

Universidad Nacional
de Educación a Distancia

Escuela Técnica Superior de
Ingeniería Informática



Universidad
Complutense de Madrid

Facultad de
Informática

AIRCRAFT DYNAMICS IDENTIFICATION BASED ON MULTIPLE NEURAL NETWORK MODELS

David Matilla Bastero

Director/a: Matilde Santos Peñas
Co-director/a: Ernesto Aranda Escolástico

Trabajo de Fin de Máster

Máster Universitario
en Ingeniería de Sistemas y de Control

[2024/Septiembre]

Abstract

Multiple neural network architectures have been utilized to identify the dynamics of a fighter aircraft. Feedforward Neural Network (FNN) and two types of Recurrent Neural Networks (RNN), LSTM RNN and Clock-work RNN (CW-RNN), were used. A deep learning design pipeline was created using MATLAB framework, with a synthetic data generator developed based on the ADMIRE fighter simulator. In fact, an original implementation of the CW-RNN was coded and validated within the MATLAB environment. Different configurations of the previous neural network architectures were tested and one candidates for each kind of network was selected. Then, the advantages and disadvantage of each model were commented from an aerospace application perspective. Finally, valid models were found for all the proposed architectures. Nevertheless, CW-RNN was confirmed as the most promising solution due to its recurrent structure together with its smaller size and reduced computational requirements compared to LSTMs.

Keywords: Neural Identification Model, Neural Networks, Fighter Aircraft, Aerospace, Feedforwad Neural Network (FNN), NARX, Recurrent Neural Network (RNN), LSTM, Clockwork RNN (CW-RNN), ADMIRE, MATLAB, Simulink, Deep-Learning.

Resumen

El objetivo del proyecto es identificar la dinámica de un avión de combate utilizando redes neuronales. Para ello, diferentes architecturas han sido evaluadas. La arquitectura "Feedforwad Neural network" (FNN) y dos tipos de red neuronal recurrente, las LSTM y las Clockwork RNN, han sido analizadas.El estudio ha sido realizado utilizando el entorno de aprendizaje profundo (Deep Learning) proporcionado por MATLAB. Usando ese "framework", Se ha elaborado una "pipeline" de diseño en la que se incluye un generador de datos sinéticos basado en el simulador de vuelo ADMIRE. Además, se ha implementado y validado la red neuronal de tipo CW-RNN como clase de MATLAB. Varias configuraciones de las mencionadas redes neuronales han sido estudiadas y un candidato de cada tipo ha sido elegido. A continuación, se han analizado las ventadas y desventajas de cada tipo de red desde la perspectiva de una aplicación aeroespacial. Por último, se logró encontrar un modelo válido para todas las architecturas propuestas. Sin embargo, se confirmó que la arquitectura CW-RNN es la más prometedora ya que auna la capacidad de adaptación de una estructura recurrente junto con tamaños y exigencias computacionales más reducidas que las requeridas por las LTSMs.

Keywords: Modelo de Identificación Neuronal, Redes Neuronales, Cazas de Combate, aerospacial, Feedforwad Neural Network (FNN), NARX, Recurrent Neural Network (RNN), LSTM, Clockwork RNN (CW-RNN), ADMIRE, MATLAB, Simulink, Aprendizaje profundo, Deep-Learning.

Contents

1 Introduction **1**

2 State of Art **3**
2.1 Neuron Model and Multilayer Perceptron Network 4
2.1.1 Perceptron Learning Rule and Backpropagation 5
2.2 Recurrent Neural Network 7
2.2.1 LSTM Network 8
2.3 Clockwork Recurrent Neural Network 9
2.4 Neural networks within the aerospace industry 11

3 ADMIRE Model **15**
3.1 Aircraft Dynamics Model 17
3.1.1 Aerodynamic model 19
3.1.2 Engine model 19
3.1.3 Actuators 20
3.1.4 Sensors 21
3.1.5 Model uncertainties 21
3.1.6 Atmospheric model 22
3.2 Baseline Flight Control System model 22
3.2.1 Longitudinal controller 22
3.2.2 Lateral controller 23
3.3 Decoupled Aircraft Dynamics 23
3.3.1 Linearised equations 24
3.4 ADMIRE Dynamics Stability Analysis 25
3.4.1 Linear stability analysis 26

4 Simulation Environment **31**
4.1 ADMIRE Simulator 31
4.2 Synthetic data generation pipeline 32
4.3 Training Pipeline 35
4.4 Post-Processing Pipeline 37
4.5 Additional tools developed for this project 38

5 Neural Dynamics Identification **41**
5.1 FNN-based Identification Model 42
5.1.1 Study A - Pitch Rate Channel Identification Model 42

5.1.2 Study B - Longitudinal Channels Identification Model 44
5.2 LSTM-based Identification Model 49
5.2.1 Study A - Pitch Rate Identification Model 49
5.2.2 Study B - Longitudinal State Identification Model 54
5.3 CW-RNN-based Identification Model 61
5.3.1 MATLAB CW-RNN Layer Validation 62
5.3.2 CW-RNN Identification Model Analysis 64

6 Conclusions and future works **71**

A Datasets **79**
A.1 Testing Dataset 79

B Training Pipeline - Methods **85**
B.1 Load NN Architecture Method 85
B.2 Define Study Method 87

C CW-RNN MATLAB Class **91**

List of Figures

2.1 Multiple-Input Neuron diagram [Martin, 2016]. 4
2.2 Single-layer network with S neurons [Martin, 2016]. 5
2.3 CW-RNN layer architecture [[Koutník, Greff, Gomez, and Schmidhuber, 2014]] 10
2.4 Evolution of the number of articles of the AIAA tagged as "Machine Learning" and "Artificial Intelligence" 11

3.1 ADMIRE sketch where the main control effectors are denoted. Furthermore, the aircraft is also equipped with thrust vectoring capability and air brakes. [Forssell, 1963-Nilsson, and forskningsinstitut., 2005] 16
3.2 ADMIRE Lateral-Directional Eingenvalues at constant altitude and Mach = [0.25, 0.5, 0.7, 0.9, 1, 1.1, 1.2] (first Mach condition is painted in grey) 27
3.3 ADMIRE Lateral-Directional Eingenvalues at constant Mach and altitude = [1000, 3000, 6000] [m] (first altitude condition is painted in grey) 28
3.4 ADMIRE Longitudinal Eingenvalues at constant altitude and Mach = [0.25, 0.5, 0.7, 0.9, 1, 1.1, 1.2] (first Mach condition is painted in grey) 29
3.5 ADMIRE Longitudinal Eingenvalues at constant Mach and altitude = [1000, 3000, 6000] [m] (first altitude condition is painted in grey) 30

4.1 ADMIRE simulator architecture 32
4.2 ADMIRE response for a sinusoidal demand on pitch rate control stick (F_{es}) . 34
4.3 ADMIRE response for a symmetrical pull-push demand on pitch rate control stick (F_{es}) 35

5.1 NARX/FNN-based Longitudinal State Identification Models - Number of Hidden Units Sweep Overview 46
5.2 NARX/FNN-based Longitudinal State Emulator (Neurons = 75) - Validation (TC4) 47
5.3 NARX/FNN-based Longitudinal State Emulator (Neurons = 75) - Validation (TC8) 48
5.4 NARX/FNN-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Training Epochs Sweep Overview 48
5.5 LSTM-based Pitch Rate Identification Models - Validation (TC6) 51
5.6 LSTM-based Pitch Rate Identification Models - Validation (TC74) 52
5.7 LSTM-based Pitch Rate Identification Models - Training History 53
5.8 LSTM-based Longitudinal State Identification Models - Number of Hidden Units Sweep Overview 55

5.9 LSTM-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Validation (TC3) 56
5.10 LSTM-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Validation (TC4) 57
5.11 LSTM-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Training History 58
5.12 LSTM-based Longitudinal State Identification Model (Neurons = 100) - Validation 59
5.13 LSTM-based Longitudinal State Identification Models - Training Epochs Sweep Overview 59
5.14 LSTM-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Validation (TC2) 60
5.15 LSTM-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Validation (TC5) 61
5.16 Comparation between CW-RNN, LSTM and Peephole-LSTM models 63
5.17 CW-RNN-based Longitudinal State Identification Models - Number of Hidden Units Sweep Overview 66
5.18 CW-RNN-based Longitudinal State Estimator Models with rates = [1 2 4] (Neurons Sweep Analysis) - Validation (TC47) 67
5.19 CW-RNN-based Longitudinal State Estimator Models with rates = [1 2 4] (Neurons Sweep Analysis) - Validation (TC52) 67
5.20 CW-RNN-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Training Epochs Sweep Overview 68
5.21 CW-RNN-based Longitudinal State Identification Models (Clock Rates Sweep Analysis) - Clock Rates Sweep Overview 69

List of Tables

3.1 Control Surfaces deflection constrains 21

5.1 FNN-based Pitch Rate Identification Models (I/O requirements analysis) - Performances Summary 43
5.2 NARX/FNN-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Performances Summary 46
5.3 NARX/FNN-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Performances Summary 47
5.4 LSTM-based Pitch Rate Identification Models - Performances Summary . . . 50
5.5 LSTM-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Performances Summary 55
5.6 LSTM-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Performances Summary 59
5.7 CW-RNN-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Performances Summary 65
5.8 CW-RNN-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Performances Summary 68
5.9 CW-RNN-based Longitudinal State Identification Models (Clock Rates Sweep Analysis) - Performances Summary 69

A.1 Testing Dataset Maneuvers 80
A.1 Testing Dataset Maneuvers 81
A.1 Testing Dataset Maneuvers 82
A.1 Testing Dataset Maneuvers 83

Chapter 1

Introduction

The aim of this project was to study the use of neural networks to identify the dynamic response of an aircraft. Multiple network architectures were assessed to confirm if the proposed models were able to estimate the non-linear dynamics of a figther aircraft.

Different solutions for dynamics modelling have been widely studied as critical components of intelligent control solutions, Failure Detection & Isolation (FDI) systems and to reduce the computational load of non-linear simulators (NLSs) by replacing the diffential equations solver by a deployed neural model.

The reference aircraft considered for this work was an open-source fifth-generation fighter model. Thus, the plant is characterized by a highly non-linear dynamics, unlike commercial aircraft that, although non-linear, their open-loop response are less agressive. It supposes an additional challenge in order to design appropiate estimation models.

The current study analyzes the characteristics of two well-known network architectures, Feedforward Neural Networks (FNNs) and Long-Short Term Memory Networks (LTSMs), and, additionaly, a relative recent proposal of recurrent network called *Clockwork Recurrent Network*, CW-RNN. The key feature of this recurrent model is that it is composed by multiple partitions of neurons which run at different clock rates. In other words, each module is only activate according its own clock rate, holding the last computed activation state when the module is disabled [Koutník et al., 2014].

The reason to consider those three neural network architectures is to perform a trade-off analysis of the pros and cons of using a more simple and easy to understand network as the FNN with respect to the performances advantages of implementing complex, but powerfull, recurrent networks as LSTM models. CW-RNN was selected due to the original concept and as possible optimal solution between the complexity and precision reached.

The report, ommitting this introduction chapter, is structured as follows:

- *State of the Art*: Brief description of the mathematics beyond the proposed neural network architectures and an analysis of the existing literature with respect to the proposed neural models, their use on the aerospace field, and, in particular, their implementation to solve dynamics identification problems.

- *ADMIRE Model*: Description of the figther model use to generate the synthetic data used in the scope of this project. Moreover, the aircraft dynammics and stability are analized.

- *Simulation Environment*: The Deep Learning setup utilized for this study is presented in this chapter. In particular, the ADMIRE simulator upgrades and the pipelines implemented to generate data, train models and post-process them automatically are explained here. Additional useful tools developed in the scope of the project are also mentioned.

- *Neural Dynamics Identification*: The core of the project is presented in this chapter. The designed identification models based on the neural network architectures mentioned previously (FNN, LSTM & CW-RNN) are described at this point. Multiple studies, varying different internal parameters, analyze the benefits and disadvantages of each kind of network when they are used as Dynamics Identification Models. Lastly, a final selection is done between the best candidates obtained from the studies.

- *Conclusiones and future works*: Last chapter of the report. All conclusions obtained during the project are listed here. Additionaly, future lines of works are mentioned.

Finally, several appendixes has been included at the end of the report in order to complete certain information exposed along this document.

Chapter 2

State of Art

Since the late 2010s, the Artificial Intelligent (AI) is experimenting a period of rapid progress, also informally called the *AI Boom* or *AI Spring*. Examples are the generative applications developed by Open AI as *Chat-GPT*, $DALL \cdot E$ and *SORA* which are able to answer questions, generate digital images from textual inputs and create realistic and imaginative video scenes from text instructions, respectivaly (detailed information can be found [OpenAI, 2024]). Moreover, additional AI model has been deployed recently in multiple areas of knowleadge as *AlphaFold* that is an AI system developed by a Google subsidiary (see [DeepMind, 2024]) which predits a protein's 3D structure from its amino acid sequence.

The *AI Boom* has not only produced technical applications which aim to revolutionize the different industries and research fields, it has also caused a significantly movement on the international markets. However, the deployment of these technologies on the aerospace sector is being slow.

The aeronautical and the space field has been always characterized by very high security standards that has been resulted in very detailed but strict regulations that every system within the aircraft environment shall fullfil. As it will be analyzed further on, those not much flexible rules affect to the kind of computer that can be installed on an aircraft and even the way in that the software (SW) of the different subsystem shall be coded and validated. For instance, currently there exist several limitations to use GPUs on the computation flows of critical systems and the SW validation process requires a determinism and repetitiveness hardly achievable by these kind of technologies. Thus, the implementation of intelligent solutions is requiring a huge invest of time, *i.e.*, money, that is slowing down its deployment.

Nevertheless, although the industry is reticent yet, the research centers have been working on different uses of deep learning architecture as optimal solutions of aerospace engineering problems. The objective of this project is to collaborate on this matter without ignoring the determinims and repetitiveness requirements that the aeronautical safety regulator demands. Thefore, not so complex models as the mentioned before, but promising, will be exposed within the scope of this project. Besides, the maths behind each neural architecture is explained below in order to asses the complexity of each proposed solution.

2.1 Neuron Model and Multilayer Perceptron Network

The neural network try to imitate the memory mechanism of the brain, where the neurons themselves has not storage capability but their association forms specific information. A neuron with R inputs is shown in the figure 2.1. The neuron output is computed as:

$$a = f\left(\mathbf{W} \cdot \mathbf{p} + b\right) \tag{2.1}$$

where $\mathbf{W}$ is the weight matrix, $\mathbf{p}$ is the input vector, b is a bias and $f(\cdot)$ named as the transfer function. Note that the weights and the bias are adjustable scalar parameters of the neuron whose values will be obtained after the training process.

The transfer function may be a linear or nonlinear function which shall be chosen depending on the specific problem that the designer is trying to solve. Denoting $n = \mathbf{W}\cdot\mathbf{p}+b$, some of the more typical transfer functions are listed below [Martin, 2016]:

- *Linear*:

$$a = n \tag{2.2}$$

- *Log-Sigmoid*:

$$a = \frac{1}{1 + e^{-n}} \tag{2.3}$$

- *Hyperbolic Tangent Sigmoid*:

$$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}} \tag{2.4}$$

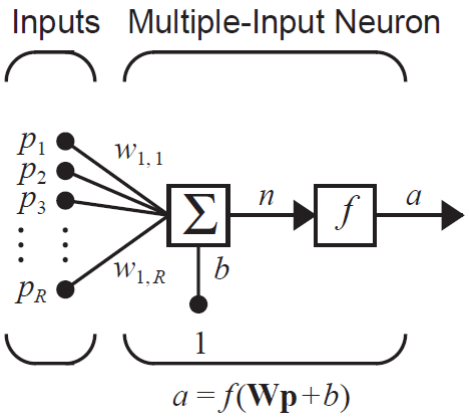


Figure 2.1: Multiple-Input Neuron diagram [Martin, 2016].

One neuron, even with many inputs, may not be satisfy the precision requested by the analyzed problem. The first way to scale the neural architecture is to add a set of neurons computing in parallel, i.e., form a layer of neurons. A single-layer network of S neurons is shown in the figure 2.2.

Each element of the input vector, $\mathbf{p}$, (called input layer by some authors) is connected to each neuron through the weight matrix $\mathbf{W}$. Additionally, a different bias is added to the product results within each neuron. The output of each node is computed as $a_k = f\left(n_k\right)$

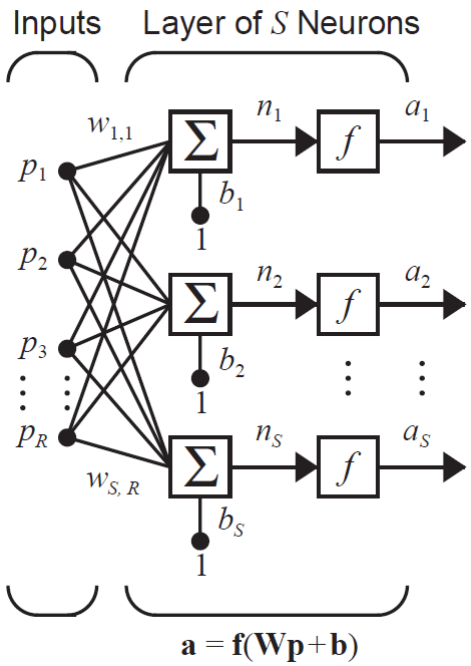


Figure 2.2: Single-layer network with S neurons [Martin, 2016].

where k denotes the neuron index. Note that it is assumed that all neurons use the same transfer function. This is not mandatory but it is a solution widely applied.

The dimensions of the previous weight matrix, $\mathbf{W}$, are (S, R) where the row, i, indices of the matrix elements, w_{ij}, indicate the destination neuron associated with that weight, while the column indices, j, make reference to the source of the input for that weight. The output of the complete network can be express in matrix terms as follows:

$$\mathbf{a} = f\left(\mathbf{W} \cdot \mathbf{p} + \mathbf{b}\right) = f\left(\mathbf{n}\right) \tag{2.5}$$

The second way to scale the network architecture is by adding multiple layer in order to estimate the target output. Each layer has its own weight matrix, $\mathbf{W}$, its own bias vector, $\mathbf{b}$, a net input vector, $\mathbf{n}$, and an output vector, $\mathbf{a}$. To distinguish between the parameters of each layer, it is common to use superscripts to identify the layers which the variable refers to. In other words, the variables of the layer m are identified as $\mathbf{W}^m$,$\mathbf{b}^m$,$\mathbf{n}^m$ and $\mathbf{a}^m$.

A layer whose output is the network output is called *Output Layer*, while the layers whose outputs are the inputs of the following layer are known as *Hidden Layer*. Note that multi-layer networks are more adjustable than the single-layer ones and allow to fit the most functions. Indeed, a two-layer network having a sigmoid first layer and a linear second layer can be trained to approximate most functions arbitrarily well [Martin, 2016].

2.1.1 Perceptron Learning Rule and Backpropagation

The perceptron learning rule is a simple algorithm used to train a perceptron, a type of artificial neural network. It's a supervised learning algorithm, meaning it requires labeled

training data to learn. Mathematically, the learning rule is defined as:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \eta(\mathbf{t} - \mathbf{y})\mathbf{x} \tag{2.6}$$

where:

- $\mathbf{w}_{new}$ is the updated weight vector.
- $\mathbf{w}_{old}$ corresponds to the previous weight vector.
- η is the learning rate
- $\mathbf{t}$ refers to the target output.
- $\mathbf{y}$ is the calculated output.
- $\mathbf{x}$ corresponds to the input vector.

This learning rule is a basic algorithm for training single-layer perceptrons but it is not able to train more complex models as the multi-layer neural networks.

Backpropagation is a supervised learning algorithm used to train artificial neural networks. It's a method of calculating the gradient of the error function with respect to the weights of the network, allowing for efficient optimization. This process is composed by two passes: Forward propagation and Backward propagation. In the forward pass, the input data is fed into the network and each neuron calculates its output based on the weighted sum of its input and an activation function. Then, the output of the final layer is compared to the desired target and an error is calculated. Next, the backward propagation starts and the error is propagated backward through the network, layer by layer. At each layer, the gradient of the error with respect to the weights and biases of that layer is calculated applying the chain rule. Finally, the weights and biases are updated using gradient descent or a similar optimization algorithm to minimize the error. A pseudo-code of the algorithm can be found below:

Algorithm 1 Backpropagation Algorithm

Input: Training data $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$, learning rate α
Initialize: Weights W and biases b randomly
Repeat:
For $i = 1$ to m:
Forward propagation:
Calculate $z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$ for $l = 1, 2, \ldots, L$
Calculate $a^{(l)} = g(z^{(l)})$ for $l = 1, 2, \ldots, L$
Error calculation:
Calculate $\delta^{(L)} = \frac{\partial C}{\partial a^{(L)}} \odot g'(z^{(L)})$
For $l = L - 1$ to 1:
Calculate $\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \odot g'(z^{(l)})$
Weight and bias update:
Update $W^{(l)} := W^{(l)} - \alpha \frac{\partial C}{\partial W^{(l)}} = W^{(l)} - \alpha \frac{\partial C}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial W^{(l)}}$
Update $b^{(l)} := b^{(l)} - \alpha \frac{\partial C}{\partial b^{(l)}} = b^{(l)} - \alpha \frac{\partial C}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial b^{(l)}}$
Until convergence or maximum number of iterations

The backpropagation is computationally efficient and flexible because it is able to train various types of neural networks, including multilayer perceptrons, convolutional neural networks, and recurrent neural networks. More information about how to implement this algorithm can be found in [Martin, 2016].

2.2 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a type of artificial neural network specifically designed to handle sequential data, such as text, time series, or speech. Unlike traditional feedforward neural networks, RNNs have feedback connections that allow them to process information sequentially and maintain a memory of past inputs. Mathematically, the basic unit of a RNN is expressed as follows:

$$h_t = f(W_h h_{t-1} + W_x x_t + b_h) \tag{2.7}$$

where $f(\cdots)$ is the activation function, usually equal to tanh, and the arguments are defined below:

- h_t is the output of the recurrent neuron, also called hidden state, at time step t.
- h_{t-1} corresponds to the hidden state at the previous time state.
- x_t refers to the input of the neuron at time step t.
- W_h and W_x are the weight matrices for connections between hidden states and between inputs and hidden states, respectively.
- b_h corrresponds to the bias vector.

This is a simplified version of an RNN unit. More complex RNN architectures, such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs), have additional mechanisms to address the vanishing gradient problem and improve learning for long sequences. In particular, LSTMs introduce gates (input, forget, and output) to control the flow of information, allowing them to better handle long-term dependencies and GRUs simplify the LSTM architecture while maintaining its ability to capture long-term dependencies.

Simple RNNs, while powerful, can face certain issues that limit their performance in some scenarios. The most common problems are:

- *Vanishing Gradient Problem*: As the RNN processes longer sequences, the gradients may become exponentially smaller, making it difficult for the network to learn long-term dependencies. This is due to the nature of the backpropagation algorithm. In other words, the gradient calculation in RNNs involves the repeated application of the chain rule, which can lead to a multiplication of small factors. Besides, The use of non-linear activation functions like *sigmoid* or *tanh* can saturate, resulting in small gradients. The main consequences are, in addition to the difficulty in learning long-term dependencies, that the learning process becomes slow and inefficient and the reached performance is poor.

- *Exploding Gradient Problem*: In contrast to vanishing gradients, gradients may also become exponentially larger, leading to instability during training. In this case, the chain rule may multiply large factor that can lead to exponentially increasing gradients. The main consequences are the instability, *i.e.*, large grandients may cause that the network weights become unstable, making it difficult to converge, and, in extreme cases, the weights may diverge. Furthermore, the training process becomes inefficient due to the need to adjust the learning rate to compensate for the large gradients.
- *Difficulty in Handling Long-Term Dependencies*: Simple RNNs struggle to capture and maintain information from distant past inputs, especially in long sequences.

Finally, in addition to new recurrent architectures, other strategies to mitigate these issues include: *Gradient Clipping*, *Weight Initialization* or *Data Normalization or Standarization*. Weight Initialization and data standarization has been used to develop the recurrent solutions proposed in this project.

2.2.1 LSTM Network

Long Short-Term Memory (LSTM) networks are a special type of recurrent neural network (RNN) designed to address the vanishing gradient problem and effectively capture long-term dependencies in sequential data. The LSTM unit is defined below:

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \\
\tilde{\mathbf{C}}_t &= \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \\
\mathbf{C}_t &= \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{C}_t)
\end{aligned} \tag{2.8}$$

where:

- $\mathbf{f}_t$ is the term called *Forget Gate*.
- $\mathbf{i}_t$ corresponds to the *Input Gate*.
- $\mathbf{o}_t$ is named *Output Gate*.
- $\tilde{\mathbf{C}}_t$ refers to the candidate cell state.
- $\mathbf{C}_t$ is the cell state.
- $\mathbf{C}_{t-1}$ refers to the cell state at the previous time step.
- $\mathbf{h}_t$ is the hidden state, the output of the LSTM neuron, at time step t.
- $\mathbf{h}_{t-1}$ corresponds to the hidden state at the previous time state.
- $\mathbf{x}_t$ refers to the input of the neuron at time step t.
- $\mathbf{W}_f$, $\mathbf{W}_i$, $\mathbf{W}_o$, $\mathbf{W}_C$ are the weight matrices of each gate and the cell state.

- $\mathbf{b}_h$ corrresponds to the bias vector.
- σ is the Sigmoid activation function.

The gates control the flow of information into and out of the cell state. A new candidate cell state, $\tilde{\mathbf{C}}_t$, is calculated based on the previous hidden state, $\mathbf{h}_{t-1}$, and the current input $\mathbf{x}_t$. Thus, the current cell state, $\mathbf{C}_t$, is updated by combining $\mathbf{C}_{t-1}$ with the candidate cell state, isuing the forget gate and the input gate to control the amount of information to retain or update. Finally, the hidden state is computed based on the new cell state and the output gate.

The cell state mechanism allows LSTMs to capture and maintain information over long sequences. On the other hand, the gates help to regulate the flow of information, reducing the likelihood of vanishing gradients. Therefore, LSTMs is characterized by superior performance on tasks that require understanding long-term dependencies, such as natural language processing and time series analysis.

Within the natural language processing field, this kind of network layers has been used to developed machine translation systems, *e.g.*, Google Translator (see [Wu, 2016]), capable of translating text between different languages with high accuracy, to automatically generate concise summaries of long documents and as sentiment analysis models. Another weel-known application are the Answer bots.

Respect to the time series analysis, typical applications are stock price prediction models (see [Gülmez, 2023]), sales forecasting (see [He, Wu, and Si, 2022]) and weather forecasting (see [Suleman and Shridevi, 2022]).

2.3 Clockwork Recurrent Neural Network

Standard Recurrent Neural Networks (RNNs) have difficulty to learn long-term dependencies encoded in the input sequences due to the *vanashing gradient* issue. [Koutník et al., 2014] proposed an original modification of the standard RNN in order to address the long-term dependency problem. In particular, the authors designed a RNN hidden layer which is divided into modules that run at different clock speeds, timing the computation ot their neuron states with specific discrete clock periods. This architecture is named Clockwork Recurrent Neural Network (CW-RNN).

The proposed solution results in a recurrent network that, besides to address the long-term dependency issues, train and evaluate faster since not all modules are executed at every time step. Moreover, it has a smaller number of learnable parameters compared to the standard solution because the slower modules do not feed back the output states from the faster modules.

The figure 2.3 represents the structure of the CW-RNN layer. It can be observed that the hidden layer is partitioned into g modules each with its own clock rate and within each partition the recurrent networks are fully interconnected. Moreover, the drawn lines show that neural units in faster module i are connecgted to neurons in slower module j only if the corresponding clock periods fullfil $T_i < T_j$.

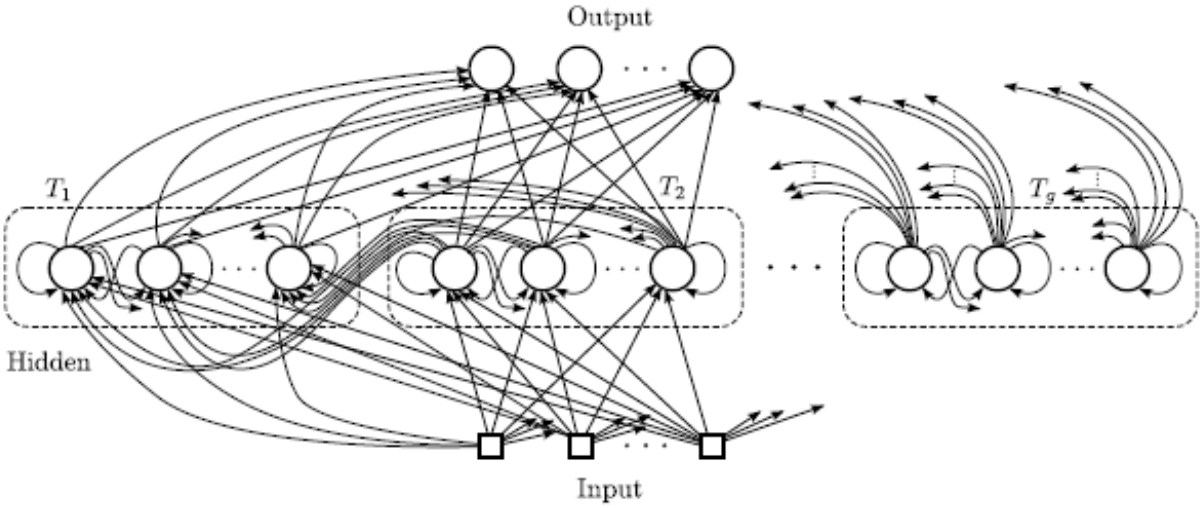


Figure 2.3: CW-RNN layer architecture [[Koutník et al., 2014]]

The mathematical model is expressed as follows:

$$\mathbf{y}_H^{(t)} = f_H\left(\mathbf{W}_H \cdot \mathbf{y}_H^{(t-1)} + \mathbf{W}_I \cdot \mathbf{x}^{(t)} + \mathbf{b}\right) \tag{2.9}$$

$$\mathbf{y}_O^{(t)} = f_H\left(\mathbf{W}_O \cdot \mathbf{y}_H^{(t)}\right) \tag{2.10}$$

where $\mathbf{W}_H$, $\mathbf{W}_I$ and $\mathbf{W}_O$ are the hidden, input and output weight matrices, $\mathbf{b}$ is the bias vector, $\mathbf{x}^{(t)}$ is the input vector at t step, and $\mathbf{y}_H^{(t)}$ is the activation of the hidden units at time step t. Functions $f_H(...)$ and $f_O(...)$ are non-linear activation functions.

