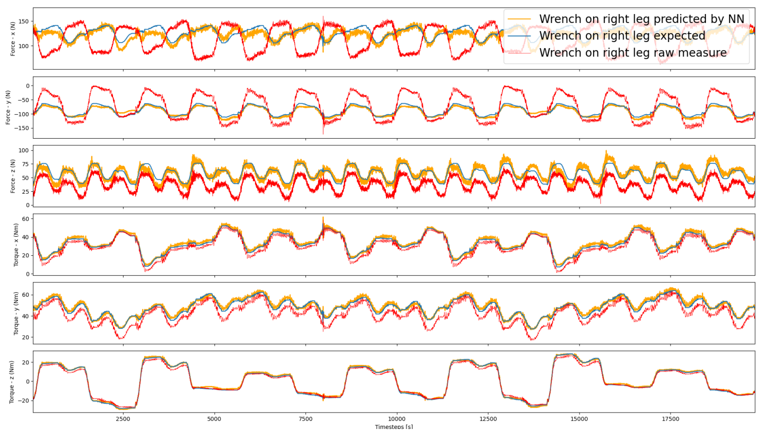
RMSE	**NN**	**raw**
RMSE global	5.18	25.72
RMSE force	7.21	36.12
RMSE torque	1.25	4.30
RMSE fx	5.41	42.06
RMSE fy	8.91	31.85
RMSE fz	6.90	33.64
RMSE tx	1.43	3.44
RMSE ty	1.47	6.40
RMSE tz	0.68	1.60

Table 5.9: Left leg FT: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data (13 inputs)

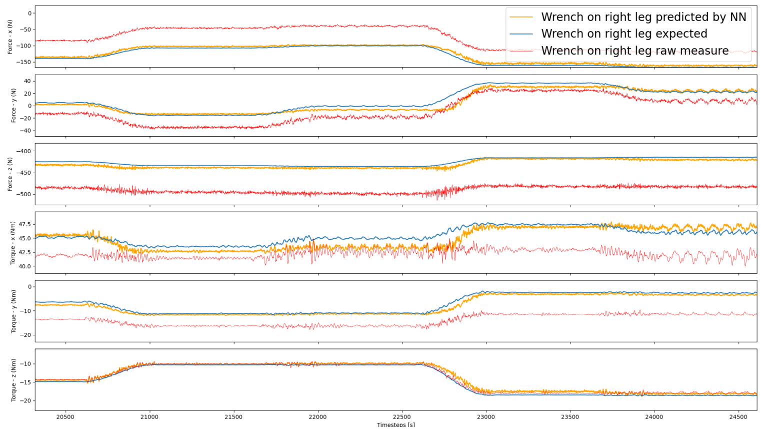
are reported respectively the MSE, MAE and RMSE committed by the NN and the actual calibration (raw value) with respect to the labels.

5.6.4 Comparison with the results obtained by using polynomial models

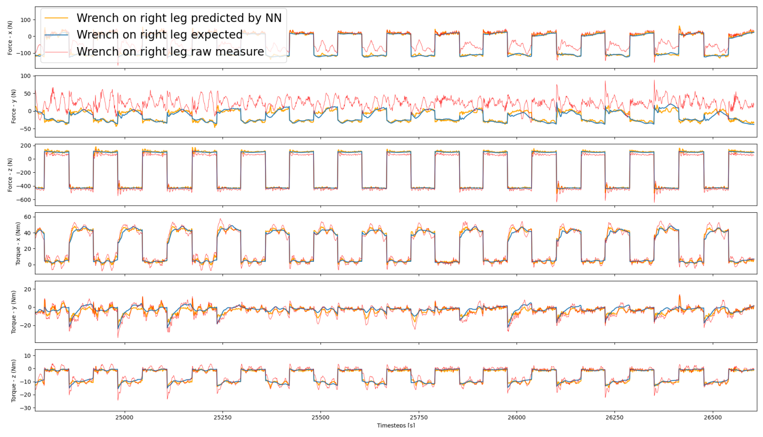
Below the plot of the wrenches during the test phase, with the RMSE between predicted and expected values for each component of the wrench.



(a) Zoom on part of dataset related to pole experiment



(b) Zoom on part of dataset related to balancing experiment



(c) Zoom on part of dataset related to walking experiment

Figure 5.13: Comparison of wrenches predicted by NN (with 7 inputs) with expected wrenches and raw values for left leg FT test phase

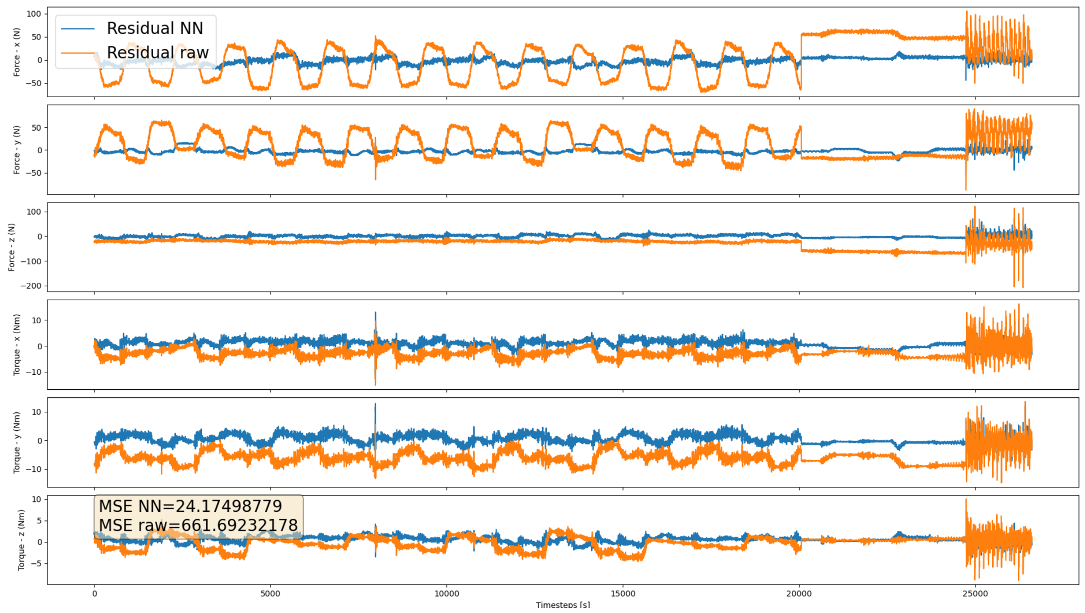


Figure 5.14: Residual obtained in test phase for left leg FT (7 inputs)

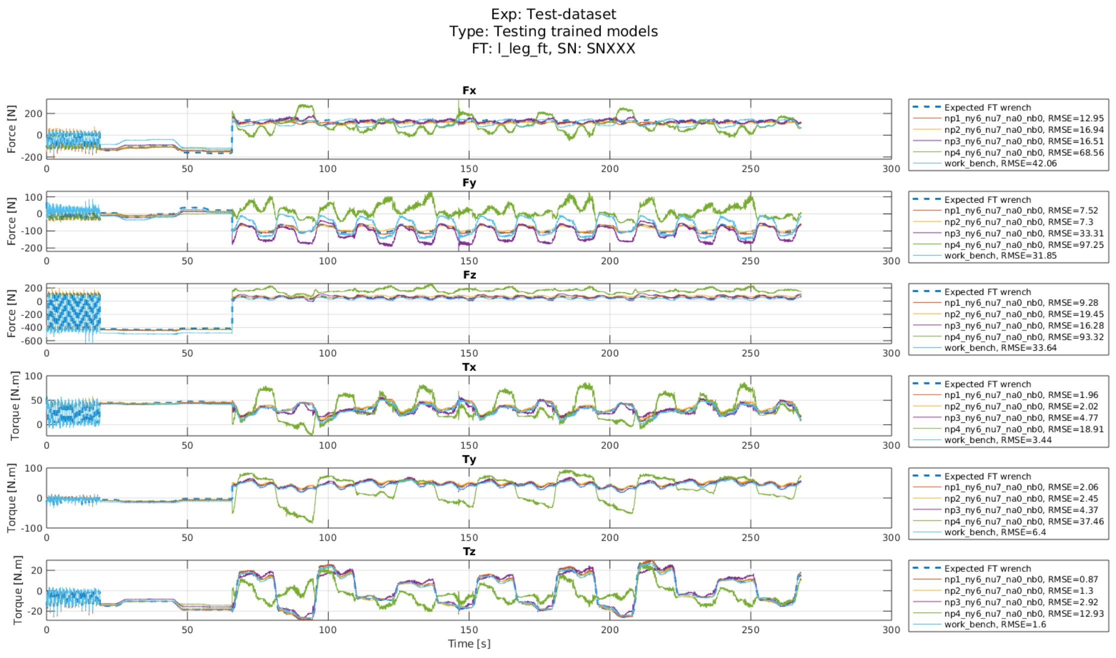


Figure 5.15: Test phase for left leg FT with polynomial models

MSE	**NN**	**raw**
MSE global	24.17	661.69
MSE force	46.09	1304.93
MSE torque	2.26	18.45
MSE fx	58.14	1768.69
MSE fy	36.50	1014.31
MSE fz	43.64	1131.80
MSE tx	2.93	11.86
MSE ty	2.94	40.96
MSE tz	0.91	2.55

Table 5.10: Left leg FT 7 inputs: Mean Squared Error (MSE) for Neural Network (NN) and raw data

MAE	**NN**	**raw**
MAE global	3.28	17.37
MAE force	5.39	31.32
MAE torque	1.17	3.41
MAE fx	6.16	37.72
MAE fy	4.89	27.46
MAE fz	5.11	28.79
MAE tx	1.39	3.02
MAE ty	1.34	5.95
MAE tz	0.79	1.25

Table 5.11: Left leg FT 7 inputs: Mean Absolute Error (MAE) for Neural Network (NN) and raw data

Comparing the RMSEs in Figure 5.15 with those obtained for NN models (specifically, the model with 13 inputs as shown in Table 5.9 and the model with 7 inputs as shown in Table 5.12), it becomes evident that the neural networks generally perform better the polynomial model across most components of the wrench. The exceptions to this trend are observed in f_y, where the polynomial models $np1$ and $np2$ perform better the NN model with 13 inputs, and in the component τ_z, where the polynomial model $np1$ performs better the NN model with 7 inputs.

RMSE	NN	raw
RMSE global	4.92	25.72
RMSE force	6.79	36.12
RMSE torque	1.50	4.30
RMSE fx	7.62	42.06
RMSE fy	6.04	31.85
RMSE fz	6.61	33.64
RMSE tx	1.71	3.44
RMSE ty	1.71	6.40
RMSE tz	0.96	1.60

Table 5.12: Left leg FT 7 inputs: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data

5.7 Models for FTs on right foot

In this section, we will first present the architecture of the NN models (with both 13 and 7 inputs) for the FT on the right foot. Subsequently, we will present the obtained results and compare them with the current calibration on the robot.

