POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Feedback Error Learning for Adaptive Soft Robot Control

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Niccolò Enrico Veronese**

Student ID: 995532
Advisor: Prof. Paolo Rocco
Co-advisor: Prof. Perla Maiolino
Academic Year: 2022-2023

Abstract

Soft robotics sits at the intersection of various disciplines, including material science, biology, continuum mechanics, and robotics. Soft robots are appealing in a wide variety of tasks thanks to their inherent advantages in safety, compliance, and adaptability. However, accurate modelling and control of soft robots are still significantly challenging. Model-based controllers, relying on physical models, face difficulties in their mathematical description and subsequent computation. A possible solution to the modelling need, lies in the development of pure data-driven controllers. However, they struggle in dealing with changes in the robot dynamics and thus they are based on long and complex offline retraining phases, after every dynamic variation.

In this context, this thesis proposes a control scheme implementing an online learning strategy, based on the Feedback Error Learning framework which consists of coupling a model-based with a data-driven controller. The architecture is composed of (i) a feedback controller based on a geometric model of the robot and (ii) a data-driven controller generating a feedforward signal. The feedback controller has two roles. Firstly, it corrects the action of the feedforward controller when the tracking error increases. Secondly, it generates a learning signal to train the data-driven controller, allowing for online adaptation of the feedforward signal with respect to changes in the dynamics of the system.

The proposed approach has been validated on a real soft robot, designed to achieve omnidirectional movement and fabricated using additive manufacturing. The robot is actuated by compressed air thanks to industrial pneumatic valves and with position measurements given by a camera motion capture system. Experimental results show that the proposed method provides better performance compared to a PID controller when applied to a trajectory following task. Furthermore, the controller is shown to be capable of online adaptation to sudden changes in the dynamics of the soft robot due to a variable payload, overcoming the offline retraining need.

Keywords: soft robotics; fedback error learning control; online adaptive control; trajectory tracking

Abstract in lingua italiana

La soft robotics si colloca all'intersezione di varie discipline, tra cui la scienza dei materiali, la biologia, la meccanica dei continui e la robotica. I soft robots sono interessanti per una vasta gamma di compiti grazie ai loro vantaggi intrinseci in termini di sicurezza, flessibilità e adattabilità. Tuttavia, la loro modellazione accurata e il loro controllo rimangono sfide significative. I controllori model-based, dipendendo da modelli fisici, incontrano difficoltà nella loro descrizione matematica e conseguente computazione. Una possibile soluzione al modello analitico risiede nello sviluppo di controllori puramente basati sui dati. Questi, tuttavia, mostrano difficoltà nel gestire cambiamenti nella dinamica del robot e si basano su fasi di retraining offline lunghe e complesse dopo ogni variazione dinamica.
In questo contesto, la tesi propone uno schema di controllo che implementa una strategia di apprendimento online, incentrata sul framework Feedback Error Learning, che consiste in una combinazione di controllori basati sia sul modello del robot sia sui dati. L'architettura è composta da (i) un controllore in feedback costruito su un modello geometrico del robot e da (ii) un controllore data-driven che genera un segnale di feedforward. Il controllore in feedback ha due ruoli: corregge l'azione del controllore in feedforward quando l'errore rispetto al riferimento aumenta e genera un segnale di apprendimento per addestrare il controllore basato sui dati. Ciò consente l'adattamento online del segnale di feedforward rispetto ai cambiamenti nella dinamica del sistema.
L'approccio proposto è stato validato su un soft robot, progettato per garantire movimenti multidirezionali e fabbricato tramite additive manufacturing. Il robot è azionato da aria compressa grazie a valvole pneumatiche e con misurazioni di posizione fornite da un sistema di motion capture con telecamere. I risultati sperimentali mostrano che il metodo proposto fornisce prestazioni migliori rispetto a un controllore PID quando applicato a un compito di tracciamento di traiettorie. Inoltre, il controllore si dimostra in grado di adattarsi online a improvvisi cambiamenti nella dinamica del soft robot, rappresentati negli esperimenti da un carico variabile, superando dunque il bisogno di retraining offline.

Parole chiave: soft robotics; feedback error learning; controllo adattativo; inseguimento della traiettoria.

Contents

1 | Introduction

Soft robotics is an emerging field at the intersection of robotics and materials science, with a focus on the development of robots and robotic components that are inherently soft and compliant. It has gained significant attention for its potential applications in a wide range of fields, including human-robot interaction, and healthcare. Research and development in this field continues to expand, leading to the creation of a diverse range of soft robotic systems with increasing capabilities and functionalities. However, the modelling and control of these systems still remain challenging since they are manufactured with soft and pliable materials having strongly non-linear behaviour when actuated.

In this respect, two main approaches have been proposed to control soft robots. The first class is model-based and requires building an approximate analytical model of the robot kinematics or dynamics. Classical feedback control techniques can then be used to control the robot in closed-loop and reject disturbances or modelling inaccuracies.

However, the performance of the controller is affected by the accuracy of the model. Therefore, an effort is required to develop complex models describing the soft robot behaviour. The derivation of accurate models is one of the bottlenecks of current soft robotic control design. Furthermore, accurate models may lead to computational issues when the equations need to be solved in real-time. A solution to this consists of introducing further approximations, which affect the overall performance of the control system.

An opposite approach is based on data-driven models, where the complex behaviour of the soft robot is learnt from data instead of being analytically modelled. The main advantage of these methods is the optimization-driven derivation of models that capture the complexity of the soft robot behavior directly from data.

The drawback of these techniques is the need to collect a significant amount of data to train the model and to properly capture the dynamic behaviour of the robot. Moreover, once deployed, the validity of data-driven models is challenged by variations in the robot dynamics, due to drift of mechanical features and to the intrinsic uncertainties of the robot-environment interaction. These appear in the form of external forces not

experienced during training, uncertain and variable loads, hysteresis, drifts in material properties due to ageing or fatigue, etc.

The combination of model-based and data-driven approaches shows promise to overcome current limitations in soft robot control. In fact, on one hand, model-based approaches make clear that the use of approximated models leads to feedback controllers capable of handling uncertain dynamics (up to a certain extent). On the other hand, the gap in model accuracy can then be reduced by data-driven solutions, capable of capturing complex dynamic behaviours. Data-driven solutions lead to a more specialized control action, thus better performance, as long as the validity of the data-driven model is not challenged. In this respect, recent works have proposed techniques to continuously adapt the data-driven model to changes in the robot's dynamics. However, these methods require the dynamics of the soft actuator to change smoothly or a new data collection, and a time-consuming offline retraining phase before being deployed to the new task.

1.1. Thesis objectives

In the light of the just described framework, the aim of this thesis, developed within the Soft Robotics Laboratory of the **Oxford Robotics Institute**, part of the **Department of Engineering Science of the University of Oxford (UK)** is to propose a novel solution to the control challenges faced by soft robots, leveraging the combination of model-based and data-driven approaches. The objective is to address the inherent issues associated with defining intricate models for the system and the necessity of recurrently offline retraining data-driven controllers. These retraining processes, often time-consuming, become essential whenever dynamic changes occur in the robot system. Hence, a control scheme implementing an online learning strategy (Feedback Error Learning) is presented, where the control signal is composed of two parts. The first is a signal generated by a closed-loop feedback controller and the second consists of a feedforward signal generated by a data-driven model. The role of the feedback controller in this architecture is twofold: (i) it corrects the action of the feedforward data-driven controller when the tracking error increases (i.e. when the learning of the feedforward action is incomplete or inaccurate); (ii) its output acts as a learning signal for the data-driven model, allowing online adaptation to changes in the dynamics.

With the proposed solution, the tracking of the reference signal at steady state is mostly performed by the feedforward part of the control system, with a minimal contribution of the feedback controller. Changes in the dynamics are addressed by both controllers.

In particular, the feedback controller works on the fast time scale, providing aggressive correction signals based on tracking error. These are eventually mitigated on a slow time scale by the feeedforward controller, via online learning / adaptation of the data-driven model.

The approach has been simulated on a simplified model and then validated on a real soft robot, comparing its performance with a closed-loop control scheme in different tasks consisting of a trajectory following. In particular, experiments on the real robot have been designed to test the online adaptability of the control scheme. For instance, different weights are added and removed to the robot tip, resulting in dynamic changes.
The soft robot has been designed taking inspiration and improving existing structures in the literature, allowing for multidirectional movement, and then fabricated via additive manufacturing technique. To implement the control algorithm in the experimental setup, a pneumatic valve system (Festo valves) was integrated with a motion capture technology (OptiTrack Motive), all operating within the same software environment based on ROS2. Results show that the proposed solution significantly improves the trajectory tracking performance and is capable of learning online sudden changes in the dynamics of the robot while the task is being executed.
Summarizing, the work developed in this thesis achieves the following outcomes:

- Design and fabrication of a soft robot allowing omnidirectional bending deformation;
- Trajectory tracking control of such soft robot without the need for complex modelling;
- Improvement of the tracking error with the proposed control scheme (more than 50% reduction in the RMSE) with respect to a PID controller;
- Online adaptation to dynamic changes, represented as additional payloads of around 30% the robot weight, via data-driven feedforward controller in online learning.

1.2. Thesis outline

The remainder of the thesis is structured as follows.

Chapter 2 presents an overview of the state of the art in soft robotics. In particular, it highlights different actuation mechanisms and designs, which will be useful for justifying the choice of the proposed design for experimental evaluation. Finally, a deeper analysis of the control technologies featured in the literature is provided, showing their respective advantages and disadvantages. This analysis highlights how the proposed approach improves the performances.

Chapter 3 describes the Feedback Error Learning control architecture. It starts with the presentation of theoretical foundations, followed by an exploration of how the control theory has been translated into practical application. The chapter concludes with the presentation of simulations conducted on a simplified model to validate the proposed methodology.

Chapter 4 details the experimental setup, highlighting the design and fabrication process of the real soft robot. Moreover, it presents an analysis of the various technologies (pneumatic valves, motion capture system), all interconnected within the same ROS2 environment.

Chapter 5 shows the experimental results obtained from the real soft arm, emphasizing the enhancements achieved through the implementation of the proposed control algorithm.

Chapter 6 draws the conclusions of the whole thesis, analysing objectives given the experimental results. Moreover, it proposes possible future work and developments.

2 | State of the art

The aim of this chapter is to give an overview of the state of the art of related topics before going into details of the control architecture for adaptive soft robot control proposed in this thesis. Hence, a literature review regarding the soft robotics field, with a particular focus on the different control techniques developed, is presented in the following.

2.1. Soft robotics

Soft robotics is a growing interdisciplinary field that intersects robotics and materials science. It primarily concentrates on the creation of robots and robotic components characterized by inherent softness and compliance. This innovative approach draws inspiration from nature, in a process called biomimicry [1], where many organisms possess the ability to interact with complex and dynamic environments in a gentle and adaptable manner [2]. The key characteristic of soft robots is their flexibility, which allows them to conform to irregular shapes and interact safely with humans and delicate objects. They often use materials such as elastomers, gels, and textiles, which provide a high degree of compliance and deformability [3]. For their unique features, soft robots have gained considerable attention for their potential use across various fields including grasping, human-robot interaction, and healthcare. In fact, they can revolutionize medical devices, with applications in minimally invasive surgery, prosthetics, and rehabilitation [4]. In addition to that, another primary advantage of soft robotics is its ability to perform tasks in unstructured and uncertain environments, exploiting the natural dexterity to navigate through tight spaces and to perform complex movements [5]. Research and development in soft robotics continue to expand, leading to the creation of a diverse range of soft robotic systems with increasing capabilities and functionalities.
As the field of soft robotics is relatively new and still in the exploratory phase, there is currently no established standard for actuation methods, design principles, materials, and control algorithms. While this lack of standardization can pose challenges for those entering the field, it also offers greater flexibility and a vast space for research exploration. In the following sections, this thesis proposes an overview of the most used technologies

in the soft robotics environment, in terms of actuation, design and, in particular, control techniques.

2.1.1. Actuation

Within the whole soft robotics environment, there are several different actuation mechanisms. The majority of soft robots are characterized by pneumatic actuation, which means that their movement is generated by the injection of pressurized air into specific chambers within them, allowing for controlled deformation. Typically, the airflow is created and regulated using compressor and valve systems. In fact, the valves, thanks to their opening-closing mechanism can guarantee the commanded pressure inside the body of the soft actuator, which will deform coherently. An example of pneumatic actuation for minimally invasive surgery can be found in the work by Garbin et al.[6]. Talman et al.'s research [7] focuses on the application of a wearable soft pneumatic actuator for elbow joint rehabilitation. Furthermore, in the literature, numerous soft pneumatic actuators have been explored with the aim of replicating animal mechanisms, such as the OctArm used by McMahan et al. [8], approximating the behaviour of an octopus arm.

A second actuation mechanism for soft robots is known as tendon-driven actuation. In this approach, the robot's movement is guided by cables, which are typically located either inside the robot's body or attached to its external surface. To achieve the desired motion, the cables are sequentially pulled, often through the use of electric motors. An example of a tendon-driven soft robot can be found in the work of Xiloyannis et al. [9], where they introduce a soft glove capable of grasping objects. In their study, the motor responsible for winding and unwinding the tendon spool is complemented by a clutch-based system that secures the hand's posture once an object is grasped. Tendon-driven soft robots, being electric motor-guided, can exert stronger forces compared to pneumatic systems. However, the presence of electric motors and cables may introduce rigidity into the robot's structure, which limits the exploitation of its inherent softness and flexibility.

In addition to the previously mentioned actuation systems, other innovative approaches have been explored. For instance, Kang et al. [10] introduced an elephant trunk-like soft robot, which is actuated using thermally responsive shape memory alloys (SMA). These unique materials function as "muscles" within soft actuators and respond to changes in temperature. They can exhibit spring-like behaviour, enabling the robot to move without the requirement of an electric motor.

2.1.2. Design

Similarly to the actuation, the design of soft robots can vary significantly according to the application and the type of hardware available. The movement of soft robots is heavily influenced by their design, falling into three primary motion categories: extension and contraction, bending, and twisting. Considering the extending and contraction motion, the most popular soft robots are the so-called McKibben actuators. Their design consists mainly of a flexible inner tube covered with a helical braided shell. In this way, under pressurization, they can inflate and generate a force between the two ends of the actuator, which can be exploited for extension and contraction [11]. One significant challenge associated with these actuators is the occurrence of the "ballooning" effect. This refers to the radial expansion of the soft actuator, which hinders its ability to elongate.

Bending is the second type of primary motion achievable by soft robots. Bending actuators often feature an asymmetric geometry, such as a series of chambers or a corrugated membrane, and soft robots with this design are commonly referred to as PneuNet actuators. Their motion is driven by the expansion of the chambers or membrane. In particular, considering each chamber as a rectangular shape, when pressurized, the top layer expands while the bottom layer undergoes a compression. This differential expansion and compression creates forces that push adjacent chambers, resulting in an overall bending of the entire actuator [12].

Focusing now on the twisting movement, the actuators are usually created considering a fiber wrapping around a single chamber. When the chamber is pressurized, the wrapped fibers induce the desired twisting motion. An illustration of this concept can be found in the study by Connolly et al. [13], where the choice of fiber wrapping shape allows for various movements, including twisting. However, it's important to note that this particular motion is not extensively explored due to the challenges associated with precise motion control.

Apart from the above mentioned motions, a huge focus of the research is now on exploring possible designs which guarantee an omnidirectional actuation. They try to exploit symmetry or to be bioinspired in order to reach the highest amount of deformation and motion capability. An example of this is explained in the study of Drotman et al. [14], in which they proposed a three chambers bellow shape actuator which is able to bend in all the direction of the plane thanks to simmetry, exploiting the PneuNet-like shape given by the bellows. In fact, similarly to what happens for PneuNet, when the actuator is pressurized, each bellow will expand in a specific region, thus creating a force towards

the closest bellows, which overall generates the bending of the entire soft actuator. In addition to that, by using 3 identical chambers connected at 120° one with each other, a bending in one chamber will generate a corresponding deformation also in the others, hence enabling a multidirectional motion. This design is used as reference base for the soft robot structure proposed in this thesis.

2.1.3. Modelling and Control

As previously mentioned, since soft robotics is far from being completely explored, there is not yet an establish standard in terms of designs, actuations and materials. Moreover, the structure of soft robot is highly task driven, thus the hardware (actuation, designs, materials,...) is selected according to the specific application. This proliferation has, in turn, presented the challenge of formulating control strategies capable of effectively managing the soft robotic structures and harnessing their inherent intelligence. Typically, when addressing the control aspect, the initial approach involves the development of model-based techniques, followed by the integration of data-driven approaches, often utilizing machine learning and artificial intelligence. However, in the case of soft robotics, this conventional relationship has been reversed. In fact, the first control algorithms to emerge were purely model-free applications [15]. The primary advantage of these algorithms lays in their ability to circumvent the need for modelling soft manipulators, which inherently exhibit non-linearity and possess an infinite number of degrees of freedom. Moreover, they are independent from the manipulator shape and hence they can be exploited in cases where modelling is almost impossible or for highly nonlinear application. For instance in [16], they proposed a control architecture of a tendon-driven robot based on an estimated value of the Jacobian, without relying on a model. The Jacobian in fact was calculated by moving independently each actuator, i.e. each tendon of the robot, of an incremental amount, Δy_i while measuring the displacement Δx of the tip of the robot. The i_{th} column of J is constructed as $J_i = \Delta x/\Delta y_i$. Starting from this estimated Jacobian, they implemented an optimal control algorithm which minimizes the L2-norm of the tensions in the tendons. In order to be developed, the control algorithm needed the online update of the robot Jacobian based on the new tendon tension. Hence, they proposed a continuous optimization algorithm based on the measured displacements from actuator and end-effector sensors between the last Jacobian estimate and the one at present time. They exploited this approach for trajectory tracking.
In a subsequent study, Giorelli et al. [17] attempted to employ a similar approach, but this time using a Feedforward Neural Network to estimate the robot Jacobian, specifically

in the context of a tendon-driven soft robot. They conducted a comparative analysis between the performance achieved through Neural Network-based Jacobian estimation and a model-based estimation derived from Cosserat rod theory. The latter involves treating the soft robot's arms as an infinite series of infinitesimal rigid bodies. The results demonstrated that the neural network-based algorithm delivered superior performance in terms of position tracking accuracy. However, it requires an extended data collection and optimization phase. In contrast, the model-based algorithm demands a smaller dataset and a shorter calibration period for implementation.

While pure model-free techniques have their advantages in simplifying control algorithms for soft robots, they are not without their drawbacks. These techniques might require extensive data collection and training, making them computationally demanding and time-consuming. Furthermore, model-free approaches are typically initially developed within a simulated environment and later deployed in a real-world system. It is the case of the approach proposed by [18], in which they introduced a reinforcement learning algorithm based on Markov chain decision model for tracking tasks on a pneumatic soft robot. They first tuned and optimized the algorithm in a simulation environment and then they deployed the architecture on the real system. The transition from simulation to prototype introduces position errors stemming from two sources: (1) Static inaccuracies, arising from factors such as manufacturing imperfections, hysteresis, and differences in control resolution between the simulation and the physical prototype, and (2) approximations of the model used during training in the simulation. As a result, the performance observed in the simulation environment differed from that in the real-world scenario, notably revealing a reduction in tracking accuracy when applied to the physical system.
Furthermore, after deployment, the reliability of data-driven models is challenged by the fluctuations in robot dynamics. These variations can be attributed to the shifting mechanical characteristics, as well as inherent uncertainties associated with the interaction between the robot and its environment. Such uncertainties manifest in various forms, including unexpected external forces that were not encountered during training, uncertain and variable loads, hysteresis, material property changes due to aging or fatigue, and more.
Therefore, while model-free techniques offer an initial advantage in avoiding complex modelling, they may fall short in providing the level of control and reliability required for certain applications of soft robotics.

For the above reasons, the approach towards controlling soft robots has undergone a notable transformation in recent years, also influenced by the fact that both theoretical

research and practical experiments have demonstrated the robustness of feedback control schemes even when applied to approximate models of soft robot dynamics. In fact, even highly simplified descriptions of the system have proven sufficient to significantly enhance performance in comparison to the model-free baseline [15].
One of the biggest challenges for the model-based controller is the definition of a model of the soft actuator. This model needs to be expressive enough to accurately characterize the behavior of the robot while also providing manageable equations that can be effectively deployed and implemented in real-time applications. The research in this field is still growing but, so far, two main modelling approaches have been explored, which will be analysed in the following.

Piecewise Constant Curvature (PCC) model

The first and the most widely used model is known as the Piecewise Constant Curvature (PCC) model. This modeling technique simplifies the soft robot's body by approximating it as a rod, in which various types of strains, such as elongation, torsion, and shear, are disregarded, except for curvature. Such rod represents the central axis of the robot which is then discretized in a series of arcs, each one assumed to have constant curvature through time, connected in nodal points. Although this assumption is quite stringent, it aligns well with the design of most soft actuators, where one axis predominantly influences their shape (e.g., Octopus arm or elephant trunk). As a result, their entire configuration can be effectively approximated by neglecting volumetric deformations and concentrating on the behavior of their central axis (fig. 2.1 (a)).
Given the discretization of the PCC model, the kinematics of the soft robot can be fully described considering n reference frames {S1},...,{Sn} attached on each node, meaning at the ends of each segment, plus one base frame {S0}. In fact, it is possible to describe the kinematics by n homogeneous transformations $T_0^{n_1}, T_{n_1}^{n_2}, ..., T_{n_n}^1$ which map each reference system to the subsequent one. (fig. 2.1 (b)). These associated transformation matrices T_i^{i+1} are parameterized by a finite set of variables which represent the robot configuration q.
To move to the dynamic model of the robot, given the kinematic, the idea is to have a map between the PCC formulation and an equivalent rigid robot body formulation with the Denavit-Hartemberg frames. The core concept behind the above process is that it is possible to approximate each CC segment of the soft robot with a sequence of rigid robot link, either rotational (R) or prismatic (P), as described in [19].

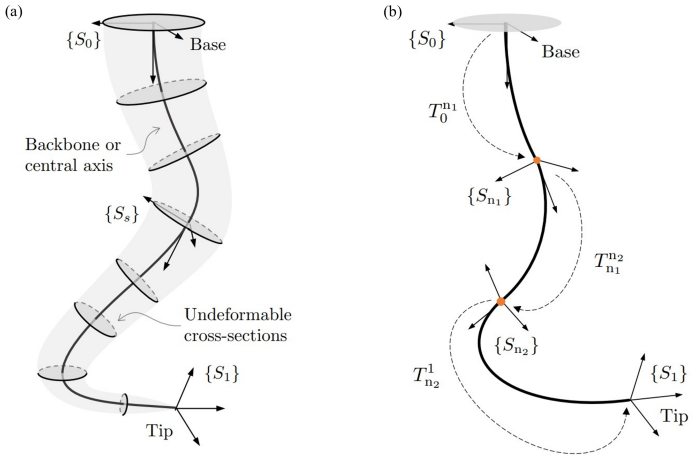

Figure 2.1: (a) Discretization of the soft arm geometry considering its central backbone. (b) PCC kinematics [15]

Starting from the PCC model, many articles proposed different control algorithms to tackle the trajectory tracking problem. For instance, in [19], they proposed a PD control action for trajectory regulation in the "joint" space on the soft arm, considering a planar case. The configuration of the robot can be described by just one variable that is the degree of curvature between two adiacent reference frames, as shown in fig. 2.2. This article was one of the first example of real application of model-based control for soft robot.

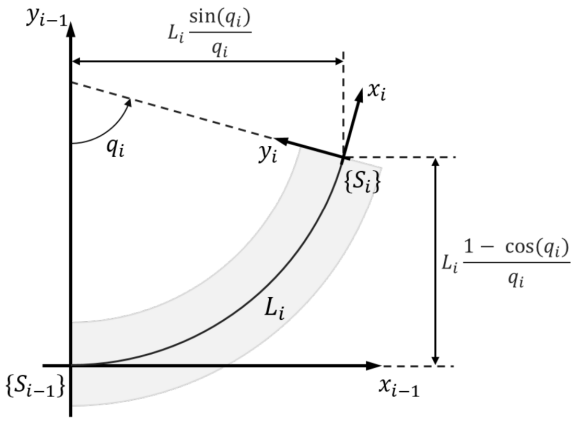

Figure 2.2: Kinematic representation of the i_{th} planar constant curvature segment. The length of the segment is L_i, and q_i is the degree of curvature [19].

Katzschmann et al. [20] extended the previous article considering a spacial movement of the robot. However, in this case, the PCC model becomes even more complex because

the robot configuration for the i_{th} segment should now be described by more than one parameter, increasing the computational effort. They tried to use a PD control for trajectory tracking but the results, although promising, leave room for improvements.
Azizkhani et al. [21] proposed a passivity based control of a soft pneumatic actuator for trajectory tracking. They compared the performance obtained with that control algorithm against the ones obtained with a more standard PD+Feedback linearization. The passivity based control showed improved performance, mainly due to the fact that the Feedback linearization relies on the assumption that the model parameters are exact and thus can be perfectly compensated. However, for such a nonlinear system as a soft pneumatic robot, it is almost impossible to guarantee that hypothesis. They also provided an adaptive passivity based scheme which tackle the problem of online adaptation to external disturbances. However, the formulation becomes more complex and difficult to implement, without a significant improvement in the tracking performance..