Due to the clock-rates modules, the weight matrices are also partitioned into g block-rows:

$$\mathbf{W}_H = \begin{pmatrix} \mathbf{W}_{H_1} \\ \vdots \\ \mathbf{W}_{H_g} \end{pmatrix} \tag{2.11}$$

$$\mathbf{W}_I = \begin{pmatrix} \mathbf{W}_{I_1} \\ \vdots \\ \mathbf{W}_{I_g} \end{pmatrix} \tag{2.12}$$

and $\mathbf{W}_H$ is a block-upper triangular matrix. At each time step of the forward pass, only the block-rows of $\mathbf{W}_H$ and $\mathbf{W}_I$ that correspond to the active modules are used to compute $\mathbf{y}_H$:

$$\mathbf{W}_{H_i} = \begin{cases} \mathbf{W}_{H_i} & \text{if } (t \text{ MOD } T_i) = 0 \\ \mathbf{0} & otherwise \end{cases} \tag{2.13}$$

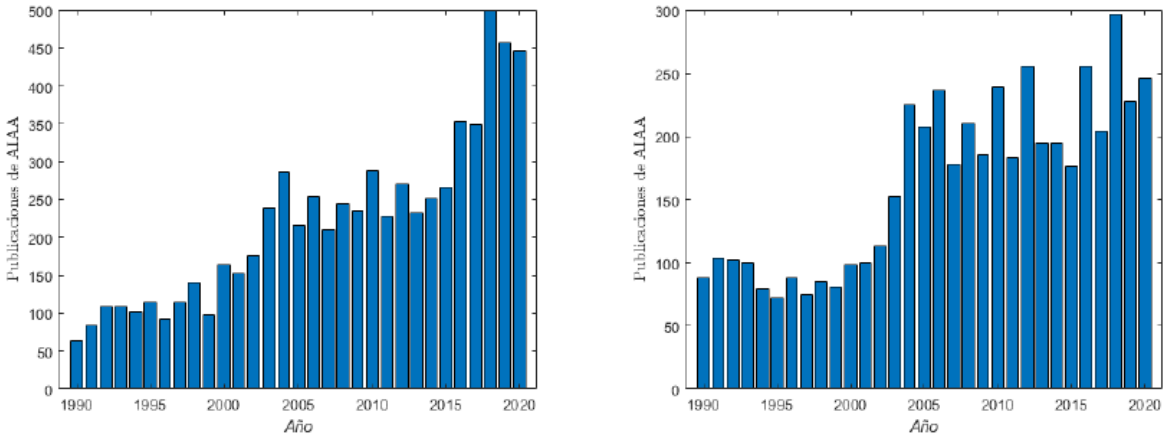
According to the authors, "*The experimental results show that the simple mechanism of running subsets of neurons at different speeds allows an RNN to efficiently learn the different dynamic time-scales inherent in complex signals*". This feature is very interesting from the dynamics identification perspective.

Multiple applications can be considered: Natural Language Processing, Time Series Analysis, Audio Processing, Video Processing... In particular, some authors have proposed to use CW-RNN models to support computer-aided design (CAD) programs (see [Charoosaei, Faraji, Sadrossadat, Mirvakili, Na, Feng, and Zhang, 2024]), for traffic flow modelling (see [Liu, Gu, Ding, Lu, Rui, and Tao, 2020]) and for wind speed prediction for offshore sites (see [Shi, Wang, and Zheng, 2022]). Specific aerospace applications have not been found on the main journals of the sector.

In summary, Clockwork Recurrent Networks offer a promising approach to modeling sequential data, particularly for tasks involving long sequences or hierarchical structures. Their ability to efficiently process information while capturing long-term dependencies makes them a valuable tool for a wide range of applications.

2.4 Neural networks within the aerospace industry

The aerospace sector also experimented the *AI Boom*, at least, in the research field. For instance, searching for publications tagged as *Machine Learning* or *Artifical Intelligence* in the journals published by the American Institute of Aeronautics and Astronautics, AIAA, it can be observed a increase tendency of articles related with these toppics. In particular, the figure 2.4 represent the evolution of the number of articles since 90s until 2020.



(a) Tag 'Machine Learning'

(b) Tag 'Artificial Intelligence'

Figure 2.4: Evolution of the number of articles of the AIAA tagged as "Machine Learning" and "Artificial Intelligence"

Although the aerospace industry has been always characterized by integrating technological advances slowly, due to its high safety standards, Artificial intelligence (AI) is transforming the aerospace industry step by step. For instance, AI applications is starting to be used for aircraft design, for autonomous flight and space exploration. A detailed list of some applications explored by the industry is attached below:

- *Aircraft Design*: AI algorithms can generate innovative designs based on specific constraints and objectives, leading to more efficient and cost-effective aircraft structures.

- *Autonomous Manufacturing*: Robotics powered by AI can automate tasks like assembly and inspection, improving efficiency and quality.
- *Predictive Maintenance* and *Health Monitoring*: AI algorithms can analyze sensor data to predict equipment failures, reducing downtime and maintenance costs. Additionaly, the same information can be used to monitor systems in real-time, identifying potential issues before they become critical.
- *Autonomous Vehicles*: Autonoumous flight, Collision Avoidance and Formation Flight applications can be powered by AI solutions in order to achieve a dafe and efficient operation.
- *Air Traffic Management*: AI-powered systems can optimize air traffic management, reducing delays and improving safety.
- *Decision Support Systems/Co-Pilot*: AI-powered systems can provide pilots and air traffic controllers with real-time recommendations and alerts.

Specific examples of the previous applications can be found on the industry. For instance, Airbus, the largest manufacturer of airliners and helicopter, has deployed *Skywise* between some of its airlines clients. Skywise is an industry data platform to address aircraft operations challenges option by combining in-glight, engineering and operational data in an analytic rich environment. On the other hand, multiple autonomous flight solutions based on AI has been flight tested by Airbus. Between these demonstrators, it can be found:

- *ATTOL*: The Autonomous Taxi, Take-off and Landing (ATTOL) project leveraged deep learning computer-vision technologies and techniques to successfully complete fully autonomous taxi, take off, approach and landing using a commercial aircraft. This flight test campaing was performed in 2020 and since them, new concepts has been developed.
- *Dragon Fly*: It was a demonstrator that explored new pilot assistance technologies to further enhance flight safety and aircraft operational efficiency. The technologies designed, developed and tested enabled an aircraft to identify features in the landscape so that it can "see" and safely manoeuvre within its surroundings. This assistantce included to be able to automated emergency operations (diversion) in cruise as well as automatic landing whatever the weather conditions.
- *Auto'Mate*: This Spanish demonstrator, with the collaboration of the German Airbus Defence division, developed and evaluated technologies which enabled autonomous air-to-air refuelling in formation flight operations. It achieved to perform an autonomous close formation flight between the tanker aircraft and several drones by tracking the drones positions using deep learning computer vision solutions.

More information about the previous and additional projects can be found in [Airbus, 2024]. It should be noticed that, except for Skywise, the rest of projects were demonstrators, *i.e.*, although the target technologies were validated in flight, a long way to go before being able to industrialize those solutions fullfilling the safety standards for critical applications.

For its part, Boeing, the American aircraft manufacturer, is working on AI-powered

predictive maintenance solutions and is enhancing the flight capabilities of its new uncrewed aircraft, the MQ-28 (see [Boeing, 2024]).

It can be concluded that, even though, the aerospace industry is working on AI-powered applications, the number of available solutions is negligible with respect to the amout of Generative AI applications, Answer Bots or Decision Support Systems applied for finances and stock markets. It is due to the rigid regulations that characterizes the aerospace industry and request a high level of confidence and safety. Those levels are not only hard to reach, furthermore, the design and validation processes defined by the regulation are applicable for the most of the AI solutions. Thus, the aeronautical and space industries are working in the development of AI capabilities and discussing with the authorities about how to certification those products. Nevertheless, the research centers and the academia has been publishing a huge amount of articles proposing AI solutions to improve systems and processes capabilities.

One of the most studied AI-powered applications within the aerospace academia has been the intelligent flight control systems. Multiple proposals have been published to increase the autonomy of the aircraft. For instance, adaptative flight controllers composed by deep learning models have been investigated during the last years.

An Intelligent Flight Control System (IFCS) is a flight control system that utilizes artificial intelligence technologies to enhance aircraft performance, safety, and efficiency. These systems go beyond traditional flight control systems by incorporating advanced algorithms, machine learning, and data analytics to make more informed and adaptive decisions. Its main feature is its capability to adapt the controller to flight condition changes, such as turbulence or engine and actuators failures, ensuring continued safe operation. Furthermore, the IFCS can reduce the pilot workload by automating routine tasks and providing intelligent assistance. [Emami, Castaldi, and Banazadeh, 2022] presents an in-depth mathematical view of IFCSs, particularly those based on artifical neural networks. The paper covers both the model-free and model-based methods. The model-free scheme does not require any prior information about the system dynamics to be used in the control design procedure. Examples of this approach are to model the entire controller by a single NN, to use the concept of Approximate Dynamic Programing or to implement Reinforcement Learning. On the other hand, the model-based neural control utilizes a nominal model of the system and the neural networks attempts either to estimate the model uncertainties or to determina the control command.

Focusing on the Neural Flight Control Systems (NFCS), which are a specific type of intelligent flight control system that use artificial neural networks to model and control the aircraft, several researchers have studied architectures which are divided into two parts: Dynamics Identification and Control. These architectures propose to include a NN-based dynamics identification model which compare its output with respect to the actual one and, based on the differences, trigger an online training on the controller neural network in order to adapt the control law to the changing flight conditions or model uncertainties. Besides, the neural dymanics estimator is also able to update itself via online training. It should be noticed that multiple implementation options exist, *i.e.*, the neural networks may model the entire controller or the complete dynamics or it may estimate uncertainties or increments corrections over the classical components. Examples of those architectures can

be found in [HAOUANI, SAAD, and AKHRIF, 2002], which proposes to update the PID gainset with a Feedforward Neural Network, [Krishnakumar, Limes, and Gundy-Burlet, 2003], that examines a neural network augmented adaptive critic as an intelligent controller, and [Yerramalla, Fuller, Mladenovski, and Cukic, 2003], which presents an Adaptive Flight Control System based on the continous adaptation of a neural network.

All the previous articles utilize a neural network models in order to identify or estimate both plant and controller dynamics. Thus, the conclusion of this study, that was focused on analyze the identification performances of the ADMIRE dynamics for multiple kinds of neural architectures, are close-related to the NFCS solutions which may be included within future related works.

Chapter 3

ADMIRE Model

In the scope of this project, an aircraft has been chosen as dynamics plant to be identified by using neural networks. In particular, the model corresponds to a fighter aircraft designed by the former Aeronautical Research Institute of Sweden (FFA) based on the Generic Aerodata Model (GAM). This aerodynamics dataset was elaborated by Saab in cooperation with the Royal Institute of Technology (KTH) as it is explained in [Bates and Hagström, 2007].

The ADMIRE is a generic model of a small single-seat fighter aircraft with a delta-canard configuration (see figure 3.1). This aircraft model is characterized by twelve dynamic states,

- V_T: [m/s] Total velocity of the aircraft
- α: [deg] Angle of attack (AoA)
- β: [deg] Angle of sideslip (AoS)
- p_b: [deg/s] Body-fixed roll rate
- q_b: [deg/s] Body-fixed pitch rate
- r_b: [deg/s] Body-fixed yaw rate
- ϕ: [deg] Bank angle
- θ: [deg] Pitch angle
- ψ: [deg] Yaw angle
- x_v: [m] X-Position in the vehicle reference coordinates
- y_v: [m] Y-Position in the vehicle reference coordinates
- z_v: [m] Z-Position in the vehicle reference coordinates

plus additional states due to actuators, sensors and the Fight Control System (FCS). The fighter can be controlled by the following effectors:

- Left canard: δ_{lc} [deg]
- Right canard: δ_{rc} [deg]
- Left outer elevon: δ_{loe} [deg]
- Right outer elevon: δ_{roe} [deg]
- Left inner elevon: δ_{lie} [deg]
- Right inner elevon: δ_{rie} [deg]
- Leading edge flap: δ_{le} [deg]
- Rudder: δ_r [deg]
- Throttle stick setting: T_{ss} [%]
- Horizontal thrust vectoring: δ_{th} [deg]
- Vertical thrust vectoring: δ_{tv} [deg]

Besides, the model also implements air brake control and the possibility to have the landing gear up or down.

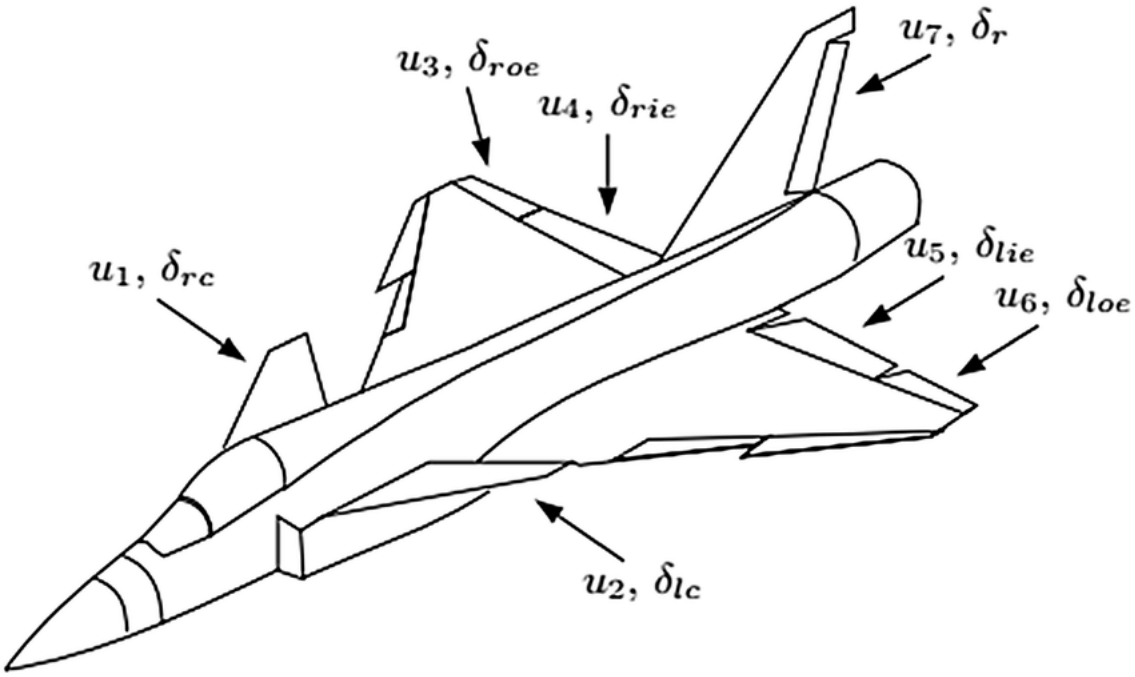


Figure 3.1: ADMIRE sketch where the main control effectors are denoted. Furthermore, the aircraft is also equipped with thrust vectoring capability and air brakes. [Forssell et al., 2005]

The ADMIRE is augmented with an FCS in order to provide stability and sufficient handling qualities within the operational envelope. The flight envelope is restricted to Mach numbers less than 1.2 and altitudes below 6 km. Moreover, within the envelope there exist

additional constrains due to aerodynamics behaviours, structural reasons and the well-being of the pilot. In particular, the angle of attack (AoA), the angle of slip (AoS) and the aerodynamics control surfaces deflections limits are function of the Mach as is shown on the Figure 2.2 of [Forssell et al., 2005]. On the other hand, the normal load factor is constrained to $-3g \leq n_z \leq +9g$ in order to fulfill the assumed structural and pilot-health requirements.

Sensors models of the measurements used by the FCS are implemented in the simulator, together with simple actuators models and characteristic computers delays (20 ms on the actuators input).

3.1 Aircraft Dynamics Model

The aircraft dynamics are modeled on the standard form:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{p}) \tag{3.1}$$

$$\mathbf{y} = g(\mathbf{x}, \mathbf{p}) \tag{3.2}$$

where $\mathbf{x}$ is the state vector of dimension 12×1, the $\mathbf{y}$ is the output vector and $\mathbf{p}$ is the uncertainty parameter space vector. In particular, the state equations are detailed below:

$$\dot{V}_T = \frac{u_b \dot{u}_b + v_b \dot{v}_b + w_b \dot{w}_b}{V_T} \tag{3.3}$$

$$\dot{\alpha} = \frac{u_b \dot{w}_b - w_b \dot{u}_b}{u_b^2 + w_b^2} \tag{3.4}$$

$$\dot{\beta} = \frac{\dot{v}_b V_T - v_b \dot{V}_T}{V_T^2 \cos\beta} \tag{3.5}$$

$$\dot{p}_b = (C_1 r_b + C_2 p_b) q_b + C_3 M_x + C_4 M_z \tag{3.6}$$

$$\dot{q}_b = C_5 p_b r_b - C_6 \left(p_b^2 - r_b^2\right) + C_7 M_y \tag{3.7}$$

$$\dot{r}_b = (C_8 p_b - C_2 r_b) q_b + C_4 M_x + C_9 M_z \tag{3.8}$$

$$\dot{\psi} = \frac{q_b \sin\phi + r_b \cos\phi}{\cos\theta} \tag{3.9}$$

$$\dot{\theta} = q_b \cos\phi - r_b \sin\phi \tag{3.10}$$

$$\dot{\phi} = p_b + \tan\theta \left(q_b + \sin\phi + r_b \cos\phi\right) \tag{3.11}$$

$$\begin{aligned}\dot{x}_v = \cos\theta\cos\psi \cdot u_b + (\sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi) \cdot v_b \\ + (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi) \cdot w_b\end{aligned} \tag{3.12}$$

$$\begin{aligned}\dot{y}_v = \cos\theta\sin\psi \cdot u_b + (\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi) \cdot v_b \\ + (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) \cdot w_b\end{aligned} \tag{3.13}$$

$$\dot{z}_v = -\sin\theta \cdot u_b + \sin\phi\cos\theta \cdot v_b + \cos\phi\cos\theta \cdot w_b \tag{3.14}$$

where the linear acceleration components over the boxy-axes reference is calculated as:

$$\dot{u}_b = r_b v_b - q_b w_b - g_0 \sin\theta + \frac{F_x}{m} \tag{3.15}$$

$$\dot{v}_b = -r_b u_b + p_b w_b + g_0 \sin\phi \cos\theta + \frac{F_y}{m} \tag{3.16}$$

$$\dot{w}_b = q_b u_b - p_b v_b + g_0 \cos\phi \cos\theta + \frac{F_z}{m} \tag{3.17}$$

and the variables C_k with $k = 1, 2, ...9$ are inertia coefficients obtained from the following expressions:

$$\Gamma = I_{xx}\left(1+\delta I_{xx}\right) \cdot I_{zz}\left(1+\delta I_{zz}\right) - \left[I_{xz}\left(1+\delta I_{xz}\right)\right]^2 \tag{3.18}$$

$$C_1 = \frac{I_{yy}\left(1+\delta I_{yy}\right) - \left[I_{zz}\left(1+\delta I_{zz}\right)\right]^2 - \left[I_{xz}\left(1+\delta I_{xz}\right)\right]^2}{\Gamma} \tag{3.19}$$

$$C_2 = \frac{I_{xx}\left(1+\delta I_{xx}\right) - I_{yy}\left(1+\delta I_{yy}\right) + I_{zz}\left(1+\delta I_{zz}\right) \cdot I_{xz}\left(1+\delta I_{xz}\right)}{\Gamma} \tag{3.20}$$

$$C_3 = \frac{I_{zz}\left(1+\delta I_{zz}\right)}{\Gamma} \tag{3.21}$$

$$C_4 = \frac{I_{xz}\left(1+\delta I_{xz}\right)}{\Gamma} \tag{3.22}$$

$$C_5 = \frac{I_{zz}\left(1+\delta I_{zz}\right) - I_{xx}\left(1+\delta I_{xx}\right)}{I_{yy}\left(1+\delta I_{yy}\right)} \tag{3.23}$$

$$C_6 = \frac{I_{xz}\left(1+\delta I_{xz}\right)}{I_{yy}\left(1+\delta I_{yy}\right)} \tag{3.24}$$

$$C_7 = \frac{1}{I_{yy}\left(1+\delta I_{yy}\right)} \tag{3.25}$$

$$C_8 = \frac{I_{xx}\left(1+\delta I_{xx}\right) \cdot \left[I_{xx}\left(1+\delta I_{xx}\right) - I_{yy}\left(1+\delta I_{yy}\right)\right] + \left[I_{xz}\left(1+\delta I_{xz}\right)\right]^2}{\Gamma} \tag{3.26}$$

$$C_9 = \frac{I_{xx}\left(1+\delta I_{xx}\right)}{\Gamma} \tag{3.27}$$

where δI_{ij} refers to inertial uncertainties. Since it is assumed that the aircraft is symmetric about the Oxz plane and that the mass is uniformly distributed, as a result, the products of inertia $I_{xy} = I_{yz} = 0$. The reader can observe in the previous C_k coefficients that this assumption has been considered. If the aircraft body axes were aligned with the principal inertia axes, the remaining product of inertia I_{xz} will be also zero. However, it is quite difficult to determine the principal inertia axes with enough precision so this simplification is not often applicable. Nevertheless, due to the symmetry of the aircraft, I_{xz} is generally very much smaller than the main inertia components I_x, I_y and I_z.

The forces and moments defined above, *i.e.*, F_x, F_y, F_z, M_x, M_y and M_z are the result of interactions between the aircraft and the surrounding air. Detailed information about how those variables are computed can be found on the reference [Forssell et al., 2005].

Lastly, the output states fulfil the following equations:

$$u_b = V_T \cos\alpha \cos\beta \tag{3.28}$$

$$v_b = V_T \sin\beta \tag{3.29}$$

$$w_b = V_T \sin\alpha \cos\beta \tag{3.30}$$

$$u_v = \cos\theta\cos\psi \cdot u_b + (\sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi) \cdot v_b + (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi) \cdot w_b \tag{3.31}$$

$$v_v = \cos\theta\sin\psi \cdot u_b + (\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi) \cdot v_b + (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) \cdot w_b \tag{3.32}$$

$$w_v = -\sin\theta \cdot u_b + \sin\phi\cos\theta \cdot v_b + \cos\phi\cos\theta \cdot w_b \tag{3.33}$$

$$n_z = -\frac{F_{Z_{aero}}}{m \cdot g_0} \tag{3.34}$$

$$n_y = -\frac{F_{Y_{aero}}}{m \cdot g_0} \tag{3.35}$$

$$M = \frac{V_T}{a(h)} \tag{3.36}$$

$$\gamma = \arcsin\left(\cos\alpha\cos\beta\sin\theta - [\sin\phi\sin\beta + \cos\phi\sin\alpha\cos\beta]\cos\theta\right) \tag{3.37}$$

$$C_D = C_N \sin\alpha + C_T \cos\alpha \tag{3.38}$$

$$C_L = C_N \cos\alpha + C_T \sin\alpha \tag{3.39}$$

Additional output variables, such as the force and moment coefficients, are calculated by the aircraft model an can be used to analyze or characterize the fighter behaviour.

3.1.1 Aerodynamic model

Aerodynamic models are mathematical representations that describe the relationship between an aircraft's shape, orientation, and the surrounding airflow.

The ADMIRE aerodata model consists of aerodata tables, interpolation routines and aerodata algorithms. These tables are calculated based on wind tunnel results of the target aircraft or of the similar ones.

The aerodynamic model is built up in a conventional way, by interpolating in (unstructured) data tables to get the different contributions or aerodynamic coefficients. Different aerodata tables are used at different Mach numbers.

3.1.2 Engine model

The ADMIRE fighter uses a jet-engine with afterburner capability. The engine thrust is calculated interpolating from two 2-dimensional data tables, one for the case when afterburner is activated and the other one for the opposite case. These tables, i.e, the thrust, are function of the altitude, h, and the Mach number, M.

The input to the engine is the Throttle Stick Setting, T_{ss}, which takes values from 0 to 1. If $T_{ss} > 0.78$, afterburner begins to work and when $T_{ss} \geq 0.82$, the system is fully

activated. Below, the thrust force computation, based-on experimental look-up tables and including the transient state, is shown:

$$T = \begin{cases} T_{AB_{Off}}\left(T_{ss}, h, M\right) & \text{if } T_{ss} < 0.78 \\ 25\left[T_{AB_{On}}\left(0.82, h, M\right) - T_{AB_{Off}}\left(0.78, h, M\right)\right]\left(T_{ss} - 0.78\right) + T_{AB_{Off}}\left(0.78, h, M\right) & \text{if } 0.78 \leq T_{ss} < 0.82 \\ T_{AB_{On}}\left(T_{ss}, h, M\right) & \text{if } T_{ss} \geq 0.82 \end{cases} \tag{3.40}$$

where $T_{AB_{Off}}$ and $T_{AB_{On}}$ are the experimental data tables when the afterburner is off and on, respectively.

The dynamic response of the engine is not instantaneous because increasing or decreasing the rotation speed of the compressor takes some time. To model this effect, a simple first-order lag filter between the commanded Throttle Stick Setting, $T_{ss_{com}}$, and the actual Throttle Stick Setting, T_{ss}, is implemented:

$$T_{ss}(s) = \frac{0.5}{s + 0.5} \cdot T_{ss_{com}} \tag{3.41}$$

Fuel consumption is modeled as the thrust force. Two 2D look-up tables obtained from experimental data are compiled and used to compute the instantaneous fuel consumption function of the Throttle Stick Setting, the flight altitude and the Mach number:

$$\dot{m}_f = \begin{cases} \dot{m}_f^{AB_{Off}}\left(T_{ss}, h, M\right) & \text{if } T_{ss} < 0.78 \\ 25\left[\dot{m}_f^{AB_{On}}\left(0.82, h, M\right) - \dot{m}_f^{AB_{Off}}\left(0.78, h, M\right)\right]\left(T_{ss} - 0.78\right) + \dot{m}_f^{AB_{Off}}\left(0.78, h, M\right) & \text{if } 0.78 \leq T_{ss} < 0.82 \\ \dot{m}_f^{AB_{On}}\left(T_{ss}, h, M\right) & \text{if } T_{ss} \geq 0.82 \end{cases} \tag{3.42}$$

Once, the engine thrust and the fuel consumption are obtained, the thrust force vector is computed. The ADMIRE model has the capability for thrust vectoring control but these effectors, δ_{th} and δ_{tv}, are not considered as control inputs neither for the baseline controller (controller included in the simulator and developed following Classical Control Theory as is explained futher on) nor for the new neural-controller designed during this project. Therefore, the thrust vector calculation is reduced to the following model:

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} T \\ 0 \\ 0 \end{bmatrix} \tag{3.43}$$

In the case, the thrust vectoring control (TCV) deflections were included on the design loop, thrust force vector computation are more complex and it is required to take into account the jet-engine parameters and variables (e.g., inlet are, air mass flow, nozzle geometry...). The ADMIRE's TCV model can be found on the *admire_tcv.h* file and further explanations about this kind of control are described in [George, 2010]

3.1.3 Actuators

The generic actuator model used by the ADMIRE simulator corresponds to a first order transfer function with limited angular deflection and maximum angular rate. Each kind of actuator is characterized by a specific time constant. In particular, all control surfaces

Control Surface	Min [deg]	Max [deg]	Angular Rate [deg/s]
Canard	-55	25	± 50
Rudder	-30	30	± 50
Elevons	-25	25	± 50
Leading Edge Flaps	-10	30	± 50

Table 3.1: Control Surfaces deflection constrains

except for the leading edge flap uses the same time constant. On the other hand, the specific angular deflection limits and maximum angular rates are gathered on the table 3.1.

The sign of the actuator deflections follows the *right-hand-rule* except for the leading edge flap which has a positive deflection down.

Finally, the figure 3.1 denotes all the control surfaces available and the reader can note that there exist four elevons: δ_{roe}, δ_{rie}, δ_{loe}, δ_{lie}. It is important to remark that the inner and outer elevons on each side always move together.

3.1.4 Sensors

In the ADMIRE, there are implemented models of air data sensors (V_t , α, β, h), inertial sensors (p_b, q_b, r_b, n_z) and attitude sensors (θ, ϕ). Below, the reader can find the transfer function used for each kind of sensor:

- *Air Data Sensors*:

$$\xi_{sensed}(s) = \frac{1}{1 + 0.02 \cdot s} \cdot \xi$$

- *Inertial Sensors*:

$$\xi_{sensed}(s) = \frac{1 + 0.005346 \cdot s + 0.0001903 \cdot s^2}{1 + 0.03082 \cdot s + 0.0004942 \cdot s^2} \cdot \xi$$

- *Attitude Sensors*:

$$\xi_{sensed}(s) = \frac{1}{1 + 0.0323 \cdot s + 0.00104 \cdot s^2} \cdot \xi$$

where ξ corresponds to the mentioned measured variables of each sensoring group.

3.1.5 Model uncertainties

The parametric uncertainties of the model are: Configuration, Aerodynamic, Sensor, Actuator. However, nominal values has been used during this project, *i.e.*, any uncertainty was applied. A detailed explanation of the uncertainty models can be found in the section 2.7 of [Forssell et al., 2005].

3.1.6 Atmospheric model

A reduced version of the International Standard Atmosphere (ISA) model is used. The atmosphere is assumed to be only dependent on the altitude and only the density and the sound speed are computed.

The simulator allows the user to introduce atmospheric turbulence or wind. This feature has not been used during the development of this project to reduce the noise in the resulting datasets.

3.2 Baseline Flight Control System model

The ADMIRE dynamics is augmented with a FCS in order to provide required stability and handling qualities within the operational envelope. The control system is divided into two parts: *Longitudinal controller* and *Lateral-Directional controller*.

Both control systems were designed following the standard process used in the aeronautical industry where the gainsets are obtained from dynamics linear models derived for several trim conditions (29 in this case). Once the gain values are obtained for all equilibrium points, a scheduling is done by using the Mach number and the altitude.

In order to model the time delays from the computation time of the physical Flight Control Computer (FCC), transport delays of 20 ms are added to all actuators. This common transfer function has been modeled as a second order Padé-approximation.

To sum up, standard linear design methods were used to synthesize the gains, in particular, the Pole Placement Method (PMM) was developed for both control system parts, selecting the augmented system poles in such a way that aircraft stability parameters and response characteristics fulfilled the specifications and regulation for this kind of aircraft.

This version of the Flight Control System will be used during the data generation process in order to get stable responses of the ADMIRE.

3.2.1 Longitudinal controller

The longitudinal controller is subdivided into two additional parts: the *Pitch* controller and the *Speed* autopilot.

The speed controller is based on a proportional gain together with a lead filter. The objective of this control system is to mantain the speed commanded by the pilot. It is achieved by varying the the throttle setting of the ADMIRE engine.

The pitch rate controller is characterized by two working modes. At lower speeds ($M < 0.58$), the control law tries to minimize the tracking error in the commanded pitch rate. This pitch rate is input via the longitudinal stick of the pilot. At higher speeds ($M > 0.62$), the controller tracks the commanded load factor.

Both control laws, the pitch rate and the load factor controllers, consist in a stabilizing inner loop and a tracking outer loop. The objective of the inner loop, as the name

describes, is to stabilize the longitudinal dynamics of the ADMIRE that is unstable within a extense part of the flying envelope. More information about the control laws used can be found in [Forssell et al., 2005] or analyzing the ADMIRE simulink model.

3.2.2 Lateral controller

The lateral-directional controller modifies the lateral channel responses in order to get enhanced handling qualities. In this case, the control law does not stabilize the system, it only varies the characteristics of the response to reach optimal performances in agreement with the certification regulation. This kind of controller are widely called *Control Augmentation System*, CAS.

In particular, the body-axis roll rate is fed back to the ailerons and the objective is to reduce the variation of the aircraft roll performance with the flight conditions. Additionaly, an approximation of the wind-axis yaw rate is fed back to the rudder in order to adjust the roll damping. A more detailed description can be found in the original reference (see [Forssell et al., 2005]) and in the open-source simulink model.