5.7.1 Architecture of the models

The architecture is the same both for the `r_foot_front_ft` and `r_foot_rear_ft`:

- **Batch size:** 1024
- **Number of epochs:** 8
- **Normalization:** batch standard normalization
- **Number of layers:** 2 (plus the output layer)
- **Number of neurons per layer:** 100
- **Dropout probabilities:** [0.0, 0.0]
- **Bias in the linear transformations in each layer:** true
- **Activation functions:** [LeakyReLU, ReLU]

- **Optimizer:** Adam (weight_decay=0)
- **Learning rate:** 0.005
- **Learning rate scheduler:** ExponentialLR, $\gamma = 0.99$, applied only when training loss reaches the threshold 0.005 (this threshold is not reached during this training)
- **Initialization of the weights of the neural network:** `xavier_uniform`
- $\lambda_{\text{ExpectedRightFootFront}} = 1.0$
- $\lambda_{\text{ExpectedRightFootRear}} = 1.0$
- $\lambda_{\text{ExpectedRightFootSum}} = 1.0$

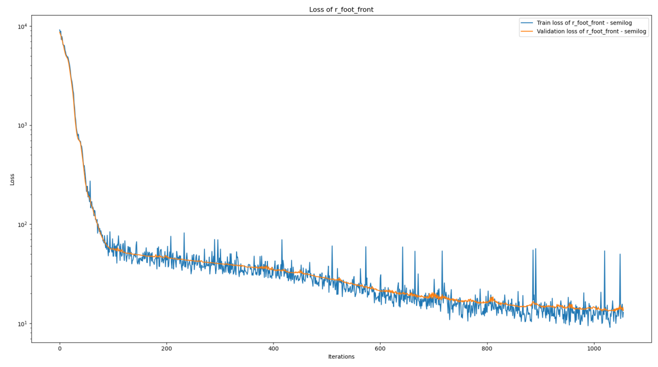
5.7.2 Results for model with 13 inputs

In figure 5.16 the loss functions during training are reported. In figure 5.17 are reported the wrenches predicted by the Neural Networks in the test phase, but only in the part of dataset related to pole experiment, in which we have available labels. Then, in figure 5.18 is reported the sum of the wrenches predicted by the Neural Network, both expressed with respect to `r_sole` frame. In such plots, the prediction is compared with the raw sum of wrenches (i.e. the output of the actual calibration on the robot) and the sum of expected wrenches (i.e. the label). In figure 5.19 is reported the discrepancy between the sum of wrenches predicted by NN and the label, and the discrepancy between the sum of raw wrenches and the label. In tables 5.13, 5.14 and 5.15 are reported respectively the MSE, MAE and RMSE committed by the NN and the actual calibration (raw value) with respect to the labels (considering the sum of the wrenches, not the single wrenches).

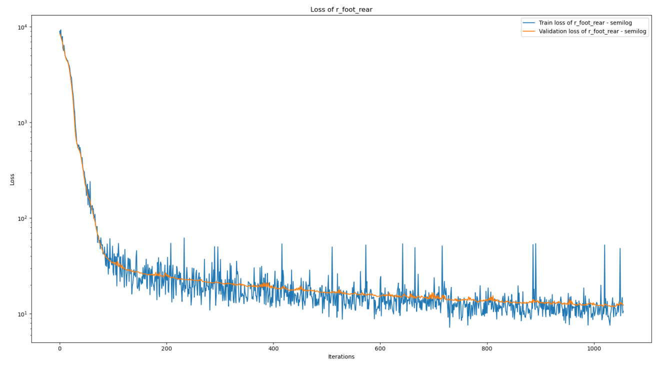
5.7.3 Results for model with 7 inputs

The architecture is the same of the model with 13 inputs presented in section 5.8.1. For that reason, we will present directly the results obtained.

In figure 5.20 the loss functions during training are reported. In figure 5.21 are reported the wrenches predicted by the Neural Networks in the test phase, but only in the part of dataset related to pole experiment, in which we have available labels. Then, in figure 5.22 is reported the sum of the wrenches predicted by the NN, both expressed with respect to `r_sole` frame. In such plots, the prediction is compared

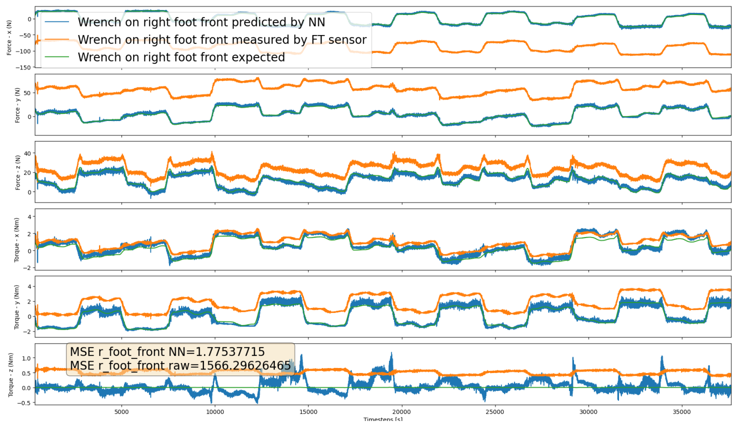


(a) Loss function during training of NN for right foot front FT

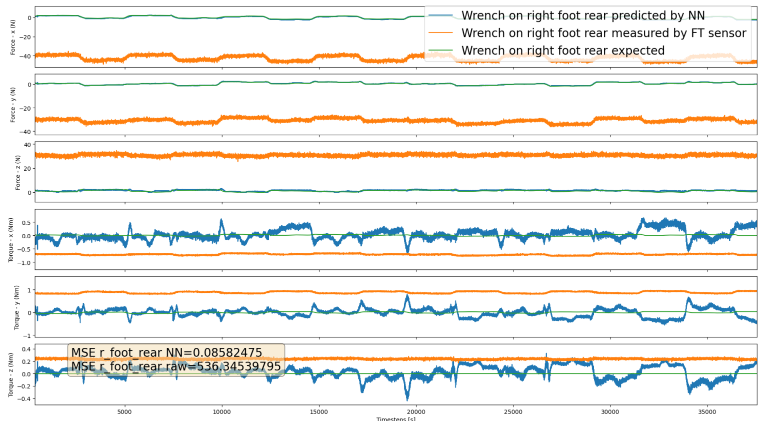


(b) Loss function during training of NN for right foot rear FT

Figure 5.16: Train and validation loss functions during training of NN for right foot FTs (13 inputs)

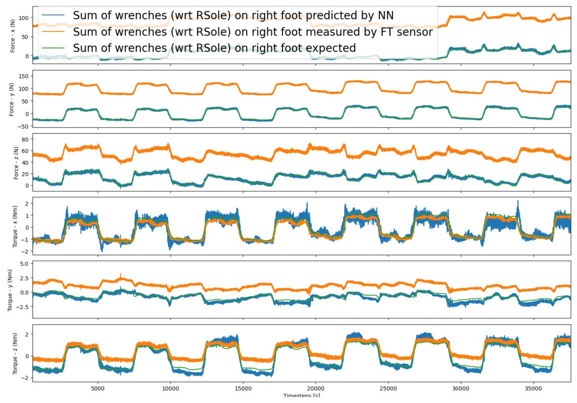


(a) Wrench on right foot front FT

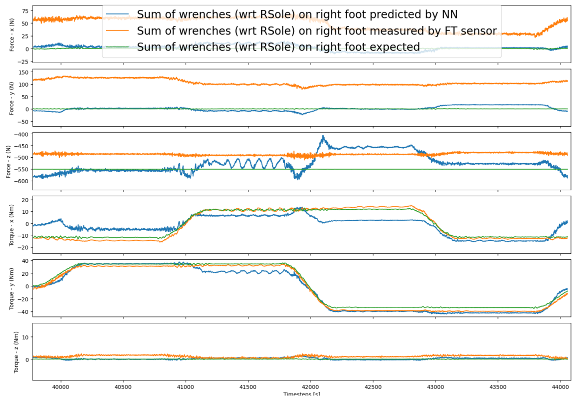


(b) Wrench on right foot rear FT

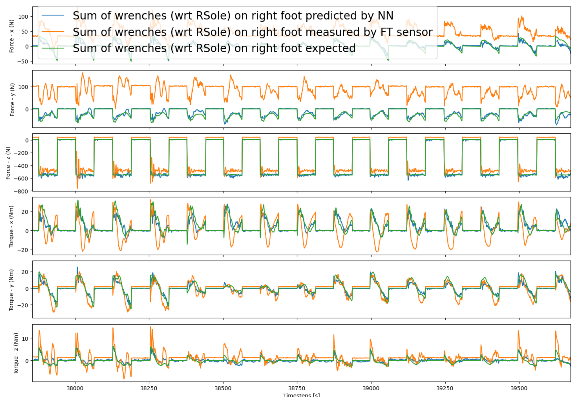
Figure 5.17: Wrenches on right foot on part of dataset related to pole experiment (13 inputs)



(a) Zoom on part of dataset related to pole experiment



(b) Zoom on part of dataset related to balancing experiment



(c) Zoom on part of dataset related to walking experiment

Figure 5.18: Comparison of the sum of wrenches predicted by the neural network (NN) with the expected and raw values during the right foot FTs test phase (13 inputs)

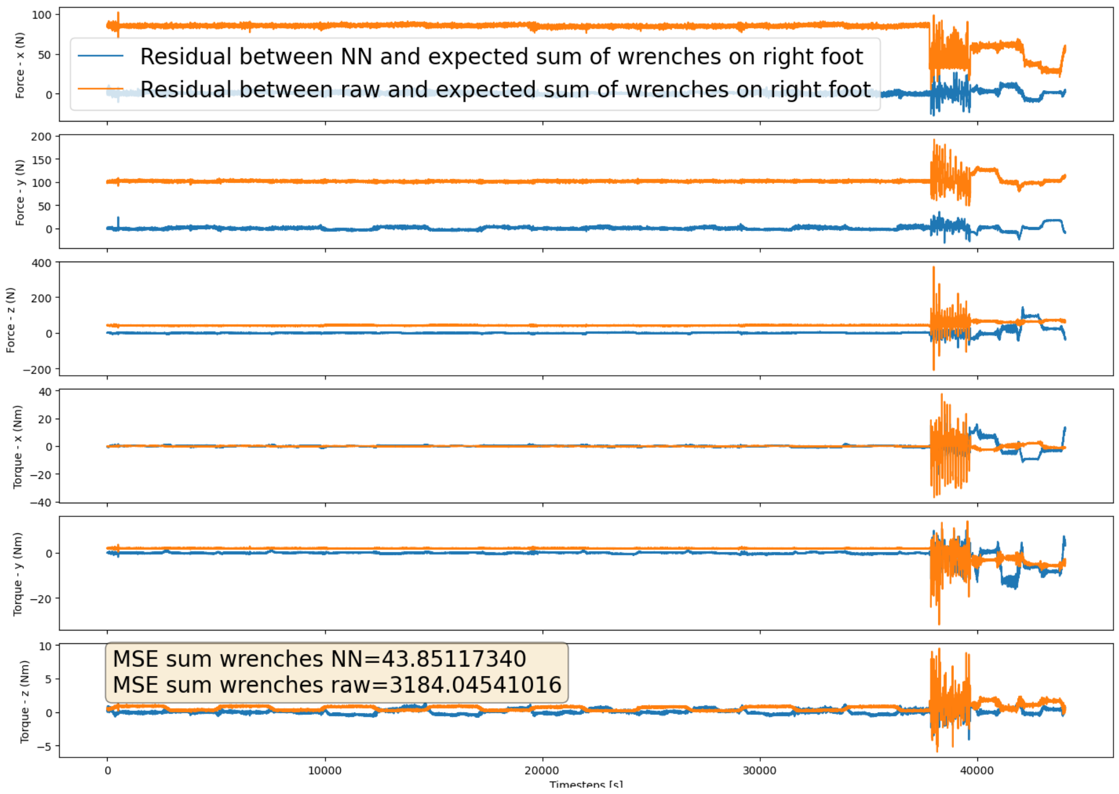
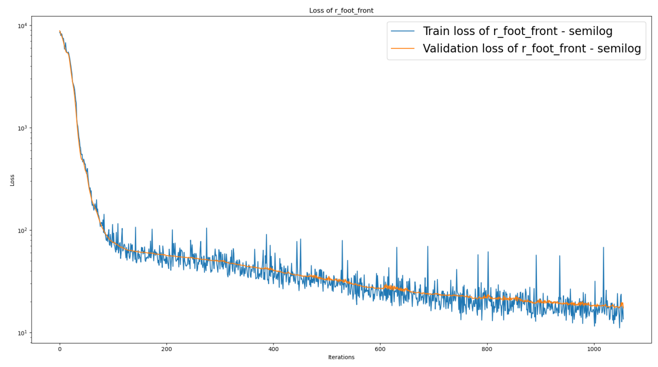
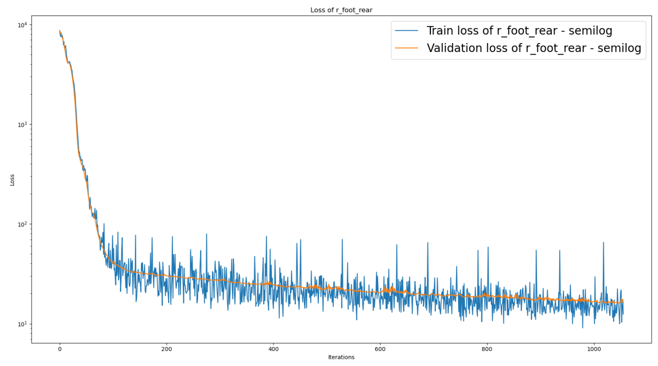