FEM model

To solve the problem of high level of model approximations or difficulty in implementing the control algorithms, a new wave of modelling formulation has grown in recent years, and it is based on more and more accurate Finite Element Models (FEM) simulators. The core principle behind FEM is that the geometric shape of the soft robot is described by identifying a mesh, which is a set of nodes together with the information about neighboring nodes. If the position of the nodes is known, an approximation of the entire volume results from interpolation. This particular type of modelling is helpful in case the soft robot volume cannot be described just by considering its central backbone.
For instance, Duriez et al. [22] utilized a real-time Finite Element Method (FEM) simulator as a model for their soft robot, which shared structural similarities with a parallel rigid robot. They discretized the robot shape using a mesh comprising approximately 3000 tetrahedral elements connected by around 1000 nodes. They implemented an open-loop controller, based on a Gauss-Seidel (GS) iterative solver, to guide the robot to a specific position in space, rather than attempting to track a reference trajectory. In a sample of 36 positions, they achieved a mean error of 1.4 mm, with the robot actual position being measured through an OptiTrack system.
One of the challenges with real-time Finite Element Method (FEM) modeling is that its accuracy is significantly influenced by the number of mesh nodes, resulting in models with a large number of degrees of freedom. However, the high dimensionality makes this approach impractical for developing model-based control laws. Consequently, model reduction techniques are employed to obtain a system with a reduced number of variables. In a subsequent article, Katzschmann et al. [23] tackled this issue by implementing

a FEM model with dimensionality reduction based on singular value decomposition to manage a more reasonable number of degrees of freedom. Additionally, they introduced a closed-loop control architecture for trajectory tracking. However, the performance was not entirely satisfactory, with a mean position error in the range of centimeters.
Another problem related to real-time FEM in the loop, is that the speed of the control action is highly related to the refresh rate of the FEM model. Thus, if the FEM simulator cannot sustain a high refresh rate, the controller will not rely on an accurate model, hence deteriorating the performance.

To address the aforementioned challenges associated with both pure model-free and pure model-based methodologies, recently new control architectures have been explored. In particular, the combination of model-based and data-driven approaches shows promise to overcome current limitations in soft robot control. With model-based approaches it is clear that, even with an approximated models, the feedback controllers are capable of handling uncertain dynamics (up to a certain extent). However, the most used PCC model is based on a strong approximation of the robot and it is highly dependant on its design, while the solution of a real-time FEM model requires high computational effort. These issues, especially the model accuracy can then be reduced by data-driven solutions, capable of capturing complex dynamic behaviours. Solutions that are model-free result in more specialized control actions, leading to improved performance, provided the integrity of the data-driven model remains unchallenged by dynamic changes in the soft robot. In this context, recent studies have introduced techniques for continuously adapting the data-driven model to alterations in the robot's dynamics [24, 25]. However, the study of Tang et al. [24] was based on the assumption that the dynamic changes happen with a smooth and slow transition, often impractical in real applications. Instead, [25] required a collection of new data, and a time consuming offline retraining phase after every new dynamic change, represented as an added payload to the robot, and before deployment in the new task.
In contrast, the algorithm proposed in this thesis, leveraging the synergy between model-based and data-driven approaches, ensures swift online adaptation of the control action in response to fast changes in robot dynamics, all without the necessity for an offline retraining phase after each change. Moreover, it is based on a static model of the robot, exploiting its geometric properties. Thus, it does not rely on a complex PCC formulation or on a high computational real-time FEM.

3 | Control Architecture

As already discussed in Chapter 1, the aim of this work is to develop a control algorithm which, thanks to the coupling of model-based and data-driven strategies, is able to online adapt to dynamic changes in the robot structure, without relying on complex or computational demanding models. Section 3.1 highlights the main ideas behind the proposed control technique, with theoretical considerations. Section 3.2 shows how the theoretical concepts have been applied in a practical scenario. Finally, Section 3.3 presents a simulation of the control architecture in a simplified scenario, analysing the performance.

3.1. Feedback Error Learning theory

In this Section, a description of the Feedback Error Learning method is provided, referencing to the control scheme shown in fig. 3.1.

The control objective is to regulate the position of the soft robot end-effector $\mathbf{x}(t)$ to follow a specific reference trajectory $\mathbf{x}^*(t)$. From this point on, to keep the notation lightweight the dependency from t is removed.

The proposed control scheme is composed of two main parts - a feedforward and a feedback controller. The control input $\mathbf{u} \in \Re^M$ to the plant, (i.e. the soft robot) is given by:

$$\mathbf{u} = \mathbf{u_{fb}} + \mathbf{u_{ff}} \tag{3.1}$$

where $\mathbf{u_{fb}} \in \Re^M$ and $\mathbf{u_{ff}} \in \Re^M$ are the feedback and feedforward signals respectively and M is the number of controllable signals of the soft robot. As previously explained, the feedforward controller consists of a data-driven model which must be tuned to possibly achieve a perfect tracking $\mathbf{x}^* = \mathbf{x}$. This is essentially an attempt to adapt the feedforward to generate the optimal input $\mathbf{u}^*$ so that

$$\mathbf{x} = \mathbf{x}^* = P\mathbf{u}^* \tag{3.2}$$

where $P(\cdot)$ is a generic nonlinear function indicating the Plant ($P(\mathbf{u}(t), \cdot)$) in fig. 3.1. In

our case, the Plant represents the soft robot.
Clearly, $\mathbf{u}^*$ is not known. However, if the feedback controller in fig. 3.1 guarantees good tracking performance, the tracking error is minimized, $\mathbf{e} = \mathbf{x}^* - \mathbf{x} \cong 0$, and we get

$$0 \cong \mathbf{e} = \mathbf{x}^* - \mathbf{x} = \mathbf{x}^* - P\mathbf{u}. \tag{3.3}$$

This implies that $\mathbf{u}$ is a good approximation of the desired input $\mathbf{u}^*$

$$\mathbf{u}^* \cong \mathbf{u} = \mathbf{u_{fb}} + \mathbf{u_{ff}} \tag{3.4}$$

That is, under the assumption that the feedback controller delivers good tracking performance, the signal $\mathbf{u}$ is a good approximation of $\mathbf{u}^*$ and can be used for learning.
Consider now the case where the feedforward part of the scheme is assumed to be function of a set of learnable weights. Therefore, the feedforward signal is defined as:

$$\mathbf{u_{ff}} = \boldsymbol{W}^\intercal \Phi(\mathbf{x}^*) \tag{3.5}$$

where $\boldsymbol{W} \in \Re^{N \times M}$ is the matrix of weights, $\Phi(\mathbf{x}^*) \in \Re^N$ is a generic non-linear function taking a number of future samples of the desired trajectory and N is the dimension of the non-linear function.

To achieve perfect tracking using the feedforward signal alone, the problem above can be reformulated as estimate the ideal weights $\boldsymbol{W}^*$ producing the ideal input $\mathbf{u}^*$ corresponding to the solution of the tracking problem:

$$\mathbf{x}^* = P\mathbf{u}^* = P\boldsymbol{W}^{*\intercal}\Phi(\mathbf{x}^*) \tag{3.6}$$

Now, let J be the cost function defined as:

$$J = \frac{1}{2}\mathbf{e_l}^\intercal \mathbf{e_l} \tag{3.7}$$

where $\mathbf{e_l}$ is the learning error, defined as:

$$\mathbf{e_l} = (\boldsymbol{W}^{*\intercal} - \boldsymbol{W}^\intercal)\Phi(\mathbf{x}^*) = \mathbf{u}^* - \mathbf{u_{ff}} \cong \mathbf{u} - \mathbf{u_{ff}} = \mathbf{u_{fb}} \tag{3.8}$$

The cost function can be minimized at run time through gradient descent, thus updating

the weights as follows:

$$\dot{\boldsymbol{W}} = -\gamma \nabla_{\mathbf{w}} J = \gamma \Phi(\mathbf{x}^*)^\intercal \mathbf{e_l} = \gamma \Phi(\mathbf{x}^*)^\intercal \mathbf{u_{fb}} \tag{3.9}$$

Considering then a $L2$ regularization term on the cost function to prevent the risk of overfitting, such as:

$$J = \frac{1}{2} \mathbf{e_l}^\intercal \mathbf{e_l} + \alpha \boldsymbol{W}^\intercal \boldsymbol{W} \tag{3.10}$$

the weight update can be written as:

$$\dot{\boldsymbol{W}} = -\gamma \nabla_{\mathbf{w}} J = \gamma \Phi(\mathbf{x}^*)^\intercal \mathbf{e_l} - \rho \boldsymbol{W} = \gamma \Phi(\mathbf{x}^*)^\intercal \mathbf{u_{fb}} - \rho \boldsymbol{W} \tag{3.11}$$

where $\gamma \in \Re^+$ is the learning rate and $\rho = 2\gamma\alpha \in \Re^+$ is a regularization coefficient, acting as a forgetting factor.

Overall, the control scheme is based on a simple idea. The feedback controller provides an aggressive reactive fast action that dominates disturbances (such as $\mathbf{u_{ff}}$ when learning is incomplete) and model uncertainties. It provides a good approximation of the ideal control signal needed to drive the soft robot towards the desired trajectory. On the slow time scale, through learning, the feedforward network stores a static representation of the ideal control input needed to achieve the desired trajectory. The cooperation of feedback and feedforward leads to improved tracking performances.

The above section gives an overview of the main idea behind the Feedback Error Learning control. A more formal proof of the convergence of the data-driven feedforward controller is given in [26]. Consider again the scheme in fig. 3.1 and the update rule in 3.9, without the regularization term which does not influence the convergence proof. The convergence property is then based on three main assumptions:

- γ is small and positive definite
- the input $\mathbf{x}^*$ is strongly mixing and strongly stationary stochastic process (processes for which the "past" and the "future" are asymptotically independent)
- the feedback controller is designed to guarantee for the convergence of $\mathbf{x}$ before learning, through learning and also after learning.

The proof exploits the technique of averaging random differential equation, Geman's theorem and the Lyapunov's second method.

The feedforward component, function of a set of learnable weights, can be rewritten as

$$\mathbf{u_{ff}} = \phi(\boldsymbol{W}, \mathbf{x}^*) \tag{3.12}$$

where $\mathbf{W}$ are the weights and $\mathbf{x}^*$ is the input. Equation (3.9) can be written as

$$\dot{\boldsymbol{W}}(t,\omega) = \gamma \left(\frac{\partial \phi(t,\omega)}{\partial \boldsymbol{W}} \right)^\intercal \mathbf{u_{fb}}(t,\omega) \tag{3.13}$$

where t is the time, ω is a sample point in a probability space and $\mathbf{u_{fb}}$ is the feedback control action which depends on the error $\mathbf{e} = \mathbf{x}^* - \mathbf{x}$ and on the weights $\boldsymbol{W}$ of the feedforward (exploiting the fact that $\mathbf{u_{fb}} = \mathbf{u} - \mathbf{u_{ff}}$). The solution of the following averaged equation is a good approximate to the solution of eq. (3.13) for small γ, as proven by Geman's theorem [27].

$$\dot{\boldsymbol{W}} = \gamma \mathbb{E}\left[\left(\frac{\partial \phi(t,\omega)}{\partial \boldsymbol{W}} \right)^\intercal \mathbf{u_{fb}} \right] \tag{3.14}$$

Thus, we consider the following function V as Lyapunov candidate for the averaged equation 3.14.

$$V = \mathbb{E}\left[L(\mathbf{x}, \mathbf{x}^*, \boldsymbol{W}) \right] \geq 0 \tag{3.15}$$

where L is defined as

$$L = \frac{1}{2} \mathbf{u_{fb}}(t,\omega)^\intercal \mathbf{u_{fb}}(t,\omega) \tag{3.16}$$

The time derivative of V can be calculated as

$$\dot{V} = \mathbb{E}\left[\mathbf{u_{fb}}^\intercal \left(\frac{\partial \mathbf{u_{fb}}(\mathbf{x}, \mathbf{x}^*, \mathbf{W})}{\partial t} \right) \right] \tag{3.17}$$

The idea is now to use the chain and linear property of the derivative to split the contribution of $\mathbf{x}, \mathbf{x}^*$ from the one of $\boldsymbol{W}$. Hence,

$$\frac{\partial \mathbf{u_{fb}}(\mathbf{x}, \mathbf{x}^*, \mathbf{W})}{\partial t} = \frac{\partial \mathbf{u_{fb}}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial t} + \frac{\partial \mathbf{u_{fb}}}{\partial \mathbf{x}^*} \frac{\partial \mathbf{x}^*}{\partial t} + \frac{\partial \mathbf{u_{fb}}}{\partial \boldsymbol{W}} \frac{\partial \boldsymbol{W}}{\partial t} \tag{3.18}$$

By substituting it inside eq. (3.17) we get:

$$\dot{V} = \mathbb{E}\left[\mathbf{u_{fb}}^\intercal \left(\frac{\partial \mathbf{u_{fb}}(\mathbf{x}, \mathbf{x}^*)}{\partial t} \right) \right] + \mathbb{E}\left[\mathbf{u_{fb}}^\intercal \left(\frac{\partial \mathbf{u_{fb}}(\mathbf{W})}{\partial t} \right) \right] \tag{3.19}$$

The first term of the above equation can be written as

$$\frac{\partial \mathbb{E}[L(\mathbf{x}, \mathbf{x}^*)]}{\partial t} \tag{3.20}$$

For the third assumption, $\mathbf{x}$ and L becomes a Baire function of the strongly stationary process $\mathbf{x}^*$. Consequently, $\mathbb{E}[L(\mathbf{x}, \mathbf{x}^*)]$ does not depend on time and thus the first term of eq. (3.19) vanishes.
Computing then the second term of eq. (3.19), leveraging the fact that it depends on the feedback contribution and thus on the feedback gain which can be tuned to guarantee the third assumption, we obtain that:

$$\dot{V} \leq 0 \tag{3.21}$$

Hence, for the Lyapunov theorem, eq. (3.14) converges and thus also eq. (3.9).

Other articles deal with properties of the entire Feedback Error Learning architecture. For instance, the stability of the control scheme is discussed in [28]. Miyamura et al. develop the proof in the simplest framework of linear time-invariant system, thus the Plant in fig. 3.1 is represented by a transfer function $P = P(s)$. In their proof, the feedforward controller is defined as an adaptive controller. This latter is function of parameters vector $\boldsymbol{\theta}$, rather than of weights of a neural network. However, the Feedback Error Learning framework is general and different architectures can be applied. The only constraint is that the feedforward should be function of tunable and adaptable parameters (or weights).
Still considering the scheme in fig. 3.1, Miyamura et al. define:

$$\mathbf{u_{ff}}(t) = \boldsymbol{\theta}^\intercal(t)\boldsymbol{\zeta}(t) \tag{3.22}$$

where $\boldsymbol{\theta}$ is the parameter vector and $\boldsymbol{\zeta}$ is the vector representing the states and the input of the adaptive feedforward controller. More in detail, the adaptive controller is parametrized as:

$$\begin{aligned} \frac{d}{dt}\zeta_1(t) &= \mathbf{F}\zeta_1(t) + \mathbf{g}\mathbf{x}^*(t) \\ \frac{d}{dt}\zeta_2(t) &= \mathbf{F}\zeta_2(t) + \mathbf{g}\mathbf{u}^*(t) \\ \mathbf{u_{ff}}(t) &= c^\intercal\zeta_1(t) + d^\intercal\zeta_2(t) + k\mathbf{x}^*(t) = \boldsymbol{\theta}^\intercal(t)\boldsymbol{\zeta}(t) \end{aligned} \tag{3.23}$$

where $\mathbf{F}$ is any stable matrix and $\mathbf{g}$ is any vector with $(\mathbf{F}; \mathbf{g})$ being controllable.
Thus, the learning rule eq. (3.9) can be rewritten as:

$$\dot{\boldsymbol{\theta}} = \gamma K_1(s)\mathbf{e}(t)\boldsymbol{\zeta}(t) \tag{3.24}$$

where the feedback controller is represented as $\mathbf{u_{fb}}(\mathbf{t}) = K_1(s)\mathbf{e}(\mathbf{t})$, with a slightly abuse of notation. The proof of stability of the Feedback Error Leaning scheme is provided under the following three assumptions:

- The plant P is stable and has stable inverse P^{-1}.
- The upper bound of the order of P is known.
- The high frequency gain $K_0 = \lim_{s\to\infty} P(s)$ is assumed to be positive.

If K_0 is negative in the third assumption, the subsequent results are valid by taking $-P(s)$ instead of $P(s)$. Hence, such hypothesis can be relaxed to the assumption that the sign of the high frequency gain is known.
Considering now the ideal feedforward controller to be estimated, its explicit parametrization is given by:

$$\begin{aligned} \frac{d}{dt}\eta_1(t) &= \mathbf{F}\eta_1(t) + \mathbf{g}\mathbf{x}^*(t) \\ \frac{d}{dt}\eta_2(t) &= \mathbf{F}\eta_2(t) + \mathbf{g}\mathbf{u}^*(t) \\ \mathbf{u}^*(t) &= c_0^\intercal\eta_1(t) + d_0^\intercal\eta_2(t) + k_0\mathbf{x}^*(t) = \boldsymbol{\theta}_\mathbf{0}^\intercal(t)\boldsymbol{\eta}(t) \end{aligned} \tag{3.25}$$

where $\boldsymbol{\theta_0}$ is the vector of ideal parameters to be estimated to generate the ideal input $\mathbf{u}^*(t)$ and $\boldsymbol{\eta}(t)$ represents the states and the input of the ideal controller. $\mathbf{F}$ and $\mathbf{g}$ are the same as eq. (3.23)
Since $\mathbf{F}$ is stable the comparison between eq. (3.23) and eq. (3.25) yields the following asymptotic relationships:

$$\begin{aligned} &\zeta_1(t) \to \eta_1(t) \\ &\zeta_2(t) \to \eta_2(t) - {d_0}^T(sI - \mathbf{F})^{-1}\mathbf{g}P^{-1}(s)\mathbf{e}(t) \end{aligned} \tag{3.26}$$

Thus, always asymptotically, it is possible to derive the following equation:

$$(G(s) + K_1(s))\mathbf{e}(t) = -\boldsymbol{\zeta}(t)^\intercal\boldsymbol{\psi}(t) \tag{3.27}$$

where:

$$\begin{aligned} G(s) &:= \left(1 - d_0^\intercal(sI - \mathbf{F})^{-1}\mathbf{g}\right)P^{-1}(s) \\ \boldsymbol{\psi}(t) &:= \boldsymbol{\theta}(t) - \boldsymbol{\theta_0} \end{aligned} \tag{3.28}$$

From eq. (3.24) we have that:

$$\dot{\boldsymbol{\psi}} = \dot{\boldsymbol{\theta}} = \gamma K_1(s)\mathbf{e}(t)\boldsymbol{\zeta}(t) \tag{3.29}$$

Combining eq. (3.27) and eq. (3.29) we get:

$$\dot{\boldsymbol{\psi}} = -\gamma \boldsymbol{\zeta}(t) K_1(s)(G(s) + K_1(s))^{-1} \boldsymbol{\zeta}(t)^\intercal \boldsymbol{\psi}(t) \tag{3.30}$$

The stability of the above differential equation is proven exploiting the notion of strict positive realness.

Theorem 3.1. *Let $L(s)$ be a strongly positive real transfer function and $\zeta(t)$ be an arbitrary time-varying vector. Then, the solution $z(t)$ of the differential equation*

$$\frac{d\mathbf{z}(t)}{dt} = -\boldsymbol{\zeta}(t) L(s) \boldsymbol{\zeta}(t)^\intercal \mathbf{z}(t) \tag{3.31}$$

tends to a constant vector $\mathbf{z_0}$ such that $\lim_{t\to\infty} \zeta(t)\mathbf{z_0} = 0$.

The proof of such theorem is in [29]. We notice that eq. (3.30) is equal to eq. (3.31) when

$$L(s) := \gamma K_1(s)(G(s) + K_1(s))^{-1} \tag{3.32}$$

hence for the above theorem the eq. (3.30) is asymptotically stable if $L(s)$ is strongly positive real. Since $G(s)$ is stable (under the three assumptions above), it is always possible to choose the feedback controller $K_1(s)$ sufficiently large and positive so that $L(s)$ is strongly positive. Thus the core idea is to develop a feedback strong enough to guarantee the positive realness and thus the stability of the scheme.

It is worth noting that the first assumption (The plant P is stable and has stable inverse P^{-1}) is rather restrictive in the context of control system design. However, Myiamura et al. [28] also provide stability proof relaxing such hypothesis. They exploit similar concepts as before but in this case they introduce an approximated inverse plant $\hat{P}^{-1} = P^{-1}W(s)$ with $W(s)$ with the same relative degree as $P(s)$. In this way the relative degree of $P(s)$ which is the cause of non-invertibility, is compensated by the relative degree of $W(s)$.

Finally, other articles present stability proofs in case of plants with time delay [30], or considering the application of Feedback Error Learning in different fields [31–33].
It is interestingly to note that such architecture was born in the context of motor control, a term used in neurophysiology to express the regulation of movements in organisms by means of a nervous system. The idea was to replicate the anatomical structure of brain neural system. In fact, Kawato [26] proposed a parallelism between the motor cortex (cerebral cortex) and the feedback controller, which is fast and reactive. The lateral part of the cerebral hemisphere was instead associated with the feedforward controller, since

it has been proven to have a learning capability.

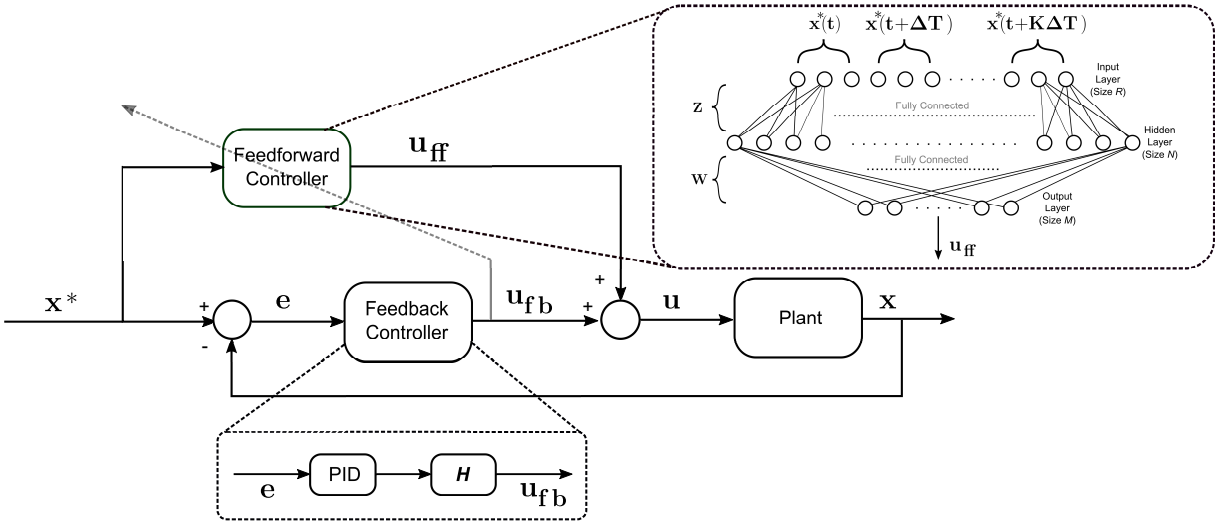

Figure 3.1: Feedback Error Learning control scheme. The data-driven model produces a feedforward control action. The feedback part of the scheme is used to correct the data-driven model (when learning is not complete) and to train it, enabling online adaptation. In the proposed implementation, the feedforward controller consists of a NN with a single hidden layer. The feedback part of the scheme is implemented with a PID controller.

3.2. Feedback Error Learning deploy

Starting from the theoretical analysis behind the Feedback Error Learning (see Section 3.1), the goal now is to move to the practical deployment on the real system.

The Feedback Error Learning scheme described in fig. 3.1 is very general, so, in principle, several different feedback and feedforward controllers can be used.
However, one of the goals of this thesis is to overcome the problems related to model-based controllers which rely complex modelling such as the PCC explained in Section 2.1.3 (see [19–21]). In fact, PCC is difficult to be developed and anyway it does not guarantee an accurate description of the robot behaviour. Moreover, we do not want to depend on FEM simulations which require high computational power.
For this reason, the idea was to consider a geometric model of the robot, leveraging symmetries in its design, and then create a linear mapping $\boldsymbol{H} \in \Re^{M\times 3}$ between the feedback control action and the actuator space, where M is the dimension of the controllable inputs and 3 represents the dimension of the actuation space (in this case 3 valves are actuated). In this way, even with a simplified modelling, it is possible to derive a control action which drives the robot.

