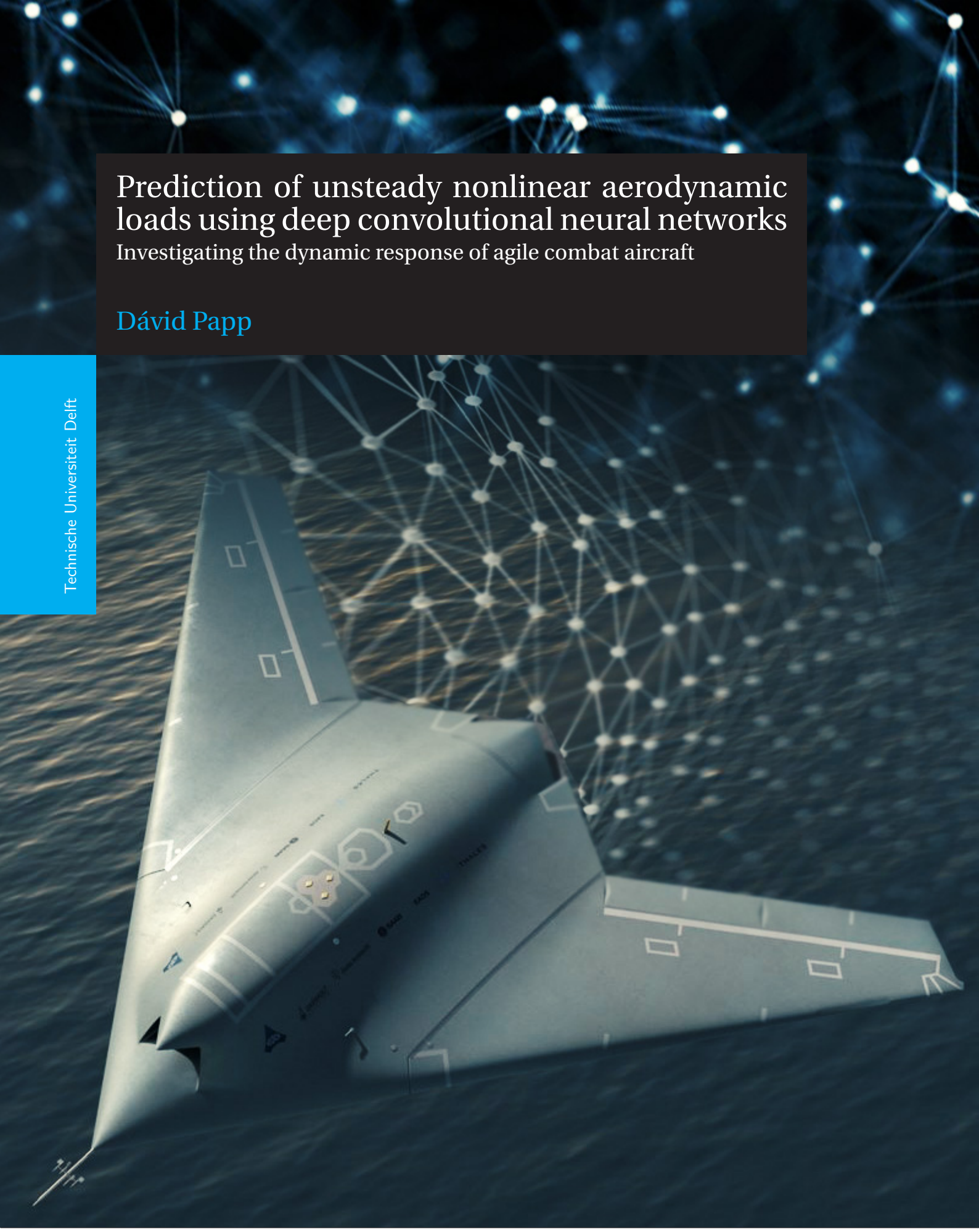
Prediction of unsteady nonlinear aerodynamic loads using deep convolutional neural networks

Investigating the dynamic response of agile combat aircraft

Dávid Papp

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

Images of the cover page can be found at [1–3].

PREDICTION OF UNSTEADY NONLINEAR AERODYNAMIC LOADS USING DEEP CONVOLUTIONAL NEURAL NETWORKS

INVESTIGATING THE DYNAMIC RESPONSE OF AGILE COMBAT AIRCRAFT

by

Dávid Papp

in partial fulfillment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,
to be defended publicly on Tuesday, July 31, 2018 at 13:00

Supervisors:	Dr. ir. M. Voskuijl,	TU Delft
	Dr. ir. M. van Rooij,	NLR
Thesis committee:	Prof. dr. L. L. M. Veldhuis,	TU Delft
	Dr. M. Voskuijl,	TU Delft
	Dr. ir. M. van Rooij,	NLR
	Dr. ir. D. M. Pool,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

nlr Dedicated to innovation in aerospace

ACKNOWLEDGEMENTS

I would like to first express my gratitude to my supervisors, Dr. ir. Michel van Rooij and Dr. ir. Mark Voskuijl, without whom this thesis would not have been possible. I could not succeed without their utmost trust, support and guidance. Their high standards and excellence kept me on my toes during the past year and made me pursue a successful outcome. I hope that I did not let them down and that my work lived up to their expectations. I am also thankful for my former colleagues at the Netherlands Aerospace Centre. They helped me cope with many of my problems, plus they made NLR the workplace where I gladly cycled to each morning.

I am unspeakably grateful for my family. You always stood behind me regardless the circumstances. You all are the absolute reason of my achievements. Your love and support helped me overcome all the difficulties in life. I can only wish that one day I can return you all the good I received.

At last but not least, I would also like to thank my friends from all around the world for giving me great company outside of work. I could not have made it this far without your pleasant memories.

SUMMARY

New generation combat aircraft are expected to operate over extended flight envelopes, including flight at high flow angles and rapid maneuvers. Conditions beyond traditional limits are giving rise to nonlinear phenomena, such as flow separation, large scale energetic vortices, fluctuations etc. These phenomena have significant impact on aircraft performance and if not resolved accurately design uncertainties are increased risking lack of performance or even costly redesigns. Thus, accurate modelling of unsteady nonlinear aerodynamics is essential for modern and future combat aircraft.

Unfortunately, conventional modelling tools either lack the required fidelity or they are too expensive. Traditional, highly-efficient approaches are not suitable for modelling nonlinear flow phenomena. Concurrently, high fidelity Computational Fluid Dynamics (CFD) simulations are computationally demanding and therefore impractical in many cases. To enhance aircraft design, it is desirable to obtain models joining the best of these two worlds. A common approach is to distill high fidelity methods into Reduced-Order Models (ROMs) that can accurately approximate unsteady aerodynamics at orders of magnitudes lower costs than CFD. Relevant literature offers many different ROM techniques for varying purposes. Nonetheless, constructing such models is still challenging and currently there is no generally agreed method.

In the current thesis a ROM technique that may be applicable to wider ranges of problems and simpler to construct is sought. The objective is to obtain a model that can promote aircraft control design, performance assessment and structural analysis throughout dynamic maneuvers over complete flight envelopes. The thesis proposes a novel approach utilizing modern, deep convolutional neural networks (CNNs). The devised model consists of three main components. First it incorporates a geometry description constituted by coordinates of an aircraft CFD surface grid. Second, a primary encoding-decoding CNN predicts pressure distribution at the grid points of the geometry. The final and third part of the model is an auxiliary encoding CNN deriving integral aerodynamic loads corresponding to the pressure field predictions of the primary network. The model evaluates and produces instantaneous values. Given a maneuver, it proceeds in timesteps. The predictions of the separate instances are computed directly without the need of subiterations (as it would be the case for CFD simulations).

As a proof of concept, the model is applied to symmetric motions in the vertical plane at fixed Mach number and altitude. The subject of the investigations is the MULDICON configuration of the 251 th Science and Technology Organization work-group of NATO. To fully exploit the advantages of reduced-order modelling, flow characteristics are inferred from a single excitation following an efficient system identification technique using Schroeder sweeps as input signals. The performance of the model is assessed by numerous test cases performed in CFD. First, steady conditions of varying incidence angles are investigated. Second, harmonic pitch and plunge oscillations around different angles of attack at different amplitudes and frequencies are considered. Third, additional test cases of a linear pitch up-down – and a climbing maneuver are studied.

Considering computational efficiency, the results show robust model performance. GPU-accelerated CNN calculations are conducted roughly 5000 times faster than CFD simulations. The primary network can accurately resolve the pressure distributions over large portions of the geometry. Lower surface predictions are very accurate. However, among certain conditions discrepancies are observable on the upper surface towards the wingtips. Still, the secondary network can predict corresponding aerodynamic forces accurately. In contrast, its moment predictions are sensitive to errors in pressure distributions. Consequently, moment predictions can largely deviate from reference data, especially when nonlinear phenomena are prominent. However, in many cases errors are attributed to insufficient regressor space coverage, i.e. certain input combinations are explored poorly by the Schroeder sweeps. Reconsidering system identification practices might mitigate those issues. Nevertheless, the thesis proves the applicability of deep CNNs to the problems at hand. Additionally, the results encourage further investigations.

CONTENTS

NOMENCLATURE

Roman symbols

$\hat{\mathbf{t}}$	Tensor of predicted outputs	–
$\mathbf{C}$	Convolutional matrix	–
$\mathbf{G}$	Geometry tensor	–
$\mathbf{h}$	Output tensor of network (layer)	–
$\mathbf{I}$	Input tensor	–
$\mathbf{K}$	Convolution kernel tensor	–
$\mathbf{S}$	Output tensor of convolution	–
$\mathbf{T}$	Transformation matrix	–
$\mathbf{t}$	Tensor of true outputs	–
A	Axial force in aerodynamic body frame, or Signal amplitude	N –
A	Normal force in aerodynamic body frame	N
a	Activation output, or Speed of sound	– $\mathrm{m\,s^{-1}}$
b	Bias	–
C	Cost	–
C_A	Axial force coefficient	–
C_M	Pitching moment coefficient	–
C_N	Normal force coefficient	–
C_p	Pressure coefficient	–
d	Tensor depth	–
e	Unit vector	–
f	Activation	–
G	Sum of gradients	–
h	Tensor height, or Altitude	– m
L	Training loss	–
l	Length	m
M	Pitching moment, or Mach number	$\mathrm{N\,m^{-1}}$ –

p	Pressure, or	Pa
	Signal power	–
q	Dynamic pressure	Pa
S	Surface area	m^2
s	Output of convolution, or	–
	Stride of convolution, or	–
	Half span, or	m
	Non-dimensional time	–
T	Temperature	K
t	Time	sec
U	Axial velocity	$m\,s^{-1}$
u	Signal input	–
V	Velocity	$m\,s^{-1}$
v	Changes	–
W	Normal velocity	$m\,s^{-1}$
w	Weights	–
X	Axial force in flight dynamic body frame	N
Z	Normal force in flight dynamic body frame	N
z	Weighted input of neuron	–

Greek symbols

α	Value of saturation in (C)ELU activations, or	–
	Decay of EMA, or	–
	Angle of attack	deg
β	Inertia of parameter update	–
δ	Partial derivative of cost	–
η	Learning rate, or	
	Ordinate of computational space –	–
γ	Flight path angle	deg
μ	Air dynamic viscosity	$kg\,s\,m^{-1}$
Φ	Phase angle	rad
ρ	Air density	$kg\,m^{-3}$
σ	Sigmoid function	–
τ	Time variable	sec
θ	Model parameter, or	–
	Pitch angle	deg
ξ	Abscissa of computational space	–

Superscripts

dec	Decoded features in network output
dist	Disturbed features of controller and regressor blocks
enc	Encoded features of input tensors
ext	Extended features of controller and regressor blocks
C	Integral load coefficients in output tensors
p	Pressure coefficients in output tensors

Subscripts

∞	Free stream properties
ref	Reference value
p	Tensors of the primary network
s	Tensors of the secondary network
T	Transpose convolution matrix

1

INTRODUCTION

1.1. PROJECT MOTIVATION

Modelling unsteady nonlinear aerodynamics is of great importance for modern and future combat aircraft as they are expected to operate beyond conventional flight envelopes. Extended operating conditions include maneuvers at high angles of attack and rapid (multi-axis) motions giving rise to nonlinear aerodynamic phenomena, such as energetic and large scale vortices, vortex bursting, separated flow, etc. These phenomena have a significant impact on aircraft performance [4–6]. In order to enhance aircraft designs and make them more capable for their roles, the use of modelling methods capturing unsteady nonlinear airloads is key. Otherwise, design uncertainties are increased risking lack of performance, which may even lead to costly redesigns [4]. On the contrary, accurate models can potentially promote control design, improve performance – and structural analysis and mitigate project risks [5]. However, devising such models in the nonlinear domain are still challenging because traditional highly-efficient approaches are not suited for nonlinear flow characteristics [4–7].

Concurrently, advances in Computational Fluid Dynamics (CFD) and computer sciences made the simulation of nonlinear physical behaviors commonplace. Important flow phenomena with millions of degrees of freedom (DOFs) can be captured accurately. That led to straightforward unsteady nonlinear aerodynamic models. These models provide accurate time-marching solutions of aerodynamic equations. On the other hand CFD simulations are computationally expensive and therefore impractical in many cases, e.g. multidisciplinary analysis and design, shape optimization, control and stability analysis, etc. This leaves a gap between available – and practically achievable model fidelity [5, 8].

The motivation of the current study is to develop a model that is able to retain the accuracy of CFD simulations but at orders of magnitudes lower costs making its performance comparable to the conventional tools. Being successful, such model can readily promote the design of modern aircraft.

1.2. THE RELEVANCE OF REDUCED-ORDER MODELLING AND THE ROLE OF NEURAL NETWORKS

Due to a mismatch in available and practically available levels of fidelity in unsteady aerodynamic modelling, reduced-order models (ROMs) are gaining importance. A ROM distills the full-order problem into a numerically less demanding model, that can provide accurate descriptions of system behaviour (within a specified range of interest) at magnitudes lower computational costs than the full-order model. They encapsulate the complete systems via appropriate projections of limited numbers of degrees of freedom (DOF). In contrary, full order models can consist of tens of millions of DOFs [5, 7, 9].

Two general categories of ROM methods are the parametric – and non-parametric techniques. Parametric methods presume certain system models in which the coefficients of the model are to be found. Non-parametric methods on the other hand pursue mappings among system inputs and outputs while they disregard the actual physical backgrounds, resulting in a black-box system modelling approach. When unsteady nonlinear aerodynamic problems are considered, many of the arising phenomena are analytically

unresolved, which hinders the construction and identification of appropriate descriptions. In those cases non-parametric methods have a definite advantage, especially intelligent methods. Intelligent algorithms learn system characteristics in a hand-off process from test or simulation data in order to predict responses for arbitrary inputs (within certain boundaries) [6, 9].

One of the most popular intelligent methods in use are neural networks (NNs or neural nets). Their applicability to aerodynamic problems has been an active field of research for decades now. Neural networks are able to capture temporal and spatial dependencies of complex, nonlinear systems, which makes them appealing for modeling complex aerodynamic problems [6, 10]. There are two prominent fields of applications of NNs considering the current context. The first one deals with predicting dynamic response of nonlinear systems in terms of integral aerodynamic properties, such as lift, drag, pitching moment, etc. The second important field of research studies the applicability of neural nets to flow field predictions in terms of distributed flow variables, for instance velocity or pressure.

For reduced-order modelling purposes the methodology of the first field is more popular at the moment. Those consider solely the dynamic responses of aerodynamic systems, that are derived from a limited number of state – and/or control variables. The actual properties of the flow around the aircraft, or at least the spatially distributed ones, are most of the time neglected. These models concentrate on the temporal dependencies among inputs and outputs. The applied algorithms are either some simple feedforward neural network or some type of Recurrent Neural Networks (RNN) that specialize in capturing time-history effects [11–14]. RNNs can store their internal states corresponding to previous predictions, therefore they are able to incorporate memory effects in their systems. In aerospace applications due to the relatively small set of observed variables they usually employ more compact architectures. Despite that, these systems are able to capture nonlinear unsteady aerodynamic characteristics. For instance, Lyu *et al.* [15] used an extreme learning machine to capture multi-axis aircraft dynamics at high angles of attack. A drawback of these systems is that they provide limited information about the actual flow. In return they can only utilize a limited amount of information and they must resort to a small number of inputs. That first raises the problem of selecting an adequate set of input variables [6], second it offers less flexibility for modeling capabilities.

Another approach is to predict not (just) the integral aerodynamic loads, but also some distributed properties of the flowfields around wings for instance. Aerospace applications of these systems mostly consider static conditions and their focus is on the spatial dependencies in the data. Usually these methods are bound to apply larger, more complex architectures to cope with spatially related dense domains. Numerous applications prove that neural networks are indeed capable of solving flows and therefore applicable to various aerodynamic problems [7, 16, 17]. A type of algorithm that are shown to be powerful solving similar problems are Convolutional Neural Networks (CNNs or ConvNets). CNNs are specialized in extracting patterns and features from topological data and are widely used in image processing. A great potential of these networks is that they can possibly substitute CFD solvers among certain scenarios and solve the flow at significantly lower costs. Concurrently, if the flow is accurately resolved, neural networks are also capable of deriving integral properties. As Schreck *et al.* [10] showed in their early studies, using only pressure distributions around wing sections was more then sufficient to accurately predict lift, drag, pitching moment, friction drag, etc. Ever since, neural networks have improved considerably and nowadays it is possible to predict surface pressure distributions on complete wings at once [7]. It is hypothesized that modern CNNs allow for extending the formulation of Schreck *et al.* [10] to complete wings and predicting pressure distributions in each time step of arbitrary dynamic maneuvers while also providing resultant integral aerodynamic loads. Having the method of Schreck *et al.* [10] to achieve high accuracy, the idea is definitely worth investigating as such an approach may provide better flexibility and improved prediction and analysis capabilities especially if detailed flow predictions are considered.

1.3. RESEARCH OBJECTIVES

Motivated by the previous observations, the current thesis investigates the applicability of modern convolutional neural networks to unsteady nonlinear aerodynamic problems. The goal is twofold. First, it is intended to provide accurate predictions of integral aerodynamic load coefficients, that may be used for flight dynamics simulations. Second, it is intended to derive the loads by first predicting surface pressure distributions on aircraft wing, that may enhance capabilities of the complete ROM. At the same time, it is also desired that predictions are generated at orders of magnitudes lower costs compared to full-order CFD methods, so advantages of reduced-order modelling can be exploited to their full extent. Hence, a CNN based surrogate

model is developed. The devised model consists of a primary network responsible for predicting pressure distributions, and of a secondary network that calculates integral loads based on the predictions of the primary network. These two subsystems are employed together. Executing the two networks yields solutions for predefined conditions directly, without the need of subiterations.

1.4. REPORT OUTLINE

After the introduction, corresponding literature of neural networks is presented in Section 2.1. First the basic topologies of neural networks are reviewed, followed by a more elaborate discussion about network optimization, explaining how these algorithms are fit to learn complex, non-convex mathematical problems. Then in Section 2.2 the idea of convolutional neural networks presented along with a few of their specific features that are important to understand the operating principles of the proposed model.

To address challenges of unaffordable training data sets and accurate high resolution labeling of dense spatial data, state of the art solutions are revised in Chapter 3. The chapter introduces architectures that are specifically built for high resolution segmentation, along with advanced convolutional layers that significantly reduce the cost of building the proposed model. Additionally, a candidate network is selected that will serve as the basis of the devised model.

In Chapter 4 first the subject of the thesis is discussed under Section 4.1 followed by the (aero)dynamic framework in Section 4.2 and Section 4.3 within which the context of the methods and experiments are stated. An efficient system identification technique is also described in Section 4.4 that is used for inferring the characteristics of the dynamical system. The actual neural network based surrogate model is devised in Section 4.5.

Results are presented in Chapter 5. The chapter first discusses the outcome of network optimization in Section 5.1. Then, general network performance is addressed via steady simulations Section 5.2 and unsteady harmonic oscillations Section 5.3. The computational efficiency of the model is addressed in Section 5.5. Additional results are enclosed in appendix A

At last, the conclusion is drawn in Section 6.1. As closing remarks, recommendations considering future work are shortly discussed in Section 6.3.

2

LITERATURE REVIEW

The proposed surrogate reduced-order model utilizes deep convolutional neural networks. To understand how such algorithms operate, a concise description of the basics is presented in this chapter. The first part of the chapter will introduce the principle of neural networks and it will discuss their basic components. After, the specialties of convolutional networks will be shortly considered. As the literature of these topics is vast, only the relevant aspects are considered.

2.1. INTRODUCTION TO NEURAL NETWORKS: UNDERSTANDING THE COMPUTATIONAL BASICS

Neural networks are intelligent algorithms that learn ways of solving problems on their own from observational data. The earliest algorithms were created to mimic the mammal nervous system, therefore the name.

Typically, NNs are constructed by a set of connected units called artificial neurons. Each neuron is connected to at least some of the other neurons in other layers of the network transmitting signals among themselves via their shared connection. Contrary to biological nervous systems, artificial neurons are mostly arranged in separate tiers, or layers in other words. The first layer is called the input layer, the last is called the output layer. Any layers in between are referred to as hidden layers. Signals typically travel in an organized manner throughout the networks from the input to the output layer. Each layer applies certain transformations to their inputs extracting more and more abstract features from the signal as it propagates through the network. Subsequent layers base their responses on the abstraction passed down by the preceding layer. In the output layer information is summarized into the final response of the network [18–20].

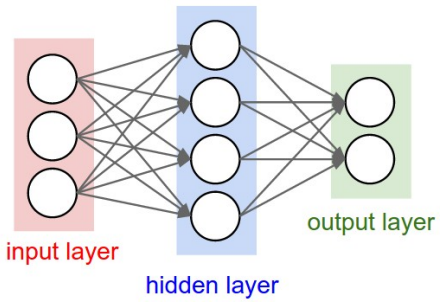
In the following sections the above-mentioned components, transformations and learning procedures of the layers and complete networks are reviewed.

2.1.1. GENERAL NETWORK TOPOLOGIES

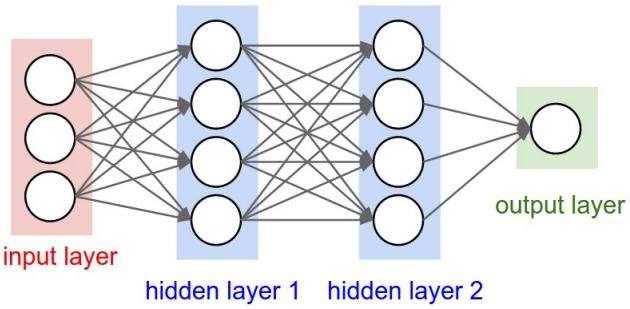
Simple neural networks commonly employ fully connected (FC) layers. In two adjacent FC layers each neuron from one layer is fully pairwise connected to all neurons in the other layer, but neurons of the same layer do not share any connections. Examples in fig. 2.1 show the schematics of a 2 and 3 layer fully connected network topology.

The basic computational units of the networks are the artificial neurons. Their model is derived from the neurons and synapses of the brain. Biological neurons receive signals from other neurons via their dendrites. After the cell body processes the signals, it sends an output down its one axon. Eventually, the axon will split into multiple branches, connecting the axon to the dendrites of numerous other neurons [21]. A sketch of the biological neuron is shown in fig. 2.2a.

In the basic mathematical formulation, the synapses are modelled as weighted signals. The dendrites carry those signals to the cell body where all the incoming signals are summed. If the result reaches above a certain threshold, the neuron is fired, sending a spike down the axon. The conditions that determine whether the neuron shall fire are defined by the activation function of the neuron [21]. The mathematical model is illustrated in fig. 2.2b.

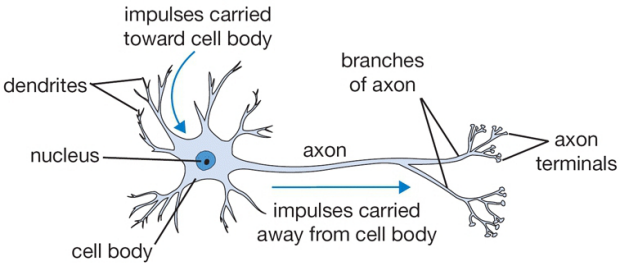


(a) A fully connected 2-layer neural network (with 3 inputs, 4 hidden and 2 output neurons). The input layer is not counted [21].

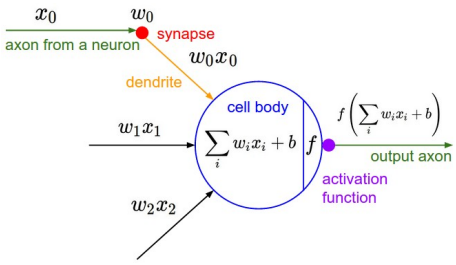


(b) A fully connected 3-layer neural network (with 3 inputs, two layers of 4 hidden and one layer of 1 output neurons). The input layer is not counted [21].

Figure 2.1: Schematics of fully connected network topologies.



(a) Sketch of biological neuron [21]



(b) Mathematical model of neurons [21]

Figure 2.2: Neurons in the brain (**left**) and their computational representation (**right**).

From fig. 2.2b, the operation carried out by neuron j in the network is expressed by Equation (2.1):

$$y_j = f(z_j) = f\left(\sum_i w_{ij} x_i + b\right) \tag{2.1}$$

where z_j is the sum of the incoming x_i signals that are weighted by w_{ij}. The term b is constant called the intercept – or bias term that biases the sum towards its own value. Ultimately, if no input is fed to the cell its output will take the exact value of the bias. The function f applied to that sum is called the activation function. It defines the output signal y_j that the neuron sends down its axon. The activation functions are applied in an element-wise manner to all z_j sums. Without being exhaustive, classical activations for instance would be the sigmoid function:

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

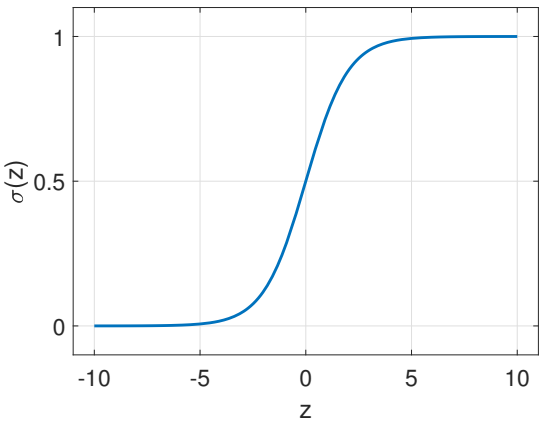
or the tangent hyperbolic activation function:

$$f(z) = \tanh(z) \tag{2.3}$$

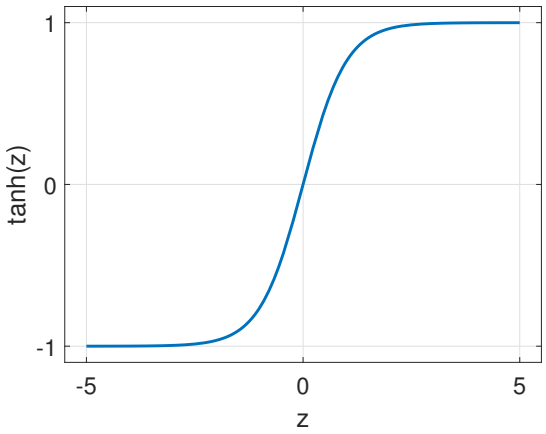
The shapes of the two functions are illustrated in fig. 2.3. Summarizing the above operations, the neurons essentially apply an affine transformation to their inputs, followed by an element-wise nonlinearity[18, 21]. Then, the basic idea of the neuron is that the weights and biases are adjustable, so the neurons can be optimized to approximate a targeted function sufficiently well. Importantly, the f activation is chosen to be a nonlinear function to make the system suitable for handling complex nonlinear tasks as well [18, 21].

2.1.2. TRAINING: OPTIMIZING NETWORK PARAMETERS

The procedure of adjusting the parameters of the neurons is called training. During training, the network is shown examples of some tasks desired to be approximated by the network. If the training is successful, the



(a) Sigmoid activation function defined as $1/(1+\exp(-z))$



(b) Tangent hyperbolic activation function defined as $\tanh(z)$

Figure 2.3: Classical nonlinear activation functions used in neural networks.

network will be able to solve problems similar to the examples without actually including them in the set of training examples.

Training is effectively an optimization. However, for learning algorithms optimization is usually applied indirectly because, the performance measure P of the optimization defined with respect to the examples is intractable in many cases. Therefore, instead of optimizing the measure P directly, training aims to improve some additional cost (or loss) function. Then, improving the cost function is hoped to improve the performance measure as well [18]. The procedure is usually realized in minimizing the cost function.

For mathematical formulation consider certain inputs $\mathbf{x}^{(i)}$ for which the desired outcome is $\mathbf{y}^{(i)}$. With $\mathbf{w}$ and $\mathbf{b}$ being the weights and biases, the cost measured over individual examples can be given in the form of $C_x(\mathbf{w},\mathbf{b},\mathbf{x}^{(i)},\mathbf{y}^{(i)})$. The cost of course depends on the $\mathbf{w}$ weights and $\mathbf{b}$ biases as well, as they are being initially unknown, and the goal of the training is to find appropriate sets for them, so the cost can be minimized. However, C_x is only the cost of a single example. The complete cost function measured over all examples in the training set can be defined as an average of the individual costs:

$$C = \frac{1}{n}\sum_x C_x(\mathbf{w},\mathbf{b},\mathbf{x}^{(i)},\mathbf{y}^{(i)}) \tag{2.4}$$

and can be minimized by altering the parameters $(\mathbf{w},\mathbf{b})$ [18, 19].

Today, training neural networks generally relies on gradient descent (GD) algorithms. The algorithms compute the direction of the steepest negative slope of the cost function, with respect to changing the parameters, and step slightly in that direction. The process is repeated until the results are satisfactory.

To find the direction of the steepest slope, it is necessary to calculate the gradient vector ∇C of the loss function with respect to all of the parameters. After, the parameters can be adjusted accordingly. If the vector of changes is given as $\Delta\mathbf{v}$, provided that those are being small, the change in the cost function can be approximated as [19]:

$$\Delta C \approx \nabla C \cdot \Delta\mathbf{v} \tag{2.5}$$

To ensure a negative change in the losses, the change in parameters is chosen to be:

$$\Delta v = -\eta \nabla C \tag{2.6}$$

where $\eta \in \mathbb{R}^+$ is a small number, called learning rate. In effect, it determines the size of the step made by the algorithm. Substituting Equation (2.6) back into Equation (2.5) yields:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta ||\nabla C||^2 \tag{2.7}$$

Since $||\nabla C||^2 \geq 0$, the change in the cost can only be zero or negative. Advancing in steps, the update rule for the changes is:

$$v \rightarrow v' = v - \eta \nabla C \tag{2.8}$$

The gradient is a multidimensional generalization containing each of the partial derivatives of the cost function with respect to each variable. The variables are now the weights and the biases of the network. Using Equation (2.8), the update rule for the variables can be expressed as:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \tag{2.9}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \tag{2.10}$$

Essentially, that is how the networks are trained. The problems boil down to finding the partial derivatives of the cost with respect to the parameters of the networks. That task however is by no means straightforward. In fact, efficient methods calculating the updates are key in machine learning. Following the breakthrough of Rumelhart *et al.* [22], the standard practice is using back-propagation algorithms. The methodology is discussed in the next section.

Another important aspect of the method is it being a gradient descent optimization. Given a complex problem, the applicability of gradient descent is questionable, nevertheless, it is still the main approach being used today. To reflect on this regard, Section 2.1.6 and Section 2.1.7 will discuss how gradient descent algorithms are applied in machine learning.

2.1.3. BACK-PROPAGATION: COMPUTING THE PARAMETER UPDATES

The problem to be solved is finding the partial derivatives of the cost with respect to the weights and biases. Consider a conventional approximation, for instance:

$$\frac{\partial C}{\partial w_k} \approx \frac{C(w + \epsilon e_k) - C(w)}{\epsilon} \tag{2.11}$$

where, $\epsilon \in \mathbb{R}^+$ is small and e_k is the unit vector in the k^{th} direction. The problem with such techniques is that they are shown to be extremely slow.. Alternatively, practitioners employ back-propagation, which is a cornerstone of learning algorithms since the study of Rumelhart *et al.* [22]. Back-propagation allows the information from the cost to propagate backwards through the network, in order to compute the gradient. The back-propagation procedure is realized in the application of the chain rule for derivatives. Given a relation of $y = g(x)$ and $z = f(g(x)) = f(y)$, the chain rule states [18]:

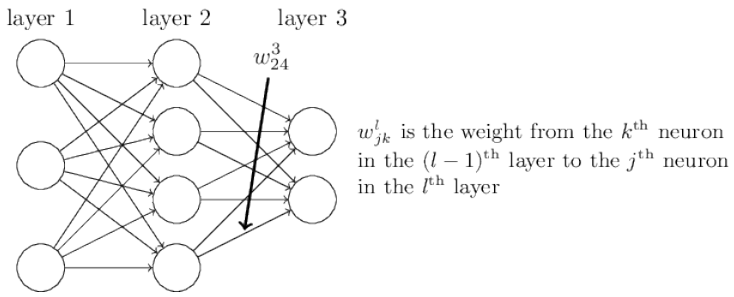
$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{2.12}$$

or expanding the formulation beyond the scalar case:

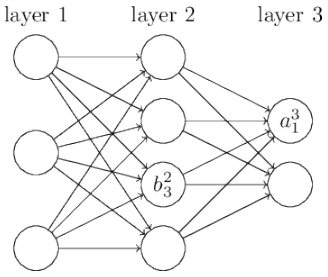
$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{2.13}$$

The premise of back-propagation according to LeCun *et al.* [23] is that "the derivative (or gradient) of the objective with respect to the input of a [layer] can be computed by working backwards from the gradient with respect to the output of that [layer] (or the input of the subsequent [layer])". This procedure can be applied repeatedly to all layers in the networks in a backward manner, starting from the output layer. Computing these gradients makes the derivation of gradients with respect to the parameters straightforward.

Take for example a simple multilayer feedforward network like the one shown in fig. 2.4. The parameter w of l and jk represents the weight between k th neuron in the $(l-1)$ th layer and the j th neuron in the l th layer as shown in fig. 2.4a. Similarly, b and a of l and j denote the bias and the activation of the j th neuron in the l th layer respectively, just as in fig. 2.4b.



(a) Weight on a connection between neurons in the network [19]



(b) Notations of bias and activation in use [19]

Figure 2.4: Notations of weights and biases.

So, the relation among the activation of the j^{th} neuron and activations from the previous layers can be expressed as:

$$a^l_j = f\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right) = f(z^l_j) \tag{2.14}$$

where the sum z^l_j is the weighted input of the j^{th} neuron in the l^{th} layer. Assuming that the estimation is imperfect, the approximation of the cell will be disrupted by a small amount of Δz, resulting in an activation of:

$$a^l_j = f(z^l_j + \Delta z^l_j) \tag{2.15}$$

This error propagates through the entire network changing the final loss by an amount of $(\partial C/\partial z^l_j)\Delta z^l_j$. The variance of the cost with respect to the weighted input is defined as the error of the j$^{\text{th}}$ neuron in the l$^{\text{th}}$ layer:

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} \tag{2.16}$$

Now the goal is to determine the changes in the cost with respect to the weights and biases, namely $\partial C/\partial w^l_{jk}$ and $\partial C/\partial b^l_j$. Using the chain rule from Equations (2.13) and (2.16) the sought quantities can be derived in a backward manner. First the error in the output layer (denoted by "L") is:

$$\delta^L_j = \frac{\partial C}{\partial z^L_j} = \sum_i \frac{\partial C}{\partial a^L_i}\frac{\partial a^L_i}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j} f'(z^L_j) \tag{2.17}$$

where the summation of the chain rules drops out since the activation a^L_k is interpreted over z^L_j only if $i = j$. The terms in the final form on the right-hand side simply measure the variance of cost with the output and the variance of activation with the weighted inputs. All these are straightforward calculations, given that the total cost is an average over the individual costs and that those costs are functions of the outputs. Since the form of the cost function is known just as the form of the activation (both chosen upfront) these derivatives can be worked out easily. For instance, the derivative of the sigmoid function introduced in Equation (2.2) would be $f'(z) = \sigma'(z) = \sigma(z)\big(1-\sigma(z)\big)$ [24]. Then, having the error in the output layer calculated, its variance with the parameters can be derived by applying Equation (2.13). First the influence of the weights is expressed:

$$\frac{\partial C}{\partial w^L_{jk}} = \sum_i \frac{\partial C}{\partial z^L_i}\frac{\partial z^L_i}{\partial w^L_{jk}} = \sum_i \delta^L_i \frac{\partial \sum_k w^l_{ik} a^{l-1}_k + b^l_i}{\partial w^L_{jk}} = \delta^L_j a^{L-1}_k \tag{2.18}$$

where the sum drops out again, since the weighted output depends only on the weight in the second

partial derivative when their indices are identical. Then, in a similar fashion, the influence of the biases can be derived as well:

$$\frac{\partial C}{\partial b_j^L} = \sum_i \frac{\partial C}{\partial z_i^L}\frac{\partial z_i^L}{\partial b_j^L} = \sum_i \delta_i^L \frac{\partial \sum_k w_{ik}^l a_k^{l-1} + b_i^l}{\partial b_j^L} = \delta_j^L \tag{2.19}$$

Changing the indices of the layers in Equations (2.18) and (2.19), a general description of the sought variances can be obtained. Then, the only remaining unknowns are the errors of the neurons in subsequent layers. So far only the error in the last layer L was calculated. In order to complete the back-propagation procedure the errors of neurons in lower tiers have to be determined as well. From Equation (2.16), the error of a neuron is computed as the variance of loss with respect to the weighted input. Rewriting the equation for the layer below the output tier (L-1) yields:

$$\frac{\partial C}{\partial z_j^{L-1}} = \frac{\partial C}{\partial a_j^{L-1}} f'(z_j^{L-1}) \tag{2.20}$$

Just as in Equation (2.17) the second term on the right hand side can be determined easily, since both the form of the activation and the value of the weighted input is known. However, propagating the error from the top to the bottom, the first term is a function of the previous layer. Therefore, a dedicated expression is required. Luckily, applying the chain rule can be applied again:

$$\frac{\partial C}{\partial a_j^{L-1}} = \sum_i \frac{\partial C}{\partial z_i^L}\frac{\partial z_i^L}{\partial a_j^{L-1}} = \sum_i \delta_i^L \frac{\partial z_i^L}{\partial a_j^{L-1}} = \sum_i \delta_i^L \frac{\partial \sum_k w_{ik}^l a_k^{l-1} + b_i^l}{\partial a_j^{L-1}} = \sum_i \delta_i^L w_{ij}^L \tag{2.21}$$

Substituting Equation (2.20) back to Equation (2.21) provides the expression for the error of the j^{th} neuron in (L-1)$^{\text{th}}$ layer:

$$\delta_j^{L-1} = \frac{\partial C}{\partial z_j^{L-1}} = \frac{\partial C}{\partial a_j^{L-1}} f'(z_j^{L-1}) = \sum_i \delta_i^L w_{ij}^L f'(z_j^{L-1}) \tag{2.22}$$

Equation (2.22) completes the cycle. As soon as the errors of a layer are calculated the procedure above can be applied repeatedly, by substituting the indices of the corresponding two layers in the place of L and L-1. This way the gradients of the weights and biases can be determined for each layer in the network. Rewriting the update rules accordingly gives:

$$w_{jk}^l \to (w_{jk}^l)' = w_{jk}^l - \eta\, \delta_j^l a_k^{l-1} \tag{2.23}$$

$$b_j^l \to (b_j^l)' = b_j^l - \eta\, \delta_j^l \tag{2.24}$$

Notice that in the update rules the prime $(.)'$ denotes the updated variable, whereas the prime of the activation (f') denotes its partial derivative with respect to the weighted inputs.

Using the presented back-propagation technique, the gradients for the optimization can be determined efficiently.

2.1.4. GENERALIZATION: APPLYING NETWORKS TO UNEXPLORED DATA

During training examples of some tasks are shown. The network is optimized in terms of its parameters such that the error measured over the shown examples, namely the training error, is minimized. If only training is considered on its own, then the problem is simply an optimization. In contrast, a learning algorithm problem includes an additional aspect, that is the expected error over novel unexplored examples is to be minimized as well. This ability is regarded as generalization and the corresponding error is called generalization error (or test error) measured over the test set of examples, that are taken separately from the training set. [18]

Since the parameters of the networks are set according to the training error, the expected value of the test error is at best equal or greater than the training error. The priority during network optimization is firstly reducing the training error to a small value. Secondly, the surplus error between training and test is desired to be small too. These criteria highlight two central aspects of training, underfitting and overfitting. The former defines a situation when the algorithm fails to achieve acceptably low training errors. The latter indicates scenarios when the gap between the two types of errors is too large. A measure of performance reflecting on these challenges is called capacity. Low capacity means large training errors whereas large capacity networks can overfit. The problem of capacity is demonstrated in fig. 2.5.