Figure 5.19: Residual obtained in test phase for right foot FT (13 inputs)



(a) Loss function during training of NN for right foot front FT



(b) Loss function during training of NN for right foot rear FT

Figure 5.20: Train and validation loss functions during training of NN for right foot FTs

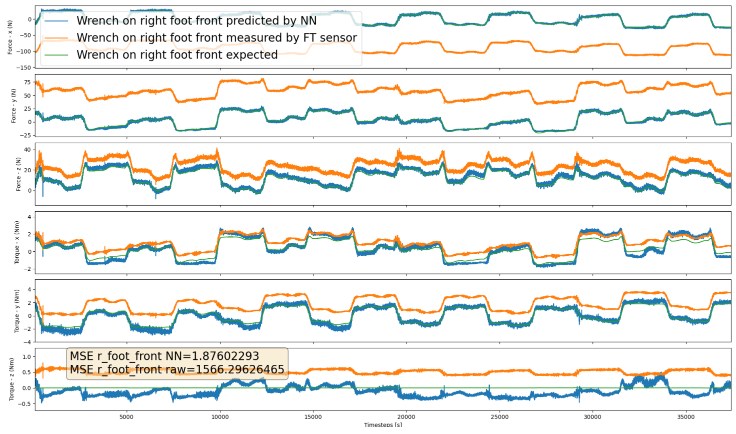
MSE	**NN**	**raw**
MSE global	43.85	3184.05
MSE force	83.91	6364.29
MSE torque	3.79	3.80
MSE fx	7.37	6567.89
MSE fy	16.08	10473.16
MSE fz	228.27	2051.82
MSE tx	5.28	4.91
MSE ty	5.97	5.81
MSE tz	0.13	0.69

Table 5.13: Sum of wrenches on right foot: Mean Squared Error (MSE) for Neural Network (NN) and raw data (13 inputs)

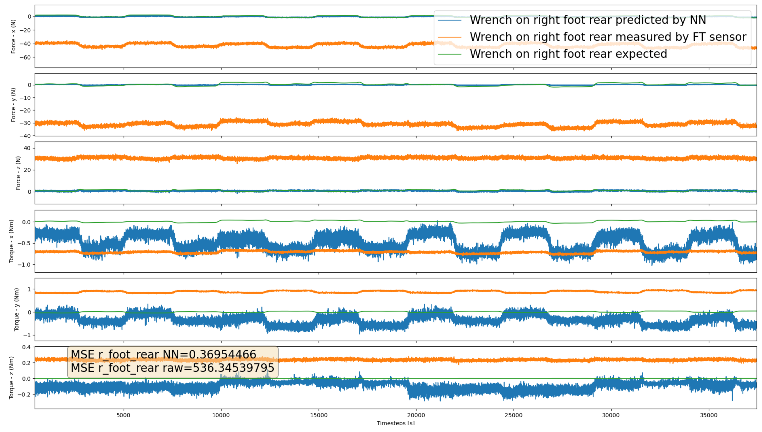
MAE	**NN**	**raw**
MAE global	1.93	38.27
MAE force	3.15	75.44
MAE torque	0.70	1.11
MAE fx	1.76	79.77
MAE fy	2.66	102.17
MAE fz	5.02	44.37
MAE tx	0.92	0.52
MAE ty	0.92	2.13
MAE tz	0.28	0.66

Table 5.14: Sum of wrenches on right foot: Mean Absolute Error (MAE) for Neural Network (NN) and raw data (13 inputs)

with the raw sum of wrenches (i.e. the output of the actual calibration on the robot) and the sum of expected wrenches (i.e. the label). In figure 5.23 is reported the discrepancy between the sum of wrenches predicted by NN and the label, and the discrepancy between the sum of raw wrenches and the label. In tables 5.16, 5.17 and 5.18 are reported respectively the MSE, MAE and RMSE committed by the NN and the actual calibration (raw value) with respect to the labels (considering the sum of the wrenches, not the single wrenches).

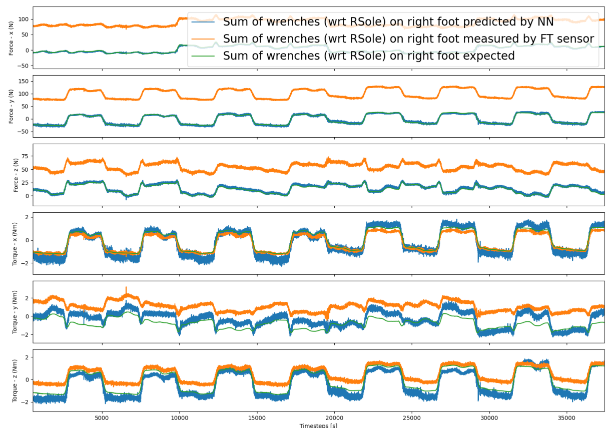


(a) Wrench on right foot front FT

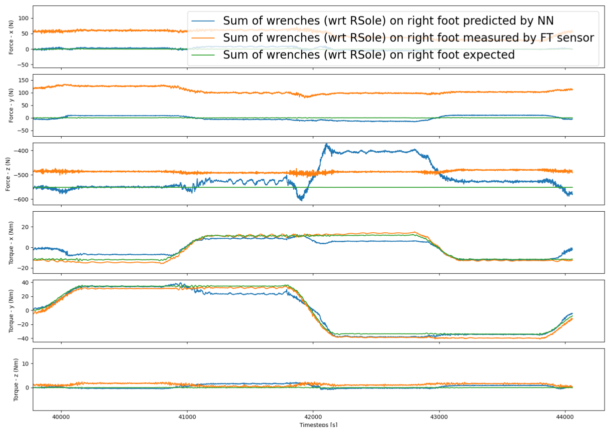


(b) Wrench on right foot rear FT

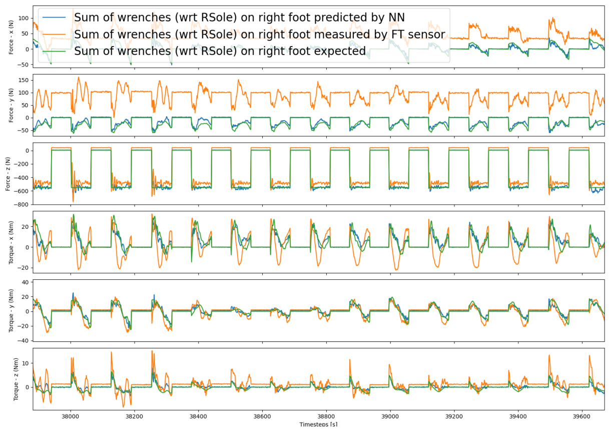
Figure 5.21: Wrenches on right foot on part of dataset related to pole experiment (7 inputs)



(a) Zoom on part of dataset related to pole experiment



(b) Zoom on part of dataset related to balancing experiment



(c) Zoom on part of dataset related to walking experiment

Figure 5.22: Comparison of the sum of wrenches predicted by the neural network (NN) with the expected and raw values during the right foot FTs test phase (7 inputs)

RMSE	**NN**	**raw**
RMSE global	6.62	56.43
RMSE force	9.16	79.78
RMSE torque	1.95	1.95
RMSE fx	2.72	81.04
RMSE fy	4.01	102.34
RMSE fz	15.11	45.30
RMSE tx	2.30	2.22
RMSE ty	2.44	2.41
RMSE tz	0.36	0.83

Table 5.15: Sum of wrenches on right foot: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data (13 inputs)

MSE	**NN**	**raw**
MSE global	83.84	3184.05
MSE force	165.42	6364.29
MSE torque	2.26	3.80
MSE fx	5.60	6567.89
MSE fy	15.30	10473.16
MSE fz	475.36	2051.82
MSE tx	2.95	4.91
MSE ty	3.61	5.81
MSE tz	0.22	0.69

Table 5.16: Right foot FTs 7 inputs: Mean Squared Error (MSE) for Neural Network (NN) and raw data

5.8 Models for FTs on left foot

In this section, we will first present the architecture of the NN models (with both 13 and 7 inputs) for the FT on the left foot. Subsequently, we will present the obtained results and compare them with the current calibration on the robot.

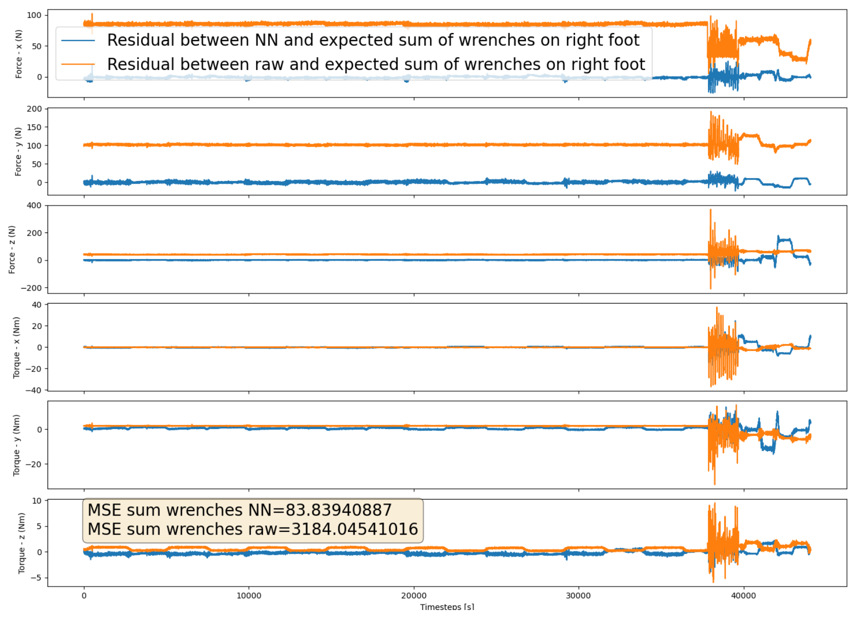


Figure 5.23: Residual obtained in test phase for right foot FTs (7 inputs)

MAE	**NN**	**raw**
MAE global	2.02	38.27
MAE force	3.37	75.44
MAE torque	0.67	1.11
MAE fx	1.59	79.77
MAE fy	2.69	102.17
MAE fz	5.84	44.37
MAE tx	0.69	0.52
MAE ty	0.95	2.13
MAE tz	0.37	0.66

Table 5.17: Right foot FTs 7 inputs: Mean Absolute Error (MAE) for Neural Network (NN) and raw data

5.8.1 Architecture of the models

The architecture is the same both for the `l_foot_front_ft` and `l_foot_rear_ft`:

- **Batch size:** 1024

RMSE	**NN**	**raw**
RMSE global	9.16	56.43
RMSE force	12.86	79.78
RMSE torque	1.50	1.95
RMSE fx	2.37	81.04
RMSE fy	3.91	102.34
RMSE fz	21.80	45.30
RMSE tx	1.72	2.22
RMSE ty	1.90	2.41
RMSE tz	0.47	0.83

Table 5.18: Right foot FTs 7 inputs: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data