The proposed feedback mechanism in this thesis employs a PID control for trajectory tracking of the robot within the task space. The tracking error $\mathbf{e} \in \Re^3$ in fig. 3.1 is represented in Cartesian coordinates in the end-effector space. The concept involves designing the PID in the simplest possible way to achieve reasonable basic performance and then employing the feedforward part (represented by a neural network) with learning techniques to attain high performance. It is important to note that, with a higher degree of modeling, more advanced control techniques can be considered. However, in the case of soft robots, such advanced modelling would correspond to highly complex architectures that, nonetheless, may not be entirely reliable due to the significant challenges in addressing the nonlinearities and the nearly infinite degrees of freedom inherent to such robots. The PID is based on the definition of matrix $\boldsymbol{H}$ which allows to implement a control scheme directly in the task space, without relying on a complex kinematic model. Moreover, with this solution, there is no need to solve the inverse kinematics problem, with the online computation of the Moore-Penrose pseudoinverse, thus eliminating the risk of propagating modelling errors.
Matrix $\boldsymbol{H}$ is obtained as an approximation of the relationship between PID control actions and actuation pressures, leveraging the symmetries inherently present in the soft arm design. In particular, the soft robot features three chambers, each positioned at 120° angles relative to the others (see Chapter 4). This geometric arrangement offers an opportunity to compute the PID control action, separately considering the three axes within the XYZ space, where the robot operates. Consequently, the robot is equipped with three PID controllers, each one dedicated to an axis, capable of delivering pressures as follows:

$$P_x = K_{px} e_x + K_{ix} \int e_x + K_{dx} \dot{e}_x \tag{3.33}$$

$$P_y = K_{py} e_y + K_{iy} \int e_y + K_{dy} \dot{e}_y \tag{3.34}$$

$$P_z = K_{pz} e_z + K_{iz} \int e_z + K_{dz} \dot{e}_z \tag{3.35}$$

where e_x, e_y, e_z represent respectively the error between reference and actual position of the tip of the robot along the x,y and z axis.
It is worth noting that the soft robot designed in this thesis cannot elongate/contract on the z axis, hence the contribution of P_z will be neglected ($P_z = 0$) in the experimental results presented in Chapter 5. Nevertheless, the methodology outlined below is general and thus can be applied to other designs that may involve elongation. Consequently, the theoretical impact of the z-PID is considered in the methodology.

Given this separation of the PID controllers, it is possible to compute matrix $\boldsymbol{H}$ leveraging the following methodology.
Consider the soft actuator described as three cylindrical chambers A, B, C which can be pressurized by three different valves, generating a corresponding pressure in each chamber of the robot as P_a, P_b, P_c (see fig. 3.2 (a)). For a more complete description of the robot design and shape please refer to Chapter 4. The three cylindrical chambers are connected together and this means that a movement in one of them will generate a displacement also in the others.

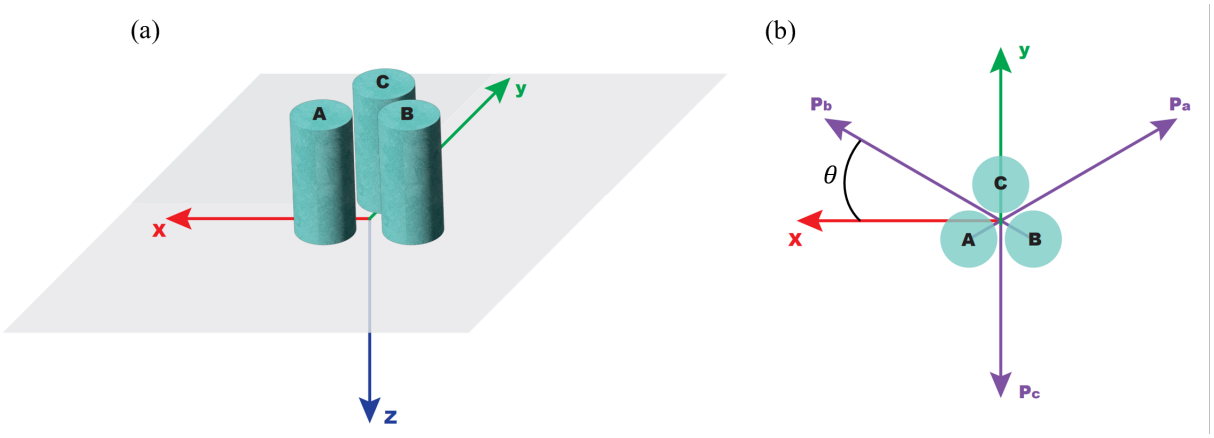

Figure 3.2: (a) Cylindrical chambers modelling the soft arm. The chambers are connected in their central part (the robot tip is the geometric centre between A,B and C). Hence a bending of one chamber will generate a corresponding bending also in the others. (b) Planar view of the soft robot. The vectors P_a, P_b, P_c represent the pressures of chamber A, B, C respectively, with their corresponding direction of bending, and thus of motion.

Given this modelling of the robot and the reference system showed in fig. 3.2, it is possible to derive the following considerations:

- an increase of P_a generates a displacement in the direction $(x, y) = (-\cos\theta, \sin\theta)$;
- an increase of P_b generates a displacement in the direction $(x, y) = (\cos\theta, \sin\theta)$;
- an increase of P_c generates a displacement in the direction $(x, y) = (0, -1)$;

with $\theta = 30°$ (see fig. 3.2 (b)).

The idea now is to consider a linear combination of the three actuation pressures P_a, P_b and P_c, correspondingly scaled by the PID contributions, to generate the movement of the tip of the robot, represented as the center of the three chambers. In the following, the combination of actuation pressures and thus of PID contributions, is splitted on each axis of the reference system for clarity.

To generate a positive movement along the x axis (fig. 3.2 (b)), it is necessary to increase the pressure in B, decrease the one in A of the same amount and not change the one in C. Thus, the PID contribution will be:

$$\begin{aligned} P_a &= -\frac{2}{\sqrt{3}}P_x \\ P_b &= +\frac{2}{\sqrt{3}}P_x \\ P_c &= 0 \end{aligned} \tag{3.36}$$

where P_x is given by 3.33.

Let's focus now on the y axis. To enable the robot to move positively along this axis, an equal amount of pressure can be simultaneously injected into chambers A and B while deflating chamber C. The pressure variation is regulated through the action of the PID controller for the y-axis (3.34). Thus, the contribution of the PID controller on the y-axis will be:

$$\begin{aligned} P_a &= +0.5P_y \\ P_b &= +0.5P_y \\ P_c &= -P_y \end{aligned} \tag{3.37}$$

For the z-axis, it is necessary to increase the pressure of the same amount in all the chambers to get a positive movement, which corresponds to an elongation. The contribution of z-PID is:

$$\begin{aligned} P_a &= +P_z \\ P_b &= +P_z \\ P_c &= +P_z \end{aligned} \tag{3.38}$$

Finally, summing all the contributions:

$$\begin{aligned} P_a &= -\frac{2}{\sqrt{3}}P_x + 0.5P_y + P_z \\ P_b &= +\frac{2}{\sqrt{3}}P_x + 0.5P_y + P_z \\ P_c &= -P_y + P_z \end{aligned} \tag{3.39}$$

In matrix form we can write:

$$\mathbf{u_{fb}} = \boldsymbol{H}\left(\boldsymbol{K_P}\mathbf{e} + \boldsymbol{K_I}\int \mathbf{e} + \boldsymbol{K_D}\dot{\mathbf{e}}\right) \tag{3.40}$$

where

$$\mathbf{u_{fb}} = \begin{bmatrix} P_a \\ P_b \\ P_c \end{bmatrix} \tag{3.41}$$

and

$$\boldsymbol{H} = \begin{bmatrix} -\frac{2}{\sqrt{3}} & 0.5 & 1 \\ \frac{2}{\sqrt{3}} & 0.5 & 1 \\ 0 & -1 & 1 \end{bmatrix} \tag{3.42}$$

which will be normalized as:

$$\boldsymbol{H} = \begin{bmatrix} -0.6667 & 0.2887 & 0.5774 \\ 0.6667 & 0.2887 & 0.5774 \\ 0 & -0.5774 & 0.5774 \end{bmatrix}. \tag{3.43}$$

The matrices $\boldsymbol{K_P}, \boldsymbol{K_I}, \boldsymbol{K_D}$ are respectively

$$\boldsymbol{K_P} = \begin{bmatrix} K_{px} & 0 & 0 \\ 0 & K_{py} & 0 \\ 0 & 0 & K_{pz} \end{bmatrix} \quad \boldsymbol{K_I} = \begin{bmatrix} K_{ix} & 0 & 0 \\ 0 & K_{iy} & 0 \\ 0 & 0 & K_{iz} \end{bmatrix} \quad \boldsymbol{K_d} = \begin{bmatrix} K_{dx} & 0 & 0 \\ 0 & K_{dy} & 0 \\ 0 & 0 & K_{dz} \end{bmatrix} \tag{3.44}$$

where the PID gains need to be appropriately tuned to generate a suitable control signal $\mathbf{u_{fb}} \in \Re^M$, which will be used for learning for the feedforward controller. In the deployment on the real soft robot, the number of controllable inputs corresponds to the number of chambers, therefore, $M = 3$.

Turning to the selection of the feedforward component, similar to the feedback, the Feedback Error Learning scheme is general and, as such, does not adhere to a specific structure. Consequently, the feedforward controller was implemented as a neural network (NN) [34], aiming to yield enhanced results without depending on overly complex machine learning structures that might not be sufficiently swift to ensure real-time control actions.

The structure of the neural network is shown in fig. 3.3 and it is based on the architecture described in 3.5.

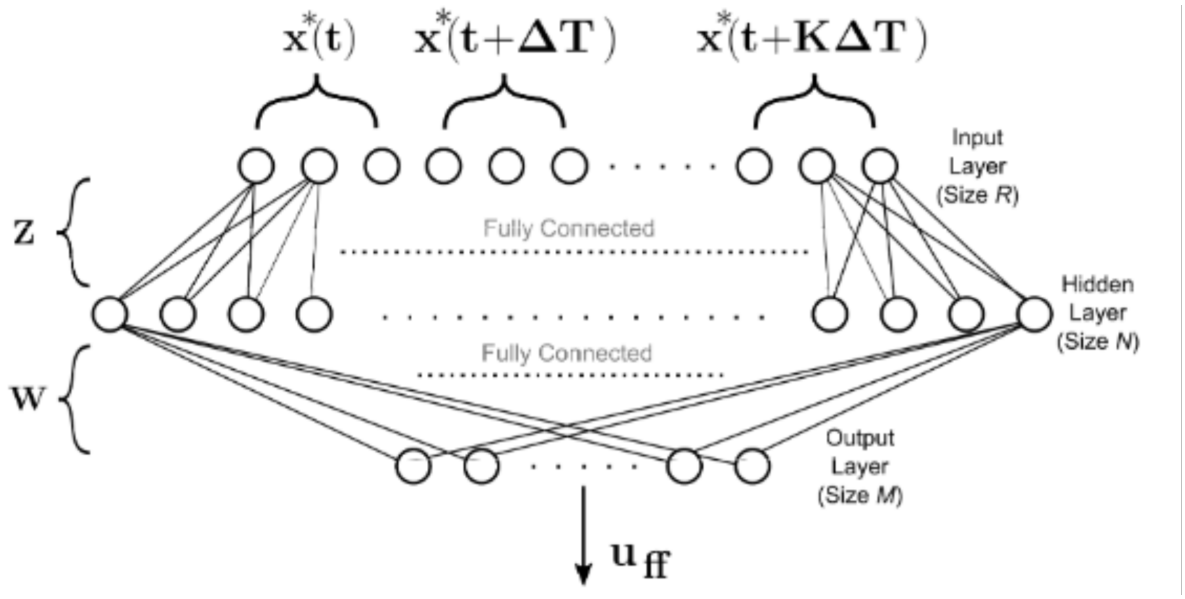

Figure 3.3: Neural Network description. The input layer takes the future reference trajectory samples (along x,y and z components), collected in the vector $r(\mathbf{x}^*) \in \Re^R$ and it propagates in the hidden layer of net, by means of $\boldsymbol{Z} \in \Re^{R\times N}$. The information is then transmitted to the output layer via $\boldsymbol{W} \in \Re^{N\times M}$ matrix, which is online updated to find the optimal weights that guarantee the tracking of the trajectory.

It has one hidden layer composed on N neurons which connects the input layer to the output layer, characterized by M neurons. As said before, since 3 chambers can be actuated $M = 3$. Therefore, the dimension of the $\mathbf{u_{ff}}$ is consistent with the one of $\mathbf{u_{fb}}$.
The input layer consists of R neurons containing future samples of the reference signal. The function $\Phi(\cdot)$ creates a non-linear mapping between the input and hidden layer and is defined as:

$$\Phi(\mathbf{x}^*) = \tanh\left(\boldsymbol{Z}^\intercal r(\mathbf{x}^*)\right) \tag{3.45}$$

where, $r(\mathbf{x}^*)$ is a function generating a vector $\in \Re^R$ containing the future samples of $\mathbf{x}^* \in \Re^3$ arranged as indicated in fig. 3.3.
In particular, given a sampling period $\Delta T \in \Re$, the three (x,y and z) components of the reference signal $\mathbf{x}^*(t + (k-1)\Delta T)$, with $k = 1, 2, \ldots, K$ are mapped into three input neurons.
Since K future samples of the reference signal are considered, the size of the input layer is $R = 3K$. The arrangement of the input components in a column vector is a process called flattening, often used in machine learning when dealing with input layers for artificial or convolutional neural network for image processing [35].

The matrix $\boldsymbol{Z} \in \Re^{R\times N}$ is randomly initialized such that $\|\boldsymbol{Z}\| = 1$ and fully connects the

input and hidden layers. This matrix is set to be constant. This is mainly due to the need for speeding up the learning to guarantee online adaptation.
Hence, the NN proposed in this thesis has the nonlearnable weights, represented by matrix $\boldsymbol{Z}$ which are kept constant through the process and they are needed to map the input layer with the hidden one.

By substituting eq. (3.45) into eq. (3.5), we get:

$$\mathbf{u_{ff}} = \boldsymbol{W}^{\intercal} \tanh\left(\boldsymbol{Z}^{\intercal} r(\mathbf{x}^*)\right). \tag{3.46}$$

in which the matrix of weights $\boldsymbol{W} \in \Re^{N \times M}$ is updated at each control step using the gradient descent method described in 3.11.

In summary, the input layer will take the future reference trajectory samples (along x,y and z components), collected in the vector $r(\mathbf{x}^*) \in \Re^R$ and it will propagate those samples in the hidden layer of net, thanks to the product by matrix $\boldsymbol{Z} \in \Re^{R \times N}$. The information is then transmitted to the output layer by means of the $\boldsymbol{W} \in \Re^{N \times M}$ matrix, which will be online updated to find the optimal weights that guarantee the tracking of the trajectory.

Overall, the artificial neural network relies on four main parameters

- γ which represents the learning rate of the online gradient descent. It determines how fast or slow the online optimization converges to the minimum of the cost function;
- ρ which is the forgetting factor used for regularization to reduce the risk of overfitting;
- N which is the number of neurons in the hidden layer;
- K which is the number of future samples of the desired trajectory. It represents how far in the future the feedforward controller needs to see to generate a good response.

The tuning of these parameters will be fundamental to guarantee the correct behaviour of the NN and thus of the entire Feedback Error Learning algorithm.
As usually happen in the machine learning field, the tuning of the parameters is a time consuming process, often based on trial and error. However, to speed up the search and improve the accuracy, a simulator (even if based on simplified model) has been considered (see Section 3.3). Instead, Chapter 5 contains a dedicated section to the experimental tuning of these values, together with the ones of the PID controller.

3.3. Feedback Error Learning simulation

The purpose of this section is to conduct an initial analysis of the Feedback Error Learning (FEL) methodology when applied to a simplified model within a simulation environment.

The model chosen for testing the control algorithm is a single degree of freedom mass-spring-damper system. This choice allowed the observation of controller behaviour while temporarily eliminating the challenges posed by nonlinearities, drifts, and changes in material properties that are inherent in a real soft robot.
Moreover, the simplified simulator enabled the analysis of the response sensitivity to neural network parameters, providing an initial insight into the values they should assume. The simulation was deployed in MATLAB r2022a within the Simulink environment. The model was characterized by the following equation:

$$m\ddot{x} + c\dot{x} + kx = u$$

where u is given by the sum of the feedback (PID) and feedforward (NN) controllers, as described in fig. 3.1. The goal was to perform a trajectory tracking task of $x^* \in \Re$, since the model had one degree of freedom.

Various simulations were conducted to assess the performance of the proposed scheme. Initially, the PID controller was tuned to ensure sufficiently accurate results, allowing $\mathbf{u_{fb}}$ to be employed as the learning signal for the neural network (NN). It was determined that a straightforward PD regulator with $K_p = 5$ and $K_d = 2$ was adequate to meet the requirements. Following this tuning phase, the NN contribution was introduced, and its hyperparameters were empirically selected. Specifically, $\gamma = 0.05, \rho = 0.01, N = 20, K = 10$ were chosen.

While these values cannot be directly extrapolated to the physical system due to differences in dimensionality and model, they provided a preliminary understanding of their potential magnitudes.
Focusing, for instance, of γ and ρ, their ratio should be around 5:1 to obtain the good trade off between tracking accuracy and overfitting avoidance. Furthermore, the number of nodes N in the inner layer of the network should be around twice that of the input layer (in fact in this case with $K = 10$ and with 1 degree of freedom, we have that $R = 1K = 10$).
The intuition behind this result is that the information in input, in this case $r(\mathbf{x}^*) \in \Re^R$

needs to be expanded within the layers of the NN and thus the inner layer should be bigger than the input one.

Sensitivity to hyper-parameter γ

The goal of this paragraph is to show a sensitivity analysis with respect to the hyper-parameter γ of the artificial neural network. It thus highlights the arguments behind the choice of the parameter value.

The parameter γ denotes the learning rate of the gradient descent algorithm in eq. (3.11). It essentially governs the pace at which the cost function is minimized. Correctly tuning this parameter is crucial because a value that is too small would result in an excessively prolonged learning phase, during which the system response may not align correctly. Conversely, a value that is too large could lead to instability, preventing the algorithm from effectively converging to the minimum point of the cost function.

As evident in fig. 3.4, when $\gamma = 0$ (g = 0, red line, indicating no adaptation), the response is unsatisfactory as the neural network remains consistently in the learning phase. It acts as a pure disturbance for the feedback controller. Consequently, the incorrect control actions generated by the NN must be continually compensated by the PID controller, resulting in suboptimal performance. On the contrary, by increasing the value of γ, the neural network effectively learns optimal weights, leading to improved control actions and enhanced performance.
However, it is essential to note that as γ increases, the time required for adaptation also grows, reaching a point where the response becomes unstable due to divergence in the gradient descent (after $\gamma = 0.15$, blue line). For these reasons the value of γ was fixed at 0.05.

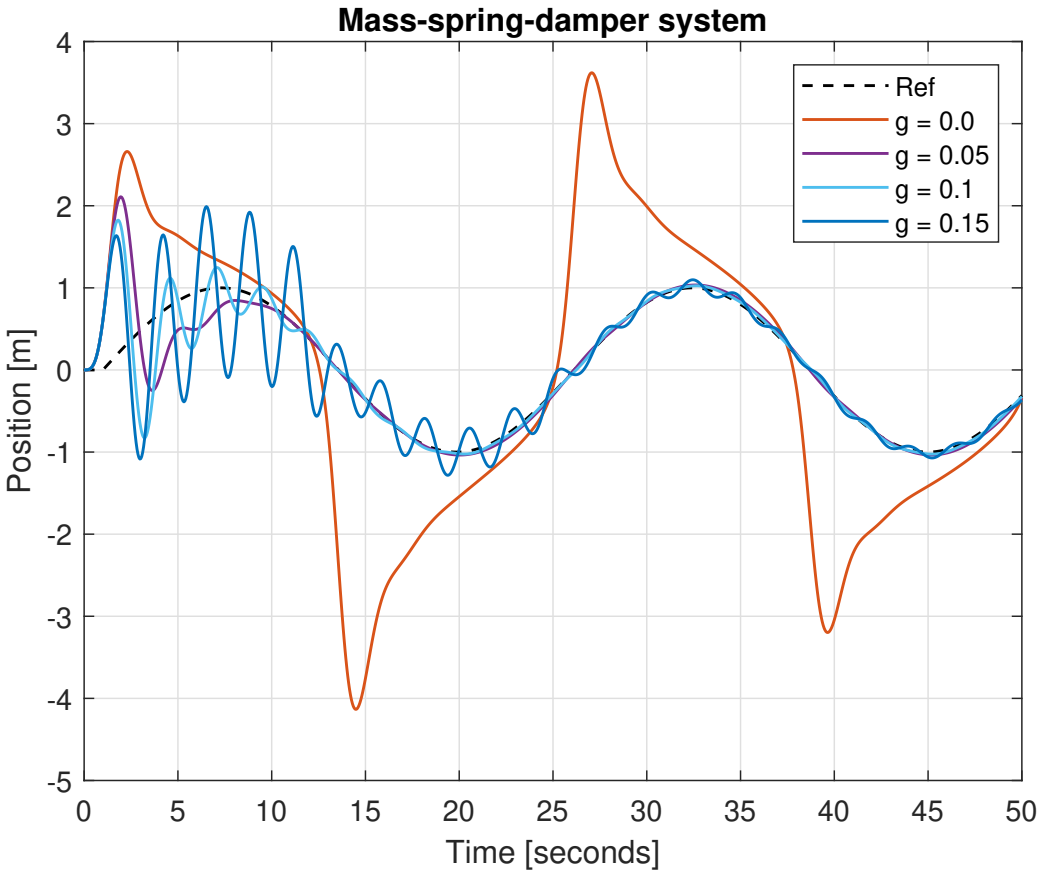

Figure 3.4: Sensitivity to γ variations. As γ increases the response becomes more oscillatory, up to divergence above $\gamma = 0.15$. The red line ($\gamma = 0$) shows the response when the NN does not update its weights, thus no adaptation. In this case the neural network is a pure disturbance for the PID controller.

Sensitivity to the hyper-parameter ρ

With this simulation the goal was to understand the effect of the parameter ρ on the response of the Feedback Error Learning architecture.

It is worth noting that the most significant impact of this parameter was not on the shape of the response, which primarily varied during the initial learning phase by a minimal amount. Instead,the notable effect was on the weight steady state. In fact, as shown in fig. 3.5 the weights in the left figure ($\rho = 0$) are more dispersed and less uniform, while the ones in the right figure ($\rho = 0.1$) are characterized by a convergence of all of them towards the same value, resulting in a more uniform adaptation.

Finally, as evident in both cases, the weights persistently oscillate around a specific value close to 0, following the behaviour of the reference trajectory.

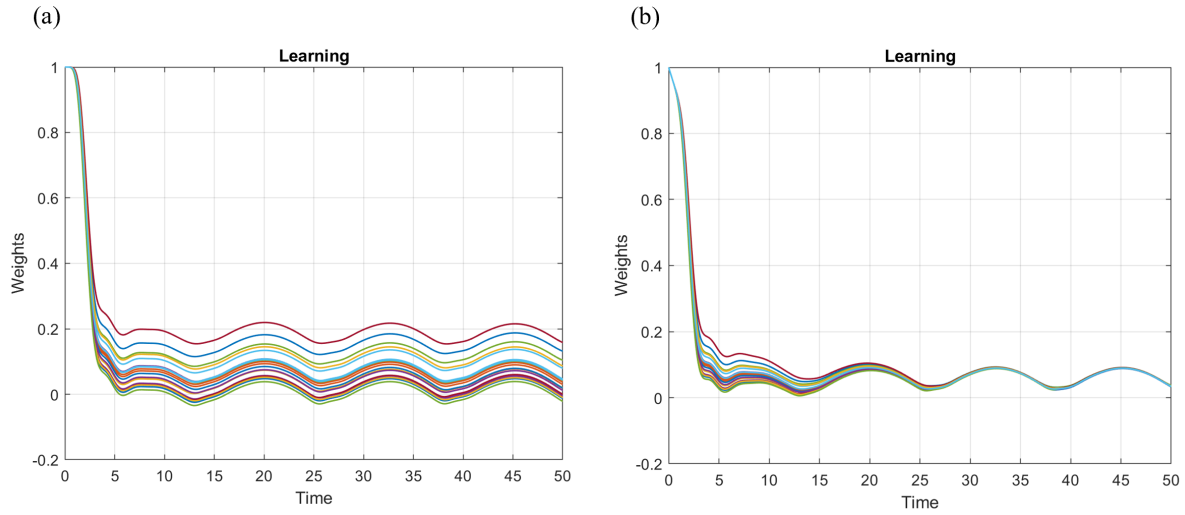

Figure 3.5: (a) Weights adaptation with $\rho = 0.0$ (b) Weights adaptation with $\rho = 0.1$. Both of them start from a random value equal to 1 and in about 15 seconds reach the steady state.

Sensitivity to the hyper-parameter N

This section outlines the sensitivity analysis with respect to the hyper-parameter N which represents the number of neurons in the hidden layer.

