IMPERIAL

IMPERIAL COLLEGE LONDON

DEPARTMENT OF BIOENGINEERING

Implementation of implicit adaptation for bio-inspired controller using Feedforward Neural Network

Author:
Diane d'Haultfoeuille

Supervisor:
Prof. Holger Krapp

Word count: 5,979

Submitted in partial fulfillment of the requirements for the MSc degree in Biomedical Engineering of Imperial College London

September 2024

Abstract

Machine learning models have proven to be valid tools for capturing non-linear relationships between data. In this project, a feedforward artificial neural network (FANN) using a multilayer perceptron (MLP) was investigated as a means to potentially enhance the adaptation capabilities of a bio-inspired control system. The focus was to explore if this MLP-based controller is able to handle and generalise well to non-linear dynamical systems. The procedure involved comparing the adaptation performance of two controllers implementing different methods to generate a feedforward command, across various dynamic systems, including linear, pendulum, non-linear damping spring-mass and adapted Lotka-Volterra dynamics. Ten distinct MLP models were created, each of which was evaluated according to their respective mean squared error.

I found that while the FANN controller could perform well under linear, pendulum, and Lotka-Volterra dynamics with some specific MLP architectures, it failed to adapt effectively under a damping spring-mass model. However, the benchmark controller consistently demonstrated robust performance across all the tested dynamics, with lower computational costs. This indicates that traditional controllers assuming a linear relationship between intrinsic data might be more practical for adaptive control.

Acknowledgments

I would like to thank my supervisor, Prof. Holger Krapp, for giving me the opportunity to join his lab and for his guidance throughout this project.

I am also grateful to Benjamin Campbell for his encouragement, and constant support. His expertise and insights have been invaluable in helping me to understand the subject and guide my research.

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNS	Central Nervous System
EMA	Exponential Moving Average
FANN	Feedforward Artificial Neural Network
FB	Feedback
FF	Feedforward
HS	Horizontal system
LR	Learning Rate
MIMO	Multiple-Inputs Multiple-Outputs
ML	Machine Learning
MLP	multilayer Perceptron
MSE	Mean Squared Error
NLDSM	Non-Linear Damping Spring Mass model
SISO	Single-Input Single-Output
SLP	Single Layer Perceptron
SSV	Steady-State Value

List of Figures and Tables

Contents

Chapter 1

Introduction

1.1 Biological insights driving advanced robotics

Animals, insects, and humans possess remarkable abilities to interact with and navigate their environment, adapting swiftly to changing and unpredictable conditions [1]. Thus, they inspire engineers to develop adaptive machines capable of performing complex tasks with human-like proficiency. Therefore, thanks to advances in neuromechanics, scientists have integrated principles of biological motor control into systems' design methods to make them more flexible, efficient, and autonomous. Ramdya and Ijspeert [2] highlight the benefit of combining insights from neuroscience and control theory to not only improve traditional robotic architectures, but also gain a deeper understanding of fundamental science. An illustration of their point focuses on the crawling locomotion of the worm *C. elegans* [3, 4]. Using a neuromechanical model of the worm and simulations on a controller mimicking sensory circuits, Izquierdo and Beer show that the crawling movements are controlled by sensory feedback loops from stretch receptors, rather than relying on central pattern generators (CPGs), which usually generate rhythmic motor patterns. Experiments in control challenged the traditional view of how animals generate movements.

1.2 Adaptive motor control in vertebrates and invertebrates

The motor control of vertebrates and invertebrates, although varied in their specifics, often share common architectures involving feedback and feedforward pathways. These structures enable high-speed and stable movements against external (wind gusts, obstacles, etc.) and internal perturbations (noise in signal transmission, sensory detection, or motor actuation, etc.). Sensory receptors and proprioceptors detect changes in body state or environment and send the information to the central nervous system (CNS), which processes them and generates appropriate motor commands to adjust the animal's state [5].
In the sensorimotor system, the FB control ensures continuously actuates to make real-time adjustments and is essential in stabilisation reflexes. FF controllers gener-

ate actuation commands that are sent to the motor plant. This command initiates muscle contraction, enacting the movement of a limb. It should be noted that the term "feedforward control" is used here in a biological context, where its meaning may differ from that in engineering control theory.
The optomotor response of flying insects is a good example for understanding the advantages of these two structures in the gaze-stabilizing task [6]. These organisms use visual and mechanosensory systems to minimize the relative motion between their eyes and the environment. Rapid compensatory head movements are initiated by FF signals from halteres - modified hindwings that act as gyroscopic sense organs allowing quick adjustments to flight path and orientation -, followed by slower FB signals from the visual system to refine these movements.

However, any living organism faces the "reflex trap", where reflexes would counteract goal-directed behaviours, forcing the organism back to its original trajectory or state. To perform voluntary movements, the fly's nervous system needs to temporarily suppress these reflexes, by means by a signal called "efference copy" [5]. Kim et al. [7] had tried to demonstrate physiological evidence of efference copies by recording activity in the Horizontal System (HS) cells in the fly's brain, which are responsible for processing visual motion information. The efference copy has the same amplitude but opposite sign of the expected signal in response to state changes in order to cancel it out. As a result, the visual system does not alter its output during the intended flight manoeuvre and will not trigger a compensating stabilisation reflex.

The integration of feedback and feedforward signals, along with the use of efference copies (Figure 1.1), highlights the sophistication of biological motor control systems. In their paper [8], B. Campbell et al. demonstrate the advantages of integrating these two structures into a control system, showing better robustness to noise and time delays compared to common engineering control schemes.

1.3 Advancements and challenges in integrating artificial neural networks into control systems

Many real-world applications exhibit complex and highly non-linear behaviours that conventional control methods, while effective for linear dynamics, often struggle to regulate. With recent advances in artificial intelligence (AI), engineers have integrated new computational models, called artificial neural networks (ANNs), into intelligent devices [9]. Inspired by the structure of the brain, ANNs are composed of interconnected nodes organized in layers, allowing for faster and more computationally efficient parallelized information processing. They also learn from experience, handle multiple inputs and outputs, and manage uncertainties [10, 11].
Among the multiple machine learning (ML) models, feedforward artificial neural networks (FANNs) stand out for their ability to approximate non-linear patterns and process signals throughout the network from the input layer to the output layer [12].

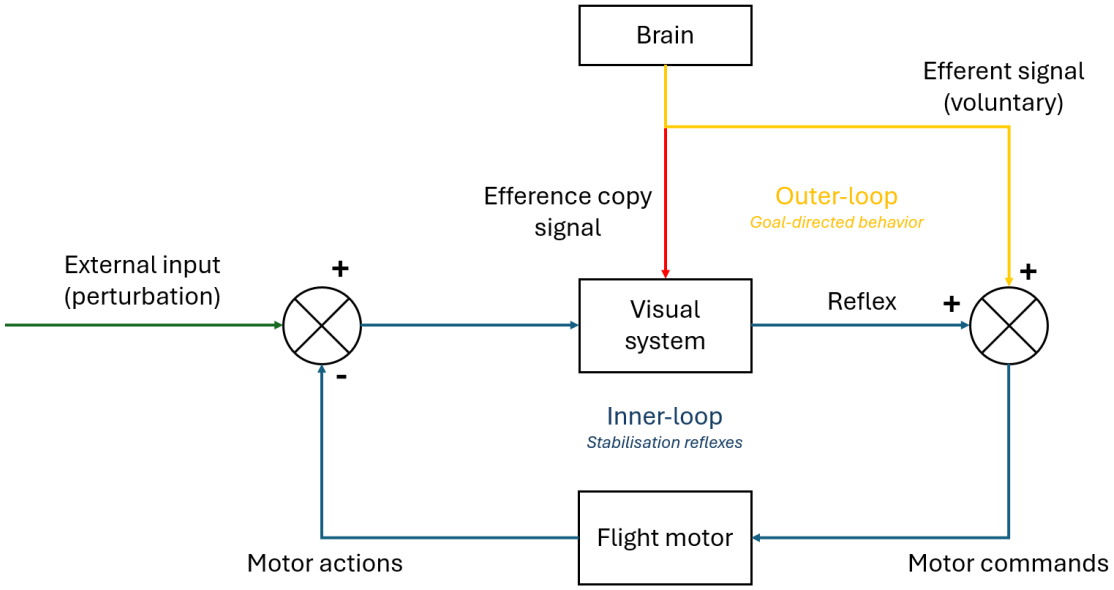


Figure 1.1: **Block diagram of integrated stabilisation reflex, optomotor response and voluntary body saccades in *Drosophilia***. The brain sends a self-induced motor action (yellow connection) and an efferent copy signal (red connection) simultaneously. Efferent copies anticipate the sensory consequences of a voluntary turn to suppress stabilisation reflex responses induced by these changes. However, the visual system remains sensitive to external perturbations, such as induced rotations as a result of turbulent air. Finally, the flight motor system receives motor commands to trigger motor actions. Visual processing and the integration of inner (blue connections) and outer-loop signals significantly benefits from a stabilized gaze. *Adapted from Krapp (2015) [6].*

Even if ANN controllers offer a powerful framework to improve performance compared to traditional control, they are often considered as “black boxes”, meaning that their internal working and decision-making processes are not easily interpretable. Also, they require typically large amounts of training data which may not be readily available, and might not generalize well to novel situations.