3.3 Decoupled Aircraft Dynamics

The decoupled equations of an aircraft are a set of equations that describe the motion of the aircraft in a simplified form, assuming that the aircraft's six degrees of freedom can be treated independently. This simplification is often valid for small perturbations around a steady-state flight condition. this decoupling is based on two reasons:

- *Physical Separation*: The longitudinal axis (pitch, speed, and altitude) and the lateral-directional axis (roll, yaw, and heading) are physically distinct and often have minimal direct influence on each other. For example, changing the pitch of an aircraft primarily affects its altitude and speed, with minimal impact on its roll or yaw. It can be confirmed analyzing the elements of the **A** matrix when the linearised dynamics expressed on the state-space representation.

- *Time Scale Separation*: The dynamics of longitudinal and lateral-directional motions often occur at different time scales. Longitudinal motions, such as changes in altitude or speed, tend to be slower, while lateral-directional motions, like turning or banking, are typically faster. This separation in time scales allows for independent control of these motions.

In fact, aircraft desings are intentionally thought in order to minimize the coupling between longitudinal and lateral-directional motions. For instance, the design of control surfaces (ailerons, elevators, and rudders) can be optimized to reduce cross-coupling effects.

Assuming that the lateral-directional motion does not have effect on the longitudinal states and that the most of the time an aircraft is flying in a wing-level steady-state (see [Stevens, Lewis, and Johnson, 2015]), the ADMIRE equations for the longitudinal dynamics can be reduced to:

$$\dot{V}_T \approxeq \frac{u_b \dot{u}_b + w_b \dot{w}_b}{V_T} \tag{3.44}$$

$$\dot{\alpha} \approxeq \frac{u_b \dot{w}_b - w_b \dot{u}_b}{u_b^2 + w_b^2} \tag{3.45}$$

$$\dot{q}_b \approxeq C_7 M_y \tag{3.46}$$

$$\dot{\theta} \approxeq q_b \tag{3.47}$$

where the linear acceleration components over the boxy-axes reference are:

$$\dot{u}_b \approxeq -q_b w_b - g_0 \sin\theta + \frac{F_x}{m} \tag{3.48}$$

$$\dot{w}_b \approxeq q_b u_b + g_0 \cos\theta + \frac{F_z}{m} \tag{3.49}$$

On the other hand, the lateral-directional dynamics of the ADMIRE is simplified as follows:

$$\dot{\beta} = \frac{\dot{v}_b V_T - v_b \dot{V}_T}{V_T^2 \cos\beta} \tag{3.50}$$

$$\dot{p}_b = C_3 M_x + C_4 M_z \tag{3.51}$$

$$\dot{r}_b = C_4 M_x + C_9 M_z \tag{3.52}$$

$$\dot{\psi} = \frac{r_b \cos\phi}{\cos\theta} \tag{3.53}$$

$$\dot{\phi} = p_b + \tan\theta \left(\sin\phi + r_b \cos\phi\right) \tag{3.54}$$

where the linear acceleration components on the y-body-axis is:

$$\dot{v}_b = -r_b u_b + p_b w_b + g_0 \sin\phi \cos\theta + \frac{F_y}{m} \tag{3.55}$$

The decoupled equations are useful for analyzing the stability and control characteristics of an aircraft, as well as for designing flight control systems.

3.3.1 Linearised equations

Linearization is a mathematical technique used to simplify complex nonlinear systems by approximating them as linear systems within a specific operating region. In the context of aircraft dynamics, linearization is essential for analysis, control design, and simulation.

The linearised equations of motion for an aircraft can be expressed in state-space form as:

$$\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{3.56}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{3.57}$$

where $\mathbf{x}$ and $\mathbf{u}$ are the state vector and the input/control vector, respectively. Besides, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are the matrices of the state-space.

The linearised equations are reached after applying the perturbation theory on the trim condition or operating point of the aircraft. In other words, each linear representation is related to a trim steady state. After introducing small perturbations to the state variables and the control inputs, the Taylor series technique is used to expand the non-linear equations

around the trim condition. Then higher-order terms are neglecting and a linear system is obtained.

The linear model of the ADMIRE was used to design the baseline flight control system as it was mentioned before. Furthermore, it has been utilized to analyze the linear stability of the open-loop plant to get more information about the dynamics that shall be identified by the neural models. A linear stability analysis is enclosed in the following section.

3.4 ADMIRE Dynamics Stability Analysis

While it might seem counterintuitive, fighter aircraft are often intentionally designed to be inherently unstable. The main objective is to enhance the maneuverability unstable aircraft are more responsive to control inputs, allowing for quicker and more aggressive turns and maneuvers. This is crucial for combat situations where agility is paramount. To prevent uncontrolled flight, fighter aircraft incorporates *Fly-by-Wire*. This advanced flight control system uses computers to process sensor data and send commands to the aircraft's control surfaces, effectively stabilizing the aircraft in real-time.

The goal is to reach a balance between instability for maneuverability and stability for control. Too much instability can lead to uncontrollable flight, while too much stability can hinder maneuverability. The design process involves careful considerations of these factors to achieve the desired level of performance.

Numerous stability analysis techniques exist such as: *Linear Stability Analysis*, *Lyapunov Stability Analysis*, *Frequency Stability Analysis*, *Numerical Simulations...* On the aerospace sector, the most common approach is to perform linear stability analysis over all the aircraft flight envelope and confirm the results with non-linear simulations.

A typical definition of the stability of a linear system, with particular reference to the aeroplane, may be described as below:

Lemma 3.4.1. *A system which is initially in a state of static equilibrium is said to be stable if, after a disturbance of finite amplitude and duration, the response ultimately becomes vanishingly small.[Michael, 2012]*

The small-perturbation equations of motion of an aircraft are linear, consequently, it is implied that the disturbing input must also be small to preserve that linearity. When, larger input disturbances are applied to the linear model, some degradation in the interpretation of stability from the observed response must be anticipated. However, according to [Michael, 2012], "*for most applications this does not give rise to major difficulties because the linearity of the aircraft model usually degrades relatively slowly with increasing perturbation amplitude. Thus it is considered reasonable to use linear system stability theory for general aircraft applications*".

Many modern combat aircraft as the ADMIRE concept, which depend on their flight control systems for achieving their normal flying qualities, present, under certain conditions, substantial non-linearity in their dynamics behaviour. For example, for large amplitude

manoeuvring at the extremes of the flight envelope, the aerodynamic properties are clearly non-linear for these kind of aircraft. Nevertheless, in the present project, only soft maneuvers and flight conditions far from envelope limits will be considered. Therefore, linear analysis can be performed on the range of those equilibrium points.

3.4.1 Linear stability analysis

In this section, an analysis of the poles of the ADMIRE dynamics is exposed. Those are computed using the decoupled dynamics of the aircraft and calculated for different points of the flying envelope. In particular, a sweep analysis was done using the following grid:

$$M = \begin{bmatrix} 0.25 & 0.5 & 0.7 & 0.9 & 1 & 1.1 & 1.2 \end{bmatrix} \text{ [-]}$$

$$h = \begin{bmatrix} 5000 & 7500 & 10000 & 12500 & 15000 \end{bmatrix} \text{ [ft]}$$

It should be noted that the stability analysis has been done to the open-loop dynamics, *i.e.*, the FCS was not included on the studied plant.

Figures 3.2 and 3.3 represent the evolution of the lateral-directional poles when the altitude and the Mach are fixed, respectively. The reader can infer that the lateral-directional dynamics is always stable within the flying envelope, *i.e.*, the real part of all poles are negative. On the other size, figures 3.4 and 3.5 show the same kind of analysis but for the longitudinal dynamics of the fighter. In this case, it can be confirmed that the longitudinal response is unstable because, at least one pole, is placed on the positive real plane.

From the previous stability analysis can be infered that if the longitudinal dynamics of the ADMIRE is not augmented, the response of the system diverges. Thus, the dynamic responses considered to identify the ADMIRE dynamics are the obtained from the augmented system due to it is useless to include in the dataset unstable responses of the open-loop plant. Those data would only cause noise to the target identification models and, furthermore, an unstable flight is not a real condition during nominal operation.

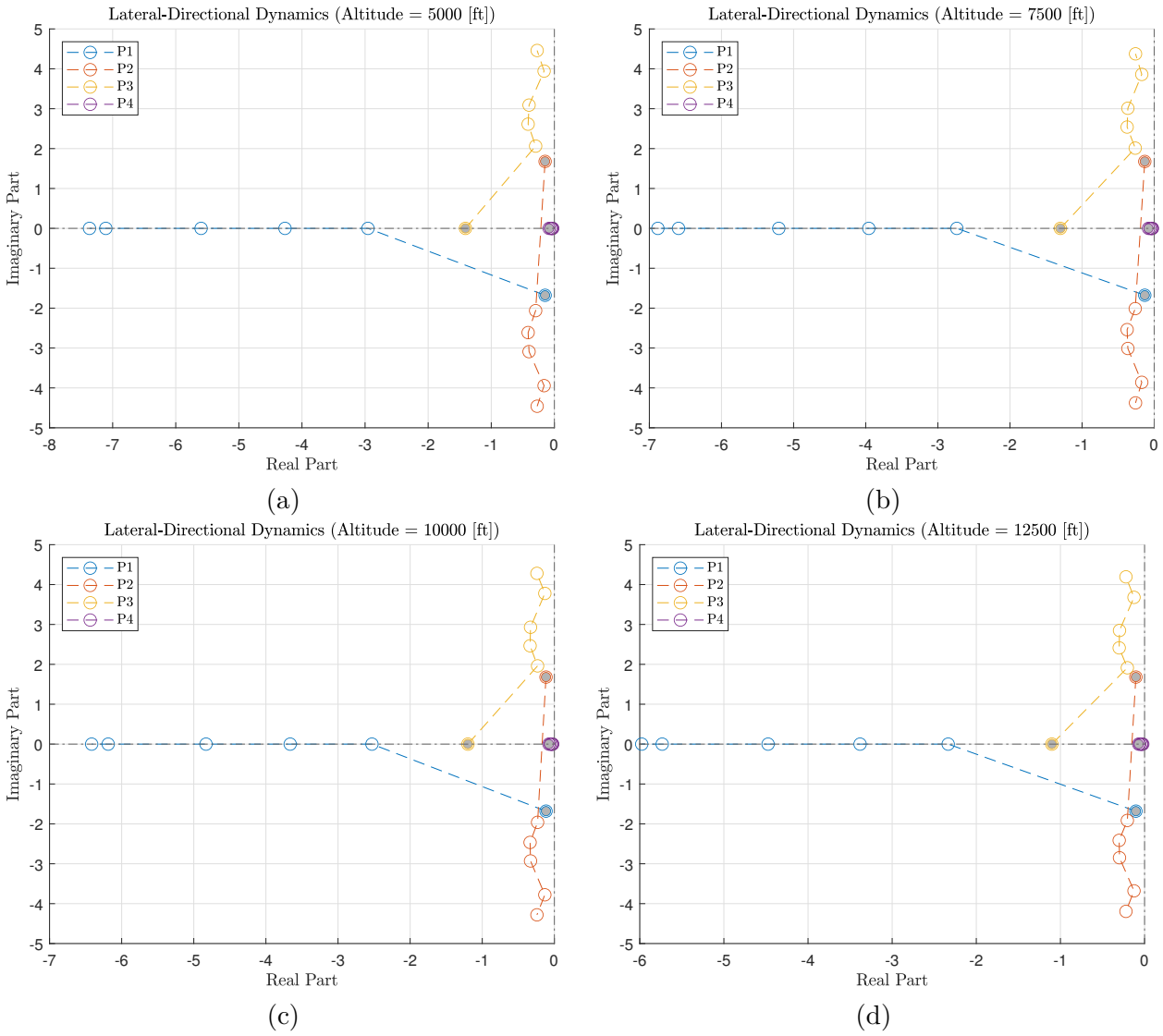


Figure 3.2: ADMIRE Lateral-Directional Eingenvalues at constant altitude and Mach = [0.25, 0.5, 0.7, 0.9, 1, 1.1, 1.2] (first Mach condition is painted in grey)

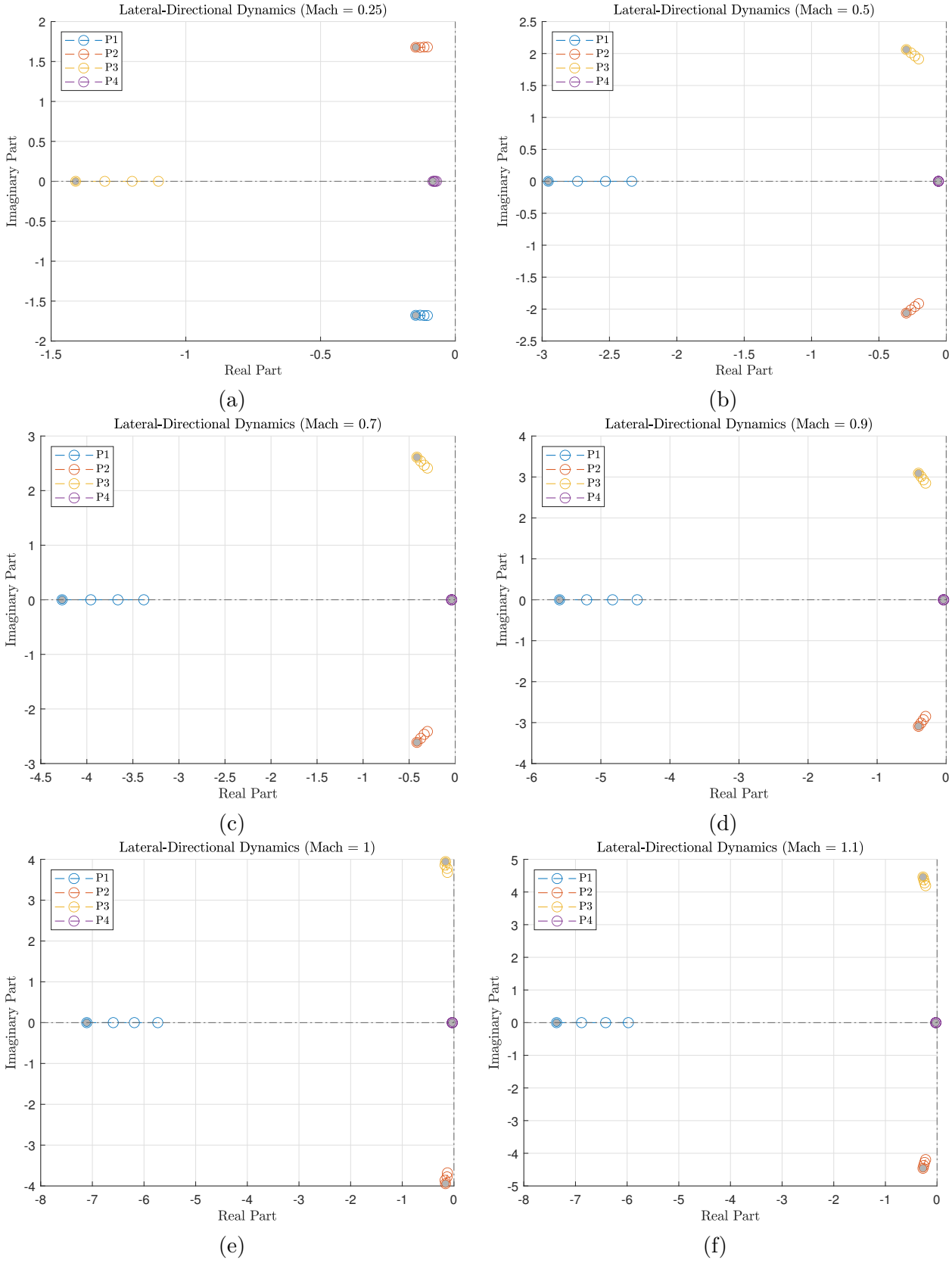


Figure 3.3: ADMIRE Lateral-Directional Eingenvalues at constant Mach and altitude = [1000, 3000, 6000] [m] (first altitude condition is painted in grey)

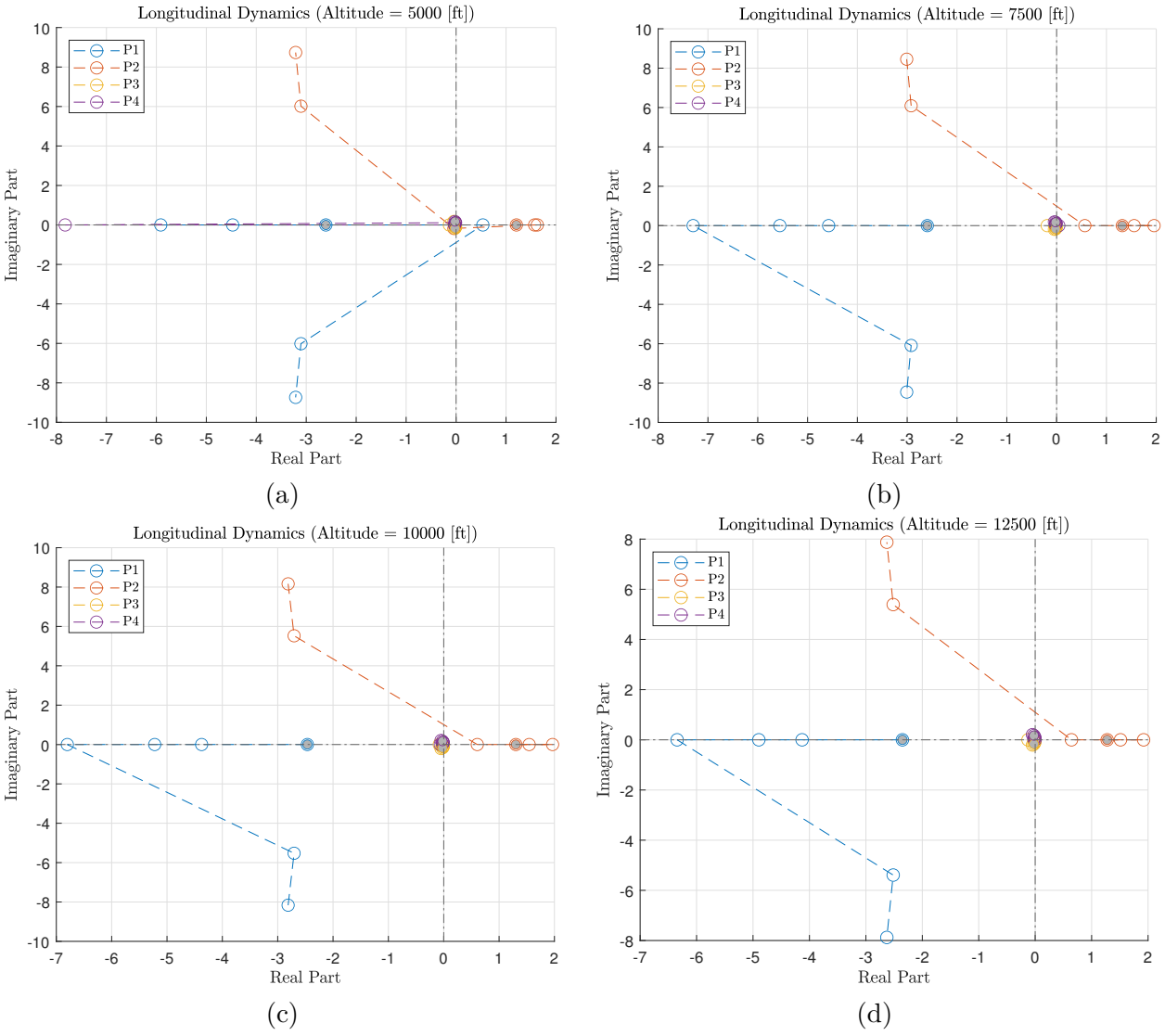


Figure 3.4: ADMIRE Longitudinal Eingenvalues at constant altitude and Mach = [0.25, 0.5, 0.7, 0.9, 1, 1.1, 1.2] (first Mach condition is painted in grey)

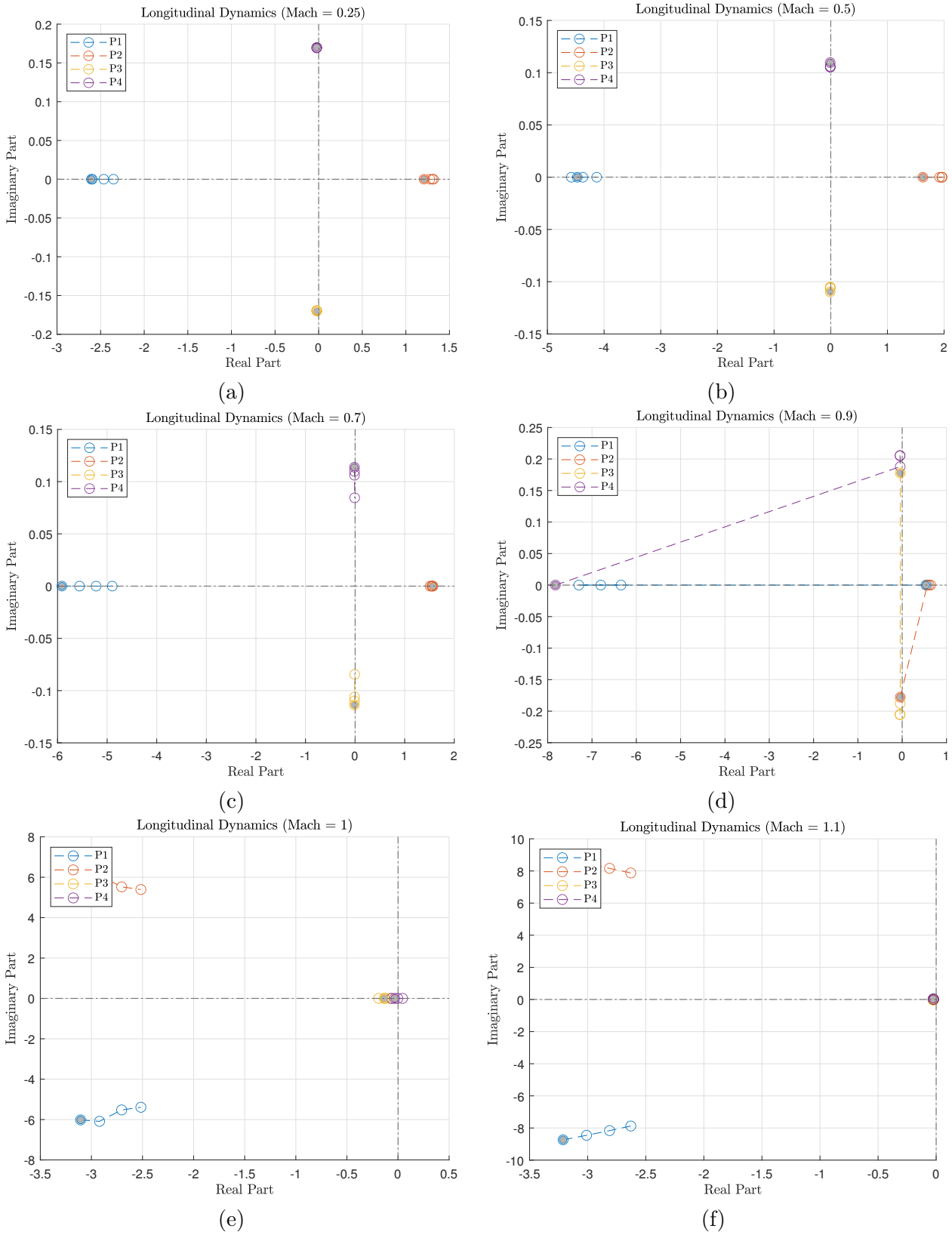


Figure 3.5: ADMIRE Longitudinal Eingenvalues at constant Mach and altitude = [1000, 3000, 6000] [m] (first altitude condition is painted in grey)

Chapter 4

Simulation Environment

The simulation environment developed within the scope of this project is based on the ADMIRE simulator. This 5th fighter model was designed by the former Aeronautical Research Insitute of Sweeden (FFA) and it is described in the chapter 3.

In this chapter, the architecture of the simulation model and its implementation is exposed. Additionally, as the original objetive of simulation test bench was to design and validate flight control laws, several modification had to be implemented on the Simulink model to fulfill the requirements of a Big Data analysis as the studied in this project.

Finally, several MATLAB pipelines has been developed to optimize the synthetic data generation, neural models training and their validation process.

4.1 ADMIRE Simulator

The ADMIRE Non Linear Simulator (NLS) is an open-source simulation environment that was developed by using the MATLAB/Simulink framework and where multiple C-dependencies are integrated. In particular, the equations which describes the non linear response of the A/C were implemented in a C library which is called from the Simulink model at each time step of the simulation. Respect to the baseline Flight Control System, the control laws were designed directly on Simulink, except for the Control Allocation module that was coded using C. This module, called Flight Control Selector within the original model, is responsible for calculating actual actuators commands based on the A/C state demands (e.g., the demanded pith rate, q_{dmd}) computed from the Control Laws. Furthermore, it shall divide the control effort between the available effectors for each control channel and it is function of the flight condition. A functional diagram of the simulator architecture can be found in the figure 4.1.

Unfortunately, although the tool is widely robust and powerful, it was not designed to run massive simulations as it is required from the point of view of the Artificial Intelligence and the Big Data analysis. Therefore, a new version of the simulation environment has been developed in the scope of this project to provide to it these capabilities. In particular, the complete Input/Output interface of the Simulink model was updated to use Simulink

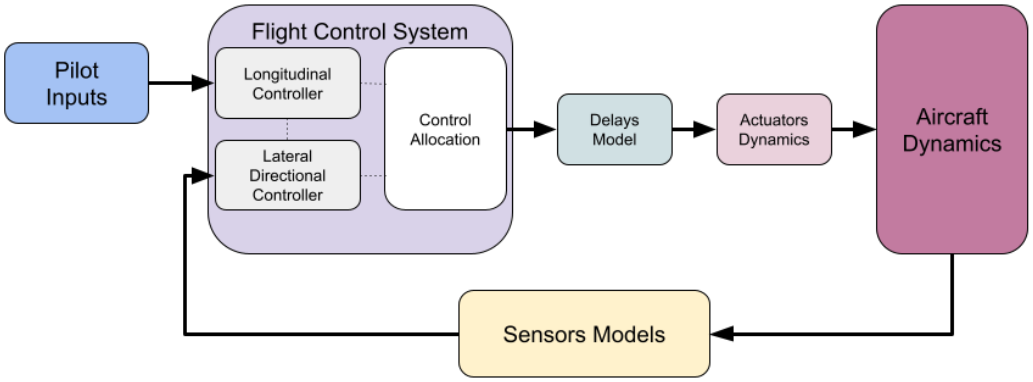


Figure 4.1: ADMIRE simulator architecture

Bus Objects and a specific toolchain, based on the Simulink-API, has been developed. The objective of this new tool, together with the new I/O data structures, is to enhance the definition of the pilot inputs, allowing the user to design custom stick maneuvers in an easier way and to automatize the simulation (or test cases) executions and the final processing of the resulting data, which will be used for the design of the neural networks. A detailed explanation can be found on the following section.

4.2 Synthetic data generation pipeline

The synthetic data generation pipeline has been built following a modular architecture. Thus, the user is able to add his custom maneuver (stick movements and speed autopilot input sequences) by creating a simple MATLAB function where the desired control inputs shall be defined as MATLAB *timeseries* objects. Once the input sequences are created, they shall be assigned to corresponding variables (within the PILOT_INPUTS bus object) in the Input Test Case template that the tool generate automatically from the model I/O interface. This template, that corresponds to a *Simulink dataset* object, is used to input the demand sequence during the simulations. It should be remarked that as the Input Test Case template is generated from the model during the pipeline execution, the user is even able to modify the I/O interface of the Simulink model and continue using this toolchain to generate synthetic data.

The user may decide to update the model and code custom maneuvers functions but, the current version of the toolchain includes already programmed functions to define periodic input signals and custom temporal inputs defined with respect to breakpoints. In other words, by the default, the tool allows the user to create sine, square and pulse inputs or specify a sequence of (*Target Amplitude*, *Settling time*, *Hold time*) breakpoints to define a custom temporal maneuver. These input modes have been used to create the baseline dataset of this project.

Although the synthetic data of the fighter aircraft is considered valid because the core model, the ADMIRE aircraft, is an aeronautical project well known by the academia and the industry, since the simulator has been upgraded to be compatible with the requirements of a Big Data workflow, a validation process has been performed. Unfortunately, real data was

not available, therefore the validation was based in the engineering judgement. In particular, several maneuvers were simulated and the corresponding aircraft responses were analyzed in accordance with the selected inputs.

In the figure 4.2, the reader can observe a slow sinusoidal demand on the pitch rate control stick (longitudinal movement of the stick). Initially, the pilot pushed the longitudinal stick (negative stick position) to demand $q_b < 0$. According to the A/C body frame of reference, a negative pitch angle, θ, means that the fighter flies down. As it was expected, the aircraft altitude was decreased and its speed, in terms of Mach in the figure, increased. When the stick position began to be positive (pull), the flight control system commanded the corresponding deflections to the flaps and the foreplane in order to get a positive pitch rate. Once θ was positive again, then the fighter began to increase its altitude and reducing speed because, although the airspeed autopilot was active, it was not able to correct the speed fast enough with respect to the accelerations caused by this maneuver. It should be remarked that the reader may notice some delay between the input signal and the aircraft response. Those delays correspond to the FCS processing time and the actuators responses, together with the own A/C dynamics characteristics times. Finally, the lateral and pedals inputs, and their characteristics angular responses, are also included in the figure. Those were only displayed in order to show that the lateral-directional dynamics of the aircraft is stable and that longitudinal demands do not modify the lateral dynamic behaviour. It can be concluded that the periodical input, generated by the toolchain, produce an expected dynamics response of the fighter from the point of view of the flight mechanics. Thus, the method to build periodic signals as pilot inputs and the simulation process and later data management, within the toolchain, are verified and validated.