As a guiding principle, the number of neurons in the hidden layer should be sufficiently large to ensure the descriptive capability of the neural network while avoiding exceeding a certain value that could introduce complexity in the adaptation law or slow down the learning phase.

As depicted in fig. 3.6, when the hidden layer comprises only one neuron (indicated by the red line), the network lacks the descriptive capability necessary for perfect trajectory tracking. Consequently, the response exhibits a delay similar to that observed in the PID-only behavior. By increasing the size of the NN, the response becomes progressively more accurate, given the additional learning power to capture the features of the ideal control action.

However, it is essential to strike a balance, as an excessively large network may generate higher initial incorrect responses, thereby necessitating increased effort from the PID controller to compensate for these errors.

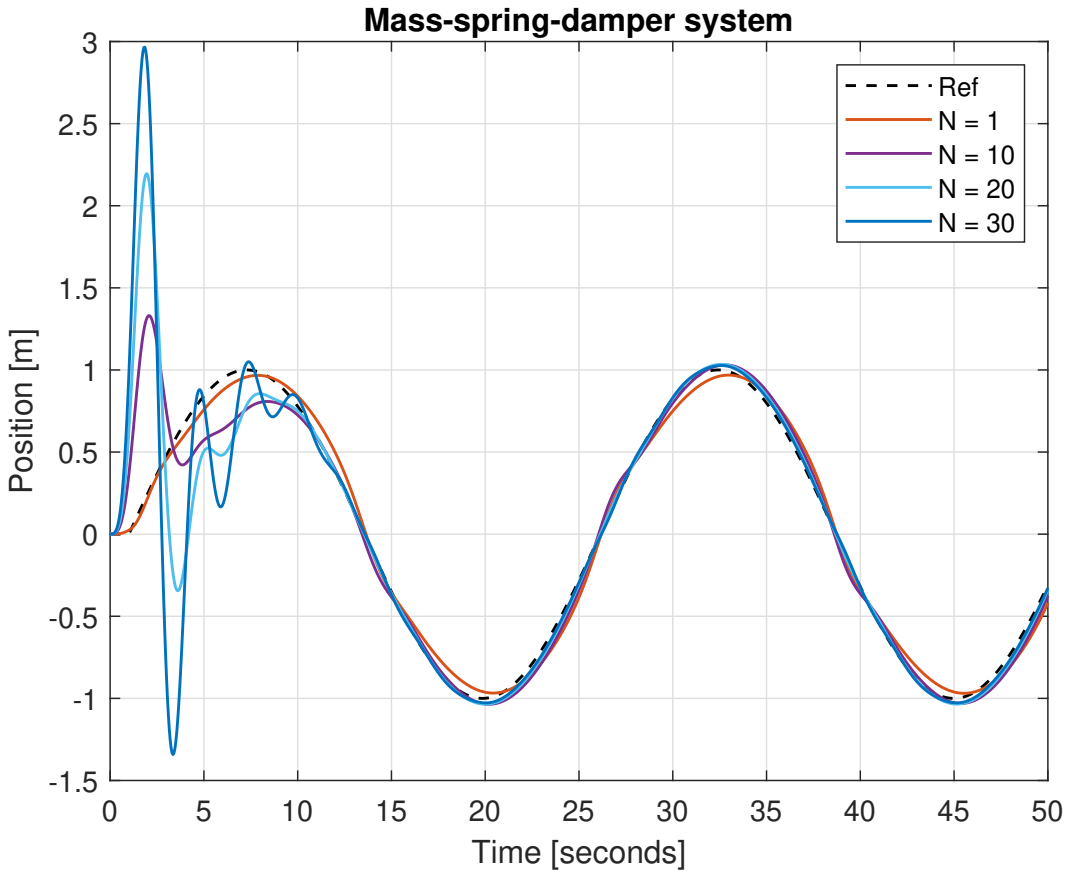

Figure 3.6: Sensitivity results with respect to the N parameter. With $N = 1$ (red line) the neural network does not have the learning capability to completely adapt to the system. Thus, the tracking is not perfect. Increasing N the tracking at steady state shows higher performances, however it takes more time to learn the optimal weights.

Sensitivity to the hyper-parameter K

This section shows the sensitivity analysis with respect to the parameter K, which represents the number of future samples that the network receives as input.

As shown in fig. 3.7, this parameter has an influence especially in the initial transient of the response. In fact, the bigger the value, the sooner the network will learn the right control action and thus the response will be closer to the reference. The intuitive idea behind this phenomenon is that, the more in advance the neural network knows, so the bigger is K, the faster it can compensate for the mismatch, especially at the beginning of the learning phase.

However, it is worth noting that a steady state the response is almost equal for all the K values, meaning that even a large future knowledge, very difficult to achieve, will not

generate a consistent improvement of the performances.

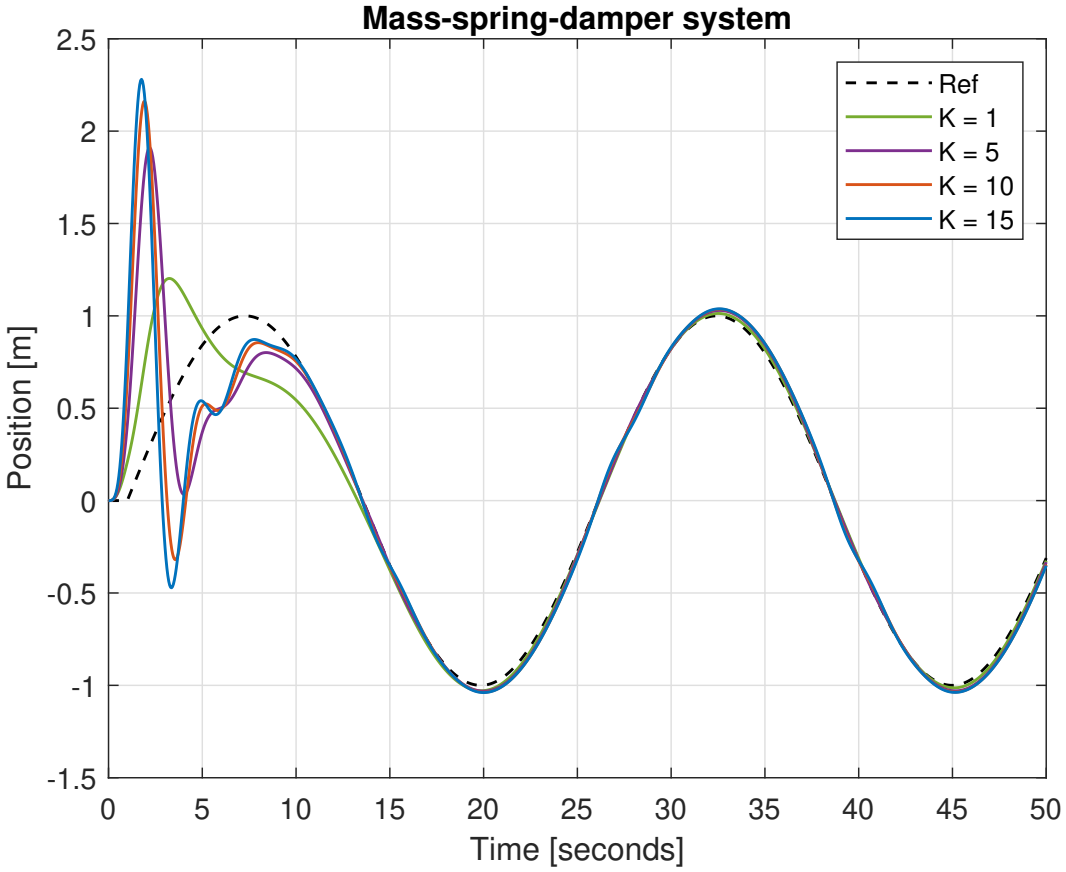

Figure 3.7: Sensitivity results with respect to the K parameter. With K = 1 (green line) the neural network takes more time to adapt to the system. Increasing K, so the more in advance the NN knows about the reference, the faster it can compensate for the initial mismatch at the beginning of the learning phase.

Simulation result PID+NN

This simulation aims to demonstrate the performance achieved with the PID+NN architecture during trajectory tracking of the sinusoidal signal $x^*(t) = \sin(0.25t)$.

As depicted in fig. 3.8, the reference trajectory exhibits nearly perfect tracking after an initial transient period of approximately 15 seconds. During this period, the network learns the correct control signal to follow the reference. It is noteworthy that, as anticipated, the contribution of the feedback controller (PID, shown in the second subplot as FB force) is significant in the initial stages, compensating for the initially inaccurate control signal from the feedforward controller (NN, illustrated in the third subplot as FF force). However, with the passage of time, the influence of the feedback controller

diminishes, and essentially, the entire contribution comes from the neural network. This indicates that the network has successfully learned the correct weights, as evidenced by the fact that the weights, initially set to a random value of 1, converge to a value near 0. The weights show residual oscillations centred around 0. The periodic oscillations are justified by the fact that the network is learning a "local" control action, over a short time horizon of length $K\Delta T$. Indeed, the weights are continuously adapted to optimize tracking.

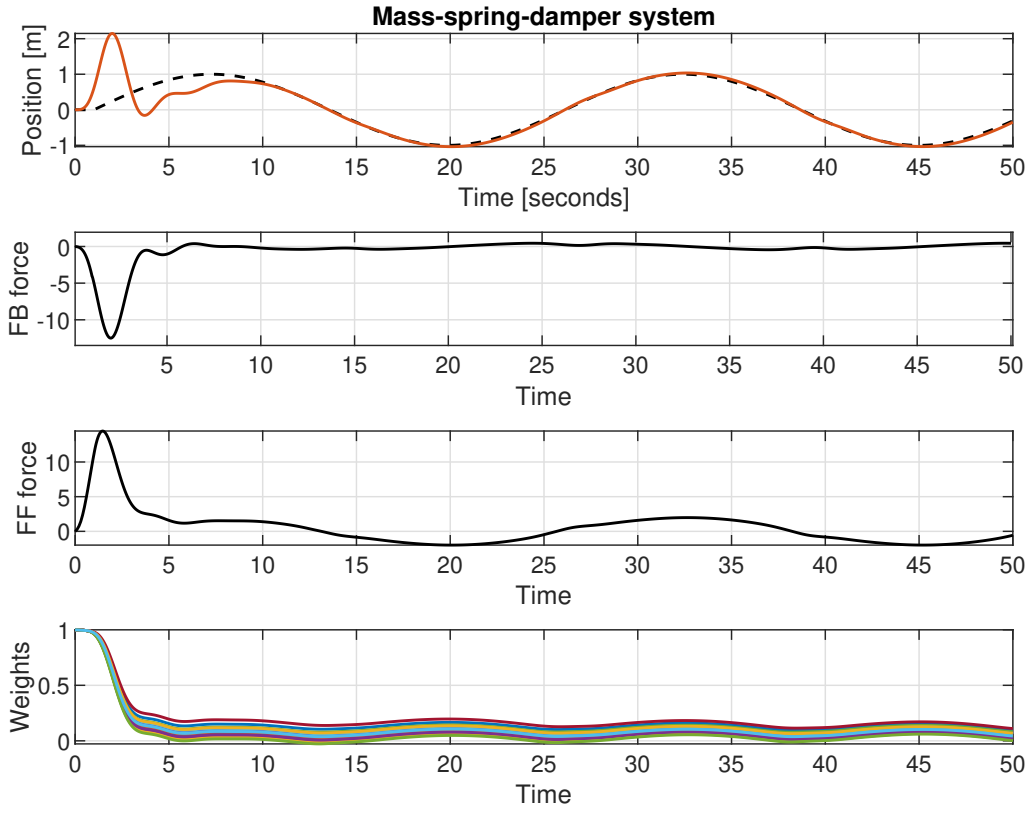

Figure 3.8: Simulation of the tracking performance with both feedforward and feedback signal. The first subplot shows the reference signal (black) vs the actual position (red). The second and third subplots show PID (FB Force) and NN (FF Force) control signals, respectively. The last subplot illustrate the trend of the weights of the NN.

Simulation result PID vs PID+NN

The goal of this simulation is to highlight the improvements in term of tracking performances of the PID+NN archictecture with respect to the solution with PID only.

The reference trajectory is represented by a sinusoidal signal $x^*(t) = \sin(0.25t)$. As illustrated in fig. 3.9, the outcome obtained with the PID-only controller is characterized by a delay, which is nearly completely eliminated when adding the feedforward term during learning steady-state operation.

The rationale behind this strategy is to prioritize long-term transient performance, acknowledging that the initial learning phase is an acceptable compromise considering the long-term benefits gained through the neural network's learning process. This not only aids in eliminating steady-state delays but also allows the system to adapt to and learn from evolving changes in dynamic behavior, enhancing the controller overall performance and adaptability.

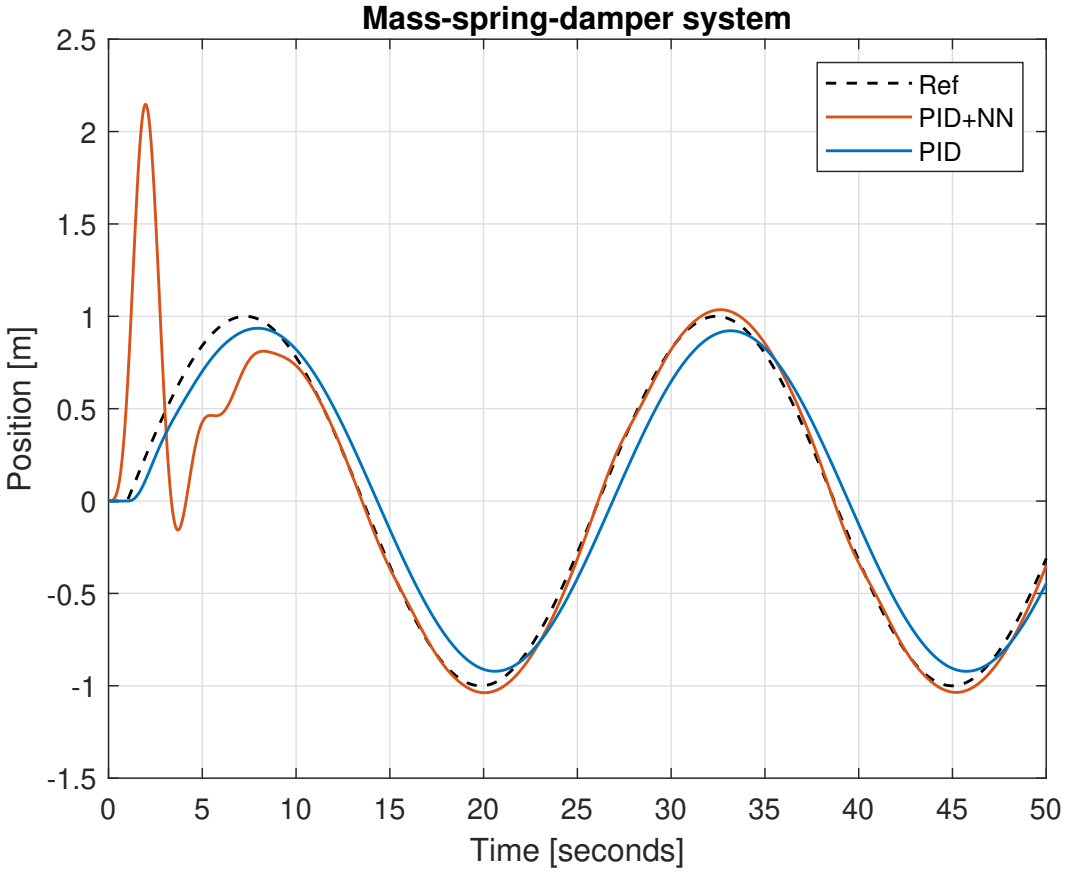

Figure 3.9: Simulated comparison between PID+NN and PID. The dashed line is the reference signal. The red and blue lines are related to the PID+NN and PID respectively. The PID+NN, after an initial assessment phase for learning, offers better tracking, with reduced error at steady state

4 | Experimental Setup

This chapter presents a description of the experimental setup. Specifically, it focuses on the design and fabrication of the soft robot, motivating the choices for material and structure parameters selection. It also shows an analysis of all the different technologies involved in the process, with their interconnections. The general scheme of the setup is shown in fig. 4.1. As it is possible to see it is composed of four main parts:

- Pneumatic Soft robot;
- Festo valves, which guarantee that the commanded pressure coming from the controller is actually injected in the robot;
- OptiTrack Motive system, which tracks the position of the robot tip with reflective markers;
- Controller, which generates the pressure needed to follow the reference trajectory $\mathbf{x}^*$, given the information of the actual position from the OptiTrack.

In the following, each of these fundamental elements will be analysed separately.

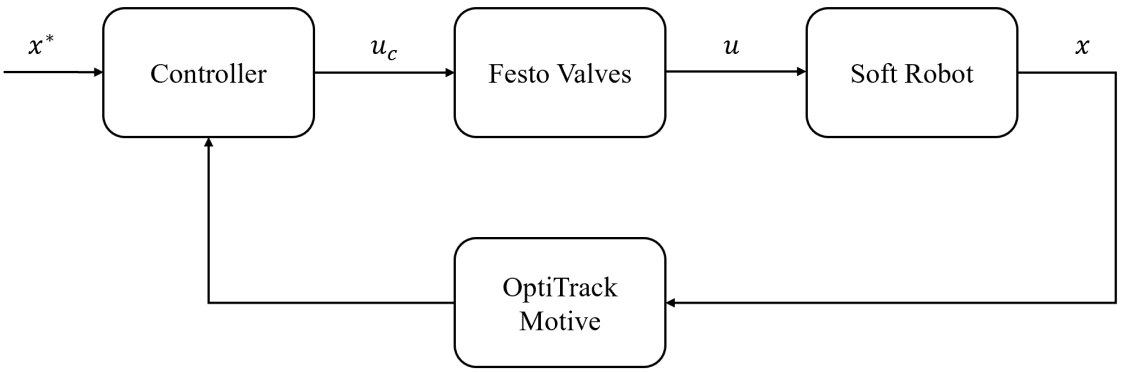

Figure 4.1: Experimental setup scheme. The Controller, given the reference trajectory $\mathbf{x}^*$ and the actual position from the OptiTrack Motive $\mathbf{x}$, computes the desired control action $\mathbf{u_c}$, in terms of pressure. Such commanded pressure will be injected in the soft robot thanks to Festo valves which deliver $\mathbf{u}$. The robot tip position $\mathbf{x}$ will change accordingly, and it will be tracked by the OptiTrack.

4.0.1. Design and fabrication of the pneumatic soft robot

The goal of this section is to provide a detailed analysis of how the soft robot has been designed and manufactured. The design parameters, derived from a literature search, have been optimized to guarantee the right trade off between softness and robustness of the robot. Moreover, the design is adapted to allow fast cleaning and supports removal, since it is fabricated via additive manufacturing.

As mentioned in Chapter 2, there is currently no established standard for both actuation technique and design of soft robots. However, pneumatic actuation, where a soft manipulator is driven by compressed air, is the most commonly used mechanism. Among the various designs, a growing focus is on those capable of omnidirectional movement, often leveraging symmetries or geometric properties.
In this context, the soft robot manufactured for this thesis is a pneumatically actuated soft manipulator with bellow shape design, inspired from [14], which guarantees multidirectional motion.

Soft Robot Design

The design proposed in this thesis is shown in fig. 4.2 and in fig. 4.3. In particular, fig. 4.2 presents a simulation of the achieved bending motion when one chamber is pressurized, while fig. 4.3 (a) shows a 3D render of the soft robot and fig. 4.3 (b) presents its top view (without the rigid blue connector). The design has been developed within the SolidWorks environment.

To achieve multidirectional movement the soft robot is composed of three parallel-connected chambers rotated by 120° about the longitudinal axis of the actuator. Each chamber has an internal cavity to let the air flow in it. When one chamber is pressurized, it bends along the direction in between the other two (see fig. 3.2 for a description of the bending directions).
Since all the chambers are connected together in the central axis of the robot (fig. 4.3 (b)), a deformation in one of them will generate a deformation for the whole soft actuator. The following fig. 4.2 shows a simulation of the bending of the soft robot when just one chamber is actuated, in this case the one to the right. As shown, all the 3 chambers deform coherently creating a bending movement for the whole robot. By selectively pressurize the chambers it is possible to generate a movement of the tip of the robot, in this case represented by the central point of the bottom of the robot, in all the directions of the plane.

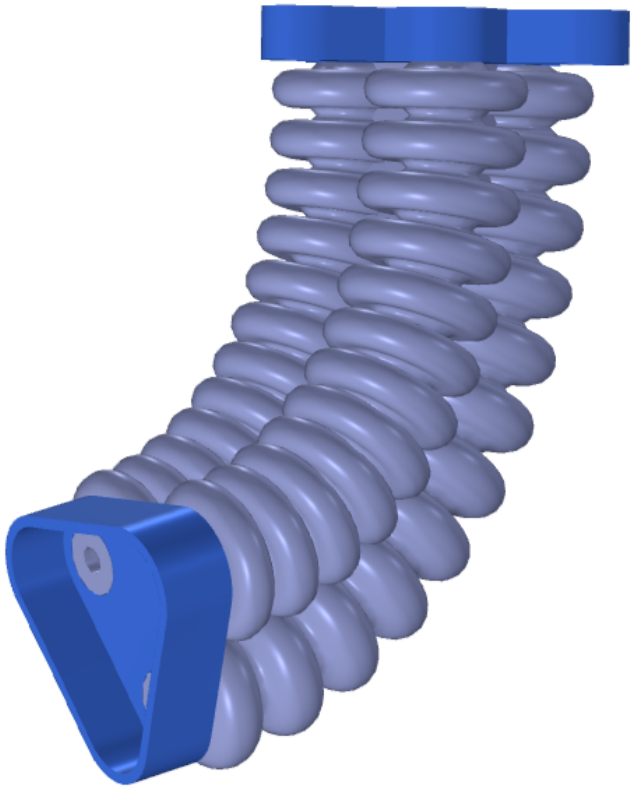

Figure 4.2: Bending movement simulation of the soft robot when just one chamber is actuated (right chamber).

At the top, the soft robot is characterized by a rigid element (dark blue in fig. 4.3 (a)) which allows the connection by means of screws to a fixturing device to place the actuator in the field of view of the OptiTrack cameras (see fig. 4.6 top white element). For the bottom part, instead, the rigid blue component is designed to guarantee the placement of the OptiTrack markers. The centre of this bottom element is considered as robot tip.
The bending motion is generated by the bellow shape design, which enables the chamber to deform without elongation once being pressurized. The chambers need to be properly sealed to avoid any air leakage which will compromise the motion accuracy. Air is injected within the soft robot by means of thin and soft tubes. For this reason, it is necessary to create holes of the same diameter of the pipes to allow the tube insertion in the top layer of the soft robot fig. 4.3 (b). With such design, the actuator is able thus to bend, but it cannot extend, because an elongation will create a deformation in the bellow design which may cause damages to the whole structure.

(a) (b)

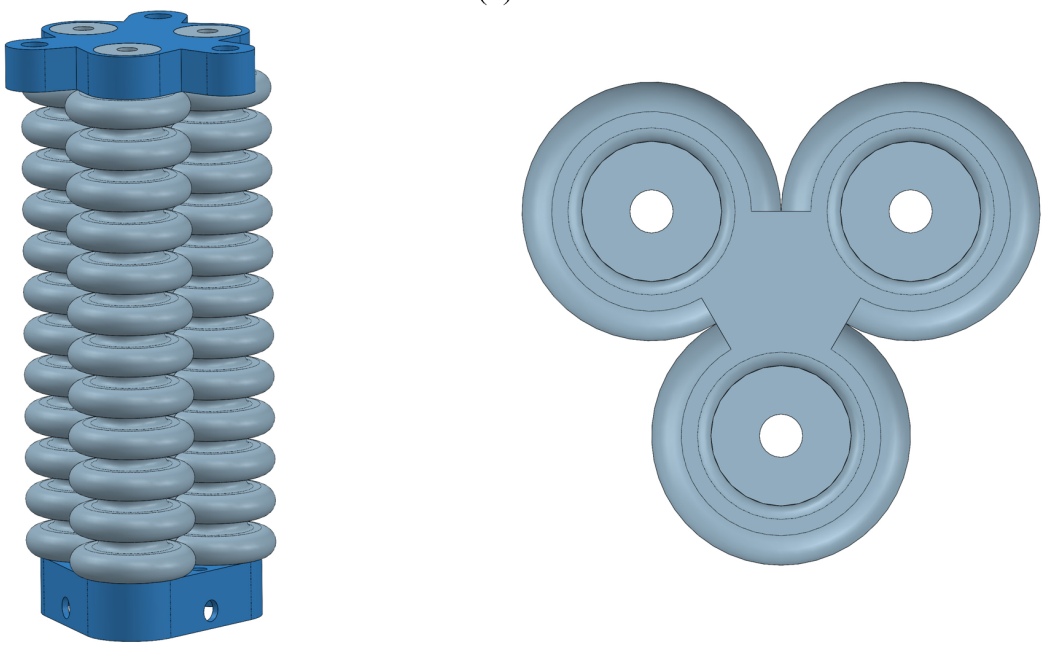

Figure 4.3: (a) 3D CAD render of the soft robot. It highlights the 3 connected chambers with bellow shape design. The top and bottom rigid elements (dark blue) are used for fix the robot in the OptiTrack frame and attach the reflective markers needed by the OptiTrack cameras. (b) Top view of the soft robot. The 3 chambers are connected in their central part. The holes allows the piping for compressed air flow.