- **Number of epochs:** 8
- **Normalization:** batch standard normalization
- **Number of layers:** 2 (plus the output layer)
- **Number of neurons per layer:** 100
- **Dropout probabilities:** [0.0, 0.0]
- **Bias in the linear transformations in each layer:** true
- **Activation functions:** [LeakyReLU, ReLU]
- **Optimizer:** Adam (weight_decay=0)
- **Learning rate:** 0.005
- **Learning rate scheduler:** ExponentialLR, $\gamma = 0.99$, applied only when training loss reaches the threshold 0.005 (this threshold is not reached during this training)
- **Initialization of the weights of the neural network:** `xavier_uniform`
- $\lambda_{\text{ExpectedLeftFootFront}} = 100.0$
- $\lambda_{\text{ExpectedLeftFootRear}} = 10000.0$
- $\lambda_{\text{ExpectedLeftFootSum}} = 1.0$

5.8.2 Results for model with 13 inputs

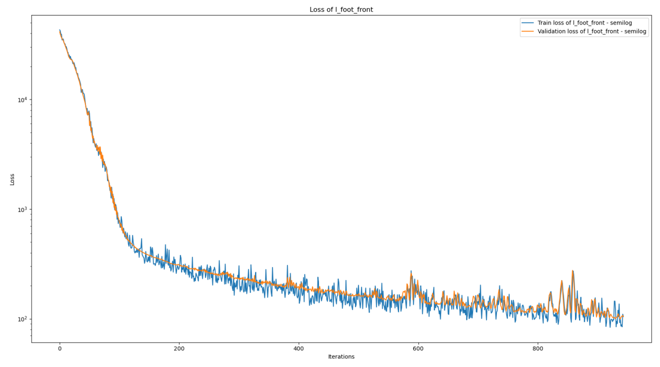
In figure 5.24 the loss functions during training are reported. In figure 5.25 are reported the wrenches predicted by the Neural Networks in the test phase, but only in the part of dataset related to pole experiment, in which we have available labels. Then, in figure 5.26 is reported the sum of the wrenches predicted by the NN, both expressed with respect to `l_sole` frame. In such plots, the prediction is compared with the raw sum of wrenches (i.e. the output of the actual calibration on the robot) and the sum of expected wrenches (i.e. the label). In figure 5.27 is reported the discrepancy between the sum of wrenches predicted by NN and the label, and the discrepancy between the sum of raw wrenches and the label. In tables 5.19, 5.20 and 5.21 are reported respectively the MSE, MAE and RMSE committed by the NN and the actual calibration (raw value) with respect to the labels (considering the sum of the wrenches, not the single wrenches).

MSE	**NN**	**raw**
MSE global	14.96	5404.08
MSE force	26.83	10802.94
MSE torque	3.10	5.21
MSE fx	11.70	14521.71
MSE fy	23.40	17364.00
MSE fz	45.38	523.12
MSE tx	4.16	10.16
MSE ty	4.99	4.37
MSE tz	0.14	1.11

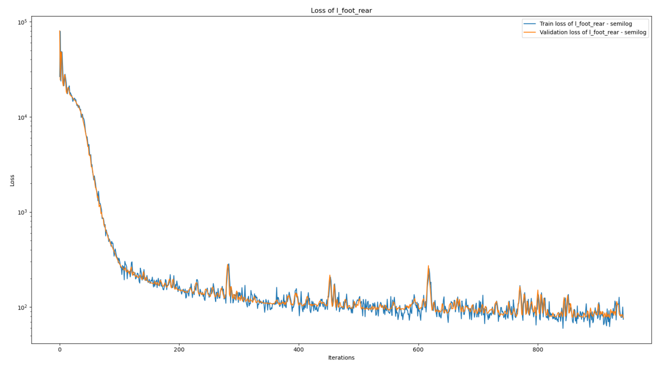
Table 5.19: Sum of wrenches on left foot: Mean Squared Error (MSE) for Neural Network (NN) and raw data (13 inputs)

5.8.3 Results for model with 7 inputs

The architecture is the same of the model with 13 inputs presented in section 5.8.1. For that reason, we will present directly the results obtained. In figure 5.28 the loss functions during training are reported. In figure 5.29 are reported the wrenches predicted by the Neural Networks in the test phase, but only in the part of dataset related to pole experiment, in which we have available labels. Then, in figure 5.30 is reported the sum of the wrenches predicted by the NN, both expressed with respect to

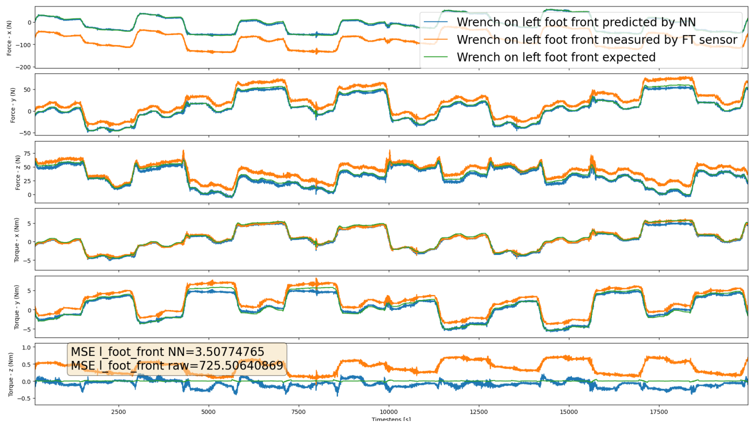


(a) Loss function during training of NN for left foot front FT

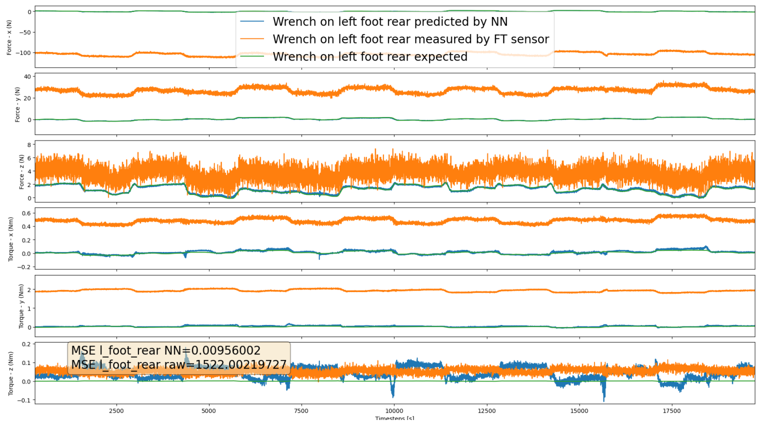


(b) Loss function during training of NN for left foot rear FT

Figure 5.24: Train and validation loss functions during training of NN for left foot FTs (13 inputs)

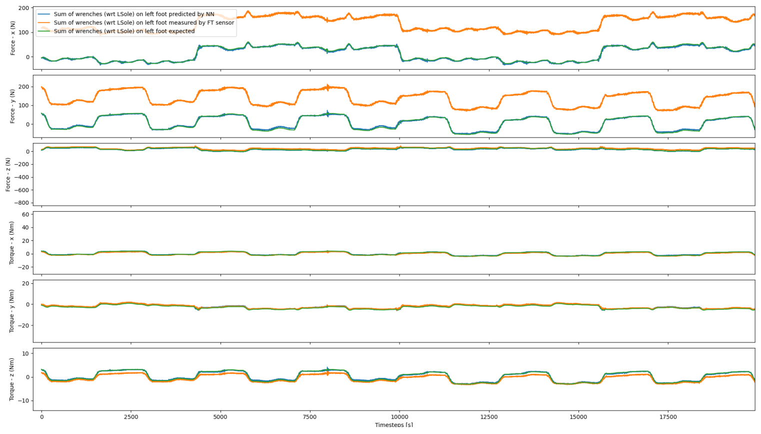


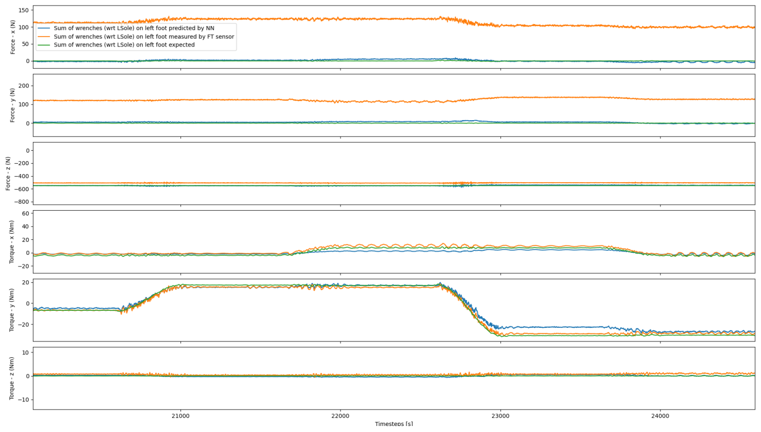
(a) Wrench on left foot front FT



(b) Wrench on left foot rear FT

Figure 5.25: Wrenches on left foot on part of dataset related to pole experiment (13 inputs)


(a) Zoom on part of dataset related to pole experiment


(b) Zoom on part of dataset related to balancing experiment

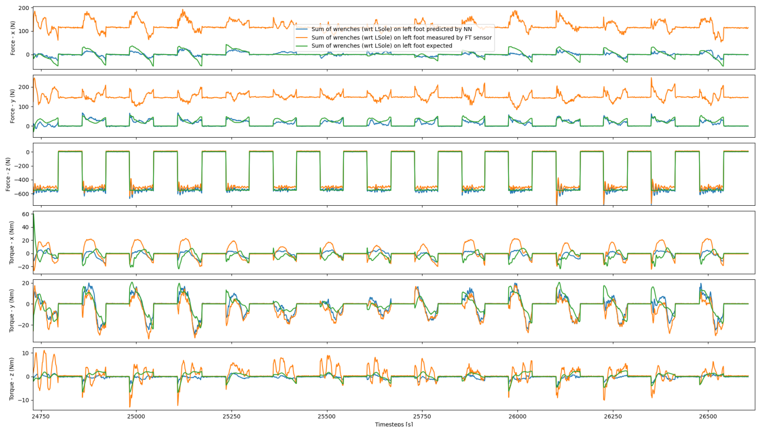

(c) Zoom on part of dataset related to walking experiment

Figure 5.26: Comparison of the sum of wrenches predicted by the neural network (NN) with the expected and raw values during the left foot FTs test phase (13 inputs)