As it was mentioned before, another useful way to define input demands is to specify a set of breakpoints which are composed by the desired amplitude, the settling time to reach the target value and the hold time that the input shall be kept once the desired value is acquired. The figure 4.3 shows a pull-push maneuver where the pilot moved the stick longitudinally (pull) up to 20 deg in 2 seconds, hold the demand during 1 second and then, pushed the stick up to -20 deg using a symmetric movement. Finally, the pilot set the stick in the trim position (0 deg). Analyzing the pitch rate demand, the reader can confirm that the angular rate was increased and decreased according to the pilot demands and that the aircraft altitude and speed evolved consequently. It can be noted that, although the demand is symmetric, the final pitch of the fighter is not the trim value (the equilibrium point of the aircraft at the initial flight condition). This is explained because the forces generated by the control surfaces are not the same if the deflection is positive or negative due to their airfoils (*i.e.*, the speed profile of the flow around the airfoil differs). Thus, as the absolute maximum pitch rate reached during the push maneuver was less that the maximum value obtained during the pull demand, the final aircraft pitch was increased with respect to the trim state. Then, the aircraft was flying up after the pilot maneuver and the altitude was increasing continuously. It should be pointed out that in these simulations, the speed autopilot, that it is part of the simulator FCS, was active and set to a constant value. Therefore, although the final θ was slightly greater than the trim value and it produced a moderated rate of climb, the speed autopilot was able to modify the thrust fast enough to compensate the decrease of airspeed, V_t, due to the rise, maintaining the demanded airspeed. Nevertheless, the Mach was lightly increased because of its inverse dependency with the sound speed. It can be

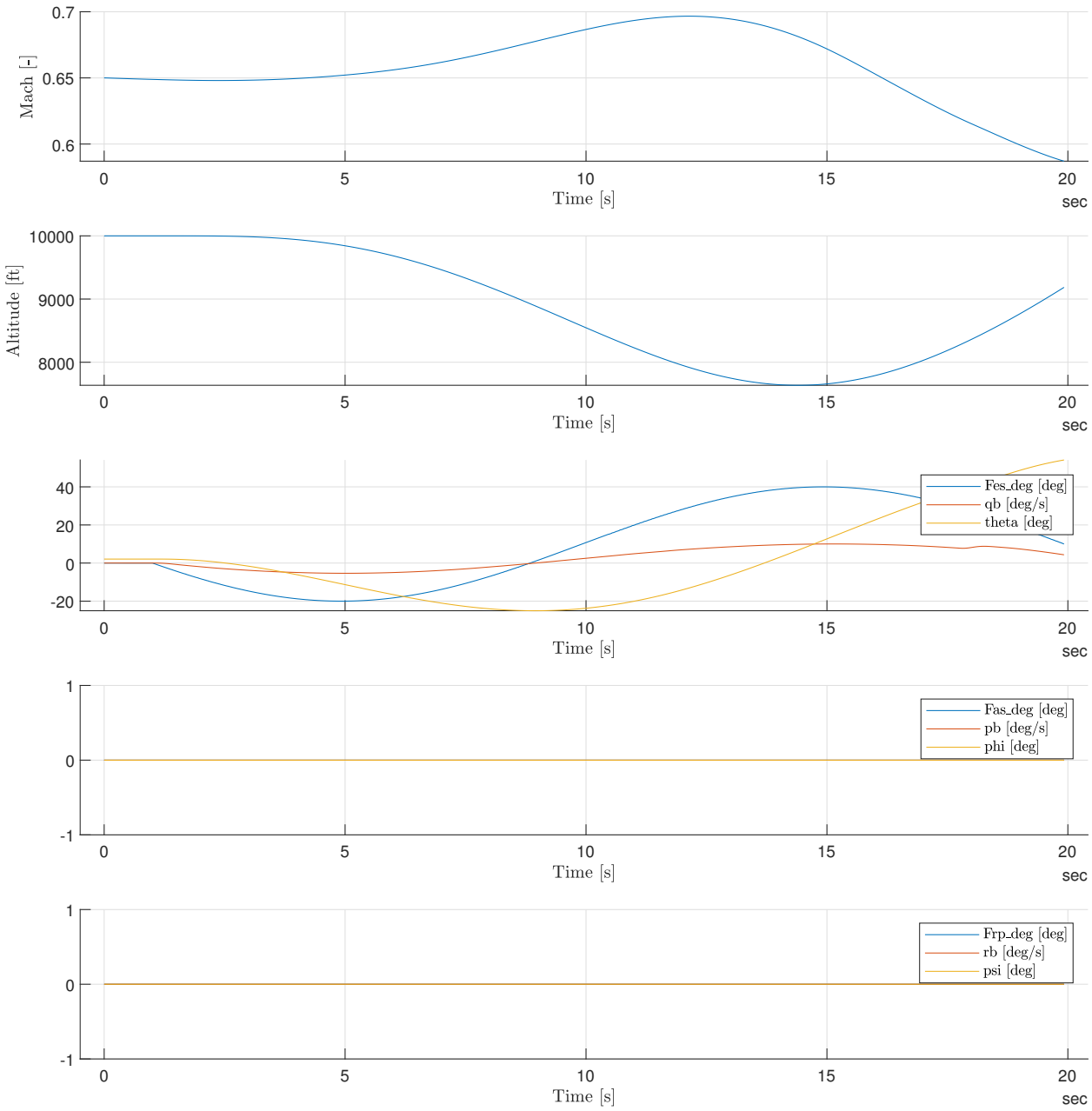


Figure 4.2: ADMIRE response for a sinusoidal demand on pitch rate control stick (F_{es})

concluded that the results of this simulation were the expected from the flight mechanics perspective and the method to specify the input demand by breakpoints is verified and validated.

Once the data generation pipeline was verified, the following step was to prepare a set of characteristic maneuvers in order to prepare the datasets that would be used for the neural networks training. A *Testing Dataset* based on the longitudinal dynamic response of the ADMIRE was prepared to continue with the current study. Detailed information about it can be found in the appendix A.1 and, in particular, the table A.1 lists the maneuvers simulated. The reader can observe that all the maneuvers consisted in longitudinal movements of the stick at a typical flight condition. The complete set of inputs was gen-

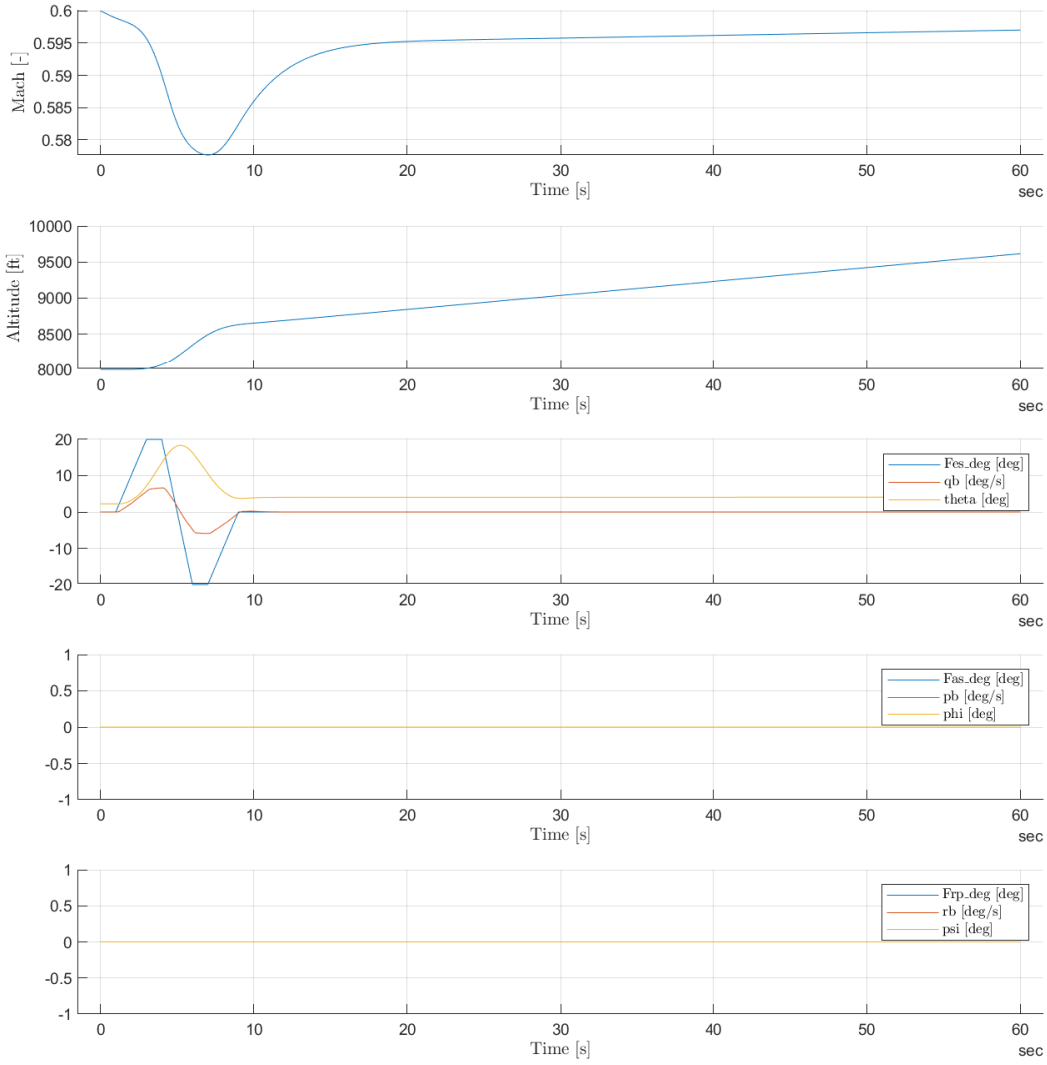


Figure 4.3: ADMIRE response for a symmetrical pull-push demand on pitch rate control stick (F_{es})

erated using the pipeline built-in functions, *i.e.*, the periodic input signal builder and the breakpoints-sequence modeller explained above.

Lastly, the following step was to code a Neural Network training pipeline to optimizate the design and validation process of the considered architectures. A detailed explanation of this training pipeline can be found in the next section.

4.3 Training Pipeline

The objetive of the training pipeline is to provide to the user an efficient way to define multiple neural architectures and get the corresponding trained models without additional

manual steps. It allows the designer to queue different network trainings from an unique definition scripts. Each study is defined by a pre-defined MATLAB function pattern which is the input of the pipeline. It should be noted that the term study makes reference to a set of different neural network configurations that are queued to the pipeline process.

The definition function is characterized by returning a cell vector which contains all the NN configuration that are going to be trained. Each cell element corresponds to a model definition. It contains a nested cell array which includes all the required information to configure the neural network, *i.e.*, the kind of neural architecture (FNN, FNN/NARX, LSTM...), the size of the hidden layers (number of hidden layers and their sizes), the activation functions used and, naturally, the number of inputs and outputs which compose the model. Additionaly, training settings can be also modified from this definition pattern. An example and more information about these study definition files can be found in the appendix B.2.

In order to facilitate the parameters modifications in the definition files, auxiliary function were coded in order to call the target neural network classes from the MATLAB Deep Learning toolbox. In particular, specific functions were designed for:

- Feedforward Neural Networks (FNN)
- FNN/NARX Networks
- Recurrent Neural Networks (RNN)
- Long-Short Term Model Networks (LSTM)
- Clockwork Recurrent Neural Networks (CW-RNN)

All these *load_nn_architecture* functions return the corresponding MATLAB network object, *i.e.*, *network* class for FNN, NARX and RNN models, and *dlnetwork* class for the LSTM and CW-RNN cases, with the specific configuration provided by their input arguments. The use of one or other class is only related with the MATLAB toolbox implementation who, initially, defined the *network* class for the classical NN approach and, since some years ago, introduzed the *dlnetwork* object to be able to implement modern deep learning architectures and be compatible with widely extended Deep Learning frameworks used by other languages, mainly, Python.

Each *load network* method define a set of common modificable parameters such as: the number of hidden layers and their sizes, the I/O definition or the training settings (optimization algortihm, maximum number of epochs...). Moreover, dependant on the type of neural network, specific variables can be edit, *e.g.*, the clock rates on the CW-RNN architectures. More information and an example of the *load functions* can be found in the appendix B.1.

It is important to remark that the definition cases may also specify custom methods in order to prepare the data for the training. MATLAB provides a built-in method to prepare a dataset based on the *network* class information. However, this method is not available for the modern *dlnetwork* objects. Therefore, if the proposed architecture is not compatible with the internal method *preparets()*, a custom preparation function shall be created and

its path shall be added to the corresponding field of the structure which return the *load function*. Nevertheless, it does not add extra effort for the designer because the pipeline is able to identify if custom preparation is required and, then, run the user-defined MATLAB function without interrumping the computation flow.

Once the definition file is ready, the pipeline will iterate over each network configuration, preparing the data and splitting them into training and testing dataset. Then, the training process will be run using parallell processing if the required toolbox is available on the computer. The current version of the tool parallelize the training of each network but it should be remarked that if a computing cluster is available, minimal modification are required to send batches of the input architectures sets.

Finally, after reaching one of the "finish training" criterias (*Max Epochs*, *Min Gradient...*), the trained model is saved together with a validation dataset that has not been used for training. All validation or postprocessing functions has been implemented in an isolated pipeline.

4.4 Post-Processing Pipeline

An isolated *Post-Processing Pipeline* was developed in order to run validation tests on the trained models and plot figures of interest for the study. The model files obtained from the *Training Pipeline* are input to the MATLAB program and the following outputs are computed:

- Performance metrics: MAE, MSE, RMSE and Huber.
- Time responses plots
- RMSE and Training Time Comparation plots
- Latex Tables and Figures

Multiple performance metrics were considered due to the different information that may provide each formula. A description of each metric can be found below:

- *MAE*: The MAE is a simple yet powerful metric used to evaluate the accuracy of regression models. It is calculated as the average absolute difference between the predicted values and the actual target values. Due to any square is applied, this metric weights errors uniformly without considering their direction. It is useful to unserstand the magnitude of the errors. Below, the reader can find the mathematical formula:

$$MAE = \frac{\sum_{i=1}^{n} \|\hat{y}_i - y_i\|}{n}$$

  where n is the number of data points, $\hat{y}_i$ the estimated value for the i-th prediction and y_i the actual value.

- *MSE*: The MSE is a measure of quality of an estimator derived from the square of the Euclidian distance. It measures the average squared difference between predicted values and the actual values in the dataset. Its main benefits with respect to other

metrics are that it penalizes large errors more heavily than small errors making it sensitive to the outliers and it is mathematically convenient for optimization problems. It is computed as follows:

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

where n is the number of predictions, $\hat{y}_i$ the estimated value for the i-th prediction and y_i the actual value.

- *RMSE*: It corresponds to the square root of the average of the squared differences between the predicted and actual values, *i.e.*, the square root of the *MSE* metric. It can be mathematically represeted as:

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

  The main advantages are similar to the *MSE* case but the unit of the *RMSE* is the same as the orginal data, making it relatively interpretable.

- *Huber*: It is a variation of the *MSE* metrics in order to make it less sensitive to outliers. The Huber loss is a combination of both linear and quadratic scoring methods and is quite a robust metric for evaluating Regression models. It is achived by using a piecewise function which is quadratic for small values and linear for the larger ones. It can be expressed mathematically as:

$$L_{delta}(\widehat{\mathbf{y}}, \mathbf{y}) = \begin{cases} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (\hat{y}_i - y_i)^2 & \text{for } |a| \leq \delta \\ \frac{1}{n} \sum_{i=1}^{n} \delta \cdot \left(|\hat{y}_i - y_i| - \frac{1}{2}\delta \right) & \text{otherwise} \end{cases}$$

  The hyperparameter δ can be tuned according to the data.

Respect to the output figures, the user can defined the desired plots and a custom Latex Writer object is used to get the corresponding Latex commands. This Latex Writer class, developed within the scope of this project contains generic methods to write TEX tabular and long tables and Figures/Subfigures orders.

All those methods have allowed to analyze multiple studies quickly and perform a commond analysis for the different network architectures evaluated.

4.5 Additional tools developed for this project

The objective of this project is to use neural network strategies to identify aircraft dynamics. Within the aerospace industry, it is very common to use the Simulink environment to implement the aircraft mathematical models. In other words, it may be interesting to be able to deploy the designed neural networks as Simulink blocks model. MATLAB includes an internal method to generate a Simulink model from a *network* object but the result is a static model, *i.e.*, the weights and biases are implemented using constant blocks. It is has a significatly limitation if the user wants to design an online training loop for the NN because the resulting blocks model. Thus, a generalization of this method was also developed in the scope of this project.

The new tool allows the user to input a MATLAB *network* object and generate the equivalent Simulink model in which, now, the tuneable parameters are inputs of the resulting block. Therefore, the user can choose between connect to the neural network architecture the constant weights and biases provided by a offline training or to implement a more complex logic to adapt the parameter values dynamically.

Chapter 5

Neural Dynamics Identification

Several neural architectures has been selected to evaluate their performance identifying the ADMIRE dynamics behaviour. It should be noted that the datasets used for this task were generated using the baseline control laws of the fighter, *i.e.*, despite of the inherited instability of the its longitudinal dynamics in open-loop, all the simulated responses were stable due to the stability augmentation system (SAS) which modifies the ADMIRE behaviour on close-loop. Thus, the neural model designed in the scope of this project are only capable to work within the same range of the baseline SAS.

Initially, standard Feedforward Neural Networks were considered. Then a later modification of them, the recurrent networks, were evaluated to compare their performances. Moreover, multiple sweep analysis were run in order to assess the variation of precision when the number of hidden units and the number of training epochs were increased.

Before the assesment, it should be explained the meaning of the model names because they contains significant information about the specific network configuration used. The generic pattern is as follows:

(#NNType)_TE(#E)_I(#P)O(#M)_(#N)_ID(#ID)_FD(#FD)

where *#NNype* refers to the type of base neural network (*e.g.*, FNN, NARX...), *#E* provides the number of training epochs used, *#P* and *#M* are the number of inputs and outputs respectively, and *#ID* and *#FD* correspond to the delay steps computed for the inputs and feedback states respectively. Nevertheless, some fields may be ommitted to reduce the name length when those mantain constant their values (*e.g.*, _I(#P)O(#M)).

Finally, it should be noted that only the longitudinal dynamics of the fighter was considered in the scope of this project. It is due to the time required to train complex and large neural models that could not be afforded by the available hardware. Thus, the presented estimators output the longitudinal states and are fed by input aircraft states that affects the longitudinal dynamics.

5.1 Feedforward neural network (FNN)

The first architecture evaluated in this project was the the classical Feedforwad Neural Network, FNN, with one hidden layer. This choice was based on the universal approximation theorem, first proved in [Cybenko, 1989], which states that a shallow neural network (with one hidden layer, containing a finite number of neurons) with a non-polynomial activation function can approximate any function.

The analysis has been divided into two study levels. The first analysis, *Study A*, simplifies the identification problem and considers only the pitch rate dynamics. This way, the number of parameters which forms the neural network is reduced and, therefore, the computational complexity or effort required is minimized. It has allowed to discard quickly neural architectures which were not able to identify the desired non-linear dynamics. In fact, below, the reader can observe how several Static and Dynamic FNN were evaluated in the first study but discarded for the following more complex studies. Finally, the *Study B* aims to identify the longitudinal dynamics of the ADMIRE (Airspeed, Angle of Attack and Pitch Rate).

5.1.1 Study A - Pitch Rate Channel Identification Model

As first approach, and in order to validate the proposed architecture, a simplification of the problem was done. Instead of trying to estimate the complete longitudinal dynamics, only the pitch rate channel was used to build the neural network. Nevertheless, multiple configurations were tested, included some which include the rest of longitudinal states as inputs of the neural networks. Additionally, the testing dataset was used. As it was mentioned in A.1, this dataset was generated by simulating multiple longitudinal maneuvers but using an unique trim envelope condition, Mach = 0.65 and Altitude = 10 kft.

Two strategies can be followed to design FNN-based regression models: *Static regression model* and *Dynamic regression model*. As for the traditional regression problems, an static approach includes just contemporary relations between the independent variables (also called explanatory variables) and the desired response (dependent variable).

$$y_t = f\left(x_{1,t}, x_{2,t}, ..., x_{k,t}\right) \tag{5.1}$$

On the other hand, a dynamic regression model uses relations between current and lagged values of the explanatory variables.

$$y_t = f\left(x_{1,t}, x_{1,t-1}, ..., x_{1,t-n}, ..., x_{2,t}, x_{2,t-1}, ..., x_{2,t-n}, ..., x_{k,t}, x_{k,t-1}, ..., x_{k,t-n}\right) \tag{5.2}$$

Additionally, the dynamic models may include feedback, *i.e.*, the regression values are computed using current and lagged explanatory values together with previous states of the model output,

$$y_t = f\left(y_{t-1}, ..., y_{t-m}, ..., x_{1,t}, ..., x_{1,t-n}, ..., x_{2,t}, ..., x_{2,t-n}, ..., x_{k,t}, ..., x_{k,t-n}\right) \tag{5.3}$$

In this project, although the literature has concluded numerous times that the performance of feedback-in-the-loop dynamic regression models are superior to the static approach

as it is denoted in [Altun, Efe, 2019 6th International Conference on Control, and Information Technologies (CoDIT), 2019], and thanks to the flexible and robust design toolchain developed within the scope of this project, it has been decided to analyze both strategies to know the differences when those models are applied to a non-linear system as the ADMIRE fighter.

A summary of the FNN architectures evaluated can be found in the table 5.1.

Models	**#P**	**Conv.**	**MSE**	**MAE**	**RMSE**	**Huber**
FNN_I8O1_20	201	No	0.18959	12.7022	0.43541	0.18959
FNN_I8O1_20_10	401	No	0.095793	6.6493	0.3095	0.095793
NARX_I9O1_10_ID1_2_FD1_2_OL	201	Yes	4.58e-07	0.023141	0.00067676	4.58e-07
NARX_I9O1_10_ID1_FD1_OL	111	Yes	5.1851e-06	0.090459	0.0022771	5.1851e-06
NARX_I9O1_20_ID1_2_FD1_2_OL	401	Yes	1.804e-06	0.055798	0.0013431	1.804e-06

Table 5.1: FNN-based Pitch Rate Identification Models (I/O requirements analysis) - Performances Summary

The reader can observe in the previous table that multiple Input/Output (I/O) configurations were evaluated. The first two models correspond to a Feedforwad Neural Network with 8 inputs. Besides, the proposed architecture is based on the static regression model without any feedback, *i.e.*, the inputs contemporary values with respect to the output, the pitch rate of the aircraft, and it is not fed back. Mathematically, both model can be expressed as:

$$q_b[k] = f\left(\delta_{lc}, \delta_{rc}, \delta_{loe}, \delta_{roe}, \delta_{lie}, \delta_{rie}, M, h\right) \tag{5.4}$$

where $q_b[k]$ is the current pitch rate, $f\left(...\right)$ is the resulting neural model and the meaning of the input arguments of the function is:

- δ_{lc}: Left canard position at current time step [deg]
- δ_{rc}: Right canard position at current time step [deg]
- δ_{loe}: Left outer elevon position at current time step [deg]
- δ_{roe}: Right outer elevon position at current time step [deg]
- δ_{lie}: Left inner elevon position at current time step [deg]
- δ_{rie}: Right inner elevon position at current time step [deg]
- M: Flight Mach number at current time step [-]
- h: Flight altitude at current time step [m]

The difference between them is that the first model is composed by an unique hidden layer of size 20, while the second proposal consists in a network of two hidden layers with sizes of 20 and 10 neurons, respectively. It can be concluded that the error metrics were significantly greater than the rest of the listed proposals. Thus, the non-linear dynamics of the pitch rate channel cannot be identify by using the I/O pattern.

After discarding the static and non-feedback regression approach, multiple models based on the dynamic regression function together with the feedback of the previous steps of the output were assessed. In particular, they all can be mathematically writen as:

$$q_b[k] = f\left(\delta_{lc}, \delta_{rc}, \delta_{loe}, \delta_{roe}, \delta_{lie}, \delta_{rie}, M, h, q_b[k-1]\right) \tag{5.5}$$

where $q_b[k]$ is the current pitch rate, $f\left(...\right)$ is the resulting neural model and the meaning of the input arguments of the function is:

- δ_{lc}: Left canard position at current time step [deg]
- δ_{rc}: Right canard position at current time step [deg]
- δ_{loe}: Left outer elevon position at current time step [deg]
- δ_{roe}: Right outer elevon position at current time step [deg]
- δ_{lie}: Left inner elevon position at current time step [deg]
- δ_{rie}: Right inner elevon position at current time step [deg]
- M: Flight Mach number at current time step [-]
- h: Flight altitude at current time step [m]
- $q_b[k-1]$: Measured pitch rate at the previous time step [rad/s]

It can be concluded that all the Nonlinear Autoregressive Exogenous, NARX, models reached errors several order of magnitude lower than the FNN models. It agrees the exposed in [Altun et al., 2019] which explains that is required to include at least the last previous output state as input in order to identify the non-linear dynamics of an air-vehicle. Thefore, this approach was considered as a baseline requirement for the rest of the project and the most advanced network architectures. Additional variation of the input and feedback delays were analyzed but non significantly advantages were identified.

The pitch rate neural estimator was used as reference in order to find an appropiate I/O pattern and validate the design toolchain and process. Nevertheless, the aim of this work was to identify the longitudinal dynamics of the ADMIRE. Below, the reader can find the proposal to scale the problem from the pitch rate channel to the complete longitudinal state model.

5.1.2 Study B - Longitudinal Channels Identification Model

Once the study A was finished and a first impression of the mandatory requirements to obtain an appropiate identification, the next step was to scale the FNN-based model to estimate the longitudinal dynamics of the aircraft. Thus, the dimension of the output layer was increased to 3. Moreover, as it was concluded in the previous analysis, a NARX configuration is required to be able to identify the aircraft non-linear dynamics. Thus, only this I/O configuration was considered from this point.

The mathematical model of the desired FNN-based estimator can be described as:

$$\begin{bmatrix} V_t[k] \\ \alpha[k] \\ q_b[k] \end{bmatrix} = f\left(\delta_{lc}, \delta_{rc}, \delta_{loe}, \delta_{roe}, \delta_{lie}, \delta_{rie}, M, h, V_t[k-1], \alpha[k-1], q_b[k-1]\right) \tag{5.6}$$

where $V_t[k]$, $\alpha[k]$ and $q_b[k]$ are the longitudinal state (airspeed, angle of attach and pitch rate) at the k-th time step, $f\,(...)$ is the resulting neural model and the meaning of the input arguments of the function is:

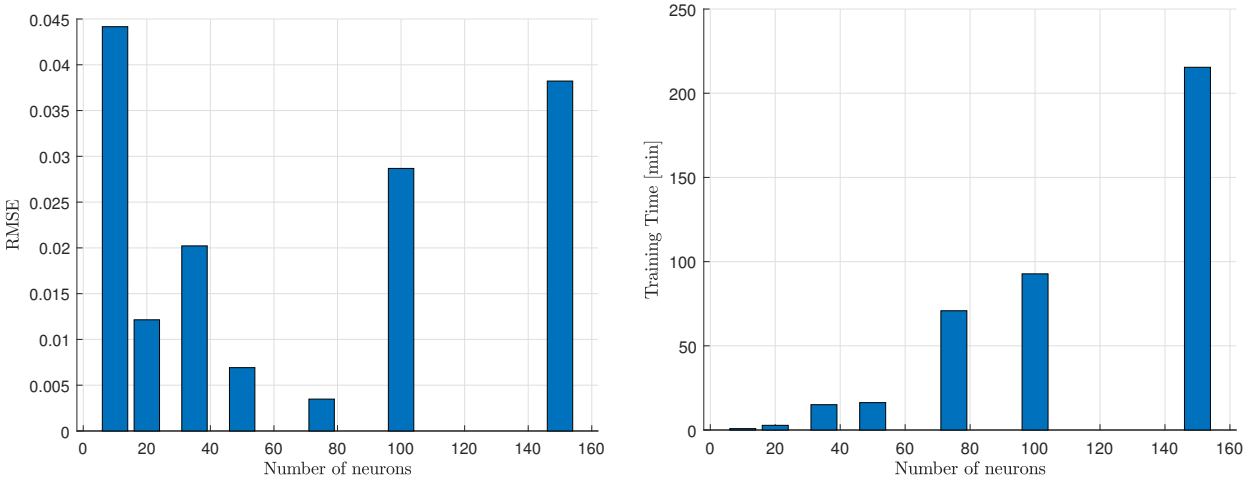
- δ_{lc}: Left canard position at current time step [deg]
- δ_{rc}: Right canard position at current time step [deg]
- δ_{loe}: Left outer elevon position at current time step [deg]
- δ_{roe}: Right outer elevon position at current time step [deg]
- δ_{lie}: Left inner elevon position at current time step [deg]
- δ_{rie}: Right inner elevon position at current time step [deg]
- M: Flight Mach number at current time step [-]
- h: Flight altitude at current time step [m]
- $V_t[k-1]$: Measured airspeed at the previous time step [m/s]
- $\alpha[k-1]$: Measured angle of attack at the previous time step [rad]
- $q_b[k-1]$: Measured pitch rate at the previous time step [rad/s]

First, a sweep analysis with respect to the number of neurons of the hidden layer was performed in order to conclude the optimal size required for the idetification task. Table 5.2 compiles the obtained performance metrics and the number of learnable parameters (weights and biases) that composed the network. It can be infered that the more precise model, considering the training setting used (Levenberg-Marquardt algorithm and 200 epochs), was the NARX/FNN model composed by 75 neurons. Indeed, in this case, all the performance metrics considered indicated this result. Furthermore, the figure 5.1 shows the variation of the RMSE and the Training Time with respect to the number of neurons of the hidden layer. In terms of required training time, it can be observed that there exist an exponential, but almost linear, relationship between the number of neurons and the time required to train the network. As the computational power is very limitated in this project, and due to the great precision gotten, the later analysis will be done selecting a size of 75 neurons.

Before continuing with the current analysis, it should be noticed that the previous metrics describe a overall behaviour of the models. It may occur that the model is able to estimate very well the stacionary states of the dynamics responds but to be characterized by a poor performance during the transient state, *i.e.*, the maneuvering. Then, it is mandatory to analyze the time response of the system before validating the proposed model. Figures 5.2 and 5.3 show the estimated longitudinal responses with respect the actual ones. The reader can confirm that the model performances are significantly good during all the evaluated

Models	#P	MSE	MAE	RMSE	Huber
NARX/FNN_TE200_10_ID1_FD1	163	0.0019501	3.1095	0.04416	0.0019501
NARX/FNN_TE200_20_ID1_FD1	323	0.00014749	0.7236	0.012145	0.00014749
NARX/FNN_TE200_35_ID1_FD1	563	0.00040877	1.5547	0.020218	0.00040877
NARX/FNN_TE200_50_ID1_FD1	803	4.7862e-05	0.32421	0.0069183	4.7862e-05
NARX/FNN_TE200_75_ID1_FD1	1203	1.2124e-05	0.17298	0.003482	1.2124e-05
NARX/FNN_TE200_100_ID1_FD1	1603	0.00082258	1.5353	0.028681	0.00082258
NARX/FNN_TE200_150_ID1_FD1	2403	0.001461	1.6497	0.038223	0.001461

Table 5.2: NARX/FNN-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Performances Summary



(a) (b)

Figure 5.1: NARX/FNN-based Longitudinal State Identification Models - Number of Hidden Units Sweep Overview

maneuvers. Thus, it can be concluded that a NARX/FNN model composed by one unique hidden layer of size 75 is able to fullfil the identification task for the longitudinal dynamics of the ADMIRE.