Although the bellow shape design is fundamental for motion, it may also create problems related to durability. In fact, the continue local deformation in the bellow region, together with an inherently soft material, can generate cracks in the structure. For this reason, it is important to optimize the bellow parameters to find the right trade off between bending deformation and strength of the soft robot.
In particular, to fast change the bellow parameters, each one is parametrized by its symmetric lines. As shown in fig. 4.4 essentially, the geometry of the bellow-SPA (soft pneumatic actuator) module is governed by four parameters r_{in}, R, t, l, where r_{in} is the radius at the input of the bellow, R is the average radius $R = \frac{r_{in}+r_{out}}{2}$, t is the layer thickness and l is the vertical length of the bellow.
The corresponding values used for the soft robot are: $r_{in} = 5.75mm, R = 8.5mm, t = 1.5mm, l = 8mm$ and overall each chamber is composed of 12 bellows. These values have been selected based both on studies in the literature [14, 36] and on empirical considerations. For example, by printing actuators with a lower layer thickness it was clear that the risk of leakage was too high because even with very small bending angles the material starts cracking. On the contrary, with higher layer thickness, the bending angle

was compromised because the bellow did not have enough room to expand.

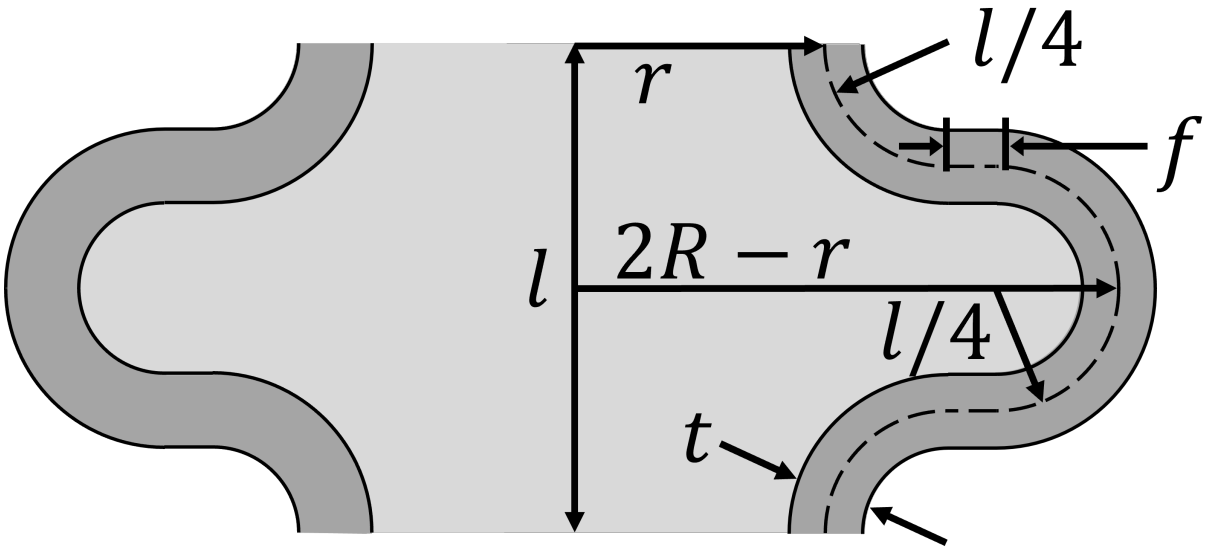

Figure 4.4: Single bellow module with relevant parameters.

Soft Robot Fabrication

The soft arm is fabricated with a multi-material 3D printer (J735, Stratasys Ltd, USA), where multiple droplets of uncured material are deposited, blended and cured by UV light. The main body of the actuator is printed with Agilus30 (a soft and rubber-like material with a quoted tensile strength of 2.1-2.6MPa and a Shore hardness of 30A, according to the Stratasys datasheet). Stratasys, like the majority of additive manufacturing companies, restricts the use of only predefined materials for printing. Therefore, the material for the soft robot was selected from a limited batch of options.

Despite this limitation, the chosen material offers an excellent balance between softness and smoothness of movement. Experiments with materials of higher Shore hardness resulted in a less smooth movement, requiring a greater variation in input pressure to generate the same motion. However, the Agilus30 has the drawback to be sensible to light, resulting in a fast degradation of its properties. This can imply the formation of cracks on the surface which can cause leakages, preventing the robot from moving correctly. For this reason, a controller which can online adapt to the robot features is essential.

The rigid blue elements of the robot are printed with Vero (a rigid plastic-like material with a quoted tensile strength of 50-65MPa and a Shore hardness of 83-86D).

The multimaterial 3-D printer fills the internal volume of the actuator chambers with support material during the printing process to prevent parts from collapsing that have

large overhanging features. These support materials are difficult to be removed, thus the idea was to 3-D print rods along the length of the internal cavity of the chambers of the bellows in place of the support material (see fig. 4.5).

The 3-D printed rods are removed after the printing process has completed, which makes the cleaning process much quicker. However, some residual supports can always be present and thus, to completely remove them, the soft arm is post-processed with chemical bathing in a tank (GEMINI SSR-550) filled with support removal solution (0.02 kg/L Sodium Hydroxide and 0.01 kg/L Sodium Metasilicate).

The net weight of the actuator is around 62g.

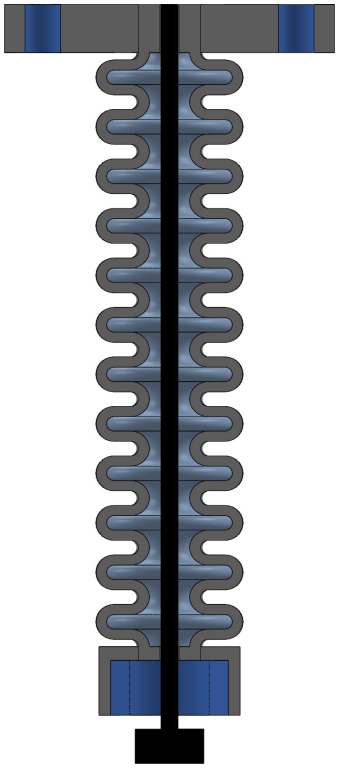

Figure 4.5: Single chamber vertical cross section with internal rod. The rod allows the printed to avoid placing internal supports difficult to be removed. After the printing process, the rod is taken out from the robot and the bottom cavities are sealed.

4.0.2. OptiTrack Motive

This section describes the OptiTrack Motive system, used for tracking in real time the position of the tip of the robot.

The OptiTrack Motive system is a motion capture solution designed for a wide range of applications, including animation, biomechanics, virtual reality, and robotics. OptiTrack Motive enables users to capture and analyze the movement of objects or subjects in three-dimensional space. It achieves exceptional accuracy by utilizing high-resolution cameras, allowing for sub-millimeter tracking precision. This level of detail is particularly essential for applications requiring precise movement capture.

The system relies on reflective markers which are typically small spheres or discs that reflect infrared light emitted by the cameras, enabling real-time calculation of their precise 3D positions (see fig. 4.6). They need to be strategically placed on the object to be tracked so that they can be seen by the cameras. During the recording of movements, cameras continue to capture images of the markers in real-time. The Motive software processes these images, identifies the markers, and calculates their three-dimensional positions based on the previously acquired calibration information. Before motion data recording begins, accurate calibration of the system is necessary. This process involves recording reference images of the markers from different angles. During calibration, the system determines the exact 3D positions of each marker.

The cameras utilized in the experimental setup are OptiTrack Flex 3 Motion Capture cameras. A total of six reflective markers were strategically positioned for tracking purposes. Specifically, three markers were affixed to the top rigid element of the soft robot, while the remaining three were placed on the bottom part. An advantageous feature of the OptiTrack system is its capability to not only capture the position of individual markers but also determine the position of a "rigid body" created by a set of three markers.

In the setup, the top three markers correspond to the rigid body located at the base of the robot, while the bottom three markers represent the rigid body at the end of the soft actuator. The OptiTrack system allows for the retrieval of the Cartesian position of the center of the "end rigid body", which corresponds to the tip of the soft actuator.
This position information is then relayed from the OptiTrack system to the controller, enabling accurate tracking and control of the soft robot's movements. The OptiTrack sends the online position with a rate of 100Hz.

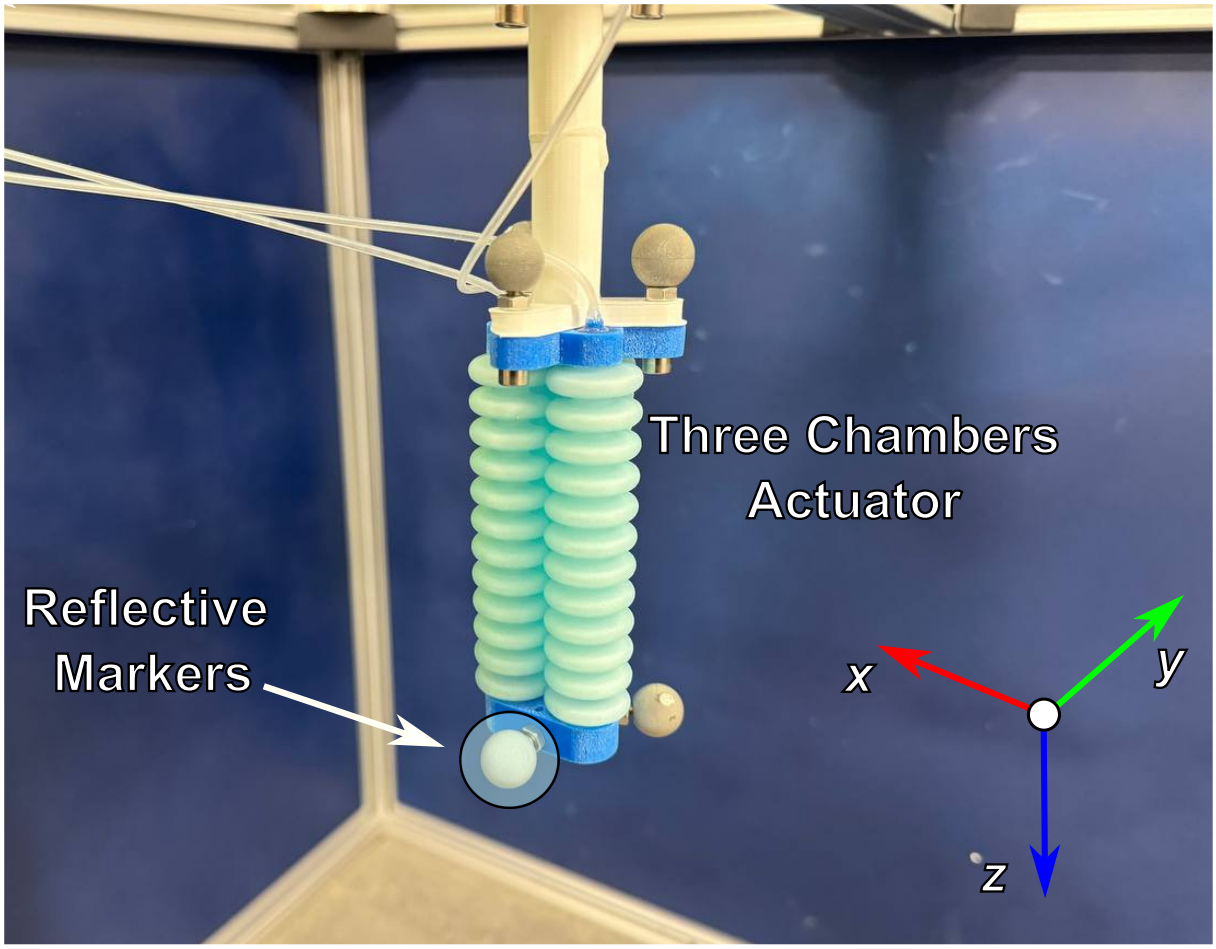

Figure 4.6: Experimental setup. A cylindric actuator with 3 chambers is connected to a rigid frame (white top element). The actuator is controlled using compressed air regulated through three Festo proportional regulators. The OptiTrack system is used to capture the position of reflective markers and to compute the position of the actuator tip. The reference frame is shown in the bottom right part of the figure.

4.0.3. Festo Valves

This section provides an analysis of the behaviour of the Festo valves, which are used to guarantee that the commanded pressure from the controller ($\mathbf{u_c}$ in fig. 4.1) is actually sent to the robot chambers.

The experimental setup is equipped with three Festo valves, specifically the VEAA-L-3-D2-Q4-V1-1R1 model, capable of accommodating a maximum pressure of 200 kPa and a minimum pressure of 0.7 kPa, with a resolution of 0.1 kPa. It's important to note that these valves do not support negative pressures. To ensure the safety of the robot, mitigating the risk of cracks or air leakages, the pressure is typically maintained within a maximum value of 25 kPa. The input ports of the valves are connected with pipes to an air compressor system which is able to generate up to 500 [kPa] of compressed air. Similarly, tubes link the valves output to the soft robot.

The valves are regulated by an internal proportional controller, ensuring precise tracking of the input pressure from the robot controller. The internal proportional regulator is equipped with an integrated sensor that measures the actual pressure in the outlet part of the valve, facilitating effective and responsive pressure control. The Festo controller works on a Teensy 4.1 USB Development board. In fact, by means of a USB cable such controller is connected to the computer where it can be integrated within the whole experimental setup. Figure 4.7 shows in detail the block scheme of the Festo valves, with their integrated controller and sensor.

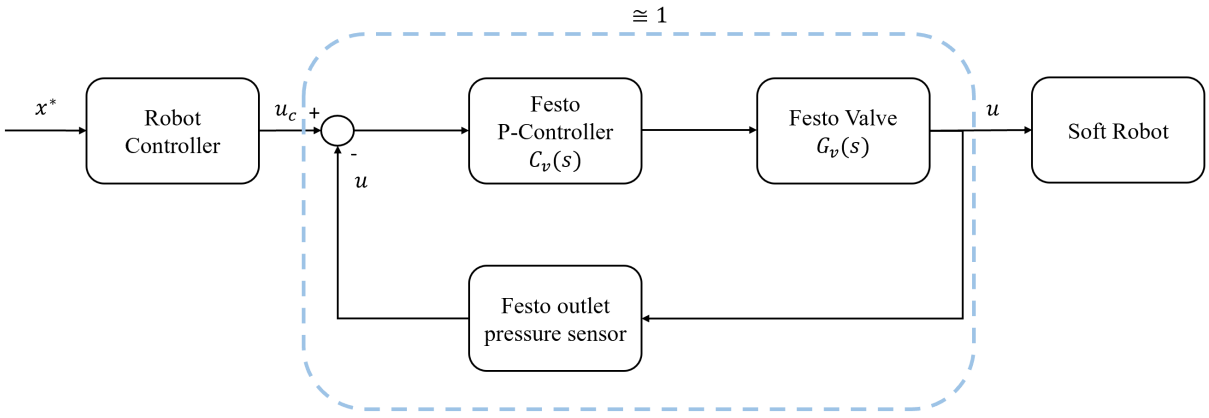

Figure 4.7: Block scheme of the components of the Festo valves. The Festo P-Controller guarantees that the outlet pressure from the valves to the robot ($\mathbf{u}$) is actually equal to the commanded pressure from the robot controller, thanks to the feedback measurement given by the internal Festo pressure sensor. For this reason, it is possible to approximate the entire inner feedback loop as a transfer function equal to 1.

Experiments have been performed to test the performance of the internal proportional controller and thus to find the relationship between $\mathbf{u_c}$, the control input from the robot controller and $\mathbf{u}$, the actual control action to the soft robot. Since the 3 valves are exactly identical, the tests focus just on one of them. The experiments are performed in openloop, meaning that the control action $\mathbf{u_c}$ is generated by an external specific code and thus it is not function of the error between reference position and actual position coming from the OptiTrack feedback. The Festo proportional controller works at a rate of 50Hz.

Experiment: Step response

The first experiment consists in tracking a step of 20 [kPa]. In one case, the step is given without considering a preload of the valves, while in the second case a preload of 2kPa is added to the system. Hence, for the second case the step is between 2kPa and 22kPa. As shown in the figure below (see fig. 4.8), in both cases the Festo proportional controller is able to track the reference pressure very fast, with a small overshoot. However, for the first case (fig. 4.8 (a)), the response is characterized by a delay.

This phenomenon is due to the fact that, without preload, the inlet pipes are completely empty. Thus, before actually deliver pressure - that is compressed air - to the valves, such tubes need to be completely filled. Hence, the delay is the time needed by the air compressor system to fill the inlet tubes of the valve. In the second case, in fact, since the initial pressure is fixed to a certain preload (in this case 2 kPa), the inlet tubes are already full and thus, as soon as the reference pressure changes, the Festo proportional controller is able to change and regulate the response without delay (fig. 4.8 (b)).

The preload value depends on the length of the tubes, so in this specific case 2 [kPa] was enough to guarantee the suppression of the delay. For the experiments performed on the soft robot, instead, it was chosen a preload of 7 [kPa], according to the dimension of the pipes. It is worth noting that, due to the symmetries present in the soft robot design, a preload pressure of 7 [kPa] in all the three chambers, will not generate a movement of the robot tip, since the different bending directions will compensate with each others.
This guarantees that the robot starts its motion from a straight resting position, as shown in fig. 4.6.

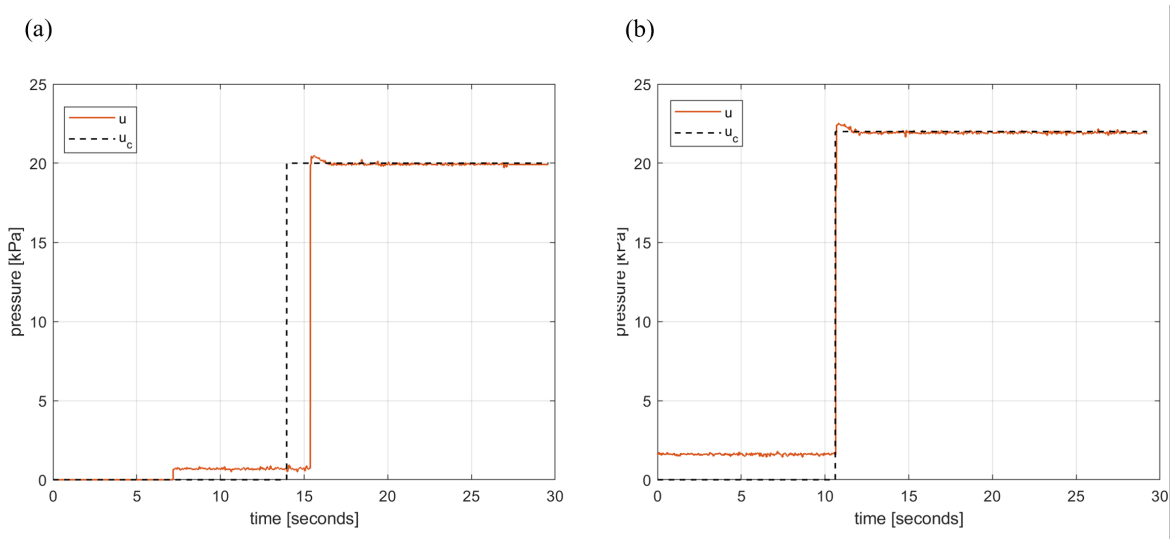

Figure 4.8: Comparison between outlet pressure without (a) or with (b) preload on the valves. The black dashed line is the commanded pressure, the red line is the actual pressure.

Experiment: Sine wave response

In a second experiment, the input is a pressure sinusoid, in order to check whether or not the Festo proportional controller is able to track this waveform.
This is relevant since the soft robot is mainly actuated for following a sine reference, so the valves need to guarantee reliable performance in this context. In particular, the test is characterized by an input sine wave defined as $u_c = 10\sin(t) + 12$. The results of the experiment are shown in fig. 4.9.

From the figure, it is clear that the Festo proportional controller is actually able to perfectly track the sine wave, always considering a preload of 2 [kPa].
However, it is worth noting that the actual pressure around 2 [kPa] is characterized by noise which has a negative effect on the tracking. In addition to that, both the peaks and the valley of the sine wave present small overshoots. This is coherent with what observed for the step reference in the previous experiment.
Another possible cause of this phenomenon could be the resolution of the valve which does not allow for such small and sudden changes in the pressure.
However, the overall performance is totally acceptable and thus the conclusion is that it is possible to rely on the Festo proportional controller also for sinusoidal pressure references.

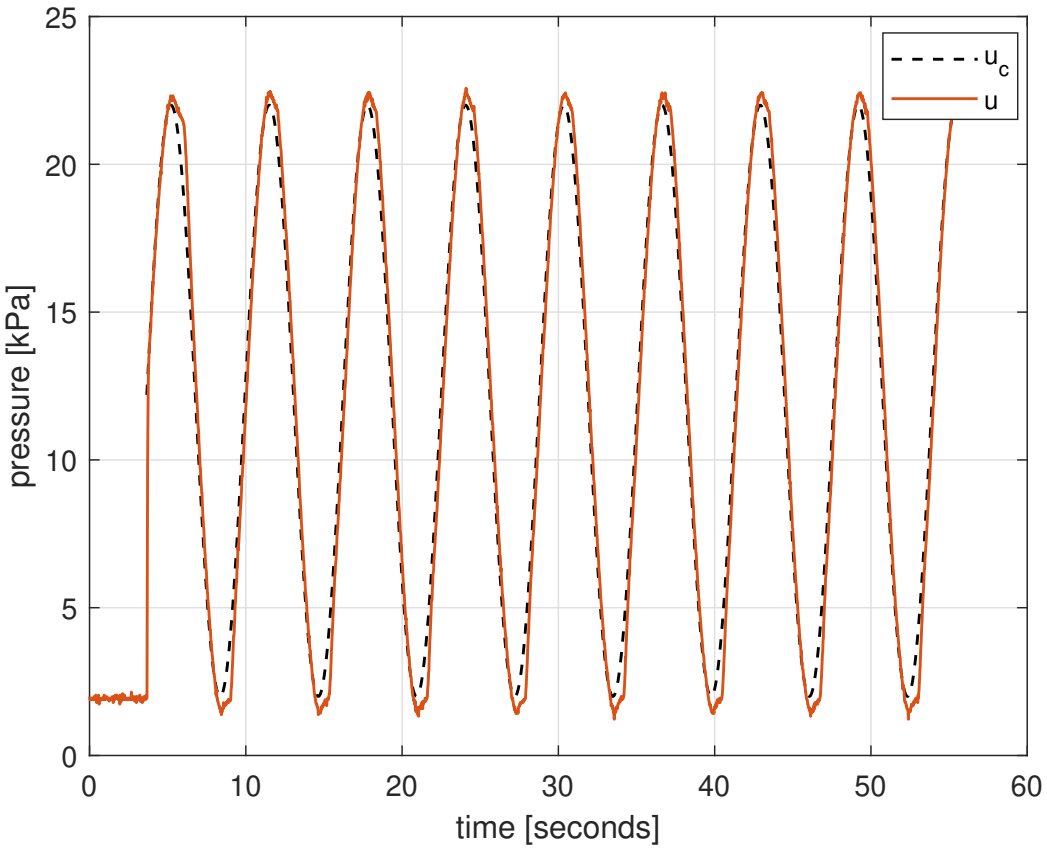

Figure 4.9: Sinusoidal tracking of the commanded pressure in openloop (no robot controller). The black dashed line is the reference pressure $u_c = 10\sin(t) + 12$, the red line is the actual pressure.

Sine wave with added payload

Finally, a third experiment is performed with the openloop configuration with a pressure reference defined as $u_c = 15\sin(5t) + 17$. The reference, in this case, is characterized by a faster frequency and bigger amplitude with respect to the previous one. In fact, the idea is to find what are the maximum amplitude and frequency which can be reached by the controller, with sufficient accuracy.

Moreover, for this experiment, the robot is characterized by an added payload which is fixed on its tip. This additional dynamic effect has the goal of further testing how the Festo controller reacts and of checking its limits. Moreover, even the robustness of the soft arm is analysed, since it is subjected to an external weight.

For this case the OptiTrack measure is added to the system, always without closing the loop with the Robot Controller. Thanks to that, the behaviour of the soft robot is analysed, checking for hysteresis effects or defects in the robot fabrication.

The behaviour of the Festo proportional controller with respect to the faster sine wave is shown in fig. 4.10.
Although the tracking is still acceptable, it is clear that the higher frequency negatively affects the performance. In fact, in this case the response is characterized by a delay and a mismatch for lower values of the reference. This can be connected again to the noise which characterizes the valves for low pressure values.
Given this result, it is possible to claim that for frequencies below 5 [rad/sec] can be assumed perfect tracking, while for frequencies higher than that, this assumption does not work anymore.