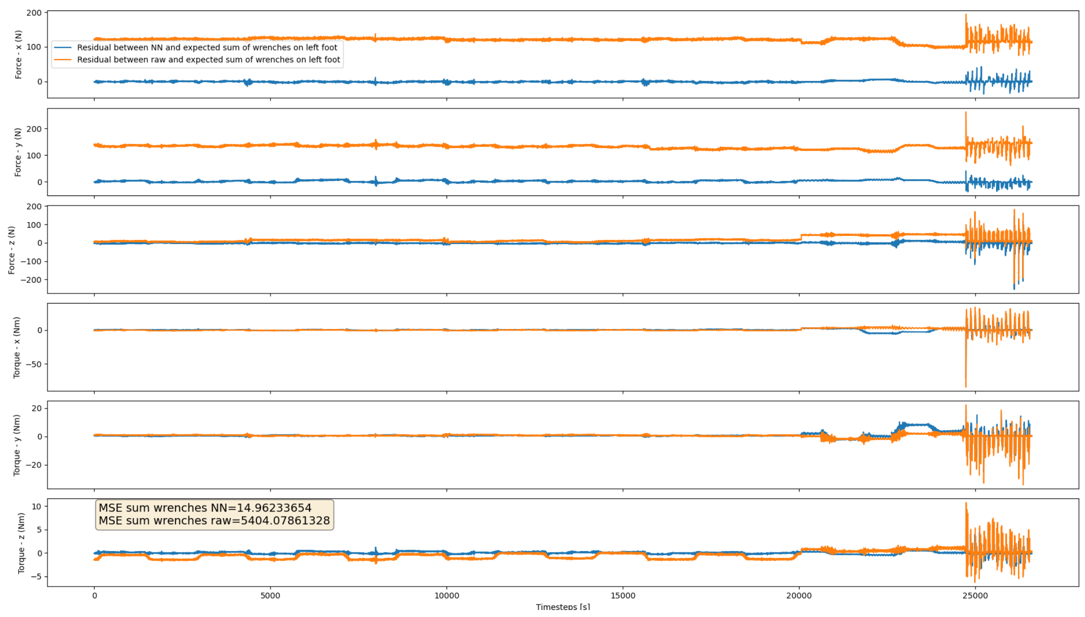
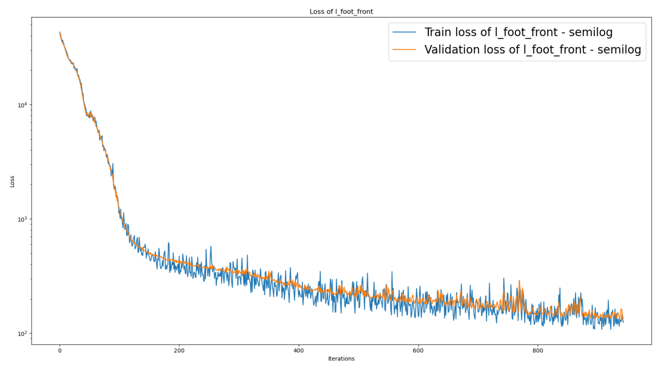


Figure 5.27: Residual obtained in test phase for left foot FTs (13 inputs)

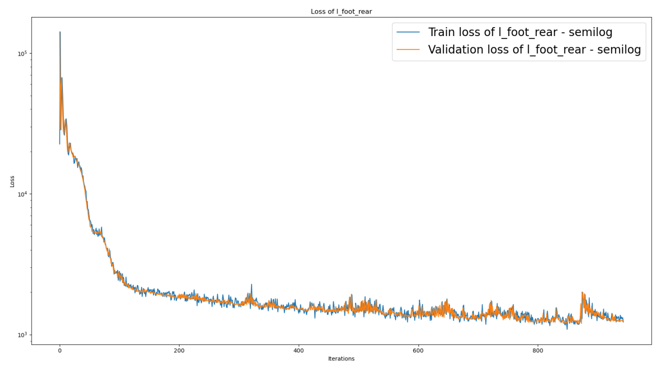
MAE	**NN**	**raw**
MAE global	1.93	45.47
MAE force	3.09	89.84
MAE torque	0.77	1.10
MAE fx	2.13	120.30
MAE fy	3.65	131.50
MAE fz	3.49	17.71
MAE tx	0.96	1.41
MAE ty	1.12	1.07
MAE tz	0.23	0.81

Table 5.20: Sum of wrenches on left foot: Mean Absolute Error (MAE) for Neural Network (NN) and raw data (13 inputs)

`l_sole` frame. In such plots, the prediction is compared with the raw sum of wrenches (i.e. the output of the actual calibration on the robot) and the sum of expected wrenches (i.e. the label). In figure 5.31 is reported the discrepancy between the sum of wrenches predicted by NN and the label, and the discrepancy between the sum of raw wrenches and the label. In tables 5.22, 5.23 and 5.24 are reported respectively



(a) Loss function during training of NN for left foot front FT



(b) Loss function during training of NN for left foot rear FT

Figure 5.28: Train and validation loss functions during training of NN for left foot FTs (7 inputs)

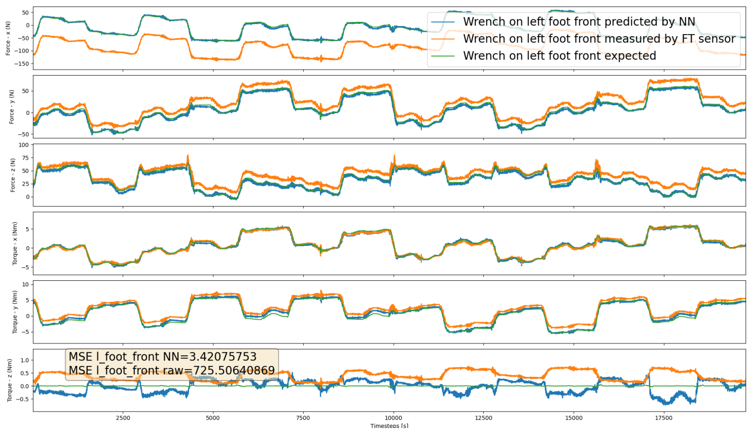
RMSE	**NN**	**raw**
RMSE global	3.87	73.51
RMSE force	5.18	103.94
RMSE torque	1.76	2.28
RMSE fx	3.42	120.51
RMSE fy	4.84	131.77
RMSE fz	6.74	22.87
RMSE tx	2.04	3.19
RMSE ty	2.23	2.09
RMSE tz	0.37	1.06

Table 5.21: Sum of wrenches on left foot: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data (13 inputs)

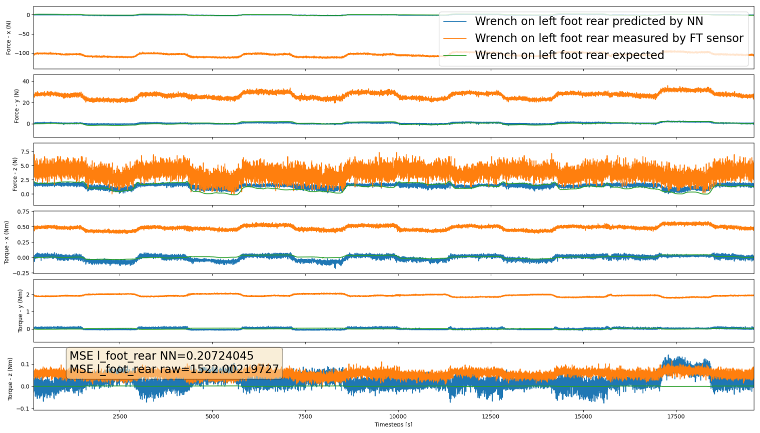
the MSE, MAE and RMSE committed by the NN and the actual calibration (raw value) with respect to the labels (considering the sum of the wrenches, not the single wrenches).

MSE	**NN**	**raw**
MSE global	15.81	5404.08
MSE force	28.18	10802.94
MSE torque	3.44	5.21
MSE fx	16.94	14521.71
MSE fy	24.96	17364.00
MSE fz	42.63	523.12
MSE tx	5.05	10.16
MSE ty	4.87	4.37
MSE tz	0.39	1.11

Table 5.22: Left foot FTs 7 inputs: Mean Squared Error (MSE) for Neural Network (NN) and raw data

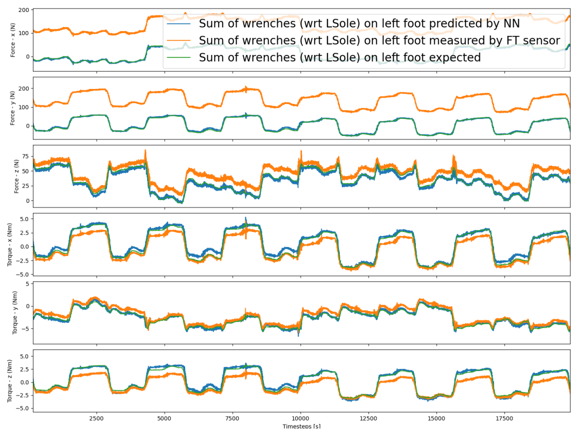


(a) Wrench on left foot front FT

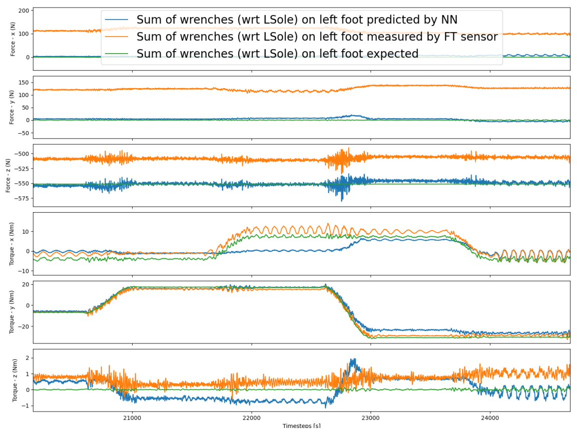


(b) Wrench on left foot rear FT

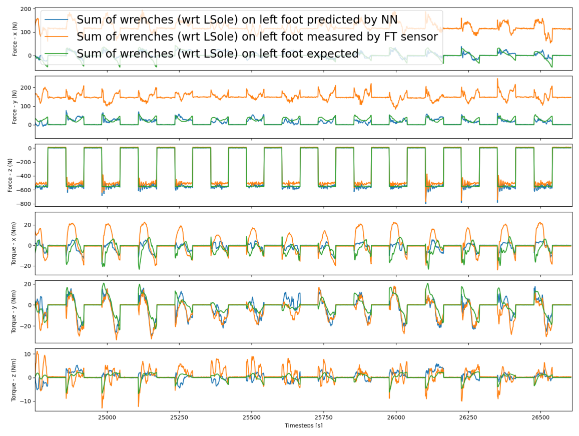
Figure 5.29: Wrenches on left foot on part of dataset related to pole experiment (7 inputs)



(a) Zoom on part of dataset related to pole experiment



(b) Zoom on part of dataset related to balancing experiment



(c) Zoom on part of dataset related to walking experiment

Figure 5.30: Comparison of the sum of wrenches predicted by the neural network (NN) with the expected and raw values during the left foot FTs test phase (7 inputs)

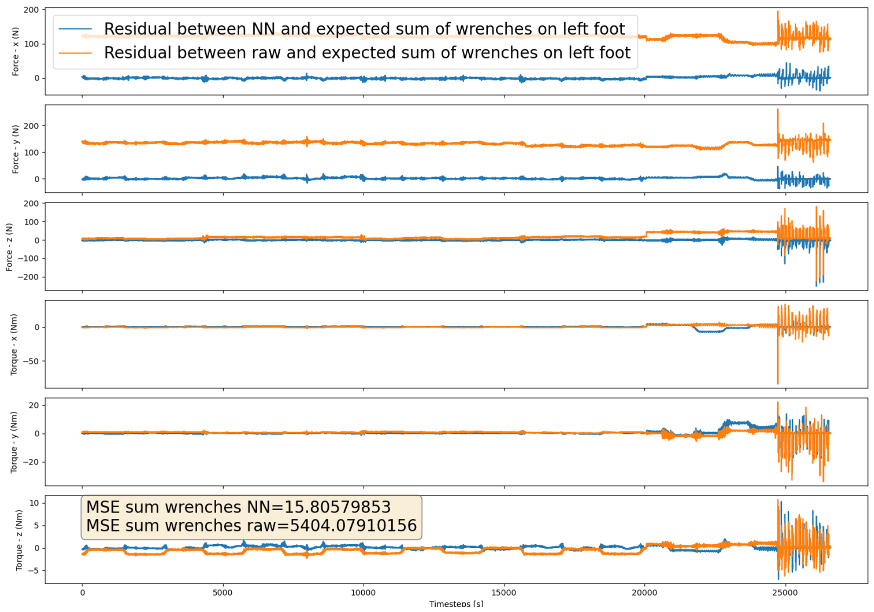


Figure 5.31: Residual obtained in test phase for left foot FTs (7 inputs)

MAE	**NN**	**raw**
MAE global	1.94	45.47
MAE force	3.10	89.84
MAE torque	0.78	1.10
MAE fx	2.84	120.30
MAE fy	3.51	131.50
MAE fz	2.97	17.71
MAE tx	1.07	1.41
MAE ty	0.91	1.07
MAE tz	0.38	0.81

Table 5.23: Left foot FTs 7 inputs: Mean Absolute Error (MAE) for Neural Network (NN) and raw data

5.9 Conclusions

In this chapter we presented the Neural Network models developed for the calibration of the force-torque sensors on upper legs and feet. Along with these models, we presented the results obtained in the testing phase, by using MSE, MAE and RMSE

RMSE	**NN**	**raw**
RMSE global	3.98	73.51
RMSE force	5.31	103.94
RMSE torque	1.85	2.28
RMSE fx	4.12	120.51
RMSE fy	5.00	131.77
RMSE fz	6.53	22.87
RMSE tx	2.25	3.19
RMSE ty	2.21	2.09
RMSE tz	0.62	1.06

Table 5.24: Left foot FTs 7 inputs: Root Mean Squared Error (RMSE) for Neural Network (NN) and raw data

as error metrics. These models (both the ones with 13 and 7 inputs), in the testing phase, manage to outperform the current calibration on each force-torque sensor taken into consideration. Furthermore, for the force-torque sensors on the legs we also carried out a comparison with the results obtained by using the polynomial models developed by [1], highlighting a superiority of the Neural Network models in almost every component of the wrench.