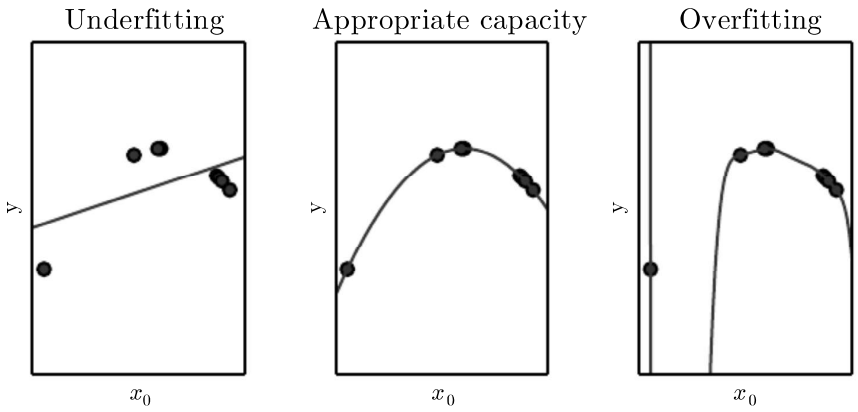


Figure 2.5: (**Left**) Linear function underfits the data, failing to capture the curvature. (**Center**) Quadratic function fits well to the data. No significant overfitting or underfitting is present. (**Right**) A 9^{th} order polynomial overfits the data [18].

As it can be concluded from fig. 2.5, learning algorithms perform best upon choosing optimal levels of capacity. One feature worth mentioning, among the many which influence the capacity, is the hypothesis space. It defines the set of functions from which the network is able to pick a designated solution. The chosen family of functions anticipates the capacity of the model. Therefore, it is referred to as the representational capacity. However, in practice the algorithm might not find the best function within the chosen set. Then, the outcome instead of being the best is only a solution which reduces the error significantly. Thus, the true, effective capacity might be lower than the representational capacity. The relevance of this argument is that a sufficiently complex description of the problem must be selected, provided that functions are still simple enough to ensure good generalization. Typically, while the training error asymptotically converges to some finite minimum error as the capacity is increased, the generalization error takes an U-shape over the training error [18]. These characteristic are depicted in fig. 2.6.

2.1.5. VALIDATION: ADJUSTING THE HYPERPARAMETERS

Networks have several settings influencing their behaviour. These are called hyperparameters en masse. For example, the variables controlling the capacity are a subset, e.g. capacity hyperparameters. What separates the hyperparameters from others like the weights and biases is that the selection of hyperparameters are not part of the training. For instance, if the algorithm would be able to specify its own capacity, it would maximize it to achieve the smallest possible error, resulting in an overfit. Hence, to asses the hyperparameters, an additional collection of examples is required, called validation set. The validation set is subtracted from the training data. Standard practice is to split the training data into two groups. The first one is the true training set, making up roughly 80% of the examples. The remaining 20% is kept for validation. The training set is used to learn the parameters, and the validation set is used to tune the hyperparameters. Consequently, the validation set underestimates the generalization error, since the hyperparameters are optimized to decrease its value over the validation set. After the full training cycle is completed and all degrees of freedom (DOFs) of the algorithm are set, additional test sets of examples can be trialed to measure the generalization error [18].

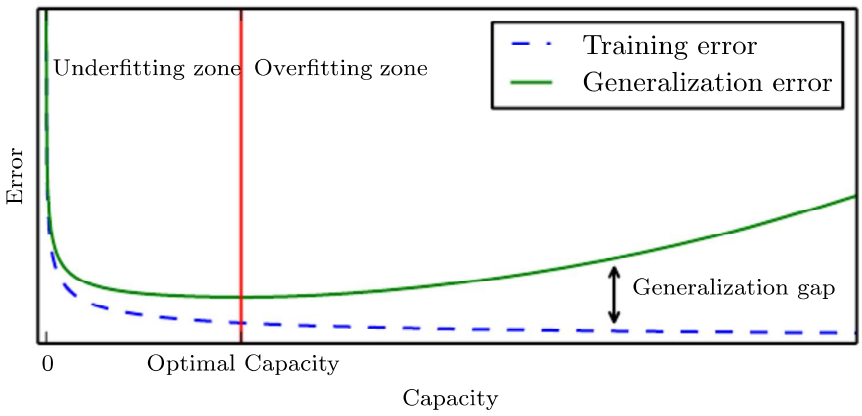
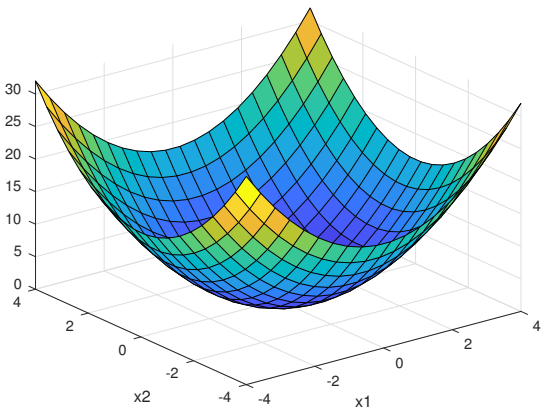


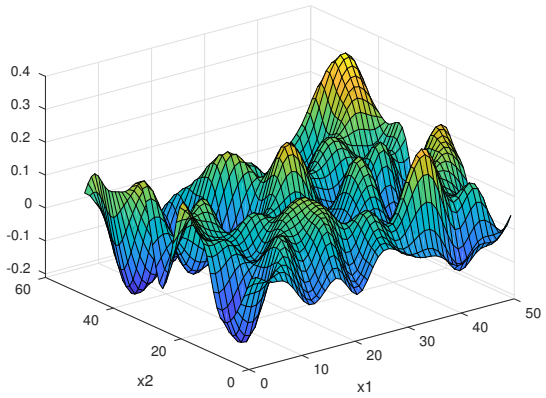
Figure 2.6: Correlation between error and capacity. The optimal capacity of the algorithm is shown by the vertical (**red**) line. Left from the optimal capacity the model is underfit. Both errors are high, although, the gap in-between is small. Increasing the capacity will lower the training error, but will increase the surplus error. Still the generalization error decreases, until reaching a minimum, which after the gap will outweigh the decrease in training error. Consequently, the test error will rise. This region is defined as the overfitting zone. The two zones are separated by the optimal capacity [18].

2.1.6. GRADIENT DESCENT IN BACK-PROPAGATION: OPTIMIZING NETWORKS FOR NON-CONVEX PROBLEMS

The previously presented training procedure, as mentioned in Section 2.1.2, is a gradient descent optimization problem. Simple problems, e.g. minimizing convex functions over convex sets, can be reduced to finding a local minimum. Then, any local minimum is the global minimum as well. See fig. 2.7a for an example. In those cases, gradient descent can be readily applied. However, loss functions of neural nets usually are non-convex, e.g. the example surface shown in fig. 2.7b. In fact, large, deep neural networks are essentially guaranteed to have large numbers of local minima. This means, that if GD is applied to non-convex network models, the optimization may get stuck in a local minimum [18, 25].



(a) Example of convex loss surface with two parameters (x_1, x_2)



(b) Example of non-convex loss surface with two parameters (x_1, x_2)

Figure 2.7: Examples of convex (**left**) and non-convex (**right**) loss surfaces with two parameters. Note that neural networks may have millions of parameters.

In past decades many practitioners believed that getting stuck in local minima were a common problem jeopardizing optimization. However, that does not appear to be a problem anymore. First, many of the local minima are due to model identifiability issues. If there would be a sufficiently large training set that can rule out all but one set of model variables, then the model is said to be identifiable. On the other hand, models that have latent variables, like neural nets, are often not identifiable because equivalent models can be obtained just by swapping the latent variables. Furthermore, depending on the the activation functions in the model, parameters of the network can be scalable. These issues can result in virtually infinite numbers of

local minima. However, all minima arising from these problems have equivalent costs and they do not mean a problematic form of non-convexity [18].

Local minima are problematic if their cost is much larger than the global optimum. Concurrently, in many cases local minima are not critical. Researchers suspect that, "for sufficiently large neural networks, most local minima have a low cost function value, and that it is not important to find a true global minimum rather than to find a point in parameter space that has low but not minimal cost" [18].

MOMENTUM OF GRADIENT DESCENT

In simple gradient descent algorithms, the direction of the update is directly controlled by current gradients. However, it can be advantageous if the history of preceding gradients is taken into account. A simple analogy would be an example of a ball, that is let gone from a hill. The path of the ball would not only depend on the instantaneous slope of the hill, but also on the inertia of the ball. Similarly, certain gradient descent algorithms mimic that inertia effect via their so-called momentum. In that case, updates are not only a functions of current gradients, but they also depend on the rate of the preceding update [18, 25]. Before formulating the update rule for gradient descent with momentum, first recall the general update rule from Equation (2.23):

$$v \rightarrow v' = v - \eta \nabla C(v) \tag{2.25}$$

In comparison, the generic formula of gradient descent with momentum is given as [18]:

$$v \rightarrow v' = v - \eta z' = v - \eta(\beta z + \nabla C(v)) \tag{2.26}$$

where a new term, z, appears. The role of this parameter is to account for past gradients. The parameter β in the expression controls the inertia of the update. The higher its value is the higher the momentum. In contrast, if β is set to 0, the original GD equation is acquired. The practical relevance of the momentum is that if it is sufficiently large, the update steps will keep their direction even if the gradient changes considerably. The direction of the steps will change only gradually. The effect of momentum on step direction is illustrated in fig. 2.8. Applying momentum helps GD algorithms to escape irregularities in the cost function while it also smooths the path of the gradient descent [25].

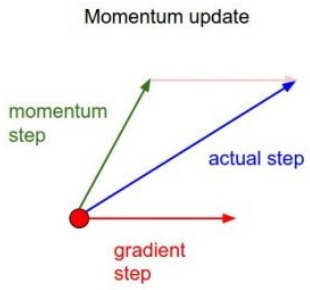


Figure 2.8: Effect of momentum on gradient step [21].

ADAPTIVE OPTIMIZATION METHODS

A difficulty of network optimization is selecting the learning rate η. To promote convergence, for basic GD algorithms it is typical to set an initial η value for all parameters which is then set to decay over time. A problem with this method is that it the learning rate is forced to follow a predefined schedule, disregarding the evolving characteristics of the loss function. Additionally, the scheduling also has to be defined. Momentum GD somewhat mitigates the problem but the initial value problem of the learning rate and its inflexibility across parameters remains [18, 25].

Adaptive methods counter those issues by treating all parameters separately assuming great variance of loss function across the individual parameters. One of the basic adaptive methods is the adaptive subgradient algorithm, called AdaGrad. AdaGrad updates the parameters based on their own gradients. To smear out the variance among the parameters the algorithm also attempts to normalize learning rates between large and small gradients [25]. The update formula for a single parameter in AdaGrad is defined as:

$$w_i \rightarrow w_i' = w_i - \frac{\eta}{\sqrt{G_i + \epsilon}} \frac{\partial C}{\partial w_i} \tag{2.27}$$

where the denominator $\sqrt{G_i + \epsilon}$ is the sum of past gradient squares of the i^{th} parameter starting from the beginning of the optimization. The term ϵ is just a very small number that ensures non-zero values for the denominator. In practice, if past gradients were large, the denominator slows down the learning rate. On the other, hand if the gradients were small it speeds the learning rate up. In effect, it drives updates towards the more gently sloped directions of the parameter space [18, 25].

AdaGrad mostly eliminates the initial value problem of the learning rate, however, the method has its own shortcomings. Its typical problem is that the accumulating squared gradients taken since the beginning of the optimization can result in a premature stop of the learning caused by excessive decrease in the effective learning rate. This effect hinders AdaGrad to solve more complex tasks efficiently. Some other algorithms like AdaDelta [26] avoid that problem by sliding a window over most recent updates only. Others, like RMSprop [18] decays the gradients by replacing the accumulations with exponentially weighted averages. These advanced algorithms offer more robust performance [18, 25].

2.1.7. BATCH, STOCHASTIC AND MINI-BATCH GRADIENT DESCENT: OPTIMIZING LARGE DATASETS

Another challenge that has to be faced when applying GD to neural nets, is that it can be extremely slow, depending on the size of the training set. Basically, all examples would have to be evaluated at each update. An important feature of neural networks, also mentioned in Section 2.1.2, is that the loss functions are usually summed over the examples of the training set. This provides the possibility to divide the training set into smaller subsets. Then, instead of calculating the exact value of the cost the updates will be based on an approximation of the cost. A motivation behind doing so is that the returns or reductions in the cost function do not scale linearly with the number of examples observed. Therefore, if large sets are to be implemented it is not worth computing the exact gradients using each sample. In fact, optimization can be run faster if rapid estimates of the gradients can be used instead of the computationally heavy exact gradients [18].

Depending on the size of the training sets are used to compute gradients, three main branches can be distinguished. The first one is called batch gradient descent (BGD). Consider the complete training set as a large batch of examples. In a BGD method, the complete batch is processed for a single update.

The second group is called stochastic gradient descent (SGD). This method is exactly the opposite of BGD. In this case, single examples of the batch are processed separately in a randomly order. Meaning that each gradient will be calculated with respect to a single point. Although it is possible that a single example may not necessarily give a good approximation of the actual gradient, but if all examples are iterated, then fluctuations of both the gradient and residual will average out and converge towards a good solution. Additionally, SGD methods generalize better over the training set and help avoiding being stuck in an undesired local minima or other irregularities [18, 25, 27].

In practice, many algorithms fall between the two previous group, constituting the third one, called mini-batch gradient descent (MB-GD). These methods use more than one example per update but use fewer than all. The whole set is randomly split into N equally sized mini-batches, each containing K samples. The number of samples can range from just a few to hundreds or maybe thousands. There is no strict recipe that would determine the number of samples in a mini-batch, instead it is driven by multiple factors. For example, larger sets provide more accurate approximations of the gradients, but work slower. If all examples are processed in parallel (usually the case), then all of them are loaded into the memory at once and the required memory scales with the batch-size. For many applications it can be a limiting factor for K. Also, certain hardware like GPUs achieve better performance with certain batch sizes. If computations are GPU accelerated, it is common to use powers of 2 size batches. Then similarly to SGD, smaller batch sizes can offer better generalization but at the same time they may require smaller learning rates to maintain convergence stability, which can increase the total number of iterations dramatically [18, 25, 27, 28].

If the problem is small, BGD can be easily applied, however, for complex and large problems MB-GD or SGD is definitely the favoured solution. Still, the exact setup to be applied is case-dependent and selection of batch sizes is empirical [18, 25, 27, 28].

An example showing the progress of a two-parameter-optimization is illustrated in fig. 2.9. BGD smoothly converges (ideal case) towards the optimum. The noisiness of the updates increases with decreasing batch size. The frequent updates and the less accurate approximations of the gradient cause high variance in the

parameters. It is possible that SGD and MB-GB will not actually find the true optimum, however, they can definitely reach good enough solutions in its proximity [18, 25, 27, 28].

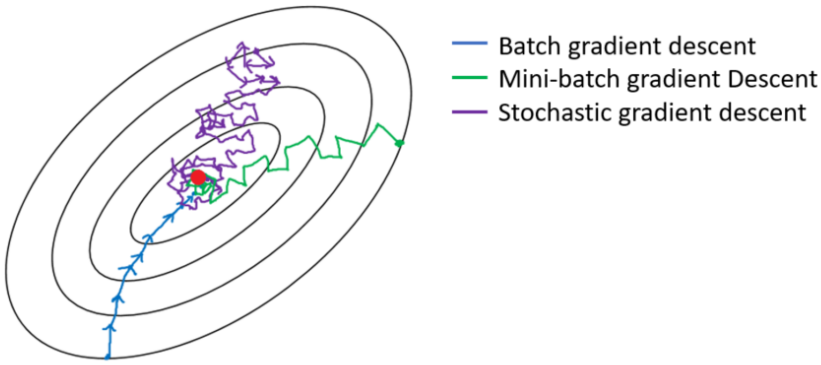


Figure 2.9: Comparison of batch (**blue**), mini-batch (**green**) and stochastic gradient (**purple**) descent convergence [27, 28].

Fluctuation in the gradient will also appear in the loss function. Therefore, contrary to BGD which should have smoothly converging gradients as well as losses, MB-GD and SGD will also have a noisy loss function as illustrated in fig. 2.10. Depending on the variance of the parameters the loss function will fluctuate to different intensities as shown in fig. 2.11. Beside the noisiness, peaks could also appear in the training curves. Peaks could be caused by escaping some irregularities in the loss function or could also be due to examples that are hard to learn [18, 28–30].

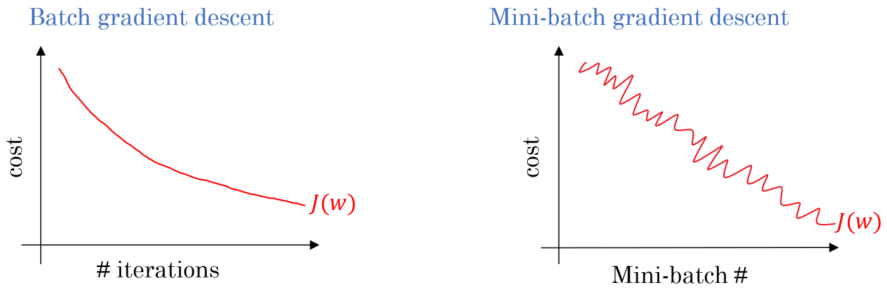


Figure 2.10: Comparison of batch, mini-batch and stochastic gradient descent training curves [27].

2.2. CONVOLUTIONAL NEURAL NETWORKS: PROCESSING SPATIALLY RELATED DATA

Having the basics of simple neural networks understood, now convolutional neural nets will be introduced. In practice, neural networks that use convolution instead of simple matrix multiplication (e.g. instead of the weighing in Equation (2.1)), in at least one of their layers, are called convolutional neural networks [18]. The layers in which convolution is used are then regarded as convolutional layers. However, similarly to the simple mathematical model of a neuron in fig. 2.2b, these layers are not limited to a single operation. Using the complex terminology of Goodfellow *et al.* [18], the term convolutional layer usually denotes a set operations, each referred to as a stage of the convolutional layers. Traditionally, convolutional layers consist up to three different stages. The first type of stage is the convolution itself. The second type (if applied) is called the detector stage, where the output of the first stage is run through nonlinear activations. Finally, third type (if applied) further modifies the output of the layer using special functions regarded as pooling. In the upcoming sections, these different stages and the properties of their operations that enable to process and solve spatially dependent data will be discussed.

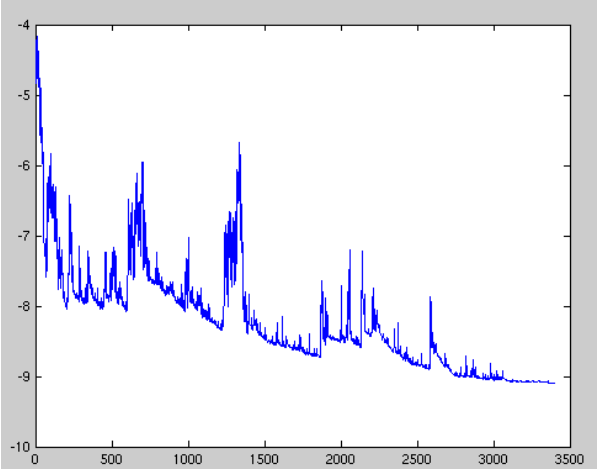


Figure 2.11: Fluctuating loss function caused by high variance parameter updates [30, 31].

2.2.1. THE CONVOLUTION OPERATION

Convolution itself is an integral operation on two functions producing a third one which expresses the overlap of one of the functions as it is shifted over the other function. The operation is usually denoted with an asterisk. It is defined as the integral of the product of two functions f and g after g is reversed and shifted [32, 33]:

$$s(t) = (f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\mathrm{d}\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\mathrm{d}\tau \tag{2.28}$$

the result s is basically the average of f weighted by g, where the first argument of the operation f can be regarded as the input. Then the second argument g, the weighing, is the kernel of the operation. Finally, the output of convolution can be referred to as the feature map. In Equation (2.28) t and τ denote time, but convolutions in general can be defined in other domains as well. For instance, it can also be performed in the spatial domain. Additionally, defining the convolution continuously over some domain is unrealistic in many cases. For that reason, discrete convolution is considered, defined as [18, 32]:

$$s[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[n-m]g[m] \tag{2.29}$$

If the kernel g of the discrete convolution has only a finite support set $\{-M, -M+1, \ldots, M-1, M\}$, then finite summation can be considered as well [32]:

$$s[n] = \sum_{m=-M}^{M} f[n-m]g[m] \tag{2.30}$$

Furthermore, if nonzero durations of both the input and kernel are found in the region $m = [0, M]$, then the summation further reduces to[32, 33]:

$$s[n] = \sum_{m} f[n-m]g[m] \tag{2.31}$$

In machine learning the input data is usually organized into multidimensional arrays. Therefore, the operation shall be extended to higher dimensions. Then, the kernels are given as multidimensional arrays as well. The input and kernel arrays may also be regarded as tensors of the algorithm. Considering a simple two-dimensional case the convolution operation van be rewritten as:

$$S[i,j] = (I * K)[i,j] = \sum_{m}\sum_{n} I[i-m, j-n]K[m,n] \tag{2.32}$$

where I denotes the tensor of inputs and K is the tensor of the kernels.

In the above equations the second formulation of Equation (2.28) is favoured, i.e. sliding the kernel over the input. The two formulations are equivalent due to the commutative property of convolution. The second formulation is favoured in machine learning because it provides a more straightforward implementation of the operation. Typically, there is less variation in m and n than in i and j. However, in this case the kernels are being flipped relatively to the input, meaning, that if the indices into the input increase, then the indices into the kernel decrease. Flipping the kernels is only required to obtain the commutative property of convolution, which bears little relevance in CNN implementation. Thus, many network libraries replace the convolution function with the closely related cross-correlation function. It is equivalent to the convolution if the kernels of convolution are not flipped [18]. In machine learning context cross-correlation operation, defined in Equation (2.33), is also referred to as convolution. The same terminology will be used in this report. An example of convolving tensors in 2D without flipping the kernels is provided in fig. 2.12.

$$S[i,j] = (I \star K)[i,j] = \sum_m \sum_n I[i+m, j+n] K[m,n] \tag{2.33}$$

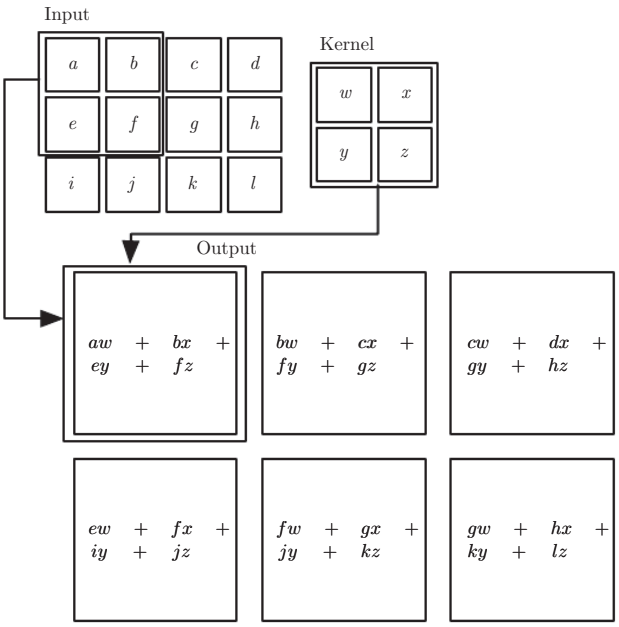


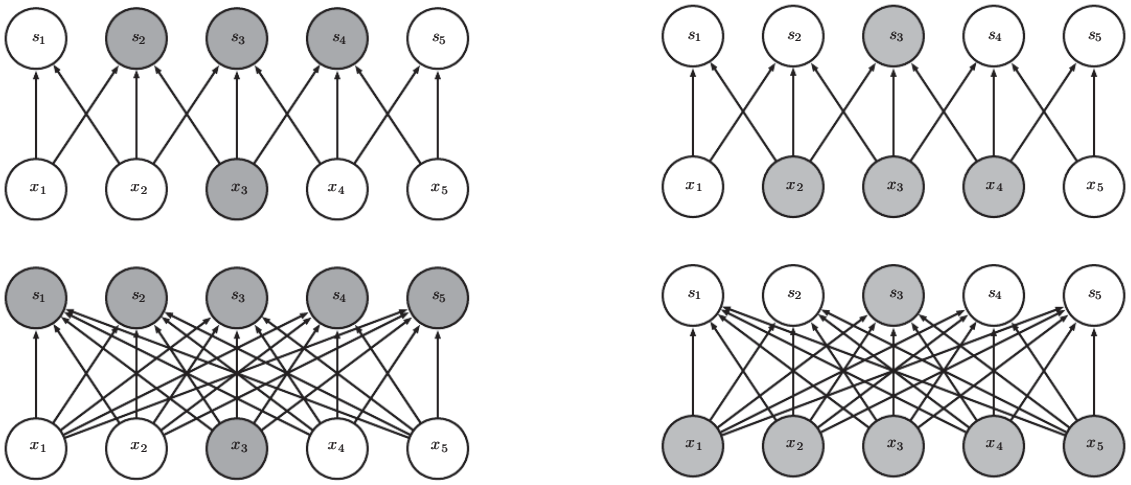
Figure 2.12: Convolution in two dimensions (without kernel flip). The kernel is restricted to the domain of the input, thus it cannot be shifted outside [18].

SPECIAL PROPERTIES OF CONVOLUTIONAL NETWORKS

Compared to other types, convolutional networks and layers utilize 3 important concepts enhancing their performance.

As it was already mentioned in Section 2.2.1 and demonstrated in fig. 2.12, the kernels are considerably smaller than the inputs. Meaning that contrary to the basic topology presented in fig. 2.1, neurons in one layer are not connected to all the neurons in the consecutive tier. This feature of CNNs are referred to as sparse connectivity. In effect sparse connectivity reduces the number of parameters and operations in the network, whilst allows searching for features in localized regions of the data. If the input and output of a tier have cardinalities of m and n respectively, in a fully connected topology it would mean $m \times n$ parameters and $\mathcal{O}(m \times n)$ runtime per example. If the number of connections to an output are limited to k, then only $n \times k$ parameters are required and the runtime is reduced to $\mathcal{O}(n \times k)$. In large networks k can be several orders of magnitude smaller than m saving huge resources. At the same time good performance is maintained [18]. Sparse connectivity is presented in fig. 2.13.

Additionally, if numerous layers are stacked, like in deep networks, neurons will still interact indirectly with large portions of the input. This way the networks can describe complex dependencies among variables by simply piling layers on top of each other [18]. This indirect interaction is illustrated in fig. 2.14.



(a) Sparse connectivity viewed from below

(b) Sparse connectivity viewed from above

Figure 2.13: Comparison of sparsely – and fully connected layers. Bottom side shows fully connected topologies. Upper side illustrates sparse connectivity using kernel size of 3 [18].

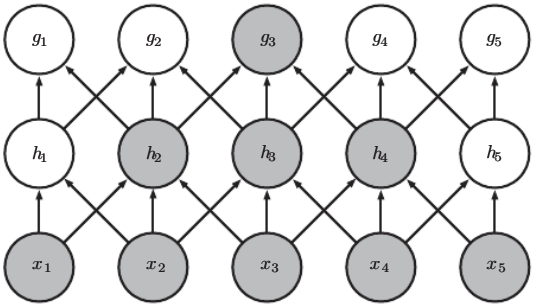


Figure 2.14: Indirect interaction of neurons in sparsely connected topologies in top-down manner [18].

Following from the previous argument and from fig. 2.12, the latter property is called parameter sharing. In a conventional, fully connected topology, each connection has its own weights, used only once. However, in CNNs, the parameters in the kernel are kept constant and used throughout the whole domain as the kernel moves. Hence, instead of learning separate sets of parameters for each unit, only a single set of parameters is learned and it is shared among the units. In practice it will mean that if a number of different kernels are employed on the same domain, the kernels can be designated for certain features in the data, acting as different filters. Such filters, or feature maps of different kernels are shown in fig. 2.15. Besides it reduces the storage requirements to k parameters [18].

The third property is a consequence of convolution and parameter sharing. It is called equivariance to translation. Meaning, if the input changes the output reacts the same way. More precisely, function $f(x)$ is equivariant to function g if $f(g(x)) = g(f(x))$. It means that convolution creates a 2-D map of where certain features appear in the input. Moving the feature in the input, will shift its representation in the output with the same amount.

APPLICATION OF CONVOLUTION IN LEARNING ALGORITHMS

Compared to theory, the functions applied in machine learning differ a bit from the mathematical formulation. First of all, as it was already implied, usually numerous kernels are used simultaneously, in order

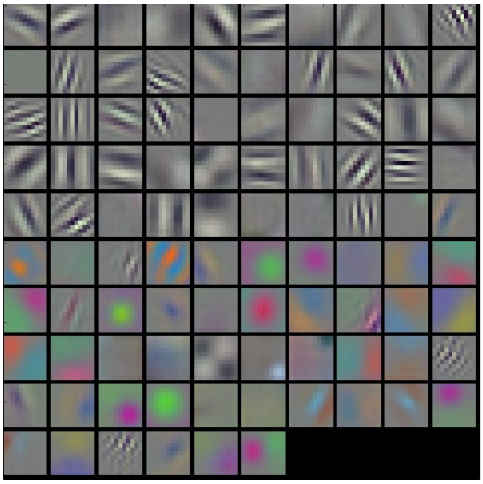


Figure 2.15: Typical-looking filters on the first CONV layer of a trained AlexNet processing images. The tiles illustrate the weights of the kernels. The filters convolve around the raw input image and fire when the examined volume contains the feature represented by the kernel [21, 34].

to extract many kinds of features. Additionally, often multiple channels are considered at the same time, meaning that multiple values are ordered to a single spatial location, e.g. RGB images. Then, processing multi-channel 2D data, convolution becomes a 3D tensor, with the 3$^{\text{rd}}$ dimension into the different channels. Such multi-channel convolution will not necessarily be commutative. They only preserve that property if the number of output channels equals the number of input channels for all operations. Considering software implementations as well, computations usually carried out in batch mode, further extending the tensor to the 4$^{\text{th}}$ dimensions, where the index represents the different examples in the batch [18].

Having a 4D tensor describing the kernels, denoted as $\mathbf{K}$, an element $K_{i,j,k,l}$ stores the weight between a unit in the i^{th} channel of the output and a unit in the j^{th} channel of the input, offset by k and l rows and columns in the spatial domain, respectively. The input data is gathered in tensor $\mathbf{I}$, where element $I_{i,j,k}$ denotes the data of the i^{th} channel at the (j,k) location. If the output tensor $\mathbf{S}$ has the same format as the input tensor $\mathbf{I}$, $\mathbf{S}$ is constructed by convolving $\mathbf{K}$ across $\mathbf{I}$ [18]:

$$S_{i,j,k} = \sum_{l,m,n} I_{l,j+m-1,k+n-1} K_{i,l,m,n} \tag{2.34}$$

In the above formula the -1 in the indices is added because in programming languages such as `Python` indexing starts from 0. Notice that the kernel is actually looking at the input in its full depth, as $\mathbf{I}$ is summed over all its channels. The index i corresponds to the number of filters applied to the input, defining the depth of the kernel [21]. A visual representation of the operation is shown in fig. 2.16.

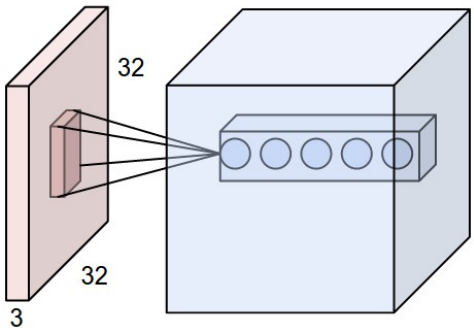


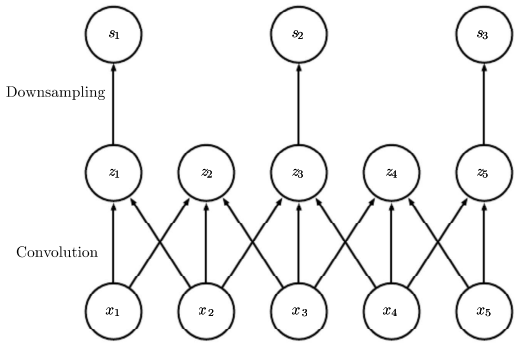
Figure 2.16: Visual example of convolution. Neurons (**circles**) in the output of the convolution (**blue**) are connected only to a limited region of the input volume (**red**) spatially, but to the full depth. The number of neurons denote the number of filters and the depth of the new layer. The 5 neurons are looking at the same region and may activate in presence of various features [21].

A usual practice in CNNs is skipping over certain positions in the domain in order to reduce computa-

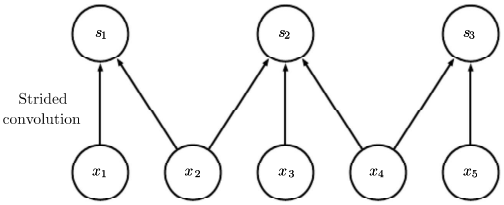
tional cost. In effect it will down-sample the output. This down-sampling in convolution is referred to as the stride, denoted by s. Sampling every s location in the domain, the down-sampling convolution function is defined in Equation (2.35) [18]. The effect of stride is illustrated in fig. 2.17.

$$S_{i,j,k} = c(\mathbf{K}, \mathbf{I}, s)_{i,j,k} = \sum_{l,m,n} \left[I_{l,(j-1)\times s+m,(k-1)\times s+n} \, K_{i,l,m,n} \right] \tag{2.35}$$

As it is evident from fig. 2.17, using strides greater than 1 will shrink the domain. Also, the same occur but to a much lesser extent if the kernel is limited to the domain of the inputs, meaning that it can visit those positions only where it entirely lies within the input volume. The latter is called unpadded or valid convolution. It decreases the size of the domain. If the width or height of the domain is m and the kernel has a size of k the output dimension will be $m - k + 1$. Of course, shrinking the volumes limits the number of convolution layers and/or kernel sizes. This behaviour can be countered by padding. Padding, or zero-padding, is an essential feature of any convolutional network. It provides the ability to pad the input volumes with zeros along the boundaries, making the domain wider. Effectively, it allows to control the output – and kernel sizes independently [18].



(a) A basic convolution with unit stride (visiting every unit), down-sampled in the consecutive step [18].



(b) A stride of 2 is applied, which practically will have the same result, however, it halves the number of convolutions and eliminates the need for the separate down-sampling procedure, hence being far more efficient [18].

Figure 2.17: Effect of stride ($s = 2$) on convolution.

2.2.2. THE POOLING STAGE: DOWN-SAMPLING CONVOLUTIONAL LAYERS

Being related to convolutions, first the third type of stage, the pooling stage is considered. For numerous tasks, handling data of varying sizes is inevitable. In image processing for instance, it can easily happen that the size of the input images varies. Then, if there is any operation in the network that is bound to receive a fixed size tensor, e.g. a fully connected layer or an element wise multiplication or summation, it must be ensured (regardless of the varying input sizes), that the tensors being fed to these operations are always of the same size. In such cases the so-called pooling operation is essential. Pooling summarizes the feature maps by replacing subregions of the received data with a statistical measure. The pooling operation works similarly to a strided convolution, but replaces the linear combination of the kernel with some other pooling functions. A popular pooling function are max pooling, average pooling, L^2 norm, etc. These functions return the maximum –, average –, L^2 norm –, etc. values of subregions [18, 19, 35]. An example of pooling is given in fig. 2.18. In the graph max pooling is applied with a kernel width of 3 and a stride of 2. Meaning, that the pooling function reads 3 neighbouring elements and outputs their maximum. Then, the kernel moves not one but two elements to the right and reads the next 3 elements. As there is no padding provided, the kernel reads only 2 values in the end.

In general, pooling simplifies information by down-sampling the features and replacing them with a summary statistic. An important property of pooling is that it makes the representations invariant to small translations in the input. In particular, detecting the presence of a feature is emphasized. Only its relative location to other features remains prominent, while its exact location is approximated more. If the exact location is of greater importance, pooling can be replaced by convolution to a certain extent, but it may sacrifice invariance to translation [18, 19, 36].

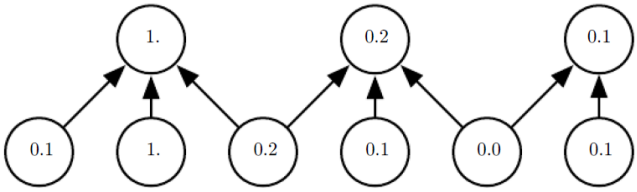


Figure 2.18: Example of pooling. The inputs are down-sampled via max-pooling that passes the maximum value forward from the kernel window. In the current example the kernel width is 3 and pooling is applied with a stride of 2 [18].

2.2.3. UP-SAMPLING: INVERTING THE CONVOLUTIONAL AND POOLING STAGES

Contrary to the down-sampling operations presented so far, i.e. strided or unpadded convolutions, or pooling, in certain cases an inverse operation is desired that is going in the opposite direction. In particular, an operation that increases the resolution of the input tensors. The goal of such operation is to obtain a tensor that has the shape of the input of some convolution or pooling, from a tensor that has the shape of the output of those convolutions or poolings, while retaining a relationship among input and output nodes compatible with those convolutions or poolings. Such operations are for instance the transpose convolutions [18, 37]. It shall be noted that it is also possible to unpool some tensors, but unpooling operation will be disregarded and will not be discussed.

TRANSPOSE CONVOLUTION

To understand the transpose convolution operation, first the basic convolution operation will be reformulated. With little modification Equation (2.33) can also be given as a concise matrix equation. To do so, first consider a 2D feature map as the input, that has k rows and l columns. Flatten this matrix to a column vector by stacking its transposed rows on top of each other. It yields a column vector $\{i\}$ with a dimension of $(k \cdot l) \times 1$.

Next, the convolutional matrix, denoted as $\mathbf{C}$, has to be constructed. The convolutional matrix is a sparse matrix where the non-zero elements are the elements of the kernel as shown in fig. 2.19. The convolutional matrix has a size of $p \times (k \cdot l)$ where p is the number of elements contained in the output of the convolution. Similarly to the input, the output can also be given as a column vector, denoted by $\{s\}$, that has p rows. Then the rows of the convolutional matrix are populated with the rows of the kernel, accordingly to the connectivity pattern between the input vector $\{i\}$ and the output vector $\{s\}$ as outlined in fig. 2.12 and in fig. 2.20. After that, the convolution can be expressed in matrix format as [37]:

1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

Convolution matrix (4,16)

Figure 2.19: Convolution matrix. The elements of the kernel (**red**) are filled in the matrix according to its connectivity patter. Remaining elements are all zeros (**blue**) [37].

$$\{s\} = \mathbf{C} \cdot \{i\} \tag{2.36}$$

The transpose of this convolution is nothing more than swapping the equation around (or the forward and backward passes in other terms) [37]. That is, obtaining the input $\{i\}$ by multiplying the output $\{s\}$ with the transpose convolution matrix:

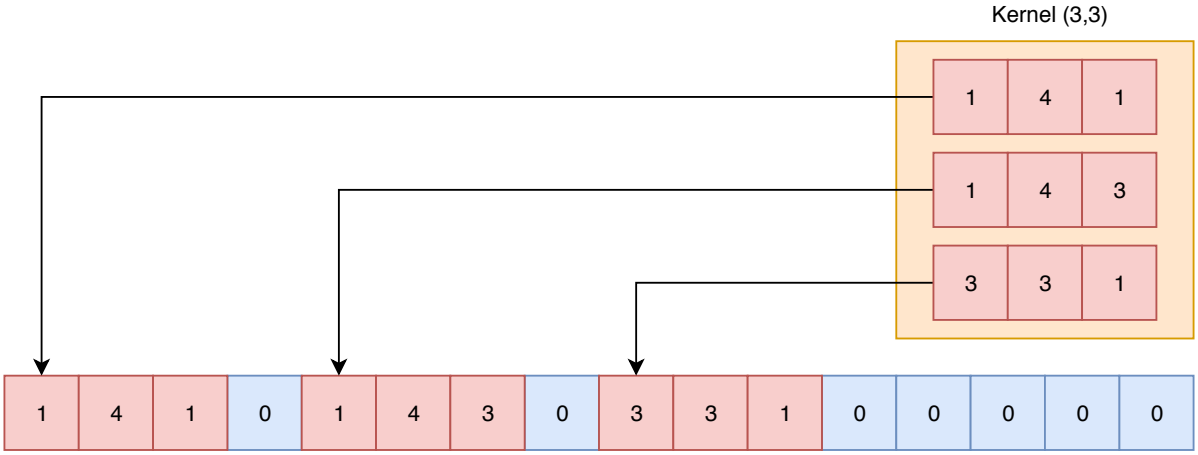


Figure 2.20: Construction of the convolution matrix. The rows of the kernel are filled in the rows of the matrix according to the connectivity pattern of the kernel [37].

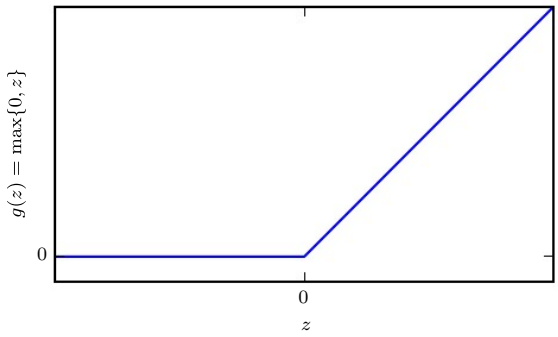


Figure 2.21: Output of the Rectified Linear Unit activation g as a function of its input z. The functions thresholds its inputs at zero, otherwise it returns the input itself [18].

$$\mathbf{C}_T \cdot \mathbf{s} = i \tag{2.37}$$

Note that the particular elements of $\mathbf{C}_T$ are not originated from the normal convolutional matrix $\mathbf{C}$. Consequently, the transpose convolution matrix is not the true mathematical transpose of $\mathbf{C}$. Transposing the convolution matrix, yielding $\mathbf{C}^T$, will only have and identical layout of non-zero elements, i.e. a connectivity pattern identical to the one of the transpose convolution matrix $\mathbf{C}_T$ [37].