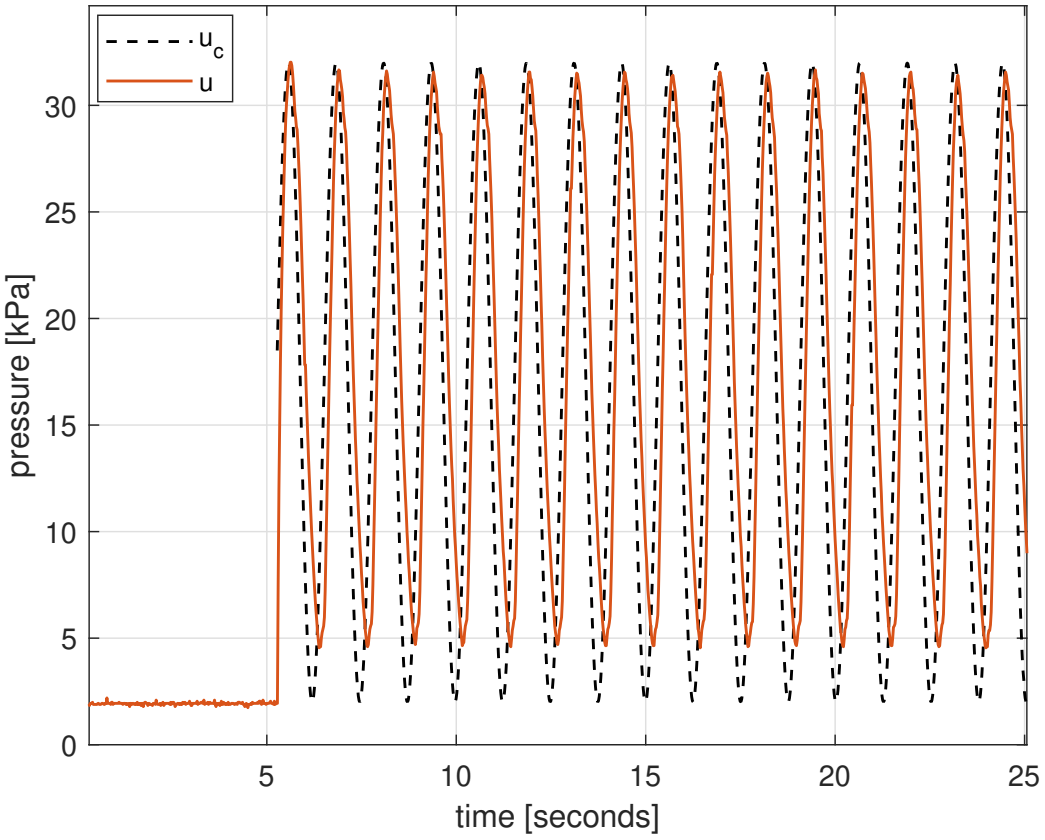

Figure 4.10: Sinusoidal tracking of the commanded pressure in openloop (no robot controller). The black dashed line is the reference pressure $u_c = 15 \sin 5t + 17$, the red line is the actual pressure.

The OptiTrack measurements related to the experiment are shown in fig. 4.11. In particular, the figure shows the 3D movement of the tip of the soft robot in the Cartesian plane.

It is clear that, even with fast oscillations, the behaviour of the robot is smooth and

repeatable. In fact, all the lines which represent the movement of the tip eventually converge on the same pattern. However, there are some hysteresis effects, especially on the z axis, meaning that the robot is going upwards (from right to left) along a different line with respect to the one which follows for coming back (from left to right). The magnitude of these effects is very small and potential causes may be the mismatch between commanded pressure and actual pressure, the nonlinearities of the material, small fabrication differences among chambers and the added weight.
Overall, from this experiment it is clear that the robot is reliable from the manufacturing point of view since, even with a payload attached on it, is able to guarantee a smooth and constant movement shape.

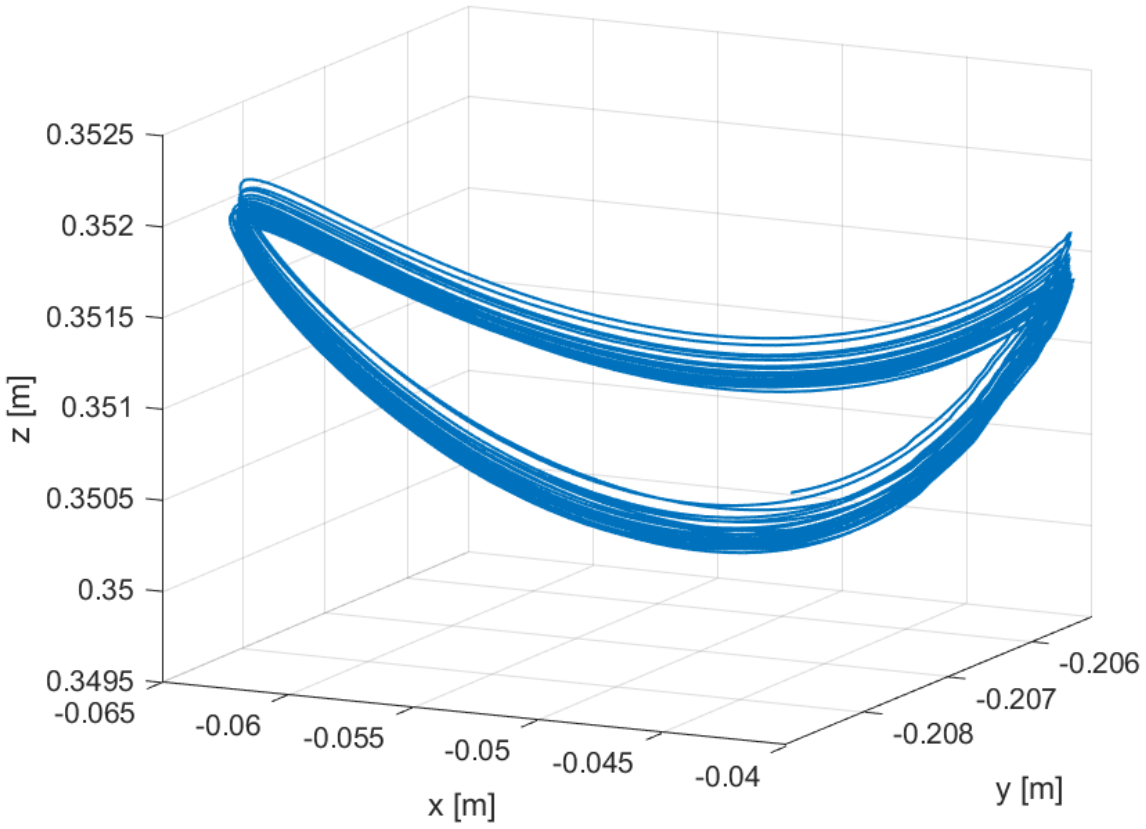

Figure 4.11: Position of the tip of the robot in the space.

In conclusion, from the above experiments it is possible to assume that, for low frequency, the control input coming from the Robot Controller $\mathbf{u_c}$ is exactly equal to the actual input that will act on the physical robot $\mathbf{u}$, thanks to the action of the Festo proportional controller. Given this, from now on the inner control loop of Festo controller and its transfer function will be considered equal to 1, as represented by the blue box in fig. 4.7.

4.0.4. Controller - ROS2 Environment

The goal of this section is to explain how the controller of the soft robot is integrated within the whole experimental setup. For this reason, the focus is on the software integration and not in the controller development (which has been discussed in Chapter 3).

The whole architecture runs in a ROS2 environment which is an open-source middleware framework designed for the development of robotic software. The core concept of ROS2 revolves around the use of Nodes and Topics, thus creating a software which can be represented as a graph of connected elements.
A node is a computational entity that performs a specific task or function within a robotic system. Nodes are typically designed to be modular and independent, allowing for a more distributed architecture. Each node can be written in a different programming language, promoting language flexibility within a robotic system. Nodes can be publisher, subscriber or both, where the first one means that the node sends a message (publish) to the other nodes, while the second means that the node receive a message from the others (subscriber). Nodes communicate with each other by publishing and subscribing to topics. This communication enables the exchange of data and coordination of activities between different components of the robot.
Hence, topics are named communication channels through which nodes exchange messages. A topic follows a publish-subscribe model, allowing nodes to communicate without needing to know each other's identity or location. A node can publish messages to a topic, and one or more nodes can subscribe to the same topic to receive those messages. This decoupling of publishers and subscribers enhances the modularity of the system.

The ROS2 architecture for the experimental setup is composed of 4 nodes. A node respectively for the OptiTrack system and for the Festo proportional regulator. The robot controller is instead characterized by 2 nodes: one for the actual control algorithm and one for the generation of the reference trajectory. The integration of all the technologies and algorithms in the same environment allows the syncronisation of the activities for a better post processing and debugging phase.

A description of the connections among the different nodes is shown in fig. 4.12, through an UML diagram. The core is represented by the class ROS2, which manages all the different nodes and topics within an Ubuntu computer, at a rate of 50Hz. As shown in the figure, the Reference Node has the goal of generating the reference trajectory which will be published on the Reference topic. Similarly, the OptiTrack Node will generate the actual position of the tip of the soft robot, based on the reading of the markers placed on

it (see section 4.0.2). The actual position will be publish on the OptiTrack topic.
The Control Node is the heart of the architecture since it will subscribe both to the Reference and the OptiTrack topic, getting in this way the corresponding messages. Furthermore, it will generate the desired control action (based on the algorithms described in Chapter 3) which will be published on the Festo control topic. Finally, the Festo Node will subscribe to that topic, thus reading the desired control action and, thanks to its internal proportional controller, it will generate the corresponding pressure in the outlet streams of the valves. Such pressure will enter in the robot chambers, modifying its position accordingly.

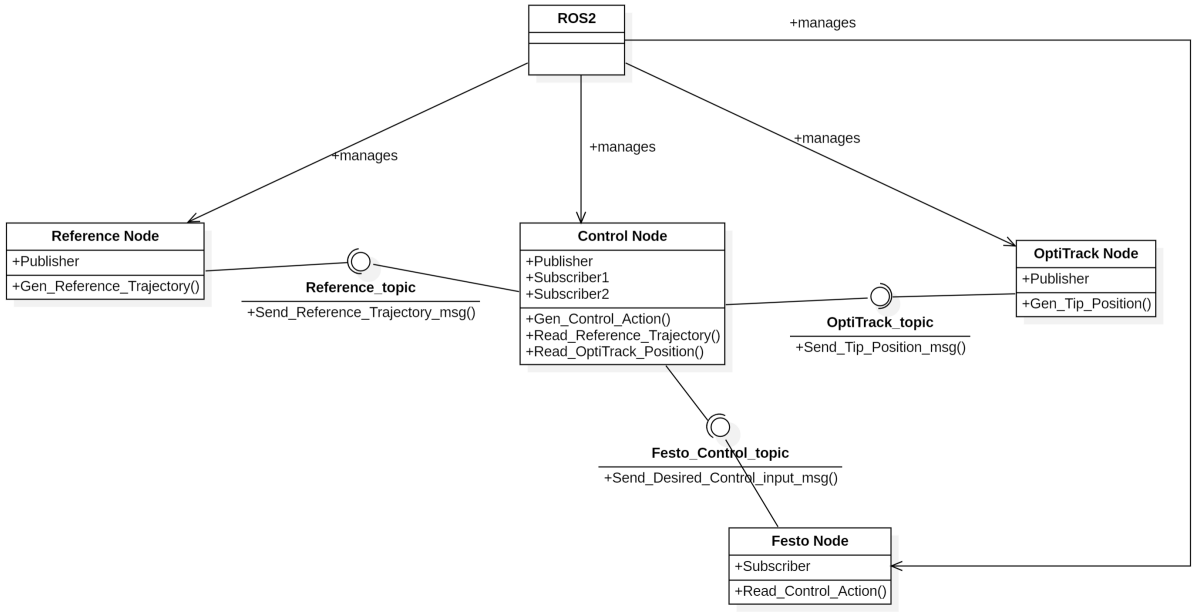

Figure 4.12: UML diagram describing the interaction among the different modules. All of them are within the ROS2 environment. The Control Node takes the reference trajectory from the Reference Node and the actual tip position from the OptiTrack Node. It computes the desired control action which will be published on the Festo control topic and read by the Festo Node. The valves will inject in the robot the actual corresponding pressure.

5 | Experimental Results

This section contains the experimental results of the Feedback Error Learning architecture applied on the experimental setup described above. It starts from the tuning of the PID and NN parameters, and then showing the improvements of the combination PID+NN in terms of trajectory tracking. Finally, it is presented an analysis of the performance of the soft robot when subjected to changes in its dynamics due to the addition/removal of payloads. This last experimental result shows the capacity of fast online adaptation provided by the control algorithm.

5.0.1. PID tuning

As discussed in Chapter 3, the feedback controller must provide good tracking performance to properly train the NN. Therefore, the first step is to tune the PID gains which are based on the definition of the $\boldsymbol{H}$ matrix, linking the PID actions to the actuator space, i.e. $\mathbf{u_{fb}}$ (see Section 3.2). It is worth noting that, although the PID is based on a simplified model of the robot, it can guarantee good performance, which will be further enhanced by the NN contribution.
In the tuning phase, the response of the system using solely the feedback signal is considered, to avoid any interference given by a neural network not properly trained. The soft robot designed for this thesis is not able to extend, as discussed in section 4.0.1, hence the PID controller on the z-axis is fixed at 0 ($K_{pz} = K_{iz} = K_{dz} = 0$) and the robot controller works on the x-y plane. However, as said in Section 3.2 the proposed methodology can be easily applied to stretchable soft manipulators where the z contribution is considered.

The proposed PID controller does not rely on an analytical description of the transfer function of the soft robot, thus it is not possible to use classical tuning techniques such as bode diagrams. For this reason, the PID is tuned empirically, by initially keeping $\boldsymbol{K_I} = \boldsymbol{K_D} = \mathbf{0}$ and increasing the proportional gain $\boldsymbol{K_P}$ gradually and stopping before the system starts oscillating. Where $\boldsymbol{K_P}, \boldsymbol{K_I}, \boldsymbol{K_D}$ are the one defined in eq. (3.44). Then, the integral gain is added to further reduce the tracking error. Finally, the derivative gain is introduced to reduce the overshoot and to make the controller more reactive.

The PID is tuned on a reference sinusoidal signal oscillating along the y-axis (see fig. 4.6) and centred at 0.203m, being the resting position of the actuator on the y-axis (i.e. no pressure applied). Since the reference is along the y-axis only, the tuned gains will be K_{py}, K_{iy}, K_{dy}. Using symmetry considerations, it is reasonable to say that the gains for the x-axis can be considered equal to the ones on the y-axis (i.e. $K_{px} = K_{py}, K_{ix} = K_{iy}, K_{dx} = K_{dy}$).
The reference trajectory is defined as:

$$\mathbf{x}^* = \begin{bmatrix} \cdot \\ 0.03\sin(0.25t) + 0.203 \\ \cdot \end{bmatrix} \tag{5.1}$$

where $\cdot$ represents an arbitrary value of low interest since the tracking is only performed on the y-axis.

Kp Sensitivity

In the following, experimental results show the sensitivity with respect to PID parameters. In particular, fig. 5.1 highlights the sensitivity with respect to the K_{py} value. The shown response is characterized by a PID with fixed values for K_{iy} and K_{dy}, while K_{py} is changed to show how it influences the performance.

With $K_{py} = 1000$ (red line) the trajectory tracking is accurate and it does not present oscillations. However, the response is clearly characterized by a delay.
Doubling the value of K_{py} (purple line), the response is negatively affected by oscillations and, moreover, the delay is still present. The oscillations may cause damages to the fragile structure of the robot and thus they need to be avoided.
Halving the K_{py} value to 500 (light blue line) the response is almost overlapped with the previous one (red line), a part from a slight mismatch in the top and bottom parts.
Given these performance, the $K_{py} = 500$ value is selected, since it allows a good tracking minimizing the risk of oscillations, dangerous for the soft robot.

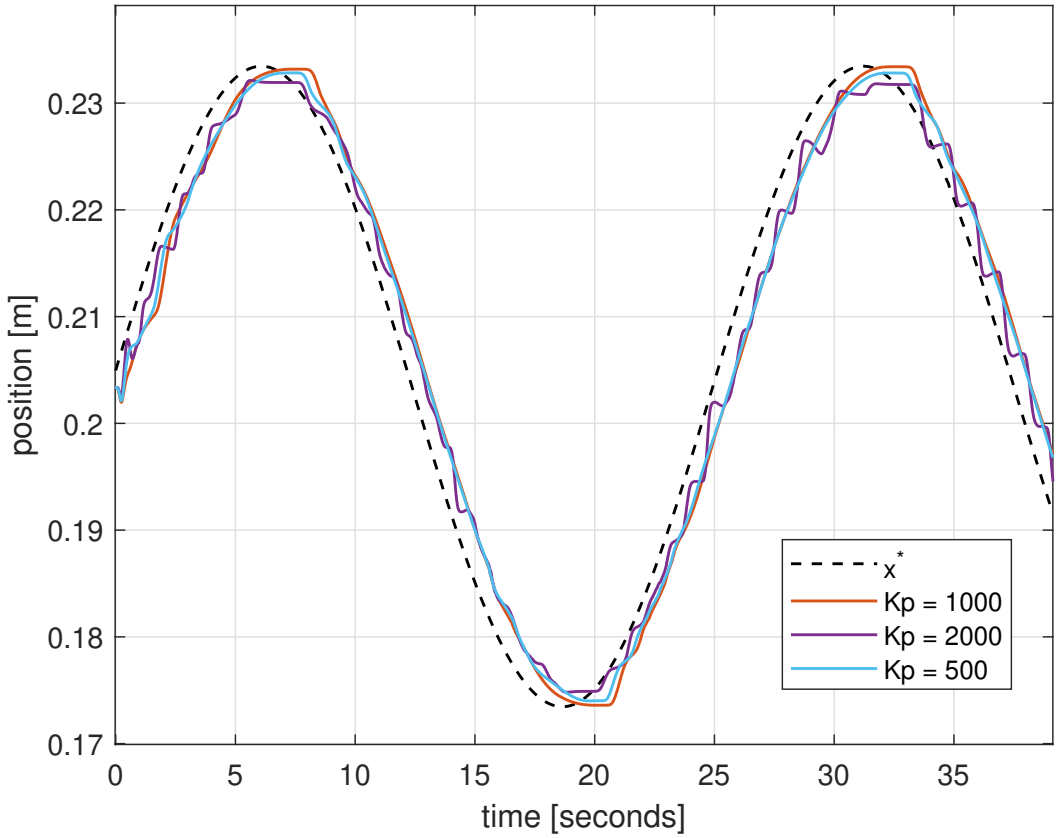

Figure 5.1: Sensitivity with respect to the Kp parameter. The figure represents the response with the PID-only controller. The black dashed line is the reference trajectory. Increasing the Kp the behaviour is more oscillatory, dangerous for the robot structure. The value of Kp = 500 is chosen because it guarantees smooth response, even if characterized by a lag.

Ki and Kd Sensitivity

This paragraph shows experimental results for the tuning and thus sensitivity of the K_{iy} value. Moreover, an analysis of the K_{dy} parameter is presented. The experiments are again based on the sine wave reference in eq. (5.1). Now the value of K_{py} is kept fixed at 500, as found from the previous experiment.

The idea is to add the integral action to reduce the delay in the response. As show in fig. 5.2, with a value of $K_{iy} = 700$, although the performance are satisfactory, they are always characterized by a delay in the response (red line).
By doubling the K_{iy} value at 1400 (purple line), the delay is actually reduced as expected, even if it is always partially present. However, in this case, the behaviour of the robot is characterized by a slight overshoot and especially by oscillations, dangerous for the robot

structure.
A last experiment consists in reducing the K_{iy} parameter at 350. As is it possible to observe, the response now is smoother, but the delay is amplified.

Given the above considerations, the value of $K_{iy} = 700$ is chosen, since it guarantees the right trade off between smoothness (avoiding oscillations) and delay of the response.
Concerning instead the derivative gain K_{dy}, its value is fixed at $K_{dy} = 40$, because even a small increase of it generates oscillations which can affect the robot robustness. Anyway, it was chosen to add the derivative parameter to improve the performance, trying to reduce the steady state error, even if its contribution is minimal with respect to the proportional and integral gains.

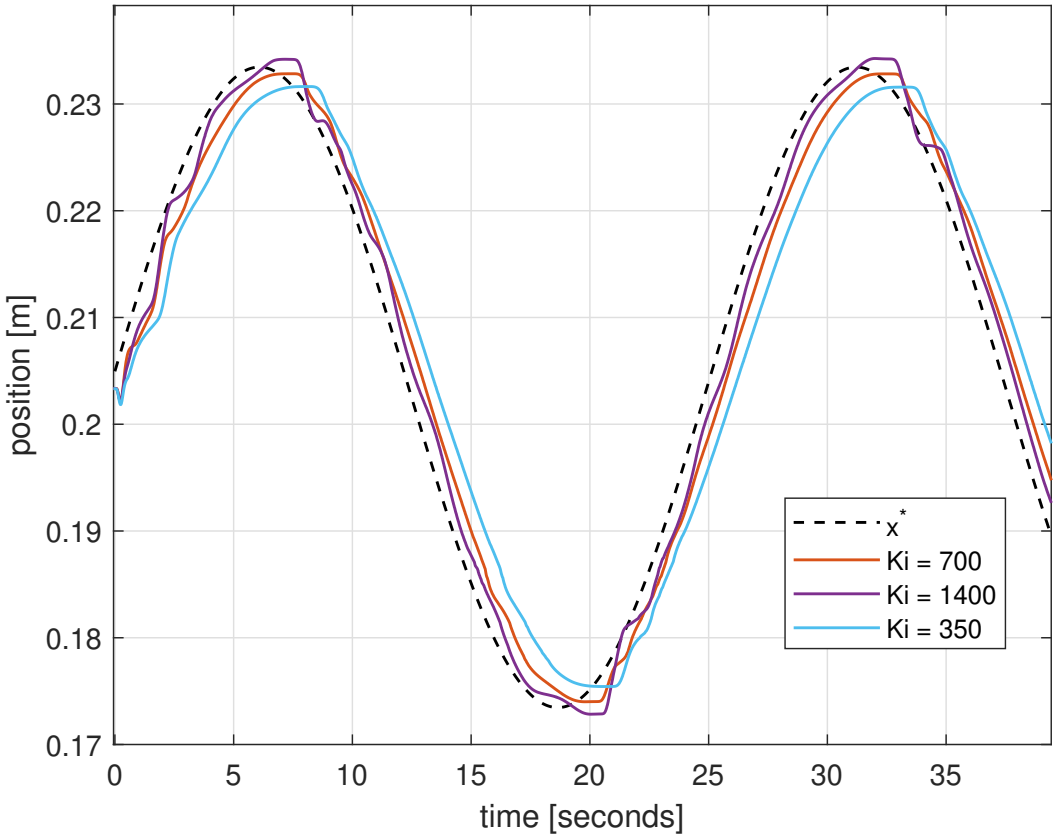

Figure 5.2: Sensitivity with respect to the Ki parameter. The figure represents the response with the PID-only controller. The black dashed line is the reference trajectory. By increasing Ki the phase lag is reduced but the behaviour is more oscillatory, dangerous for the robot structure (purple line). Reducing Ki generates too sluggish response (light blue line). The value of Ki = 700 is chosen because it guarantees the best trade off (red line).

Summarizing, the tuned PID values are: $K_{px} = K_{py} = 500$, $K_{ix} = K_{iy} = 700$, $K_{dx} = K_{dy} = 40$. It is worth noting that the PID controller alone guarantees acceptable performance in terms of trajectory tracking, and thus this confirms how, even with the simplified static mapping described in Chapter 3, it is possible to perform operations in the task space of the robot.
Furthermore, since the control action of the PID controller is accurate, it can be used as learning signal for the Neural Network. The contribution of the data-driven approach will be now to enhance the PID performance and especially to guarantee the online adaptation to external disturbances which can be represented as payloads or changes in the material properties.

5.0.2. NN Tuning

This section describes the passages followed for the tuning of the neural network, together with sensitivity analysis to justify the choices of the parameter values.

Once the PID is properly tuned, it can be deployed to update online the weights of the neural network using eq. (3.11). As said in Section 3.2, the performance of the network is affected by several parameters, such as the learning rate γ, the regularization term ρ, the number of neurons composing the hidden layer N and the number of the future samples in input K. For the tuning of the network parameters, the same reference signal previously used to tune the PID (eq. (5.1)) is considered. The tuning starts by adjusting the learning rate, keeping fixed $N = 60$, $K = 11$ and $\rho = 0.0$. The choice of these parameters is related to the results found from the simulation. In fact, with $K = 11$ the input layer has a dimension of $R = 3K = 33$, hence the number of neurons in the hidden layer should be bigger than that to guarantee the right learning capability to the network. A good trade-off developed in simulation is to select N as double of R. Thus in this case, fixed $K = 11$, it is selected $N = 60$.