In the next chapter, the models presented in this chapter will be applied on the ErgoCub robot.

Chapter 6

Deployment and tests of the models on real robot

In the last chapter, Neural Network (NN) models for calibrating force-torque sensors on upper legs and feet have been developed and trained. Now, it's time to apply these models on the ErgoCub robot, to test its true effectiveness. In this chapter, we will focus on deploying the secondary calibration model for the force-torque (FT) sensor only on the right leg. More specifically, in section 6.1 the first test of the Neural Network model with 13 inputs for the FT of the right leg will be discussed, and its noisy output will be highlighted. For this reason, we will decide to train new models, which do not take linear acceleration and angular velocity of the sensor as input. Therefore, in section 6.2 we move on to the deployment of the models characterized by having only the raw value of the wrench and the temperature (i.e. 7 inputs) of the FT as inputs; specifically, a walking test will be conducted. At the end, in section 6.3 another test will be conducted to quantify the goodness of the secondary calibration via the Neural Network model with 7 inputs; in this test, it will be highlighted how the torque control on the robot's legs improves thanks to the NN model.
Therefore, this chapter will present three tests:

- Test A: The NN model the right leg sensor with 13 inputs is deployed on the robot. In this experiment, the robot remains stationary, and the sensor's output data is analyzed.
- Test B: The NN model the right leg sensor with 7 inputs is deployed on the robot. In this experiment, the robot is walking, and the sensor's output data is analyzed.

- Test C: The NN model the right leg sensor with 7 inputs is deployed on the robot. In this experiment, the robot is controlled using a PD (Proportional and Derivative) + gravity compensation torque controller. We will perform this test twice: once by controlling the right leg (specifically, the `r_hip_pitch` joint) and applying an external force to it, and once by controlling the left leg (`l_hip_pitch` joint) and applying an external force to it. This experiment serves as an indirect measurement of the sensor calibration quality, as the estimated torque on the controlled joint is closely linked to the force-torque sensor measurements. By observing in which of the two cases the robot better maintains torque and position tracking, we can determine which force-torque sensor provides more reliable measurements: the right leg with secondary NN calibration or the left leg without secondary calibration.

6.1 Test A

In this first experiment, the robot is standing still. The figure 6.1 shows the plot of the force f_z on the sensor FT on right leg (the one with secondary NN calibration), compared with f_z on left leg FT (the one without any secondary calibration); theoretically the values of f_z on the two sensors should be similar, as the robot is balanced on its legs. However, as we can clearly see from the image, the FT output of the right leg is much noisier than the left leg.

To understand the origin of the noise present in the output of the neural network, the input data of this model in this experiment have been analyzed. In particular, as we can see in images 6.2, the angular velocity and linear acceleration (e.g. along the x-axis) of the force-torque sensor on the right leg are very noisy.

For this reason, it was decided to try to train new neural network models that took as input only the raw wrench of the sensor (i.e. the output of the primary calibration) and its temperature, without considering angular velocity and linear acceleration. The results will be shown in the next sections.

6.2 Test B

As shown in the previous section, the application of the neural network model for the force torque sensor of the right leg did not give the desired results; in fact, the output of the Neural Network seems very noisy, especially when considered with the output of the sensor on the left leg (in which there is no secondary calibration). For this reason, we decided to deploy the model with 7 inputs (raw wrench and

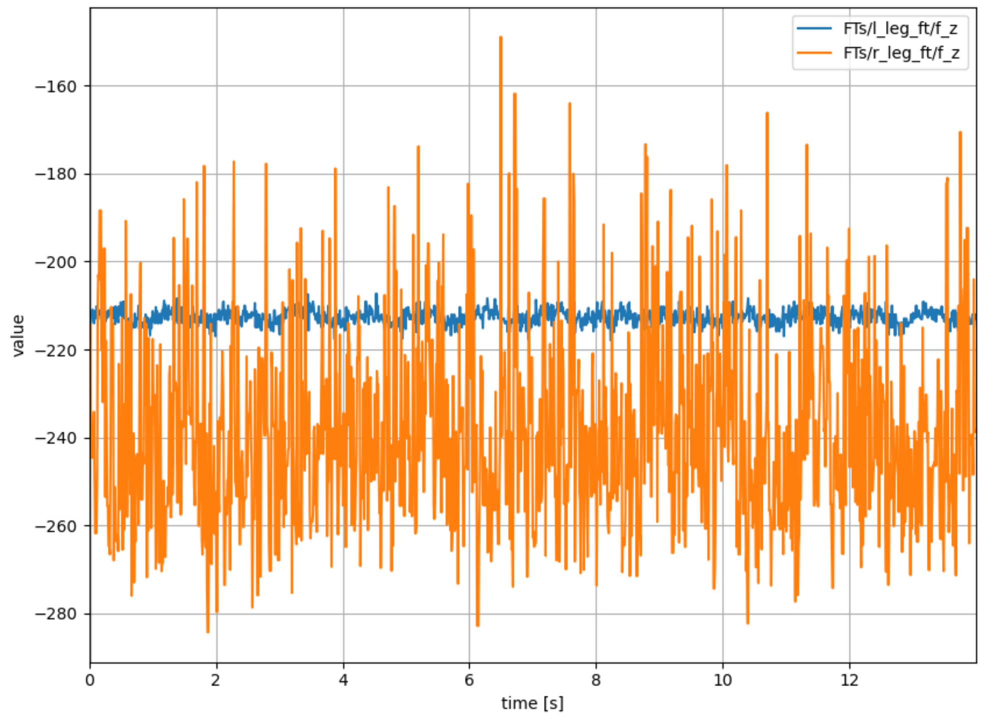
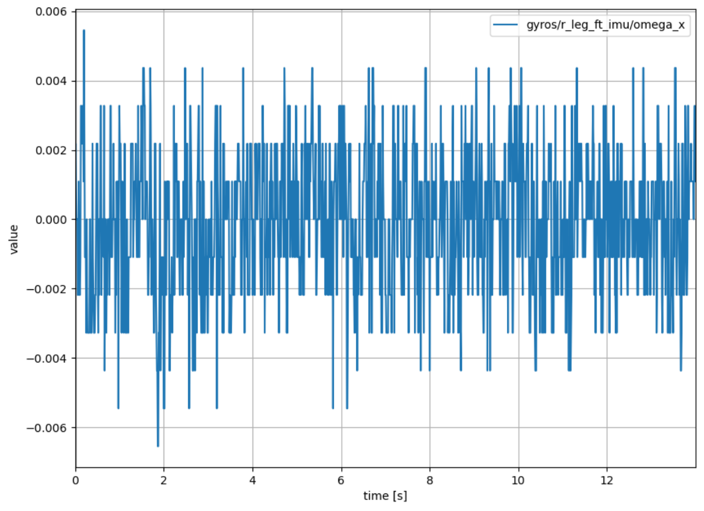
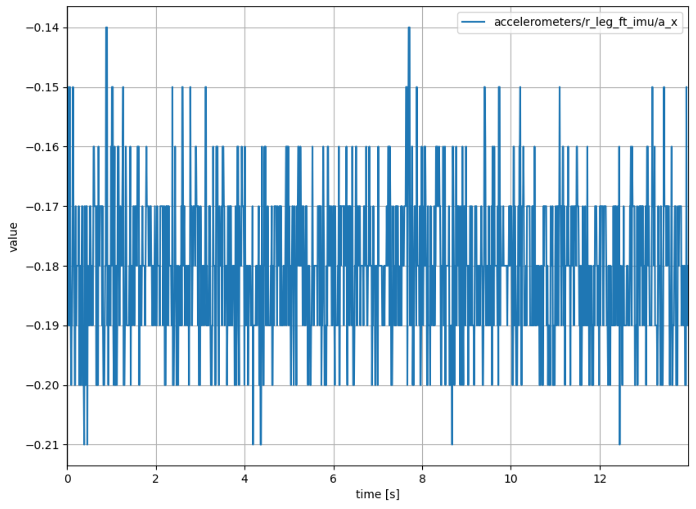


Figure 6.1: Comparison between f_z on left leg FT (without secondary calibration) and right leg FT (with secondary NN calibration)

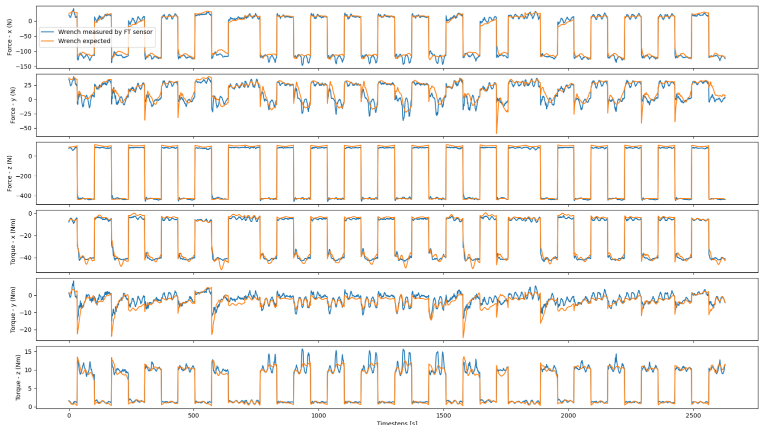


(a) Angular velocity of `r_leg_ft along x axis`

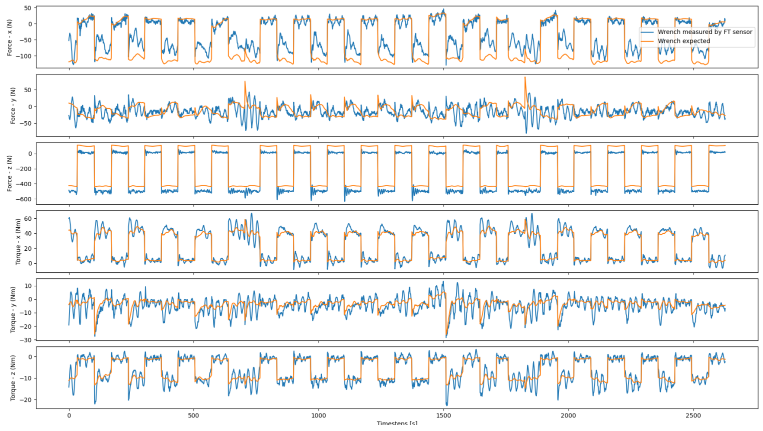


(b) Linear acceleration of `r_leg_ft along x axis`

Figure 6.2: Angular velocity and linear acceleration of `r_leg_ft along x axis`



(a) Wrench on right leg FT predicted by NN vs label



(b) Wrench on left leg FT predicted by NN vs labels

Figure 6.3: Wrench on legs FTs predicted by NN vs labels (walking experiment)

temperature). Subsequently, a test was conducted while the robot was walking. As in the previous section, there is secondary calibration only on the FT of the right leg. Once the dataset was collected, iDynTree has been used to calculate the expected wrenches (the labels) and compare them to the measurements obtained on the legs sensors (figure 6.3). Looking at the tables 6.1, 6.2 and 6.3, it is clear that the error committed by the FT on the right leg (with secondary NN calibration) is much smaller than the error made by the FT on the left leg (without secondary calibration).