2.2.4. THE DETECTOR STAGE: NONLINEAR ACTIVATIONS IN MODERN CONVOLUTIONAL NEURAL NETWORKS

Finally, the detector stage, the one that applies the nonlinear activations, is discussed.

In Equation (2.2) and Equation (2.3) two traditional sigmoidal nonlinear activations were introduced. These functions showed robust performance in the past [38]. However, in modern convolutional neural networks, their usage is very limited. Nowadays the most popular activation is the so called Rectified Linear Unit (ReLU) activations [18, 21]. ReLUs were proven to greatly accelerate convergence and result better accuracy compared to the classic sigmoidal activations [38, 39]. The basic ReLU activations employ the rectifier function, defined as:

$$f(z) = \text{ReLU}(z) = \max(0, z) \tag{2.38}$$

so the function simply returns the positive part of its argument, otherwise it returns zero [40]. Therefore, ReLUs in effect threshold their inputs at zero[21]. The shape of the function output is illustrated in fig. 2.21. As the function is being non-differentiable in 0, its derivative in practice is also rule based [41]:

$$f'(z) = \begin{cases} 1, & \text{if } z > 1 \\ 0, & \text{otherwise} \end{cases} \tag{2.39}$$

The hard saturation of the function at 0 might suggest adverse effects on network optimization, for instance preventing back-propagation of the gradients. Although, experimental results tend to contradict the generality of such hypothesis [42], it can actually happen that ReLUs are updated in a way that they will never get activated for any inputs they receive. Such ReLUs are regarded as dead. Once a neuron become dead it will probably get stuck and remain dead because the slope of the activation is 0 in the negative halfplane. From that point on, the dead neurons do not contribute to the output of the neural network. Moreover, it is possible that the dead-zone will spread across the network making large portions of it practically useless [21, 43].

To avoid dying ReLUs, among other things, additional variations of the rectifiers have been proposed, that reconsider the behaviour of ReLUs in the negative halfplane. Without being exhaustive, a few examples are:

The shifted ReLUs (SReLU) which replace the 0 threshold with a negative value:

$$\text{SReLU}(z) = \max(\alpha, z) \quad \alpha < 0 \tag{2.40}$$

Leaky ReLUs (LReLU) allow a small positive gradient in the negative halfplane:

$$\text{LReLU}(z) = \begin{cases} z, & \text{if } z > 1 \\ \alpha \cdot z, & \text{otherwise} \end{cases} \quad \alpha > 0 \tag{2.41}$$

Or exponential linear units (ELUs) which replace the negative side of the activation with an exponential function:

$$\text{ELU}(z) = \begin{cases} z, & \text{if } z > 1 \\ \alpha(\exp(z) - 1), & \text{otherwise} \end{cases} \quad \alpha > 0 \tag{2.42}$$

where the hyperparameter α, defines the threshold of the SReLU (Equation (2.40)), or the slope of the LReLU (Equation (2.41)), or the value to which an ELU saturates for negative inputs (Equation (2.42)). The shape of the modified activations are compared to the original ReLU in fig. 2.22. As it is evident from the graph, SReLUs have the same shape, but the function is shifted downwards along the positive stem of the function, as defined by the new threshold. Leaky ReLUs still have a sharp break, but the function still preserves a small slope on the negative side. In contrast, the ELU activations smoothly converge to the value where they saturate, as defined by exponential function in Equation (2.42).

Besides being more resilient against vanishing gradients, as Clevert *et al.* [44] argues, the new variants have an advantage over the basic ReLUs. Having negative values the modified linear units bring the mean of the activations closer to zero, thereby close the gap between the normal and natural gradients [45], resulting a speedup in training. Furthermore, as special property for ELUs, contrary to other ReLU activations, "ELUs have a clear saturation plateau in [their] negative regime", allowing for more robust and stable representations. As an outcome, the modified versions can outperform the original ReLU activations over various datasets. Clevert *et al.* [44] also concluded that out of the 4 activations (including the basic ReLU), their newly defined ELU activations showed the best performance. A comparison of CNN performance using the different activations is shown in fig. 2.23. The upper side of the graph illustrates training loss, whereas the bottom side shows the test errors for the CIFAR-100 dataset [46]. As it is evident from the figures, ReLUs have the lowest performance and ELUs have the highest.

2.3. SUMMARY OF LITERATURE REVIEW

Neural networks are built from artificial neurons that are arranged in separate layers. Neurons in the layers are connected to at least some of the other neurons of other layers and they transmit signals among them-

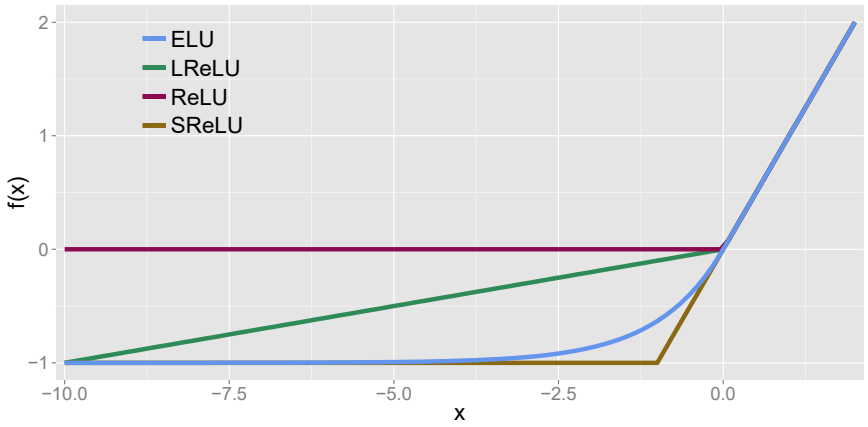
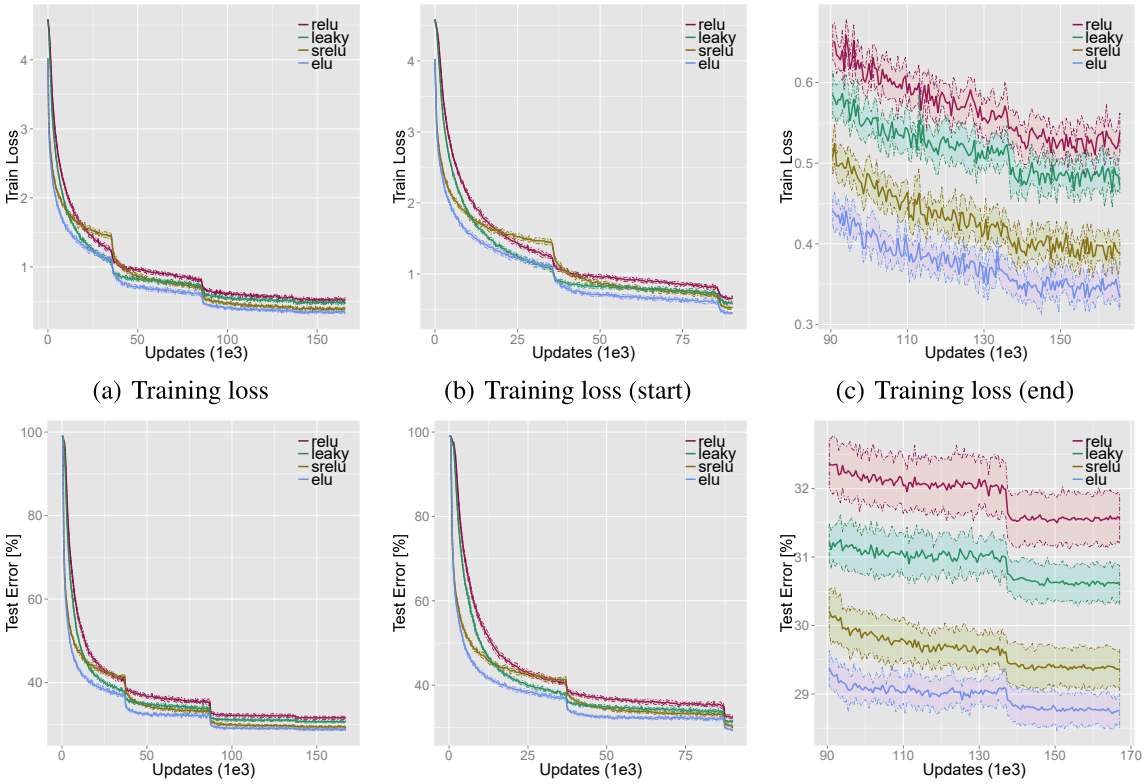


Figure 2.22: Comparison of the ReLU (**purple**), the SReLU (**brown**, $\alpha = -1$), the LReLU (**green**, $\alpha = 0.1$) and the ELU (**blue**, $\alpha = 1.0$) activations [44].



(a) Training loss (b) Training loss (start) (c) Training loss (end)

(d) Test error (e) Test error (start) (f) Test error (end)

Figure 2.23: Performance comparison of different CNN architectures using ReLU (**purple**), SReLU (**brown**), LReLU (**green**) and ELU (**blue**) on CIFAR-100 [44]. Figures a–c show the training loss, and figures d–f illustrate the test error. The CNN with ELU activations achieved the lowest errors [44].

selves via their shared connections. Signals typically travel in an organized manner from one end to the other. The inputs of the networks are mapped to their outputs via the numerous transformations and operations of the neurons. The mathematical model of the neurons is conceived as a set of affine transformations followed by a nonlinear activation. The basic idea behind neural networks is that the parameters of the linear transformations are adjustable, so the networks can be tuned to approximate arbitrary signals [18–20].

The procedure in which the parameters are tuned is regarded as network optimization or training. During training example problems are shown to the networks, from which they try to learn the underlying mapping among inputs and outputs. To evaluate the progress of training, loss or cost functions are defined prior to the optimization as performance measures. The goal of the training is to minimize the costs. To do so, the parameters of the networks are updated based on the costs measured over the examples. The update rules of the parameters are defined using the chain rule. The chain rule allows for working the gradients of the costs backwards from the outputs to the inputs. This way, the gradient of the cost with respect to a single parameter of the network can be computed [18, 23]. This methodology is called back-propagation [22].

back-propagation in effect is gradient descent optimization. Applying the technique to complicated, non-convex problems is non-trivial. First the constructed neural networks must be able to provide sufficiently complex descriptions of the investigated problems. Still, given the complexity of the problems, usually optimizations only pursue solutions that reduce the losses significantly, but might not actually be the best possible solution. Consequently, the optimization may end up in a local minima instead of the global one. However, most local minima of networks with sufficiently complex description have low costs and are not problematic, therefore it is again not important to find the true global optimum. Additionally, to help avoiding bad local minima and other irregularities, more advanced optimization algorithms are employed, which utilize gradient descent with momentum and/or adaptive methods. Momentum in optimization algorithms accounts for past gradients. It forces the direction of the step updates to change gradually, helping to escape irregularities in cost functions and smoothing the path of optimization. In contrast adaptive methods update the parameters based on their own gradients. To reduce variance among parameter updates adaptive methods also try to normalize the updates. As an outcome they drive updates towards more gently sloped gradients [18, 25].

Problems usually incorporate a large amount of data. Computing gradients on the complete data set is not efficient. Hence, the data usually split into relatively small mini-batches. Using mini-batch gradient descent the value of the current gradient can be quickly approximated. Although, the approximation may have varying accuracy, and so the loss will fluctuate as optimization proceeds. Still, training converges towards equivalently small costs, but significantly faster than batch gradient descent, which computes the actual gradient over all examples [18, 25, 27, 28].

Convolutional neural networks are of primary interest due to their ability of processing spatially dependent data. Networks that use convolutions instead of the simple affine transformations in at least one of their layers are called convolutional networks. However, in practice many neural network libraries use the closely related cross-correlation operation instead of actual convolution, due to practical considerations. Still, the operation is commonly referred to as convolution. As the problems of interest are usually discrete, the computations are realized by sliding a convolutional kernel over the input data. Entries in the input are weighted by the entries of the kernel and summed as a single output [18].

As a direct consequence of the convolution operation, convolutional networks bear three important features. The first is sparse connectivity, meaning that only subregions of subsequent layers are directly connected. In effect, sparse connectivity greatly reduces the total number of variables of the network, making computations more efficient, while it still allows to search for features in localized regions. The second feature is called parameter sharing. The kernel of the convolution is constant, hence as it slides over the input, eventually all entries in the input will see the same set of constants. Therefore, applying multiple layers of kernels on the data allows the separate layers to search for distinct features in the data. The third property of convolutional networks is a consequence of parameter sharing. As a consequence of parameter sharing convolutional networks are equivariant to translations in the input. If a feature is shifted in the input, its representation in the output will be shifted as well with the same amount [18].

Besides the convolution operation, a convolutional layer may include additional transformations. In some cases, it can be necessary to down-sample the data. If so, a simple way to do that is applying pooling. Pooling in effect replaces subregions in its input with summary statistics. Pooling is realized similarly to convolution, i.e. a pooling window is slid over the input. Typical pooling function are for instance max or average pooling [18, 19, 35].

Contrary to pooling, sometimes it is desired not to down-sample, but to up-sample the inputs. For that

purpose, transpose convolutions are considered. Transpose convolutions virtually swap the backward and forward passes of the basic convolution operation. Meaning that the transpose convolution will take a tensor as its input that has the shape of the output of some convolution and will generate an output with a shape of the input of that convolution. The connectivity pattern of the transpose convolution is equivalent to the connectivity pattern of the transposed convolution matrix of the swapped convolution. However, the weights in the transpose convolution matrix are different [18, 37].

Considering modern convolutional neural networks, the most popular nonlinear activation as of today is the rectified linear unit [18, 21, 43]. That activation employs the rectifier function [43], that simply returns the positive part of its argument. Despite providing robust performance, the rectified linear units can be prone to the dying activation phenomena, caused by the hard saturation of the function at zero [18, 21, 43]. Addressing that problem other activations have been considered. A prominent new activation is the exponential linear unit which replaces the zero threshold of the rectifier with an exponential function [44]. As a result, the exponential linear unit is less prone to vanishing gradients that may hurt network optimization. At the same time, it can offer significant improvement in overall network performance due to is clear saturation plateau and because the mean of the activation comes closer to zero.

3

STATE OF THE ART: MODERN DEEP CONVOLUTIONAL NEURAL NETWORKS

The premise of the current study is that modern deep convolutional neural networks are capable of solving the desired tasks outlined in Section 1.3 using only limited resources. To exploit the advantages of ROMs, building the models must not be computationally too demanding. In that regard it is crucial to make sure in advance that state of the art solutions bear the necessary features. Otherwise the success of the project is jeopardized. Moreover, it is not intended to build and trial networks from the ground, instead the project rather aims to implement promising techniques. For these reasons novel applications were reviewed and a candidate network was selected prior to model development. The candidate network serves as a basis that will be tailored to the present problem. The current chapter will introduce the candidate network and will discuss what are the key network properties that motivated the selection.

3.1. CANDIDATE NETWORK

A main objective of the current study is to predict aircraft/wing surface pressure distributions throughout dynamic maneuvers. That requires the candidate network to be able to operate on dense spatial data. Neural networks are often used for classification tasks, that is labeling inputs with limited numbers of classes (e.g. classifying the objects shown in images). Those networks typically encode the features of their inputs, e.g. images, into abstract descriptions of decreasing resolution from which they predict the corresponding labels [18, 21]. A simple example is provided in fig. 3.1. In the figure the a picture of a dog is classified using a convolutional network. A common practice is to subtract features of the image via a series of convolutional and down-sampling layers, where down-sampling is achieved by pooling for instance. When the output tensor of the layers becomes sufficiently small a limited number of fully connected layers classify the subtracted features by predicting the probabilities of predefined labels [18, 21].

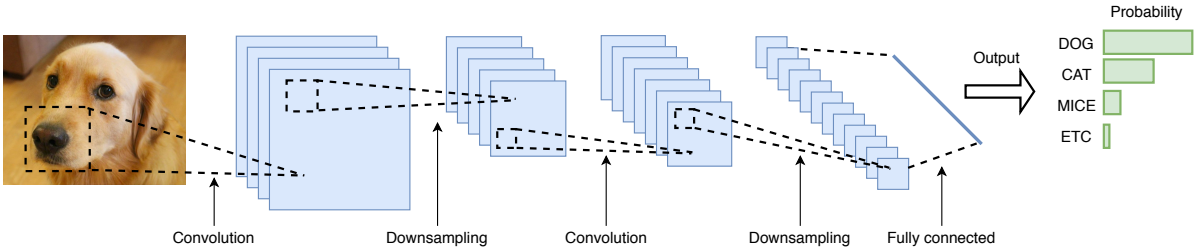


Figure 3.1: Example of image classification using an encoding convolutional network. Such networks would typically employ a series of convolutions, down-samplings, etc. Eventually a limited number of fully connected layers would classify the subtracted abstract features by predicting the probabilities of possible labels [18, 21].

Even though those algorithms process dense and spatially correlated data their outputs are restricted to only a few discrete variables. Whereas now the output should (also) be spatially distributed. Similar problems

arose in numerous image processing tasks as well, especially in the field of semantic segmentation [47], where not just the picture had to be label as whole, e.g. it shows a bus, but also each pixel in the image was to receive its own label, e.g. demarcate the bus in the image. These requirements led to development of networks that preserve the resolution of the inputs in their outputs. A general technique is to employ an encoding network just as before, but then the network is appended by a decoding part [47]. The encoding part still extracts the abstract features of the input but the decoding part now up-samples those features to higher resolutions, by using transpose convolution for instance, so each pixel can receive a label. These networks are regarded as encoding-decoding architectures. An overview of segmentation network and its workflow is provided in fig. 3.2.

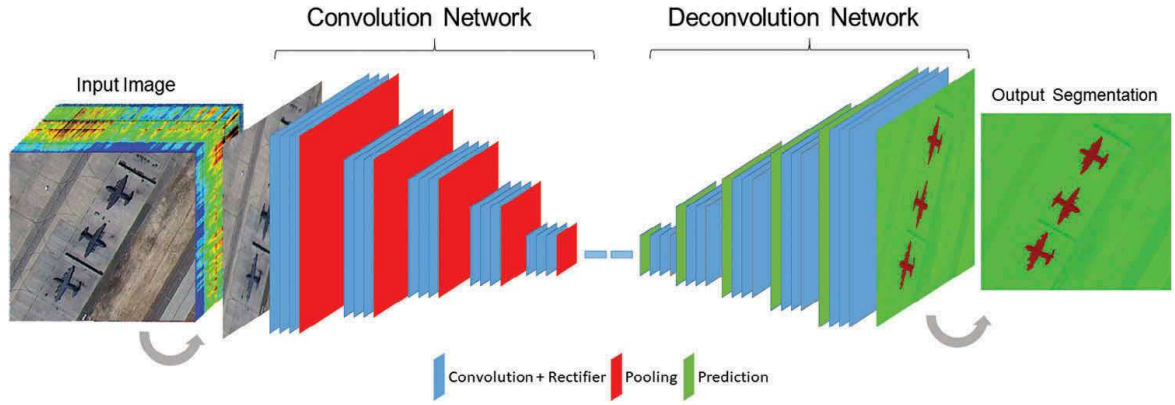


Figure 3.2: Workflow of semantic segmentation. Similarly to classification an encoding path subtracts the features of input images. The features then are up-sampled by a decoding path which in the end restores the original image resolution and labels each pixel. In this particular example the network demarcates object, i.e. airplanes on the ground [48].

Recent studies showed, that these encoding-decoding architectures are in fact applicable to steady state aerodynamic problems. Guo *et al.* [16] for example used such a network to predict steady velocity fields around various 2D and 3D objects. As its inputs the network was fed geometry descriptions of computational domains. The inputs basically described computational grids enveloping the objects. The encoding path extracted the abstract features of those geometry descriptions. Then, the decoding path predicted the velocity components at each grid point around the objects. To train and test the network they obtained their reference data from steady CFD simulations using the Lattice Boltzmann Method (LBM) [49]. A set of their results are illustrated in fig. 3.3. Comparing their test case results against reference CFD data, it can be concluded that the predictions can indeed capture the flow behaviour around the objects with high accuracy. As these problems and results were highly relevant to the current study, confidence was put in encoding-decoding networks.

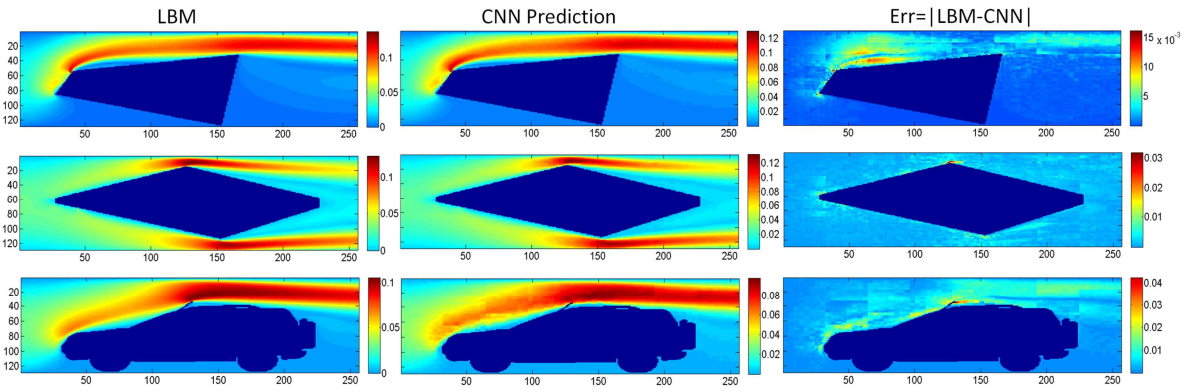


Figure 3.3: Comparison of LBM CFD simulations (**left**) and corresponding encoding-decoding CNN predictions (**middle**) of steady velocity fields. The errors among simulations and predictions are shown on the right [16].

However, a major issue still existed, namely the cost of network optimization. In many cases researchers employ hundreds of thousands or even millions of examples for training. Even Guo *et al.* [16] used some 100 000 examples. Since the examples for the current study are all being collected from CFD, such a great amount of examples would dramatically increase the costs of the complete system, questioning the advantages of the proposed ROM. Hence, it is vital to have more efficient networks.

Fortunately, other researchers already proposed solutions that in part addressed the problems of large training datasets. An important work in this regard is of Hennigh [17]. He took the idea of Guo *et al.* [16] and applied a more advanced architecture combining the developments of Ronneberger *et al.* [50] and Salimans *et al.* [51]. The problems considered were still formulated similarly to Guo *et al.* [16], i.e. the network received geometry descriptions as its inputs. Also, the encoding path extracted the relevant features again, whilst The decoding path up-sampled those features and predicted the flow properties at each point of the computational domain. The important differences were made in the internal network structure. First, Hennigh [17] built his network around the so called U-net framework of Ronneberger *et al.* [50]. Additionally, to further increase network performance, he replaced the simple convolutional layers to advanced convolutional blocks, i.e. gated residual blocks from Salimans *et al.* [51]. As an outcome, the new network only required some 3000 examples for training (5000 in total split into training and test sets) and it was able to retain prediction accuracy for problems similar to the ones investigated by Guo *et al.* [16]. Considering that he also obtained the training and reference data from LBM CFD simulations, such reduction is truly significant. Another important aspect of his study is that he extended the application of the network to compute pressure fields around airfoil geometries. The network generalized so well that it was possible to employ it in optimization procedures searching for ideal airfoil geometry.

Due to its great performance in terms of accuracy and cost, and its high relevance considering its applications, the neural network proposed by Hennigh [17] was selected as the candidate network. To understand the key features of this architecture the upcoming sections will discuss the building blocks of the network.

3.1.1. U-NET ARCHITECTURE: THE FRAME OF THE NETWORK

The U-net of Ronneberger *et al.* [50] was built specifically to do image segmentation, i.e. pixel-wise labeling, on biomedical images. Due to the scarce amount of examples, a driving factor of network development was to achieve good performance using only limited amount of data.

As mentioned earlier the U-net starts with an encoder contracting path that reduces tensor sizes and extracts the features of the inputs. After, the encoder is followed by a decoding expansive path which up-samples the extracted feature maps via successive layers. In the expansive path of the the decoder restores the original resolution of the input image and labels each pixel accordingly to the up-sampled feature maps. Importantly, to allow information to propagate fluently among the different layers, the network employs large number of feature channels. Moreover, to localize detailed features from the contracting path, the output of the contracting layers are combined with the expansive layers via skip connections. The resulting architecture yielded quasi-symmetric contracting and expansive paths showing a U-shape layout as illustrated in fig. 3.4.

The contracting side of the network is akin to a typical CNN architecture. A layer consists of stacked convolutions, each followed by a non-linear activation. At the end of the layer max pooling is applied which halves the width and height of the feature maps, but at the same time, the number of feature channels are doubled. The layers in the expansive path are basically their inverted equivalents. They start with a transpose convolution that doubles the map size but halves the number of feature channels. After up-sampling, corresponding feature maps from the contracting layers are concatenated with the up-sampled data via skip connections. The concatenated feature maps are fed through stacked convolutions again, each followed by a non-linear activation. In total the network has 23 convolutions throughout 5 layers. In the middle, the network has as many as 1024 feature channels. For future reference, the network process images of 572 by 572 pixels.

The trained network achieved very good performance on various biomedical applications. As it was desired by the authors, the network indeed needed very few images for training. Importantly for the candidate network and for the current application, the author notes, “the u-net architecture can be applied easily to many more tasks” [50].

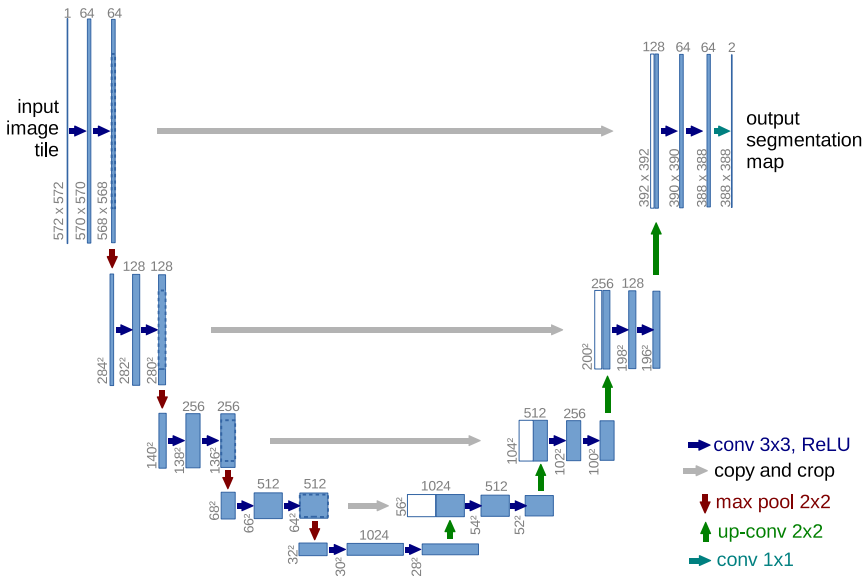


Figure 3.4: The encoding-decoding U-net architecture. The blue blocks denote the forward passing feature maps, whereas the white blocks are the tensors from the skip connections (gray arrows). The convolutions are denoted by the blue –, the pooling by the red – and the transpose convolution by the green arrows. The final segmented image is derived from the last feature map via a simple convolution. Feature map dimensions are noted around the blocks [50].

3.1.2. GATED RESIDUAL BLOCKS: ENHANCING THE PERFORMANCE OF THE CONVOLUTIONAL LAYERS

To further reduce training costs and to enhance accuracy, Hennigh [17] replaced the conventional convolutional networks of the U-net with gated residual blocks from Salimans *et al.* [51]. Concurrently, the gated residual blocks are based on the residual (ResNet) blocks from He *et al.* [52].

So, first consider the original ResNet block illustrated in fig. 3.5. Compared to the basic fully connected –, convolutional – or pooling layers discussed in Chapter 2 the key difference to be noted is the so called identity connection on the right, regarded as the identity shortcut connection. The conventional layers propagated the data along a single route. Meaning, that all values in the output of a layer had gone through the very same operations. Having the identity connection added, it establishes an additional route that clearly bypasses the operations of the layer, yet it is added to the outputs. In that regard, the identity connection is somewhat analogous to the skip connection of the U-net in fig. 3.4, but it has a different purpose as discussed below.

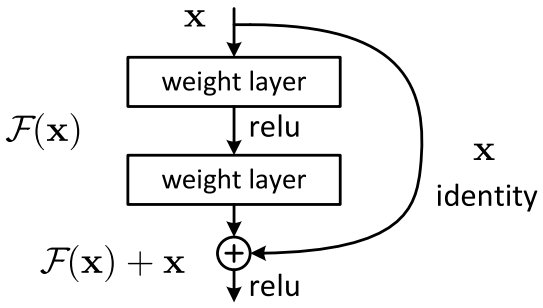


Figure 3.5: Schematics of the basic residual blocks. The identities are tunneled through the shortcut connections whereas the residuals are derived via a set layers [52].

Consider the model in fig. 3.5 as a single block now. Within the block, first there is the identity shortcut

connection, a direct path to the output. Second, there is the other path, called the residual connection, that may go through multiple operations before reaching the output. In the current example the residual connections has two weighted layers, i.e. fully connected or convolutional layers, and a ReLU activation. Given certain input x, the desired mapping of the complete block, or the output in other words, is $\mathcal{H}(x)$. Let the final output of the weighted layers be $\mathcal{F}(x)$. Then, using the identity connection the residual path is let to fit another mapping, in particular $\mathcal{F}(x) := \mathcal{H}(x) - x$. To retain the desired mapping the residual path $\mathcal{F}(x)$ and the identity path x are recast into $\mathcal{F}(x) + x$ [52].

The idea behind using identity connections is that fitting a residual mapping $\mathcal{F}(x)$ is hypothesized to be easier for the operations along the residual path, than directly predicting the complete mapping without the identity connections. In extreme scenarios for instance, if identity connections provide optimal mapping, it is easier to have residual connections outputting zeros than fitting them to the desired mapping. He *et al.* [52] compared for instance an 18 and 34 layer plain CNN, with 18 and 34 layer CNN which employed the identity connections, as illustrated in fig. 3.6. From the results it can be concluded that the adding the connections first made the networks to converge faster; second it also resulted in smaller errors both for training and for validation. These effects are indeed favourable.

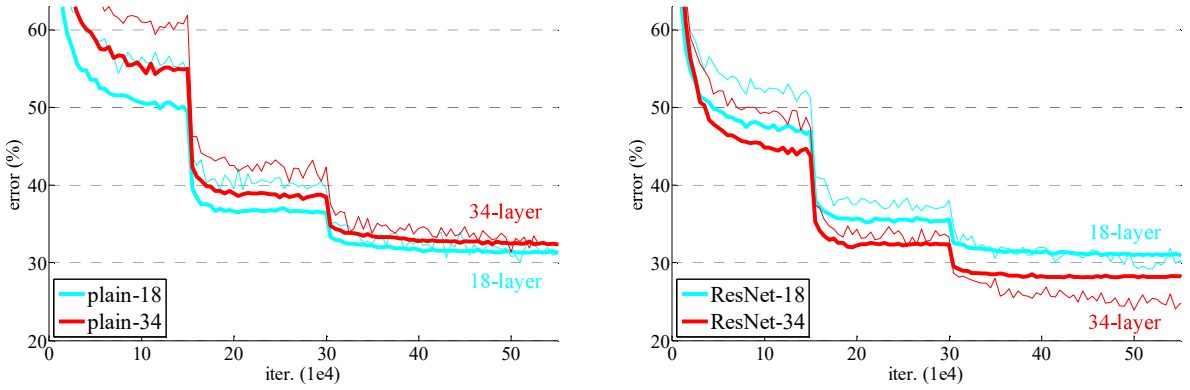


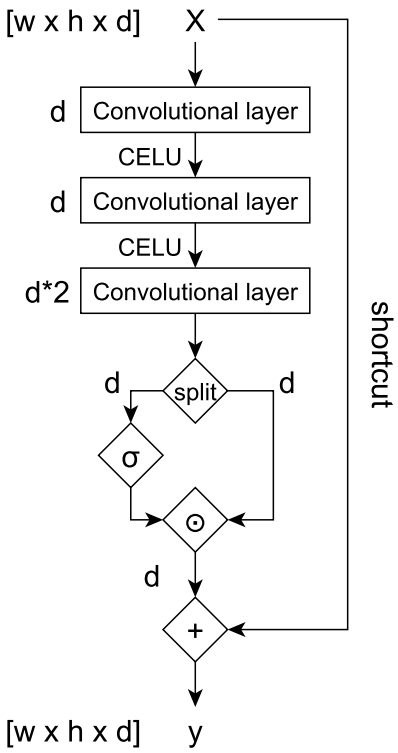
Figure 3.6: Performance comparison of plain CNNs (**left**) and CNNs with identity connections (**right**). Thin curves denote the training errors and thick curves show the validation errors. The errors are plotted as a function of iterations times 1×10^4. The modified CNNs achieved faster convergence and lower errors [52].

IMPLEMENTATION OF THE RESIDUAL BLOCKS

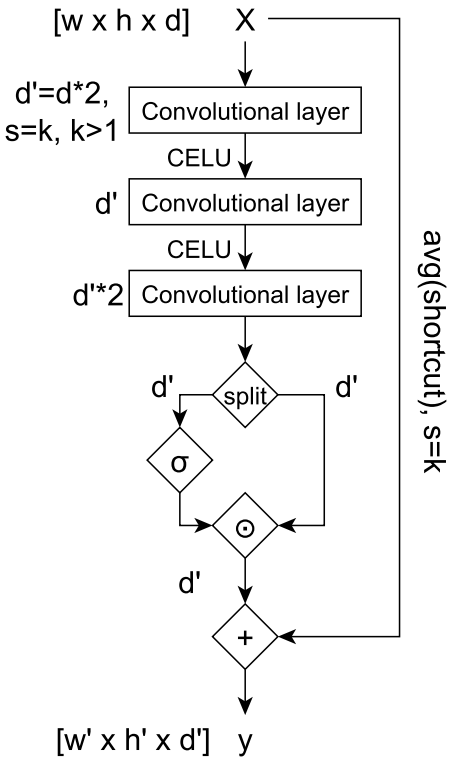
After that, now consider the modified gated residual blocks, referred to as GRB, as proposed by Hennigh [17]. The schematics of the block is shown in fig. 3.7a. It consists of three convolutions in a series connection the first two of which followed by a non-linear activation denoted by the term CELU (discussed later). Those two convolutions keep both the map size and the number of filter channels unchanged. The third convolution still keeps the maps size the same, but it doubles the channels. Then, the output of the convolution is split in half along its depth, yielding two tensors of equivalent sizes. The second half of the split is fed through a sigmoid gate, denoted by σ. The sigmoid gate is in effect a sigmoid activation, defined in Equation (2.2). Recalling its shape from fig. 2.3a, its output exists between 0 and 1. If the input is very negative it produces a zero output, and if it is very positive the output is 1 [18]. After he sigmoid gate, the gated and the non-gated arrays are multiplied together, in an element-wise manner, denoted by $\odot$. The sigmoid gate combined with the multiplication basically defines how much of the residual path is let through [53]. The output of the residual connection is then added to the shortcut connection. The advantage of using such gated residual blocks is that the gating mechanism potentially improves network performance and convergence speed [54].

Besides the above, Hennigh [17] also uses the gated residual blocks to downsample data. down-sampling can in fact be achieved by a simple modification. This modified version is regarded as the down-sampling gated residual block (DGRB). Its schematics are shown in fig. 3.7b. The new block has 2 differences. On one hand, the first convolution of the block applies a stride larger than 1, i.e. when the kernel slides through the input it moves not just one node but more. The second difference is found in the shortcut connection. Accumulating the residual and identity routes and retaining the reduced the desired downsampled size requires tensors of equal sizes, i.e. the tensor in the identity connection has to be downsampled as well. In order to

have equal residual and identity map sizes, the data in the shortcut connection is average pooled with a stride equal to that of the convolution.



(a) Simple gated residual block (GRB). Input and output tensor dimensions are identical.



(b) down-sampling gated residual block (DGRB). The block doubles the depth ($d' = d \cdot 2$) and applies a stride s larger than 1. The identity skip is pooled accordingly, so dimensions at the summation will match.

Figure 3.7: Schematics of the simple (**left**) and the down-sampling (**right**) gated residual blocks as defined by Hennigh [17]. Input and output tensor size are given in width, height, depth ($w \times h \times d$) format. The current depth d and d' is noted next to the blocks. The sigmoid gates are denoted by σ, and the element-wise multiplications by $\odot$.

ACTIVATION FUNCTIONS WITHIN THE RESIDUAL BLOCK

An additional feature of gated ResNet blocks implemented by Hennigh [17], is that he replaced the original activations in the block (all except for the sigmoid activation) to concatenated exponential linear unit (CELU) activations.

Compared to a basic ELU activation a concatenated ELU, or CELU, is an ELU activations that is applied not just to the input of the activation, but also to its negated pair. In practice it is achieved by copying the input of the activation and then negate it. After that, both arrays are fed through the activations. The equation of CELUs can be formulated as:

$$\text{CELU}(\mathbf{x}) = \text{ELU}\big(|\mathbf{x}| - \mathbf{x}\big) \tag{3.1}$$

where $\mathbf{x}$ is the input tensor of the activations. The possible advantage of concatenated activation is that it preserves both positive and negative phase information of the inputs which can considerably enhance network performance [55]. Notice that a concatenated activation will double the size of the domain. However, this growth is countered by the predefined number of feature channels that the convolutions produce. Hence, the domain size will be consistent throughout the convolutional layers.

USAGE OF RESIDUAL BLOCKS WITHIN THE U-NET ARCHITECTURE

A layer in the candidate network consists of multiple residual blocks, typically one or two (depending on the application), but the different layers in a network always have same number of residual blocks. The U-net of Ronneberger *et al.* [50] orignially used a pooling function for down-sampling in each layer, which Hennigh [17] replaced by and additional residual block. In that block the first convolution will perform a strided convolution with a stride of two. It basically halves the size of the feature maps just as the pooling function does in the U-net, but it utilizes the advantages of convolutional down-sampling discussed in Section 2.2.2. The up-sampling in the expansive path is still solved by transpose convolutions, just as in the original U-net.

3.1.3. NETWORK OPTIMIZATION: THE ADAM ALGORITHM

Besides the previous components, both the U-net from Ronneberger *et al.* [50] and the candidate network from Hennigh [17] were trained using an advanced optimization algorithm, called the ADAM algorithm [56]. The name ADAM comes from the term of adaptive moments. The method combines the momentum GD and adaptive methods (more specifically AdaGrad and RMSprop), introduced in Section 2.1.6 and Section 2.1.6, exploiting the advantages of the two methods [18, 25].

Consider a noisy objective (stochastic) function $f(\theta)$ that is differentiable with respect to the θ parameters. The ADAM algorithm seeks the minimum of the expected value $\mathbb{E}[f(\theta)]$ with respect to its parameters θ. Let the gradient g_i be the the partial derivative vector of $f_i(\theta)$, where the index i denotes the current step. So, $g_i = \nabla_\theta f_i(\theta)$. The algorithm computes the new parameters by using the moments of updated exponential moving averages of the gradient g_i and the squared gradient $g_i^2 \equiv g_i \odot g_i$. The decay of the two averages are controlled by the hyperparameters $\beta_1, \beta_2 \in [0,1)$. The averages are the estimates of the first momentum m_i (seen in Section 2.1.6) and the second momentum v_i (seen in Section 2.1.6) of the gradient [56].

The complete ADAM algorithm applies additional operations to correct for the biases experienced by the moments, but that issue is disregarded now. A pseudo code of the uncorrected ADAM algorithm is enclosed in algorithm 1.

Algorithm 1 Uncorrected ADAM algorithm (Kingma and Ba [56])

Require: α (learning rate)
Require: $\mathbb{E}[(\theta)]$ (exponential decay rates for the moment estimates)
Require: $f(\theta)$ (objective function)
Require: θ_0 (initial parameter vector)
$m_0 \leftarrow 0$ (initialize first moment vector)
$v_0 \leftarrow 0$ (initialize second moment vector)
$i \leftarrow 0$ (initialize step)
while θ_i not converged **do**
    $i \leftarrow i + 1$ (initialize step)
    $g_i \leftarrow \nabla_\theta f_i(\theta_{i-1})$ (calculate gradients)
    $m_i \leftarrow \beta_1 \cdot m_{i-1} + (1-\beta_1) g_i$ (update first moment estimate)
    $v_i \leftarrow \beta_2 \cdot v_{i-1} + (1-\beta_2) g_i^2$ (update second moment estimate)
    $\theta_i \leftarrow \theta_{i-1} - \alpha[m_i/(\sqrt{v_i} + \epsilon)]$ (update parameters)
return θ_i (resulting parameters)

ADAM is shown to be fairly robust to the choice of hyperparameters, however the learning rate sometimes has to be adjusted to the given task from the suggested default (0.001). For instance, Hennigh [17] set α to be 0.004. Overall, ADAM is proven to be robust and well-suited for various non-convex optimization problems in the field machine learning. Moreover ADAM can efficiently solve deep learning problems making it a suitable algorithm for the candidate and for the to-be-built network.