The integration of such advances in multidisciplinary fields like robotics, neuroscience and artificial intelligence has seen the emergence of new autonomous systems capable of adapting, learning and anticipating future events as a result of their self-induced state changes. These characteristics have opened up new possibilities that could transform various industries such as manufacturing [13, 14], autonomous transportation [15, 16], or even healthcare with advanced prosthetic and rehabilitation devices for example [17, 18].

1.4 Aim and objectives

In a first attempt to understand the *sensorimotor learning* process, referring to the brain's ability to adjust motor commands over time and learn to model their relationship with desired states, B. Campbell implemented a linear single-layer perceptron (SLP) in the FF filter of a biologically-inspired adaptive controller. It has demonstrated its effectiveness in tracking reference signals under linear dynamics [19]. However, taking advantage of AI, I decided to change the SLP into a multilayer perceptron (MLP), a type of FANN that could capture non-linear transformations between data.

This project therefore seeks to find out whether an MLP-based feedforward controller can effectively learn complex relationships between input and output data, and improve the reference tracking performance under non-linear dynamics. In particular, I am interested in investigating the generalisation abilities of the MLP, specifically whether there is a general neural network architecture that could allow adaptation under new, unseen dynamics.

Chapter 2

Methodological background

2.1 Implicit supervision

The concept of *implicit supervision* [20] plays a crucial role in the sensorimotor learning process. It also highlights the two filter types required to generate the motor command, and thus where the introduction of an MLP could be a valuable addition to this process.

Let us consider a Multi-Input Multi-Output (MIMO) system with N inputs x_i and M outputs u_j that depend on time t such that $\overrightarrow{x} = \begin{pmatrix} x_1(t) & x_2(t) & ... & x_N(t) \end{pmatrix}^T$ and $\overrightarrow{u} = \begin{pmatrix} u_1(t) & u_2(t) & ... & u_M(t) \end{pmatrix}^T$. Assuming a linear relationship, each expected output u_j of the FF controller can be modeled as the sum of the inputs x_i multiplied by a scalar weight w_{ij} :

$$u_j(t) = \sum_{i=1}^{N} w_{ij} \cdot x_i(t) \tag{2.1}$$

Let's define W, the matrix of size $M * N$ containing each weights w_{ij} :

$$W = \begin{pmatrix} w_{11} & w_{12} & ... & w_{1N} \\ w_{21} & ... & ... & w_{2N} \\ ... & ... & ... & ... \\ w_{M1} & ... & ... & w_{MN} \end{pmatrix}$$

With vectorisation : $\overrightarrow{u} = W \cdot \overrightarrow{x}$. Now, let's define a *loss function* $L(t)$ depending on feedback error $e(t)$, which measures how well output states match the actual target values:

$$L(t) = \frac{e(t)^2}{2} \tag{2.2}$$

assuming only one error in the problem for simplification.

Consequently, the model needs to find the best values of w_{ij} that minimise the loss and ensure accuracy of the commands u_j. We choose that the time derivative of w_{ij} and the derivative of the loss function according to w_{ij} are linked by the following

relationship [19]:

$$\frac{dw_{ij}}{dt} = -\alpha \cdot \frac{dL}{dw_{ij}} \qquad \forall i = 1, .., N \quad \text{and} \quad \forall j = 1, ..., M \tag{2.3}$$

where α is a scalar value.
Using the chain rule, we get:

$$\frac{dL}{dw_{ij}} = \frac{dL}{de} \cdot \frac{de}{dw_{ij}} \tag{2.4}$$

By equation 2.2, we have $\frac{dL}{de} = e$ and :

$$\frac{de}{dw_{ij}} = \frac{de}{du_j} \cdot \frac{du_j}{dw_{ij}} \tag{2.5}$$

Equation 2.5 introduces the concept of *sensitive derivatives* $\frac{de}{du_j}$, which quantify how the system performance depends on FF commands. In their paper, Abdelghani et al. hypothesise that these sensitive derivatives are not explicitly provided by a supervisor but learned through experience via a self-supervised process called *implicit supervision* [20].

Let's define the vector $\overrightarrow{F} = \left(\frac{\partial e}{\partial u_1} \quad \frac{\partial e}{\partial u_2} \quad \cdots \quad \frac{\partial e}{\partial u_M}\right)$. Mathematically, we can estimate the sensitive derivatives $\frac{\partial e}{\partial u_j}$ using a second filter that predicts the derivative $\frac{\hat{de}}{dt}$, knowing that :

$$\frac{\hat{de}}{dt} = \sum_{j=1}^{M} \frac{\partial e}{\partial u_j} \cdot \frac{\partial u_j}{\partial t} = \sum_{j=1}^{M} F_j \cdot \frac{\partial u_j}{\partial t} \tag{2.6}$$

The terms e and u_j are both known variables, therefore, so are their temporal derivatives.

In conclusion, the brain is able to generate motor commands through two adaptive filters called the *inner* and the *outer* filters [19]. In the outer filter, the parameters w_{ij} are updated using a traditional gradient descent method depending on the feedback error e. In the inner filter, the sensitive derivatives $\frac{de}{du_j}$ (or F_j) are updated using the same method based on an error between the estimated $\frac{\hat{de}}{dt}$ and the actual $\frac{de}{dt}$. In particular, it takes into consideration the K past values of the input signal and the time derivative of the command $\frac{du}{dt}$.

Figure 2.1 represents the overall bio-inspired controller architecture with the detail of the adaptive FF control to compute the command.

2.2 Machine learning model

Equation 2.1 assumes a linear relationship between the input states $\overrightarrow{x}$ and the motor commands $\overrightarrow{u}$. However, this assumption is overly simplistic for many real-word