During the γ tuning process, the value of $\mathbf{u_{ff}}$ and $\mathbf{u_{fb}}$ are monitored to check if the network learned the correct control output. While the feedforward network progressively learns the right control action, $\mathbf{u_{ff}}$ progressively replaces the action of the PID controller. The result is that the average magnitude of the PID action reduces over time. Results show how with large γ the NN can become unstable, while with small γ it needs a long learning phase. Experimentally, the value of γ is fixed at 0.7.

After the tuning of the learning rate γ, the focus shifts on the regularizing term ρ (forgetting factor).

For $\rho = 0$, the elements in $\boldsymbol{W}$ present high oscillation over time. By increasing ρ instead $\boldsymbol{W}$ converges more rapidly with reduced and limited oscillations. However, an excessively large ρ prevents the network from learning. Similar results can be observed in the simulations on the simplified system fig. 3.5. A trade-off is experimentally found with a dichotomic search at $\rho = 0.1$.

For what concerns N and R, after the initial selection, both parameters have been increase and the RMSE between the reference signal and the real robot position has been evaluated. However, a growth of these values did not significantly change the performance of the system. Therefore, N and R have not been changed with respect to their initial values.

The following paragraphs show more in detail the experimental results used for validation and selection of the hyper-parameters of the neural network.

γ and ρ sensitivity

Figure 5.3 shows how the behaviour of the system with the PID+NN architecture changes according to the values of γ and ρ. The PID controller is kept fixed, together with the other neural network parameters.

As said above, the starting point for the tuning of these parameter was to keep $\rho = 0$ and increase γ until the desired performance are met. However, it is worth noting that an additional contribution was given by the results found in simulation in which the ratio between γ and ρ should be around 5:1. Hence, a similar proportion has been kept even for the experimental evaluation, specifically 7:1.
Starting with low values of γ and ρ ($\gamma = 0.175$, $\rho = 0.025$, light blue line), the response of the system is accurate but it takes more time compared to the others to reach the steady state (the neural network learns slower, as expected). Moreover, although the delay in the response is reduced with respect to the PID, with low γ the behaviour is characterized by the biggest mismatch at steady state among the PID+NN architectures.
Increasing the values of γ and ρ proportionally ($\gamma = 0.35$, $\rho = 0.05$, blue line), it is possible to note how the network converges faster to the steady state value, reducing also the delay with respect to the previous case. It is worth noting that in this case the initial learning transient is characterized by oscillations which vanish as soon as the weights adapt correctly. Doubling again the value of γ and ρ ($\gamma = 0.7$, $\rho = 0.1$, red line), the response shows fast learning transient, with oscillations which are reduced rapidly, and almost perfect tracking without delayed response. Finally, with $\gamma = 1.4$ and $\rho = 0.2$, the neural network is characterized by a too large learning rate for the weights adaption which will result in persisting oscillations that affect the performance and that can cause damages to the soft robot. For this reason, values above $\gamma = 1.4$ have not been considered.

In conclusion, the chosen values are $\gamma = 0.7$ and $\rho = 0.1$ because they guarantee the right trade-off between learning time, reduced oscillations and trajectory tracking. However, it is worth noting that, apart from the high γ case, all the responses of the PID+NN architecture converge to similar trajectory, as expected also from the simulation.

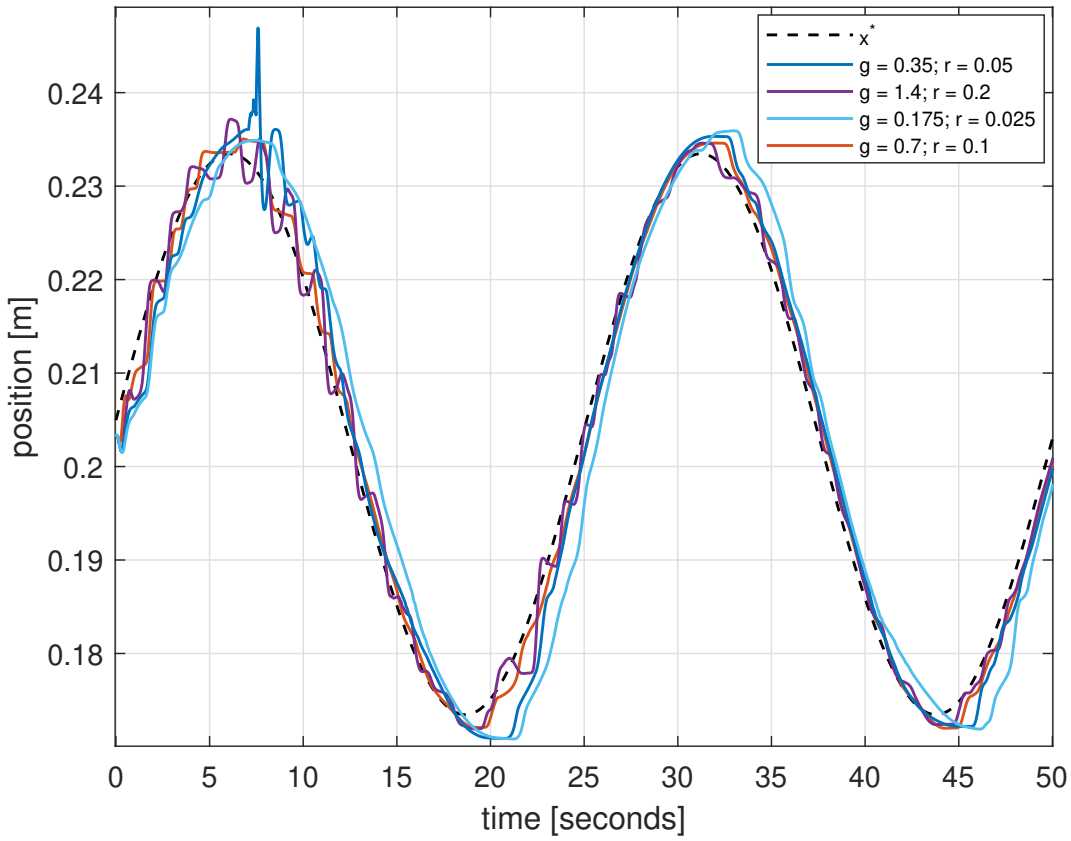

Figure 5.3: Sensitivity with respect to γ and ρ parameters. With high γ (purple line $\gamma = 1.4$) the learning is fast but the response is characterized by dangerous oscillations. With low γ (light blue line $\gamma = 0.175$) the network is not able to adapt sufficiently fast to perfectly follow the reference but the behaviour is smooth. The chosen value is $\gamma = 0.7$ which corresponds to the best trade-off. The black dashed line is the reference trajectory.

N and K sensitivity

This paragraph contains the experimental results which justify the choice of the parameter values N and K. For all the experiments, $\gamma = 0.7$ and $\rho = 0.1$ are considered.

Figure 5.4 highlights the sensitivity with respect to the N parameter, keeping fixed the value of K. As it is possible to see, the number of nodes in the inner layer has an influence mainly on the adaptation time since, once the learning is complete, the response

is almost the same for all the different values. Starting with $N = 60$ (red line), the response guarantees an optimal trade-off between accuracy and learning time. Doubling the value $N = 120$ (purple line), the response is characterized by a similar steady state behaviour. However, it takes more time to learn and adapt. This is because more weights need to be tuned to reach the optimal condition. This situation is even amplified for the case of $N = 240$ (light blue line), which needs more than on period to adapt, as shown by the high overshoot present also in the second peak of the reference. Given these results, the natural choice is to keep $N = 60$. However, it is worth noting that this value highly depends on the task and the adaptive capability needed by the network. For more complex systems, which may perform more complex tasks a too low number of neurons corresponds to a lost of information, thus decreasing the performance.

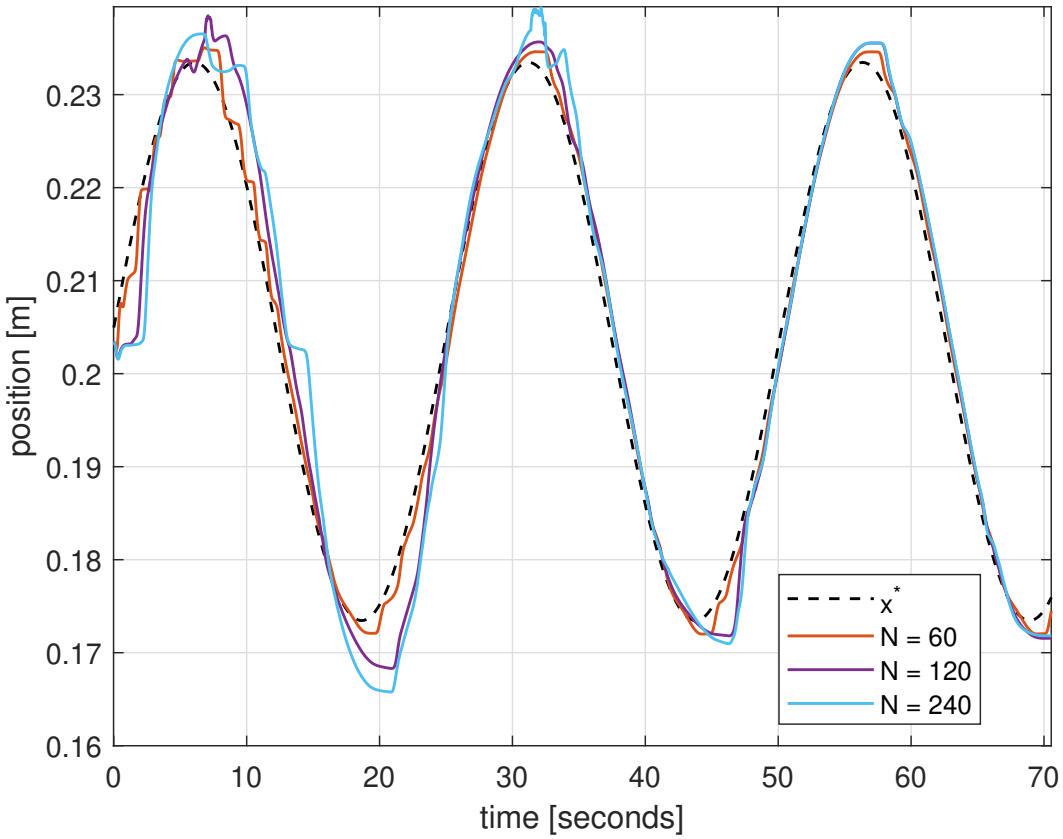

Figure 5.4: Sensitivity with respect to N parameter. With high N (light blue line $N = 240$) the learning is slow, in fact it takes two peaks before reaching the steady state. With low N (red line $N = 60$) the network adapts faster and is able to perfectly follow the reference at steady state. Hence, the chosen value is $N = 60$. The black dashed line is the reference trajectory.

The results concerning the sensitivity with respect to K are shown in fig. 5.5. For all the experiments but the last one, the value of $N = 60$ has been selected.

As it is visible, considering $K = 11$ (red line) guarantees good performance in terms of trajectory tracking, with the smoothest behaviour among all.

Increasing the values at $K = 22$ (purple line) corresponds to an increase of the learning time before reaching the steady state, without actually a benefit from the trajectory tracking point of view.

The doubling of such value ($K = 44$, light blue line), generates a response which is characterized by oscillations, even after the learning phase. This is due to the fact that the input layer has dimension ($R = 3K = 132$) bigger than the hidden one ($N = 60$), thus the information is not correctly expanded from one layer to the other.

To further prove this concept, the idea is to increase the value of N proportionally to the value of K, thus respecting the 2:1 ratio found also in the simulation.

In fact, increasing N, together with K, generates a performance which is comparable to the first one (red line) at steady state (yellow line). This is because the ratio between N and $R = 3K$ is again around 2:1. However, in this case, the learning time is greater than the one for the first case.

Thus, in conclusion, the parameters which guarantees the optimal trade-off are $N = 60$ and $K = 11$. In fact, they guarantee a fast learning time with a smooth and precise behaviour at steady state. This is coherent also with the funding of the simulation on the simplified model (see Section 3.3).

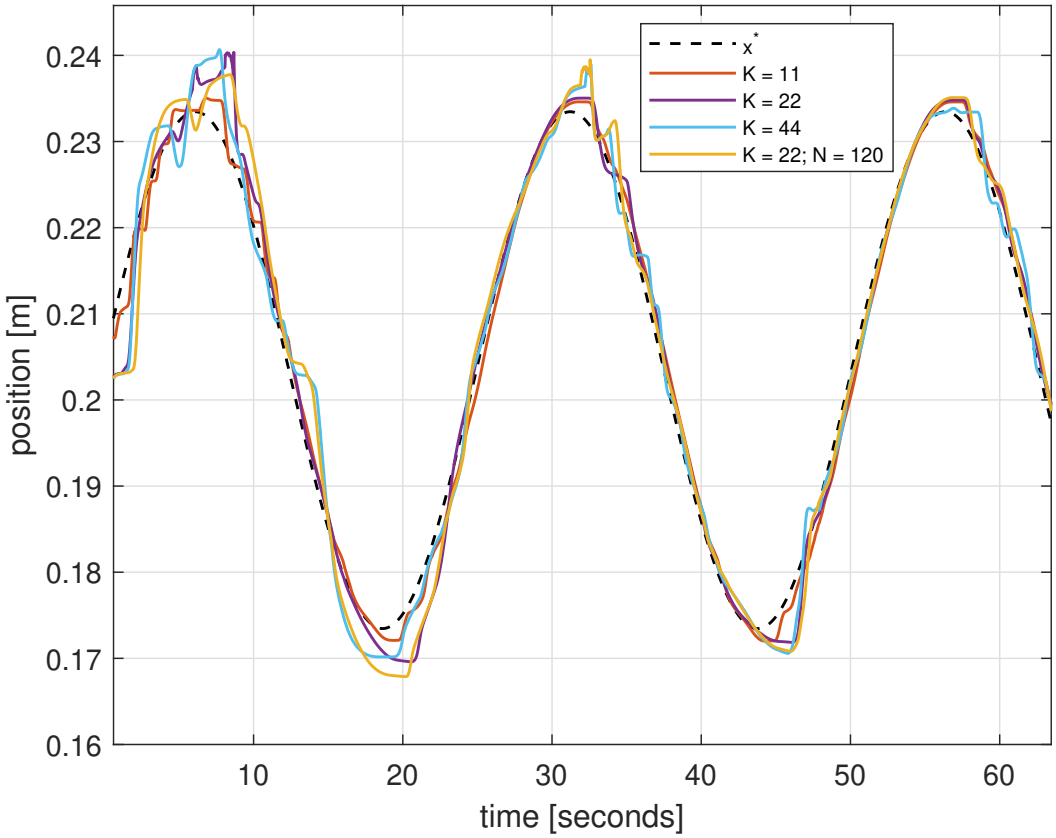

Figure 5.5: Sensitivity with respect to K parameter. N is fixed to 60. With $K = 11$ (red line) the neural network correctly learns the control input to perfectly track the reference. With high K the neural network does not have enough hidden neurons to adapt to the entire input signal and thus the response is characterized by persistent oscillations ($K = 44$ light blue line). Increasing K and increasing N proportionally (yellow line) generates a response at steady state which is accurate. However, the learning is slower due to the increased number of nodes. Hence, the chosen value is $K = 11$. The black dashed line is the reference trajectory.

5.0.3. PID+NN performance

This section analyses the performance of the PID+NN architecture with the values of the parameters described above ($\gamma = 0.7$, $\rho = 0.1$, $N = 60$, $K = 11$).

The results of the Feedback Error Learning architecture are shown in fig. 5.6. In particular, the figure is composed of 4 sub-plots. The first one shows the reference and the position of the tip of the actuator over time. The second and the third show the control pressures $\mathbf{u_{fb}}$ and $\mathbf{u_{ff}}$ over time (see fig. 3.1). Each curve is related to one of the actuated chambers. The last plot reports the trend of the weights.

From fig. 5.6 it can be seen that in the first 20s, the PID corrects the action of the neural network. Indeed, during these first instants, the NN is still learning and cannot provide the correct control action. In fact, the magnitude of $\mathbf{u_{ff}}$ is very high in this first phase. This effect can be also seen in the fourth plot, looking at the adaptation of the weights from a random initial value of 1.

After 20s, the neural network is learning the feedforward signal needed to track the reference and thus the action of the PID starts to decrease in terms of magnitude. The NN weights present smaller oscillations, further proof of the completed learning phase.

The steady state is eventually reached after 25s. Most of the contribution of the control signal is given by $\mathbf{u_{ff}}$, while the PID is used for small corrections only. The weights show residual oscillations centred around 0. The periodic oscillations are explained by the network's acquisition of a "local" control strategy, designed for a short time horizon of length $K\Delta T$. Here, ΔT denotes the sampling period of the reference trajectory (in the experiments $\Delta T = 0.05s$), as detailed in Chapter 3. The continuous adaptation of weights is aimed at optimizing online the tracking process.

These results are coherent with the simulations, even if on the simplified model. Furthermore, they experimental prove the robustness of the Feedback Error Learning control. In fact, when the learning of the neural network is not yet complete, the PID controller compensate for it, as disturbance rejector. When instead the NN provides a good control action, the PID reduces its workload and it will act just for high frequency variations. These latter cannot be immediately tackled by the NN, which needs time to adapt, and thus it works in the low frequency, generating the main contribution to track the reference.

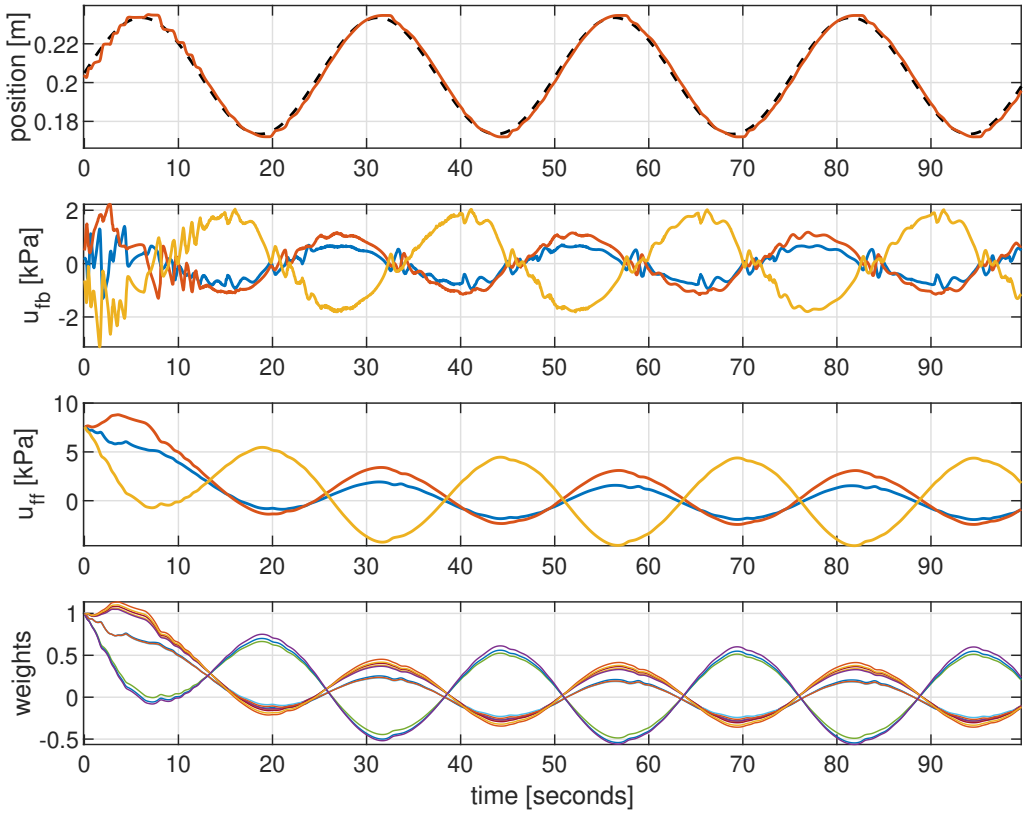

Figure 5.6: Tracking performance of the control system with both feedforward and feedback signal. The first subplot shows the reference signal (black) vs the actual end-effector position of the robot (red). The second and third subplots show PID and NN control signals, respectively. The last subplot illustrate the trend of a subset of weights of the NN (a selection among 180 weights). The RMSE obtained at steady state is of 0.0011 m. This is an improvement in comparison to the PID controller alone.

5.0.4. PID vs PID+NN Linear Tracking

In this section, the performance of the PID-only controller are compared with the ones achieved with the PID+NN architecture, showing the improvements brought by the latter. The reference trajectory is defined as eq. (5.1) and the metric used for comparison is the Root Mean Square Error between the reference and the actual tip position at steady state. The RMSE is defined as

$$RMSE = \sqrt{\Sigma_{i=1}^{n} \left(\frac{x_i^* - x_i}{n} \right)^2}$$

In terms of tracking performance, the system PID+NN performs better than the PID. The two are compared in fig. 5.7, where the focus is on a time interval where the outputs of the two controllers are at the steady state. As visible from the plot, although the controller PID+NN (red line) provides a slightly higher overshoot (less than 2mm), the phase delay between $\mathbf{x}^*$ and $\mathbf{x}$ is much lower compared to the PID (blue line). In terms of quantitative metric, the proposed method provides an RMSE of 0.0011m, which is less than half of the result obtained with the PID alone (0.003m).

This result clearly shows the superior performance in terms of trajectory tracking achieved with the Feedback Error Learning architecture.

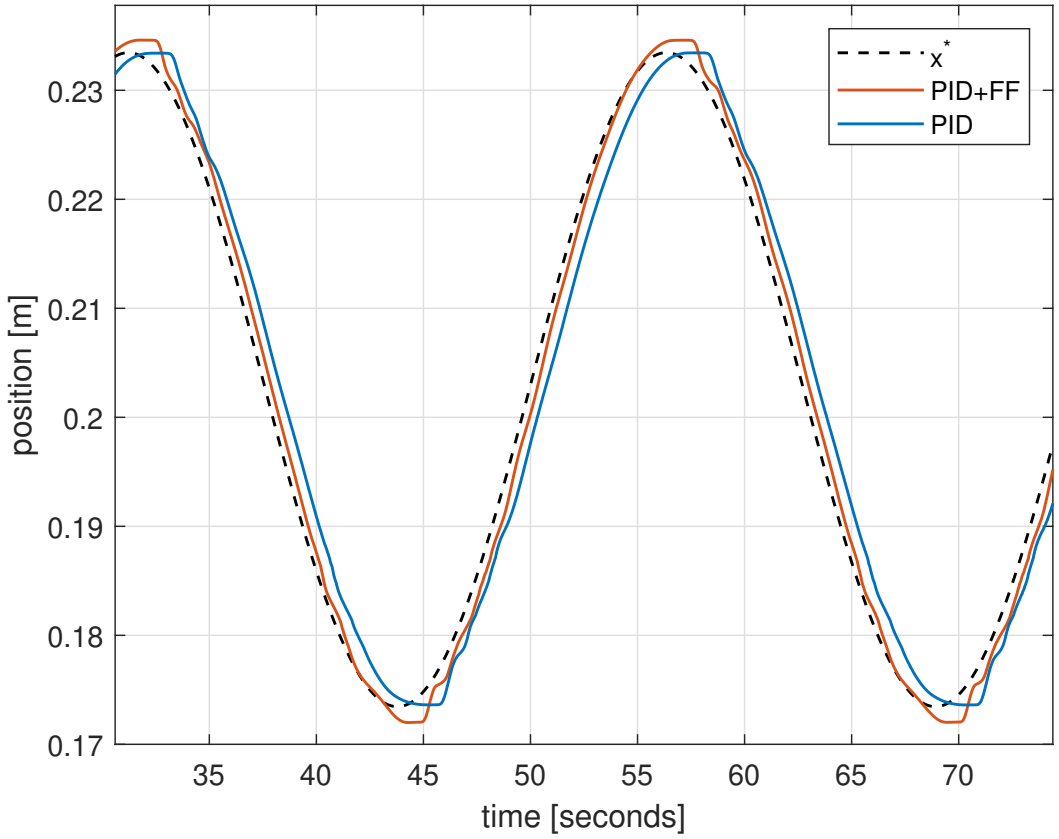

Figure 5.7: Comparison between PID+NN and PID at steady state. The dashed line is the reference signal. The red and blue lines are related to the PID+NN and PID respectively. The PID+NN offers better tracking, with reduced lag.

5.0.5. PID vs PID+NN Circle Reference

This section analysed the performance of the two control strategies when a circular reference in the plane x-y has given, as shown in fig. 5.8 and in fig. 5.9. The soft robot is commanded to follow a circular path with a radius of 0.03 meters in the $x-y$ plane. The reference to the robot is given as:

$$\mathbf{x}^* = \begin{bmatrix} 0.03\cos(0.25t) + 0.032 \\ 0.03\sin(0.25t) + 0.203 \\ \cdot \end{bmatrix} \tag{5.2}$$

where $\cdot$ is an arbitrary value along the z-axis, not considered during the task. The circle trajectory is a standard test for trajectory tracking performance, often used in literature [19, 21, 23]. The two controllers are compared by evaluating their performance in terms of RMSE. The PID+NN (red line) control scheme exhibits a longer initial transient compared to the PID controller (blue line).