3.1.4. CANDIDATE NETWORK END-TO-END ARCHITECTURE AND HYPERPARAMETERS

The end-to-end model of Hennigh [17] is illustrated in fig. 3.8. The network receives the geometry and domain boundaries as its input. The inputs are fed to the encoder path first. The encoder consists of 7 D-Res blocks. The D-Res blocks are a series of gated residual blocks, where the last gated residual block downsamples the data using strided convolution. The down-sampling halves the feature maps and doubles the number of feature channels (filters). Until then, the size of feature maps and the number of filters are constant throughout

the blocks. The first block starts with 8 filters. In the middle of the network a simple gated residual block is present having 512 filters. The middle block is followed by the decoder with 7 U-Res blocks. The U-Res blocks are basically inverted D-Res blocks, where first the data gets up-sampled via a transpose convolution. The i^{th} U-Res block (counted from the end) is connected to its corresponding pair, the i^{th} D-Res block (counted from the front) via skip connections. After up-sampling the data is concatenated with tensor from the D-Res block, and they are fed through a series of gated residual blocks. The transpose convolution halves the filters while doubles the size of the feature maps. The convolutions in the block set the feature map size equal to the tensor size that was received via the skip connection. The last U-Res block results a tensor that has the size of the input but with 8 channels as it was defined for the first D-Res block. Therefore and additional convolution collects the different channels into a single channel. Then the output is conditioned by the boundary, i.e. flow at the boundary is set to zero, yielding the final output of the network. Throughout the network, convolutions use 3×3 sized kernels. Both down-sampling – and transpose convolutions use strides of 2.

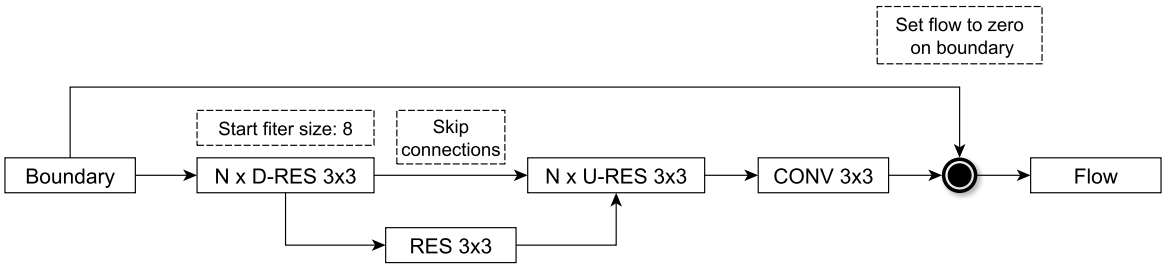


Figure 3.8: Schematics of the candidate network as defined by Hennigh [17]. Flow solutions are acquired via the encoding-decoding paths. Final feature maps are collected to a single channel by a simple convolution. Solutions are derived by enforcing the boundary conditions on the predictions.

The network was trained over 500 000 iterations using the ADAM algorithm. Data was fed in mini-batches of 8 examples.

3.2. SUMMARY

One major field of application of convolutional neural networks is image processing, i.e. labeling images or identifying certain features in them. A common approach among practitioners is to employ an encoding architecture that extracts the features of the images via stacked convolutional layers. Such layers decrease the size of the feature maps but will eventually increase the number of feature channels. as an output, the network will provide for instance the probability of certain labels that classify the image as whole [18, 21]. These networks are successful in identifying small details and features in the images, however, in the current application it is desired not just to label the input as whole, i.e. provide integral load coefficients, but to also label each location in the input. In particular, provide the pressure values on the wing. A similar field of application of convolutional networks is called semantic segmentation [47]. In such applications the goal might be to demarcate complete objects in the input. These networks do so by employing an additional path on top the encoder, namely a decoder path, which up-samples the extracted features to the resolution of the input. Despite this concept is extensively used in image processing, recent studies showed, that encoding-decoding architectures can accurately predict velocity and pressure fields around various objects. Therefore, such networks are pursued.

In order to exploit the advantages of reduced-order modelling, it must not be too computationally demanding to establish the proposed surrogate model. That is where the candidate network [17] comes into the picture. It reconsiders a simpler encoding-decoding architecture [16]. The candidate network is built around a U-shaped framework [50]. That U-shaped framework was originally devised for semantic segmentation problems where the available number of training examples are relatively low. Its key features in that regard is firstly, the number of feature channels throughout the network is high, in order to allow information to propagate throughout different layers. Secondly, the network also has a great number of layers. An additional convenience of the framework is that it is quasi symmetric. The first half of the layers constitutes the encoder. The second half of the network, the decoder, is simply the inverse of the encoder.

Besides the revised framework, in the candidate network simple convolutional layers are replaced by more

advanced residual blocks [51]. The key feature of such blocks is having identity skip connections that bypass their internal operations, establishing a second route along which data can propagate in the network. The convectional route, in which the operations are applied is regarded as the residual connection. Then, considering a single block, its output is established by the sum of the identity and residual connections. The importance of the identity connection is that, the operations in the residual connection do not have to map the complete values in the output, only the residuals that shall be applied to the inputs. As an outcome the residual blocks allow for faster convergence during network optimization and lower losses [52].

To further boost network performance, in the candidate network the concatenated version of exponential linear unit activation is used. These activations are simultaneously applied to the actual input and to its negated equivalent. This way they are able to preserve both the negative and positive phase information of the data, which can greatly enhance network performance [55].

Finally, instead of simple algorithms, the candidate network is optimized by and advanced algorithm called ADAM [56], using adaptive moments. The algorithm combines the advantages of momentum in gradient descent and adaptive methods. As a result, the algorithm works robustly on various non-convex problems.

Using an encoding-decoding architecture and by employing the discussed advanced techniques, the candidate network shows excellent performance on sample problems. It predicts pressure and velocity fields around various objects, including airfoils among other things, with great accuracy. Additionally, it only requires significantly smaller number of examples compared to previous studies. In particular, roughly 37 hundreds of examples are sufficient for training. In contrast a network with a simpler architecture may require hundreds of thousands of examples for solving similar problems [17].

4

METHODOLOGY: FLIGHT DYNAMICS, AERODYNAMIC AND SURROGATE MODEL DERIVATION

The current chapter discusses the framework of the research in which the problems are defined. The experiments are conducted on a case-study aircraft at fixed flight conditions. The following sections hence first review the aircraft itself and its mission profile motivating the selection of ambient conditions. Then, the variables describing aircraft motion is discussed, followed by the full-order aerodynamic model that is used for data acquisition. Finally, the neural network model constituting the surrogate for CFD computations is introduced.

4.1. THE MULDICON UNMANNED COMBAT AERIAL VEHICLE

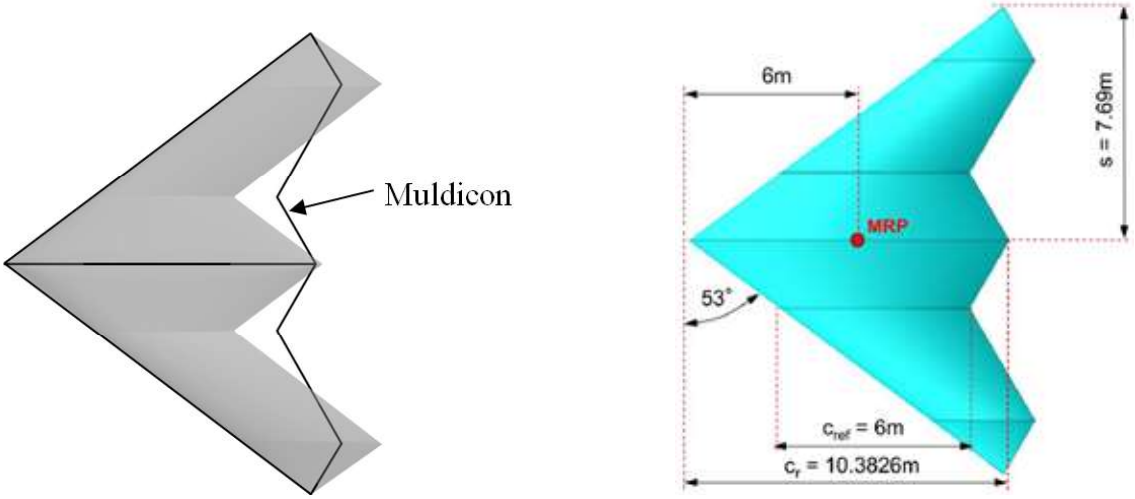
The test subject of the study is the agile Unmanned Combat Aerial Vehicle (UCAV) test configuration of the 251 th work-group of NATO Science and Technology Organization (STO) [57], called MULti-DIsciplinary CONfiguration (MULDICON) [58]. It is a tailless Blended-Wing-Body (BWB) aircraft used by numerous NATO STO participants for various studies. The main parameters of the concept are reported in Table 4.1.

Table 4.1: MULDICON main parameters [58]

Parameter	Description
Propulsion	1 Turbofan engine w/o afterburner
Engine integration	Buried
Payload storage	Internal
Payload mass	2×1000 kg
Design range	3,000 km w/o aerial refueling
Fuel reserve	$\approx$ 45 min
Cruise altitude	11 km
Cruise Mach number	0.8 [-] (all altitudes)
Stability margin	0 – 3 % MAC (stable)

The MULDICON UCAV is an enhancement of the DLR-F19 / SACCON configuration [59]. The new configuration preserves the general shape of its origins. Changes were mostly applied to airfoils and trailing edge sweeps. Old and new geometries are compared in fig. 4.1 and a generic sketch of the SACCON/MULDICON configuration is provided in fig. 4.2. The preliminary model of the MULDICON configuration is illustrated in fig. 4.3. In the current project only a generic body of the aircraft is considered lacking any inlets, outlets, mechanization or control surfaces. The reference values of the geometry are listed in Table 4.2.

The Moment Reference Point (MRP) in Table 4.2 refers to the aerodynamic moment reference point of the vehicle. For the current study it was assumed that the centre of gravity (CG) coincides with the MRP of the aircraft.



(a) MULDICON planform change [60]

(b) MULDICON planform dimensions [60]

Figure 4.1: Comparison of SACCON and MULDICON geometries.

The mission profile of MULDICON (and SACCON) is closely related to the "Bomber Low Level Penetration" military standard profile [61]. The mission radius is 1500 km without aerial refueling. Relevant design points of the mission profile are presented in Table 4.3. A detailed illustration of the mission profile is shown in fig. 4.4.

The mission starts with take-off at zero meters of altitude (above sea level). A Mach-number of 0.2 is considered. The aircraft is desired to be able to attain a maximum angle of attack of 20 degrees. Furthermore, the airplane shall be able to roll 30 degrees over 1.1 seconds (27.27 deg/s), pitch 20 and yaw 15 degrees over 1 second.

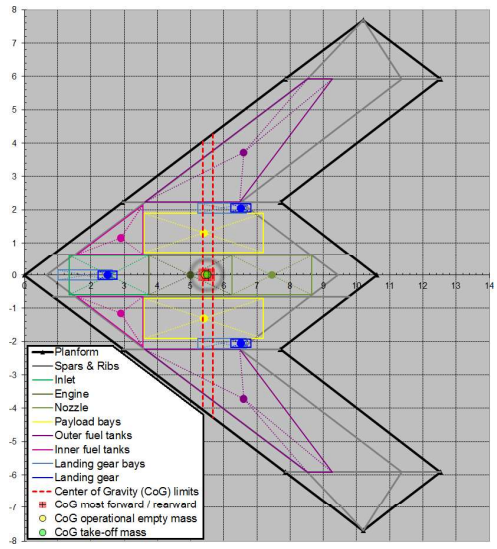
After take-off, the mission continues with climbing to high cruising altitude which is maintained until reaching the proximity of the target area. The cruising altitude is 11 thousand meters (above sea level), and the cruising Mach number is 0.8 (-). The maximum angle of attack requirement is reduced to 14 degrees. However, the roll rate requirement is increased to 90 degrees per 1.7 seconds (52.94 deg/s). The rest is unchanged.

Following the cruise, the aircraft descends to the target area keeping the Mach number constant (M=0.8). The aircraft penetrates the airspace following a slightly declining trajectory towards the target. The requirements during engagement are defined by design point combat at low altitude in Table 4.3. Compared to cruise conditions, the roll rate is further increased to 130 degrees per second. Shall the aircraft have any excessive power, it should further accelerate towards Mach 0.9, as illustrated in fig. 4.4 (despite the Mach requirement).

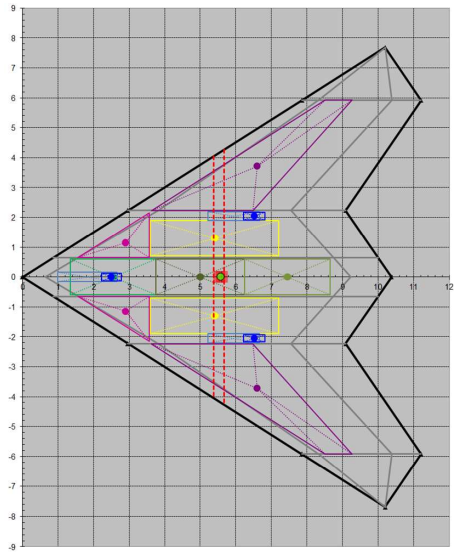
After striking the target the aircraft makes a turn and withdraws following the same trajectory in reversed order. The mission ends with the landing design point. Similarly to take-off, landing is considered at 0 meters

Table 4.2: MULDICON reference values (reference points measured from the aircraft's nose) [58]

Parameter	Abbreviation	Unit	Value
Reference area	S_{ref}	[m^2]	77.8
Reference length – pitching moment coefficient	MAC; l_{ref}	[m]	6.0
Reference length – rolling and yawing moment coefficient	Halfspan; s	[m]	7.69
Moment reference point x	MRP$_x$; x_{ref}	[m]	6.0
Moment reference point y	MRP$_y$; y_{ref}	[m]	0.0
Moment reference point z	MRP$_z$; z_{ref}	[m]	0.0



(a) DLR-F19 / SACCON subsystem configuration [58]



(b) SACCON configuration adapted to the MULDICON planform [58]

Figure 4.2: Overview of the SACCON & MULDICON subsystem configuration. Unlike the figures, for calculation purposes the CG is assumed to coincide with the AC defined in Table 4.2 as the moment reference point.

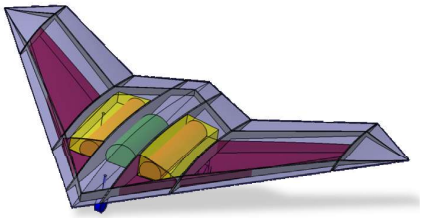


Figure 4.3: MULDICON preliminary CAD model. The current dummy design incorporates the airframe (**gray**), the landing gears (**blue**), the fuel tanks (**maroon**), the engine (**green**) and the bomb bays (**yellow**) [58].

of altitude (above sea level), but at a Mach number of 0.4. The angle of attack requirement is undecided, it may be between 14 and 20 degrees. The rate requirements are the same as for take-off.

Table 4.3: MULDICON design requirements [58]. Rates should be read as *change in degrees / given time.*

		Design points			
Requirement	Unit	Cruise	Take-off	Landing	Combat at low altitude
Mach number	[-]	0.8	0.2	0.4	0.8
Altitude	[m]	11,000	0.0	0.0	0.0
Roll rate	[deg /sec]	90/1.7	30/1.1	90/1.7	130.0/1
Pitch rate	[deg /sec]	20/1.0	20/1.0	20/1.0	20/1.0
Yaw rate	[deg /sec]	15/1.0	15/1.0	15/1.0	15/1.0
AoA (max)	[deg]	14	20	14 – 20	14

4.1.1. FLIGHT CONDITIONS

Investigated flight conditions are derived from the mission profile of the aircraft. For the sake of simplicity, relatively simple cases are sought. Therefore, the current study is limited to two-dimensional planar motions

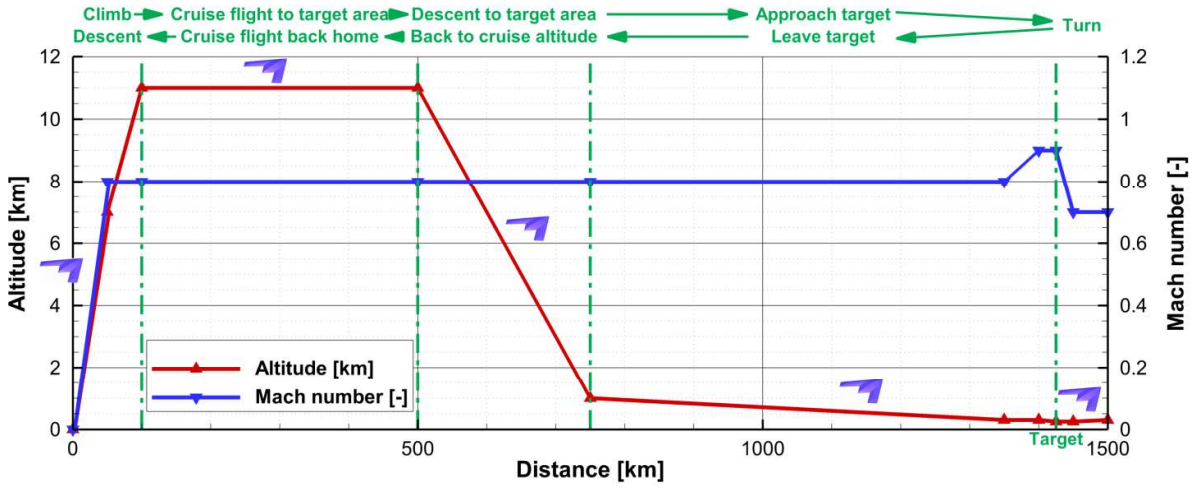


Figure 4.4: MULDICON design mission [58].

in the vertical plane. Concurrently, in the current project the take-off design point from Table 4.3 is considered only. The corresponding requirements allow to investigate both linear and nonlinear phenomena at minimum costs compared to other design points. The free-stream Mach number and altitude are taken as constants. Relevant properties are summarized in Table 4.4.

Table 4.4: Flight and ambient conditions

Variable	Symbol	Unit	Value
Mach number	M	[-]	0.2
Altitude	h	[m]	0.0
Density	ρ	[$\mathrm{kgm^{-3}}$]	1.225
Pressure	p	[Pa]	101325
Temperature	T	[K]	288.15
Speed of sound	a	[$\mathrm{ms^{-1}}$]	340.3
Dynamic viscosity	μ	[Pas]	1.79E-05
Velocity	V_∞	[$\mathrm{ms^{-1}}$]	68.06
Dynamic pressure	q	[Pa]	2837.17
Reynolds number	Re_x	[-]	2.80E+07

4.2. MANEUVER DESCRIPTION: DEFINING MOTION TRAJECTORIES

In order to prescribe motions or maneuvers for both CFD simulations and reduced-order modelling, a simple description of aircraft dynamics is considered. In the current model only the aircraft attitudes, linear and angular velocities and/or accelerations are of interest. It is not intended to create a complete flight dynamics model. The purpose of the discussed model is to generate maneuver trajectories that are performed in the simulations.

4.2.1. REFERENCE FRAMES

For the problems considered a localized model is of primary interest without trans-global navigation. Investigated maneuvers and motions are relatively short. That permits the application of simpler models [62]. First flight above flat earth is assumed. A reference horizontal plane is defined at constant altitude (at sea level), in accordance with the fixed flight conditions from Section 4.1.1. The attitude of the aircraft is given relatively to the horizontal plane at all times. The horizontal plane is defined by $(o_E x_E y_E)$ from fig. 4.5 and is parallel to the corresponding conventional earth axes, illustrated in fig. 4.5. Its $o_E x_E$ axis points to the arbitrary flight

direction of the aircraft, $o_E z_E$ points vertically down (along the gravity vector) and $o_E y_E$ points starboard. The three axis form a right-handed orthogonal axis system, called datum-path earth axes [62]. The origin o_E is placed at the origin of the aircraft body axes, defined as the aircraft reference point. The aircraft reference point is hypothesized to be both the moment reference point and the centre of gravity as mentioned earlier.

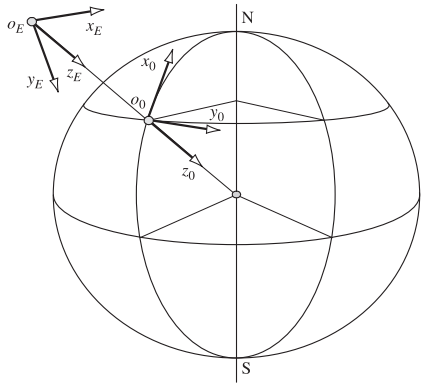


Figure 4.5: Earth reference frames. Axis system $(o_0 x_0 y_0 z_0)$ illustrates the conventional earth axes and $(o_E x_E y_E z_E)$ illustrates the datum-path earth axes [62].

The body axis system $(ox_b y_b z_b)$ is defined as customary in flight dynamics. So $(ox_b z_b)$ is the plane of symmetry of the aircraft with ox_b pointing from tail to nose and oz_b pointing downwards. The oy_b axis points starboard. It is convenient to define a third axis system $(ox_w y_w z_w)$ called the wind axis system. The wind axis system is oriented such that the ox_w axis points in the direction of the total velocity vector. Cook [62] defines the wind axes correspondingly to a steady or undisturbed state of the aircraft. However, the current study considers dynamic maneuvers rather than disturbances, therefore the orientation of the wind axes now is a function of time and they are always aligned such that ox_w matches the direction of the current total velocity. Still, the wind axes are just a particular version of the body axes, rotated about oy_b through the current incidence angle (angle of attack). The wind axes are also fixed to the aircraft and their origin coincides with the aircraft reference point. The body and wind axis systems are illustrated in fig. 4.6.

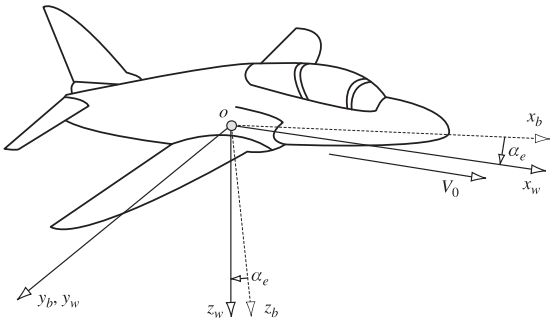


Figure 4.6: Body and wind axes. Axis system $(ox_b y_b z_b)$ illustrates the body axes and $(ox_w y_w z_w)$ illustrates the wind axes. The wind axes are rotated about oy_b through the incident angle α_e [62].

4.2.2. MOTION VARIABLES

The motion of the aircraft is prescribed by its attitude and position in time. Having the earth frame as reference, the attitude is commanded by the body incidence or angle of attack $\alpha(t)$ and by the aircraft pitch attitude $\theta(t)$. The maneuvers are initiated from steady non-accelerating conditions defined by the incidence and attitude values of α_0 and θ_0. Then, motions are resolved into components with respect to the datum-path earth axis system. The positive sense of the variables is determined by the right-hand rule. Given the scope of

the project, i.e. symmetric flight, out of plane components are neglected. The remaining variables are shown in fig. 4.7. The aircraft has an absolute flight velocity vector V_∞ and a pitch rate of q. The velocity is resolved into the components of U and W along x_E and z_E using the flight path angle γ. The orientation of the aircraft relatively to the earth axes is defined by the pitch angle θ. The corresponding incidence angle is denoted by α.

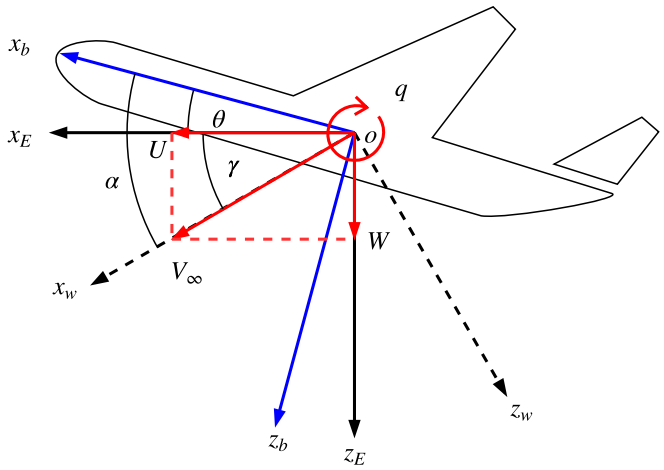


Figure 4.7: Motion variables in the flight dynamic reference frames. Positive senses of the variables are determined by the right-hand rule.

The motions and maneuvers are defined such that the total velocity and Mach number remain constant at all times. Then, velocity components can be expressed at every time instant of the motion using the angular relations among the reference frames. Since only planar motions are considered, relations can be expressed as rotations around the y axes. Solving equations in vector format, the general transformation matrix of the rotation can be written as [63]:

$$\mathbf{T}_y(k) = [k]_y = \begin{bmatrix} \cos k & 0 & -\sin k \\ 0 & 1 & 0 \\ \sin k & 0 & \cos k \end{bmatrix} \tag{4.1}$$

where k is the angle of rotation. Then, from fig. 4.7, recalling the right-hand rule, the flight path angle γ can be expressed as:

$$\gamma(t) = \theta(t) - \alpha(t) \tag{4.2}$$

Using the vector notation:

$$\mathbf{r} = (x,\ y,\ z) \begin{Bmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{Bmatrix} = (x,\ y,\ z)\ \{\mathbf{E}\} \tag{4.3}$$

where $\{\mathbf{E}\}$ is the short notation of the column of unit vectors, the total flight velocity can be written in the wind axis system as:

$$\mathbf{V}_\infty(t) = \big(U_w(t),\ V_w(t),\ W_w(t)\big)\ \{\mathbf{E}_w\} = \big(V_\infty(t)\big),\ 0,\ 0\big)\ \{\mathbf{E}_w\} \tag{4.4}$$

which can easily be converted to the earth axes using Equation (4.1):

$$\mathbf{V}_\infty(t) = \big(V_\infty(t),\ 0,\ 0\big)\ \{\mathbf{E}_w\} = \big(V_\infty(t)\big),\ 0,\ 0\big)\ [\gamma(t)]\ \{\mathbf{E}_E\} \tag{4.5}$$

Having the velocity components resolved and knowing the attitudes and the length of the time steps, additional variables of interest, e.g. position, angular rates and linear or angular accelerations can be derived by numerical integration or differentiation. The main importance of the current description is that it provides the necessary motion variables for CFD simulations.

4.3. AERODYNAMIC MODELS: AIRCRAFT REPRESENTATIONS AND COMPUTATIONAL DOMAINS

Next, aerodynamic models used for full-order CFD and reduced-order neural network computations are introduced. Aircraft responses in terms of aerodynamic loads are derived from these computations. The CFD results provide the samples and reference data to which the neural network predictions are compared to and from which the reduced-order representation of the system is identified.

4.3.1. FULL-ORDER MODEL FOR COMPUTATIONAL FLUID DYNAMICS SIMULATIONS

All CFD simulations were carried out using the in-house flow solver ENSOLV of the Netherlands Aerospace Centre (NLR), an advanced code capable of solving three-dimensional, time-dependent flow around complex configurations. Neither CFD model selection, validation and verification, nor mesh generation was part of the research. The applied settings were defined and selected prior to the current study. In order to remain consistent with parallel projects and for the sake of simplicity the complete setup was copied. Only inputs defining dynamic motions were newly generated.

The maneuvers are implemented as time-accurate simulations of unsteady flows around the aircraft. The flow equations are solved in a basic inertial reference frame $\{\mathbf{E}_I\}$. Motions of the configuration are performed by rigid motions of the grid. The attitude and position of the aircraft is represented by another, body fixed reference frame, relative to the inertial reference frame. The aerodynamic loads are resolved in the body axes.

The frames are similar to the ones used for flight dynamics in Section 4.2.1. They are both right-handed orthogonal axis systems, but their principal directions are defined as customary in aerodynamics. The x and z axes point in opposite directions compared to the flight dynamics frames, so from head to tail and upwards, respectively. The basic inertial reference frame is an equivalent of the undisturbed datum-path earth axes and typically coincide with the body axes at $t = 0$. The aerodynamic body frame $\{\mathbf{E}_{b,A}\}$ has its origin fixed to the aircraft reference point, just as before. The aerodynamic frames are illustrated in fig. 4.8. In current the figure, the resultant aerodynamic loads are included as well. The normal force is denoted as N and the axial force as A. The pitching moment is denoted by M.

To solve the problems the full Reynolds-Averaged Navier-Stokes equations are discretized following a cell-centered finite-volume method. The computations are performed using an adaptation of the two-equation Shear Stress Transport (SST) model [64]. Each time-step has 50 subiterations. The simulations proceed in fixed physical time-steps Δs with a value of $\Delta s = 0.05\,(-)$. Its relations to simulation time is given as:

$$\Delta s = \Delta t \, \frac{V_\infty}{l_{\text{ref}}} \tag{4.6}$$

where Δt is the time-step of the simulation in seconds, V_∞ is the free stream velocity and l_{ref} is the reference length. The reference values are taken from Table 4.2.

The simulations are solved using only the half-span geometry of the MULDICON model, since the performed motions are symmetrical. A multi-block structured grid, generated prior to the current project for similar problems, is implemented. An O-type mesh envelops the body. Overall, the grid contains 12 million cells. The mesh around the wing has 112 cells along the chord and 128 along the span. The trailing – and wingtip edges are modelled with a finite thickness, having 32 cells across their height. On average, the $y+$ value of the first cells is 0.75 at $\alpha = 0\,(\text{deg})$ [65]. The mesh around the aircraft is illustrated in fig. 4.9.

For each time-step the resulting wing pressure distributions in terms of non-dimensional pressure coefficients (C_p) and the corresponding integral aerodynamic loads, in terms of normal and axial force coefficients (C_N, C_A) and pitching moment coefficient (C_M), are saved. The coefficients are expressed as:

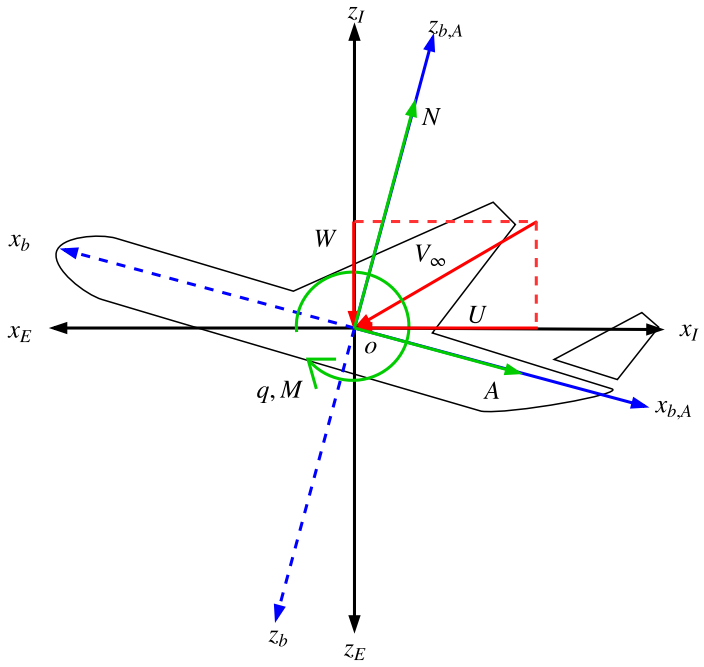


Figure 4.8: Motion variables in the aerodynamic reference frames. Positive senses of the variables are determined by the right-hand rule.

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty V_\infty^2}$$

$$C_k = \frac{k}{\frac{1}{2}\rho_\infty V_\infty^2 S_{\text{ref}}}, \text{where} \quad k = N, A \tag{4.7}$$

$$C_M = \frac{M}{\frac{1}{2}\rho_\infty V_\infty^2 S_{\text{ref}} l_{\text{ref}}}$$

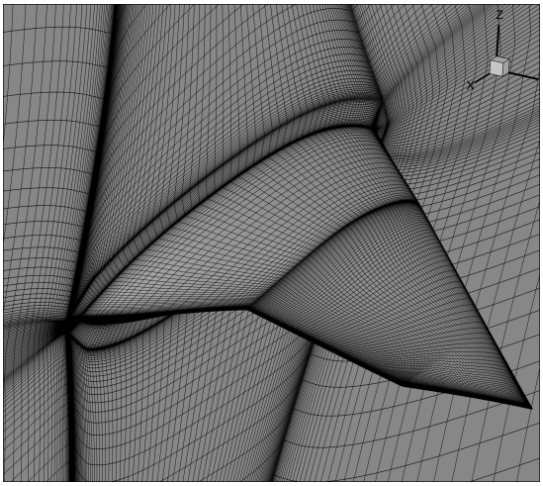
where ρ_∞ and V_∞ are the free-stream density and velocity, in respective order, p is the static pressure at the grid points and p_∞ is the free stream pressure. The denominator S_r is the reference area and L_r is the reference length from Table 4.2. The terms k denotes the axial and normal forces along the x and z axes, respectively.

4.3.2. REDUCED-ORDER AERODYNAMIC MODEL FOR NEURAL NETWORK COMPUTATIONS

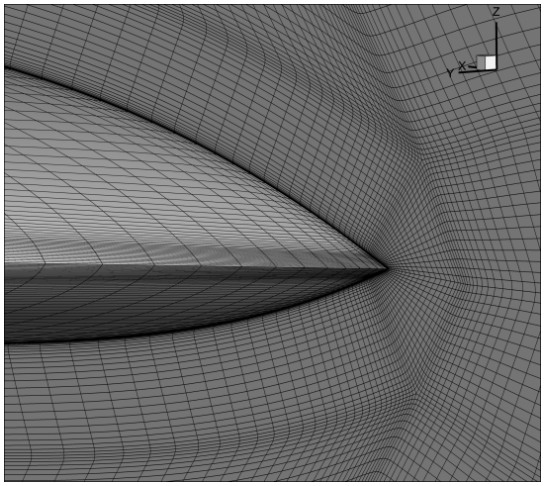
For surrogate modelling a reduced-order representation of the complete CFD domain is established. The representation serves as the aerodynamic model or framework on which the neural networks operate.

Flow-wise wing surface pressure distributions are of interest, hence the domain around the body is completely neglected. Only the grid of the wing is considered. The grid consists of separate zones or panels, represented by mesh vertices. In CFD simulations pressure coefficients are computed in those vertices. The original face topology of the wing is shown in fig. 4.10.

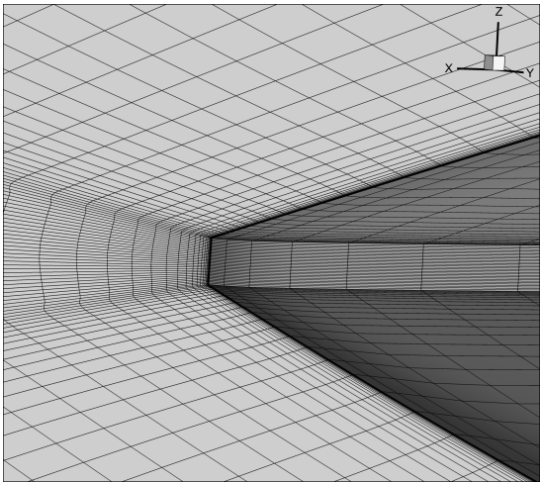
For neural network computations it is convenient to have data stored in a matrix format. The nature of the problem implies ordering the pressure data into series of two-dimensional matrices where each matrix



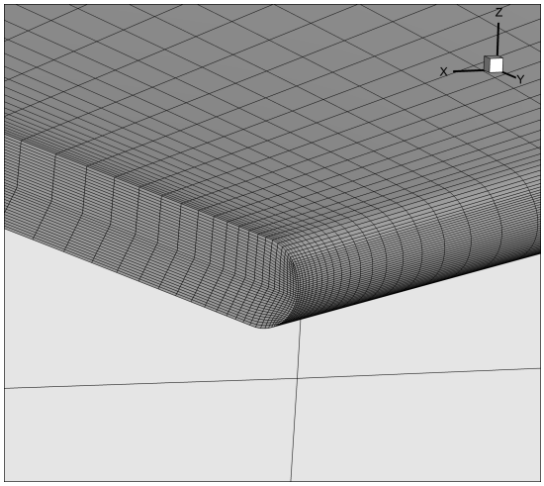
(a) Rear view



(b) Leading edge apex region



(c) Trailing edge at mid-plane



(d) Wing tip region at leading edge

Figure 4.9: Illustration of half-span MULDICON grid on the mid-plane and wing surfaces used for ENSOLV computations [65]

corresponds to a certain time instant. Then, only the organization of data within the matrices is to be decided. From fig. 4.10, the paneling already offers a clear and simple layout. However, there are a few things to consider.

First, values of both upper and lower surfaces have to be stored in a single matrix. As a design choice, it is decided to unfold and flatten the data along the leading edge, so lower surface entries occupy the upper half of the matrix and the upper surface occupies the lower half. The upper left corner of the matrix starts from the trailing edge at centerline. The columns go from centerline to starboard in left-to-right direction and the rows go from trailing edge to leading edge in up-down direction, until the middle rows. There upper and lower surface leading edge nodes are present. As the first row of the upper surface is reached, the ordering of the rows is flipped. So, the rows of the upper surface go from the leading edge to the trailing edge in up-down direction. The ordering of the columns remains unchanged. The lower right corner of the matrix corresponds to the trailing edge of the upper surface at the wingtip.

Second, adjacent zones share bordering vertices, meaning that vertices along zone boundaries appear twice in the data. Being completely unnecessary, and for the sake of simplicity, the duplicate rows and columns are removed except for the first rows of upper and lower surface leading edges, along which the surfaces are joined.

Third, it is assumed that surfaces closing the trailing edge – and wingtip gaps have little contribution to the overall flow behaviour and aerodynamic loads. Therefore, those are neglected as well. The resulting data structure is illustrated in Section 4.3.2.

The outlined setup encompass 29 154 (226 × 129) entries for a single time step. First versions of the model

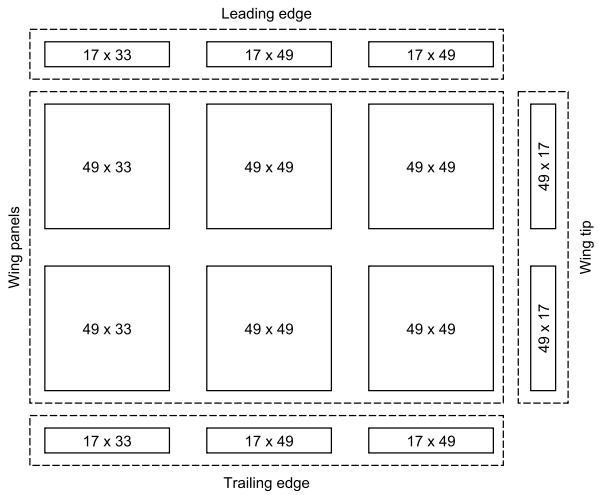


Figure 4.10: Organization of surface pressure data in the mesh domains. Representing the half side of the half wing (e.g. upper surfaces only). The opposite side has an equivalent structure. The rectangles (**solid**) illustrate separate domains on the surface. The borders (**dashed**) group the domains by their position on the wing. The first and second numbers in the domains represent the rows and columns of data respectively. The data contains the flow variables calculated at the cornerpoints of the cells. The neighbouring edges of two domains represent exactly the same points, thus having the exact same values.

were trialed with this setup. To further boost neural network model generation and evaluation the domain is further reduced by removing every second row and column, starting from leading edge centerline nodes of upper and lower surfaces separately. Halving the domain yields 114×65 nodes (7410 elements) compared to which the original CFD domain has 1600 times more vertices. Halving the nodes allowed to optimize networks on a daily basis. At the same time, it did not have any major effects on prediction accuracy.

Flow behavior is identified in these reduced-order representations. Corresponding integral loads predicted by the ROM are also derived from the reduced domain.

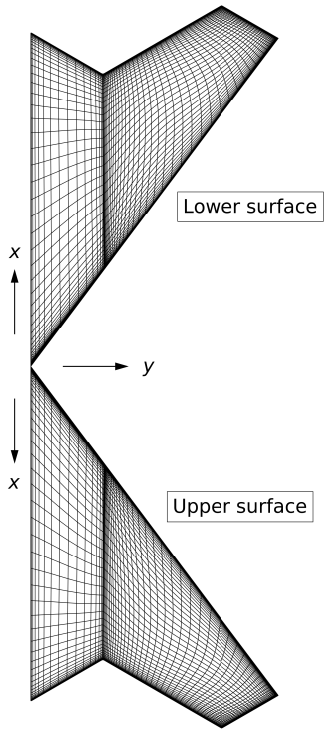
4.4. EXPERIMENT DESIGN: TRAINING MANEUVER FOR SYSTEM IDENTIFICATION

Identification of nonlinear aerodynamic systems is challenging because a wide spectrum of relevant flow phenomena has to be excited. That requires excitations that cover the regressor space sufficiently well. Concurrently, full-order CFD simulations used for generating data are expensive. Hence, to exploit the cost saving offered by the surrogate model, simulations used for system identification must not be too computationally demanding [7, 66].