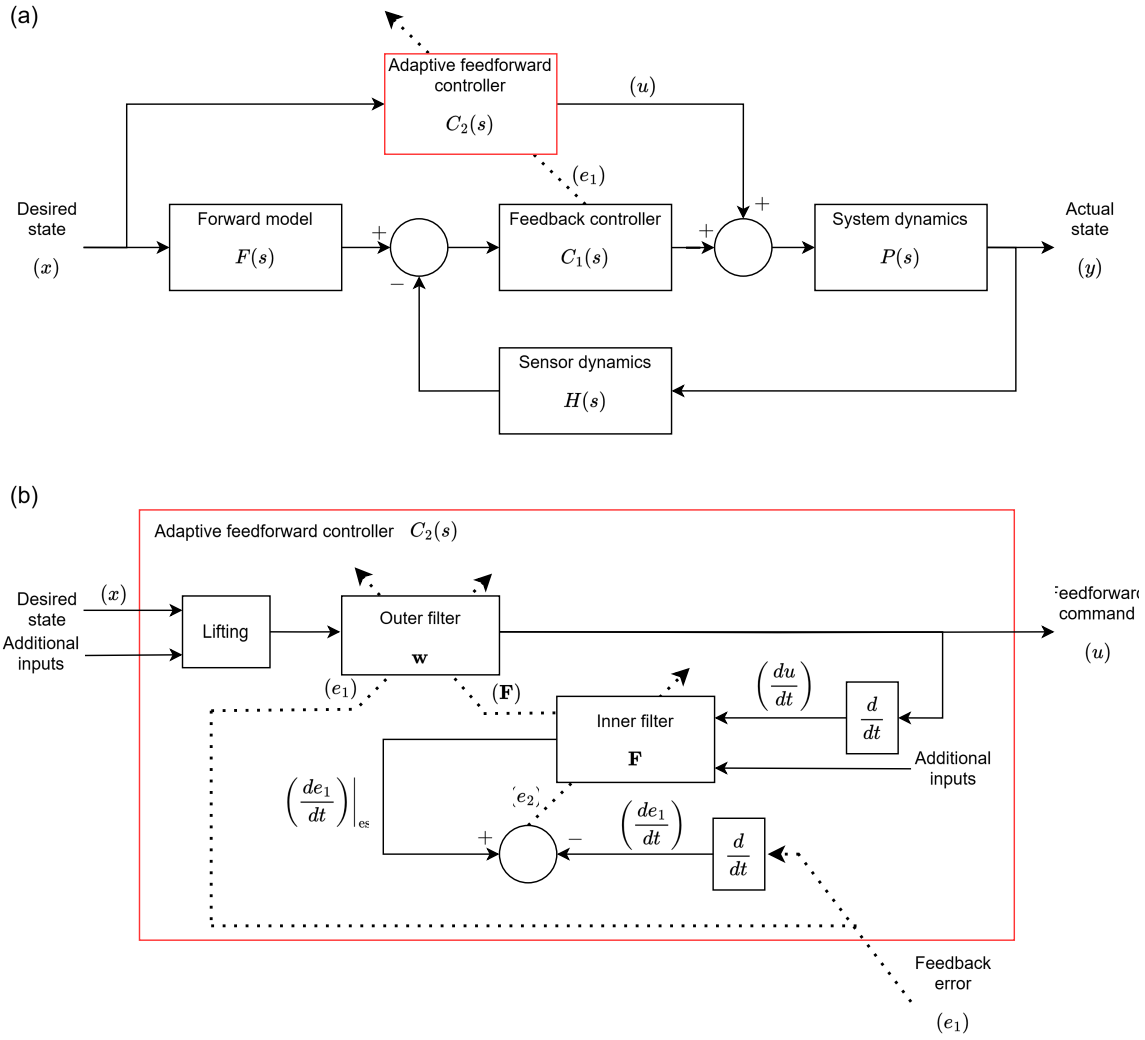


Figure 2.1: **Adaptive controller architecture with implicit feedforward supervision model**. (a) General architecture of the controller combining feedforward and feedback control to track a desired state x. The adaptive FF controller $C_2(s)$ generates a motor command from x. (b) Detail of $C_2(s)$. The difference e_1 between the actual state y and x is provided to the outer filter to adjust the weights w and to the inner filter which computes its derivative. The difference e_2 between the actual and the predictive derivative $\frac{de}{dt}$ is used to adjust the sensitive derivatives $F = \frac{de}{du}$. *From B. Campbell (2023), unpublished.*

systems and biological mechanisms. In the non-linear case, the process of adjusting the weights w_{ij} becomes more complex. To address this challenge, ML models have proven to be effective tools for capturing intricate and non-linear relationship within data.

ML algorithms can be broadly classified into 3 categories: supervised learning, unsupervised learning and reinforcement learning [21]. The model developed in this thesis and that was applied as the outer filter is called a multilayer perceptron (MLP)

[22], and utilizes the FB error e as a guide - or supervisor - to adjust the weights w_{ij} between nodes of the network. However, unlike traditional supervised learning approaches, it does not rely on labeled datasets containing known input-output pairs. Indeed, for each input x of a supervised learning algorithm, the corresponding output r (i.e. reference) is known. Over time, the algorithm learns to map the input to the output by minimizing the error between r and the actual output y of the model. In the algorithm of this project, the appropriate command u (output) needed to track the desired input x (input) is unknown.

Furthermore, data-driven neural networks typically require large amounts of structured data that might not be available. In this work, continuous real-time signals are provided to the MLP. They can represent various parameters of the desired state, such as position, speed, or orientation. Thus, "data" flows continuously into the system and the MLP must adapt in real-time. This approach mimics the way the CNS processes and adapts to continuous sensory inputs. This "online" adaptation overcomes the challenge of unstructured or weak datasets. In addition, it offers the opportunity for real-time hardware implementation.

The particularities of the MLP implemented in this project present a unique challenge in model development and training.

Chapter 3

Methods

In this chapter, the description of tools, methods, and workflow used during this project will be explained.

3.1 Multilayer Perceptron

MLPs are widely used in advanced control theory due to their computational simplicity and effectiveness. These fully connected networks excel in both single input, single output (SISO) and multi inputs, multi outputs (MIMO) systems.
This section covers a brief description of the common architecture of a MLP [23].

MLPs are characterised by interconnected neurons - or *nodes* - organized into cascaded *layers*. A typical MLP consists of an input layer, followed by hidden layers and an output layer. Each layer consists of computational units that effectively encode the relationship between variables, making them powerful function approximators. Within a single layer, the weighted sum of each neuron's input p_i, added with a *bias* b, is calculated:

$$z = w_1 \cdot p_1 + w_2 \cdot p_2 + ... + w_N \cdot p_N + b \tag{3.1}$$

$$z = \sum_{i=1}^{N} (w_i \cdot p_i) + b \tag{3.2}$$

The *weights* represent the strength of the interconnections between each node p_i and the output z, i.e how much influence one neuron has on another. With vectorisation, we have:

$$z = \overrightarrow{w} \cdot \overrightarrow{p} + b \qquad \text{where } \overrightarrow{w} \text{ is the vector of weights} \tag{3.3}$$

This sum is then passed through an *activation function* f, yielding the layer's output: $a = f(z)$. It can change between each layer and is typically chosen between rectified linear unit ($ReLU$), sigmoid, and hyperbolic tangent ($tanh$) to limit the amplitude of the output into a certain range. The weights vector $\overrightarrow{w}$ and the bias b are both adjustable and updated during a *training* process. Figure 3.1 illustrates the relationship between the inputs and the outputs of a SLP.

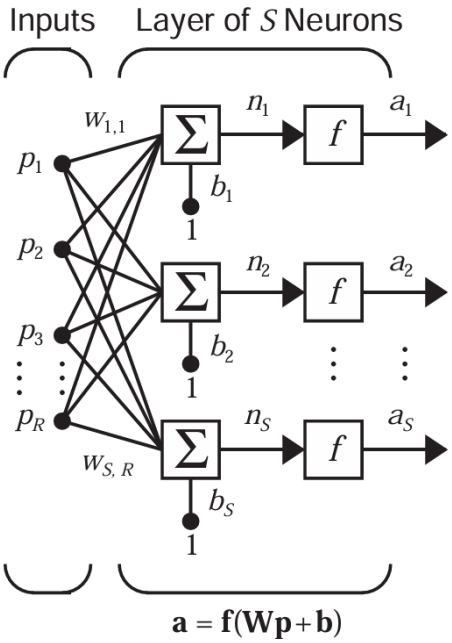


Figure 3.1: **Structure of the first hidden layer with S neurons.** Each p_i input is connected to each neuron. If R is the size of the input vector and S the size of the output vector, W is a matrix of size $R*S$ containing weights w_{ij}. *From Hagan (1999) [23].*

In multilayer configurations, each layer's outputs serves as inputs for the subsequent layer. An example of the architecture of a typical MLP is given in Figure 3.2.

3.2 Backpropagation

An effective training process is known as the *backpropagation* algorithm [24]. The process is based on the gradient descent method, where weights and biases are updated iteratively in the direction of the minimum of the loss function, and thus the error:

$$W_l^{k+1} = W_l^k - \mu \cdot \frac{dL}{dW_l^k}, \tag{3.4}$$

$$b_l^{k+1} = b_l^k - \mu \cdot \frac{dL}{db_l^k} \tag{3.5}$$

where μ is the learning rate, k the index of the training iteration and l the index of the layer.

Using the network of Figure 3.2 as an example, and considering that y is the output of the adaptive model in the outer filter, so $u = y = a_3$, we can easily find the relationship between L and W_3 for the output layer:

$$\frac{dL}{dW_3} = \frac{dL}{de} \cdot \frac{de}{du} \cdot \frac{du}{dW_3} \tag{3.6}$$