This is due to the online learning phase. At the beginning of training, the NN generates the wrong control action and the PID needs to intervene. However, the neural network eventually learns the correct pressure pattern to be applied. At steady state, the proposed approach delivers superior performance with respect to the PID.

In terms of quantitative tracking performance, at steady state the RMSE for the PID+NN controller is 0.0016m, which is significantly lower than the PID, providing an RMSE of 0.0041m. This is mostly due to the reduced tracking "lag" offered by the PID+NN control scheme, as it is possible to see in fig. 5.8. This result is coherent with what previously shown in fig. 5.7 for linear tracking.

It is important to note that, while the oscillations observed in the PID-only response are partially mitigated in the PID+NN architecture, they are not entirely eliminated. This is because the neural network relies on the control action derived from the feedback, and therefore, it is influenced by the existing oscillations.

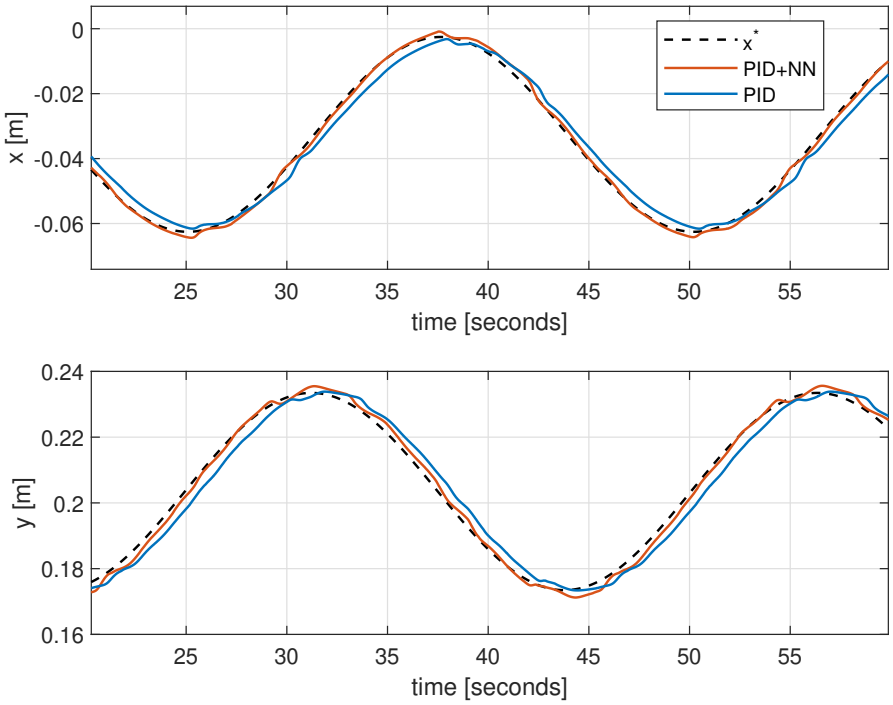

Figure 5.8: Comparison between PID+NN and PID in the behaviour of $x(t)$ and $y(t)$, for circular trajectory tracking.

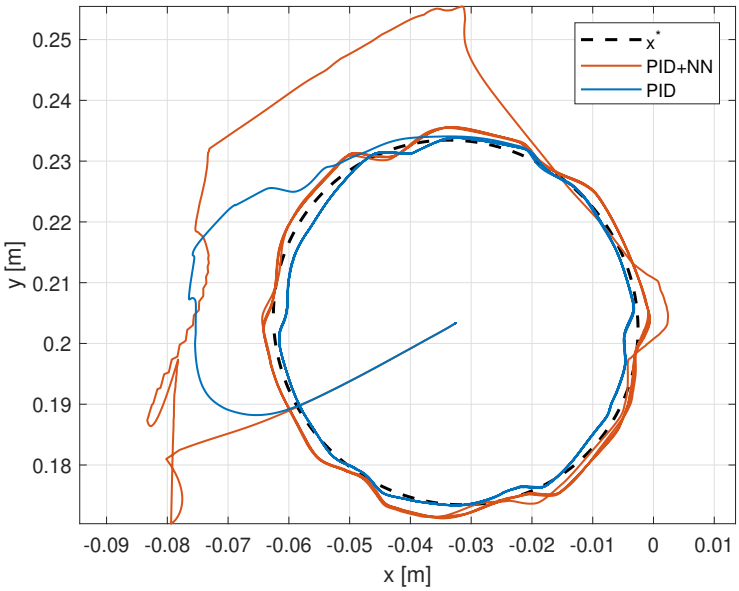

Figure 5.9: Circular trajectory tracking. The blue and red lines indicate the output of the system with PID and PID+NN respectively. The RMSE of PID-only is 0.0041m while the one of PID+NN is 0.0016m

5.0.6. Linear Tracking with variable load

This section aims to showcase the adaptability of the online learning process. A sinusoidal reference signal along the y-axis is commanded having the form:

$$\mathbf{x}^* = \begin{bmatrix} . \\ 0.03\sin(0.25t) + 0.203 \\ . \end{bmatrix}$$

During the task execution, a change in the dynamic of the robot is force by attaching and removing payloads, consisting of small weights connected to the tip of the robot with a thread. More in detail, the robot starts the task without any attached payload. Then, an initial weight of 10g is manually attached to the soft arm tip. Importantly, this weight was not rigidly connected to the robot, allowing it to oscillate along with the robot's movements, thereby increasing the complexity of the trajectory tracking task. The weight is then subsequently removed and substituted with a heavier load of 20g, corresponding to about the 30% of the robot weight. In the last part of the experiment, the payload is removed, thus changing again the dynamic of the system.
The results are shown in fig. 5.10. The plot shows both the tracking of the sinusoidal signal and a selection of the weight trends of the neural network. The Figure includes 4 labels indicating when the dynamic of the system changes due to load variations at the end-effector.
Similar to fig. 5.6, after an initial transient, the NN learns the correct control action and its weights oscillate around zero. After 55s a 10g payload is manually added. This is labelled in fig. 5.10.(a). This swift change in the dynamics of the system increases the tracking error causing the PID to take over. After a brief transient period, the NN learns the new control action. This can be noted by looking at the trend of the weights between fig. 5.10.(a) and (b), whose oscillations are slightly higher in amplitude. When the load is removed in fig. 5.10.(b), the PID corrects the tracking error, and the NN learns again the necessary control action. The oscillations of its weights change accordingly. It is also worth noting that the trend is the same visible before the application of the 10g weight.
In fig. 5.10.(c) a 20g load is applied. After the transient the amplitude of the weights' oscillations is slightly higher than in the case of the 10g payload. This is reasonable: the feedfordward action needs to be stronger to compensate for the heavier load.
When the payload is removed in fig. 5.10.(d), the PID reacts, the neural network adapts its weights, and the tracking performance is restored. The weights of the NN return to the no-load trend at the beginning of the experiment. In this experiment, the proposed approach guarantees good tracking performance with a RMSE of 0.0017m.

Most importantly, the results shown in fig. 5.10 experimentally demonstrate the controller's ability to deal with real time changes in the dynamic of the system, thus eliminating the need for a retraining phase each time before of the new task [25].
It is worth noting that, once the weights have converged around 0 during the first learning period, the adaptation becomes much faster and hence it can compensate for sudden dynamic changes as in this case. Furthermore, the online learning architecture can be used for dealing with material changes. For example, the material used for the soft robot described in Chapter 4 is sensible to UV light and it will degrade with the passing of the time. Thanks to this technique it would be possible to adapt online to the properties changes, which are expected to be inherently slower than the addition/removal of payloads. Thus, performance and robustness can be enhanced.

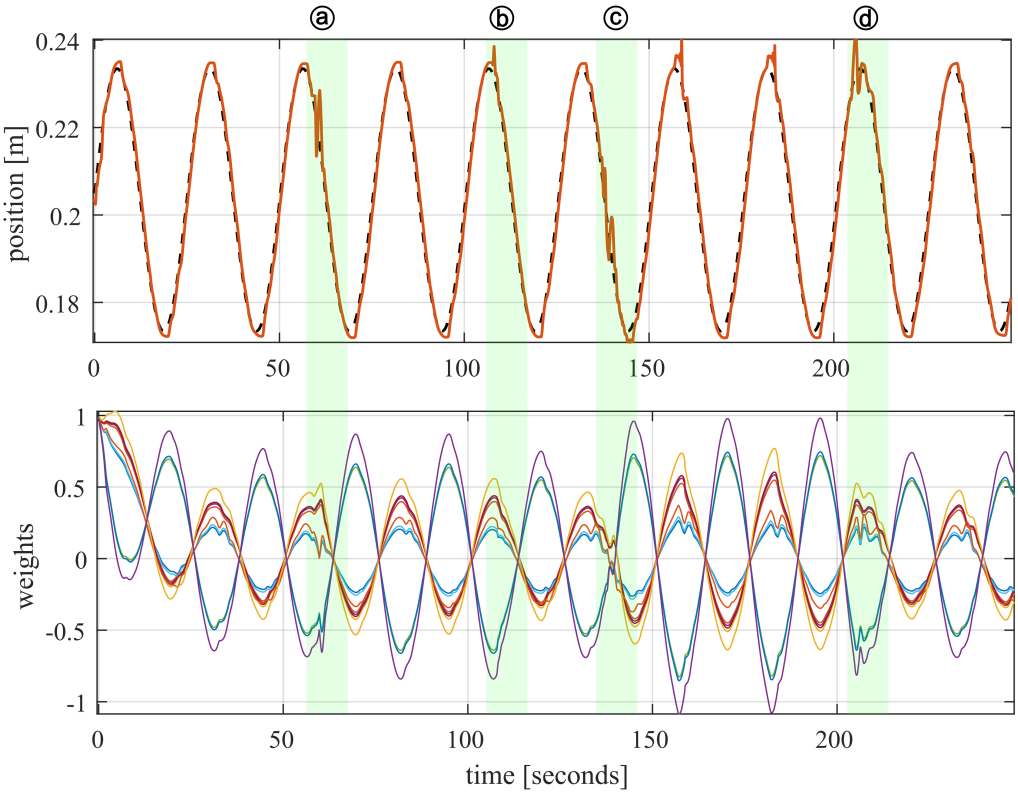

Figure 5.10: Response of PID+NN controlled system for sinusoidal reference and variable loads. Labels (a)-(d) indicate transient adaptations due to load variations: (a) 10g load added; (b) load removed; (c) 20g load added; (d) load removed. The NN weights and thus the NN output adapt to the changes in the dynamics. With added payload, the weights settle down on higher values because a stronger action is required to move the robot. When the payload is removed, the weights come back to their initial steady state values.

6 | Conclusions and future developments

This chapter summarizes the results achieved in the thesis and suggests possible future developments.

The aim of this work is to develop a control architecture for soft robots which can overcome the problems of the current methodologies. More specifically, pure model-based controllers rely on the definition of a complex model for the system, not able anyway to tackle the inherently nonlinearities, drifts, hysteresis effects within the soft robot. On the other side, with pure data-driven solutions, the modelling issue is overcome and the control action is tailored for the specific application. However, these controllers are very sensitive to changes in the dynamics of the robot and hence they often need an offline retraining phase, time and eventually cost consuming, to deal with external disturbances, added payloads or changes in the material properties.

For the above reasons, this thesis focuses on an hybrid approach exploiting the Feedback Error Learning framework, guaranteeing trajectory tracking without the need for complex modelling, and online adaptation to dynamic changes. The control action to the soft robot is given by two contributions: a feedback model-based and a feedforward data-driven controller. The idea is to generate a feedback control action using a simplified model of the robot, exploiting the symmetries in its design. The performances are then enhanced using the data-driven feedforward contribution, which can also online adapt to changes in the robot dynamic, thanks to an online learning mechanism based on the signal coming from the feedback controller. In this way, the proposed approach overcomes the need for retraining phases.
To validate the methodology, a real soft arm, featured with omnidirectional movement, has been designed and fabricated. The experimental results show how the proposed approach enhanced the performance, with a reduction of more than 50% in terms of RMSE of trajectory tracking. Furthermore, they highlight the online adaptation capability to

external disturbances, represented by additional payloads, with a weight around 30% the one of the entire robot.

The approach has been validated using a PID and a 1-layer neural network as feedback and feedforward controllers, respectively. However, it can be easily extended to more powerful architectures. More specifically, the adaptation law for the neural network developed in this thesis is given by the standard gradient descent algorithm, keeping the learning rate γ constant over time. However, it can be improved by considering other optimization algorithms such as Adagrad which online adapts the learning rate according to the new data coming, solving thus the problem of parameter tuning.
Moreover, the proposed update rule optimizes the weights only from hidden to output layer ($\mathbf{W}$), keeping fixed at random value the ones from input to hidden layer ($\mathbf{Z}$). The performance can thus be improved selecting $\mathbf{Z}$ based on a pre-training experiment, rather than randomly. This is coherent with the concept of transfer learning, often used for neural networks focused on image or pattern recognition [37]. It considers NNs with initial layers weights pretrained on a general dataset and updates just the task-specific final layers weights (head of the net). In this way, the training becomes faster and more specialized, allowing to re-use pretrained networks for different applications. In the context of this thesis, the pretraining may reduce the learning time and the initial overshoots due to the incorrect control action.
An alternative solution to the proposed learning rule is the computation of the gradient using the back-propagation algorithm. In this way, all the weights can be optimized, potentially increasing the performance. However, the online computation of back-propagation can be time and computing consuming. Thus, there may be issues for an online adaptation of the control signal. This is one of the reasons why neural networks, so far, are usually trained offline based on a time-consuming process and then they are deployed on the real system without being retrained anymore.

To test the different architectures and compare their performance, a necessary improvement is a simulator of the soft robot. This can be tackled in different ways:

- Physical model of the robot.
  The soft robot can be described, with analytical equations, as a spring and mass system or exploiting the PCC modelling. However, in this way, the robot description, although reliable, cannot be perfectly accurate. Thus, a potential problem is the gap between simulation and real deployment (often present in literature as Sim-to-Real-Gap). In fact, simulations may differ from actual system results, preventing from an accurate analysis of the methodology.

- Data-driven model of the robot.
  It could be possible to use a neural network, based on data from the real system, to describe the soft robot. This solution can guarantee high accuracy and thus reducing the Sim-to-Real-Gap problem. However, it is not immune from drawbacks. In fact, the NN model should be trained on a very exhaustive dataset, to learn all the possible movements. This will imply a time consuming data collection phase. Moreover, the robot material can degrade over time, leading to a potential decline in the accuracy of the model. Hence, as discussed through the whole thesis, retraining phases are needed.

- FEM simulation of the robot.
  The soft robot model can be simulated in real time using FEM. Although this option guarantees good accuracy, it faces computational challenges. A possible solution to speed up the process can be to reduce the mesh accuracy of the FEM and thus reducing the number of variables. However, this can decrease the simulation reliability, thus falling again in the Sim-to-Real-Gap problem.

Further developments can tackle also the experimental setup. Specifically, improvements can come mainly from three sources: (i) soft robot material (ii) soft robot design (iii) measurement coming from proprioceptive sensors. With different material the soft robot can be more robust and reliable, with constant properties over time. However, the various materials experimented so far have not yielded the same performance standards achieved with Agilus30. A solution involves retaining the Agilus30 as core material and applying on it a UV-resistant layer. This protective layer prevents changes in properties induced by light exposure, ensuring sustained performance.
The soft robot design can be improved by adding elongation features, exploiting FEM simulations (COMSOL) to understand its behaviour before 3D-printing it.
Considering the feedback measurement, to give the robot the capability of working even in an unstructurated environment, proprioceptive sensors can be used, solving the need for cameras feedback. As an example, tactile sensors can be used to build a relationship between the input pressure and the robot configuration, from which it is possible to get the position of the tip, as described in [38].

The aforementioned observations will be the focus of future research, aiming to enhance the controller performance and make the robot suited for a diverse range of potential applications.

Bibliography

[1] Freya Mathews. Towards a deeper philosophy of biomimicry. *Organization & Environment*, 24(4):364–387, 2011.

[2] Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521:467–475, 2015.

[3] Stephen Coyle, Carmel Majidi, Philip LeDuc, and K Jimmy Hsia. Bio-inspired soft robotics: Material selection, actuation, and design. *Extreme Mechanics Letters*, 22:51–59, 2018.

[4] Panagiotis Polygerinos, Zheng Wang, Kevin C Galloway, Robert J Wood, and Conor J Walsh. Soft robotic glove for combined assistance and at-home rehabilitation. *Robotics and Autonomous Systems*, 73:135–143, 2015.

[5] Dong Wang, Baowen Zhao, Xinlei Li, Le Dong, Mengjie Zhang, Jiang Zou, and Guoying Gu. Dexterous electrical-driven soft robots with reconfigurable chiral-lattice foot design. *Nature Communications*, 14:5067, 2023.

[6] Nicolo Garbin, Long Wang, James H. Chandler, Keith L. Obstein, Nabil Simaan, and Pietro Valdastri. A disposable continuum endoscope using piston-driven parallel bellow actuator. In *2018 International Symposium on Medical Robotics (ISMR)*, pages 1–6, 2018.

[7] Carly M. Thalman, Quoc P. Lam, Pham H. Nguyen, Saivimal Sridar, and Panagiotis Polygerinos. A novel soft elbow exosuit to supplement bicep lifting capacity. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018*, pages 6965–6971, December 2018.

[8] W. McMahan, V. Chitrakaran, M. Csencsits, D. Dawson, I.D. Walker, B.A. Jones, M. Pritts, D. Dienno, M. Grissom, and C.D. Rahn. Field trials and testing of the octarm continuum manipulator. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2336–2341, 2006.

[9] Michele Xiloyannis, Leonardo Cappello, Dinh Binh Khanh, Shih-Cheng Yen, and

Lorenzo Masia. Modelling and design of a synergy-based actuator for a tendon-driven soft robotic glove. In *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1213–1219, 2016.

[10] Minchae Kang, Ye-Ji Han, and Min-Woo Han. A shape memory alloy-based soft actuator mimicking an elephantrsquo;s trunk. *Polymers*, 15(5), 2023.

[11] B. Tondu and P. Lopez. Modeling and control of mckibben artificial muscle robot actuators. *IEEE Control Systems Magazine*, 20(2):15–38, 2000.

[12] Bobak Mosadegh, Panagiotis Polygerinos, Christoph Keplinger, Sophia Wennstedt, Robert F. Shepherd, Unmukt Gupta, Jongmin Shim, Katia Bertoldi, Conor J. Walsh, and George M. Whitesides. Pneumatic networks for soft robotics that actuate rapidly. *Advanced Functional Materials*, 24(15):2163–2170, April 2014.

[13] Fionnuala Connolly, Conor Walsh, and Katia Bertoldi. Automatic design of fiber-reinforced soft actuators for trajectory matching. *Proceedings of the National Academy of Sciences*, 114, 12 2016.

[14] Dylan Drotman, Michael Ishida, Saurabh Jadhav, and Michael T. Tolley. Application-driven design of soft, 3-d printed, pneumatic actuators with bellows. *IEEE/ASME Transactions on Mechatronics*, 24(1):78–87, 2019.

[15] Cosimo Della Santina, Christian Duriez, and Daniela Rus. Model-based control of soft robots: A survey of the state of the art and open challenges. *IEEE Control Systems*, 43:30–65, 6 2023.

[16] Michael Yip and David Camarillo. Model-less feedback control of continuum manipulators in constrained environments. *IEEE Transactions on Robotics*, 30:880–888, 8 2014.

[17] Michele Giorelli, Federico Renda, Marcello Calisti, Andrea Arienti, Gabriele Ferri, and Cecilia Laschi. Neural network and jacobian method for solving the inverse statics of a cable-driven soft arm with nonconstant curvature. *IEEE Transactions on Robotics*, 31:1–12, 8 2015.

[18] S Satheeshbabu, N K Uppalapati, T Fu, and G Krishnan. Continuous control of a soft continuum arm using deep reinforcement learning. In *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, pages 497–503, 2020.

[19] Cosimo Della Santina, Robert K. Katzschmann, Antonio Biechi, and Daniela Rus. Dynamic control of soft robots interacting with the environment. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 46–53, 2018.

[20] Robert K. Katzschmann, Cosimo Della Santina, Yasunori Toshimitsu, Antonio Bicchi, and Daniela Rus. Dynamic motion control of multi-segment soft robots using piecewise constant curvature matched with an augmented rigid body model. In *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, pages 454–461, 2019.

[21] Milad Azizkhani, Anthony L. Gunderman, Isuru S. Godage, and Yue Chen. Dynamic control of soft robotic arm: An experimental study. *IEEE Robotics and Automation Letters*, 8(4):1897–1904, 2023.

[22] Christian Duriez. Control of elastic soft robots based on real-time finite element method. In *2013 IEEE International Conference on Robotics and Automation*, pages 3982–3987, 2013.

[23] Robert K. Katzschmann, Maxime Thieffry, Olivier Goury, Alexandre Kruszewski, Thierry-Marie Guerra, Christian Duriez, and Daniela Rus. Dynamically closed-loop controlled soft robotic arm using a reduced order finite element model with state observer. *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, pages 717–724, 2019.

[24] Zhi Qiang Tang, Ho Lam Heung, Kai Yu Tong, and Zheng Li. A probabilistic model-based online learning optimal control algorithm for soft pneumatic actuators. *IEEE Robotics and Automation Letters*, 5(2):1437–1444, 2020.

[25] Francesco Piqué, Hari Teja Kalidindi, Lorenzo Fruzzetti, Cecilia Laschi, Arianna Menciassi, and Egidio Falotico. Controlling soft robotic arms using continual learning. *IEEE Robotics and Automation Letters*, 7(2):5469–5476, 2022.

[26] Mitsuo Kawato and Hiroaki Gomi. A computational model of four regions of the cerebellum based on feedback-error learning. *Biological Cybernetics*, 68:95–103, 1992.

[27] Stuart Geman. Some averaging and stability results for random differential equations. *SIAM Journal on Applied Mathematics*, 36(1):86–105, 1979.

[28] Aiko Miyamura and Hidenori Kimura. Stability of feedback error learning scheme. *Systems & Control Letters*, 45(4):303–316, 2002.

[29] Anderson Brian D. O., Bitmead Robert R., Johnson C. Richard, Kokotovic Petar V., Kosut Robert L., Mareels Iven, Praly Laurent, and Riedle Bradley D. *Stability of adaptive systems : passivity and averaging analysis*. «The» MIT Press series in signal processing, optimization, and control. MIT Press, Cambridge, Mass, C 1986.

[30] Aiko Miyamura and Hidenori Kimura. Stability of feedback error learning method

for general plants with time delay. *IFAC Proceedings Volumes*, 35(1):303–308, 2002. 15th IFAC World Congress.

[31] Jun Nakanishi and Stefan Schaal. Feedback error learning and nonlinear adaptive control. *Neural Networks*, 17(10):1453–1465, 2004.

[32] H. Gomi and M. Kawato. Learning control for a closed loop system using feedback-error-learning. In *29th IEEE Conference on Decision and Control*, pages 3289–3294 vol.6, 1990.

[33] A. K Ishihara. *Feedback error learning in neuromotor control*. PhD thesis, Stanford University, 2008.

[34] M.T. Hagan and H.B. Demuth. Neural networks for control. In *Proceedings of the 1999 American Control Conference*, volume 3, pages 1642–1656 vol.3, 1999.

[35] Radhey Shyam. Convolutional neural network and its architectures. *Journal of Computer Technology & Applications*, 12(2):6–14p, 2021.

[36] Yao Yao, Liang He, and Perla Maiolino. Spada: A toolbox of designing soft pneumatic actuators for shape matching based on surrogate modeling. *2023 IEEE 5th International Conference on Soft Robotics (RoboSoft)*, 2023.

[37] Jaya Gupta, Sunil Pathak, and Gireesh Kumar. Deep learning (cnn) and transfer learning: A review. *Journal of Physics: Conference Series*, 2273(1):012029, may 2022.

[38] Wenye Ouyang, Liang He, Alessandro Albini, and Perla Maiolino. A modular soft robotic arm with embedded tactile sensors for proprioception. In *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft)*, pages 919–924, 2022.

List of Figures

Acknowledgements

I would like to thank Prof. Paolo Rocco who gave me the opportunity to develop this thesis abroad, trusting my skills. A special thanks also to Prof. Perla Maiolino, who accepted me in her lab in the University of Oxford (UK), making me feel part of the group from day one. I would like to say thank you to Alessandro, Giammarco, Yao, Daniele and all the guys of the Soft Robotics Lab for the support.
Thanks to my friends Elisa, Vazzo, Naza, Mila, Vano who have always stayed in touch with me, making me feel their closeness. A thanks with love to Gaia, who supported me through the process and who never stopped showing her love, even if distance separated us.
Finally, I would like to thank my family who trusted, pushed and sustained me more than ever. Mom, you believed in me more than anyone else. Thank you for making me the man I am. I love you, this thesis is for you.

Oxford - Milano,

19^{th} *December 2023*