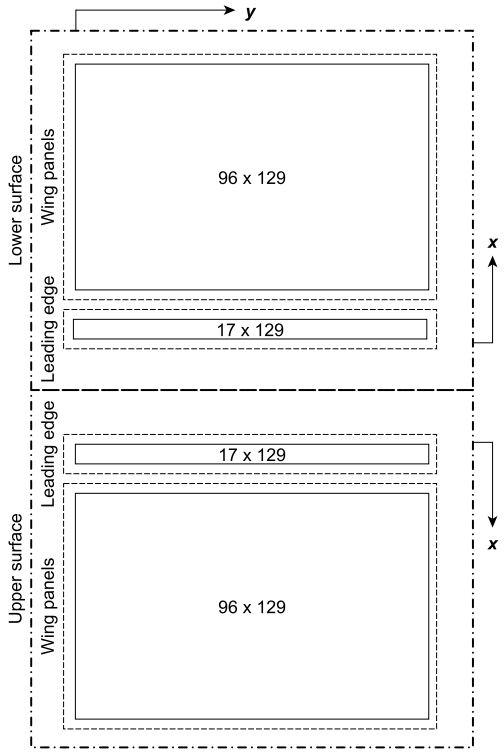
Numerous identification methods are given in the corresponding literature. Inputs traditionally used for system identification include doublets, impulses (stick raps), multi-steps, and frequency sweeps. Recently, a time-efficient input design technique has been developed using Schroeder-sweeps [67] as input signals. These signals are phase-shifted sums of sinusoids, optimized for minimum input amplitudes and wide-band frequency content. Schroeder sweeps are generally superior for dynamic model identification when compared against conventional techniques, e.g. frequency – and amplitude sweep. Schroeder sweeps provide more universal coverage of regressor spaces while providing average performance over broad ranges of frequencies. The coverage of the regressor space was shown to be crucial for capturing a wide spectrum of relevant flow phenomena [66, 68]. By utilizing these features of Schroeder sweeps it is possible to define a single maneuver from which the dynamic behaviour of the aircraft can be derived. Therefore, in the current study Schroeder-sweeps are applied to generate the training maneuver from which flow behaviour is inferred. In the following the methodology of constructing optimized Schroeder-sweeps will be discussed.

4.4.1. MULTISINE INPUT DESIGN

The general idea is to excite aircraft dynamics by perturbing all inputs simultaneously. Usually, the excitations are applied to the control surface deflections. However, since the current model lacks control surfaces, in this



(a) Flattened surface meshes. Lower surface on top facing in opposite direction.



(b) Matrix format of flattened surface meshes. Lower and upper surfaces are joint along the leading edge.

Figure 4.11: Representation of aircraft surface mesh in matrix format .

study the perturbations are applied directly to the motion variables, i.e. to the incidence angle α and to the pitch angle θ. The amplitude of the excitations will then define the angles, while their frequencies will define the rates of change of the angles.

Each perturbation $\mathbf{u}_j$ of the j^{th} input is defined as a Schroeder sweep, which is a sum of harmonic sinusoids with individual phase shifts [68]. Then, the signal $\mathbf{u}_j$ is given as

$$\mathbf{u}_j = \sum_{k=1}^{N} A_k \cos\left(2\pi f_k t + \Phi_k\right) \tag{4.8}$$

where N is the total number of components (and so the number of related frequencies), A_k is the amplitude of the k^{th} sinusoidal component, f_k is the frequency of the k^{th} component and Φ_k is the phase shift of that component. The variable t denotes discrete time instances at which the signal is evaluated. Obviously, in this case, the time instances are identical to the time-steps of the CFD simulation. So, $\mathbf{u}_j$ is the vector in which each element corresponds to a certain time instant in the simulation [68].

The frequencies of the signal are defined as a sequence from the lowest observable frequency resolution to the desired maximum in steps of the lowest resolution. The lowest frequency is the inverse of the signal time length given as $f_{min} = \frac{1}{T}$, where T is the duration of the maneuver [68].

Next, the amplitudes of the different harmonics are to be defined. In order to achieve uniform power distribution among all frequency components, the A_k amplitudes in Equation (4.8) are defined as [68]:

$$A_k = \frac{A}{\sqrt{N}} \quad \forall k \tag{4.9}$$

where N is the total number of components. In this formulation $1/\sqrt{N}$ can be regarded as the relative power p_k of a single component ($\sum_{k=1}^{N} p_k = 1$) [67]. Then the problem of selecting the amplitudes simplifies to selecting a single value [68]. If the excitations are to be applied around a certain magnitude other than 0, a

nominal value shall be added to the sum. Then Equation (4.8) can be rewritten as:

$$\mathbf{u}_j = A_0 + \sum_{k=1}^{N} \frac{A}{\sqrt{N}} \cos\left(2\pi f_k t + \Phi_k\right) \tag{4.10}$$

Next, the phase shifts of the harmonic components are defined. To do so, first an important property of the signals is introduced, called the peak factor (PF). The peak factor is defined as:

$$\mathrm{PF}(\mathbf{u}_j) = \frac{\left(\max(\mathbf{u}_j) - \min(\mathbf{u}_j)\right)/2}{\mathrm{rms}(\mathbf{u}_j)} \tag{4.11}$$

where $rms(\mathbf{u}_j)$ is the root mean square of the elements in the vector $\mathbf{u}_j$. A single component from Equation (4.8) would have a peak factor of $\sqrt{2}$. Compared to that, a relative measure can be defined for the complete sum of the harmonics, called the relative peak factor (RPF), defined as:

$$\mathrm{RPF}(\mathbf{u}_j) = \frac{\mathrm{PF}(\mathbf{u}_j)}{\sqrt{2}} \tag{4.12}$$

The relative peak factor is a measure of efficiency. A single sinusoidal has an *RPF* of 1. Low peak factors are preferable, because they do not drive the "dynamic system too far away from the reference condition" [68]. In order to achieve minimal *RPF*, adequate phase shifts have to be searched. To do so, their values are initialized according to the method of Schroeder [67], that is:

$$\Phi_k = \Phi_1 - 2\pi \sum_{l=1}^{k-1} (k-l)\, p_l \quad k = 1, 2, \cdots, N \tag{4.13}$$

where p_l is the power of the l^{th} harmonic, i.e. $A/\sqrt{N}$. After generating the signal with the initial Φ_k values, the phase shifts of the components can be optimized so the summed signal yields minimum *RPF*.

A drawback of the Schroeder sweep is that the sweeps start from varying magnitudes because of summed harmonics [5, 68]. A way to tackle that is to introduce a time shift in Equation (4.8) and Equation (4.10) [68]. Then the equations can be rewritten as:

$$\mathbf{u}_j = A_0 + \sum_{k=1}^{N} \frac{A}{\sqrt{N}} \cos\left(2\pi f_k (t + t_0) + \Phi_k\right) \tag{4.14}$$

where t_0 is time offset applied to all components. In effect t_0 shifts the signal along the time axis such that the first element of the $\mathbf{u}_j$ vector will coincide with the desired value. Since the signal components are "harmonics of the base frequency with period T", the final value will also be the same. Downside of the shift is that the frequencies of the harmonics will change, so the phase and time shifts have to be iterated in a loop to achieve both minimal RPF and desired starting values [68]. For both inputs the initial values were set to be 0 degree.

Finally, in order to enhance regressor space coverage and to minimize correlations among the input signals, a phase offset is also added to the equation:

$$\mathbf{u}_j = A_0 + \sum_{k=1}^{N} \frac{A}{\sqrt{N}} \cos\left(2\pi f_k (t + t_0) + (\Phi_k + \Phi_0)\right) \tag{4.15}$$

where Φ_0 is an arbitrary constant but different for each $\mathbf{u}_j$ input.

The complete procedure of generating the input signals can be summarized in the following steps [68]:

1. Select the duration of the inputs T. Derive the smallest frequency resolution.
2. Select the frequency band $[f_{min}, f_{max}]$ of the signal and partition it into N number of discrete frequencies.
3. Generate the initial signal $\mathbf{u}_j$ using the initial phase shifts Φ_k.
4. Optimize the phase shifts so that minimum *RPF* is achieved.

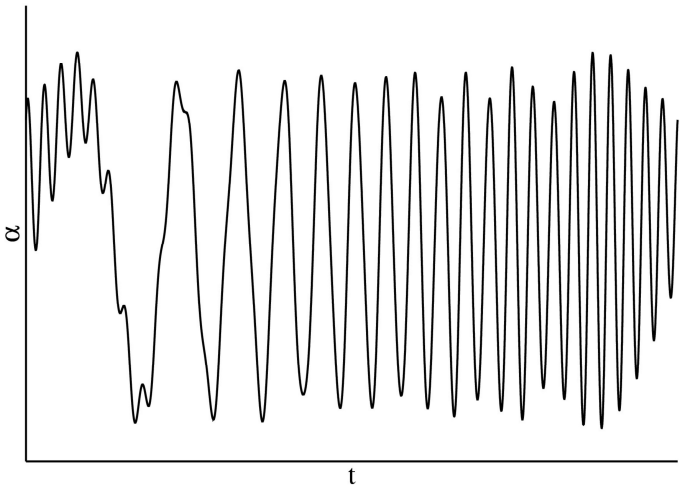


Figure 4.12: Angle of attack as a function of time as defined by a Schroeder sweep [66]

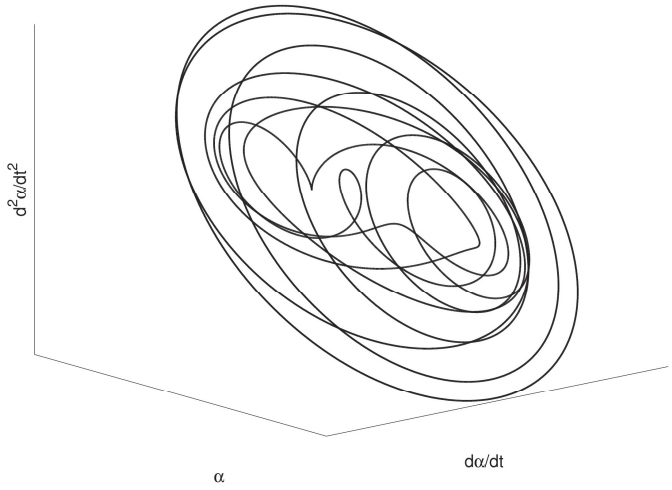


Figure 4.13: Regressor space coverage of the Schroeder sweep example [66]

5. Search for adequate time offset so that desired boundary conditions are met at first and final elements of $\mathbf{u}_j$
6. Return to step 4 and repeat the final steps until solutions converge. Usually 50 iterations were adequate to reach optimized solutions.

An example of the Schroeder sweep from Jirasek *et al.* [66] defining an angle of attack excitation is provided in fig. 4.12. The regressor space coverage of the signal in terms of α, $\dot{\alpha}$ and $\ddot{\alpha}$ is illustrated in fig. 4.13.

4.5. AERODYNAMIC SURROGATE MODEL

The aerodynamic surrogate incorporates the neural networks that are intended to replace the CFD solver. The developed surrogate model consists of three main components. First, it adopts the reduced-order aerodynamic model from Section 4.3.2. Second, the model involves a primary network, an encoding-decoding architecture that predicts pressure distributions at the retained vertices of the reduced-order aero model. Finally, a secondary encoding network computes the integral aerodynamic loads based on the pressure distributions of the primary net. The networks are connected in series. The model processes one instance or time step at a time. Complete maneuvers can be simulated by iterating over all time steps.

The reason behind having two networks instead of one is that this setup allows for separating the different tasks reducing the complexity of the models. Concurrently, separate architectures can be tailored specifically for a single purpose.

Now one may argue, that the integral loads could simply be computed from the pressure distributions, hence there is no need for the second network. However, there is an important factor that necessitates its usage. The aerodynamic forces and moments originate from two sources, namely the pressure and the shear stress distributions. Considering the body reference frame, the normal forces and the pitching moments are usually dominated by pressure forces (at least for the given flight conditions). Hence, they can probably be computed with small errors solely from the predicted pressure distributions. Nonetheless, shear stress can have significant contribution to the axial force (coefficient). Consequently, the pressure distributions on their own are insufficient to accurately predict the axial forces [69]. Hence a mapping is required for the axial force component. That motivates the usage of the secondary network. Still, it would not be necessary to include the normal force coefficient and the pitching moment coefficient. At the same, adding two extra outputs to the network should only increase the costs of training. Additional computational costs during testing and simulation should be negligible. Therefore, not including them in the output of the network would actually increase the computational costs, since extra calculations would be required for deriving the remaining terms.

The upcoming sections will elaborate the final architectures that are derived from the candidate network.

4.5.1. PRIMARY NETWORK: SURFACE PRESSURE PREDICTION

A simplified block diagram of the flow predictor is shown in fig. 4.14.

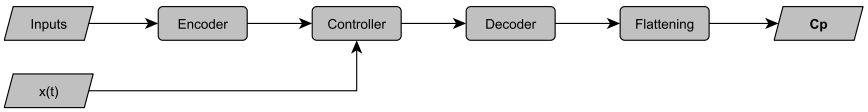


Figure 4.14: Schematics of the flow predictor network.

Although it is not evident from fig. 4.14, the network inherited the U-shaped architecture. The blocks encoder and decoder denote the contracting and expansive paths, respectively. Both the encoder and decoder remained fully convolutional accordingly to the candidate network using the very same D-Res and U-Res blocks. However, the convolution between the two branches got replaced by a newly constructed controller block that employs a fully connected layer, coupling the inputs with the motion variables, denoted by $x(t)$. The flattening block at the end serves the purpose of the final convolution in the candidate network, that is flattening the feature maps into a single channel. CELU remained the global activation function of the network. All activations, except for the sigmoid gates in the gated ResNet blocks, are CELU activations.

An important modification compared to the candidate network (and also to the original U-net) was the removal of the skip connections among the contracting and expansive layers. Skip connections were found to transfer errors which either resulted in oversensitive responses in the output, or caused a catastrophic divergence from which the network could not recover.

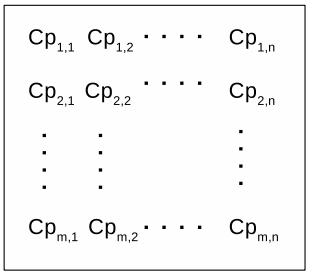
Next, the different components will be discussed in detail.

ENCODER INPUTS

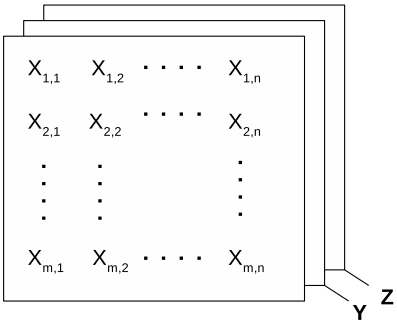
The first implementations of the network followed the paradigm of Faller *et al.* [70]. That is generating responses, i.e. updated pressure coefficients, based on preceding predictions. This setup had the advantage of incorporating history effects in terms of the inputs. The newly generated predictions were fed back via a recurrent connection and used as the next inputs. The network easily learnt the desired task and showed good performance for certain maneuvers. However, this setup also had a reoccurring problem. For slow motions (slowly varying) and for linear motions, where the rates are constant, the network generated somewhat randomly fluctuating responses. Consequently, the solution also randomly diverged from reference values.

The issue was solved by replacing the input tensors with geometric data of the aircraft. Those were the x,y and z coordinates of the grid points on the surface of the aircraft body, at which the pressure coefficients are evaluated. The coordinates are defined in the body reference frame, and their values are constant at all times. Meaning, that the complete encoder becomes a constant as well. It is arguably not an efficient solution, since

the encoder could be replaced by a limited number of scalars. However, the networks were developed in parallel and it was decided to remain consistent with the architectures. Additionally, it is easier to relate the encoder to some data that actually bears a physical meaning instead of using otherwise meaningless constants. Furthermore, this change was implemented effortlessly, since the grid tensor had the same width and height as the pressure (coefficient) tensors, except it had three channels. Since the encoder is fully convolutional this change had minor influence on the computations. From a practical point of view, the only difference is feeding a different input tensor to the encoder. The original pressure tensor **P** and the final geometry tensor **G** are visualized in fig. 4.15.



(a) Structure of the pressure input tensors used in the initial models. The pressure coefficients are organized in a single 2D layer. The values in the tensor correspond to the previous time-step similarly to the method of Faller *et al.* [70]. This input is time dependent.

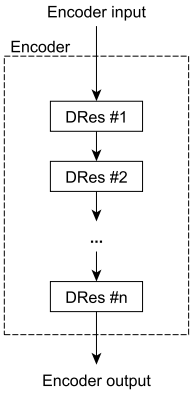


(b) Structure of the grid input tensors used in the finalized architectures. The grid point coordinates are organized in 3 layers. Each separate layer has the same dimension as a pressure tensor. The grid tensor is constant.

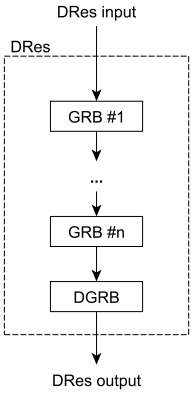
Figure 4.15: Comparison of the pressure (**left**) – and the grid (**right**) input tensors.

ENCODER AND DECODER

The encoder receives the geometry tensor **G** as its input, and its stacked convolutional layers extract features of the geometry. The convolutional layers are down-sampling gated residual blocks (DRes) from [17], which are stacked GRBs where the last block is a down-sampling GRB. The schematics of the encoder and the down-sampling blocks are shown in fig. 4.16.



(a) Layout of the encoder in the flow predictors. The encoder is a series of DRes blocks. The separate DRes blocks are regarded as layers of the encoder



(b) Layout of a DRes block within the encoder. The DRes block is a series of gated residual blocks (GRB). The last residual block in a DRes block down-samples the data (DGRB).

Figure 4.16: Encoder internal structure

The operations of the encoder yields the encoded feature tensor $\mathbf{h}_p^{\text{enc}}(\mathbf{G})$ of the geometry, where subscript p denotes the primary network, is given as:

$$\mathbf{h}_p^{\text{enc}}(\mathbf{G}) = \text{Encode}\Big(\mathbf{G}\Big) = \text{DRes}^l\Big(\mathbf{G}\Big) \tag{4.16}$$

where l is the number of DRes blocks in the encoder, DRes(.) denotes the mapping of a single DRes block and Encode(.) denotes the mapping from $\mathbf{G}$ to the geometry feature tensor $\mathbf{h}^{\text{enc}}$.

An additional requirement towards the encoder is that it shrinks the size of the inputs to sufficiently small dimension. In the middle of the network the controller employs a fully connected layer, so small changes in the dimensions of the encoder's output would increase the size of the fully connected layer exponentially. Consequently, the cost of training would rise too. Therefore, tensor dimensions has to be reduced in order to obtain small arrays of manageable sizes. The size of the output is controlled by the number of DRes blocks in the encoder. Similarly to the candidate network the DRes blocks employ a strided ResNet as their last component. The effect on the encoder on the domain size is illustrated in fig. 4.17.

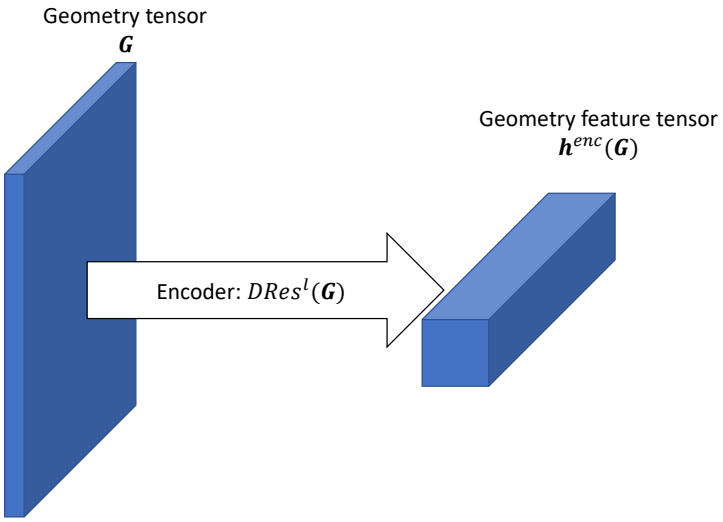


Figure 4.17: Illustration of encoder operations. Geometry features are extracted via stacked DRes blocks.

CONTROLLER: INTRODUCTION OF MOTION VARIABLES

In the middle a controller block is added to the architecture. The schematic of the controller is illustrated in fig. 4.18.

The controller receives both the abstraction of the encoder and the motion variables of the aircraft. The role of the controller is to couple the motion variables with the outputs of the encoder initiating the response of the system for the consecutive time-step. Originally, the motion variables were limited to the AoA, the rate of change of AoA and to the pitch rate. It was later found out that despite the training motions are not optimized for the second derivatives of the angles in any regard, adding them to the motion variables enhanced network accuracy. However as the magnitude of the second derivatives are generally large, their value are normalized by the ratio of the reference length and the free stream velocity, in particular l_{ref}/V_∞. So, the final vector of the control variables is given as

$$\mathbf{x}(t) = \Big(\alpha(t), \dot{\alpha}(t), \hat{\ddot{\alpha}}(t), q(t), \hat{\dot{q}}(t)\Big) \tag{4.17}$$

where the hat over the second derivatives denote the normalized values. In order to combine the two

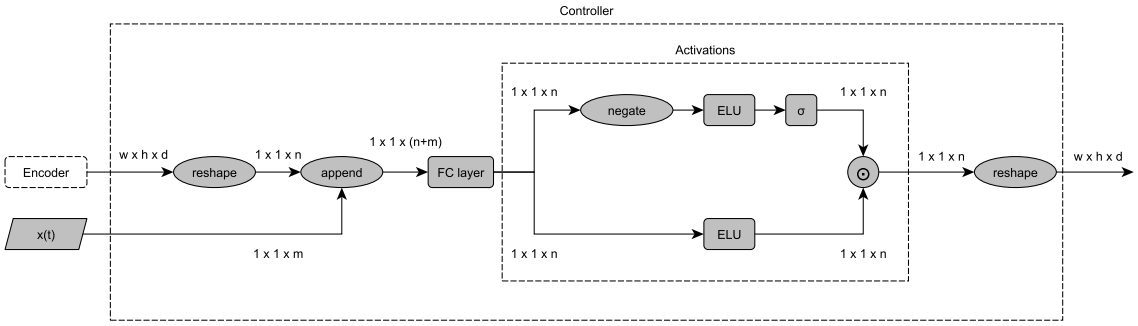


Figure 4.18: Schematic of controller in the flow predictor network. The shapes of the tensors at interim stations are shown in width, height, depth format ($w \times h \times d$). The third dimension of the flattened tensor (**n**) equals the product of the dimensions of the input tensor (**w**·**h**·**d**).

threads, first the output of the encoder is flattened into a 1D vector of $\{h_p^{\text{enc}}\}$. Then, the vector is appended by state variables yielding the extended feature vector:

$$\{h_p^{\text{ext}}\} = \left(\{h_p^{\text{enc}}\}^T \mid \mathbf{x}(t)\right)^T \tag{4.18}$$

This vector is fed through a fully connected layer, where the hidden layer has as many elements as the output of the encoder. Similarly to the rest of the network, CELU activation is applied to the output of the perceptrons. The response of the activations is fed through a sigmoid gate, restoring the number of elements seen in the input of the controller countering the doubling size effect of the activation. Finally, since the output of the residual gate has the same number of elements as the output of the encoder, it is possible to reshape the data to the exact dimensions the encoder produced. Doing so will result the disturbed feature tensor $\mathbf{h}^{\text{dist}}$. So, the mapping of the controller can be formulated as:

$$\mathbf{h}_p^{\text{dist}}\left(\mathbf{G}, \mathbf{x}(t)\right) = \text{Control}\left(\mathbf{h}_p^{\text{enc}}\left(\mathbf{G}\right), \mathbf{x}(t)\right) \tag{4.19}$$

where Control(.) denotes the mapping of the controller including the activations and the sigmoid gate. Reshaping the tensor makes building the decoder convenient, because then the decoder can be constructed by inverting the encoder and the feature maps will regain the width and height of the input tensors. The transformations applied on the tensors are illustrated in fig. 4.19.

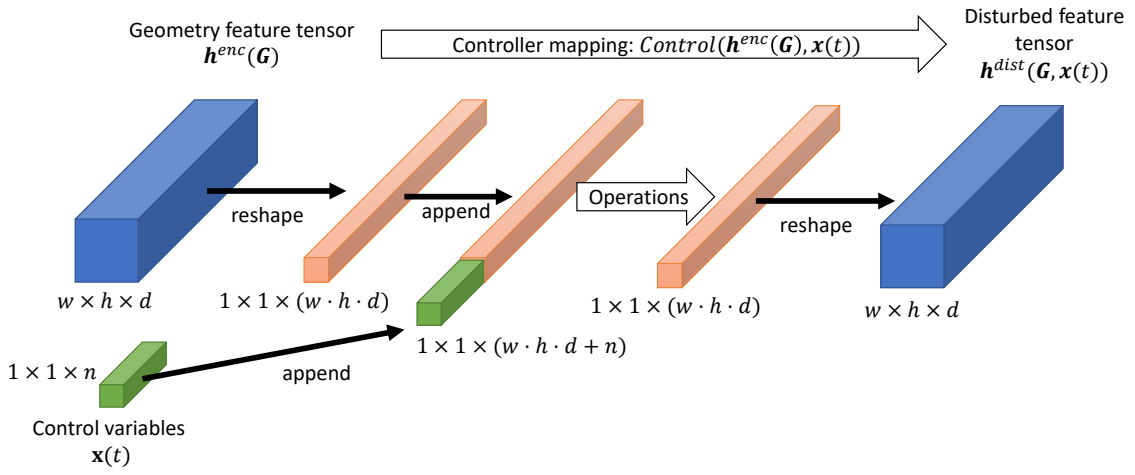


Figure 4.19: Illustration of controller operations. Encoder outputs are flattened into a 1D vector and appended by the control variables. The original tensor dimensions are restored at the end.

DECODER: UP-SAMPLING MODEL FEATURES

Finally, the decoder up-samples the disturbed feature tensors restoring the resolution the encoder inputs and maps the up-sampled feature maps to pressure coefficients at the mesh vertices in the reduced-order aerodynamic model. Finally, the decoding path maps the disturbed feature matrix $\mathbf{h}^{\text{dist}}$ into multi channel decoded feature tensor $\mathbf{h}^{\text{dec}}$ that has the same width and height as the input tensor $\mathbf{G}$. Similarly to the candidate network, the decoder employs stacked up-sampling gated residual (URes) blocks. The blocks start with a transpose convolution up-sampling the received feature maps, followed by common GRBs. The operation can be formulated as:

$$\mathbf{h}_p^{\text{dec}}\big(\mathbf{G},\mathbf{x}(t)\big) = \text{Decode}\Big(\mathbf{h}_p^{\text{dist}}\big(\mathbf{G},\mathbf{x}(t)\big)\Big) = \text{URes}^l\Big(\mathbf{h}_p^{\text{dist}}\big(\mathbf{G},\mathbf{x}(t)\big)\Big) \tag{4.20}$$

where l is the number of URes blocks (equivalent to the number of DRes blocks) in the decoder, URes(.) denotes the mapping of a single URes block and Decode(.) denotes the mapping of the decoder. The network ends with a final convolution layer that keeps the width and height of the tensors unchanged, but collects the feature channels of the decoded feature tensor $\mathbf{h}^{\text{dec}}$ into a single channel. The final convolution directly yields the pressure distribution at the predefined grid locations in the reduced aerodynamic model. The final predicted pressure tensor $\hat{\mathbf{t}}^p$ is given as:

$$\hat{\mathbf{t}}^p\big(\mathbf{G},\mathbf{x}(t)\big) = \text{Flat}\Big(\mathbf{h}^{\text{dec}}\big(\mathbf{G},\mathbf{x}(t)\big)\Big) \tag{4.21}$$

where Flat(.) denotes the flattening operation. The schematics of the decoder are illustrated in fig. 4.20.

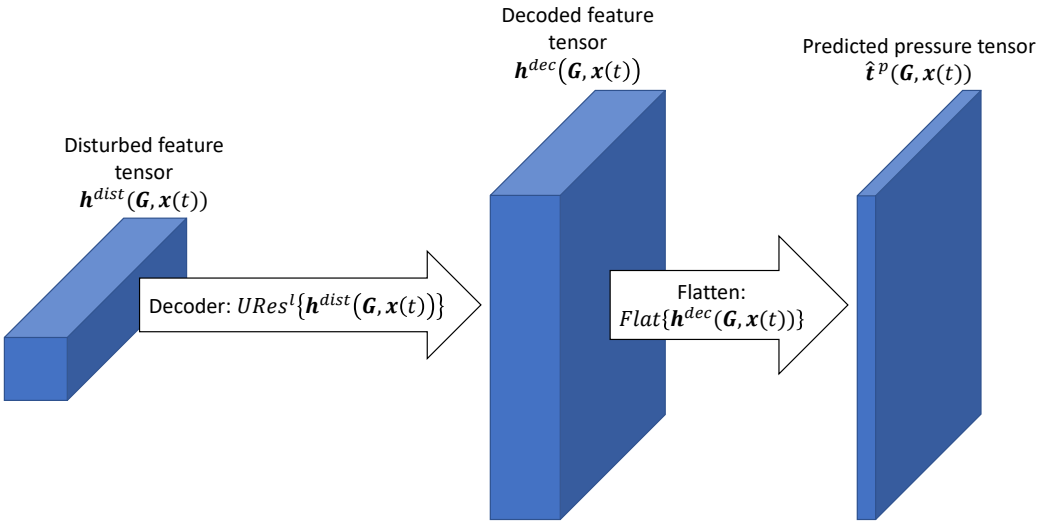


Figure 4.20: Illustration of decoder operations. The decoder up-samples the disturbed feature maps via stacked URes blocks. A final convolution collects the feature maps of the decoder's output to a single channel, yielding the predicted pressure distributions.

4.5.2. SECONDARY NETWORK: INTEGRAL LOAD PREDICTION

Integral aerodynamic loads are derived by a secondary net from the predictions of the primary network. Virtually, the force predictor consists of the first half of the flow predictor.

The secondary network includes basically an identical encoder as seen in Section 4.5.1 and a fully connected block that is also similar to the controller from Section 4.5.1. The encoding path is regarded again as the encoder, whereas the fully connected block now is called the regression block.

The first version of secondary net received only tensors of pressure coefficients. Results showed that the

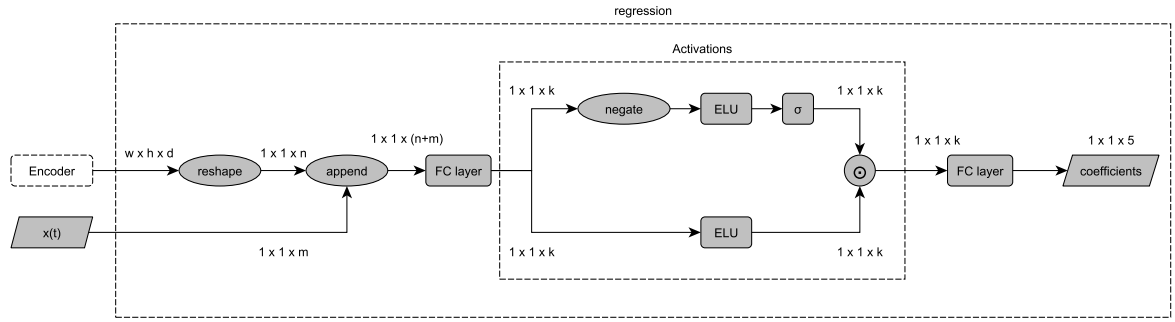


Figure 4.21: Schematic of the regression blocks in the force predictor network. The shapes of the tensors at interim stations are shown in width, height, depth format ($w \times h \times d$). The third dimension of the flattened tensor (**n**) equals the product of the dimensions of the input tensor ($\mathbf{w} \cdot \mathbf{h} \cdot \mathbf{d}$). The depth of the tensor is significantly reduced by the fully connected layers (**FC layer**), so $n >> k$ and $k >> 5$. The ELU blocks denote the Exponential Linear Unit activations. The second fully connected layer directly yields the sought aerodynamic coefficients.

pressure predictions are sufficient to derive the sought integral aerodynamic coefficients. In fact, if the true CFD results are shown to the network, its predictions almost exactly overlap the CFD results. However, the pitching moment coefficient was shown to be sensitive for inaccuracy in pressure distributions. In certain scenarios the inaccuracy of the primary network produced pressure distributions that resulted considerably different pitching moments compared to the CFD predictions. Although the predictions of the secondary net corresponding to the erroneous pressure fields could still be accurate, the predictions of pitching moment might not represent the actual flight conditions. To correct for the errors in the pressure fields, the control variables were also introduced to the secondary net. Similarly to the controller in Section 4.5.1, the encoded features of the pressure fields are coupled with the same control variables.

The encoder of the secondary net basically has the same architecture as the encoder of the primary net. The differences is that now the encoder either receives a pressure distribution $\mathbf{t}^p(t)$ from the CFD simulation or a prediction $\hat{\mathbf{t}}^{\,p}(t)$ from the primary net. Otherwise, the operations of the secondary net can be formulated the same way, and its encoded feature map can be given as:

$$\mathbf{h}_s^{\text{enc}}(\mathbf{X}) = \text{Encode}\Big(\mathbf{X}\Big) = \text{DRes}^l\Big(\mathbf{X}\Big) \quad \text{where: } \mathbf{X} \in [\hat{\mathbf{t}}^{\,p}(t), \mathbf{t}^p(t)] \tag{4.22}$$

where the subscript s denotes the secondary network. The rest is equivalent to Section 4.5.1. In the following only the regression block will be discussed.

REGRESSION: GENERATING THE AERODYNAMIC COEFFICIENTS

The regression block, similarly to the controller, starts with flattening the encoded feature tensor of the encoder to a 1D vector. That is, $\mathbf{h}_s^{\text{enc}}$ is reshaped into $\{h_s^{\text{enc}}\}$. Then, the flattened feature tensor is appended by the control variables from Section 4.5.1 resulting the extended feature vector $\{\text{h}_s^{\text{ext}}\}$ of the secondary network:

$$\{\text{h}_s^{\text{ext}}\} = \left(\{\text{h}_s^{\text{enc}}\}^T \mid \mathbf{x}^T(t)\right)^T \tag{4.23}$$

After, the extended feature vector is fed through two fully connected layer, the first of which is followed by a CELU activation and a sigmoid gate. The two fully connected layers gradually decrease the domain size. The second layer is not activated and it produces the integral loads directly. At the beginning it was trialed to derive the coefficients using only one fully connected layer. However, adding a second layer enhanced accuracy. The operations of the regression block are formulated as:

$$\hat{\mathbf{t}}^{\,C}(\mathbf{X}, \mathbf{x}(t)) = \left(\hat{C}_N(\mathbf{X}, \mathbf{x}(t)), \hat{C}_T(\mathbf{X}, \mathbf{x}(t)), \hat{C}_M(\mathbf{X}, \mathbf{x}(t))\right) = \text{Reg}\Big(\{\text{h}_s^{\text{ext}}\}(\mathbf{X}, \mathbf{x}(t))\Big) \quad \text{where: } \mathbf{X} \in [\hat{\mathbf{t}}^{\,p}(t), \mathbf{t}^p(t)] \tag{4.24}$$

where Reg(.) denotes the operations of the two fully connected layers with the activation and sigmoid gate in between. The term $\hat{\mathbf{t}}^{\,C}$ is the vector of the approximated integral load coefficients. The internal structure of the regression block is illustrated in fig. 4.21.

4.5.3. MODEL OPTIMIZATION: PERFORMANCE MEASURES AND TRAINING SETTINGS

The two networks are trained separately. Doing so promotes assessment of changes in network architectures and also speeds up the optimization of a single case. Both networks are optimized using the Mean Squared Error (MSE) defined as:

$$L(\theta) = \text{MSE}(\hat{\mathbf{t}}^{\,i}, \mathbf{t}; \theta) = \frac{1}{N}\sum_{n=1}^{N} |\hat{\mathbf{t}}^{\,i}(\theta) - \mathbf{t}|^2 \quad \text{where: } i = p, C \tag{4.25}$$

where θ denotes the parameters of the networks subject to optimization, $\hat{\mathbf{t}}$ are the estimators of the true values $\mathbf{t}$ [16, 18]. The networks are trained via MB-GD using the ADAM algorithm. Due to the mini-batches the MSE results are noisy throughout the training procedure. In order to track progress and additional performance merit was defined, the Exponential Moving Average (EMA) [71] of the MSE loss. That is:

$$\text{EMA}_i = \begin{cases} Y_1 & i = 1 \\ \alpha Y_i + (1-\alpha)\text{EMA}_{i-1} & i > 1 \end{cases} \tag{4.26}$$

where Y_i is the value of interest at the i^{th} step, i.e. MSE and α is the decay. The decay was set to be 0.999 (−). It should be emphasized that EMA was used only for overseeing the progress of optimization. Trials were made using EMA as the loss function, however it did not increase the accuracy but made the error to decrease somewhat slower. Therefore, MSE was kept as the global loss function.

4.5.4. NETWORK HYPERPARAMETERS

So far only the general outlines of the network architectures have been reported. For the sake of clarity, the hyperparameters of the different architectures are specified here.

Both networks have 4 layers. Consequently, the primary network employs 4 DRes and 4 URes blocks, whereas the secondary network uses 4 DRes blocks. DRes blocks include a one GRB and one DGRB block. URes blocks are equipped with one transpose convolution and one GRB block. The simple convolutions in the blocks are defined with a kernel size of 3×3 for both networks. Down-sampling convolutions and average poolings use kernels of 2×2 and a stride of $s = 2$. Transpose convolutions in the primary network employ 3×3 kernels and a stride of $s = 2$. All activations, except for the sigmoid gates, are CELUs. The first layer in the encoders are set to have 8 channels in both networks. The hidden layer in the controller and the first hidden layer in the regression block have as many nodes as the number of elements in the encoded feature tensors. The number of nodes in the second hidden layer of the regression block equals the filter size of the encoded feature tensor, i.e. 128. The complete end-to-end model is illustrated in fig. 4.22, where the tensor shapes are denote above each block.

Both networks are trained over 5×10^5 iterations, similarly to [17]. The dataset is divided into training and test sets making up 80% and 20% of the data, respectively. The training set is shuffled and divided into mini-batches of 8 examples. The learning rates are set as 1×10^{-4} (−) for both networks as seen in [17]. These settings were proved to be robust for different architectures tested during network development.

4.6. SUMMARY

The test subject of the study is the MULDICON [58] blended wing body unmanned combat aircraft of NATO STO work-group 251 [57]. All experiments are conducted on this aircraft. The model used in the current thesis only includes the generic shape of the aircraft, i.e. its outer shell. The model lacks any inlets, outlets, mechanization and control surfaces.

The mission profile of the MULDICON aircraft is closely related to the military standard Bomber Low Level Penetration [61]. From the various design points described in the mission profile (Table 4.3), this project is limited to take-off conditions, meaning an altitude of 0 meters and a Mach number of 0.2 (-). Furthermore, the study only investigates two dimensional planar motions in the vertical plane. Relevant design point requirements prescribe an attainable angle of attack of 20 degrees and a pitch rate of 20 degrees per second for the given conditions (Table 4.3). Accordingly, a simple description of motions is considered, relatively to the Earth inertial reference frame [62]. The motions are defined such that the absolute velocity of the airplane remains constant, as defined by the Mach number. The motions are directed by the incidence and pitch angles

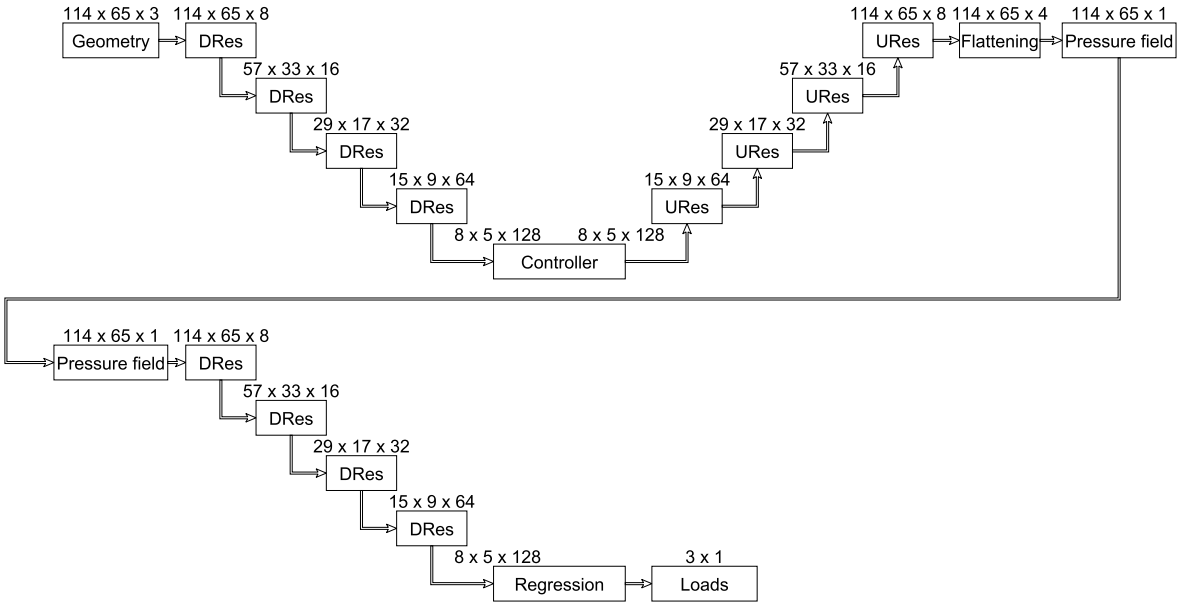


Figure 4.22: Complete end-to-end model architecture. The numbers above the block denote the tensor dimensions in the blocks in width × height × depth format.

of the aircraft. Given the boundary conditions, the remaining motion variables, i.e. position of the aircraft, the velocity components, the Euler angles and the angular rates can be readily calculated at all times.