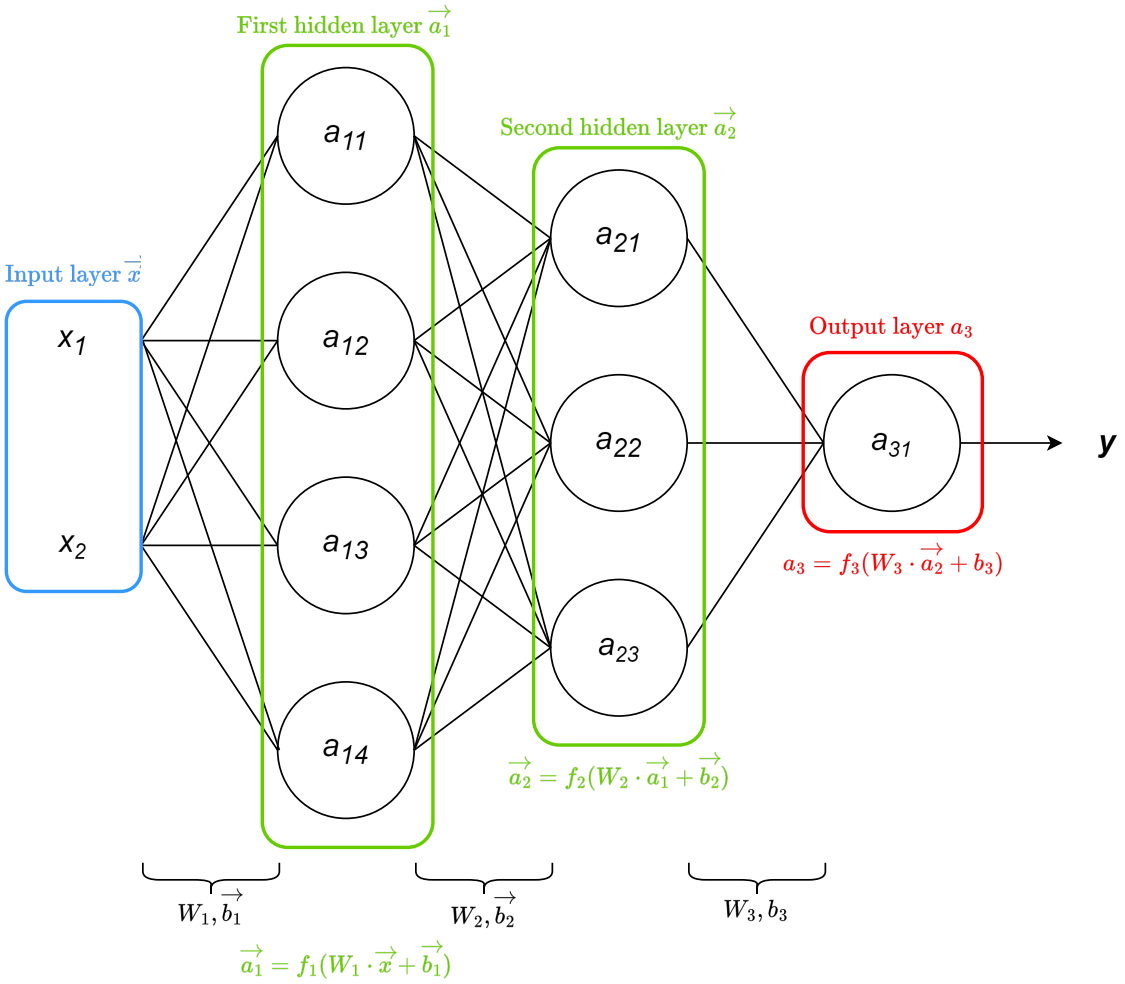


Figure 3.2: **Example of a fully connected neural network.** The MLP contains 3 layers: 2 hidden layers and an output layer. Each vector $\vec{a_l}$ contains the result $a_{ln} = f(z_l)$ of each node. W_x are matrices of scalar weights.

$$= e \cdot \frac{de}{du} \cdot \frac{du}{dW_3} \qquad \text{using equation 2.2} \qquad (3.7)$$

$$= e \cdot \frac{de}{du} \cdot a_2 \cdot f'(z_3) \qquad \text{using properties of derivative} \qquad (3.8)$$

and

$$\frac{dL}{db_3} = e \cdot \frac{de}{du} \cdot f'(z_3) \qquad \text{using properties of derivative} \qquad (3.9)$$

Because a network is a sequence of layers, each output being the input of the next layer, it is possible to calculate the derivative of each parameters $\frac{dL}{dW_l}$ and $\frac{dL}{db_l}$ by applying chain rule of differential calculus[1].

[1] f is differentiable and continuous

The learning rate μ controls the step size that the optimisation algorithm takes toward the minimum of the loss function, i.e how much $\overrightarrow{w}$ and b are adjusted during each training iteration. If μ is too high, it can cause the model to overshoot the optimal parameters w and b, leading to divergence and oscillations. If it is too low, the model learns very slowly, leading to inefficiency and increased computational costs. Additionally, it can get stuck in local minima due to the very small steps it takes.

In a MLP, during the forward pass, the signal flows from the input layer to the output layer. In the backward pass, using backpropagation and the chain rule of calculus, partial derivatives of the error concerning the various weights and biases are back-propagated through the MLP.

3.3 Procedure

This section details the development and testing of the adaptive FF controller integrating a multilayer perceptron (which I will now refer as "FANN controller"). I will compare its performance with the "benchmark controller" developed previously by B. Campbell, that assumes linear relationship between u and x (section 2.1).

The code for the adaptive FF controller was written in MATLAB, and simulations were performed with MATLAB/Simulink.

3.3.1 Model architectures

The success of a MLP depends on selecting appropriate *hyperparameters* which include the number of neurons per layer, the total number of layers, the activation function for each layer, and others depending on the complexity of the task. If the network is too complex, i.e there are too many nodes or layers, the model may learn not only the underlying patterns but also the noise and random fluctuations in the data (overfitting). However, if it is too simple, it will not be able to effectively capture the complexity of the relation between x and u (underfitting). It is then crucial to strike a balance in the network's architecture, especially to enhance its generalisation properties and how it deals with local minima.

There are no general guidelines for defining the best combination of hyperparameters. I then selected 10 distinct configurations (Table 3.1) to test for each dynamic. The learning rates (LR) were selected from a range of 0.1 to 0.001. Both $sigmoid$ and $tanh$ functions were used as activation functions between the input layer and the first hidden layer, as well as between each hidden layer. A linear function is set between the last hidden layer and the output layer. Model architectures vary from 1 to 3 layers, with each layer containing 3, 5, 15, 32, or 50 neurons. I observed that a larger number of nodes or layers tended to overfit.
I evaluated the benchmark controller's performance using the same strategy, varying the inner and outer LR to create different configurations (Table 3.2).

Table 3.1: **Hyperparameters configurations for FANN controller**. It includes the number of layers and nodes per layer (*Layers*), the learning rates of outer and inner filters (*LR outer / inner*) and the activation function (*Activation*) used between the hidden layers.

Model index	Layers	LR inner	LR outer	Activation
1	[5]	0.001	0.01	Sigmoid
2	[15]	0.01	0.1	Sigmoid
3	[15]	0.001	0.001	Tanh
4	[50]	0.001	0.1	Tanh
5	[15 15]	0.001	0.01	Sigmoid
6	[5 5]	0.01	0.1	Sigmoid
7	[5 5]	0.001	0.001	Tanh
8	[15 32 15]	0.001	0.01	Sigmoid
9	[3 3 3]	0.01	0.1	Sigmoid
10	[3 3 3]	0.001	0.001	Tanh

Table 3.2: **Learning rates configurations for benchmark controller**. It includes the learning rates of the outer and inner filters (*LR outer / inner*).

Model index	LR inner	LR outer
1	0.001	0.1
2	0.001	0.01
3	0.001	0.001
4	0.01	0.1
5	0.01	0.01
6	0.01	0.001
7	0.1	0.1
8	0.1	0.01
9	0.1	0.001

3.3.2 Evaluation metrics

During simulation, I measured the exponential moving average (EMA) of the mean squared error (MSE). EMA is a statistical technique that calculates the average MSE in a specific window of data points. It smooths out short-term fluctuations and provides a dynamic view of performance over time. The EMA formula is:

$$\text{EMA}(t) = \alpha * e(t) + (1 - \alpha) * \text{EMA}(t - 1) \tag{3.10}$$

where $e(t)$ is the feedback error at time t, and α is a smoothing factor set as 0.999.

During simulations, several types of results were observed, which I have divided into 3 categories:

- Valid tests: simulations for which the MSE reached a steady-state, and thus for which the output signal were able to stabilise.

- Invalid tests: simulations for which the MSE was diverging or oscillating, resulting in the instability of the output signal.
- Failure: simulations that stopped prematurely due to an MSE signal diverging to infinity or due to intrinsic Simulink errors (described in chapter 5).

For valid simulations, I reported the value of the steady-state (SSV) and the settling time, i.e the time required for the signal to reach and stay within a specified tolerance band that I set as : $SSV \pm 0.05$. Finally, I also reported the execution time of each simulation.

3.4 System dynamics

The controller shown is Figure 2.1 uses FB and FF signals to control the system's dynamic $P(s)$, also called the plant model. It describes how the system behaves under various conditions, such as changes in the desired state, additional inputs, or disturbances. Although the controller has previously shown effectiveness in managing linear dynamics using the approach described in 2.1, integrating an ML model could enhance its ability to handle non-linear dynamics.

In this section, I will detail the dynamics chosen to test the performance of the FANN and benchmark controllers. For non-linear systems, I concentrated on 3 specific models: the pendulum, the non-linear damping spring mass [25, 26], and an adapted version of the Lotka-Volterra model. The systems were selected to cover a wide range of non-linear characteristics [27].
Note that for all the state-space models described below, f represents the input of the plant model, x the state vector and y the output signal.

3.4.1 Linear

The linear dynamics chosen are described by a second-order differential equation: $\ddot{y}(t) + a\dot{y}(t) + by(t) = cf(t)$. It can also be represented in the frequency domain by a transfer function $P(s) = \frac{Y(s)}{F(s)} = \frac{1}{as^2+bs+1}$.
Following the recommendations of B. Campbell and based on his unpublished research, I set $a = \frac{1}{80}$ and $b = \frac{1}{10}$. These values are actually consistent with the bandwidth range used in the paper [8].

The Simulink Continuous library allows direct implementation of transfer functions (Simulink block), enabling faster simulations and simple implementation.

3.4.2 Pendulum

The pendulum system is defined by a second-order differential equation:

$$\ddot{\theta}(t) + \frac{g}{l} sin(\theta(t)) = f(t) \tag{3.11}$$

where $\theta(t)$ is the angular displacement, $f(t)$ the external force applied, g the gravitational acceleration and l the pendulum's length. The non-linearity arises from the sinusoidal term for large angles.