In the next section we will perform another test to verify the effectiveness of the secondary NN calibration.

MSE	**Right leg**	**Left leg**
MSE global	65.76	1336.98
MSE fx	79.22	1172.43
MSE fy	69.82	404.28
MSE fz	233.93	6390.51
MSE tx	4.82	30.66
MSE ty	5.90	19.08
MSE tz	0.87	4.93

Table 6.1: Mean Squared Error (MSE) for right leg FT (with secondary NN calibration, 7 inputs) and left leg FT (without secondary calibration)

MAE	**Right leg**	**Left leg**
MAE global	4.93	21.32
MAE fx	6.60	25.95
MAE fy	5.83	14.75
MAE fz	12.98	78.18
MAE tx	1.77	4.04
MAE ty	1.82	3.37
MAE tz	0.60	1.64

Table 6.2: Mean Absolute Error (MAE) for right leg FT (with secondary NN calibration, 7 inputs) and left leg FT (without secondary calibration)

RMSE	**Right leg**	**Left leg**
RMSE global	8.11	36.56
RMSE fx	8.90	34.24
RMSE fy	8.36	20.11
RMSE fz	15.29	79.94
RMSE tx	2.19	5.54
RMSE ty	2.43	4.37
RMSE tz	0.93	2.22

Table 6.3: Root Mean Squared Error (RMSE) for right leg FT (with secondary NN calibration, 7 inputs) and left leg FT (without secondary calibration)

6.3 Test C

Another possible way to test the effectiveness of the secondary NN calibration on the right leg is to try torque control on the robot. Specifically, we could try to control the robot with a PD (Proportional and Derivative) + gravity compensation controller. This controller can be mathematically expressed as:

$$\tau^d = K_p(q^d - q) + K_d(\dot{q}^d - \dot{q}) + G(q), \tag{6.1}$$

where:

- τ^d is the desired torque;
- q is the vector collecting the position values of the joints, while q^d is the vector collecting the desired values;
- $\dot{q}$ is the vector collecting the velocity values of the joints, while $\dot{q}^d$ is the vector collecting the desired values;
- K_p and K_d are respectively the proportional and derivative gains;
- $G(q)$ is the gravity term of the dynamics equations of the system.

In our specific case, we would like to try this type of control twice:

- once by controlling only the `l_hip_pitch` joint, moving the left leg of the robot externally while the root link remains fixed on the pole;
- once by controlling only the `r_hip_pitch` joint, moving the right leg of the robot externally while the root link remains fixed on the pole.

Similarly to the previous tests, a secondary calibration NN (with 7 inputs) acts on the force-torque sensor of the right leg, while on the FT of the left leg there is no secondary calibration. The estimate of the torque acting on the joint `l_hip_pitch` largely depends on the measurements of `l_leg_ft`. This dependency is similar for `r_hip_pitch` and `r_leg_ft`. Therefore, the robot's performance in tracking the desired torque can serve as a reliable indicator of the measurement accuracy of the force-torque sensors. The purpose of this test is to determine which of the two cases yields better results in tracking the torque exerted on the controlled joint and its position. Consequently, it aims to evaluate how much the NN secondary calibration improves the measurement accuracy of the FT sensor on the right leg. In this kind of test, the robot has the root link fixed on the pole, and the controller is only working on `l_hip_pitch` (in the test on the left leg) or `r_hip_pitch` (in the test on the right leg). The controller's task is to maintain the torque acting on the joint and therefore its position. Once the controller is activated, an external force is applied to the left leg to assess the robot's capability of maintaining joint tracking.

In the case of the left leg test, the robot begins to vibrate after being subjected to an external force on the left leg and is unable to correctly track torque and position (see Figure 6.4).

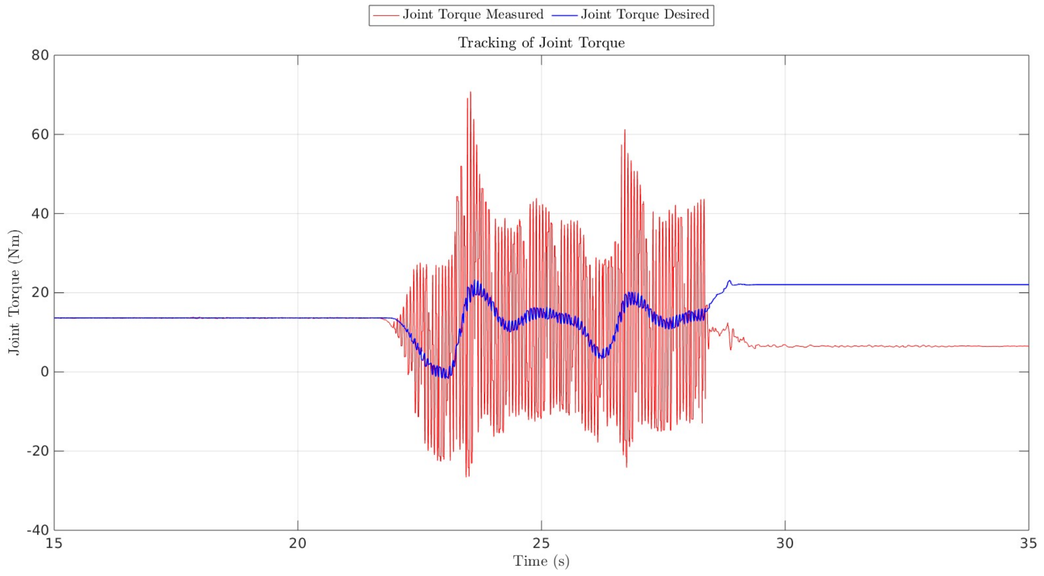
In contrast, when performing this test on the right leg (noting that the `r_leg_ft` sensor has secondary calibration with the NN model using 7 inputs), the robot is able to execute the task correctly. Figure 6.5 shows the corresponding tracking of torque and position for the `r_hip_pitch` joint. The maximum absolute error in torque tracking is $11.06Nm$, while the absolute error in position tracking (after the transient phase) is approximately 0.23 degrees.

6.4 Conclusions

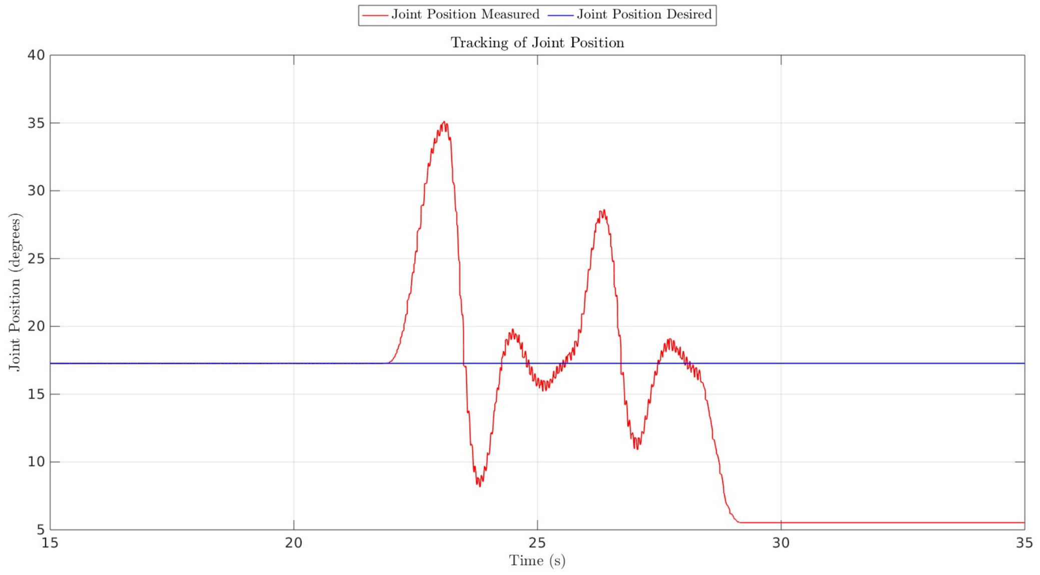
This chapter discussed the deployment of the models developed in Chapter 5 on the real robot and the subsequent tests conducted. We considered Neural Network models for calibrating the force-torque sensor on the right leg. In the first test (section 6.1), we deployed the NN model with 13 inputs; the output of this was very noisy. Such noise was probably due to noisy inputs, namely the linear acceleration and angular velocity of the sensor. Therefore, we opted to deploy the model for the right leg, which uses only the raw wrench (i.e., the output of the primary calibration) and the sensor temperature as inputs, for a total of 7 inputs. The tests conducted with these models turned out to be quite positive.

In section 6.2 the data obtained during a walking experiment were analyzed, and the ability of the NN model on the right leg to outperform the primary calibration of the left leg was highlighted (in which instead there was no secondary calibration).

Finally, in section 6.3, another type of test has been conducted, in which the robot was controlled via a PD (Proportional and Derivative) + gravity compensation torque control. Specifically, this control was tested twice: once in which only the `l_hip_pitch` joint was controlled, and another one in which only the `r_hip_pitch` joint was controlled . Since the torque measured on these joints depends significantly on the measurement of the force-torque sensor present in the related leg, this was a way to evaluate the accuracy of the secondary calibration of the sensor via Neural Network compared to the primary calibration. In fact, in the test in which the `l_hip_pitch` joint was controlled, the robot was not able to maintain torque and position tracking on the joint due to the poor precision of the measurement output from `l_leg_ft` (in which there was no secondary calibration) following external stresses. On the contrary, when the same control was carried out on the `r_hip_pitch` joint, the robot was able to respond adequately, due to the more accurate measurement of forces and moments in the sensor `r_leg_ft`.