Two aerodynamic models are considered, used for full- and reduced-order simulations. The first one, regarded as the full-order model, is used for generating reference data. Data is gathered exclusively from CFD simulations. The full-order model constitutes of a multi-block structured grid, enveloping the aircraft body. The grid includes only the right half of the aircraft, and it contains 12 million cells [65]. Maneuver simulations are performed as time-accurate simulation of unsteady flow around the aircraft, using the full-order domain and the full Reynolds-Averaged Navier-Stokes equations employing a modified two-equation Shear Stress Transport turbulence model [65]. Resulting aerodynamic loads are resolved in the aerodynamic body frame, where the x axis points head to tail, the y axis points starboard and the z axis points upwards. The simulations proceed in fixed-length non-dimensional timesteps. For each time step, the non-dimensional pressure coefficients on the aircraft surface along with the normal and axial aerodynamic force coefficients and the pitching moment coefficient are saved.

The second aerodynamic model is derived from the full-order model. It includes a reduced representation of CFD surface mesh, where every second grid point is removed along span and chord directions. Additionally, the grids between upper and lower trailing and wingtip edges are neglected. The final model contains 114 times 65 grid points together on upper and lower surfaces. Similarly to the full-order model, again, the non-dimensional pressure coefficients and integral load coefficients are considered. Model geometry coordinates and corresponding pressure coefficients are organized into matrices. The upper and lower surfaces are unfolded in a single matrix, where the upper left corner corresponds to the trailing edge of the lower surface at the centerline. The middle rows correspond to upper and lower surface leading edges. The bottom right corner of the matrix corresponds to the trailing edge of the upper surface at the wingtip.

To infer system characteristics an efficient system identification technique is considered. The idea is that all inputs of the system are excited simultaneously, in a single maneuver. The input signals are defined as Schroeder sweeps with minimized relative peak factors. A Schroeder sweep is a sum of harmonic sinusoids with individual phase shifts and frequencies. The relative peak factor of the signal is a measure of its efficiency. Low values of peak factors are desired because larger values tend to drive the dynamic system away from reference conditions. The frequencies are defined as sequences between the lowest and highest frequency of interest. The amplitudes of the harmonics are set as a single constant value, that cover the region of interest. Having a constant amplitude defined will result uniform power distribution among all frequencies that are contained in the input signal. If the signals are not centered around zero, a constant nominal value can be defined. However, due to the accumulating harmonics, the signals not necessarily start from that value. After

initializing the variables of the signals, they can be optimized to have minimum relative peak factors and to start from desired nominal values in an iterative manner [68].

The devised aerodynamic surrogate model has three main components. First it includes the reduced-order aerodynamic model. Second it incorporates a primary convolutional network that predicts surface pressure distributions on the reduced model. Third, it includes a secondary convolutional network that derives the integral aerodynamic loads corresponding to the predicted pressure distributions. The primary network inherited its encoding-decoding architecture from the candidate network. Its encoder receives the geometry description of the reduced model and extracts its features, resulting the encoded feature tensors. The encoded features are passed to the controller, which couples them with the control variables via a fully connected layer, yielding the disturbed feature tensors. The control variables consist of the instantaneous values of the angle of attack, its first and second derivative, and the pitch rate, and its first derivative. The second derivative of the angle of attack and the first derivative of the pitch rate are normalized by the ratio of the aircraft reference length and free stream velocity. The decoder of the network up-samples the disturbed feature tensors providing the decoded feature tensors. Having multiple channels, the decoded tensors are convolved into a single a channel by a simple convolutional layer, directly yielding the pressure distribution on the surface in terms of non-dimensional pressure coefficients. The secondary network consists of an encoder and of a regressor block. It has an architecture that is almost equivalent to the first half of the primary network. Its encoder receives the tensor of the predicted pressure distribution from the primary network, and extracts its features, resulting the encoded feature tensor of the secondary network. Then its regressor couples the encoded features with the same control variables that the primary network received via a fully connected layer. The operation yields the disturbed feature tensors of the secondary network. That tensor is then fed through another fully connected layer yielding the predicted integral loads of normal and axial force coefficients and pitching moment coefficient. The devised neural networks employ the gated residual blocks and the concatenated exponential linear unit activations of the candidate network. The encoders of both networks and the decoder of the primary network have 4 layers. Encoding layers halve the feature maps but double the feature channels. In contrast, decoding layers double the size of the feature maps but halve the feature channels. The two networks are connected in series. Executing them will yield the predictions of a single (time) instant. Maneuvers can be simulated by iterating over all timesteps.

The networks of the model are optimized by using the ADAM [56] algorithm. The networks employ the the mean square error as the cost function. During network optimization, the training data is processed in mini-batches of 8 examples. Being the mean square error noisy, due to mini-batch gradient descent [28], an additional measure is used to track training progress. For that purpose the moving exponential average the training loss is computed.

5

SIMULATIONS AND RESULTS

In the current chapter, maneuvers simulated by the surrogate model are introduced and compared to the CFD simulations. The chapter will start with specifying the training maneuver from which system characteristics are inferred. Then, to assess basic model performance first steady simulations are investigated, followed by sets of harmonic excitations. In the end, some arbitrary maneuvers will be visited. Additionally, costs of the model will also be discussed.

5.1. TRAINING: INFERRING SYSTEM DYNAMICS

Before deploying the surrogate model, its parameters have to be optimized. For that purpose, the technique outlined in Section 4.4.1 is applied. To generate the actual input signals, their amplitudes, their frequency ranges and the number of harmonics are decided next.

The basic requirements towards the training maneuver are to cover the range of incidence angles and pitch rates outlined in Section 4.1. That means a range of [0,20] (deg) for the incidence angle. To make sure that the two extremes of the [0,20] (deg) range are covered, the amplitude of the AoA excitations is set as 12.5 degrees around $A_0 = 10$ degrees of angle of attack. With the angular rates being identical for AoA and pitch, it is convenient to have the same amplitudes defined for the pitch angle. However, the pitch angle itself is not of particular interest, hence its perturbations do not have to be shifted and they can be centered around 0 degrees.

The pitch rate requirements are defined less clearly. Table 4.3 only tells that the maximum desired pitch rate is 20 degrees per second. Given the circumstances, the prescribed pitch rate was interpreted as the angular speed of a harmonic motion that covers the [0, 20] degree regime. Then, the pitch rate requirement translates into a frequency requirement, in particular 0.5 (Hz). Eventually, since it was unclear what performance should have been achieved for the rate of change of angle of attack, the same conditions were considered for that case.

Then, the frequencies of the perturbations are defined as sequences from the desired lowest frequency resolution to the desired maximum in steps of the lowest resolution. The lowest frequency is at least the inverse of the signal time-length given as $f_{min} = 1/T$, where T is the duration of the excitation. On one hand, to exploit the cost saving offered by the surrogate model, simulations used for system identification must not be too computationally demanding, and therefore have to be limited in time. On the other hand, it is also desired to cover the domain of quasi-static responses. In his study Greenwell [72] showed, that motions with reduced frequencies of 0.01 (-) can be regarded is quasi-static. The reduced frequency can be computed as:

$$k = f \pi l_{ref} / V_\infty \tag{5.1}$$

where f is the frequency in Hz, l_{ref} is the reference length and V_∞ is the free stream velocity. An initial design choice was made to have a minimum angular speed of 2 (deg/s) which resulted a frequency of 0.04 (Hz), and a reduced frequency of 0.0111 (-) for the given amplitude. The corresponding simulation duration is 25 seconds. Concurrently, with the CFD solver settings defined in Section 4.3.1, 25 seconds translates into 5667

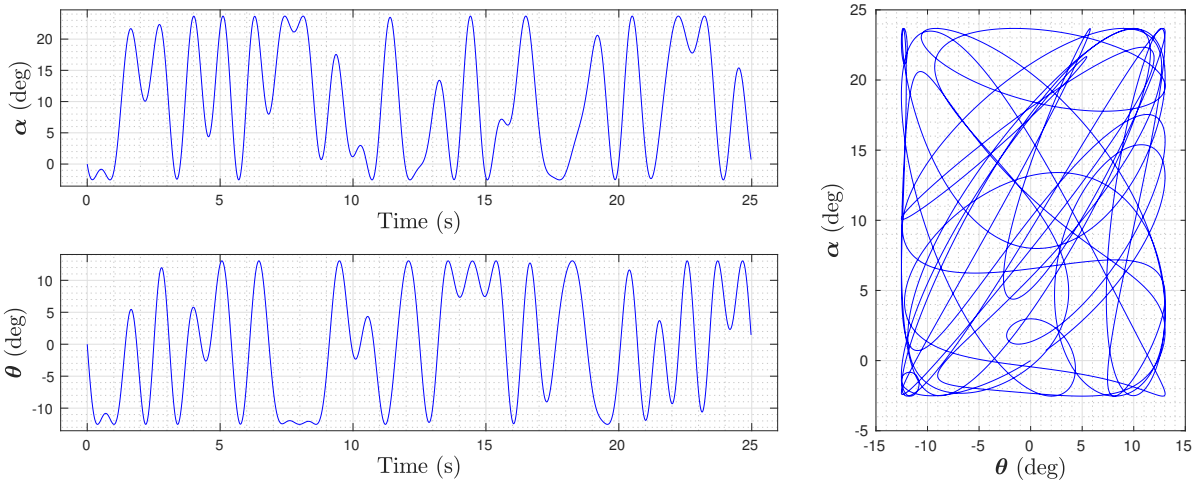


Figure 5.1: Optimized multisine excitations of the motion variables. **Left:** The perturbations are plotted against the time of the excitation. The relative peak factors of the signals are RPF($\mathbf{u}_\alpha$) ≈ 1.05(−) and RPF($\mathbf{u}_\theta$) ≈ 1.02(−). **Right:** The inputs are plotted against each other. The motion covers the complete range of both variables. The normalized cross-correlation at zero lag is $\hat{R} \approx 0.05(-)$.

timesteps for the CFD simulation. It was assumed that these setting provided adequate frequency coverage and sufficient numbers of training examples, while simulation costs were still affordable.

The upper bound of the frequency range in the end was also a design choice and the requirements were only used as guidelines. The maximum was set as 1 Hz. It yielded an angular speed more than double the pitch rate requirement, namely 50 (deg/s). The reason behind the maximum value was twofold. For starters, it was intended to be able to investigate motions faster than the requirement, second, increasing the maximum of the angular speed provided more oscillations in the given time-frame and somewhat better coverage as well. Since the frequency resolution is defined as the lowest frequency, the signals in the end are built from $1/0.04 = 25$ harmonics. A summary of the frequencies is provided in Table 5.1.

Table 5.1: Frequency distribution of the generated Schroeder sweeps

Parameter	Dimension	Value
Signal duration	(s)	25
Lowest frequency	(Hz)	0.04
Maximum frequency	(Hz)	1
No. of frequencies	(-)	25

The resulting signals are shown in fig. 5.1. Corresponding additional motion variables (resolved in the inertial earth reference frame) including x and z location and U and W velocity components are illustrated in fig. 5.2. The relative peak factors of the final signals are RPF($\mathbf{u}_\alpha$) ≈ 1.05(−) and RPF($\mathbf{u}_\theta$) ≈ 1.02(−), whereas their normalized cross-correlation at zero lag is $\hat{R} \approx 0.05(-)$. The power spectrum distribution of the signals are shown in fig. 5.3. Additional regressor space coverages are illustrated in fig. 5.4.

Overall, the signals achieve relatively good coverage over the regressor space, although they do not fit the predefined ranges perfectly. Nonetheless, corresponding power distributions are nicely uniform among the frequencies until 1 Hz reached. After that, signal powers are cut off.

A drawback of the Schroeder sweeps is that the signals are concentrated at their perimeters, as it can be observed in fig. 5.4b, providing less coverage at slower rates. The latter phenomena is typical for Schroeder sweeps [66]. Unfortunately, there is no single solution that can satisfy all the criteria of desired inputs, hence such compromise is inevitable if training costs are to be kept minimal.

5.1.1. TRAINING SET RESULTS

The optimization of the primary and secondary networks took 12h 24m and 5h 32m, respectively. The 5667 timesteps are converted into 5666 examples, 80% of which (4532) was used for training. The convergence data of MSE and EMA losses is illustrated in fig. 5.5. Note that EMA is only included to provide a better overview of

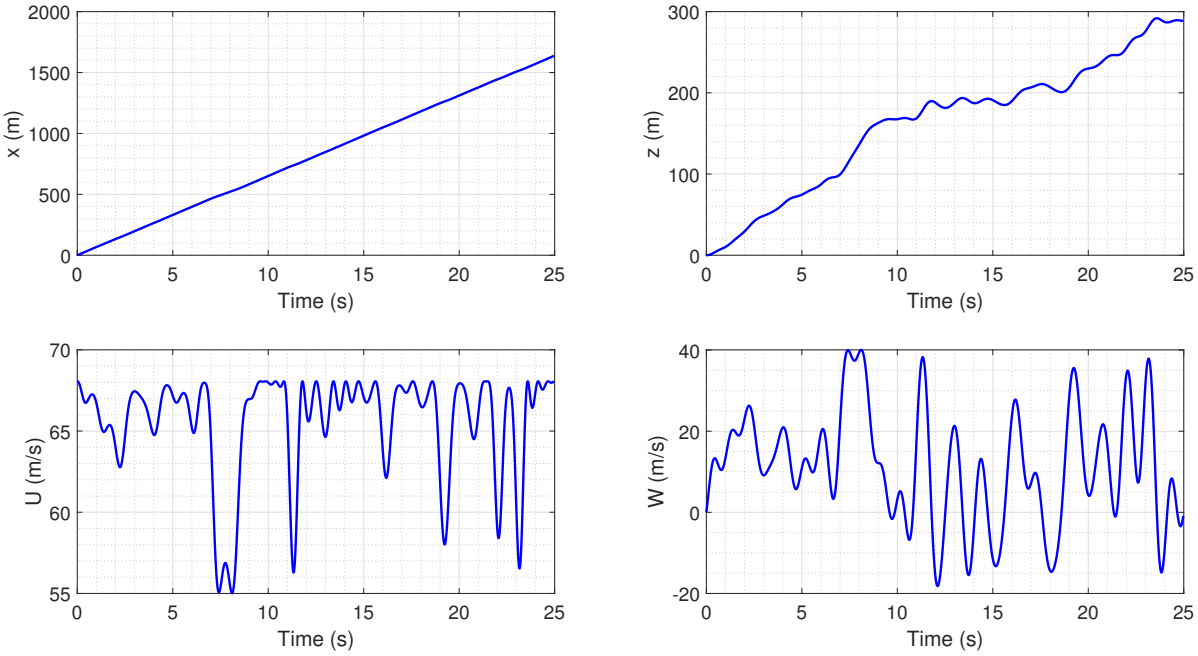


Figure 5.2: Additional motion variables as a function of time. The variables are resolved in the inertial earth reference frame.

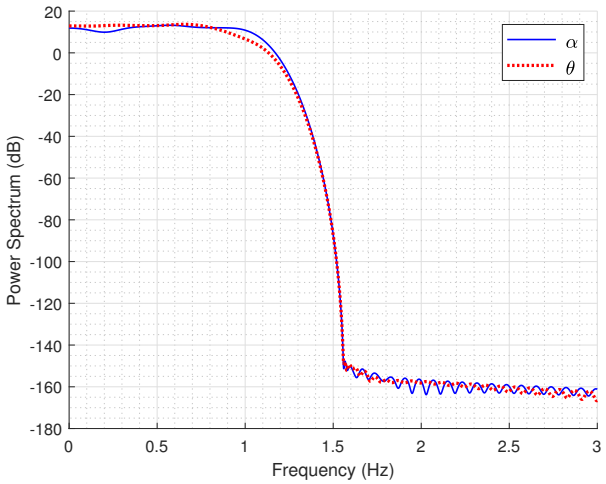
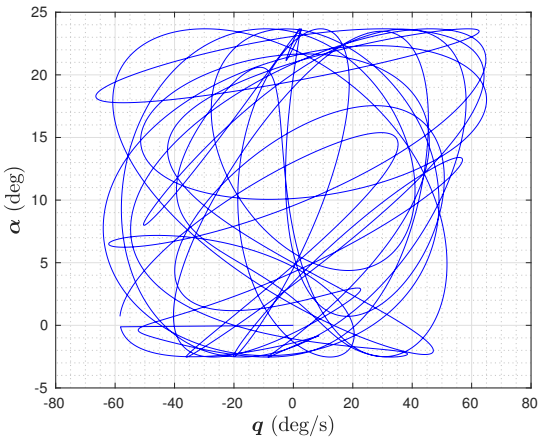
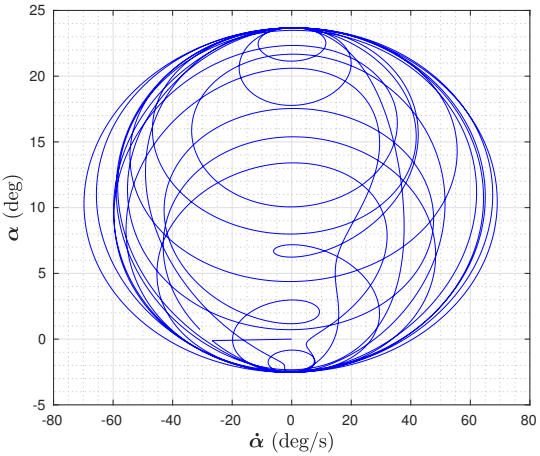


Figure 5.3: Power spectrum of input signals as a function of frequency.

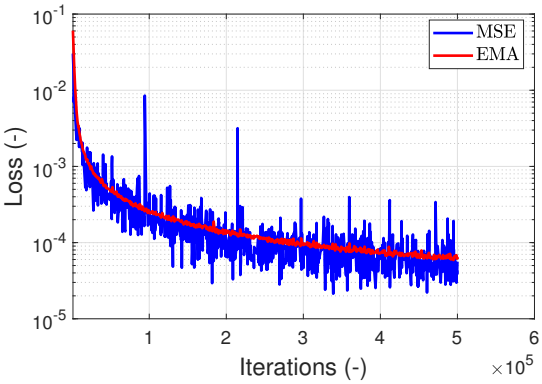


(a) Angle of attack versus pitch rate

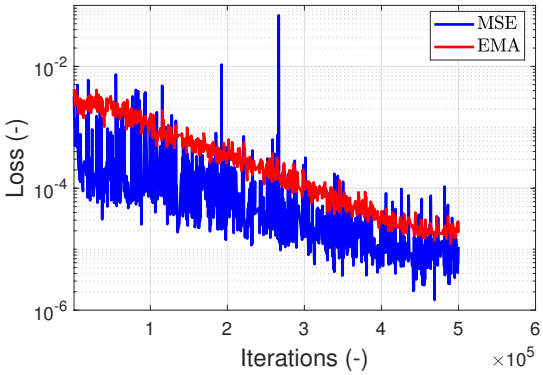


(b) Angle of attack versus change of rate of angle of attack

Figure 5.4: Training maneuver regressor space coverage



(a) Primary network training loss convergence



(b) Secondary network training loss convergence

Figure 5.5: Evolution of training losses.

the progress.

After training the networks were evaluated on the test sets, i.e. on the remnant 20% data from the training maneuver incurring 1134 examples (timesteps). The two networks are assessed together, meaning that the secondary network received the predictions of the primary network. The results are shown on in fig. 5.6 and fig. 5.7. The steps at which the test set is evaluated is highlighted in fig. 5.6.

As evident from the graphs, the network achieves good accuracy on the training set. The largest error recorded for the pressure fields is $\mathrm{MSE}(\hat{\mathbf{t}}^{\,p}, \mathbf{t}; \theta) = 0.2083(-)$ that is calculated over all values in the pressure tensors. Otherwise, errors remained orders of magnitudes lower, typically in the range of $[1e-04,\ 1e-02]$. Concurrently, a general tendency can be observed. The peaks of the errors occur at high AoA conditions, usually in the vicinity of maximum AoAs. Nonetheless, those errors are practically immeasurable in corresponding integral loads, which virtually overlap the CFD results. To gain a comprehensive picture about network performance additional experimenting is necessary.

5.2. STEADY SIMULATIONS

Following network optimization, a set of steady conditions are evaluated. These simulations cover the range of incidence angles from 0 to 20 degrees with zero rates and accelerations. The conditions and the errors measured over the pressure distributions are illustrated in fig. 5.8.

As the figure shows, the magnitudes of the errors are comparable with the errors of the test set. Additionally, similar tendencies are observable. That is, the errors increase for higher angles of attack. To see if these

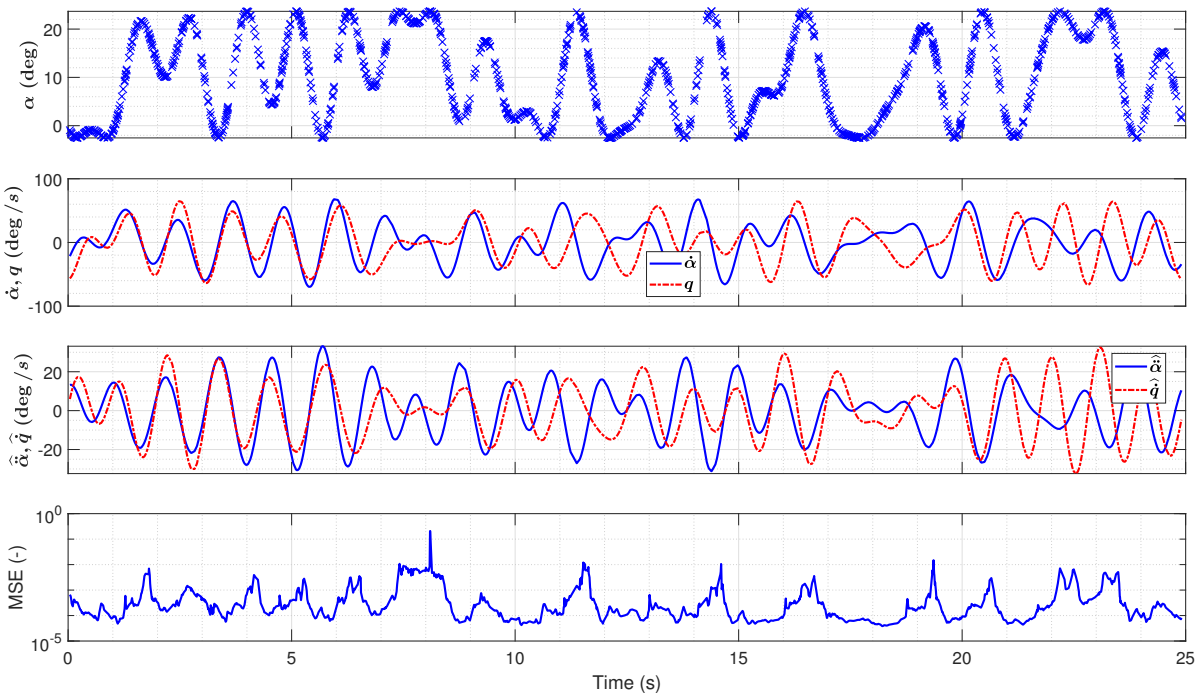


Figure 5.6: Test set control inputs and corresponding MSE of the pressure coefficient predictions. The marker × denotes the α values at which the test set is evaluated.

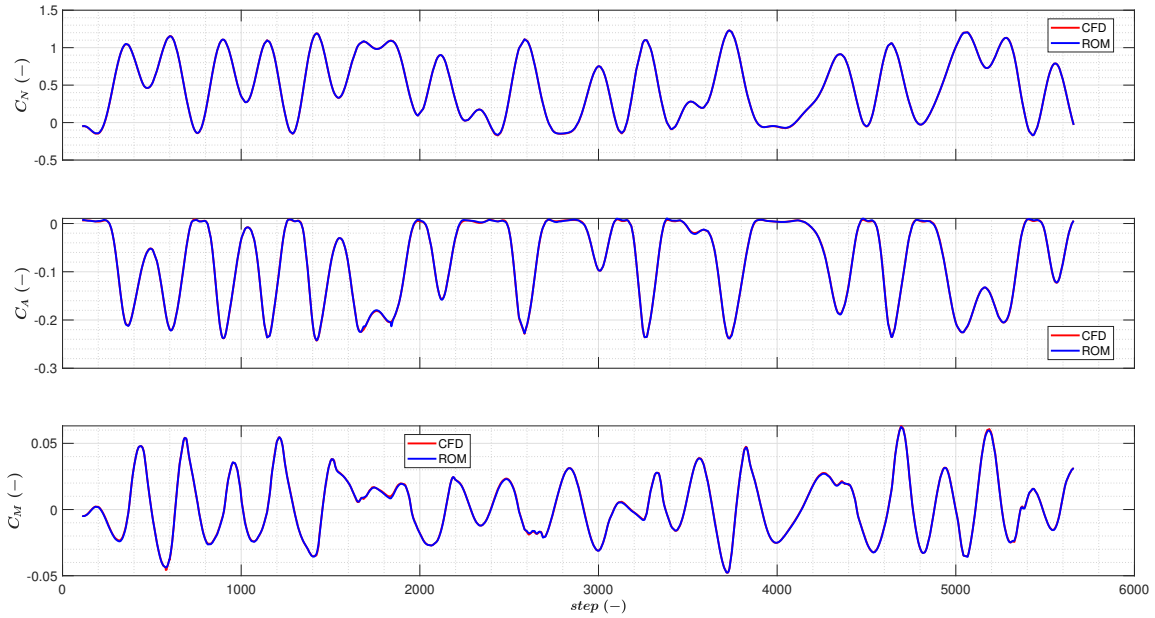


Figure 5.7: Integral load prediction over test set

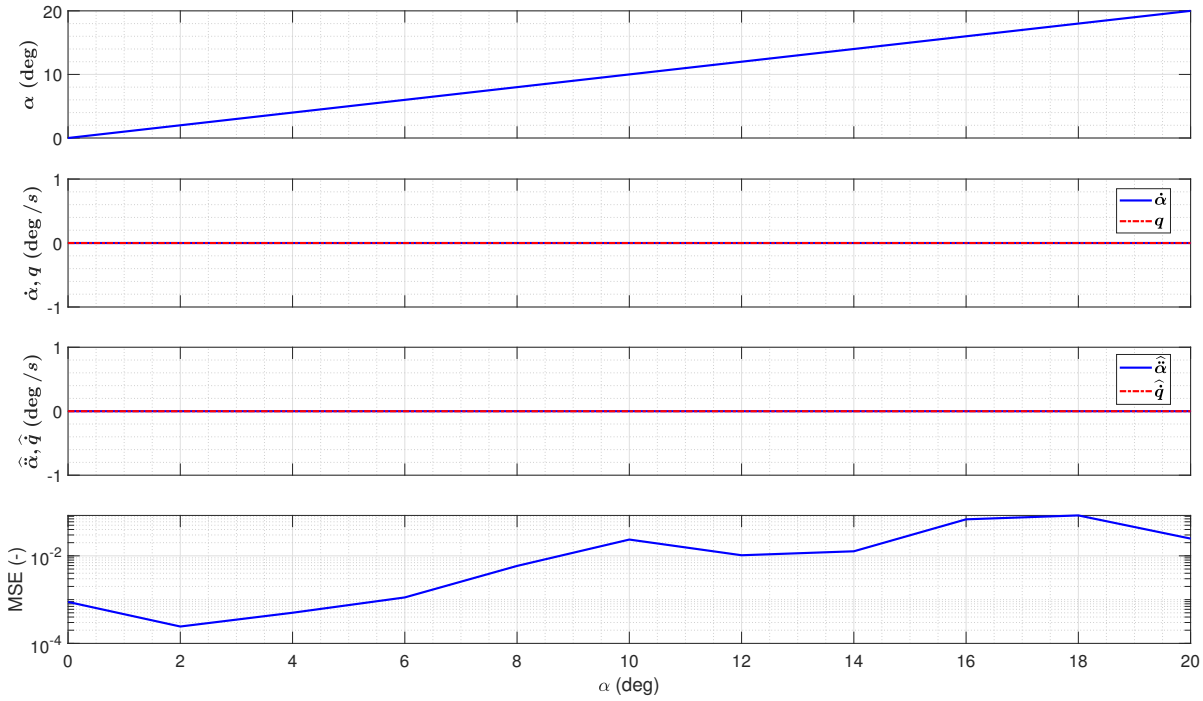


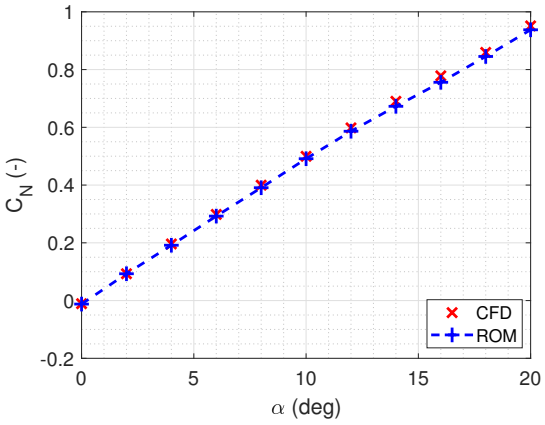
Figure 5.8: Inputs of steady simulations and corresponding MSE of the pressure coefficient predictions.

errors now have any effects on the integral loads, the corresponding coefficients are provided in fig. 5.9.

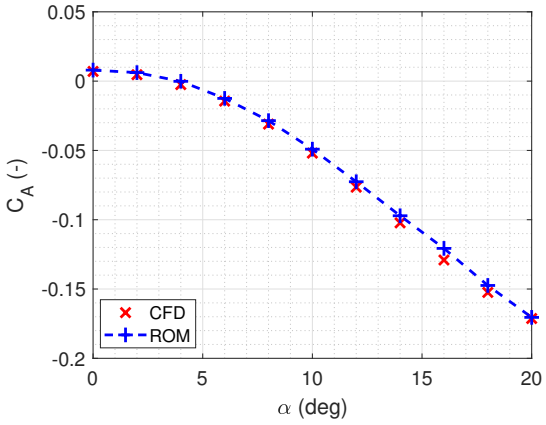
From the graphs it can be concluded that predicted force coefficients retain accuracy for quasi-steady conditions. Errors are negligible. On the other hand, considerable discrepancies occur among simulated and predicted pitching moment coefficients. Predictions in the low angle of attack regime are accurate, but as the errors in the flow predictions rise around 1×10^{-2} at 8 degrees of angle of attack (5^{th} step in the set), errors in pitching moment predictions begin to increase as well. It is assumed that increasing errors are in connection with intensifying nonlinear flow phenomena. Nevertheless, to account for the differences in the predictions surface pressure distributions are investigated, first for $\alpha = 20$(deg) in figs. 5.10 and 5.11. The errors among reference and predicted pressure fields are derived by subtracting the absolute pressure values of the predictions from absolute pressure values of the CFD results and then non-dimensionalized the residuals with the dynamic pressure. The relative dimensions of the aircraft along the x and y axes are highlighted. The $\bar{x}$ and $\bar{y}$ dimensions are normalized with the length and half span of the aircraft, respectively.

First of all, it shall be noted that the lower surface matches well with the CFD results, differences can only be seen at the very edge of the wing. Most errors are observed on the upper surface. Three main differences can be identified. First, the neural network predicts a slightly more energetic vortex emanating from the nose. At the same time, it predicts somewhat higher pressures on the left side of the vortex. Second, right behind the leading edge, the neural network predicts faster pressure recovery and higher pressures. Thirdly, the wingtip has considerably different pressure distribution. In this particular case, CFD results imply a second weak vortex originating from the edge of the high suction area of the leading edge resulting in considerably lower pressure towards the second kink of the trailing edge. Additionally, despite that the neural net is also predicting a small patch of over-pressure at the most outer boards, it misses its exact location, and therefore relatively large portions have lower and relatively large portions have higher pressure at the very wingtip than the CFD simulation. Comparing results at 14 degrees AoA in fig. 5.12, most of the previously mentioned errors on the upper surface are mitigated, only a small patch of error is present at the wingtip. Still, recalling results from fig. 5.9c, pitching moment prediction incurs relatively large errors.

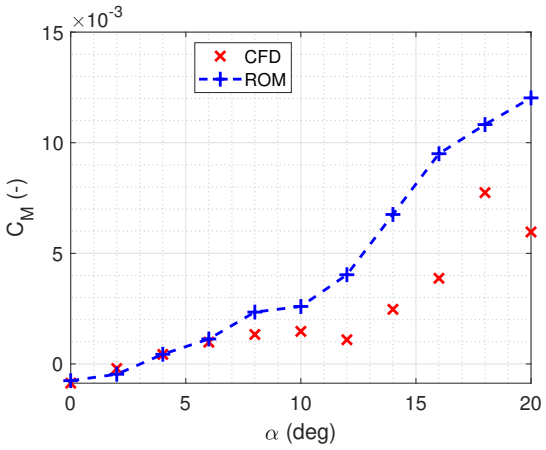
To gain better understanding, the cell wise pitching moment contributions of the surface grids are investigated. First, the pressure forces are computed for each cell using the cell area and the average of the pressures measured at the vertices of the cell. The forces are computed as three element vectors using the surface normals of the cells. Then, the moment contribution of each cell-force is computed. Now consider the matrices as rectangular grids of a computational space in which the locations are mapped to the locations of the physical space. The physical space is the one represented by the x,y z coordinates of the CFD



(a) Steady normal force coefficient



(b) Steady axial force coefficient



(c) Steady pitching moment coefficient

Figure 5.9: Comparison of steady CFD results with ROM predictions.

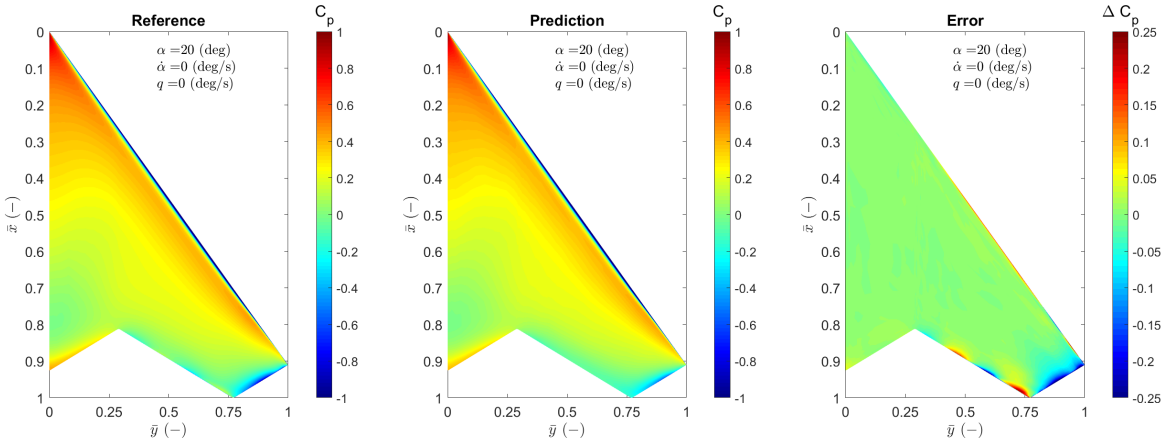


Figure 5.10: Comparison of steady CFD and ROM lower surface pressure distributions for $\alpha = 20\,(\text{deg})$. Variables $\bar{x}$ and $\bar{y}$ denote the normalized dimensions of the aircraft.

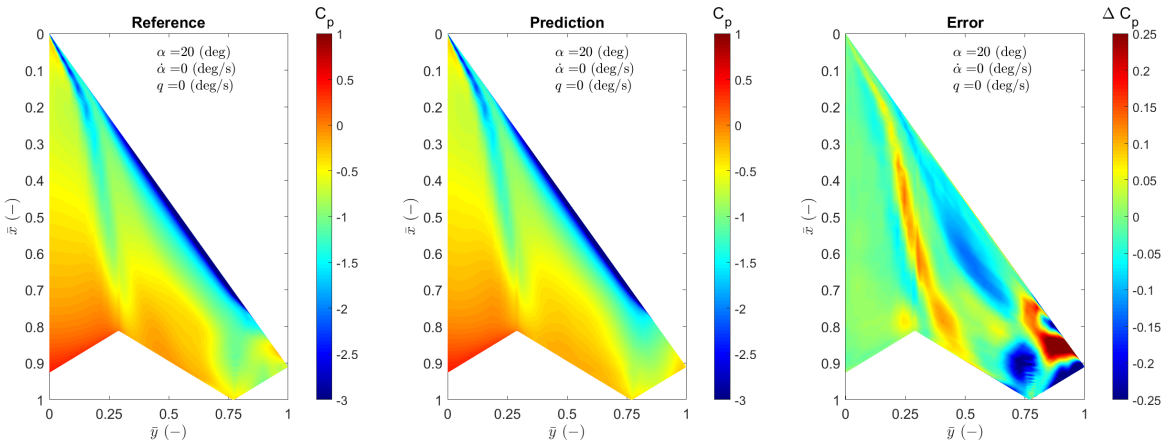


Figure 5.11: Comparison of steady CFD and ROM upper surface pressure distributions for $\alpha = 20\,(\text{deg})$. Variables $\bar{x}$ and $\bar{y}$ denote the normalized dimensions of the aircraft.

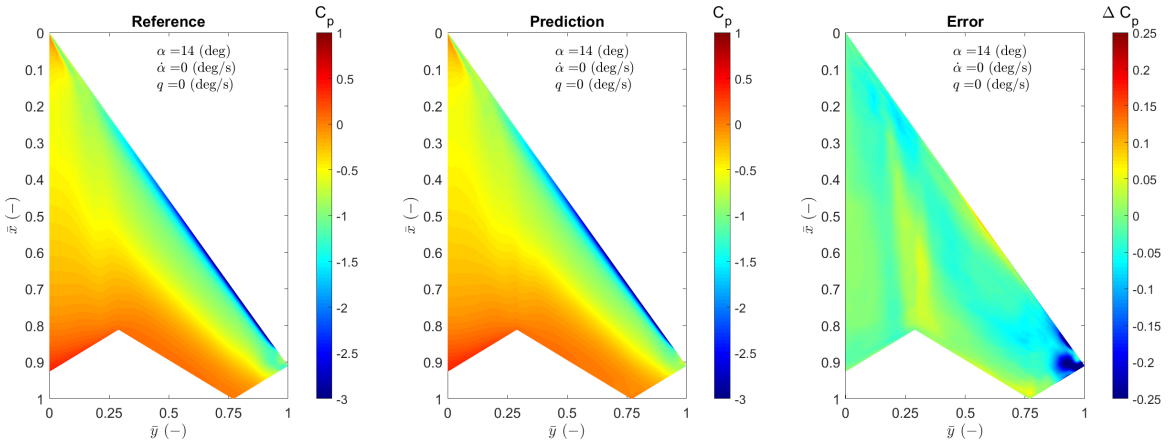


Figure 5.12: Comparison of steady CFD and ROM upper surface pressure distributions for $\alpha = 14\,(\text{deg})$. Variables $\bar{x}$ and $\bar{y}$ denote the normalized dimensions of the aircraft.

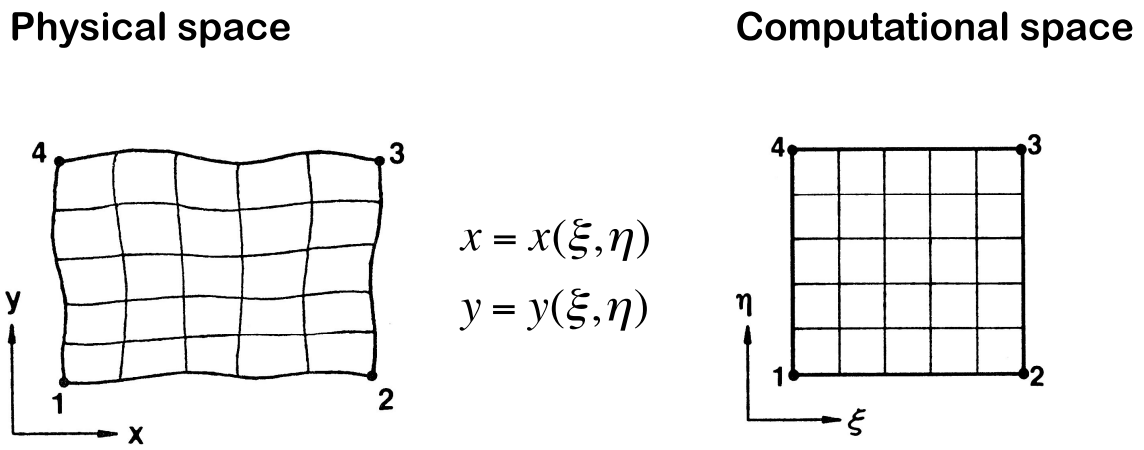


Figure 5.13: Simple example of mapping between physical and computational spaces [73].