State-space representation (given that $x_1 = \theta$ and $x_2 = \dot{\theta}$):

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= f - \frac{g}{l} sin(x_1) \\ y &= x_1 \end{aligned}$$

To simulate real-world conditions, I fixed $g = 9.81$ and $l = 2$.

3.4.3 Non-linear damping spring mass (NLDSM)

The NLDSM dynamic is defined by a second-order differential equation:

$$m\ddot{y}(t) + c\dot{y}(t) + k_1 y(t) + k_2 y^3(t) = f(t) \tag{3.12}$$

The system models the motion of a mass attached to a spring. Here, $y(t)$ represents the displacement of the mass m, c is the damping coefficient, k_1 and k_2 are stiffness coefficients, and $f(t)$ the external force applied to the system.

State-space representation (given that $x_1 = x$ and $x_2 = \dot{x}$):

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= \frac{1}{m} f - \frac{c}{m} x_2 - \frac{k_1}{m} x_1 - \frac{k_2}{m} x_1^3 \\ y &= x_1 \end{aligned}$$

I set $\frac{c}{m} = 2$, $\frac{k_1}{m} = 1$, $\frac{k_2}{m} = 1$ and $\frac{1}{m} = 1$. The high damping coefficient reduces oscillations, while the large coefficient $\frac{k_2}{m}$ ensures that the cubic non-linarity will become significant to overcome the linear regime of the system. These values are inspired on the Duffing system parameters used in [27].

3.4.4 Adapted Lotka-Volterra dynamic

The following state-space model represents a dynamic adapted from the Lotka-Volterra model, which traditionally describes the interaction between two species in an ecosystem and their evolution over time. However, my approach differs in that I only consider a single input state, rather than two distinct variables. Therefore, I have formulated the third dynamic as follows:

$$\dot{x_1} = \alpha x_1 - \beta x_1 x_2 + f \tag{3.13}$$

$$\dot{x_2} = \delta x_1 x_2 - \gamma x_2 \tag{3.14}$$

$$y = x_2 \tag{3.15}$$

I set $\alpha = 1$, $\beta = \frac{1}{60}$, $\delta = \frac{1}{100}$ and $\gamma = \frac{1}{50}$. The parameters were determined through trail and error. High values resulted in simulation failures, so smaller coefficients were chosen to maintain small state changes of the system's dynamic.

3.5 Limitations

The feedback controller integrates a PID-filtered signal to stabilize the output and ensure robustness to noise and disturbances. It is then crucial that the PID parameters are tuned appropriately, which can be done automatically by Simulink. However, while the software tuning worked effectively for linear systems and some non-linear ones, it failed for more complex non-linear dynamics, limiting the choice of applicable systems.
Additionally, the system's current configuration only considers a single feedback error. However, for proper MIMO systems, more than one error should be considered.

The execution time for the simulations may depend on the specifications itself of my computer, which can change over time due to factors like system load or resource availability.

Chapter 4

Experimental Results

4.1 Settings

The simulation time was set to 200 seconds. This duration was chosen as it strikes a balance between allowing sufficient time for the model to adapt while maintaining reasonable computational efficiency. The sampling interval was fixed at 0.05 seconds. The input of the controller is a sinusoidal signal with an amplitude of 0.8 and a frequency of 1.5 Hz. To evaluate the consistency of the results, I conducted 5 simulations for each model, resulting in 45 simulations for the benchmark controller and 50 for the FANN controller.

For the benchmark controller, the outer weights w_{ij} of the FF controller were initialized by random values drawn from the uniform distribution in the interval (0,1), and normalized by the factor $K = 50$ (section 2.1). The learning rates (LR) were kept fixed, as indicated in Table 3.2.
For the FANN controller, I tested 3 different random distributions to initialise the outer weights: uniform, normal, and a restricted range of (-3,3) excluding small weights between -1 and 1. The third option turned out to produce the best results in terms of stability of the MSE and output signals, consistency between simulations of a same model, and overall performance across all simulations. Models that persistently gave inconsistent results have been excluded from the evaluations presented below.

4.2 Evaluation of performance

For each dynamic and controller combination, the MSE signal of each model will be used as a baseline to evaluate the performance of the controller under a specific hyperparameter setting. By tracking MSE during training, we can gain insights into how well the controller is learning and generates an output signal aligning with the reference one. Adaptation capacities of a controller are validated if the error is minimised, which can be visualised by MSE stabilising at a small value below 0.05 and input and output signals matching. If this does not happen within the considered time frame, the adaptation process takes too long or is impossible.

A decrease in MSE indicates that the controller is improving its predictions. The steeper the slope of the curve, the faster the controller is learning. Whenever a curve reaches a steady-state, the controller's output becomes a stable, periodic wave with relatively consistent amplitude, frequency, and phase. Finally, if the MSE increases, the controller's output is unstable, showing oscillations or divergence towards infinity.

4.2.1 Linear dynamic

First, this section covers the behaviour of both controllers under linear dynamic.

Benchmark controller

In figure 4.1, we can see that the MSE of models 1,4,7 and 2,5,8 are decreasing and stabilising at 0, but more rapidly for the first 3. This means that all models allow adaptation, faster with a high outer LR of 0.1 and regardless of the value of the inner LR. The remaining models, sharing the same low value of outer LR (0.001), could not provide satisfactory results within the simulation timeframe as their MSE were still decreasing (models 6,9) or increasing over time (model 3).
The average execution time across the 45 simulations was 2.32$\pm$ 0.85 minutes.

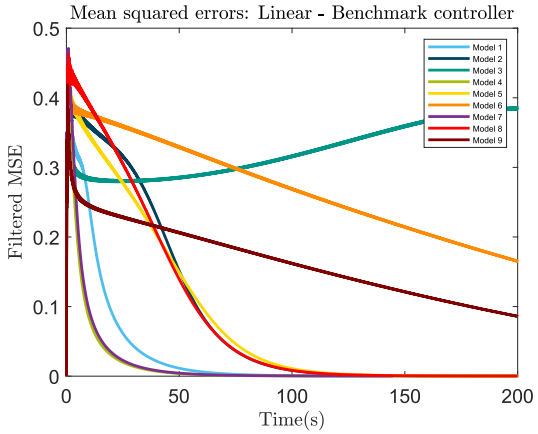


Figure 4.1: **MSE signals for 9 distinct simulations performed with the benchmark controller for linear dynamic**. Settling time for valid models (Mx) in seconds: (M1) 80.25, (M2) 125.4, (M4) 63.7, (M5) 130.05, (M7) 63.7, (M8) 125.5. The curves reach a null steady-state, meaning that the reference tracking performance is validated. Models with higher outer LR (1,4,7) exhibit a more pronounced exponential decrease. Models 3,6, and 9 failed to adapt within the simulation timeframe.

FANN controller

Model 4 was excluded from evaluation due to its MSE diverging to infinity. In figure 4.2, the MSE of models 2,5,6,8, and 10 reach steady-states below 0.05, while the

MSE of model 3,7 and 9 are stuck in local mimima. The MSE of model 1 starts to increase from 120 seconds, leading to instability in the output signal. All models, except model 9, exhibit a sharp peak in MSE at the beginning, suggesting that the initial prediction of the controller is inaccurate. This inaccuracy is linked to the random initialisation of the weights for both the inner and outer filters. The higher the peak, the longer it takes for the controller to stabilise.
There is no specific combination of hyperparameters that gives good results. In fact, for well-performing models, it can go from 1 to 3 layers, from 3 to 32 neurons per layer, and consider several LR values between 0.001 and 0.1.

The average execution time between simulations is 6.34±0.55 minutes. I noticed that the execution time increases slightly with the complexity of the network.