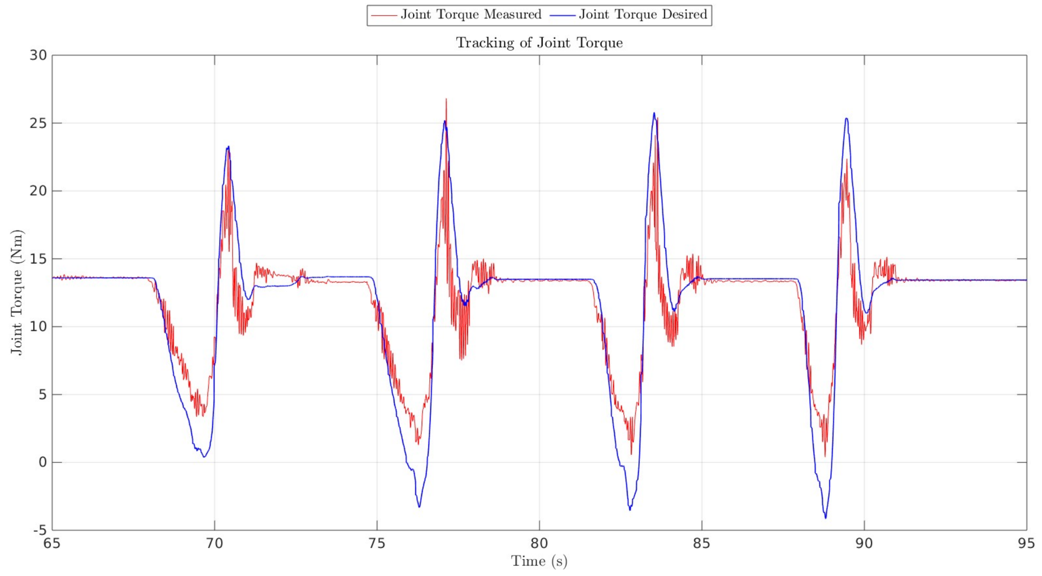


(a) Tracking of torque on `l_hip_pitch`

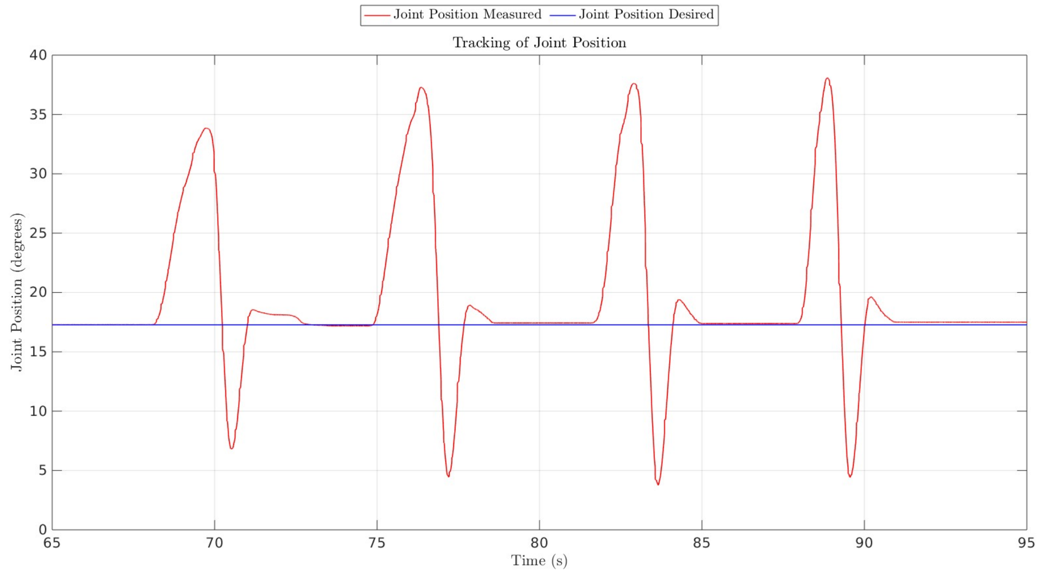


(b) Tracking of position on `l_hip_pitch`

Figure 6.4: Tracking of torque and position on `l_hip_pitch`



(a) Tracking of torque on `r_hip_pitch`



(b) Tracking of position on `r_hip_pitch`

Figure 6.5: Tracking of torque and position on `r_hip_pitch`

Chapter 7

Conclusions and future perspectives

This chapter concludes the present thesis work. The topic addressed was the calibration of $6D$ strain gauge force-torque (FT) sensors integrated into the ErgoCub robot developed by the Italian Institute of Technology. The calibration of such sensors involves establishing a relationship between the sensor output (in the case of these sensors, a potential difference expressed in bits) and the *wrench* (i.e., the $6D$ vector containing forces and torque) applied to the robot. The calibration of these sensors is crucial in the field of humanoid robotics. Accurate knowledge of the forces and moments acting on the robot is essential not only for understanding how it interacts with the surrounding environment but also for improving its controllability, for example, through control over the joint torques of the robot.

A classical method adopted for the calibration of such sensors is linear regression; this is also the model currently implemented in the FT sensors of the robot. However, the results of this linear model often prove unsatisfactory primarily because it assumes a simple linear relationship between the sensor output and the applied wrench; unfortunately, in nature, there are very few phenomena accurately described by a linear model. For this reason, literature has proposed polynomial models of generic order [1] or approaches based on Neural Networks [15], [16]. Furthermore, in the specific case of humanoid robotics, several other disturbance factors affect the measurement, notably the sensor's temperature. Indeed, these sensors mounted on the robot are located near electric motors that generate heat, which in turn affects the FT sensors. This temperature variation causes strain gauges within the sensor to deform inaccurately due to the applied wrench, thereby altering the measurement. For this reason, some authors have proposed models that also consider the sensor's temperature [14], [1]; furthermore, these authors suggest an *in-situ* calibration, utilizing data obtained from experiments where the sensor is already mounted on the robot. This in-situ procedure is necessary because if a force-torque sensor is calibrated using ex-situ data and then mounted on the robot, a decrease in performance has been observed, for instance due to the tightening force of screws [5].

Given the existing literature, in this study, we proposed an approach that seeks to combine the merits of the aforementioned methods. Specifically, in this thesis, our aim was to perform a secondary calibration, where the model we developed takes as input the raw wrench output from the primary linear calibration already implemented on the robot. Our approach is characterized by the following features:

- The model utilized is a Neural Network. Its advantages include a vast number of parameters that define the model, enabling it to effectively describe nonlinear phenomena without the need for an explicitly defined formula.

- Calibration is conducted in-situ, using data collected in experiments where the sensor is already mounted on the robot; this approach is advantageous over *ex-situ* calibration, which often suffers from reduced accuracy when the sensor is mounted on the robot.

- In addition to the raw wrench output from the primary calibration, the model also receives other inputs. Specifically, we proposed two models: one that takes as input raw wrench, angular velocity, linear acceleration, and sensor temperature (resulting in a $13D$ vector input), and another that takes as input only wrench and temperature (resulting in a $7D$ vector input).

To the best of the author's knowledge, there are no calibration methods in the literature for $6D$ force-torque sensors in humanoid robotics that encompass all these characteristics. The results obtained in the testing phase on several datasets demonstrate the ability of the models developed in this study to significantly improve upon the current linear calibration present on the robot and, furthermore, they have shown superior performance compared to polynomial models developed by [1] using the same datasets for calibration.

Subsequently, the newly developed NN models were applied to the actual robot. While the models characterized by 13 inputs reported very noisy measurements (mostly due to noise in accelerometer and gyroscope data), the models with 7 inputs yielded good results and notably improved the measurement. This improvement was observed through a pair of tests (denoted as tests B and C in chapter 6), which provided both a direct measure of improvement by comparing FT data with expected measurements (test B) and an indirect measure by showing enhanced PD + gravity compensation control when using secondary calibration via NN (test C).

Below, a concise summary of the content presented in each chapter. Part I is dedicated to introducing the fundamental theoretical knowledge underlying this work, specifically:

- Chapter 1 introduces $6D$ strain gauge force-torque sensors and the physical principles upon which they are based;
- Chapter 2 presents the main theoretical foundations of neural networks;
- Chapter 3 provides the basics of rigid multibody dynamics, useful for defining the loss functions required for model training.

Part II, on the other hand, focuses on the contribution this work makes to the state of the art, specifically:

- Chapter 4 talks about the collection and processing of data required for model training and testing;
- Chapter 5 describes the development of NN models and the results obtained in the testing phases; furthermore, a comparison of results with polynomial models from [1] using the same datasets for calibration is performed;
- Chapter 6 presents the deployment of NN models and the results obtained in the testing of models with 7 inputs.

Future perspectives

Although the calibration process presented in this thesis demonstrates satisfactory results both in comparison to the current calibration on the robot and to a benchmark present in the state of the art [1], there may still be further room for improvement. One possible avenue for development could be the use of a Physics Informed Neural Network that somehow considers the variation in the internal dynamics of the sensor due to temperature; another potential development could be the use of a Recurrent Neural Network, which could exploit the relationship between the measured wrench at two close instants of time.

Bibliography

[1] Mohamed H.A.O., Nava G., Vanteddu P.R., Braghin F., Pucci D., *Nonlinear In-situ Calibration of Strain-Gauge Force/Torque Sensors for Humanoid Robots.* https://doi.org/10.48550/arXiv.2312.09846 (2023)

[2] Doebelin E. O., *Measurement Systems Application and Design*, Mc Graw Hill (2004)

[3] Fraden J., *Handbook of Modern Sensors*, Fourth Edition, Springer (2010)

[4] Stefănescu D. M., *Handbook of Force Transducers: Principles and Components*, Springer (2011)

[5] Andrade Chavez F. J., *Force-Torque Sensing in Robotics*, PhD Thesis (2018). https://fjandrad.github.io/PhDThesis/Thesis.pdf

[6] Hoffmann K., *Applying the wheatstone bridge circuit*, HBM Germany (2009)

[7] Cybenko G., *Approximation by Superpositions of a Sigmoidal Function*, Mathematics of Control, Signals and Systems (1989)

[8] Hornik K., Stinchcombe M., White H., *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks (1989)

[9] Aggarwal C. C., *Neural Networks and Deep Learning*, Springer Cham, 2nd edition, (2023). https://doi.org/10.1007/978-3-031-29642-0

[10] Bengio Y., Simard P., Frasconi P.*Learning Long-Term Dependencies with Gradient Descent is Difficult.* IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 5, NO. 2, MARCH I994

[11] Pascanu R., Dauphin Y.N., Ganguli S., Bengio Y. *On the saddle point problem for non-convex optimization.* https://arxiv.org/abs/1405.4604 (2014)

[12] Rumelhart D., Hinton G., Williams R. *Learning representations by back-propagating errors*, Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

[13] Diederik P. Kingma, Jimmy Ba. *Adam: A Method for Stochastic Optimization.* In International Conference on Learning Representations (ICLR), 2015.

[14] Andrade Chavez F. J., Nava G., Traversaro S., Nori F., Pucci D. *Model Based In Situ Calibration with Temperature compensation of 6 axis Force Torque Sensors* https://arxiv.org/pdf/1812.00650 (2018)

[15] Tien-Fu L., Grier C.I.L., Juan R.H. *Neural-Network-Based 3D Force/Torque Sensor Calibration for Robot Applications*, Engineering Applications of Artificial Intelligence Vol. 10, No. 1, pp. 87-97 (1997)

[16] Hyun S.O., Gitae K., Uikyum K., Joon K.S., Won S.Y. and Hyouk R.C., *Force/-Torque Sensor Calibration Method by Using Deep-Learning*, 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI) (2017)

[17] Traversaro S., *Modelling, Estimation and Identification of Hu- manoid Robots Dynamics*. PhD thesis. (2017). url: https://github.com/traversaro/ traversaro-phd-thesis

[18] Romualdi G., *Online Control of Humanoid Robot Locomotion*. PhD thesis. (2022). url: https://github.com/GiulioRomualdi/romualdi-phd-thesis

[19] Ferigo D., *Simulation Architectures for Reinforcement Learning applied to Robotics*. PhD thesis. (2022). url: https://github.com/diegoferigo/phd-thesis

[20] Lynch K.M., Park F.C., *Modern Robotics Mechanics, Planning, and Control*

[21] Nori F., Traversaro S., Eljaik J., Romano F., Del Prete A., Pucci D., *iCub Whole-body Control through Force Regulation on Rigid Noncoplanar Contacts*. Frontiers in Robotics and AI, volume 2, number 6. url: http://www.frontiersin.org/humanoid_robotics/10.3389/frobt.2015.00006/abstract doi: 10.3389/frobt.2015.00006 (2015)

[22] Savitzky A., Golay M. J. E., *Smoothing and Differentiation of Data by Simplified Least Squares Procedures.* Analytical Chemistry, 36(8), 1627-1639. (1964)

[23] Sorrentino I., Romualdi G., Pucci D., *UKF-Based Sensor Fusion for Joint-Torque Sensorless Humanoid Robots.*, https://arxiv.org/html/2402.18380v1 (2024)

[24] Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Köpf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J., Chintala S., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, https://arxiv.org/abs/1912.01703 (2019)

[25] Akiba T., Sano S., Yanase T., Ohta T., Koyama M., *Optuna: A Next-generation Hyperparameter Optimization Framework*, https://arxiv.org/abs/1907.10902 (2019)

[26] Glorot X., Bengio Y., *Understanding the difficulty of training deep feedforward neural networks* (2010)

[27] *https://it.linkedin.com/pulse/reti-neurali-cosa-sono-come-funzionano-e-le-applicazioni-bnova-qv1cf*

[28] *https://www.elprocus.com/smart-sensor/*