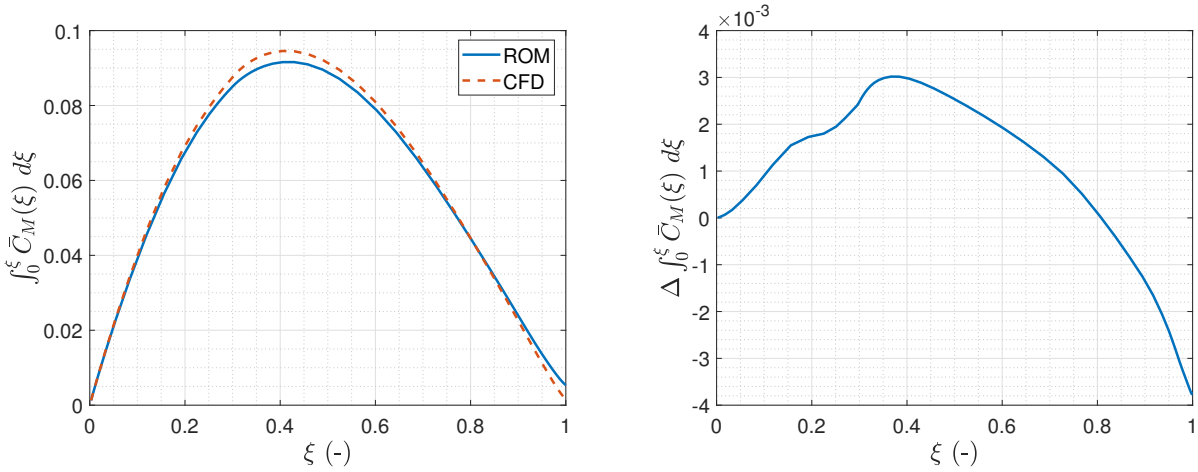


Figure 5.14: Comparison of accumulated pitching moment distributions as a function of relative distance travelled along the leading edge at $\alpha = 14\,\text{deg}$.

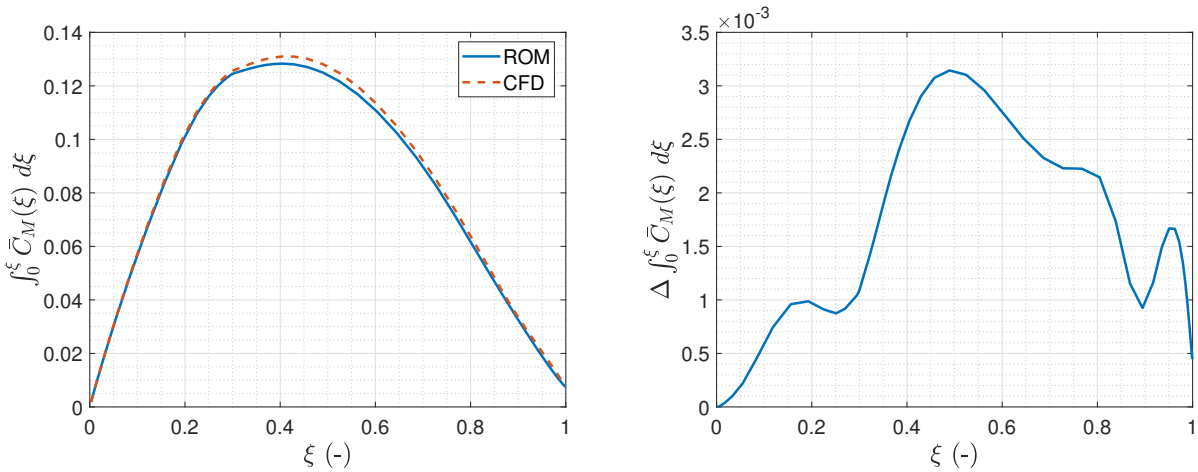


Figure 5.15: Comparison of accumulated pitching moment distributions as a function of relative distance travelled along the leading edge at $\alpha = 20\,\text{deg}$.

mesh. An example of such a mapping is provided in fig. 5.13. Then, the rows of the matrices are denoted by η_i and the columns by ξ_i. Having the data stored in matrix format as outlined in Section 4.3.2, it is convenient to summarize the pitching moment contributions along the η directions. It is somewhat analogous to summarizing the pitching moment contributions above the chords. These sums are denoted by $\bar{C}_M(\xi)$. Let the ξ abscissa be the relative distance travelled along the leading edge from the centerline toward the wingtip. Then, plotting values above ξ is analogous to plotting above the span. Finally, instead of the simple values and errors, consider the integrals of $\bar{C}_M(\xi)$ and the errors among CFD and ROM integrals along ξ. These integrals, for the sake of clarity, are regarded as accumulated pitching moment distributions from now on. By plotting the accumulated pitching moment distributions, the contribution of stations along the leading edge can bet better interpreted. Such comparisons are illustrated in fig. 5.14 and in fig. 5.15 for $\alpha = 14\,\text{deg}$ and $\alpha = 20\,\text{deg}$, respectively.

Investigating the figures reveals two different scenarios. First, the errors that are accumulated over the inboard stations of the two cases show similar tendencies. They both reach roughly the same maximum around $\xi = 0.4(-)$. The errors of the $\alpha = 20\,\text{deg}$ case exhibit sharper changes, which is possibly due to the larger errors between CFD and ROM pressure distributions predictions in the vicinity of the main vortices. At the same time, the outer boards differ considerably. On one hand the errors for $\alpha = 20\,\text{deg}$ show greater variance, which is not that surprising, considering that pressure field predictions suffer more from errors. Nevertheless, the accumulated errors cancel out towards the wingtip. Yet, in fig. 5.9c a great error can be observed. Note that the accumulated values show the values that computed directly from CFD and ROM

pressure distributions and it is not the pitching moment prediction of the neural net. This means, that even though the pressure fields would result in roughly the same pitching moments, the neural network greatly over predicts these values. Concurrently, for $\alpha = 14\,\text{deg}$ after the maximum value is reached, the error starts to sharply decrease towards the wingtip. In the end, roughly an error of -4×10^{-3} (-) is accumulated. By comparing that value with the prediction in fig. 5.9c, it can be concluded that both the magnitude and the sign of the error are in good agreement with the difference between C_M CFD calculations and ROM predictions. Additionally, the accumulated pitching moment is close to the value that is predicted by the neural network. This means that for $\alpha = 14\,\text{deg}$ the neural network captures the measurable C_M of the corresponding pressure distribution with quite good accuracy.

From these two cases two different sources of errors can be learnt. First, errors between CFD and ROM C_M values can be due to the inaccurate pressure distribution predictions of the primary network. Importantly, the case of $\alpha = 14\,\text{deg}$ shows that even small patches of inaccurate pressures around the wingtip can cause large differences in the pitching moments. Second, errors can originate from inaccurate integral load predictions of the secondary network. Although, this source of error is harder to account for, two causes are considered. On one hand the training maneuver at best includes harmonics of very low frequencies, but it does not include true steady conditions. Second, note that the secondary network also couples the motion variables with the encoded feature tensors. Again, since no true steady conditions are included in the training maneuver, it can easily be that the network falsely identifies a different scenario, for which the pitching moment should be larger.

Even though some conclusion could already be drawn, the main focus of the project are the unsteady conditions. Therefore, to get a more complete picture about actual model performance, unsteady maneuvers must be investigated as well, about which the current simulations provide little information. Hence, the next section will study model performance for unsteady maneuvers.

5.3. HARMONIC OSCILLATIONS: MODEL PERFORMANCE OVER THE REGRESSOR SPACE

To assess the model performance over the regressor space a set of harmonic excitations are defined that are applied directly to the incidence angle, just as the Schroeder sweeps. Two types of motions are investigated, namely pitch and plunge oscillations. Pitching motions are realized through the pitch angle while the absolute velocity vector of the aircraft is kept unchanged. In contrast, plunges are applied via the normal and longitudinal components of the absolute velocity vector whereas the pitch angle remains unchanged. For both motions only the AoA is commanded, other motion variables are derived from the boundary conditions. Therefore, the input signals are identical for both cases.

The aircraft is excited with an amplitude of 5 degrees around 5, 10 and 15 degrees of angle of attack. Additionally, an extra set is defined around 10 degrees of AoA with 10 degrees of amplitude. The frequencies for the 5-degree amplitude motions are 0.25, 0.5 and 1 Hz whilst for 10 degrees 0.125, 0.25 and 0.5 Hz. The corresponding angular speeds are 5, 10 and 20 degrees per second. The generic sinusoidal input of the excitations is shown in fig. 5.16. Corresponding motion variables for pitch oscillations are shown in fig. 5.17 in non-dimensional form. As it is evident from the graphs, for pitch oscillations the velocities are constant, as defined by the boundary conditions. Consequently, position the aircraft changes linearly. The variables of the plunge oscillations are included in fig. 5.18 and fig. 5.19. The motion variables of the plunge oscillations resemble the sinusoidal input of the incidence angle, as now the pitch angle is fixed and the angles of attacks are realized through the velocity components. As it is evident from the graphs, the outlook of the variables depends on the nominal value and amplitude. Since the AoA inputs of the plunge oscillations are realized solely via the velocity components, if the AoA becomes zero, so does the vertical velocity. At the same time, the horizontal velocity flattens, due to the fixed Mach number condition. The latter happens for $A_0 = 5(\text{deg})$ and for $A_0 = 10(\text{deg})$ with $A = 10(\text{deg})$. Typical values of the motion variables are summarized and listed in Table 5.2 and Table 5.3 for pitch and plunge oscillations, respectively.

The full set of resultant integral loads are enclosed in appendix A.1. In the main text only the key findings will be shown along with a few examples. Mind that certain oscillations are simulated over two periods, due to the high fluctuations observable in CFD reference data.

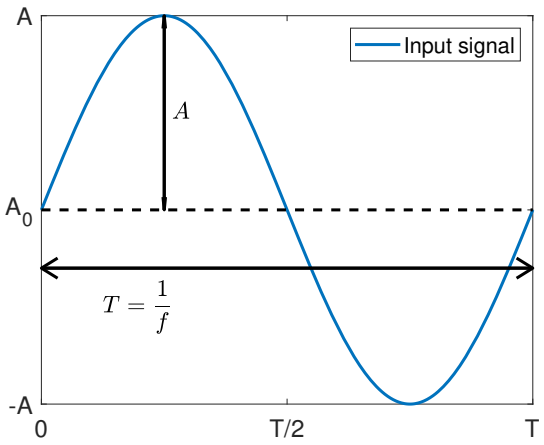


Figure 5.16: Input signal of harmonic excitations. The signal commands the angle of attack of the aircraft. The remaining inputs are derived from the boundary conditions.

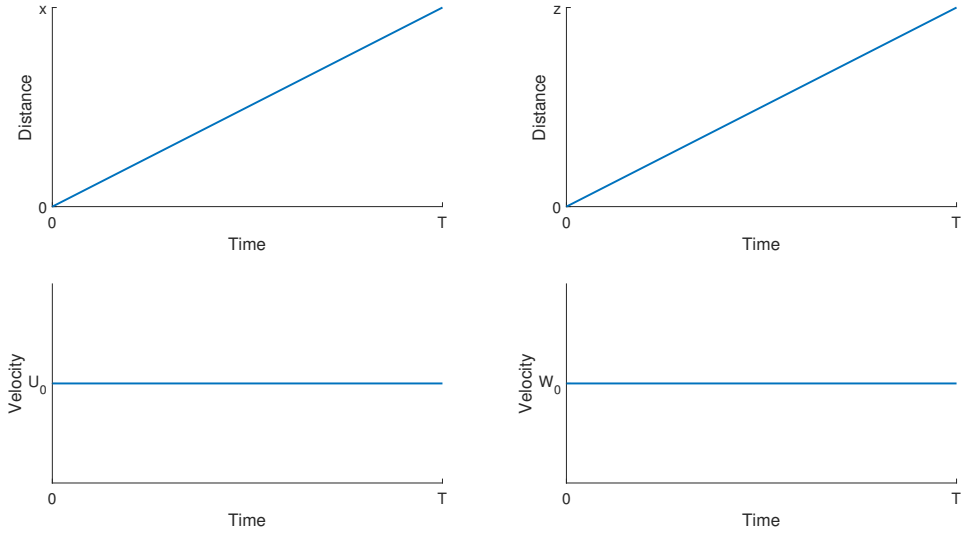


Figure 5.17: Pitch oscillation motion variables as a function of time. The variables are resolved in the inertial earth reference frame.

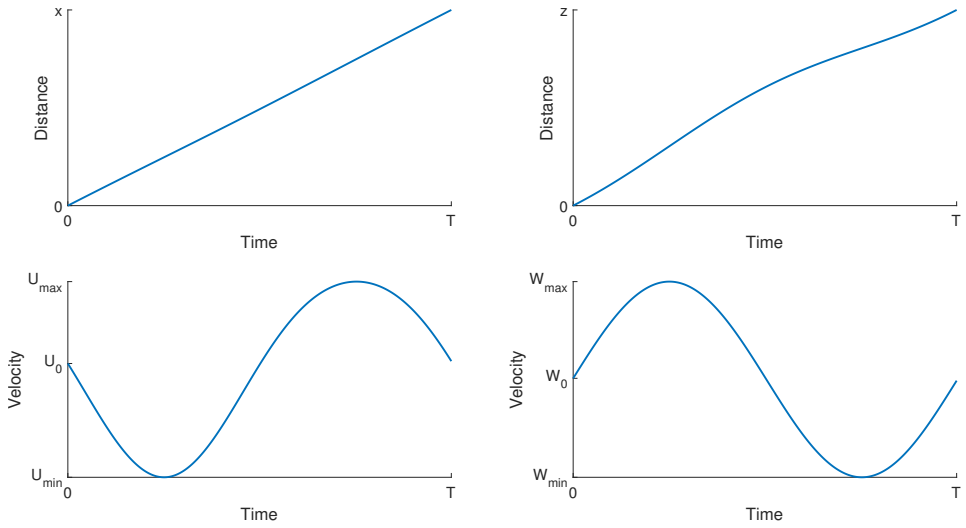


Figure 5.18: Plunge oscillation motion variables as a function of time, $\alpha_{min} > 0$. The variables are resolved in the inertial earth reference frame.

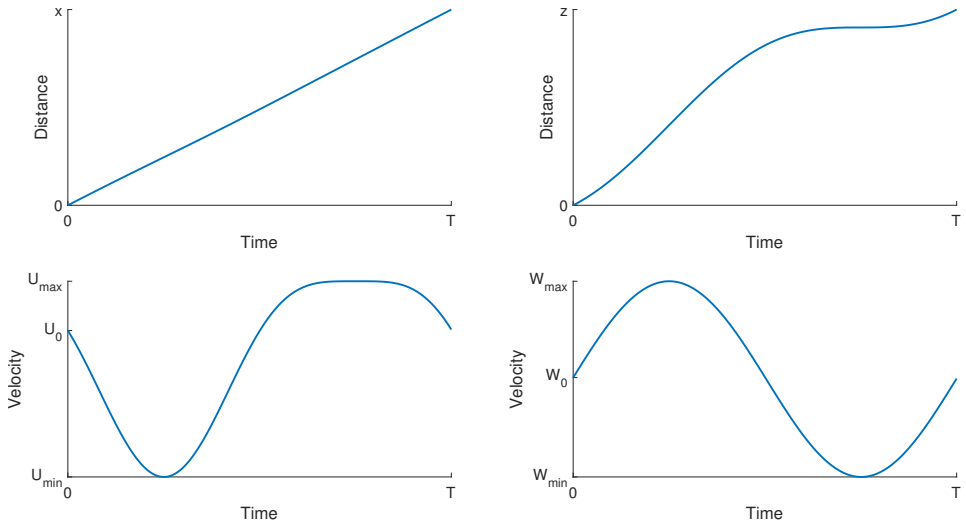


Figure 5.19: Plunge oscillation motion variables as a function of time, $\alpha_{min} = 0$. The variables are resolved in the inertial earth reference frame.

Table 5.2: Pitch oscillation motion variables resolved in the inertial earth reference frame.

Variable	Symbol	Dimension	Values											
Nominal AoA	A_0	(deg)	5			10			10			15		
Amplitude	A	(deg)	5			5			10			5		
Frequency	f	(Hz)	0.25	0.5	1.0	0.25	0.5	1.0	0.125	0.25	0.5	0.25	0.5	1.0
Duration	T	(s)	4	2	1	4	2	1	8	4	2	4	2	1
Horizontal distance	x	(m)	271.2	135.6	67.8	268.0	134.0	67.0	536.0	268.0	134.0	262.8	131.4	65.7
Vertical distance	z	(m)	23.7	11.8	5.9	47.2	23.6	11.8	47.2	23.6	11.8	70.4	35.2	17.6
Horizontal velocity	U_0	(m/s)	67.8			67.0			67.0			65.7		
Vertical velocity	W_0	(m/s)	5.9			11.8			11.8			17.6		

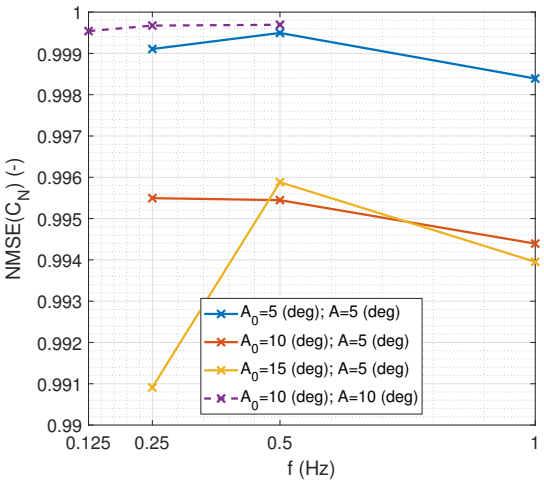
5.3.1. RESULTS OF HARMONIC EXCITATIONS

Generally speaking, flow prediction errors are higher than in the previous cases. Nevertheless, the tendencies observed in the results are similar to the quasi-steady simulations. That is, the force coefficients are usually accurate both for pitching and plunging motions, whilst pitching moment predictions are erroneous. To provide qualitative comparison of model performance over the regressor space, the Normalized Mean Squared Error (NMSE) of the results are computed. The NMSE values are calculated as:

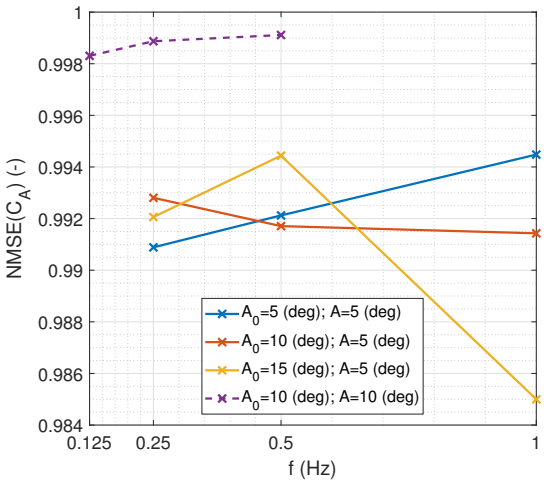
$$\text{NMSE}(\hat{\mathbf{t}}^{\,i}) = 1 - \frac{\|\mathbf{t}^{\,i} - \hat{\mathbf{t}}^{\,i}\|^2}{\|\mathbf{t}^{\,i} - \mathbb{E}(\mathbf{t}^{\,i})\|^2} \tag{5.2}$$

where $\mathbf{t}^{\,i}$ and $\hat{\mathbf{t}}^{\,i}$ denote the vectors of the reference and predicted values of the i^{th} variable, respectively. Furthermore $\mathbb{E}(\mathbf{t}^{\,i})$ is the expected value of the reference vector, which is now defined as the mean. The NMSE of the predictions would be 1 if all elements of the tensors exactly matched their reference values. An MNSE of 0 would mean predictions that were not better at matching the reference than fitting one straight line through the complete dataset [74]. NMSE values smaller than 0 are worse than that. The measure has no lower bound (-inf). The NMSE results of the normal and axial force coefficients for pitch oscillations are illustrated in fig. 5.20. The NMSEs of corresponding pitching moment coefficients are provided in fig. 5.21.

First consider the motions with an amplitude of 5 degrees. As fig. 5.20a and fig. 5.20b show, the force coefficients for pitching motions are highly accurate. Although, their values might seem to have great variance, note that the NMSE values are all very high and close to 1. Concurrently, the pitching moment coefficient actually shows great variance with respect to the incidence angle and frequency. Despite having only a few number of simulations in the set, the results of fig. 5.21 imply that fast motions can be simulated relatively accurately. In contrast, as the AoA increases decreasing frequencies have more adverse effects. In the most extreme case ($\alpha = 15$(deg), $f = 0.25$ Hz), the prediction of the neural network fits the CFD data hardly any better than a linear regression would do, as an NMSE of roughly 0.2 (-) is recorded. Also, note that the most accurate



(a) Normal force coefficient NMSE values for pitch oscillations at varying nominal values and amplitudes as a function of frequency.



(b) Axial force coefficient NMSE values for pitch oscillations at varying nominal values and amplitudes as a function of frequency.

Figure 5.20: Accuracy of force coefficients in terms of NMSE for pitch oscillations.

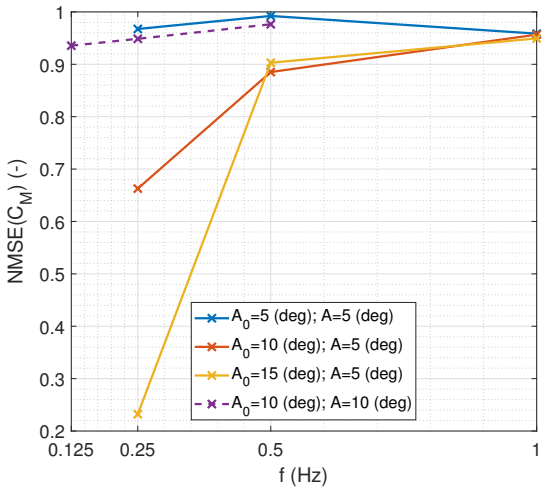


Figure 5.21: Pitching moment coefficient NMSE values for pitch oscillations at varying nominal values and amplitudes as a function of frequency.

Table 5.3: Plunge oscillation motion variables resolved in the inertial earth reference frame.

Variable	Symbol	Dimension	Values											
Nominal AoA	A_0	(deg)	5			10			10			15		
Amplitude	A	(deg)	5			5			10			5		
Frequency	f	(Hz)	0.25	0.5	1.0	0.25	0.5	1.0	0.125	0.25	0.5	0.25	0.5	1.0
Duration	T	(s)	4	2	1	4	2	1	8	4	2	4	2	1
Horizontal distance	x	(m)	270.5	135.1	67.4	267.5	133.6	66.6	531.9	265.9	123.8	262.3	131.0	65.4
Vertical distance	z	(m)	23.7	11.8	5.9	47.2	23.6	11.8	93.8	46.9	23.4	70.3	35.1	17.5
Horizontal velocity	U_0	(m/s)	67.8			67			67			65.7		
Horizontal velocity min.	U_{max}	(m/s)	67.3			65.7			64.0			64.0		
Horizontal velocity max.	U_{min}	(m/s)	68.1			67.8			68.1			67.0		
Vertical velocity	W_0	(m/s)	5.9			11.8			11.8			17.6		
Vertical velocity min.	W_{max}	(m/s)	0.0			5.9			0.0			11.8		
Vertical velocity max.	W_{min}	(m/s)	11.8			17.6			23.3			23.3		

set is consists of the motions around 5 degrees of AoA. That is possibly due to the simple flow solutions and the lack of strong nonlinearities. Now consider the motions with an amplitude of 10 degrees. Interestingly, the simulations exhibit consistent performance among all conditions. The previously observed problems of the pitching moment coefficients are simply gone, and the prediction accuracy for pitching moment coefficients is in the order of magnitude of the prediction accuracy observed for force coefficients. Nevertheless, it does not mean that predictions are perfect. Predictions can still deviate locally, although, it is true that other parts of the curves of ROM predictions generally align well with CFD data.

Again, to gain some better insight, two examples are considered. The first has a high NMSE for C_M set ($\alpha = 15(\text{deg})$, $A = 5(\text{deg})$, $f = 0.5\,\text{Hz}$). The control inputs and the integral loads of the simulation are illustrated in fig. 5.22 and Section 5.3.1, respectively. The errors of the pressure distributions are plotted in terms of MSE. For the integral loads the Root Square Errors (RSE) is considered. Both the flow and the integral load predictions exhibit the largest errors in the vicinity of 20 degrees, between 0.2 and 0.3 seconds of simulation time. In the current case, the force coefficients also show some deviation from the CFD reference. An additional snapshot comparing of CFD and ROM pressure coefficient distributions at the time of the largest errors $t \approx 0.22(s)$ is provided in fig. 5.24. As it is evident from the graph there is now great difference among the pressure distributions. The neural network over predicts suction under the main vortex, but at the same time it under predicts the suction at the leading edge towards the wingtip. Additionally, in the CFD results pressures tend to recover more regularly. In contrast, the neural network predicts higher suction towards the trailing edges of the wingtip. As a consequence, at the given time there is a considerable difference between the CFD and ROM C_M values. Comparing the accumulated pitching moment distributions in fig. 5.24 shows, that the large deviations are accumulated over the outer portions of the wing, where the larger errors in the pressure distributions can be observed. The final error computed for the CFD and ROM pressure distributions is approximately -10×10^{-3}. That agrees with the difference observed in fig. 5.23c, therefore it is assumed that the errors of the pitching moments are mostly due to the errors in the pressure distribution. Interestingly the force coefficients, especially the normal force coefficient, are less sensitive even in such cases. Note that despite this maneuver has good NMSE, in local regions considerable errors can be present, just as the previous examples showed.

After that, now the second pitching case is considered, which has the lowest NMSE for C_M ($\alpha = 15(\text{deg})$, $A = 5(\text{deg})$, $f = 0.25\,\text{Hz}$). The control inputs and integral loads are illustrated in fig. 5.26 and Section 5.3.1, respectively. For the current motion a snapshot at 2 seconds of simulation time is investigated. At that instant the pressure predictions are of average order of magnitude and the errors in the pitching moment coefficients are large. Note that the pitching moments are a magnitude smaller than in the previous case, yet the RSE values are of the same magnitude. A comparison of CFD and ROM upper surface pressure distributions is included in fig. 5.28. By investigating the pressure distributions, it can be concluded that large portions of wing are indeed resolved accurately. However, there are some errors at the wing tip. Contrary to the previous case the, now the neural network predicts a more regular pressure recovery. At the same time, it predicts greater section just ahead of that region. Considering the accumulated pitching moment distributions suggest that it is again the error in the flow prediction that drives the prediction of C_M away from CFD data. The magnitude of the accumulated error (approx. 8×10^{-3}) accounts for the difference between the neural network prediction and CFD calculations. Despite the large errors in the pitching moment the force coefficients are still accurate at the given instant, possibly due to the fact that the pressure distribution is still relatively accurate except for the wingtip.

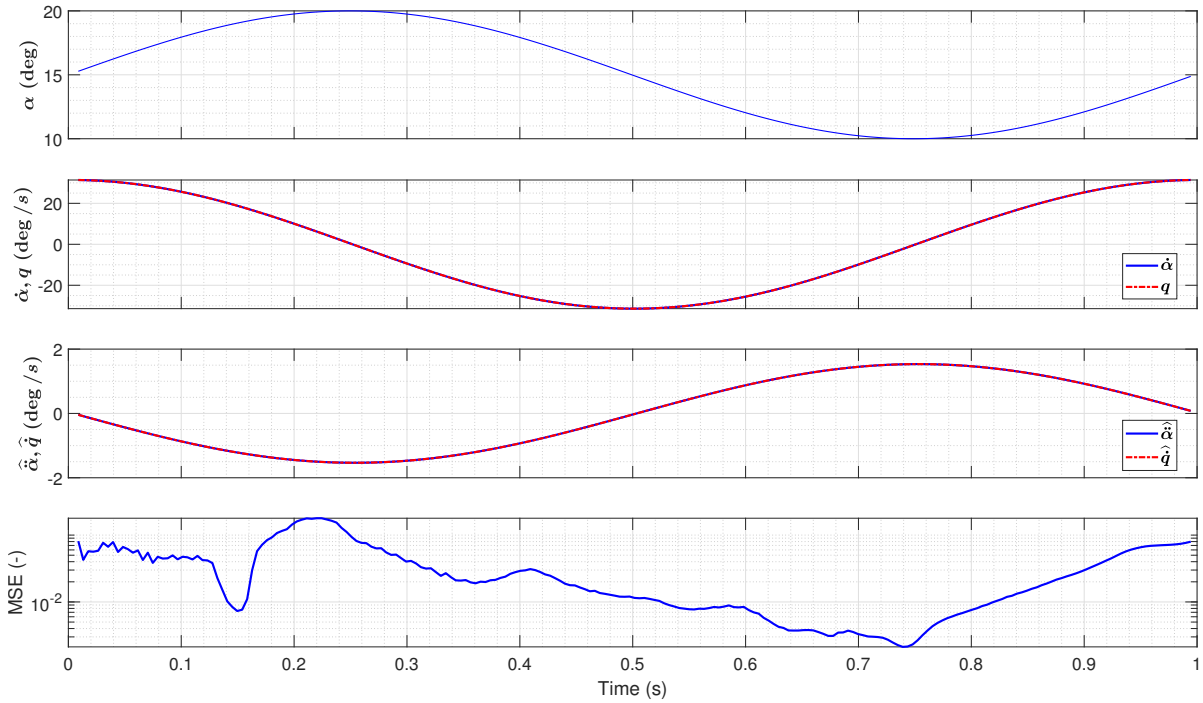
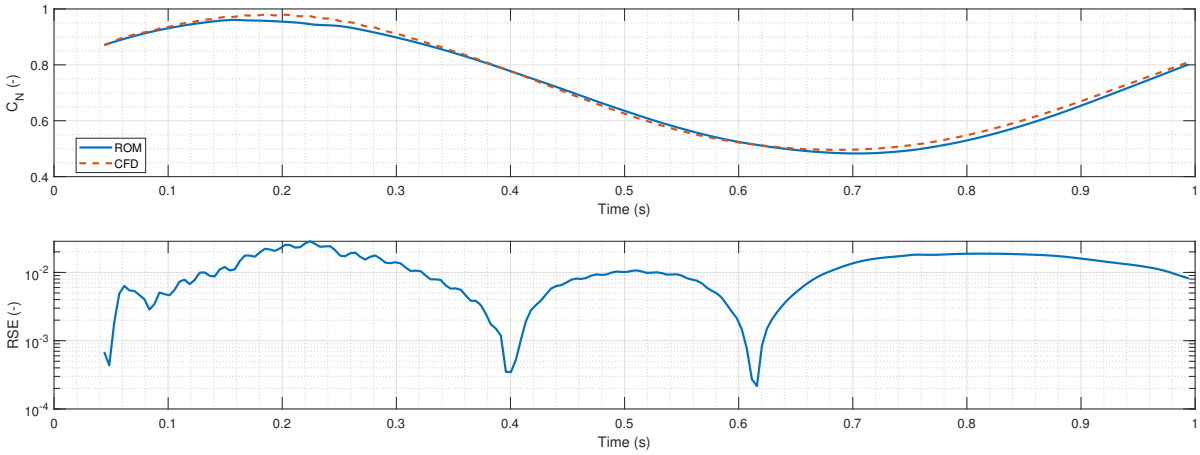


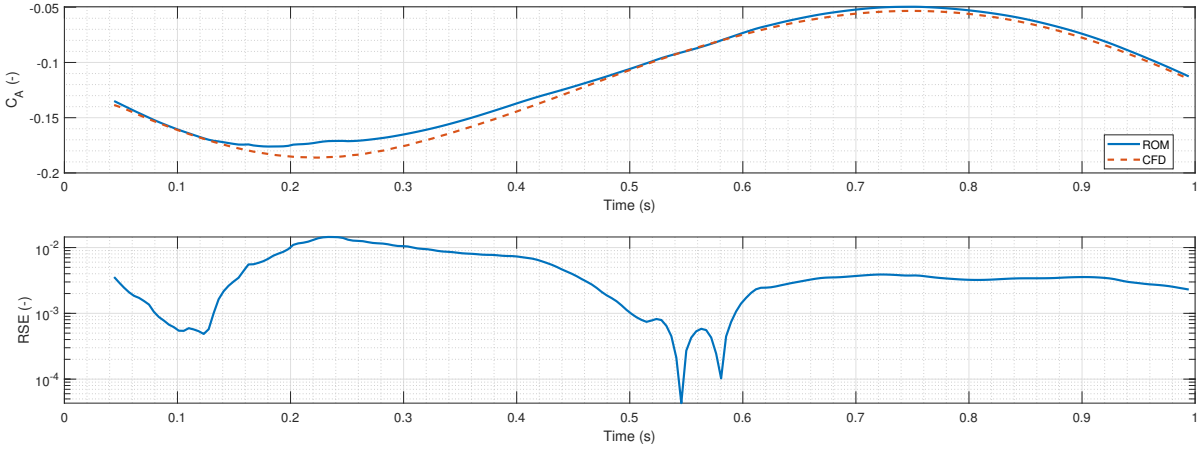
Figure 5.22: Neural network control inputs for pitch oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$.

Next, similar investigations are carried out for the plunge oscillations. The NMSE values observed for the force coefficients are illustrated in fig. 5.30 while the corresponding chart of C_M NMSE is included in fig. 5.31. Again, consider motions with 5 degrees of amplitudes. C_N and C_A fit well the CFD results, and just as for the pitch oscillations, moment coefficient predictions incur large variance and errors. Besides, errors in pitching moments exhibit different tendencies with respect to AoA and frequency. As fig. 5.31 shows, the largest errors are experienced around 5 Hz and 10 degrees of AoA. Considering motions with 10 degrees of amplitude, the issues of inaccurate pitching moments are mitigated just as for the pitch oscillations. Similarly, the best fits and most consistent performance is achieved by the 10-degree amplitude set around 10 degrees of AoA. However, the accuracy of pitching moments for plunges in general is inferior to pitch oscillations.

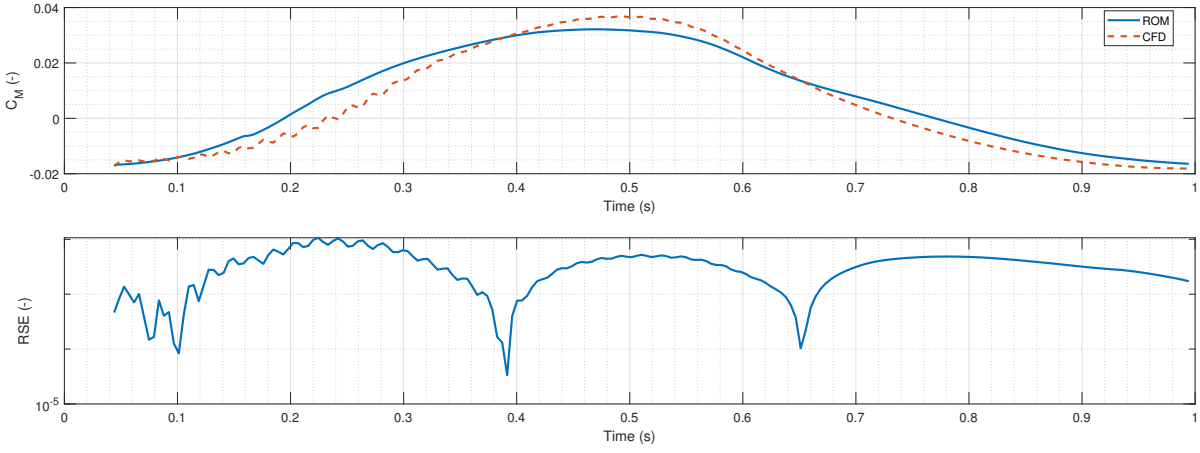
To understanding, why plunge oscillations have such large errors for pitching moment coefficients, the motion with the worst C_M NMSE ($A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$) is investigated. The inputs of the motion are shown in fig. 5.32. The corresponding integral loads are provided in Section 5.3.1. As it can be seen in the figures, the ROM prediction of C_M exhibits quite different tendencies for increasing AoA compared to CFD simulations. The largest error in C_M arise at 0.5 seconds, at the apex of the AoA curve. At the same time, the error observed over the corresponding pressure distribution is not abnormally large. Examining pressure field predictions, e.g. fig. 5.34, it is believed once again that errors at the wingtip are the main cause of the problems. Reading off the accumulated error (approx. $-6 \times 10^{-3}\,(-)$) in fig. 5.35 suggest that this error is responsible for the difference between CFD and ROM results. Notice that the magnitude of the error is not larger than for pitching motions, however, the CFD results for this particular case are a magnitude smaller than the pitching moments of the pitch oscillations. Hence, errors of the same magnitude will cause proportionally larger relative errors.



(a) Comparison of CFD and ROM results of C_N as a function of time.



(b) Comparison of CFD and ROM results of C_A as a function of time.



(c) Comparison of CFD and ROM results of C_M as a function of time.

Figure 5.23: Comparison of CFD and ROM results of integral loads as a function of time for pitch oscillation at $A_0 = 15$ (deg), $A = 5$ (deg), $f = 1$ (Hz). The first 10 timesteps are cut off so the initialization of CFD is excluded.

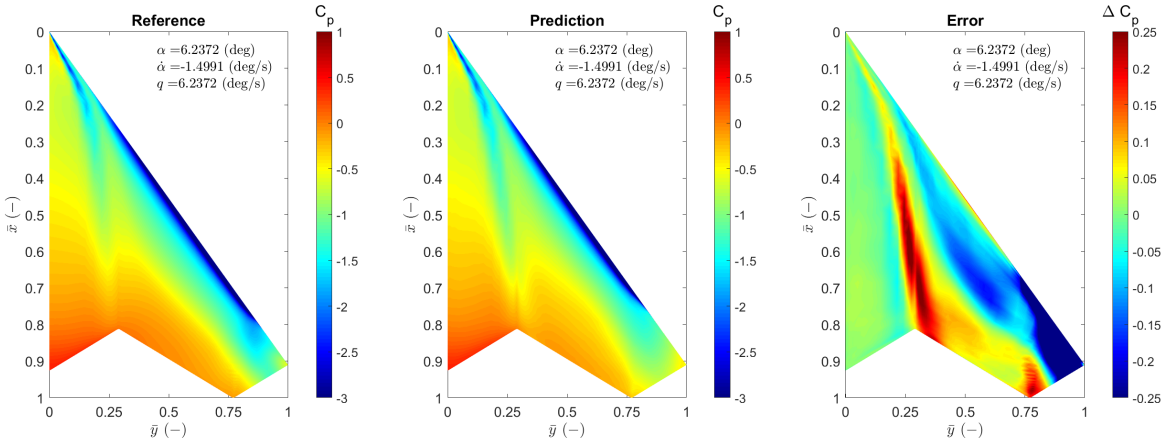


Figure 5.24: Comparison of CFD and ROM results of upper surface pressure distributions for pitch oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$; $t \approx 0.22(s)$. Variables $\bar{x}$ and $\bar{y}$ denote the normalized dimensions of the aircraft.

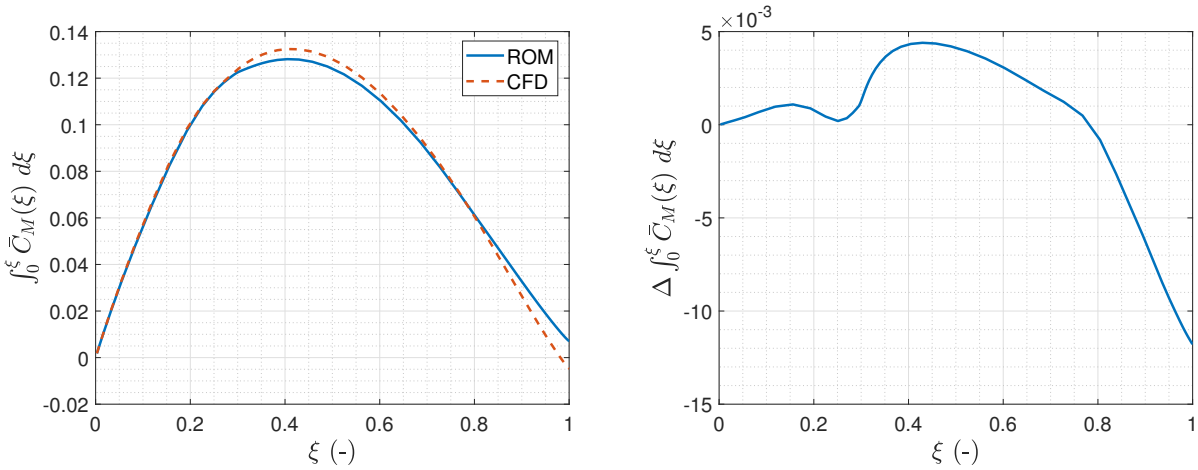


Figure 5.25: Comparison of accumulated pitching moment distributions as a function of relative distance travelled along the leading edge for pitch oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$; $t \approx 0.22(s)$.