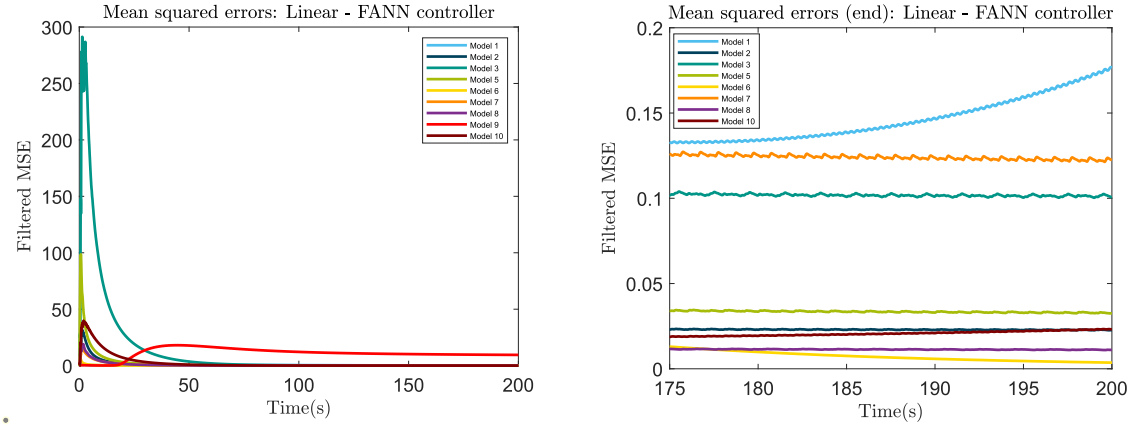
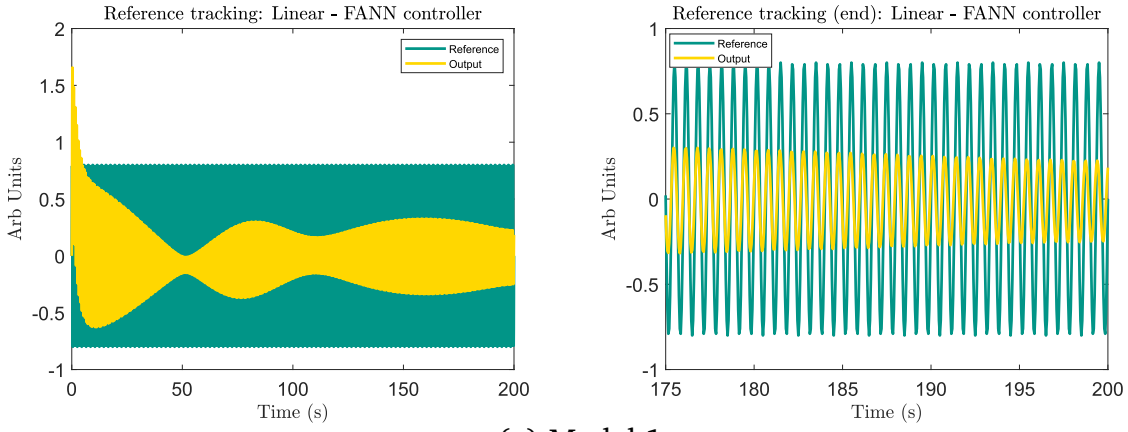


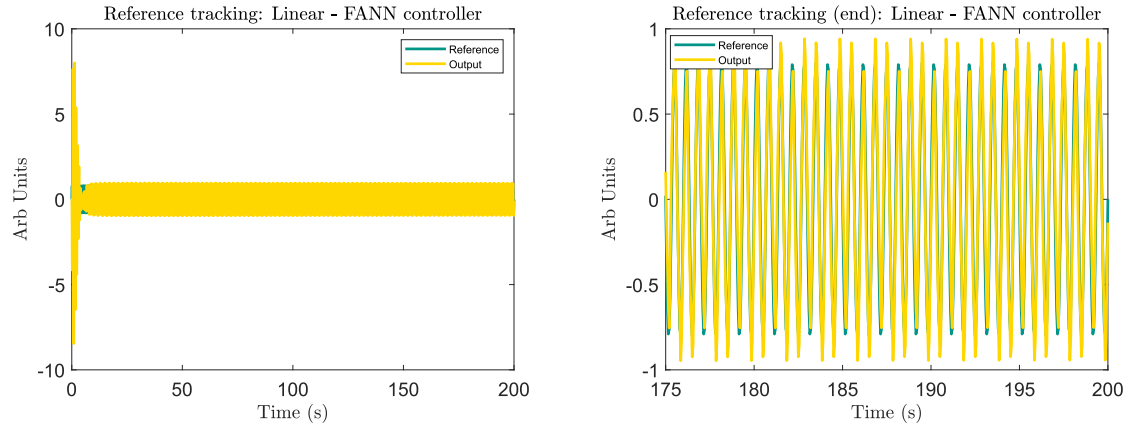
Figure 4.2: **MSE for distinct simulations performed with the FANN controller for linear dynamic during the entire simulation timeframe (left) and during the last 25 seconds (right)**. Most signals exhibit a sharp initial spike, indicating a significant error at the beginning of the control process. Over time, the MSE decreases, suggesting that the controller is gradually improving its performance.
Settling time for valid models (Mx) in seconds: (M2) 34.8, (M3) 117.5, (M5) 84.3, (M6) 32.1, (M7) 80.9, (M8) 65.0, (M9) 183.5, (M10) 89.9.
Steady-state values for valid models: (M2) 0.023, (M3) 0.012, (M5) 0.032, (M6) 0.004, (M7) 0.122, (M8) 0.011, (M9) 9.483, (M10) 0.023.

Both the reference and FANN controllers performed effectively when the plant model was linear. While it was expected for the benchmark controller, it confirms the proper functioning of the MLP algorithm for the new controller. A comparison between good and poor reference tracking performance is shown in figure 4.3.

I will now continue in the following sections with the primary focus of this project, which is the study of the controller's behavior under non-linear dynamics, and I will start with the pendulum one.



(a) Model 1



(b) Model 2

Figure 4.3: **Reference tracking performance for models 1 and 2 during the entire simulation timeframe (left) and during the last 25 seconds (right).** (a) The amplitude of the output signal of model 1 is varying (left) and cannot match that of the input signal, indicating poor tracking performance. The MSE was above 0.12 and started to increase from 180 sec. (b) The model 2 output signal quickly matches the input signal at 34.8 seconds (left), with only minor differences in amplitude (right), showing good adaptability. MSE of model reached a steady-state of 0.023.

4.2.2 Pendulum dynamic

Benchmark controller

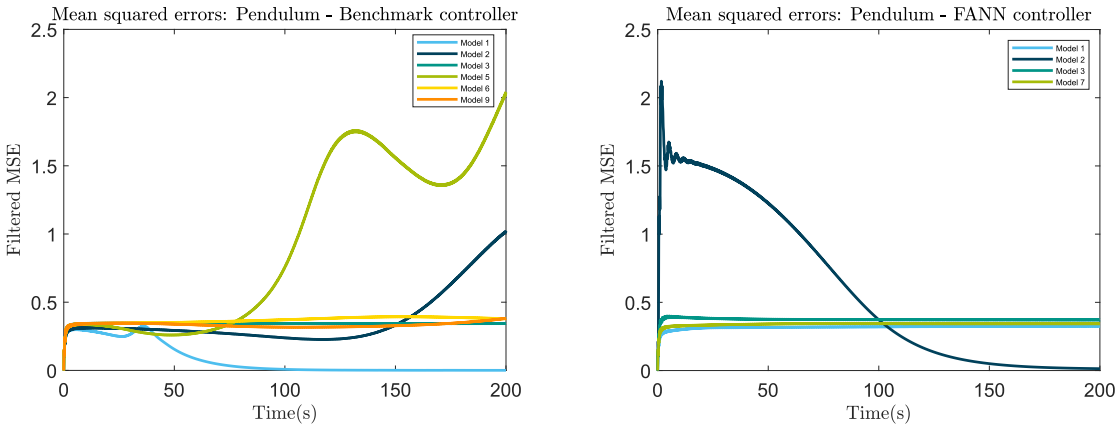
Models 4, 7, and 8 were excluded from the evaluations due to a divergent error towards infinity or inconsistency between results.

Among all the MSE signals depicted in figure 4.4a., only model 1 reached a steady-state value of zero. In this model, a high LR in the outer filter (0.1) was compensated for by a low LR in the inner filter (0.001). Furthermore, reducing the outer LR to 0.01 appeared to increase the MSE to values above 1 (models 2 and 5), while further decreasing it to 0.001 seemed to trap the MSE in a local minimum around 0.35 (models 3,6,9). The execution time for the 35 simulations (excluding the failed ones) was

high, with an average of 17.41 ± 4.91 min, and lower efficiency for the models with the lowest inner LR value.

FANN controller

60% of the simulations failed, resulting in only four models for which the simulations were completed. Only model 2 presented successful results, while the other immediately got stuck in local minima around 0.35 (figure 4.4b.). The architecture of model 2 is a single hidden layer with 15 neurons, a sigmoid activation function between the input and hidden layers, a LR of 0.01 for the inner filter, and a LR of 0.1 for the outer filter. The average execution time was 28.05±2.36 minutes.



a b

Figure 4.4: **MSE for distinct simulations performed with the benchmark controller (left) and the FANN controller (right) for pendulum dynamic.** a. Settling time for models Mx in seconds: (M1) 128.85 (M3) 25.25 (M6) 20.85 (M9) 26.35. While MSE of models 3,6,9 get stuck in a local minimum of 0.35±0.01, the MSE of models 2 and 5 increase over time, leading to instability in the output signals. b. The MSE of model 2 slowly decreases toward 0.01, reached at 183.25 seconds. The others signals get stuck at a local minimum of 0.35±0.02 at 15.5 seconds.

Under pendulum dynamic, only one model demonstrated good reference tracking performance for both controllers.