It is important to remark that the training settings, *i.e.,* the optimization algorithm and the training parameters, also called hyperparameters, affect the performance results of the networks. Thus, a second sweep analysis was done in order to study the relationship between the network performance and the maximum number of training epochs selected. For that, the model size selected previously was used as baseline. The table 5.3 lists the obtained validation metrics for the selected neural network when the training epochs were set to 25, 50, 100 and 200. Additionally, the figure 5.4 represents the evolution of the RMSE with respect to the number of epochs and the required training time. From both souces, it can be concluded that the more training epochs, the more precise is the resulting estimator, but the more time is required to train the model. From the designer perspective, an error order of 10^{-4} could be enough for the expected task, in other words, 100 training epochs shall be sufficient to get a valid model. Nevertheless, this decision is deeply related with the available hardware setup and dataset size. It is possible that if the dataset is increased, for instance, including additional flight conditions, a larger error would be obtained and 100

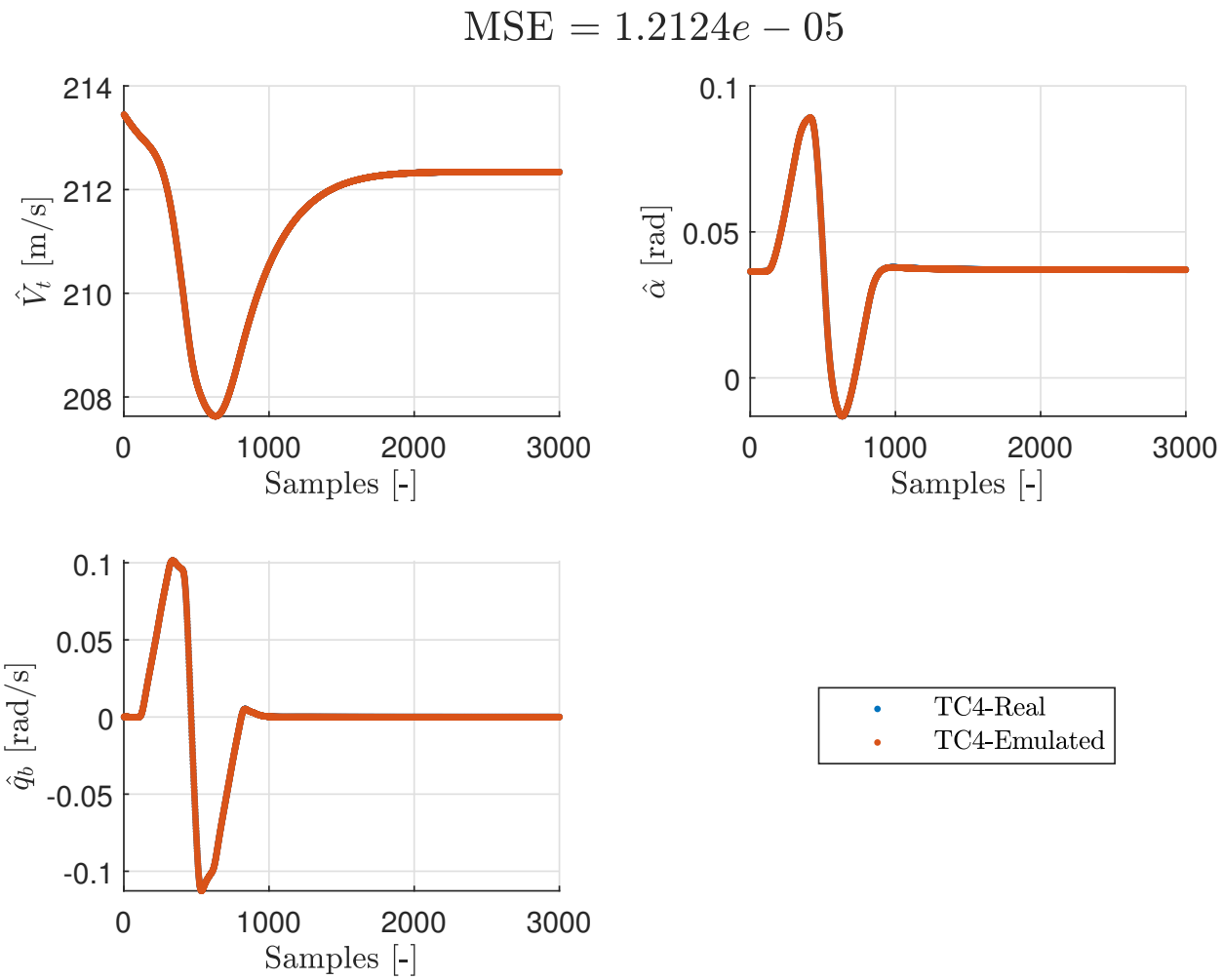


Figure 5.2: NARX/FNN-based Longitudinal State Emulator (Neurons = 75) - Validation (TC4)

training epochs would not be enough to get a valid model. However, the proposed working flow and the current design pipelines elaborated within the scope of this project would allow the designer to conclude valid settings for the new problem.

Models	**#P**	**MSE**	**MAE**	**RMSE**	**Huber**
NARX/FNN_TE25_75_ID1_FD1	1203	0.021204	9.5882	0.14562	0.021204
NARX/FNN_TE50_75_ID1_FD1	1203	0.013559	9.0264	0.11644	0.013559
NARX/FNN_TE100_75_ID1_FD1	1203	0.00060688	1.9059	0.024635	0.00060688
NARX/FNN_TE200_75_ID1_FD1	1203	1.2124e-05	0.17298	0.003482	1.2124e-05

Table 5.3: NARX/FNN-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Performances Summary

For all the above reasons, the candidate neural network to estimate the figther response using a FNN/NARX architecture is the model composed by one only hidden layer of size 75, using *tansig* as activation function and an *purelin* output layer which computes the three longitudinal states of the aircraft: V_t, α and q_b. Besides, the input layer is included delayed inputs (1 step) and, according to the NARX architecture, the last output state.

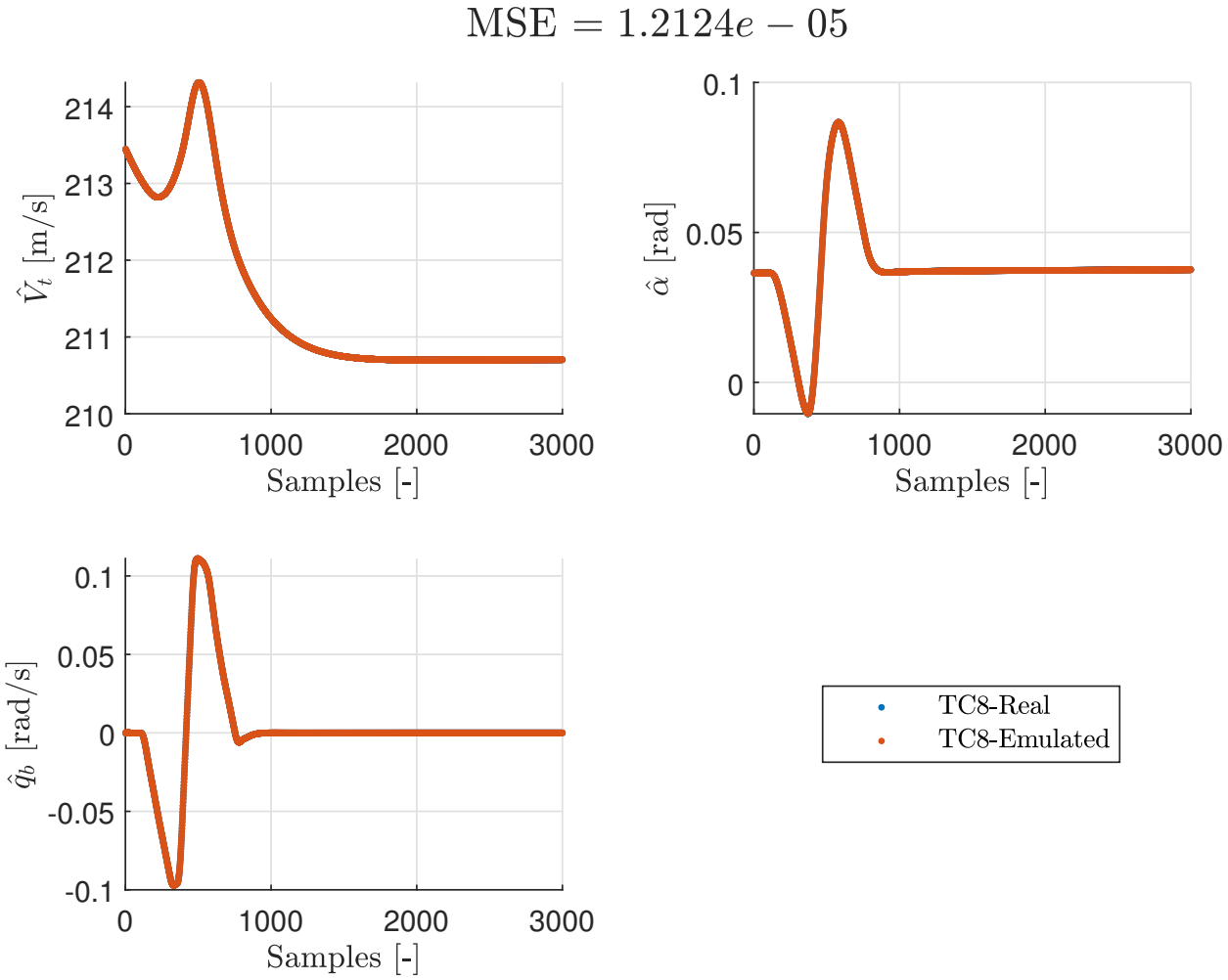
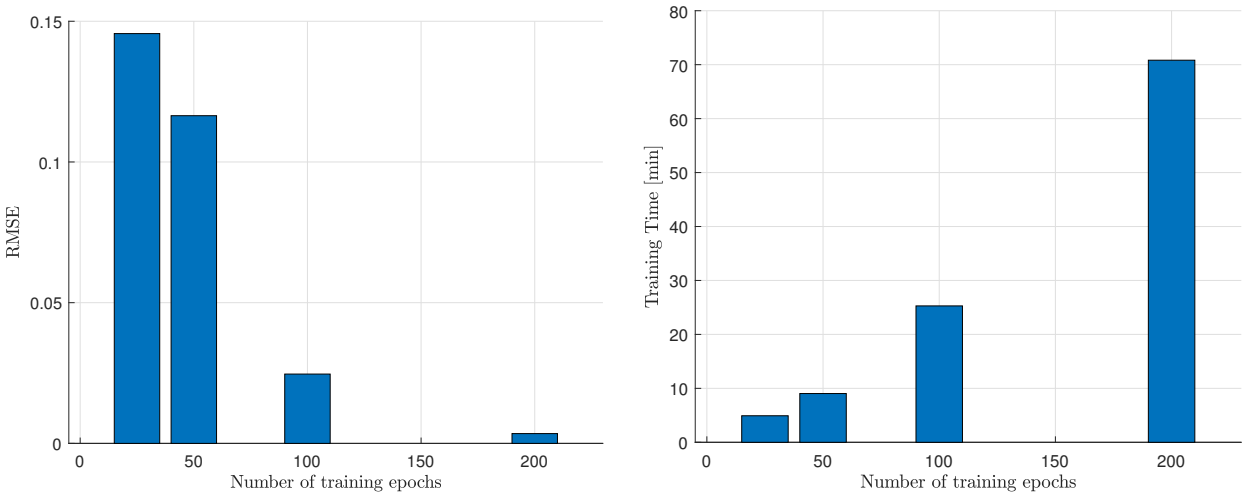


Figure 5.3: NARX/FNN-based Longitudinal State Emulator (Neurons = 75) - Validation (TC8)



(a) (b)

Figure 5.4: NARX/FNN-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Training Epochs Sweep Overview

5.2 LSTM Network-based Identification Model

The Long Short-Term Memory (LSTM) neural networks or layers are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. As it was explained on the section 2.2.1, this kind of recurrent network is widely used in the industry and in the academia because, on the contrary of the traditional RNN, it is able to learn long-term dependencies and dealing with the inherit gradient vanishing problem of the RNNs. The LSTM models address this problem by introducing a memory cell, which is in charge of hold information about previous states for an extended period. Therefore, LSTM architectures are well-suited for tasks such as language translation, speed recognition, and time series forecasting.

MATLAB provides this kind of neural architecture as a layer object within its Deep Learning toolbox. Unlike the traditional FNN architectures, which was initially integrated in MATLAB as *network* objects, the LSTM architecture is coded as a *layer* object which can be used to create the *dlnetwork* object. It means that the designer can define multiple LSTM layers, each one with a user-defined number of neuron units, and/or connect them to other kind of network architectures or processing functions in a easy way. However, this different data objects used by the MATLAB toolbox required to upgrade the first version of the development toolchain in order to be compatible with both kind of data structures, keeping an unique toolchain which process both kind of MATLAB network equally.

As it was already concluded in the previous section, to identify the aircraft non-linear dynamics it is necessary to include as input the previous states of the desired target variables. In other words, the following proposed architectures use the inputs/outputs configurations of a NARX network.

Again, as first approach, only the pitch rate channel was estimated to confirm the validity of the network configuration and to get a first impression about the number of LSTM units and training epochs that are necessary to build a dynamics identification model based-on this type of neural networks.

Once the first study was finished, and because of the promising results, specific analyses were performed to get estimation models for the longitudinal dynamic state and for the complete dynamic state. All the studies are explained in detail below.

5.2.1 Study A - Pitch Rate Identification Model

The identification model proposed in this section is composed by a unique hidden layer of type LSTM and by an output fully-connected layer. Weights and biases of both layers were modified during the training processes.

The LSTM layer was configured to use a *tanh* activation function for the cell and hidden states and a *sigmod* activation function for gates. No variation of the activation function was performed in this analysis. On the other hand, the fully-connected layer, as its name denotes, connect all neurons of the previous layer to its own neurons and multiplies the input values by a weight matrix and then adds a bias vector.

The identification model can be mathematically described as:

$$q_b[k] = f\left(\delta_{lc}, \delta_{rc}, \delta_{loe}, \delta_{roe}, \delta_{lie}, \delta_{rie}, M, h, q_b[k-1]\right) \tag{5.7}$$

where $q_b[k]$ is the pitch rate at the k-th time step, $f\left(...\right)$ is the resulting neural model and the meaning of the input arguments of the function are:

- δ_{lc}: Left canard position at the current time step [deg]
- δ_{rc}: Right canard position at the current time step [deg]
- δ_{loe}: Left outer elevon position at the current time step [deg]
- δ_{roe}: Right outer elevon position at the current time step [deg]
- δ_{lie}: Left inner elevon position at the current time step [deg]
- δ_{rie}: Right inner elevon position at the current time step [deg]
- M: Flight Mach number at the current time step [-]
- h: Flight altitude at the current time step [m]
- $q_b[k-1]$: Measured pitch rate at the previous time step [rad/s]

It should be noted that if the variables are not followed by [...], it means that the value used is the corresponding to the current time step. The only exception is the target output $q_b[k]$ that was denoted to avoid confusion with respect to the pitch rate value used as input.

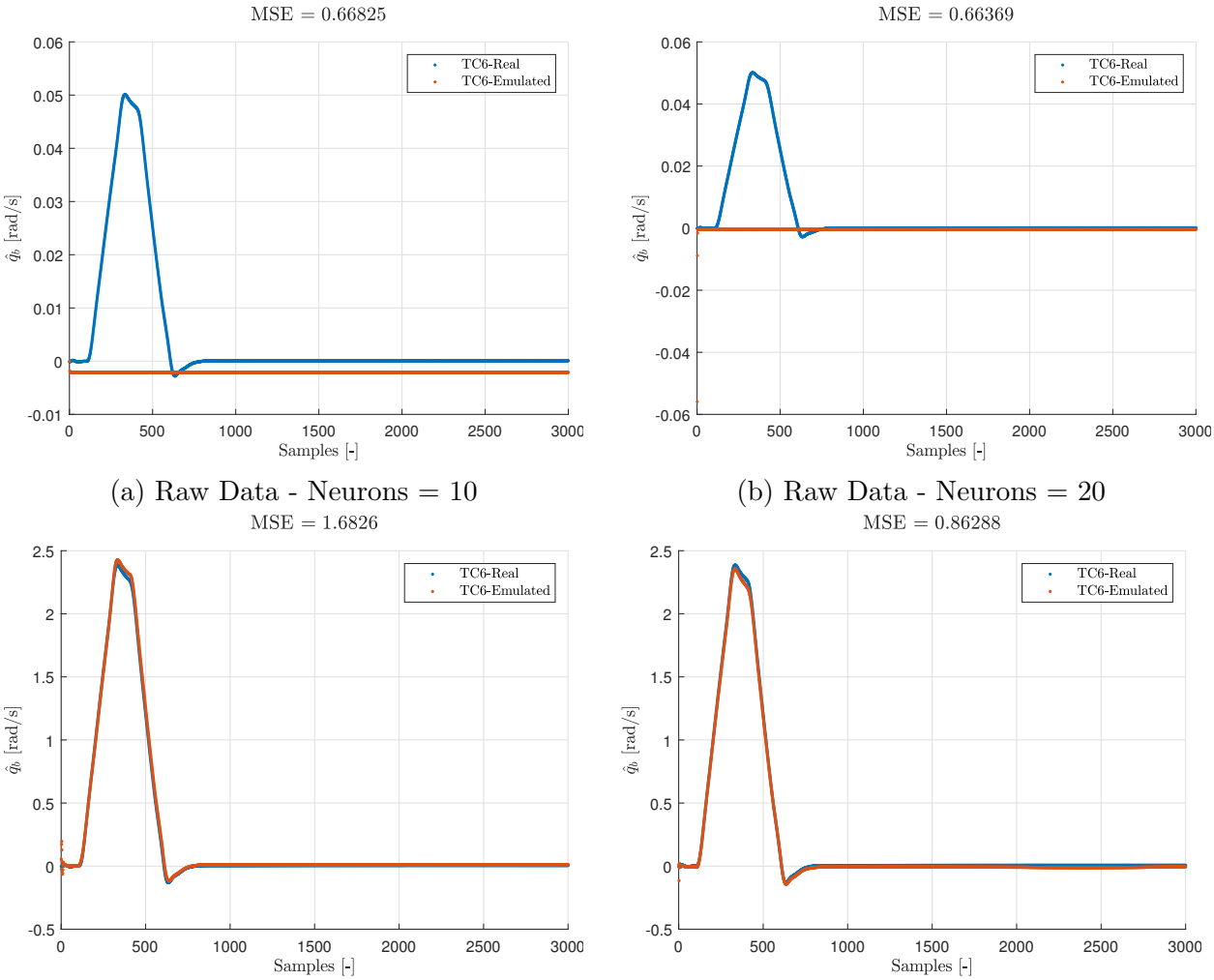
Two slight architecture were evaluated within this study. In particular, a network composed by 10 and 20 LSTM neuron units were trainned. In addition, both configuration were trainned with raw and standarized data in order to analyze the performance of using normalized/standarized data. The performance results of those models are listed in the table 5.4. It can be observed that the overall metrics seem to denote a better performance when the training data is not normalized. Besides, the model characterized by a greater number of neurons is slightly more precise. It should be noted that for this analysis the training epochs were restricted to 30 because the aim of this analysis were mainly evaluated the benefits of the normalization process.

Models	**#P**	**MSE**	**MAE**	**RMSE**	**Huber**
LSTM_TE30_I9O1_10	811	0.66825	25.9912	0.81746	0.66825
LSTM_TE30_I9O1_20	2421	0.66369	22.0763	0.81467	0.66369
LSTM_TE30_Norm_I9O1_10	811	1.6826	34.8156	1.2971	1.6826
LSTM_TE30_Norm_I9O1_20	2421	0.86288	35.0149	0.92891	0.86288

Table 5.4: LSTM-based Pitch Rate Identification Models - Performances Summary

The objective of this kind of model is to identify the dynamic response of the plant. Depending on the inputs, the response may be characterized by a initial transient and a long stacionary state. It this case, the previous metrics may show a good overall performance, however, the model may provide a poor result identifying the transient state. This behaviour was shown during the validation when the time responses were analyzed in detail.

Figures 5.6 and 5.5 represent the time response of the models with respect to the actual sequence. The reader can observe that the models which not standarize the training data were not able to identify the transient state. On the contrary, the normalized networks estimated correctly the transient state. In addition, it can be confirmed that at least 20 neurons are needed to identify the pitch rate response of the ADMIRE in a precise way. Thefore, as the literature recommended, a normalization function was applied to the data for the rest of the studies performed within the scope of this project.



(a) Raw Data - Neurons = 10

(b) Raw Data - Neurons = 20

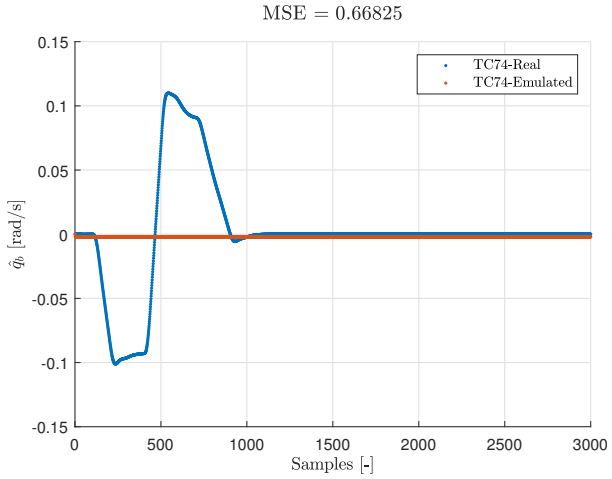
(c) Normized Data - Neurons = 10

(d) Normized Data - Neurons = 20

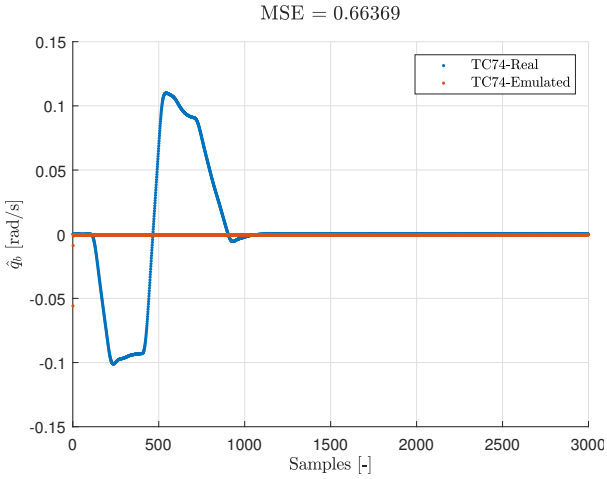
Figure 5.5: LSTM-based Pitch Rate Identification Models - Validation (TC6)

Notice that the normalization process has a deep impact in the performance metrics and that model with different normalization functions can not be compared in terms of their metrics because of the range of the data is modified.

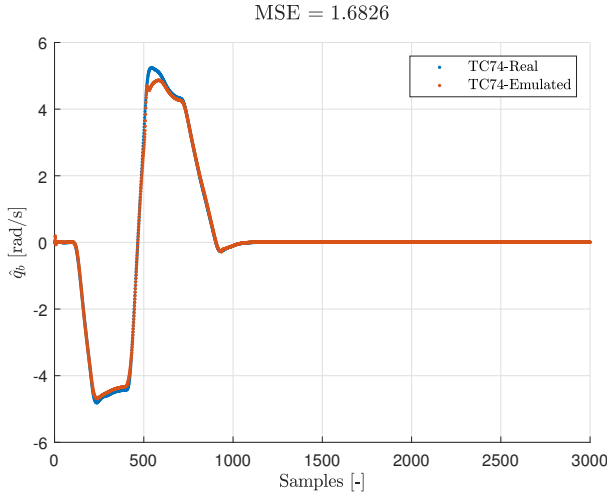
Finally, although the need of a normalization has been already demonstrated, it is interesting to analyze the effect of this preparation step on the training process. The figure 5.7 shows the evolution of the RMSE metric during the training process. The reader can infer that the use of normalized data reduces the noise in the training process, minimizing the erronous paths computed by the gradient algorithm, and, achieving a more efficient optimization process.



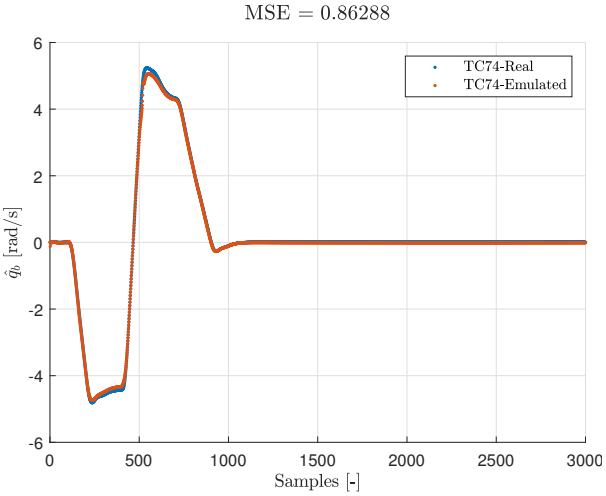
(a) Raw Data - Neurons = 10



(b) Raw Data - Neurons = 20

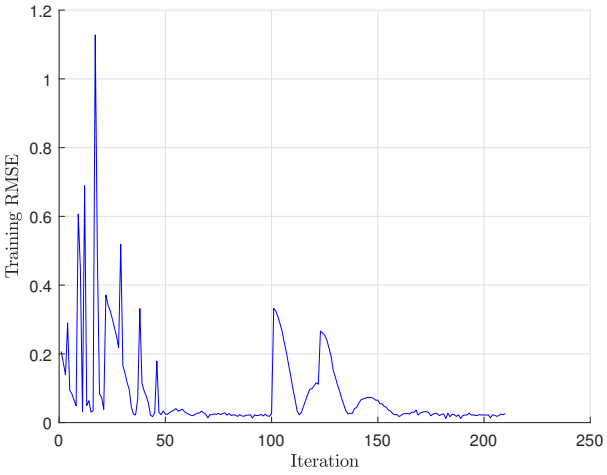


(c) Normized Data - Neurons = 10

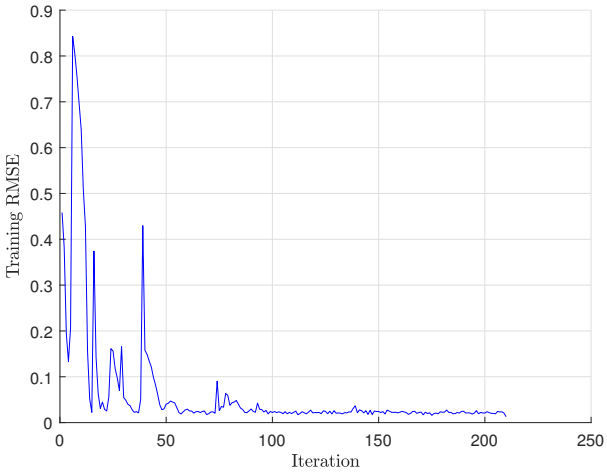


(d) Normized Data - Neurons = 20

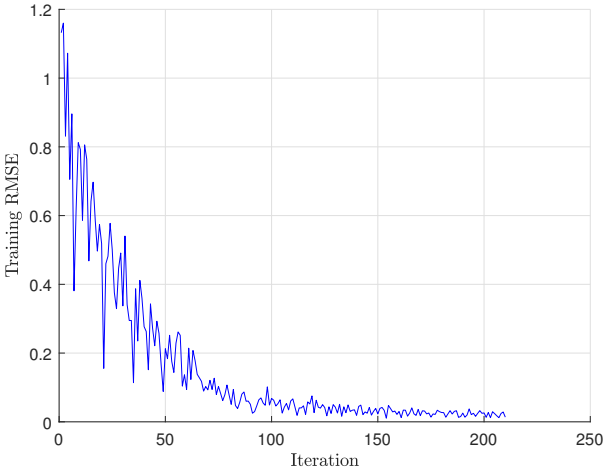
Figure 5.6: LSTM-based Pitch Rate Identification Models - Validation (TC74)



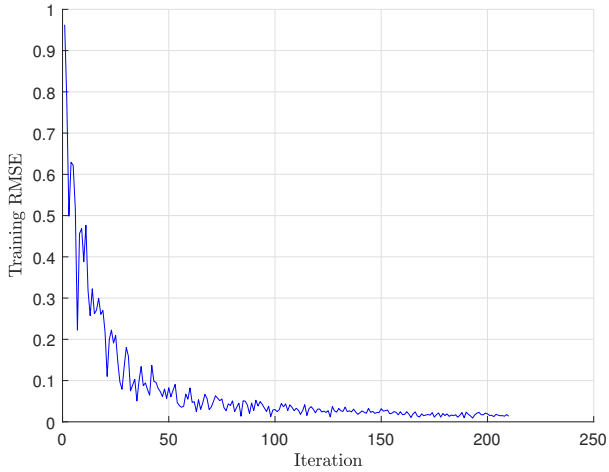
(a) Raw Data - Neurons = 10



(b) Raw Data - Neurons = 20



(c) Normalized Data - Neurons = 10



(d) Normalized Data - Neurons = 20

Figure 5.7: LSTM-based Pitch Rate Identification Models - Training History

5.2.2 Study B - Longitudinal State Identification Model

Once the LSTM-based Pitch Rate identification model was analyzed, the next step was to scale the estimator to try to identify the complete longitudinal dynamics of the ADMIRE. As in the previous section, the model proposed is composed by a unique hidden layer of type LSTM and by an output fully-connected layer. Weights and biases of both layers were modified during the training processes. The LSTM layer was configured to use a *tanh* activation function for the cell and hidden states and a *sigmod* activation function for gates. No variation of the activation function was performed during this analysis.

In this case, the output layer, *i.e.*, the fully-connected layer, shall estimate three variables. Thus, the model can be mathematically described as:

$$\begin{bmatrix} V_t[k] \\ \alpha[k] \\ q_b[k] \end{bmatrix} = f\left(\delta_{lc}, \delta_{rc}, \delta_{loe}, \delta_{roe}, \delta_{lie}, \delta_{rie}, M, h, V_t[k-1], \alpha[k-1], q_b[k-1]\right) \tag{5.8}$$

where $V_t[k]$, $\alpha[k]$ and $q_b[k]$ are the longitudinal state (airspeed, angle of attach and pitch rate) at the k-th time step, $f\left(...\right)$ is the resulting neural model and the meaning of the input arguments of the function are:

- δ_{lc}: Left canard position [deg]
- δ_{rc}: Right canard position [deg]
- δ_{loe}: Left outer elevon position [deg]
- δ_{roe}: Right outer elevon position [deg]
- δ_{lie}: Left inner elevon position [deg]
- δ_{rie}: Right inner elevon position [deg]
- M: Flight Mach number [-]
- h: Flight altitude [m]
- $V_t[k-1]$: Measured airspeed at the previous time step [m/s]
- $\alpha[k-1]$: Measured angle of attack at the previous time step [rad]
- $q_b[k-1]$: Measured pitch rate at the previous time step [rad/s]

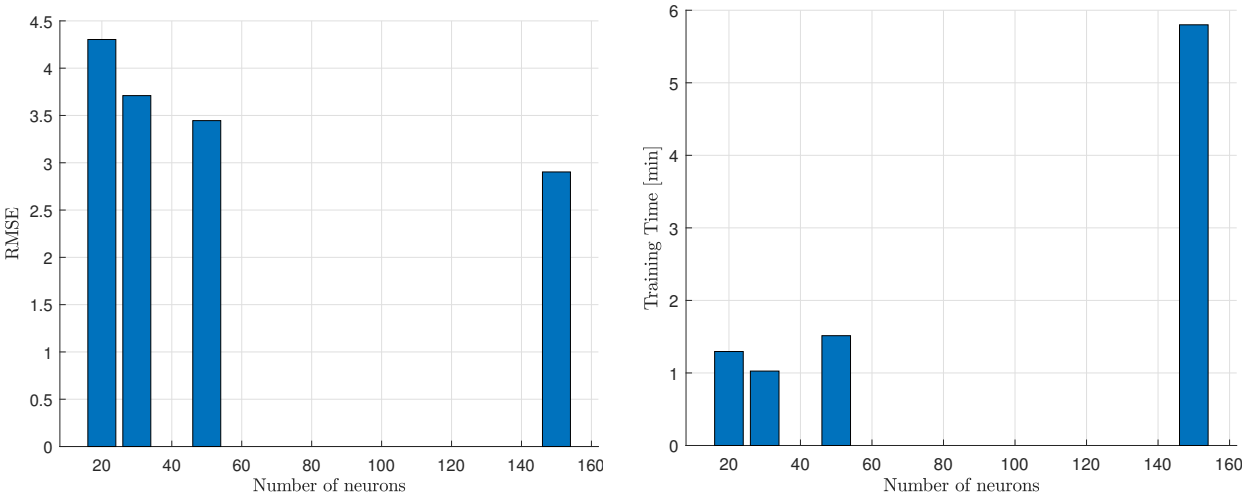
On the other hand, the conclusions deduced from the previous study were used as baseline for this analysis. Therefore, all the models presented here were trained and validated using normalized data. In addition, it was checked that 20 neurons provided an accurate model when only the pitch rate channel was considered. In order to confirm the minimum number of neurons required to estimate the longitudinal dynamics, a sweep analysis was performed using 20 neural units as initial suggestion.