4.2.3 Adapted Lotka-Volterra dynamic

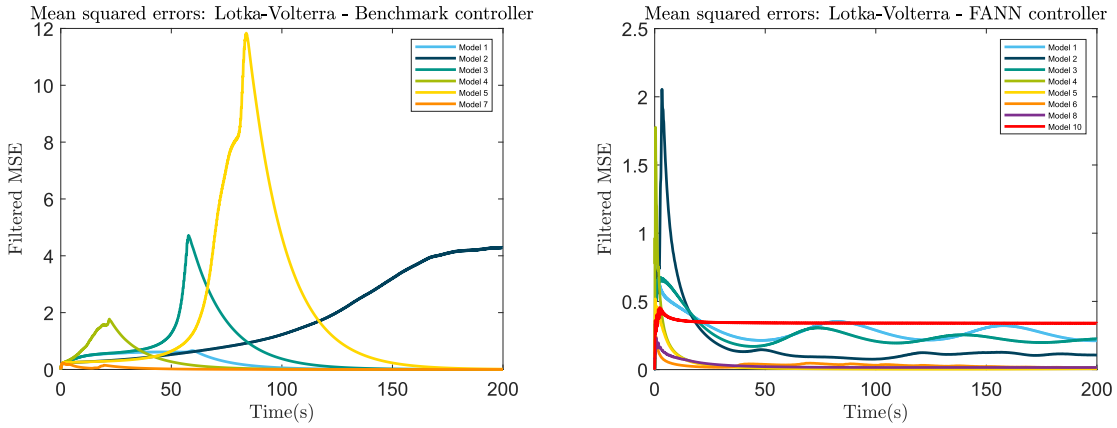
Now examining how both controllers managed the adapted Lotka-Volterra dynamic.

Benchmark controller

Between the MSE of the models presented in figure 4.5a., all converge to small values below 0.05, except for model 2 for which the MSE increases towards 4.15. The fastest adaptation is achieved with high values of inner and outer LR (model 7), and settling time increases with the decrease of the inner LR.
The mean execution time for valid simulations was 6.34±0.57 minutes.

FANN controller

Only considering models with consistent behaviour across simulations, models 4,5,6, and 8 successfully reached zero steady-state (figure 4.5b), indicating that the controller effectively tracks the input signal. Various combinations of hyperparameters could be effective, including either 1,2 or 3 layers. However, it appears that choosing 2 layers with an outer LR of 0.1, as in model 4, is preferable to achieve a balance between performance and computational efficiency. A lower LR tends to introduce more variability between simulations, making the results less interpretable.
The average execution time was 13.62± 6.67 minutes, with an increase in the architectural complexity of the MLP resulting in longer execution times.



a b

Figure 4.5: **MSE for distinct simulations performed with the benchmark controller (left) and the FANN controller (right) for adapted Lotka-Volterra dynamic.** a. The MSE of model 2 slowly increases over time. The MSE of the other models reach steady-states closed to zero. Settling-time for models Mx presented (in seconds): (M1) 136.55, (M3) 131, (M4) 95.05, (M5) 156.85, (M7) 84.45 and their steady-state values: (M1) 0.003, (M3) 0.038, (M4) 0.002, (M5) 0.015, (M7) 0.002. b. Models 4, 5, 6, and 8 demonstrate strong performance, with their MSE rapidly converging to zero and stabilizing at 62, 118.6, 110, and 106.05 seconds, respectively. The MSE of models 2 and 10 settle around 0.19 and 0.33, while models 1 and 3 exhibit oscillatory behavior around 0.23.

After successfully identifying several models that yielded satisfactory results for the adapted Lotka-Volterra model, let's proceed with our investigation of the non-linear damping spring-mass dynamic.

4.2.4 Non-linear damping spring mass dynamic

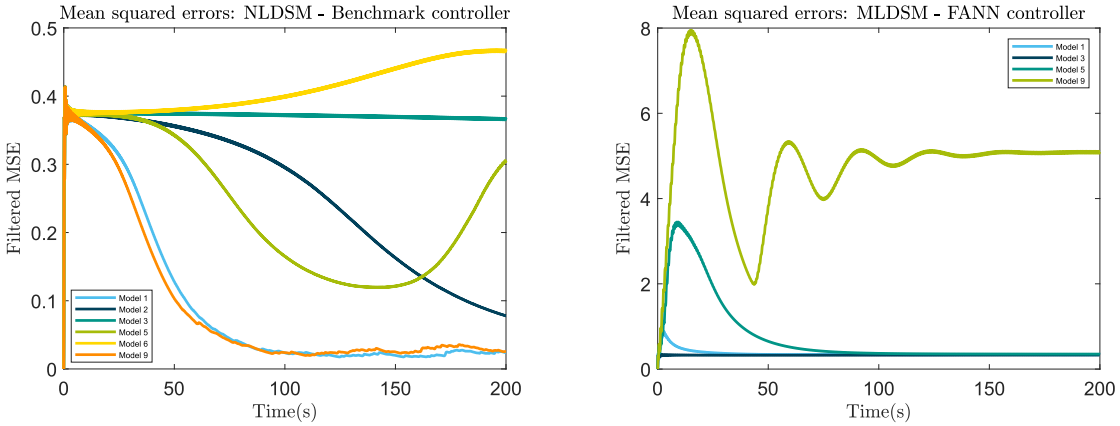
Figure 4.6 shows the MSE curves for the valid simulations performed on the reference and the FANN controller.

Benchmark controller

Simulations for models 4,7 and 8 failed. Adaptation was achieved using configurations 1 and 9, which have opposite LR values for the outer and inner filters, indicating that the fast weight adaptation in one filter is balanced by slow adaptation in the other.
I could have extended the simulation time to see if the MSE of simulation 2 could stabilise, but since I already had 2 tests performing well, I avoided additional simulations, knowing that the test took 27.22 minutes to run. Furthermore, increasing the inner LR above 0.01 significantly reduced execution time from an average of 23.15 $\pm$4.00 minutes for the first 3 tests to 2.81$\pm$0.13 minutes for the last 3.

FANN controller

For the pendulum dynamics, 60% of the simulations failed. Only models 1,3,5 and 9 produced results, even if showing poor tracking performance with a MSE signal stabilizing at 0.35 for the first 3 models and at a very high value of 5 for model 10.



a b

Figure 4.6: **MSE for distinct simulations performed with the benchmark controller (left) and the FANN controller (right) for NLDSM dynamic.** a. The MSE for models 1 and 9 starts at 0.38 and then decreases rapidly before leveling off at 53 seconds at a relatively low MSE of 0.023. They slightly oscillate from halfway through the simulation. While the model 3 signal maintains a constant MSE of around 0.4, models 6,2 and 5 are unable to stabilise within the simulation timeframe. b. Model 3 MSE stabilizes immediately at 0.33, whereas model 9 MSE exhibits damped oscillations before settling on its final value of 5.07. Settling time (in seconds) : (M1) 39.6, (M3) 4.7, (M5) 106.3, (M10) 154.05.

The FANN controller failed to adapt under NLDSM dynamic, while the benchmark controller did.

In conclusion, the FANN controller demonstrated adaptive capabilities under nonlinear dynamics for the pendulum and the adapted Lotka-Volterra systems but not for the NLDSM. In contrast, at least one model produced the desired results for the

benchmark controller. The following chapter will discuss the observed results and explore potential improvements.

Chapter 5

Discussion

This project investigates whether incorporating a machine learning model called multilayer perceptron can enhance the adaptation capabilities of a bio-inspired controller. I compared the performance of the MLP-based controller to a benchmark controller implementing a single-layer perceptron with linear activation. Both controllers incorporate a feedforward pathway generating motor commands sent to a plant model that computes an output signal that should align with the input signal. The focus is on their ability to adapt well to various system's non-linear dynamics.

5.1 Performance analysis

As stated in section 3.3.2, the adaptation performance of a controllers is validated if the MSE reaches a steady-state value below 0.05. Above this threshold, I observed delays or differences in amplitude between the input and output signals, illustrating that the controller struggles to generate a state corresponding to that desired.

Investigating the MLP-based controller's ability to adapt under non-linear dynamics revealed limitations, particularly with the non-linear damping spring mass system. First, adaptation highly depends on the specific neural network architecture, suggesting a lack of universal hyperparameter configuration that could work for any non-linear dynamics. This generalisation issue is critical in real-world applications where the underlying dynamics may be unknown, and continuously adjusting hyperparameters would be inefficient. Additionally, the MLP significantly increased computational cost, which can be a concern for applications requiring faster adaptation.
One notable issue was the MSE's tendency to become trapped in local minima. This occurs when the derivatives of the weights and biases (dW and db) are minimal or zero, because the value of the loss function is smaller than its neighboring points, but it is not necessarily the smallest solution across all possibilities. Thus, the weights and biases are not updated anymore, explaining the error stabilisation. This can happen due to overfitting, caused by an overly complex neural network or inadequate learning process.

In contrast, the benchmark controller consistently demonstrated satisfactory reference tracking performance for various learning rate configurations (Table 3.2), and especially with model 1 (LR inner = 0.001 ; LR outer = 0.1). This consistent architecture is promising in terms of generalisation.

In conclusion, the implementation of a MLP did not prove to be a reliable method for improving control performance. As a single perceptron layer may be sufficient to capture underlying data effectively, while avoiding unnecessary computation costs and local minima issues.