The table 5.5 shows the performance metrics for the same type of neural network and training process but varying the number of LSTM units which compose the recurrent

layer. In particular, a sweep analysis was done and network of 20, 30, 50 and 150 neurons were evaluated within this study. The figure 5.8 represents the variation of the RMSE metric and the training time with respect to the selected number of neurons. Although the chosen sweep grid was not uniform, the bar plot denotes clearly how the model accuracy improves with the increment of the number of neurons. On the other hand, the slope of the resulting curve decrases in the same way. Focusing on the training time, it can be checked that there exist a quasi-linear dependency. It should be noted that the smallest model was the first in the training queue, *i.e.*, the parallel workers were set up during its training process. That is the reason why the 20-neurons model requested more time to train than the 30-neurons network.

Models	**#P**	**MSE**	**MAE**	**RMSE**	**Huber**
LSTM_TE100_20	2703	18.518	190.9339	4.3033	18.5168
LSTM_TE100_30	5253	13.7636	180.8322	3.7099	13.7636
LSTM_TE100_50	12753	11.8717	143.2295	3.4455	11.8717
LSTM_TE100_150	98253	8.4265	197.3835	2.9028	8.4265

Table 5.5: LSTM-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Performances Summary

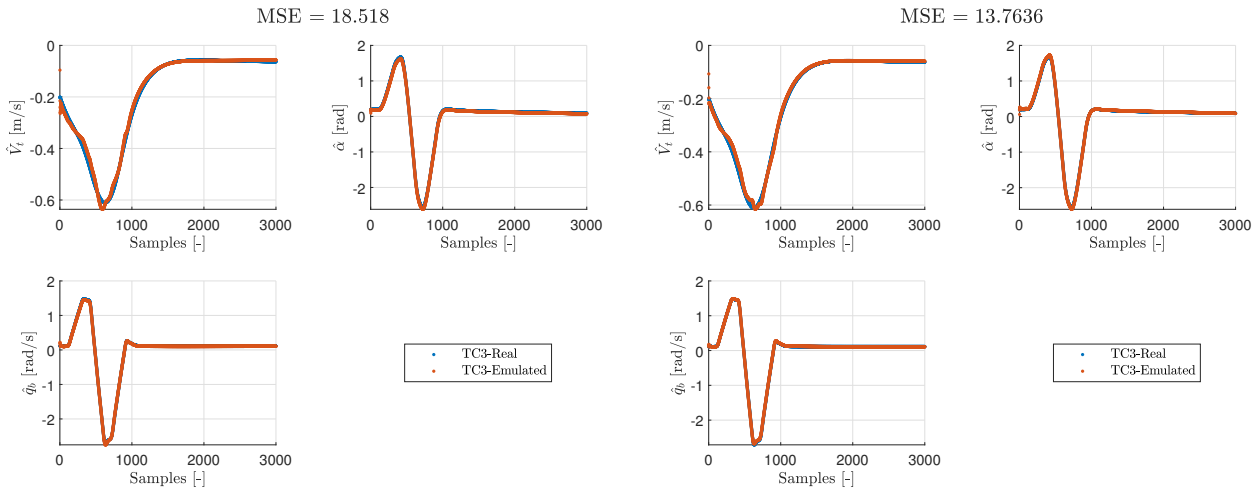


(a) (b)

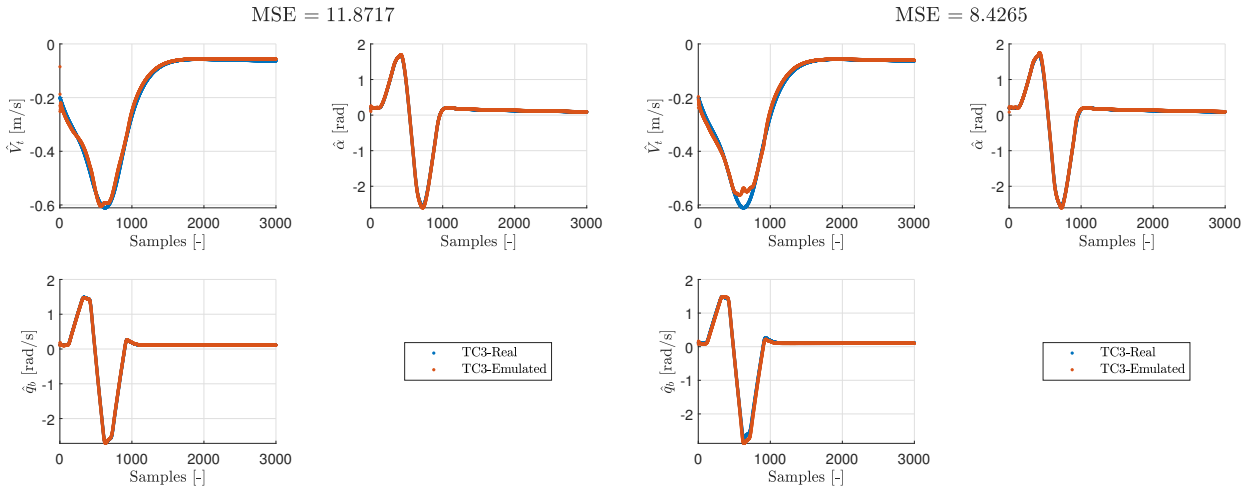
Figure 5.8: LSTM-based Longitudinal State Identification Models - Number of Hidden Units Sweep Overview

Figures 5.9 and 5.10 show the time response of the models for the test cases TC3 and TC4. Although, according to the performance metrics the largest model (Neuros = 150) is the most accurate, both figures shows that this model identified worse the transient state. Nevertheless, the most of the metrics were not able to characterize this behaviour because the model was more precise estimating the rest of the sequence than the slighter networks. Only the MAE formula denoted that the 150-neurons network was the worse model of the study. The reason of this behaviour is related to the amount of samples well-estimated with respect to the failing intervals. Even though the squared metrics, MSE, RMSE and Huber, should emphasized the large errors, such as the observed during the transient, the effect of small errors are also minimized by the square function. Therefore, as the estimation

error is very small during the most of the sequence, and the 150-neurons network was more precise for that interval, the resulting squared metrics are more affected by the small error minimizing behaviour. Then, for this kind of analysis, where the most of the sequence were very well fitted, the absolute error metric allowed to detect outliers.



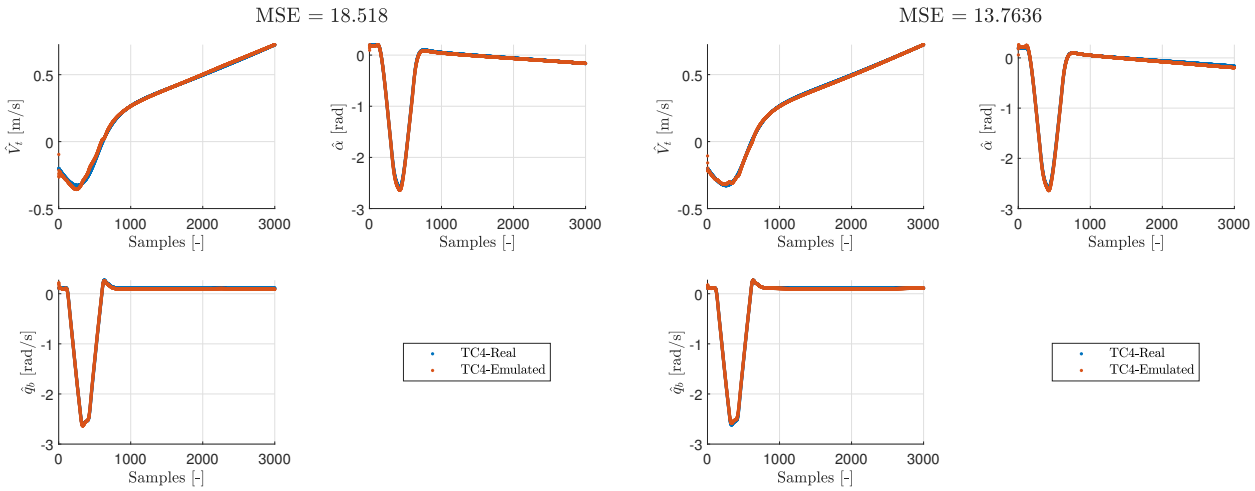
(a) Neurons = 20 (b) Neurons = 30



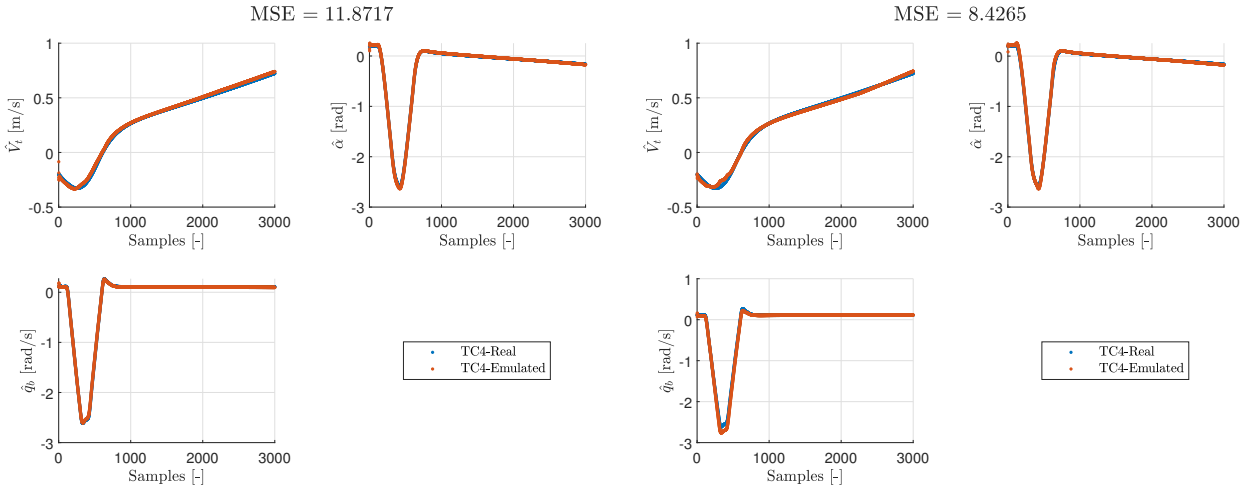
(c) Neurons = 50 (d) Neurons = 150

Figure 5.9: LSTM-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Validation (TC3)

The reader may think that the largest network should have approximate better the dynamic response of the aircraft. This idea is quite fair because a higher number of parameters to adjust the regression shall provide to the model a better robustness. Nevertheless, it should be kept in mind that the sweep analysis was performed by using a common configuration of the training options, in particular, a shared number of training epochs (Training Epochs = 100). It may occur that more training epochs were needed by the model in order to reach a global minimum of it loss function. Figure 5.11 shows the evolution of the RMSE metric during the training process. It can be observed that the 150-neurons model got worse during the last iterations/epochs of training. It can be caused because a larger model, *i.e*, a higher number of parameters to be fitted, complicates the optimization process, penalizing the robustness of the computed gradient path and increasing the sensitive of the model to



(a) Neurons = 20 (b) Neurons = 30

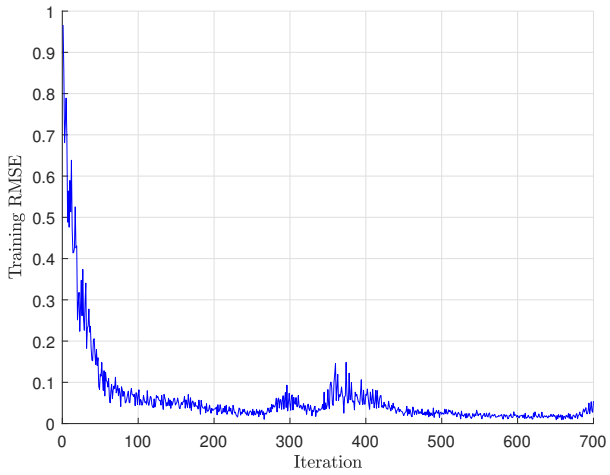


(c) Neurons = 50 (d) Neurons = 150

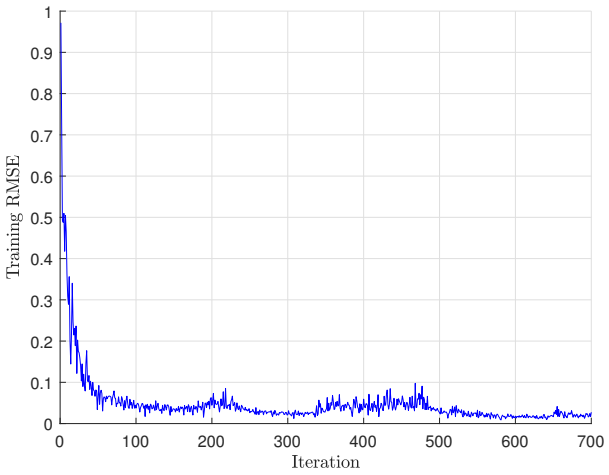
Figure 5.10: LSTM-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Validation (TC4)

the noise process. Nevertheless, these issues can be mitigated by increasing the number of training epochs in order to allow the algorithm to return to the global minimum area of the optimization hyperspace. Similar behaviour can be infered for the rest of models. However, the noisy transient occurred in the middle of the process and the optimization was able to converge again to more precise solutions.

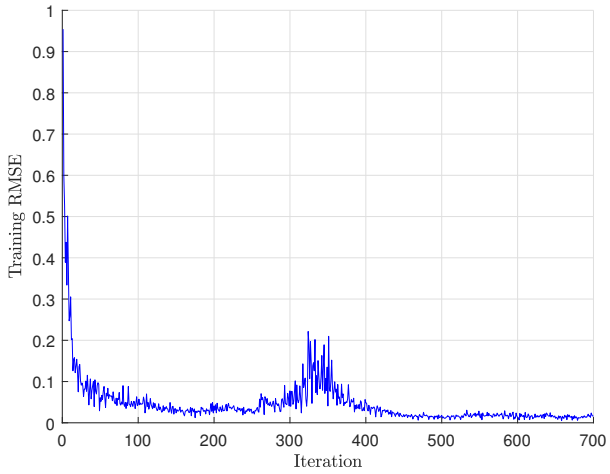
Due to all the above, the compromise solution between the required training time and the expected precision corresponded to a model composed by 100 LSTM neurons. It should be noticed that, because of the limitations in the development hardware setup, the training time was an important metric to be considered. As that specific size was not considered during the sweep analysis, a new model was trained and a validation was done. The figure 5.12 represents the time response of the identification system with respect to the actual one. It can be confirmed that the behaviour is good enough although the figure 5.12a shows some error in the airspeed, V_t, channel. However, the maneuver evaluated in 5.12b fitted perfectly. It is expected that if the training epochs are increased, the obtained error will be minimized.



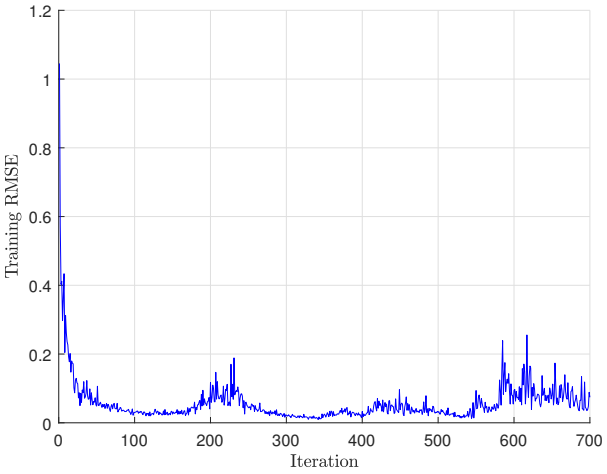
(a) Neurons = 20



(b) Neurons = 30



(c) Neurons = 50



(d) Neurons = 150

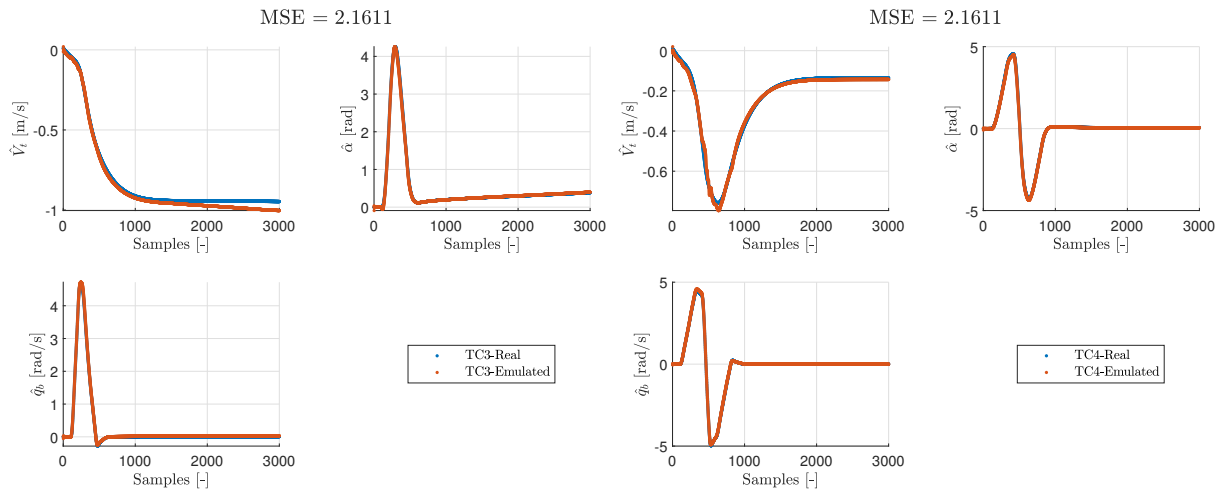
Figure 5.11: LSTM-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Training History

Thus, a LSTM neural network composed by 100 units was used for the following studies.

As it was mention above, the number of training epochs can alter the final accuracy of the model. Thus, a specific sweep analysis was done to study the effect of this hyperparameter on the resulting precision.

Table 5.6 list the performance metrics for the same neural network but trained using a different hyperparameter for the maximum training epochs. As it can be infered from the model name (TE#Epochs), the sweep covered 50, 100, 150 and 200 maximum training epochs. It is important to note that all considered models ended their training after reaching the maximum training epochs. On the other hand, the network was composed by 100 LSTM units. This setting was chosen in agreement with the previous results and the training time required by the available setup.

The mentioned metrics show that the higher the number of training epochs, the more precise the resulting model is (see figure 5.13). It should be emphasized that the improvement between 50 and 100 training epochs was quite higher that the following 50-

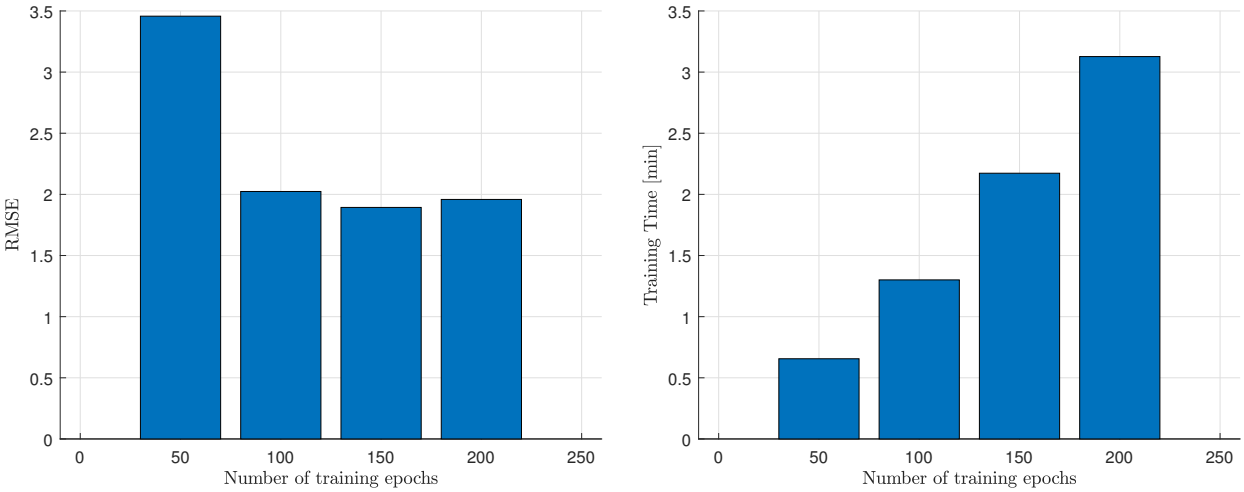


(a) TC-38 (b) TC-43

Figure 5.12: LSTM-based Longitudinal State Identification Model (Neurons = 100) - Validation

Models	#P	MSE	MAE	RMSE	Huber
LSTM_TE50_100	45503	11.9532	198.9112	3.4573	11.9532
LSTM_TE100_100	45503	4.0955	101.9565	2.0237	4.0955
LSTM_TE150_100	45503	3.5845	119.4308	1.8933	3.5845
LSTM_TE200_100	45503	3.8363	121.5895	1.9586	3.8363

Table 5.6: LSTM-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Performances Summary



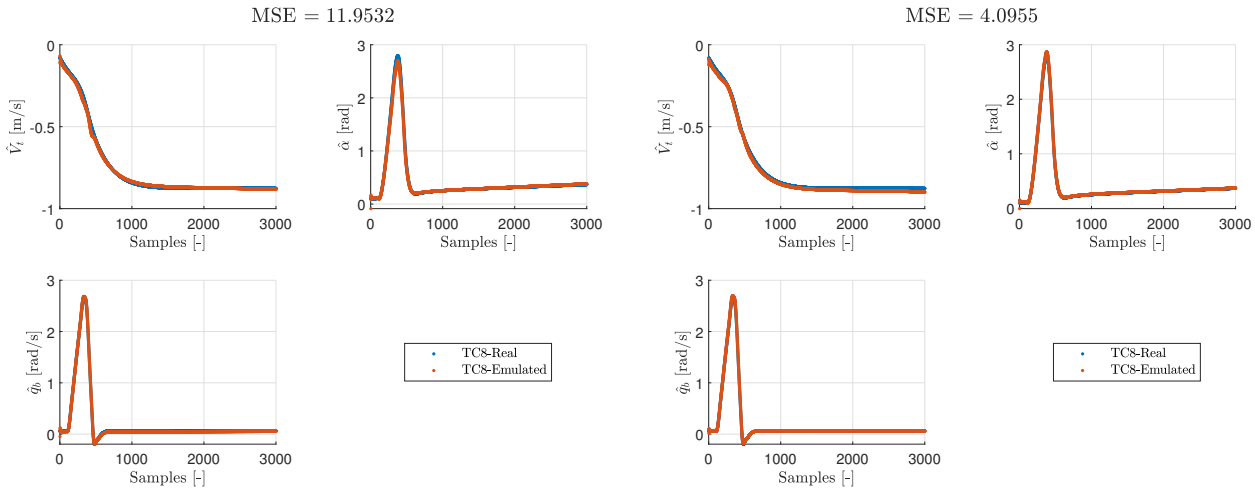
(a) (b)

Figure 5.13: LSTM-based Longitudinal State Identification Models - Training Epochs Sweep Overview

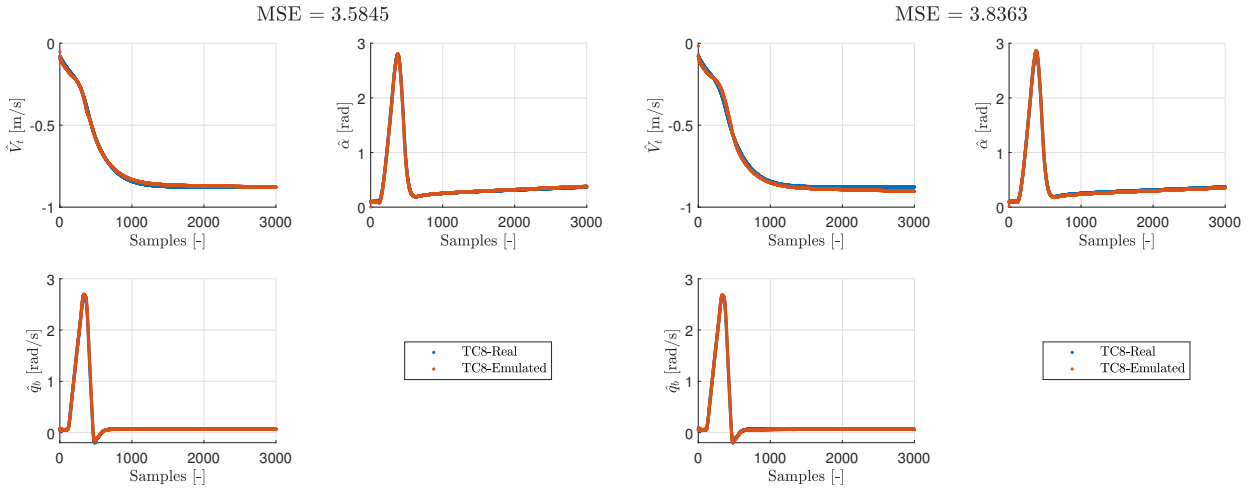
epochs increments. It can be concluded that for the configured neural network (100 neurons), to increase the number of epochs over 100 may be not efficient. Naturally, this optimal point shall be traded off in function of the model size and the admissible error. Nevertheless, it

is not possible to increment the epochs without limit because, in addition to the training time limitations related to the available hardware and model size, an exceed of training may cause overfitting on the resulting neural network.

Lastly, following the same criteria than in the previous studies, the time response of the model is represented in the figures 5.14 and 5.15. Any outstanding behaviour, with respect to the performance metrics information, is infered from the obtained sequences. The reader can observe that the higher was the training iterations, the better the model fitted except for the last case where during the final iterations, the error was increased. Additional training epochs would have allow to reduce again the estimation error and, likely, to reach a slightly more precise model.



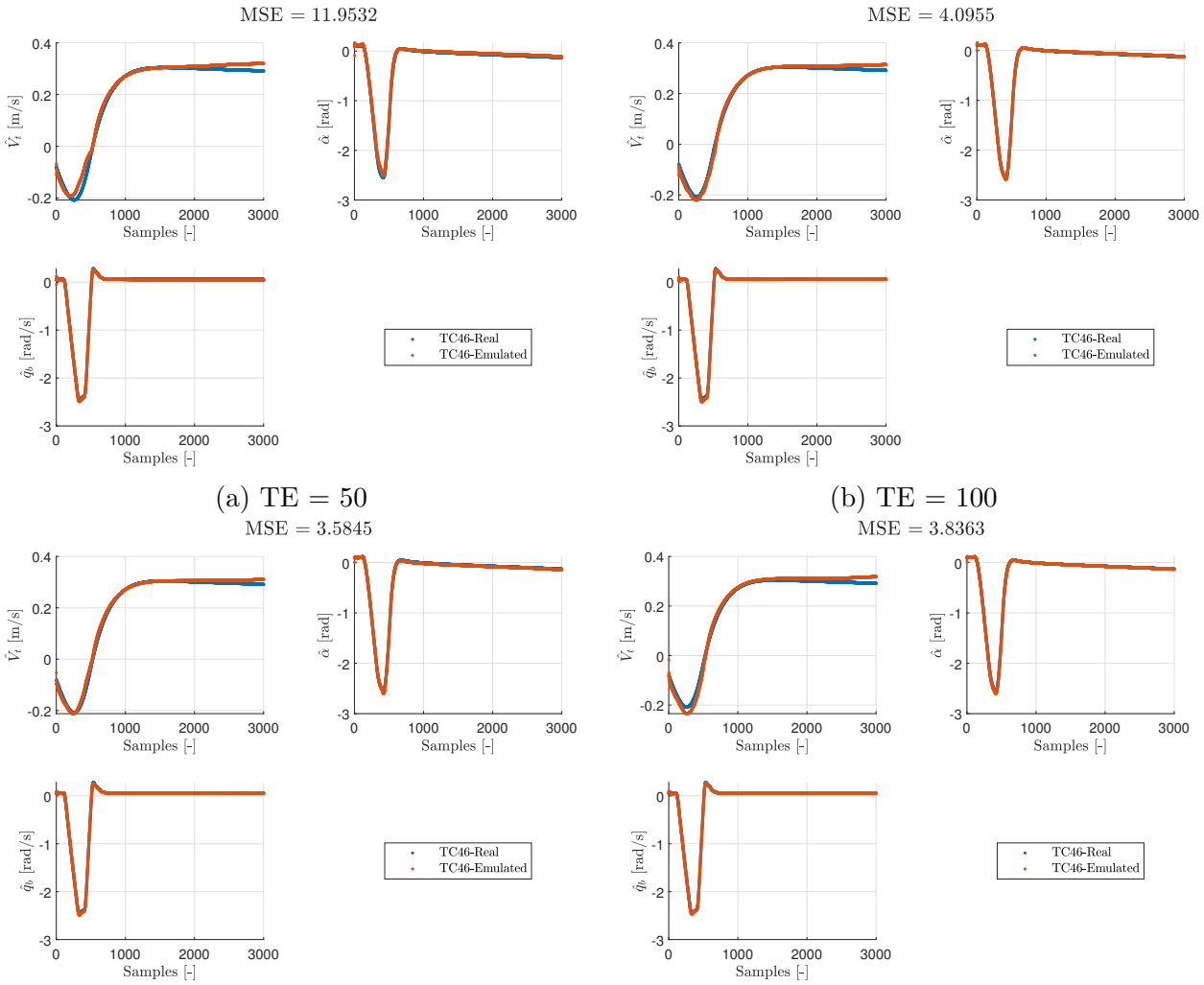
(a) TE = 50 (b) TE = 100



(c) TE = 150 (d) TE = 200

Figure 5.14: LSTM-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Validation (TC2)

It can be concluded that the optimal model reached from this study corresponded to a LSTM network composed by 100 neurons and a maximum number of training epochs of 100.