5.2 Limitations and challenges

Limitations of this project are probably mainly related to the design of the controller and settings of the simulations in Simulink.

A common issue I encountered was simulation failure due to the derivative of the plant model's integrator that was not finite, which might be linked to a problem of discretization [28]. Indeed, the error suggests singularities or instabilities in the system dynamic, or that the numerical solver used is inefficient in handling fast state transitions of the nonlinear system. In simulation, a finite number of steps can lead to inaccuracies in modeling. Although reducing the sampling interval to 0.01 was tested, it did not resolve the issue, implying that the error might come from the fixed solver Simulink employs. A variable-step solver could potentially better adapt to rapid state changes.
Also, I used an automatic tuning of the PID parameters offered by Simulink. However, this function struggles to find appropriate values for the parameters when subjected to non-linear dynamics, leading to poor stabilisation of the controller's output signal by the feedback controller. While manual tuning is an option, it involves a long, difficult process of trial and error, with simulations taking a considerable amount of time to run.
However, discretization or stabilisation issues don't explain why the benchmark controller performed better than the FANN controller, given that both were applied to the same plant model, PID controller, and under the same Simulink solver. It's possible that for the nonlinear dynamics chosen, assuming a linear relationship between control commands u and input states x was sufficient to handle the system. Therefore, increasing the model's complexity may lead to poor generalisation and failure to adapt, as well as local minima issues.

5.3 Potential improvements

To improve results, other machine learning methods, such as recurrent neural networks (RNNs) [29, 30, 31] or radial basis function networks (RBFNs) [32, 33, 34], can offer a more flexible framework for capturing intricate patterns between data and managing online training. RNNs are designed for sequential data and can main-

tain a state through feedback connections, making them ideal for tasks with time dependencies. This could be beneficial if we process batches of data, where RNNs' memory capabilities can better handle the continuous input states. On the other hand, RBFNs are more efficient for tasks like function approximation or interpolation. They tend to train faster than other neural networks and are less sensitive to outliers. However, similar to the MLP, these models will need to be adapted as we lack labeled data and rely solely on the feedback error for training.
Additionally, optimisation methods such as mini-batch or momentum gradient descent [35] could improve weight updates. Mini-batch gradient descent updates weights based on small subsets of data, while momentum gradient descent builds a moving average of gradients, which may accelerate convergence and help overcome local minimum issues. However, these methods also come with the risk of overshooting the optimal solution.
Finally, the current design restricts the inner filter to a single layer so that the chain rule assumption remains valid. However, exploring multilayer structures or alternative methods for updating the weights might be beneficial - because the more accurate the inner filter is, the faster and better the controller will adapt.

5.4 Conclusion

This project explored the potential benefits of using a multilayer perceptron within a bio-inspired controller. While the MLP-based controller sometimes demonstrated adaptive capabilities, it encountered challenges such as computational cost, local minima issues, sensitivity to hyperparameters and simulation failures, and therefore did not outperform the benchmark controller. Future research could explore alternative machine learning methods and optimization techniques to produce better results than the ones observed in this project. Although feedback-feedforward AI-based controllers are well-documented, the combination with implicit supervision process, a specific machine learning method, and an online training process makes this a novel and challenging application.

Bibliography

[1] S. Akesson et al. Animal movement across scales. *Oxford University Press*, 9:151–178, 2014.

[2] A.J. Ijspeert P. Ramdya. The neuromechanics of animal locomotion: From biology to robotics and back. *Science Robotics*, 8(78), 2023.

[3] R. D. Beer E. J. Izquierdo. From head to tail: A neuromechanical model of forward locomotion in caenorhabditis elegans. *Philosophical transactions of the Royal Society of London. Serie B, Biological sciences*, 373:1758, 2018.

[4] N. Cohen J.H. Boyle, S. Berri. Gait modulation in c. elegans: An integrated neuromechanical model. *Frontiers in computational neuroscience*, pages 6–10, 2012.

[5] H.G. Krapp B.J. Hardcastle. Evolution of biological image stabilization. *Current Biology*, 9(20):1010–1021, 2016.

[6] H.G. Krapp. How a fly escapes the reflex trap. *Nature Neuroscience*, 18(9):1192–1194, 2015.

[7] G. Maimon A.J. Kim, J.K. Fitzgerald. Cellular evidence for efference copy in drosophila visuomotor processing. *Nature Neuroscience*, 18(9):1247–1255, 2015.

[8] H.G. Krapp B.P. Campbell, H-T Lin. A control engineering perspective on the advantages of efference copies. *bioRxiv*, 2023.

[9] P.J. Antsaklis. Neural networks for control systems. *IEEE transactions on neural networks*, 10(3):3–5, 1990.

[10] K.J. Hunt et al. Neural networks for control systems - a survey. *Automatica*, 6:1083–1112, 1992.

[11] V. Snàsel V.K. Ojha, A.A. Abraham. Metaheuristic design of feedforward neural networks: A review of two decades of research. *Engineering Applications of Artificial Intelligence*, 60:97–116, 2017.

[12] M.A.N. Coelho F.G. Martins. Application of feedforward artificial neural networks to improve process control of pid-based control algorithms. *Computers Chemical Engineering*, 24(2):853–858, 2000.

[13] S.J.Plathottam et al. A review of artificial intelligence applications in manufacturing operations. *Journal of Advanced Manufacturing and Processing*, 5(3), 2023.

[14] R.A. Adebayo et al. Ai-enhanced manufacturing robotics: A review of applications and trends. *World Journal of Advanced Research and Reviews*, 21(3):2060–2072, 2024.

[15] J.Athavale et al. Ai and reliability trends in safety-critical autonomous systems on ground and air. *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, pages 74–77, 2020.

[16] O. Illiashenko et al. Security-informed safety analysis of autonomous transport systems considering ai-powered cyberattacks and protection. *Entropy*, 25(8):1123, 2023.

[17] X.B.Peng et al. S. Song, Ł. Kidziński. Deep reinforcement learning for modeling human locomotion control in neuromechanical simulation. *NeuroEngineering Rehabil*, 18(126), 2021.

[18] A. Faisal N. Wannawas, M. Subramanian. Neuromechanics-based deep reinforcement learning of neurostimulation control in fes cycling. *2021 10th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 381–384, 2021.

[19] B. Campbell. Implicit adaptation notes. *Unpublished*, 2023.

[20] D.B. Tweed M.N. Abdelghani, T.P. Lillicrap. Sensitivity derivatives for flexible sensorimotor learning. *Neural Computation*, 20(8):2085–2111, 2008.

[21] L.F. Abbott P. Dayan. Theorical neurosscience - computational and mathematical modeling of neural systems. *MIT Press*, 8-9, 2005.

[22] Ruth Vang-Mata. Multilayer perceptrons - theory and applications. *Nova Science Publishers*, 1, 2020.

[23] Demuth M.T. Hagan, H.B. Neural networks for control. *Proceedings of the 1999 American Control Conference*, 3:1642–1656, 1999.

[24] R. Golden Y. Chauvin D.E. Rumelhart, R. Durbin. Backpropagation : The basic theory. *Psychology Press*, pages 1–34, 1995.

[25] U. Demircioğlu. H. Bakır. Artificial intelligence-based position control: reinforcement learning approach in spring mass damper systems. *Physica Scripta*, 99(4), 2024.

[26] JG. Barto J.T. Buckingham, J.C Houk. Controlling a nonlinear spring-mass system with a cerebellar model. *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 1–6, 1994.

[27] H. Robinson et al. Physics guided neural networks for modelling of non-linear dynamics. *Neural Networks*, 154:333–345, 2022.

[28] A. A. Zaher. On the discretization of continuous-time chaotic systems for digital implementations. *Journal of Physics: Conference Series*, 1141(1), 2018.

[29] S. Yildirim. Design of adaptive robot control system using recurrent neural network. *Journal of Intelligent and Robotic Systems*, 44:247–261, 2005.

[30] Y. Fang T. WS. Chow. A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics. *IEEE transactions on industrial electronics*, 45(1):151–161, 1998.

[31] M.M Gupta L. Jin, P.N. Nikiforuk. Dynamic recurrent neural networks for control of unknown nonlinear systems. *J. of Dynamic Systems, Measurements and Control*, 144(4):567–576, 1994.

[32] E. Burdet A. Kadiallah, D.W. Franklin. Generalization in adaptation to stable and unstable dynamics. *Public Library of Science San Francisco, USA*, 7(10), 2012.

[33] J. Liu. Radial basis function (rbf) neural network control for mechanical systems: design, analysis and matlab simulation. *Springer Science & Business Media*, pages 1–24,71–92, 2013.

[34] H. Yu et al. Advantages of radial basis function networks for dynamic system design. *IEEE Transactions on industrial electronics*, 58(12):5438–5450, 2001.

[35] S. Sun et al. A survey of optimization methods from a machine learning perspective. *IEEE Transactions on Cybernetics*, 50(8):3668–3681, 2020.