(a) TE = 50 (b) TE = 100

(c) TE = 150 (d) TE = 200

Figure 5.15: LSTM-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Validation (TC5)

5.3 Clockwork RNN-based Identification Model

At this point, although the previous neural network architectures proposed resulted in promising estimation models, a relative recent recurrent nerwork solution has been analyzed. In 2014, a simple but powerful modification of the standard RNN architecture (1 hidden layer), the Clockwork RNN (CW-RNN), was introduced in [Koutník et al., 2014]. As it was mentioned on the section 2.3, this kind of neural network is partitioned into separe modules based on different working clock rates. Then, each module processes the inputs according to its own working rate, making computations only at its defined clock rate. In other words, this network does not compute all its neuron states at all the time steps, in fact, the neurons groups are only evaluated at the corresponding time step with respect its own clock. Thus, this architecture are lighter than the standard recurrent network and it is expected that this model may fit the aircraft dynamics minimizing the number of neurons by adjusting the modules clock rates accordingly.

The first issue to implement this type of architecture was that MATLAB does not provide a built-in layer object for this network class. In fact, only a few Python repositories have been found for this network architecture on the state of the art. To move the analysis to the available Python environments was a option, however, it would have concerned the design pipeline (data generation, training and post processing pipelines) and caused a reduction of the flexibility and capacity to perform multiple analysis. On the other hand, to code a new layer object within the MATLAB framework enabled to use all the developed design pipeline capabilities. Moreover, a new MATLAB CW-RNN layer class meant a clear original outcome from this project. Therefore, a custom MATLAB CW-RNN class has been designed, implemented and used within the scope of this project. The reader can find more information in the appendix C.

It should be noted that the first functional version of the class did not define an specific back-propagation method. Thus, the MATLAB engine automatically determines the gradients using automatic differentiation. However, this process takes much more time to train the network than the native implemented networks, and, as it will be shown below, it has affected to the number and size of the studies done.

5.3.1 MATLAB CW-RNN Layer Validation

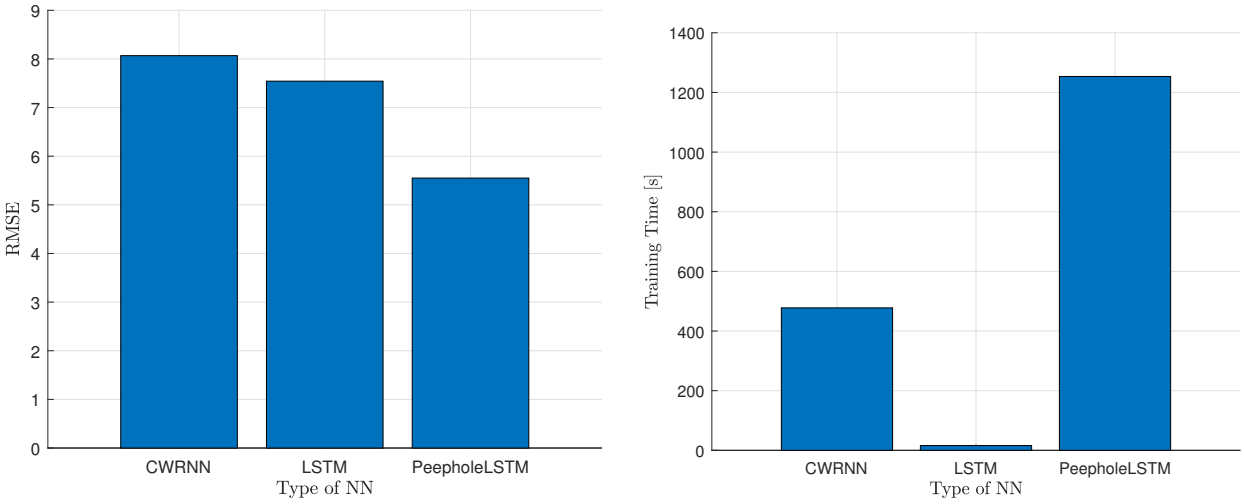
In the C is briefly explained how the MATLAB Deep Learning Toolbox allows the user to create custom layer object to be used as element of the designed DL network. To summarize, a empty layer object templated is provided in the Mathworks documentation (see [Mathworks, 2024a]) and, besides, a specific example about design a custom recurrent deep learning layer is shown. In particular, the reader can find in [Mathworks, 2024b] how to code a *Peephole LSTM Layer* within the MATLAB DL framework.

After implementing the maths behind the CW-RNN, explained in [Koutník et al., 2014], as MATLAB layer class, the first step of the validation process was to confirm that clock rates modules was activated correctly during the sequence evaluation and that, when the modules were disabled, the last evaluated state was held and used to compute the layer output. To do this, the Debugger mode of the MATLAB Integrated Development Environment (IDE) was utilized. Evaluating step by step different parts of the input sequences, it was confirmed that the forward function of the layer was computing the neurons states correctly.

Once the mathematical coding was validated, the CW-RNN class was integrated in the design pipeline in the same way that the previous studied networks, and a few initial trainings, based on the testing dataset, were performed. At this point, it was found out that the required time to train and evaluated the CW-RNN layer was significantly higher than the needed by the previous built-in layers (*i.e.*, FNN and LSTM classes) used in this project. Thus, in order to discard a possible bug on the custom layer code, several comparation test were done between the custom Peephole LSTM layer, provided by MATLAB as example, and the custom class designed within the scope of this work.

The figure 5.16b shows the training time which spent each kind of neural network during the training. All the architectures were specified with the same Input/Output definition and the same number of neurons (30), besides, they were trained using the same

dataset. All the training hyperparameter were common for all the trainings and, in particular, the maximum training epochs was set to 10 and all of them stopped theirs training after reaching the maximum allowed epochs. In other words, the only different between the networks was its implementation, from a mathematical and coding point of view. It is important to remark that CW-RNN and Peephole LSTM layers are coded using the available custom layer template, however, the standard LSTM is built-in class within the MATLAB framework. The reader may observe that, although the Peephole LSTM class is provided by the official documentation of Mathworks, it took significantly more time to complete the 10 training epochs than the CW-RNN class developed in this project. On the other hand, the built-in LSTM layer required much less time to train than the classes based-on on the custom layer template. It can be concluded, that the implementation of CW-RNN on the MATLAB framework was correct and the wide time required to train the custom classes with respect to the built-in classes, and the previous proposed architecture in this report, is only related with the MATLAB Deep Learning framework. Additionaly, it can be infered that, as it was exposed in [Koutník et al., 2014] and was expected according to the maths behind the CW-RNN definition, for a same size of network, the CW-RNN is slighter with respect other recurrent networks. Moreover, for this case, it can be also confirm that the resulting validation errors reached by both custom networks are of the same order of magnitude (see figure 5.16a).



(a) (b)

Figure 5.16: Comparation between CW-RNN, LSTM and Peephole-LSTM models

Therefore, it can be confirm that the custom CW-RNN layer developed within the MATLAB framework is valid. Nevertheless, for future works, it may be interesting to migrate this kind of analysis to a more extended Deep Learning frameworks in the industry, *e.g.*, PyTorch, Keras..., that may be more efficient. MATLAB framework is a quite new DL framework comparing to those environements and a optimal solution could be to used MATLAB/Simulink environment as data generation framework but to develop the training and postprocessing pipelines based-on those well-known Python frameworks.

Finally, despite of the hardware limitations and the required time to train custom layers, multiple analysis has been done in order to evaluate the benefits of using a CW-RNN as a dynamics identification model of the ADMIRE. These studies can be find in the next

section.

5.3.2 CW-RNN Identification Model Analysis

The estimation model proposed in this section is composed by a unique hidden layer of type CW-RNN and by an output fully-connected layer. Weights and biases of both layers were modified during the training processes.

The CW-RNN layer was configured to use a *tanh* activation function to compute the hidden states. No variation of the activation function was performed during this analysis. On the other hand, the fully-connected layer, as its name denotes, connect all neurons of the previous layer to its own neurons and multiplies the input values by a weight matrix and then adds a bias vector. The fully-connect layer is used as output layer.

Unlike the FNN and LSTM studies, here, only longitudinal dynamics identification models are shown. Thus, all models compiled in this section can be mathematically described as:

$$\begin{bmatrix} V_t[k] \\ \alpha[k] \\ q_b[k] \end{bmatrix} = f\left(\delta_{lc}, \delta_{rc}, \delta_{loe}, \delta_{roe}, \delta_{lie}, \delta_{rie}, M, h, V_t[k-1], \alpha[k-1], q_b[k-1]\right) \tag{5.9}$$

where $V_t[k]$, $\alpha[k]$ and $q_b[k]$ are the longitudinal state (airspeed, angle of attach and pitch rate) at the k-th time step, $f\left(...\right)$ is the resulting neural model and the meaning of the input arguments of the function are:

- δ_{lc}: Left canard position at the current time step [deg]
- δ_{rc}: Right canard position at the current time step [deg]
- δ_{loe}: Left outer elevon position at the current time step [deg]
- δ_{roe}: Right outer elevon position at the current time step [deg]
- δ_{lie}: Left inner elevon position at the current time step [deg]
- δ_{rie}: Right inner elevon position at the current time step [deg]
- M: Flight Mach number at the current time step [-]
- h: Flight altitude at the current time step [m]
- $V_t[k-1]$: Measured airspeed at the previous time step [m/s]
- $\alpha[k-1]$: Measured angle of attack at the previous time step [rad]
- $q_b[k-1]$: Measured pitch rate at the previous time step [rad/s]

As in the previous cases, it should be noted that if the variables are not followed by [...], it means that the value used is the corresponding to the current time step, $[k]$.

First, the optimal number of CW-RNN units required to identify the ADMIRE dynamics was analyzed. In table 5.7, the reader can find the validation metrics of six CW-RNN-based models, specifically, when 30, 45, 60, 75, 90 and 105 neurons were used for the

recurrent layer. It should be noticed that the previous number of neural units are refered to the complete size of the layer, not to the number of neurons per clock rate module. In particular, the six models were composed by three modules which were activated according to the following sample rates: 1, 2 and 4. In other words, the fastest module, assigned to the sample rate, $R = 1$, was computed for all the time steps, while the $R = 2$ and $R = 4$ modules were run each 2 and 4 times steps respectively. A specific study to analyzed how the selected rates affect to the performance of the models is enclosed later.

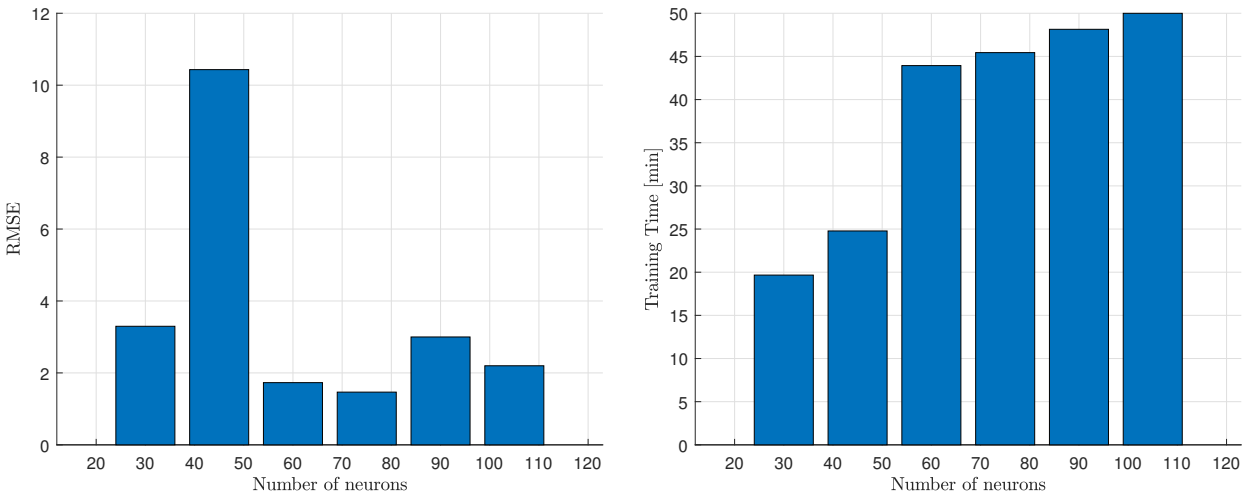
Models	**#P**	**MSE**	**MAE**	**RMSE**	**Huber**
CWRNN_TE30_30_R1_2_4	1383	10.8674	292.541	3.2966	10.8674
CWRNN_TE30_45_R1_2_4	2748	108.8386	841.6378	10.4326	108.8386
CWRNN_TE50_60_R1_2_4	4563	2.9914	160.2581	1.7296	2.9914
CWRNN_TE50_75_R1_2_4	6828	2.1489	109.5324	1.4659	2.1489
CWRNN_TE50_90_R1_2_4	9543	8.9961	248.4003	2.9994	8.9962
CWRNN_TE50_105_R1_2_4	12708	4.8342	217.5116	2.1987	4.8342

Table 5.7: CW-RNN-based Longitudinal State Identification Models (Neurons Sweep Analysis) - Performances Summary

The table shows that the most precise model, according to the overall metrics, corresponds to a network composed by 75 CW-RNN units, *i.e.*, 25 neuros per clock rate module. Moreover, a slightly smaller model, the 60-neurons network also provided promising results. These two models are analyzed with more detailed below in order to select a CW-RNN-based candidate. The rest of architectures did not achieve the expected performances. Neither the two smallest network nor the two largest cases were able to reach an appropiate accuracy. On one hand, the smallest models seem to require more paramaters to be able to identify the non-linear dynamics of the ADMIRE. On the other hand, the largest architectures were not able to converge on more precise models with the selected training settings. Likely, if the number of training epochs would have been increased, a better performance would have been reached. Nevertheless, as the training time is a constraint and the hardware setup was limitated, those networks were discarded.

A comparation between the RMSE error of each model and the required training time can be observed in the figure 5.17. The reader can infer the larger the model is, the more time it needs to be trained. Besides, a bigger network does not mean a more precise model due to the effect of the training settings in the final result. In particular, it should be noticed that the 45-neurons model resulted in the worst case. A deep analysis on the training metrics showed that the error evolved as expected during the most of the training but, it was increased just before reaching the maximum number of training epochs. That behaviour was not observed for the rest of proposals. Thus, it can be concluded that the resulting precision of the model is explained due to a training issue and it is not related to the number of neural units selected. In other words, if the model would have re-trained or the maximum iteratiations would have modified, the behaviour of this specific model would have been as expected. This activity was not performed at this phase of the study because it was considered that it would not have resulted in additional information about this kind of networks.

After understanding the causes of the outlier model, the following step is to con-

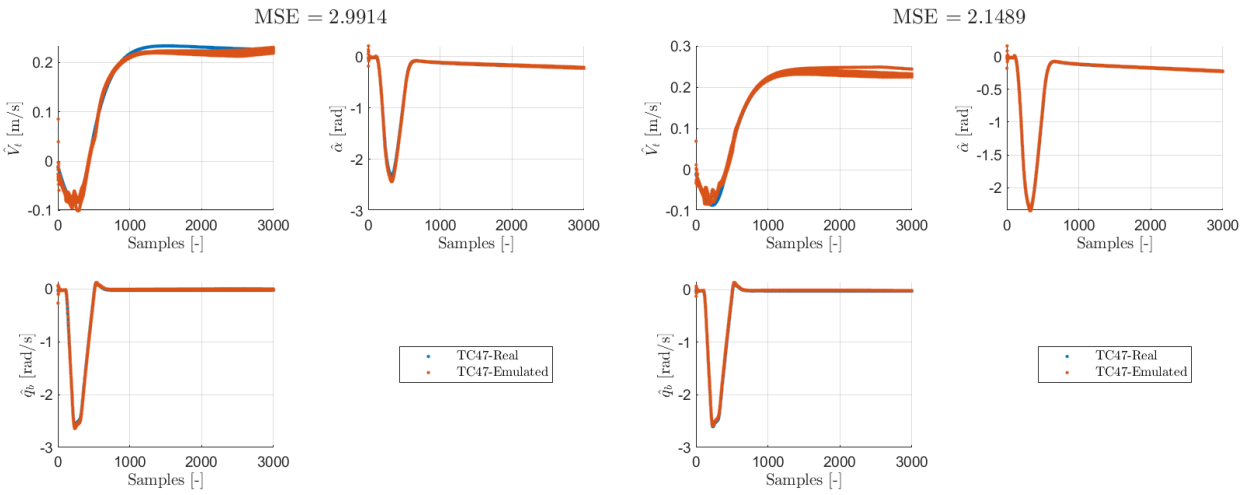


(a) (b)

Figure 5.17: CW-RNN-based Longitudinal State Identification Models - Number of Hidden Units Sweep Overview

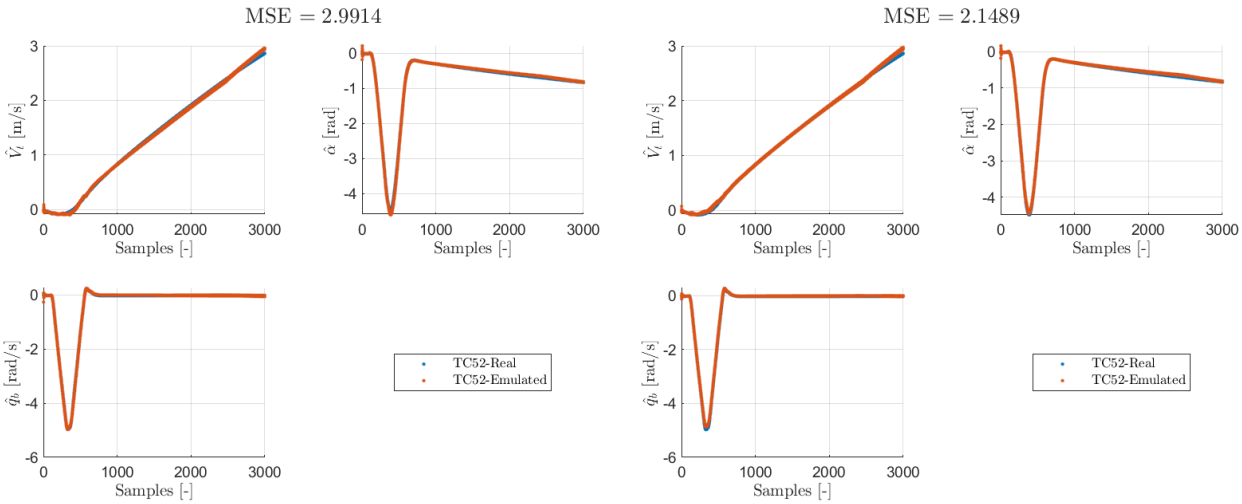
tinue with the standard analysis done up to the moment. Previously, two CW-RNN-based models were selected due to their promising error metrics. It is important to remark that the error metrics provide the designer overall information of the identification perfomance. Nevertheless, as the aircraft dynamics are usually characterized by short transient state and followed by long steady state behaviours after a specific maneuver, the overall error metrics may hide a bad performance during the transient. Thus, as it was done before, it is necessary to combine the metrics analysis with time-series representations. Figures 5.18 and 5.19 show the estimated response of the identification networks with respect to the actual one for the test cases TC5 and TC8 and for the considered models in order to select one as candidate. It can be observed that, even though the errors are not excessive, both neural networks failed to estimate the airspeed of the aircraft and provided a noise estimated variable that might affect later computations based on this estimated value, such as control laws feedback or gainset adjustment. On the other side, the α and q_b channels were estimated correctly and the resulting signals are smooth.

Additional design iteration shall be done in order to improve the airspeed channel results. Two standard strategies are to modify the training settings and, as the estimation function is based on Clockwork Recurrent Neural Networs, varying the selected clock rates. Additional advanced strategies may be to increase the number of clock rate modules or try to estimate another dynamic variable, such as the longitudinal aircraft speed, and then compute the airspeed from the estimated values instead of calculating the airspeed directly. Only the standard strategies were performed within the scope of this project.



(a) TE = 50; Neurons/Module = 20 (b) TE = 50; Neurons/Module = 25

Figure 5.18: CW-RNN-based Longitudinal State Estimator Models with rates = [1 2 4] (Neurons Sweep Analysis) - Validation (TC47)



(a) TE = 50; Neurons/Module = 20 (b) TE = 50; Neurons/Module = 25

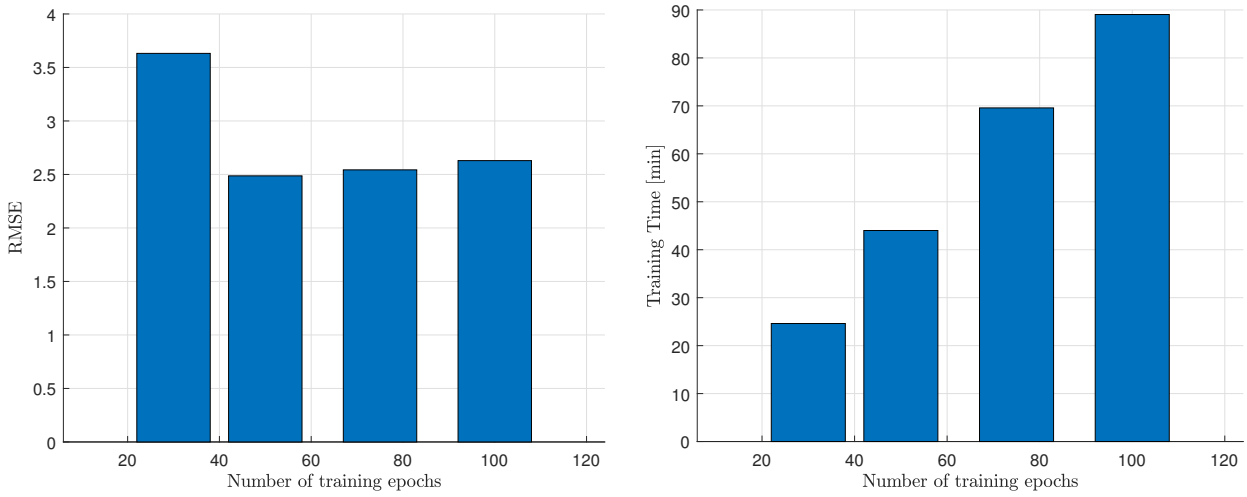
Figure 5.19: CW-RNN-based Longitudinal State Estimator Models with rates = [1 2 4] (Neurons Sweep Analysis) - Validation (TC52)

The effect of the number of training epochs was analyzed in detail by modifying this hyperparamter for a common CW-RNN model. Above, it has been watched that the most precise models were the network composed by 60 and 75 neural units. In particular, the 75-neurons model was the most precise according to all the considered metrics (MSE, MAE, RMSE and Huber). However, as the training time is a constraint for the current setup, it was interesting to consider the 60-neurons model as candidate for the following analyses. The table 5.8 lists the results of the selected model when the maximum training epochs was set to 30, 50, 75 and 100. It can be observed that the optimal number of training epochs was 50 and that, although the hyperparameter was increased up to 100, any improvement was infered according to the error metrics. On the other hand, the figure 5.20 displays the evolution of the RMSE and the training time with respect to the training epochs. As expected, there is

a linear relationship between the training time and the maximum training epochs. Thus, it can be concluded that the optimal training epochs for the chosen configuration was 50 and another strategy is required to improve the model performances.

Models	**#P**	**MSE**	**MAE**	**RMSE**	**Huber**
CWRNN_TE30_60_R1_2_4	4563	13.1883	180.4958	3.6316	13.1883
CWRNN_TE50_60_R1_2_4	4563	6.1848	137.0042	2.4869	6.1848
CWRNN_TE75_60_R1_2_4	4563	6.4664	156.7433	2.5429	6.4664
CWRNN_TE100_60_R1_2_4	4563	6.9135	153.1356	2.6294	6.9135

Table 5.8: CW-RNN-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Performances Summary



(a) (b)

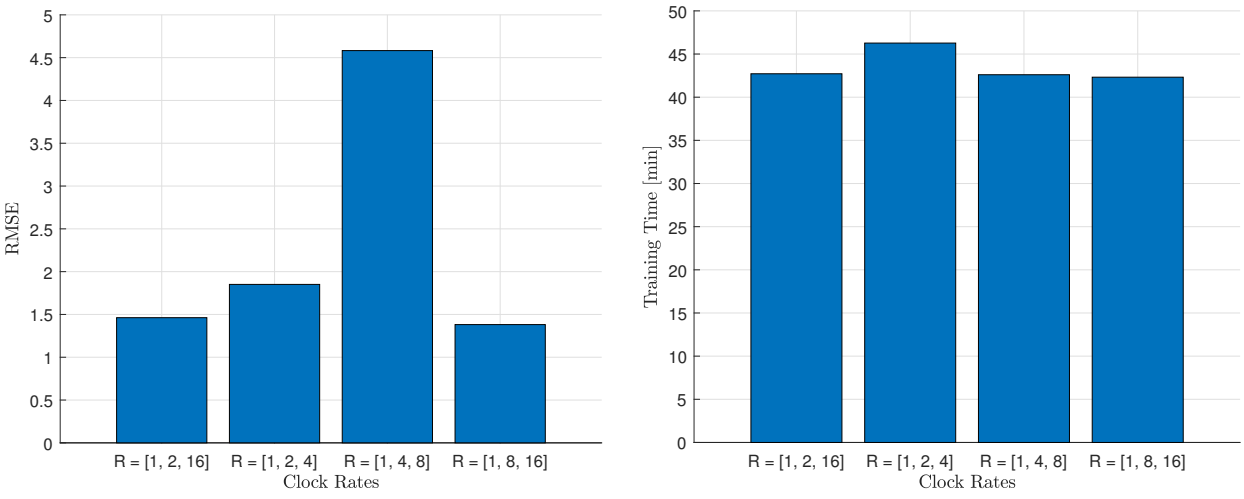
Figure 5.20: CW-RNN-based Longitudinal State Identification Models (Training Epochs Sweep Analysis) - Training Epochs Sweep Overview

Another critical parameter that influence in the network performance is the clock rates selected for each module. In [Koutník et al., 2014], the authors of this kind of recurrent architecture used rates based on the powers of 2. The reader might notice that, up to this point, this study has been using that approximation, in fact, clock rates based on the 2^0, 2^1 and 2^2 has been utilized for all the previous models.

In the table 5.9 and the figure 5.21 the reader can observe the precision of the resulting models when the selected clock rates were changed. It should be noted that for this sweep analysis, a common structure of 3 clock rate modules and 20 neurons per group was used. In other words, the proposed networks were characterized by the same number of parameters and the only difference was the activation rates of each neurons group. Besides, the same training setting was used for all the cases. It can be infered that the model with the clock rates equal to [1 8 16] fitted better the ADMIRE longitudinal dynamics at the specific flight condition used to build the testing dataset. On the other hand, as expected from the mathematical model, the neural network with higher clock rates required more time to finish the training due to its modules were activated more frequently during the process.

Models	#P	MSE	MAE	RMSE	Huber
CWRNN_TE50_60_R1_2_4	4563	3.426	155.6194	1.851	3.426
CWRNN_TE50_60_R1_4_8	4563	20.9993	392.5306	4.5825	20.9993
CWRNN_TE50_60_R1_2_16	4563	2.1381	115.5673	1.4622	2.1381
CWRNN_TE50_60_R1_8_16	4563	1.9085	103.4302	1.3815	1.9085

Table 5.9: CW-RNN-based Longitudinal State Identification Models (Clock Rates Sweep Analysis) - Performances Summary



(a) (b)

Figure 5.21: CW-RNN-based Longitudinal State Identification Models (Clock Rates Sweep Analysis) - Clock Rates Sweep Overview

Additional clock rates studies were performed where, based on the previous result, a fourth clock rate group was added to the networks. In particular, the most promising selection, R = [1 8 16], was modified as R = [1 2 8 16]. Thus, the resulting network was 20 neurons larger than the previous cases because of the inclusion of this new clock rate module. Any advantage was observed, in fact, the gathered metrics were significantly worse than the baseline configuration. It is related with the increasing of the network size but keeping the same training setting and, likely, this new intermediate module did not track any specific dynamic mode of the ADMIRE at that flight condition. Thus, only a noise source was added to the model.

Chapter 6

Conclusions and future works

A design and validation process to build neural identification models of the dynamics of a fighter aircraft has been presented in this report. In particular, the longitudinal dynamics of the augmented (*i.e.*, stable) system known as ADMIRE has been used. Multiple issues related to the nonlinearities of the aircraft has been analyzed and design solutions has been proposed. Below, the reader can find the main conclusions obtained within the scope of this project.

First, a data synthetic generation pipeline is necessary to design robust models by compensating the lack of real data in specific areas of the working envelope. Real aircraft data, even commercial information, is expensive and difficult to obtain. However, the use of synthetic data based on precise engineering aircraft simulator, together with the available real data, allows the designer to train precise models without requiring a huge amount of flight hours.

MATLAB/Simulink environment is widely extended in the aeronautical industry to model the aircraft behaviour and to design flight control laws. However, its Deep Learning toolbox is not so mature as the Python frameworks and it is not very used on the AI industry or academia. In fact, during this project has been confirmed that the Mathworks tools are very useful and efficient to model and simulate the aircraft dynamics but it presented some limitations and performance issues when non-standard neural networks were needed. Thefore, for future works, it would be interesting to consider to divive the development pipeline into two pieces, the synthetic data generator based on MATLAB/Simulink and the neural network definition and training pipeline based on some of the widely extended Python frameworks as PyTorch or TensorFlow. The post processing pipeline can fulfill the expected needs in both environments but, as the data shall be prepared for the training, to use Python also for this task seems the optimal choice.

A parametric and parallelizable setup is required in order to generate huge amount of data easily. In this project, a fighter aircraft simulator developed in Simulink was available since the beginning. However, this simulator was very limitated with respect to the way that the pilot maneuvers were input and, even, the initial flight condition. Indeed, only hardcoded maneuvers before the Simulink model compilation were possible. This flexibility lacking was an issue to generate the minimum data required to train a robust neural network. Therefore,

an upgrade of the simulation was done in order to reach a parametric and parallelizable environment.

The selection of the input features in the designed network is a critical step in order to reach the expected precision. It was concluded that any of the models considered within the scope of this project was able to estimate the figther dynamic states without information of, at least, the values of the previous time step. This Input/Output architecture is known as NARX. The reason behind this behaviour can be extracted from the well-known non-linear equations which describe the aicraft motion. It is a set of partial differential equations whose solution is dependent on the initial state of the system. Therefore, NARX I/O architecture is mandatory to model the dynamics using neural network.

Standard feedforward neural network and two types of recurrent networks were evaluated. FNN/NARX-based models were able to capture the complex nonlinear relationships of the longitudinal dynamics of the ADMIRE. The main advantage of this kind of network is that promising "small" models were found. Around 1200 learneable parameters were required to achieve the expected estimation performance. It is a reduced number compared to the 45000 parameters which composed the selected LSTM model. For its part, the proposed CW-RNN model required around 4500 weights (and biases) to fulfill the performance requirement.

Clockwork Recurrent Networks (CRNs) and Long Short-Term Memory (LSTM) networks are both powerful recurrent neural network (RNN) architectures designed to handle sequential data. However, they employ different mechanisms to address the vanishing gradient problem and capture long-term dependencies. As it could see, CRNs introduce a hierarchical structure of layers, each operating at a different frequency. This allows for efficient processing of long sequences by focusing computational resources on relevant time steps, while LSTMs use a gating mechanism to control the flow of information, allowing them to selectively remember or forget information over time. The choice between CRNs and LSTMs depends on the specific requirements of the study. In this case, as the selected maneuvers resulted in very long sequences of data (30 seconds samples at 100 Hz), CW-RNNs might be more efficient due to its hierarchical structure and fit the limites computational resources of an aircraft equipment. Nevertheless, as the fighter dynamics is highly non-linear, i.e., the dependencies in the data are complex and require precise control over information flow, LSTMs might be a better choice. This issue could be mainly observed on the airspeed channel, which is the result of a square of time-dependant non-linear terms. Depending on the maneuver aggressiveness, CW-RNNs were not able to estime the state variable correctly. However, LSTMs, as they are a more complex and robust models, are able to reach a more precise estimator for that channel by improving training settings and hardware setup. Nevertheless, it has been demonstrated that CW-RNN architectures allow to obtained equivalent level of precision as widely extended LSTM nets but with slighter models. Furthermore, optimal estimation models can be obtained by adjusting the clock rates of the neural modules to the characteristic frequencies of the dynamic systems. In particular, the response of an aircraft can be described by a set of well-known dynamics modes which are characterized by specific frequencies. Then, it can be concluded that the CW-RNN may be an optimal solution to deploy recurrent models on systems with limited computational resources

RNN may improve slightly the precision level of the identification models but the im-

plicit increase of complexity shall be also considered. Aerospace industry is a high regulated environment and the current quality and certification standard are based on determinims (each computed value shall be as expected, black boxes are not allowed). Moreover, although the industry is working with the certification authority on this topic, the current regulation does not permit the use of GPUs on critical system of the aircraft as it is the flight control system and the use of multi-core computers is also quite recent. Thefore, this is a huge constraint for the kind of deep learning models that can be deployed on a real critical system.

Lastly, a collection of the identified possible future works is listed below:

- *Scale the identification model*: The current study has reduced the dataset due to setup constrains to aircraft data flying on an specifc flight condition. Next steps would be to increase the size of the dataset, adding new maneuvers and performing those over all the flying envelope, to build new estimation models. Likely, an unique neural network to cover the complete envelope is not the optimal solution. A division of the envelope in multiple areas may be performed and, then, to compute a new neural network for each zone and to design a switching method to change between models when the aircraft flight conditions vary. Both paths should be explored and a trade-off considering technical and hardware integration issues, in addition to the required workload, should be done.

- *Adjust clock rates of the CW-RNN to the characteristic frequencies of the system*: It was confirmed that modified the selected clock rates impacted on the final precision of the estimation models. This modification was done following a power of 2 but better improvements are expected if the clock rates modules are adjusted to the real characteristic frequencies of the plant to be identified. It should be noticed that these frequencies can vary with the operating point. Thus, maybe the use of multiple neural identification models dependant on the operating point is mandatory for these kind of network.

- *Assess advantages and disadvantages of building a model of the complete dynamics (not apply the decoupled approach)*: Another possible work would be to increase the dataset to include lateral maneuvers and to create a full dynamics model. The increase of complexity should be compared to the possible improvement of the estimation precision if the network is able to learn the physical bounds between the longitudinal and the lateral-directional dynamics.

- *Integrate the neural identification model in final application*: The current identification models by theirself do not provide any value. Multiple solutions of intelligent control system and failure, detection and isolation systems has been proposed. Once the identification model is mature enough, the following step should be to integrate it as component on final applications as the mentioned before.

Bibliography

T. Hagan Martin. *Neural network design*. s. n.], S. l., . 2016. ISBN 9780971732117.

Jan Koutník, Klaus Greff, Faustino Gomez, and Jürgen Schmidhuber. A clockwork rnn, 2014. URL https://arxiv.org/abs/1402.3511.

Lars Forssell, Ulrik 1963-Nilsson, and Totalförsvarets forskningsinstitut. *ADMIRE The Aero-Data Model In a Research Environment Version 4.0, Model Description*. FOI - Totalförsvarets forskningsinstitut, Linköping, 2005. ISBN 1650-1942 ;. URL https://ucm.on.worldcat.org/oclc/185339819. TA:.

OpenAI. Main page of openai company, 2024. URL https://openai.com/. Last accessed 21 August 2024.

Google DeepMind. Alphafold protein structure database - home page, 2024. URL https://alphafold.ebi.ac.uk/. Last accessed 21 August 2024.

Yonghui Wu. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

Burak Gülmez. Stock price prediction with optimized deep lstm network with artificial rabbits optimization algorithm. *Expert Systems with Applications*, 227:120346, 2023. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2023.120346. URL https://www.sciencedirect.com/science/article/pii/S0957417423008485.

Qi-Qiao He, Cuiyu Wu, and Yain-Whar Si. Lstm with particle swam optimization for sales forecasting. *Electronic Commerce Research and Applications*, 51:101118, 2022. ISSN 1567-4223. doi: https://doi.org/10.1016/j.elerap.2022.101118. URL https://www.sciencedirect.com/science/article/pii/S1567422322000023.

Masooma Ali Raza Suleman and S. Shridevi. Short-term weather forecasting using spatial feature attention based lstm model. *IEEE Access*, 10:82456–82468, 2022. doi: 10.1109/ACCESS.2022.3196381.

Fatemeh Charoosaei, Amin Faraji, Sayed Alireza Sadrossadat, Ali Mirvakili, Weicong Na, Feng Feng, and Qi-Jun Zhang. High-speed nonlinear circuit macromodeling using hybrid-module clockwork recurrent neural network. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(2):767–780, 2024. doi: 10.1109/TCSI.2023.3337115.

Wan Liu, Yuanli Gu, Ying Ding, Wenqi Lu, Xiaoping Rui, and Lu Tao. A spatial and

temporal combination model for traffic flow: A case study of beijing expressway. In *2020 IEEE 5th International Conference on Intelligent Transportation Engineering (ICITE)*, pages 76–81, 2020. doi: 10.1109/ICITE50838.2020.9231430.

Yuxuan Shi, Yanyu Wang, and Haoran Zheng. Wind speed prediction for offshore sites using a clockwork recurrent network. *Energies*, 15(3), 2022. ISSN 1996-1073. doi: 10.3390/en15030751. URL https://www.mdpi.com/1996-1073/15/3/751.

Airbus. Airbus - artificial intelligence: Capitalising on the value of data, 2024. URL https://www.airbus.com/en/innovation/digital-transformation/artificial-intelligence. Last accessed 21 August 2024.

Boeing. Boeing - mq-28, 2024. URL https://www.boeing.com/defense/mq28#overview. Last accessed 21 August 2024.

Seyyed Ali Emami, Paolo Castaldi, and Afshin Banazadeh. Neural network-based flight control systems: Present and future. *Annual Reviews in Control*, 53:97–137, 2022. ISSN 1367-5788. doi: 10.1016/j.arcontrol.2022.04.006. URL http://dx.doi.org/10.1016/j.arcontrol.2022.04.006.

Madjid HAOUANI, Maarouf SAAD, and Ouassima AKHRIF. Flight control system design for commercial aircraft using neural networks. *IFAC Proceedings Volumes*, 35(1):1–6, 2002. ISSN 1474-6670. doi: https://doi.org/10.3182/20020721-6-ES-1901.01226. URL https://www.sciencedirect.com/science/article/pii/S1474667015396476. 15th IFAC World Congress.

Kalmanje Krishnakumar, Greg Limes, and Karen Gundy-Burlet. An adaptive critic approach to reference model adaptation. *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 08 2003. doi: 10.2514/6.2003-5790.

Sampath Yerramalla, Edgar Fuller, Martin Mladenovski, and Bojan Cukic. Lyapunov analysis of neural network stability in an adaptive flight control system. 2704:77–91, 06 2003. doi: 10.1007/3-540-45032-7_6.

Declan Bates and Martin Hagström. *Nonlinear analysis and synthesis techniques for aircraft control*. Springer Verlag, Berlin, 2007. ISBN 9783540737193; 3540737197; 9783540737186; 3540737189. URL https://ucm.on.worldcat.org/oclc/191453897http://site.ebrary.com/id/10206250.

P. Sutton George. *Rocket propulsion elements*. Wiley, Hoboken, N.J., ©. 2010. ISBN 9780470080245.

B.L. Stevens, F.L. Lewis, and E.N. Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. Wiley, 2015. ISBN 9781118870983. URL https://books.google.es/books?id=lvhcCgAAQBAJ.

V. Cook Michael. *Flight Dynamics Principles : a Linear Systems Approach to Aircraft Stability and Control*. Elsevier Science, Oxford, . 2012. ISBN 9780080982762.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of*

Control, Signals and Systems, 2(4):303–314, 1989. doi: 10.1007/BF02551274. URL https://ucm.on.worldcat.org/oclc/5654345561http://dx.doi.org/10.1007/BF02551274.

Akif Altun, Mehmet Onder Efe, Decision 2019 6th International Conference on Control, and 2019 April 23 2019 April 26 Information Technologies (CoDIT). *Aircraft Control with Neural Networks*. 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT). 2019. doi: 10.1109/CoDIT.2019.8820660. URL https://ucm.on.worldcat.org/oclc/8219394149http://dx.doi.org/10.1109/CoDIT.2019.8820660.

Mathworks. Define custom deep learning layers, 2024a. URL https://www.mathworks.com/help/deeplearning/ug/define-custom-deep-learning-layers.html. Last accessed 21 August 2024.

Mathworks. Define custom recurrent deep learning layer, 2024b. URL https://www.mathworks.com/help/deeplearning/ug/define-custom-recurrent-deep-learning-layer.html. Last accessed 21 August 2024.

Mathworks. List of deep learning layers, 2024c. URL https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html. Last accessed 21 August 2024.

Appendix A

Datasets

A.1 Testing Dataset

Test Case ID	Trim Mach [-]	Trim Altitude [ft]	Control Input	Maneuver Description
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (1s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_10	0.65	10000	F_{es}	Pull 10 (1s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (1s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_0_20	0.65	10000	F_{es}	Pull 20 (1s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (1s), Push -10 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (1s), Push -10 (1s), Hold (1s), Trim (2s), Hold (3s)

(Continued on next page)

Table A.1: Testing Dataset Maneuvers

Test Case ID	Trim Mach [-]	Trim Altitude [ft]	Control Input	Maneuver Description
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (0.5s), Push -10 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Pull 10 (2s), Hold (0.5s), Push -10 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Pull 10 (1s), Hold (0.5s), Push -10 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Pull 10 (1s), Hold (2s), Push -10 (1s), Hold (2s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (1s), Push -20 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (1s), Push -20 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (0.5s), Push -20 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Pull 20 (2s), Hold (0.5s), Push -20 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Pull 20 (1s), Hold (0.5s), Push -20 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Pull 20 (1s), Hold (2s), Push -20 (1s), Hold (2s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_5	0.65	10000	F_{es}	Pull 5 (2s), Hold (1s), Push -10 (2s), Hold (2s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (2s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (2s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (2s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (2s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (2s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (2s), Hold (1s), Trim (1s), Hold (3s)

(Continued on next page)

Table A.1: Testing Dataset Maneuvers

Test Case ID	Trim Mach [-]	Trim Altitude [ft]	Control Input	Maneuver Description
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (1s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_0	0.65	10000	F_{es}	Push 10 (1s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (2s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (2s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (2s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (2s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (2s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (2s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (1s), Hold (0.5s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_0	0.65	10000	F_{es}	Push 20 (1s), Hold (1s), Trim (1s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Push 10 (2s), Hold (1s), Push -10 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Push 10 (2s), Hold (1s), Push -10 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Push 10 (2s), Hold (0.5s), Push -10 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Push 10 (2s), Hold (0.5s), Push -10 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Push 10 (1s), Hold (0.5s), Push -10 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg10_10	0.65	10000	F_{es}	Push 10 (1s), Hold (2s), Push -10 (1s), Hold (2s), Trim (2s), Hold (3s)

(Continued on next page)

Table A.1: Testing Dataset Maneuvers

Test Case ID	Trim Mach [-]	Trim Altitude [ft]	Control Input	Maneuver Description
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Push 20 (2s), Hold (1s), Push -20 (2s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Push 20 (2s), Hold (1s), Push -20 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Push 20 (2s), Hold (0.5s), Push -20 (1s), Hold (1s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Push 20 (2s), Hold (0.5s), Push -20 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Push 20 (1s), Hold (0.5s), Push -20 (1s), Hold (0.5s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg20_20	0.65	10000	F_{es}	Push 20 (1s), Hold (2s), Push -20 (1s), Hold (2s), Trim (2s), Hold (3s)
M0p65_H10000_Fes_deg_Custom_Lims_Neg5_10	0.65	10000	F_{es}	Push 5 (2s), Hold (1s), Push -10 (2s), Hold (2s), Trim (2s), Hold (3s)

Table A.1: Testing Dataset Maneuvers

Appendix B

Training Pipeline - Methods

B.1 Load NN Architecture Method

A common structure was defined for the load functions. Each method build a specific neural network architecture based on the provided input arguments. Two input arguments are mandatory and additional settings controlled via optional input arguments. In particular, the mandatory arguments correspond to the list of input and output names that characterize the network. The size of those lists are used to configure the input and output layer sizes and, downwash in the code, to extract the specific data from the data tables. On the other hand, the optional arguments allow the designer to modify the number of hidden layers and their sizes, the activation functions, the sequence padding proccess and the training configuration. If not optional arguments are specified, default values are used.

Respect to the output arguments, all the coded methods return two output variables. The first one is the network object, that can be a MATLAB *network* class or a MATLAB *dlnetwork* object depending on the kind of neural network selected. The second argument corresponds to a MATLAB data structure which contains useful information (*e.g.*, training settings...) that will be used downwash in the pipeline.

Below, the reader can find the load method implemented to build CW-RNN network objects. It can be checked that, particular settings of this kind of network (*i.e.*, the clock rates) are set via optional arguments.

```
1 function [net,userData] = load_nn_architecture(inputNames,
     outputNames,options)
2 %% Description
3 % Function to define desired network architecture, including
     selected
4 % inputs and outputs from the signals database.
5 % Specific database files filter string may be defined for
     posterior
6 % processing.
7 % Specific datasets preparation function may be defined for
     posterior
```

```
% processing.
% Base Architecture = CW-RNN
% - Inputs = Actuation Commands[k-1] (drc,dlc,droe,drie,dloe,dlie), Angular
% Rate[k-1], Mach[k-1]
% - Output = Angular Rate[k]
% - Hidden Layers Architecture:
%% Arguments
arguments (Input)
    inputNames cell = {...
     'CONTROL_INPUTS.drc',...
     'CONTROL_INPUTS.dlc',...
     'CONTROL_INPUTS.droe',...
     'CONTROL_INPUTS.drie',...
     'CONTROL_INPUTS.dloe',...
     'CONTROL_INPUTS.dlie',...
     'FLIGHT_CONDITION.Altitude',...
     'FLIGHT_CONDITION.Mach',...
     'ADMIRE_STATE.qb'
     };
    outputNames cell = {'ADMIRE_STATE.qb'}
    options.Rates {mustBePositive} = [1 2 4]
    options.HiddenSizes {mustBeVector,mustBeInteger} = [10]
    options.HiddenTransferFcns string = "tanh"
    options.OutputTransferFcn string = "purelin"
    options.TrainFcn char = 'adam'
    options.TrainEpochs {mustBeInteger,mustBePositive} = 30
    options.MiniBatchSize {mustBeInteger,mustBePositive} = 10
    options.SeqLength {mustBeMember(options.SeqLength,["longest","shortest"])} = "longest"
    options.SeqPaddingDirection {mustBeMember(options.SeqPaddingDirection,["right","left"])} = "right"
    options.DataFilesFilters cell = {''}
    options.PrepareDataFcn char = 'prepare_data'
    options.NormalizeData logical = false;
    options.ModelNamePrefix string = "CWRNN"
end
arguments (Output)
     net
     userData
end
%% Configure NN-CWRNN
% Get number of hidden layers
nHiddenSizes = length(options.HiddenSizes);
% Process NN name
if nHiddenSizes == 1
     nNeuronUnits = options.HiddenSizes*length(options.Rates);
```

```
52 else
53     nNeuronUnits = sum(options.HiddenSizes);
54 end
55 nnName = options.ModelNamePrefix + "_TE" + mat2str(options.
       TrainEpochs);
56 if options.NormalizeData
57     nnName = nnName + "_Norm";
58 end
59 nnName = nnName + "_I" + string(length(inputNames))...
60     + "O" + string(length(outputNames))...
61     + "_" + mat2str(nNeuronUnits)...
62     + "_R" + replace(mat2str(options.Rates),{'[',']',' '},{'','',
       '_'});
63
64 % Get NN base architecture
65 layers = [sequenceInputLayer(length(inputNames),Name="In")];
66 cwLayers = [];
67 for nLayer = 1:nHiddenSizes
68     cwLayers = [cwLayers; cwRNNLayer(options.Rates,...
69         options.HiddenSizes(nLayer))];
70 end
71 layers = [layers; cwLayers; fullyConnectedLayer(length(
       outputNames),Name="Out")];
72 net = dlnetwork(layers);
73 % Set Auxiliary Information
74 userData.InputVariables = inputNames;
75 userData.OutputVariables = outputNames;
76 userData.TrainFunction = options.TrainFcn;
77 userData.TrainEpochs = options.TrainEpochs;
78 userData.MiniBatchSize = options.MiniBatchSize;
79 userData.SeqLength = options.SeqLength;
80 userData.SeqPaddingDirection = options.SeqPaddingDirection;
81 userData.DataFilesFilters = options.DataFilesFilters;
82 userData.PrepareSetsFunction = options.PrepareDataFcn;
83 userData.NormalizeData = options.NormalizeData;
84 userData.Name = nnName;
85 end
```

Listing B.1: *load_nn_architecture* function for CW-RNN

B.2 Define Study Method

Once the generic load architecture functions were implemented, a function template was created to define the neural network studies performed during this project. In other words, a cell array structure was specified in order to input the different combination of neural network that will be evaluated in the training pipeline. This definition files are the input of the training pipeline. It can consist in one or multiple network architectures to be trained.

Each study case, *i.e.*, each network configuration to be trained, is stored in a cell element. Within this cell element, two sub-elements are defined. The first one denotes the kind of neural network, *i.e.*, the *load_nn_architecture()* function that shall be used. The second one correspond to a nested cell array which contains all the input arguments of the load NN method (*i.e.*, the list of input and output variable names, the layer sizes...).

Below, the reader can find an example of study definition where a baseline neural network configuration is defined and then, the cell element is cloned but modifying the number of hidden units. It definition function was used to perform network sizes sweep analysis.

```
function [nnArchitectures] = define_nn_architectures()
nnArchitectures = {};
nNN = 0;
% First study
nNN = nNN + 1;
nnArchitectures{nNN} = {...
    ... NN type
    'CW-RNN',{...
    ... NN Configuration (Load fcn arguments)
    {'CONTROL_INPUTS.drc',...
    'CONTROL_INPUTS.dlc',...
    'CONTROL_INPUTS.droe',...
    'CONTROL_INPUTS.drie',...
    'CONTROL_INPUTS.dloe',...
    'CONTROL_INPUTS.dlie',...
    'CONTROL_INPUTS.tss',...
    'FLIGHT_CONDITION.Altitude',...
    'FLIGHT_CONDITION.Mach'...
    'ADMIRE_STATE.Vt',...
    'ADMIRE_STATE.alpha',...
    'ADMIRE_STATE.qb'},...
    {'ADMIRE_STATE.Vt',...
    'ADMIRE_STATE.alpha',...
    'ADMIRE_STATE.qb'},...
    'HiddenSizes',[10],...
    'TrainEpochs',30,...
    'NormalizeData',true,...
    'SeqLength',"longest",'SeqPaddingDirection',"right"...
    'ModelNamePrefix',"LongState_NARX_CWRNN_SeqLongRight"}...
    };
% Additional Studies - Include Hidden Sizes sweep
hiddenSizesSweep = [15 20 30];
for n = 1:numel(nnArchitectures{1}{2})
    if isstring(nnArchitectures{1}{2}{n}) || ischar(
   nnArchitectures{1}{2}{n})
        if contains(nnArchitectures{1}{2}{n},'HiddenSizes')
            index = n + 1;
```

```
            break
        end
    end
end
for hiddenSizes = hiddenSizesSweep
    nNN = nNN + 1;
    nnArchitectures{nNN} = nnArchitectures{1};
    nnArchitectures{nNN}{2}{index} = [hiddenSizes];
end
end
```

Listing B.2: Definition Study function for a number of neuron sweep analysis of a CW-RNN network

Appendix C

CW-RNN MATLAB Class

Since some years ago, MATLAB environment has implemented its own Deep Learning framework compatible with several of the most extended Python frameworks. In particular, the *Deep Learning Toolbox* provides functions, apps, and Simulink blocks for designing, implementing, and simulating deep neural networks. The toolbox provides a framework to create and use many types of networks, such as convolutional neural networks (CNNs) and transformers. Furthermore, the framework allows to the designer to import pretrained models from *PyTorch*, *TensorFlow* and *ONNX* and export the MATLAB-based models to *TensorFlow* and *ONNX* formats.

The toolbox provides an extended list of built-in layers, *e.g.*, LSTM, Hyperbolic Tangent (tanh) Activation, 2-D Convolutional... layers (more information can be found on [Mathworks, 2024c]), to facilitate the design flow. In addition, all of this layer objects are optimized to work within the MATLAB environment. Nevertheless, if there does not exist the built-in layer that the designer needs for his task, then he can define her own custom layer. The Mathworks documentation has a dedicated page to explain how to create a custom layer compatible with the MATLAB Deep Learning environment (see [Mathworks, 2024a])

In order to implement a new layer within the MATLAB environment, the designer shall define at least the following methods for the class:

- *Layer constructor*: Special function that creates an instance of the class. Input arguments can be declared to modify the initial properties of the class object.
- *Predict*: Method use by the layer during a forward pass at precition time. If a specific *Forward* function is not defined, the layer object also uses the *Predict* method to compute the output state during the training process.

Naturally, the class properties shall also be declared in agreement with the requirements of the previous methods. Properties shall be divided between *State Properties* and *Learnable Parameters*. Weights and biases shall be defined as learnable properties, while, internal states of the layer units. *e.g.* the hidden state of a recurrent network, shall be specified as state parameters.

The reader may notice that the *Background* method, used to train the learnable

parameters, is not mandatory. It is due to the MATLAB framework includes *Reverse Mode Automatic Differentiation* which is a set of techniques for evaluating derivatives (gradients) numerically based on the computational graph coded in the *Forward* function. Therefore, the framework is able to train the layer without needing to compute derivatives manually for optimization. Nevertheless, when a network with a custom layer is trainned without a backward function, the software traces each object used by the custom layer forward function to determine the computation graph used for automatic differentiation. This tracing process can take some time and can end up recomputing the same trace, then, the custom layer may become slow. Specify a custom background pass function speeds up the resulting network and allows the user to define custom behaviour during the optimization.

The version of the CW-RNN class used in the scope of this project is based on the Automatic Differentiation algortihm. It may penalize the benefits of this kind of network during the backgroud pass because, although a neural units is not active at a specific sample time, the algortihm is trying to compute its derivative due to not specific backgroud algorthim was implemented to facilitate the class definition. Nevertheless, the resulting derivative is zero because the forward function do not modify the state of disabled neurons.

The custom class was validated by debugging the computation flow manually and by comparing the developed layer with respect to the custom RNN layer example proposed on the Mathworks documentation, *i.e.*, a Peephole-LSTM layer. These layers were used to build two longitudinal dynamics emulators of the ADMIRE aircraft, both with the same number of neurons and trained/tested using the same datasets. In addition to confirm that both networks approximated correctly the dynamic responses, it was validated that the CW-RNN was faster than the Peephole-LSTM network as it was expected due to the maths behind both king of layers. In other words, it was verified that not all the neurons of the CW-RNN are evaluated and updated at all the time steps, making them faster than a LSTM-kind network. More information about this analysis can be found within the respective section of the report (see 5.3).

Lastly, the reader can find the custom class code below. It should be noticed that only two input arguments are mandatory: the working rates of each neural module and the size of each module.

```
classdef cwRNNLayer < nnet.layer.Layer & nnet.layer.Formattable
    % & nnet.layer.Acceleratable % (Optional)
    %% Properties
    properties
        % Layer properties.

        Rates {mustBePositive}
        NumHiddenUnits {mustBePositive}
        OutputMode
        CWMask
    end

    properties (Learnable)
        % Layer learnable parameters.
```

```
15
16          InputWeights
17          RecurrentWeights
18          Bias
19      end
20
21      properties (State)
22          % Layer state parameters.
23
24          HiddenState
25      end
26      %% Public Methods
27      methods (Access = public)
28          function layer = cwRNNLayer(rates,numHiddenUnits,args)
29              % Create a cwRNNLayer (CW-RNN Layer)
30
31              % Parse input arguments.
32              arguments (Input)
33                  rates {mustBePositive} = [1 2 4]
34                  numHiddenUnits {mustBeInteger,mustBePositive} =
    [15]
35                  args.Name = "";
36                  args.OutputMode = "sequence";
37              end
38              % Assign properties
39              layer.Rates = rates;
40              layer.NumHiddenUnits = numHiddenUnits;
41              layer.Name = args.Name;
42              layer.OutputMode = args.OutputMode;
43
44              % Set layer description.
45              layer.Description = "CW-RNN with " + length(rates) +
    ...
46                  "rate modules and " + mat2str(numHiddenUnits) +
    ...
47                  " hidden units each thread";
48          end
49
50          function layer = initialize(layer,layout)
51              % Initialize layer learnable and state parameters.
52              %
53              % Inputs:
54              %         layer  - Layer to initialize
55              %         layout - Data layout, specified as a
    networkDataLayout
56              %                  object
57              %
```

```
58            % Outputs:
59            %         layer - Initialized layer
60            %
61            %  - For layers with multiple inputs, replace layout
    with
62            %    layout1,...,layoutN, where N is the number of
    inputs.
63
64            % Get number of neurons per layer (hidden units *
    rate modules)
65            nModules = length(layer.Rates);
66            nNumHiddenUnits = length(layer.NumHiddenUnits);
67            if nNumHiddenUnits == 1
68                layer.NumHiddenUnits = ...
69                    repmat(layer.NumHiddenUnits,[1 nModules]);
70            elseif nNumHiddenUnits ~= nModules
71                error("Number of hidden units specified shall be
    scalar "...
72                    +"or a vector with same dimensions as rate
    modules.")
73            end
74            % Get dimensions
75            numHiddenUnits = sum(layer.NumHiddenUnits);
76            numChannels = layout.Size(finddim(layout,"C"));
77            % Initialize input weights.
78            biasInitializationFcn = @(sz) rand(sz,'single') *
    0.0001;
79            if isempty(layer.InputWeights)
80                sz = [numHiddenUnits numChannels];
81                layer.InputWeights = layer.initializeOrthogonal(
    sz);
82            end
83            % Initialize recurrent weights.
84            if isempty(layer.RecurrentWeights)
85                sz = [numHiddenUnits numHiddenUnits];
86                layer.RecurrentWeights = layer.
    initializeOrthogonal(sz);
87            end
88            % Initialize bias.
89            if isempty(layer.Bias)
90                layer.Bias = biasInitializationFcn([
    numHiddenUnits,1]);
91            end
92            % Initialize hidden state
93            layer.HiddenState = zeros(numHiddenUnits,1);
94
95            % Initialize Clockwork mask (constant)
```

```
96              cwMask = zeros(numHiddenUnits,numHiddenUnits);
97              rowSIdx = 1;
98              for nModule = 1:nModules %Faster module always runs
99                  rowEIdx = nModule*layer.NumHiddenUnits(nModule);
100                 columnSIdx = rowSIdx;
101                 cwMask(rowSIdx:rowEIdx,columnSIdx:end) = ...
102                     ones(rowEIdx-rowSIdx+1,numHiddenUnits-
    columnSIdx+1);
103                 rowSIdx = rowEIdx+1;
104             end
105             layer.CWMask = cwMask;
106         end
107
108         function [Y,hiddenState] = predict(layer,X)
109             %PREDICT CW-RNN predict function
110             %   [Y,hiddenState] = predict(layer,X) forward
111             %   propagates the data X through the layer and
    returns the
112             %   layer output Y and the updated hidden state.
113             %   X is a dlarray with format "CBT" and Y is a
    dlarray with
114             %   format "CB" or "CBT", depending on the layer
    OutputMode
115             %   property.
116
117             % Initialize sequence output
118             numHiddenUnits = sum(layer.NumHiddenUnits);
119             numBatches = size(X,finddim(X,"B"));
120             numTimeSteps = size(X,finddim(X,"T"));
121
122             if layer.OutputMode == "sequence"
123                 Y = zeros(numHiddenUnits,numBatches,numTimeSteps
    ,"like",X);
124                 Y = dlarray(Y,"CBT");
125             end
126
127
128             % Get x
129             x = stripdims(X);
130
131             %
132             Wx = pagemtimes(layer.InputWeights,x) + layer.Bias;
133
134             % Initial states
135             if size(layer.HiddenState,2) == 1 && numBatches > 1
136                 hiddenState = repmat(layer.HiddenState,[1
    numBatches]);
```

```
137             else
138                 hiddenState = layer.HiddenState;
139             end
140             recurrentWeights = layer.RecurrentWeights.*layer.
    CWMask;
141 
142             % Loop over time steps
143             for t = 1:numTimeSteps
144                 % Get active neurons
145                 activeRowIndices = [];
146                 sIndex = 1;
147                 for nRate = 1:length(layer.Rates)
148                     eIndex = sIndex + (layer.NumHiddenUnits(nRate
    )-1);
149                     idx = mod(t,layer.Rates(nRate)) == 0;
150                     if idx
151                         activeRowIndices = [activeRowIndices,...
152                             sIndex:eIndex];
153                     end
154                     sIndex = eIndex + 1;
155                 end
156 
157                 % Calculate R*h_{t-1} (Partial State)
158                 Rht = recurrentWeights(activeRowIndices,
    activeRowIndices)*hiddenState(activeRowIndices,:);
159 
160                 % Calculate W*x_{t} (Partial State)
161                 % Wxt = Wx(activeRowIndices,:,t);
162 
163                 % Calculate Partial State (only desired rows are
    updated)
164                 hiddenState(activeRowIndices,:) = tanh(Rht+Wx(
    activeRowIndices,:,t)+layer.Bias(activeRowIndices));
165 
166                 % Update sequence output.
167                 if layer.OutputMode == "sequence"
168                     Y(:,:,t) = hiddenState;
169                 end
170             end
171 
172             % Last time step output.
173             if layer.OutputMode == "last"
174                 Y = dlarray(hiddenState,"CBT");
175             end
176         end
177 
178         function layer = resetState(layer)
```

```
179             %RESETSTATE Reset layer state
180             % layer = resetState(layer) resets the state
    properties of the
181             % layer.
182
183             numHiddenUnits = sum(layer.NumHiddenUnits);
184             layer.HiddenState = zeros(numHiddenUnits,1);
185         end
186     end
187     %% Private Methods
188     methods (Access = private)
189         function parameter = initializeOrthogonal(layer,sz)
190             %INITIALIZEORTOGONAL
191
192             arguments (Input)
193                 layer
194                 sz {mustBeFinite,mustBePositive}
195             end
196             Z = randn(sz,'single');
197             [Q,R] = qr(Z,0);
198
199             D = diag(R);
200             Q = Q * diag(D ./ abs(D));
201
202             parameter = dlarray(Q);
203         end
204     end
205 end
```

Listing C.1: CW-RNN *dlnetwork* class