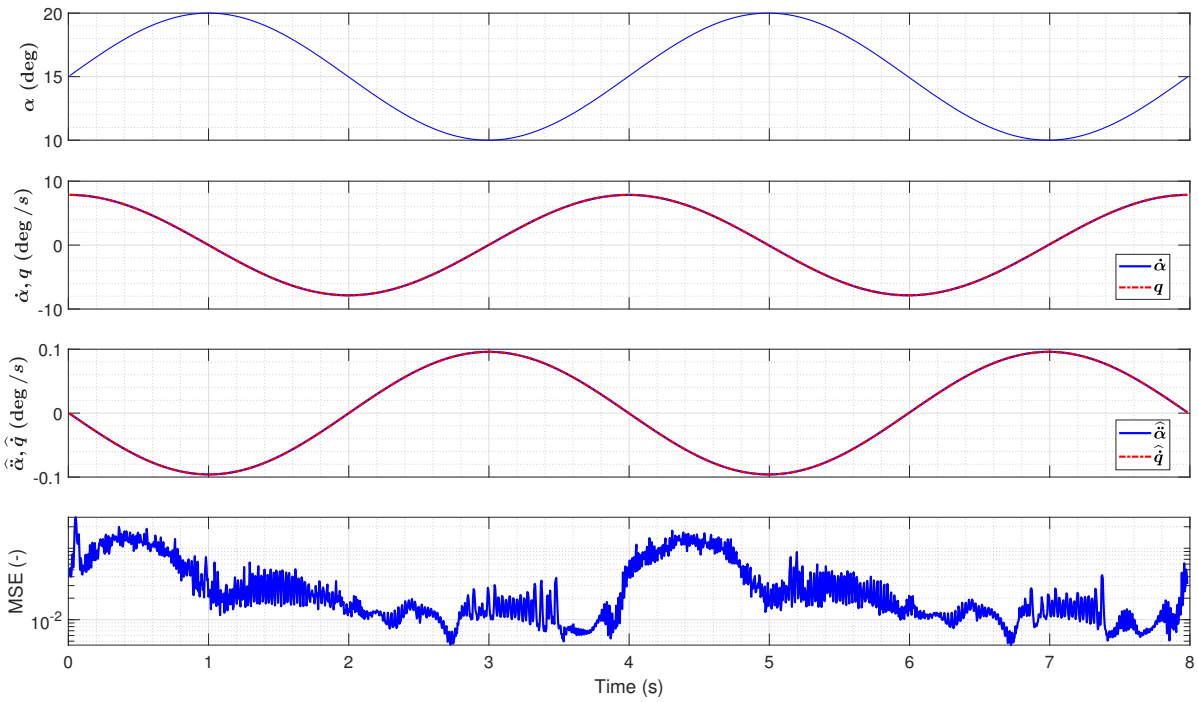
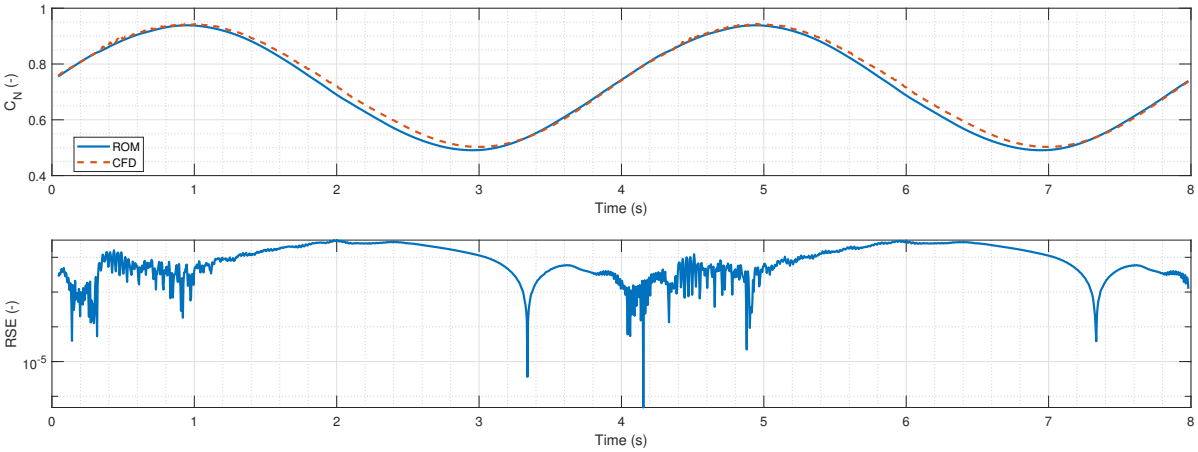
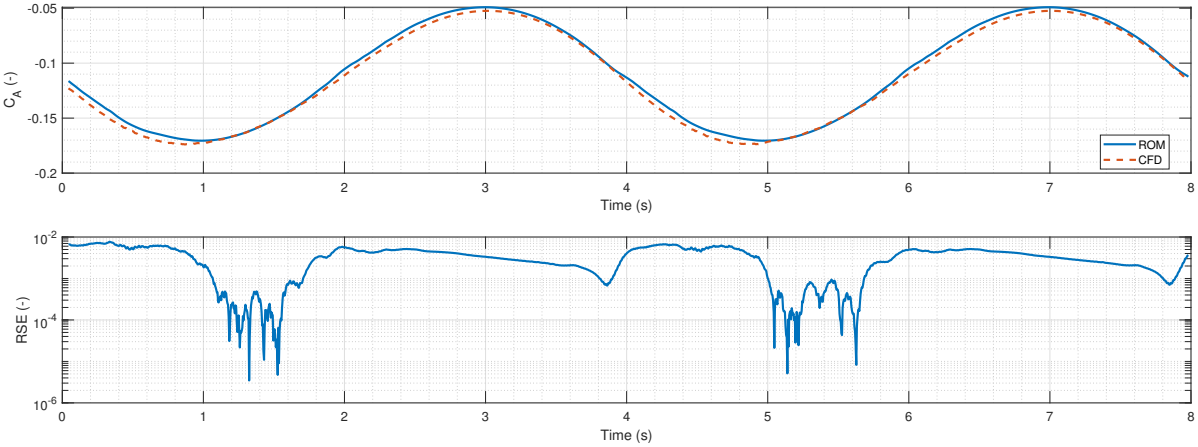


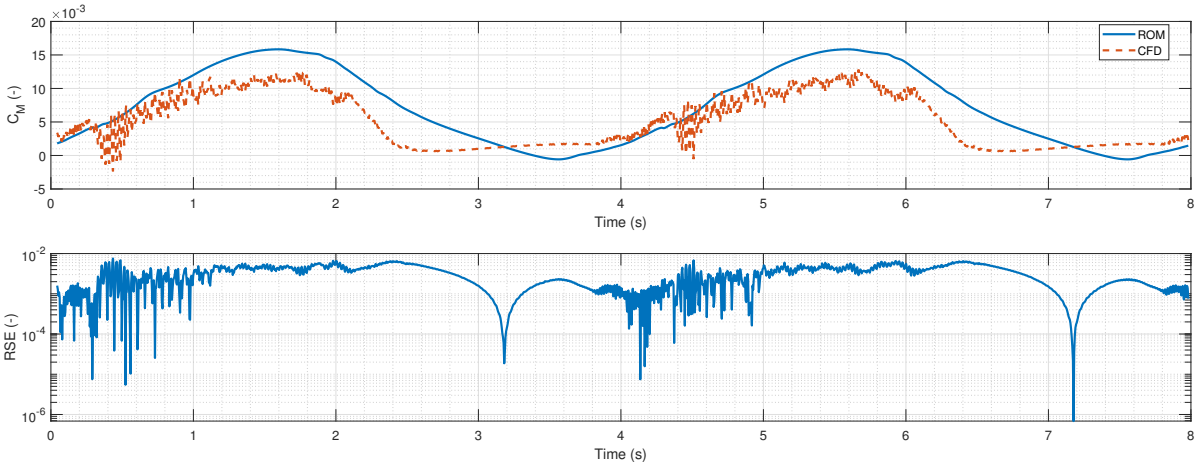
Figure 5.26: Neural network control inputs for pitch oscillation at $A_0 = 15$ (deg), $A = 5$ (deg), $f = 0.25$ (Hz).



(a) Comparison of CFD and ROM results of C_N as a function of time.



(b) Comparison of CFD and ROM results of C_A as a function of time.



(c) Comparison of CFD and ROM results of C_M as a function of time.

Figure 5.27: Comparison of CFD and ROM results of integral loads as a function of time for pitch oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$. The first 10 timesteps are cut off so the initialization of CFD is excluded. Due to abrupt fluctuations the motion is simulated over two periods of time ($2T$). Variables $\bar{x}$ and $\bar{y}$ denote the normalized dimensions of the aircraft.

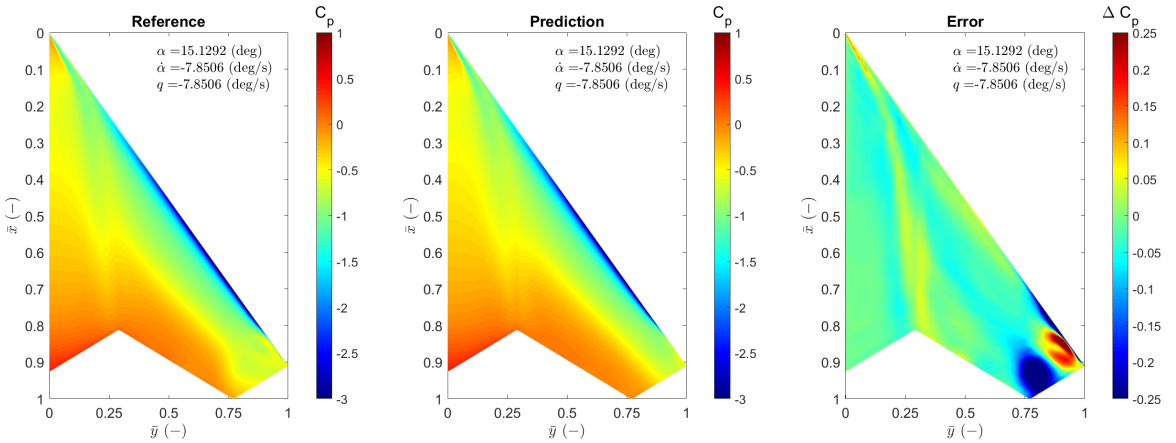


Figure 5.28: Comparison of CFD and ROM results of upper surface pressure distributions for pitch oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$; $t \approx 2.0(s)$.

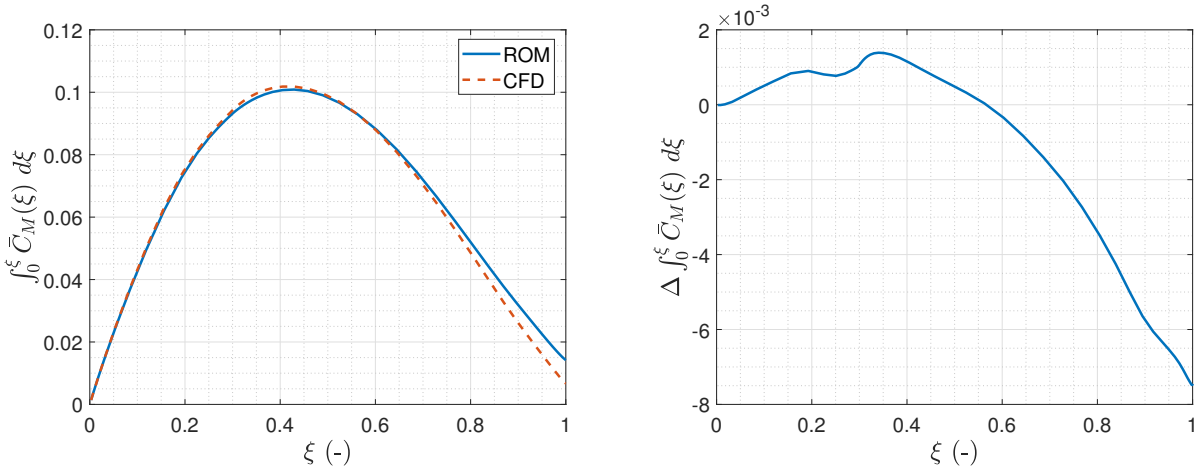
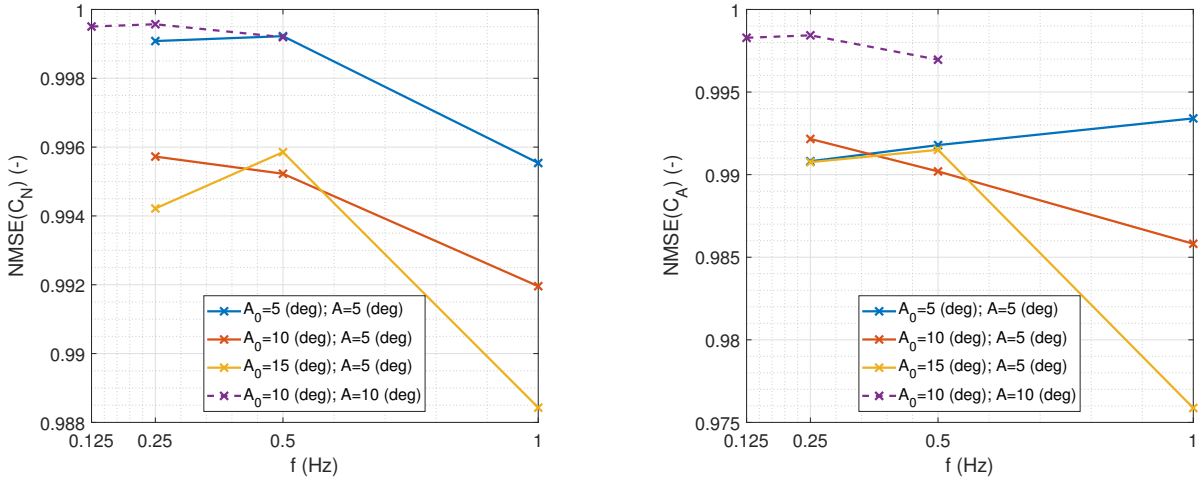


Figure 5.29: Comparison of accumulated pitching moment distributions as a function of relative distance travelled along the leading edge for pitch oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$; $t \approx 2.0(s)$.



(a) Normal force coefficient NMSE values for plunge oscillations at varying nominal values and amplitudes as a function of frequency.

(b) Axial force coefficient NMSE values for plunge oscillations at varying nominal values and amplitudes as a function of frequency.

Figure 5.30: Accuracy of force coefficients in terms of NMSE for plunge oscillations.

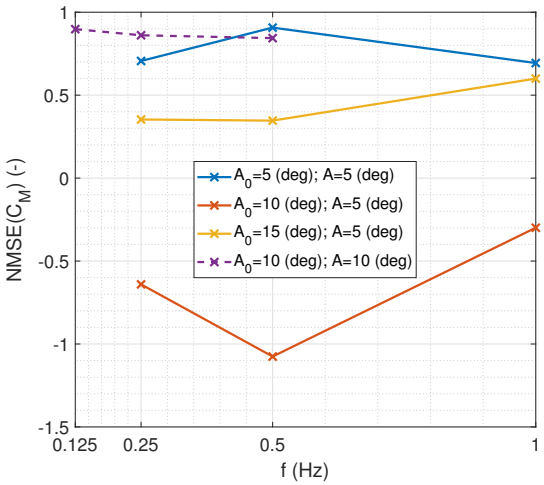


Figure 5.31: Pitching moment coefficient NMSE values for plunge oscillations at varying nominal values and amplitudes as a function of frequency.

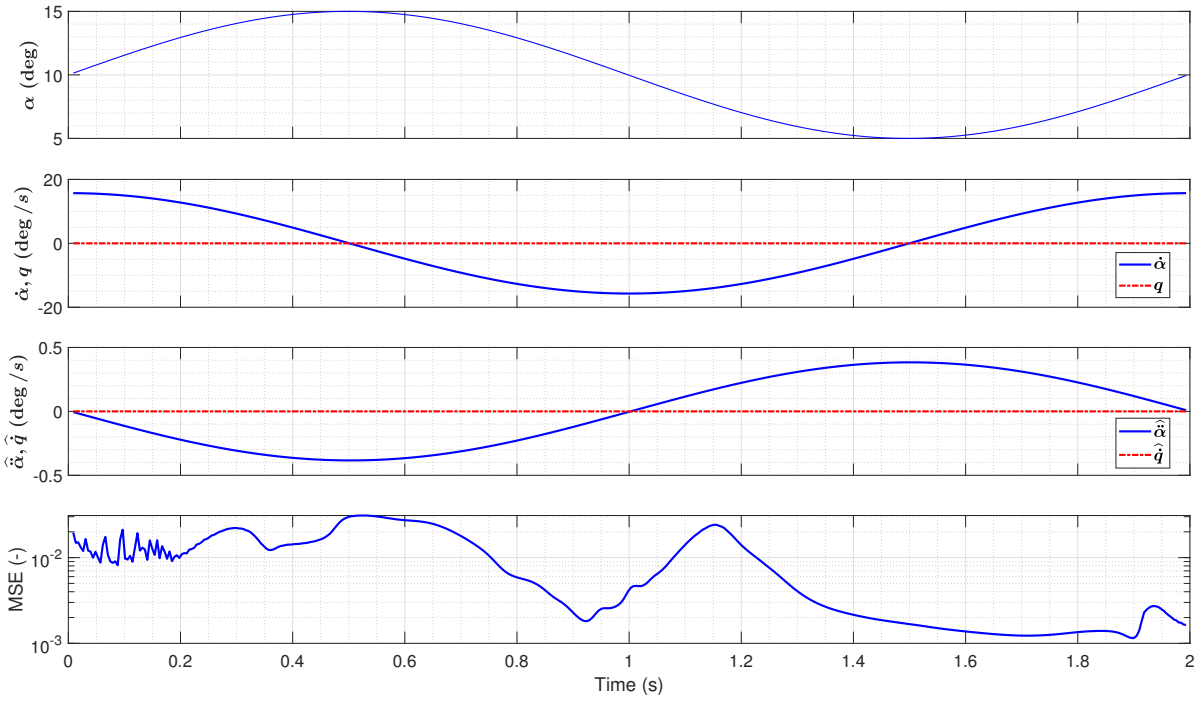
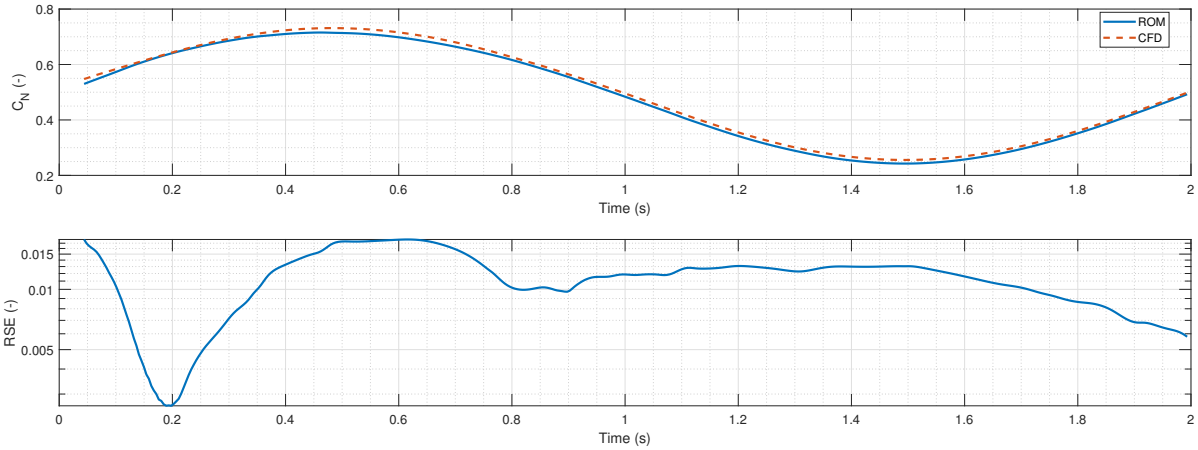
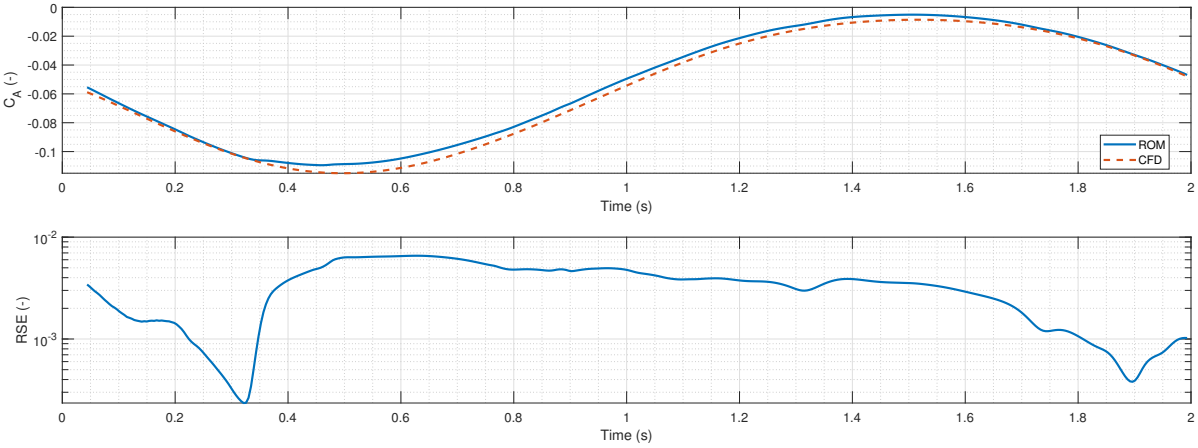


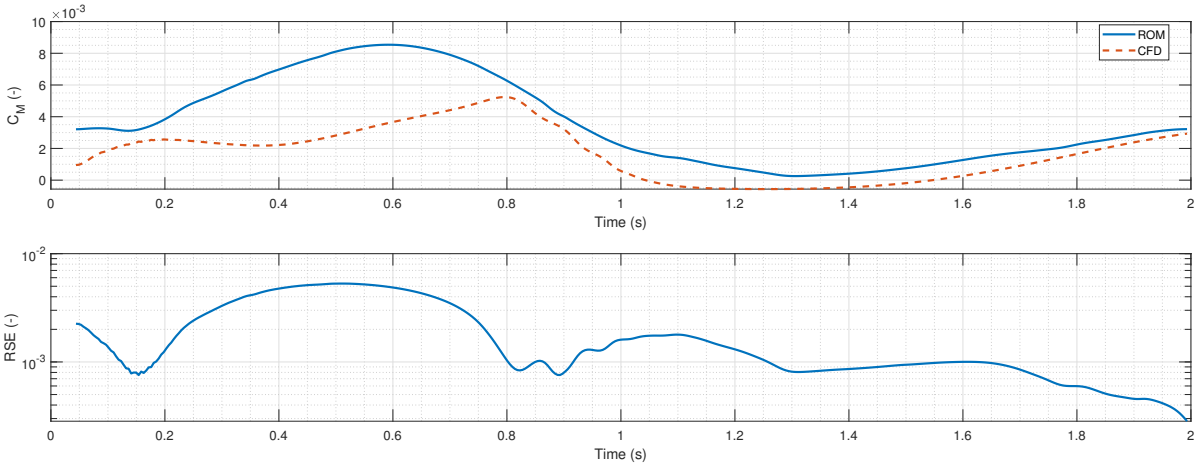
Figure 5.32: Neural network control inputs for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.



(a) Comparison of CFD and ROM results of C_N as a function of time.



(b) Comparison of CFD and ROM results of C_A as a function of time.



(c) Comparison of CFD and ROM results of C_M as a function of time.

Figure 5.33: Comparison of CFD and ROM results of integral loads as a function of time for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$. The first 20 timesteps are cut off so the initialization of CFD is excluded. Variables $\bar{x}$ and $\bar{y}$ denote the normalized dimensions of the aircraft.

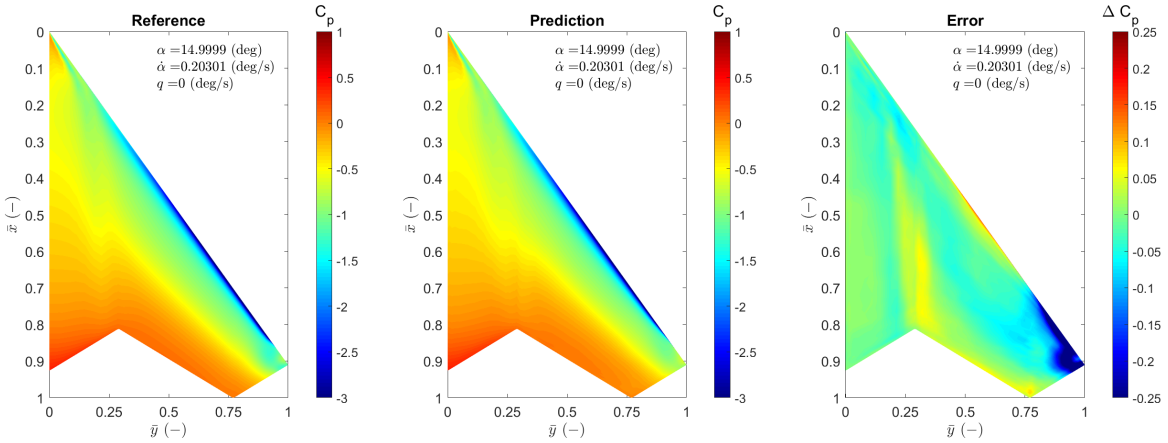


Figure 5.34: Comparison of CFD and ROM results of upper surface pressure distributions for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$; $t \approx 0.5(s)$. Variables $\bar{x}$ and $\bar{y}$ denote the normalized dimensions of the aircraft.

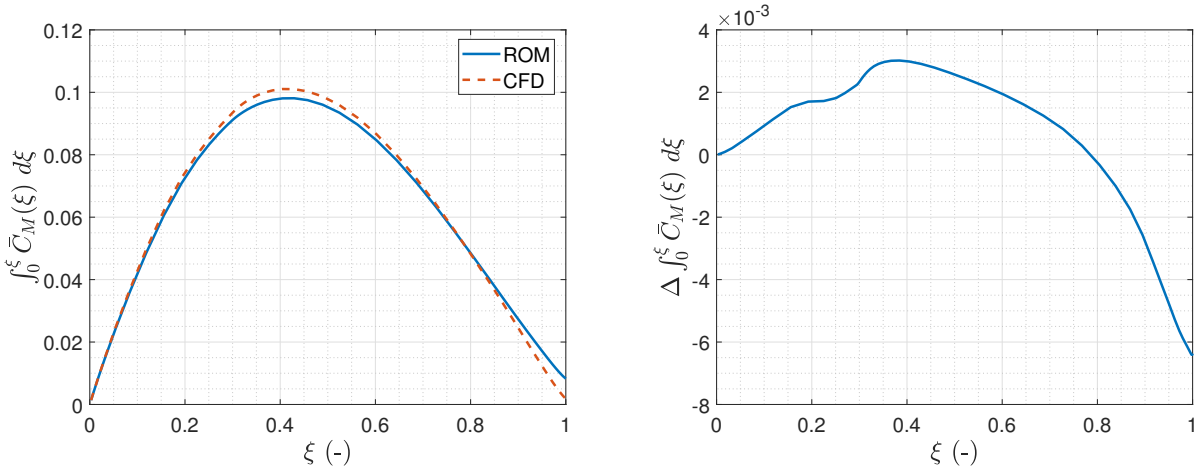
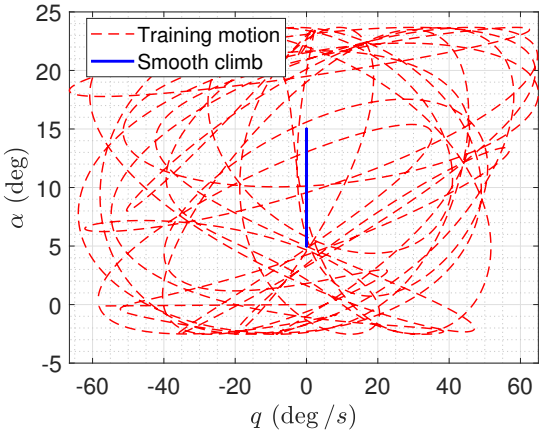


Figure 5.35: Comparison of accumulated pitching moment distributions as a function of relative distance travelled along the leading edge for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$; $t \approx 0.5(s)$.

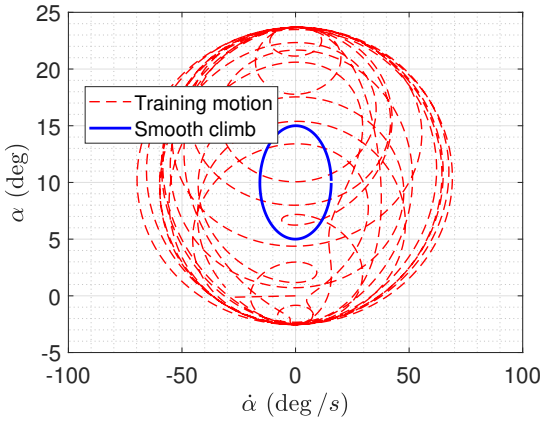
REMARKS ON HARMONIC MOTION ACCURACY

Force coefficient predictions are rather robust. However, the pitching moment coefficient is very sensitive to errors at amplitudes of 5 degrees. Pitch oscillations exhibit issues at low frequencies especially at higher angles of attack. Plunges on the other hand perform better at the two extremes and experience significant losses in accuracy at 10 degrees of AoA. Raising the amplitudes from 5 degrees to 10 degrees mitigates the issues to a certain extent.

The inaccuracies and their location can be motivated by training data regressor spaces. Plunging motions can be interpreted in the $\alpha - \dot{\alpha}$ plane from fig. 5.4b. Evidently, the vicinity of the $\alpha = 10(\text{deg})$ axis offers one of the poorest coverages in the observed regime. Consider for instance the example plunge oscillation from fig. 5.32 with $A = 5(\text{deg})$, $f = 0.5(Hz)$ at $A_0 = 10(\text{deg})$. Plotting the oscillation on the regressor space in fig. 5.36, it is no surprise the motion incurs large errors, as the motion is barely in the vicinity of any training motion sweeps. Applying a larger amplitude explains the improvements in moment coefficient prediction accuracy as more examples (combinations of α and $\dot{\alpha}$) are explored towards the extremes (0 and 20 degrees of AoA), hence pressure distribution predictions should be more accurate.



(a) Plunge oscillation with $A = 5(\text{deg})$, $f = 0.5(Hz)$ at $A_0 = 10(\text{deg})$ compared to the training maneuver in the q α plane.



(b) Plunge oscillation with $A = 5(\text{deg})$, $f = 0.5(Hz)$ at $A_0 = 10(\text{deg})$ compared to the training maneuver in the $\dot{\alpha}$ α plane.

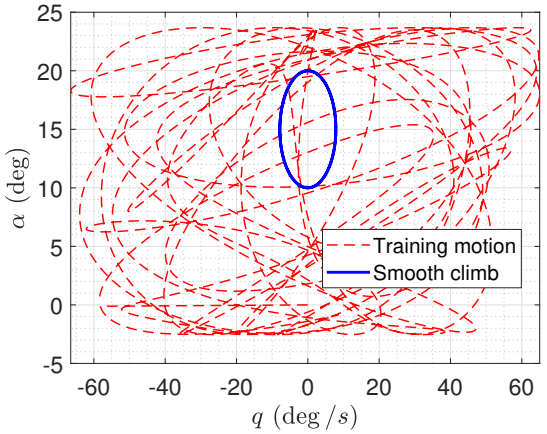
Figure 5.36: Plunge oscillation with $A = 5(\text{deg})$, $f = 0.5(Hz)$ at $A_0 = 10(\text{deg})$ compared to the training maneuver in the regressor space.

For pitching motions poorly covered areas are not so conspicuous. The problems of the $\alpha - \dot{\alpha}$ plane are still present, but for pitching motions the $\alpha - q$ plane is relevant, too. fig. 5.4a display somewhat better and more random space exploration for the observed pitch rates. However, there are indeed a few white patches in the upper right quarter plane around 15 degrees of AoA. Considering that the flow solution is possibly more complex at high AoA, those regions may reason the decreasing accuracy of the predictions. Plotting for instance the pitch oscillation with the worst pitching moment coefficient fit ($A = 5(\text{deg})$, $f = 0.25(Hz)$ at $A_0 = 15(\text{deg})$) over the training motion in fig. 5.37, it can be seen that motion indeed sweeps through unexplored domains of the regressor space. Raising the amplitude to 10 degrees, the $\alpha - q$ plane also enjoys denser coverage, which again can explain the improvements of 10 degree amplitude motions.

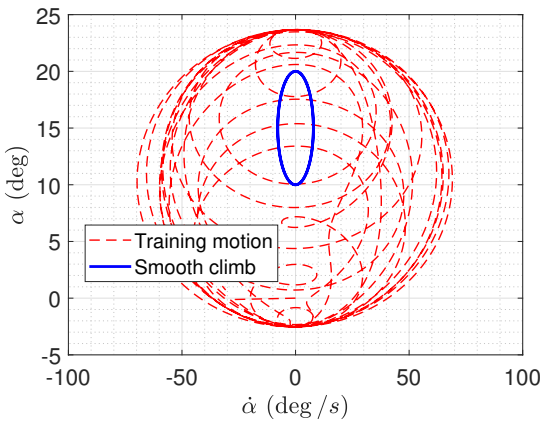
5.4. SPECIAL MANEUVERS: APPLYING THE MODEL TO NEW TYPES OF MOTIONS

Finally, two special maneuvers are considered that differ in types from the training maneuver, i.e. they are not harmonic oscillations. The first one is a piece-wise linear motion, regarded as a sharp pitch up-down maneuver, taken from Rooij and Cummings [65]. The accelerations are zero except for the instances when the otherwise constant rates are applied. The inputs of the motion are illustrated in fig. 5.38. The motion starts with steady conditions at an AoA of 15 degrees. Shortly after, a pitch rate of 20 degrees per second is introduced, seen as the step in derivatives. From that point on the incidence and pitch angles increase linearly, until the peak value is reached (roughly 20 degrees). After that, the rates are instantaneously negated and the AoA decreases towards the initial value linearly again. The additional motion variables of the sharp

h



(a) Pitch oscillation with $A = 5(\text{deg})$, $f = 0.25(Hz)$ at $A_0 = 15(\text{deg})$ compared to the training maneuver in the q α plane.



(b) Pitch oscillation with $A = 5(\text{deg})$, $f = 0.25(Hz)$ at $A_0 = 15(\text{deg})$ compared to the training maneuver in the $\dot{\alpha}$ α plane.

Figure 5.37: Pitch oscillation with $A = 5(\text{deg})$, $f = 0.25(Hz)$ at $A_0 = 15(\text{deg})$ compared to the training maneuver in the regressor space.

pitch up-down maneuver are shown in fig. 5.39. As can be seen in the figure, velocity is exchanged greatly between the two components, due to the enduring large value of pitch rate. Consequently the aircraft climbs considerably, i.e. it climbs almost half the distance it travels horizontally. However, due to the the angle of attack only reaches a value of 22 degrees. After that the inputs are inverted and the aircraft returns to its initial condition.

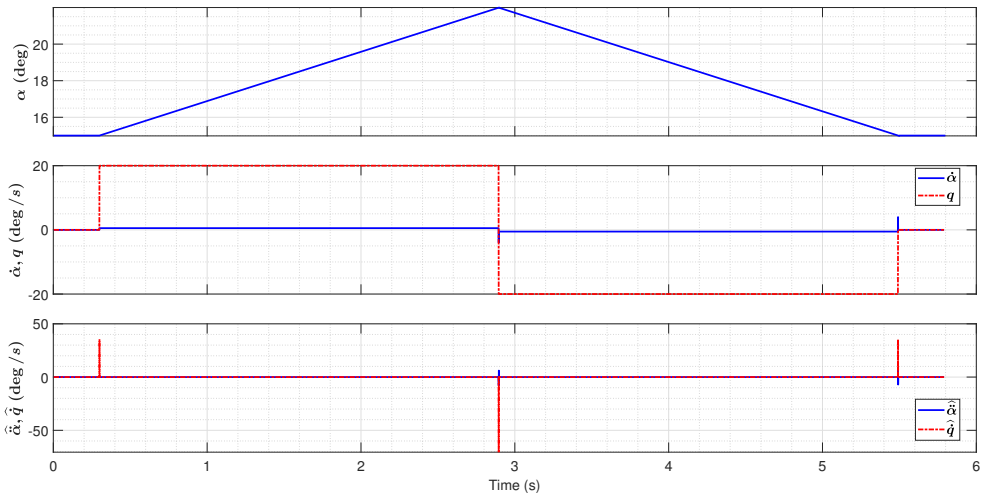


Figure 5.38: Sharp pitch up maneuver input signals.

The results of the sharp motion are provided in fig. 5.40. Even though this maneuver is not realistic, a few interesting points can be made. Firstly, since the rates of the instantaneous inputs are outside of the scope of the training maneuver, they generate sharp peaks in the integral loads. It shows that large extrapolations may cause predictions to blow up. Second, since no feedback or recurrent connections are employed in the networks, the transients around the changing inputs are not captured by the network. On the other hand, after the transients smooth out, the neural network aligns well with the CFD predictions. Interestingly, in the current maneuver, the pitching moment coefficient follows the tendencies of the CFD predictions relatively well. As usual, the normal force coefficient is very accurate. The axial force coefficient is also in good agreement with the CFD data until moderate AoA, it deviates from reference data only around the apex of

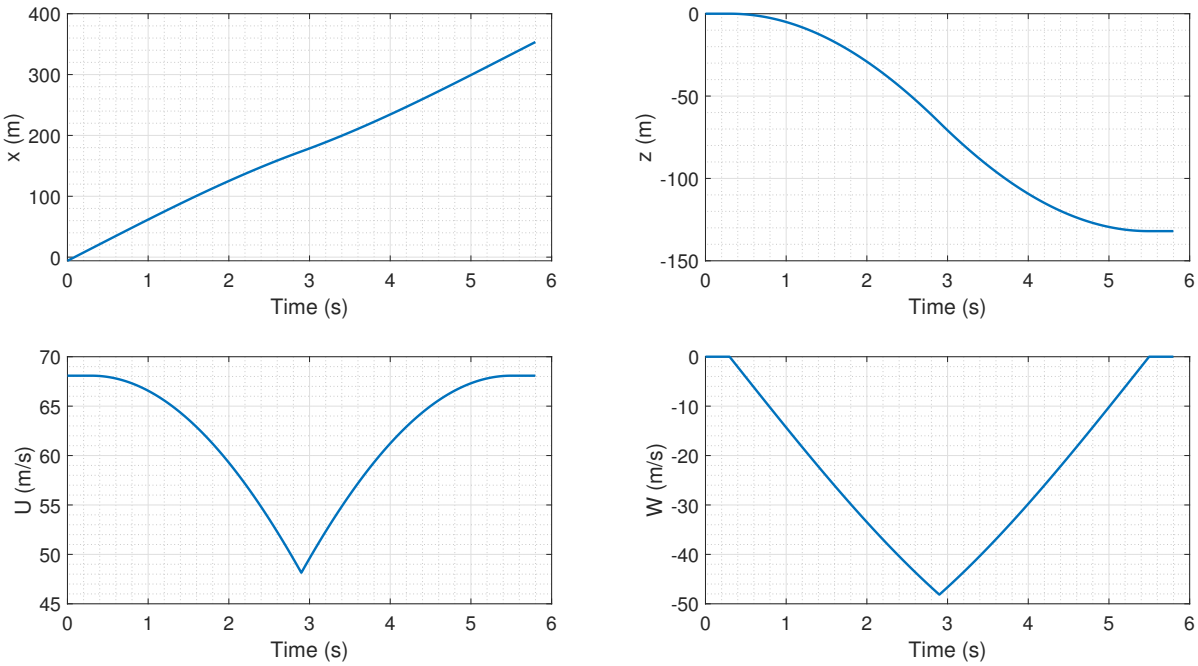


Figure 5.39: Sharp pitch up maneuver motion variables resolved in the inertial earth reference frame.

the motion. An important aspect of the maneuver is that the model can approximate significantly different types of maneuvers, that are otherwise not included in the training set. Nevertheless, extrapolation should be handled cautiously.

Another importance of the sharp maneuver is that it offers a chance for direct comparison with other type of reduced order models. The maneuver was simulated using another ROM based on indicial step responses [75] by Rooij and Cummings [65]. Similarly to the current model, the indicial ROM also utilized CFD simulations to infer system dynamics. The cost of establishing the indicial ROM is was also similar. Observing the results of the indicial ROM shows that prediction accuracy of the two approaches is comparable for normal force coefficient and pitching moment coefficient. However, an advantage of the current CNN based surrogate model is that it also able to approximate the axial force coefficient with good accuracy, contrary to the indicial ROM. Considering all that, the present methodology is competitive with at least on of the ROM techniques.

The second special maneuver is regarded as a smooth climbing maneuver Ketelaars [76]. This motion is realistic equivalent of the sharp pitch up-down maneuver frome above. The inputs of the motion are illustrated in fig. 5.42. The additional motion variables are shown in fig. 5.43. Instead of the sharp step inputs, the pitch rate of 20 degrees per second are applied and reversed gradually. Consequently, the aircraft will only realize a moderate pitch. Therefore the exchange in velocity among the components is less relevant and the airplane reaches an angle of attack of 20 degrees while its horizontal velocity decreases only by a small amount. The corresponding integral loads are shown in fig. 5.44.

The results are similar to the one observed for the harmonic oscillations. The normal and axial force coefficients are accurate throughout the motion, whereas the pitching moment coefficient deviates from CFD reference data. The errors may be caused by insufficient regressor space coverage again. The motion incurs an almost negligible angle of attack rate and slowly creeping pitch rate. Combined with high angle of attack the first part of the motion sweep through white patches of the regressor space, as shown in fig. 5.45. The rest of the motion is in the relative vicinity of the training maneuver sweeps. However, the magnitudes of the pitching moment coefficients are very small again, hence any small prediction errors result larger relative deviations. Over the other parts of the motion the pitching moment coefficient predictions follow CFD values with better accuracy.

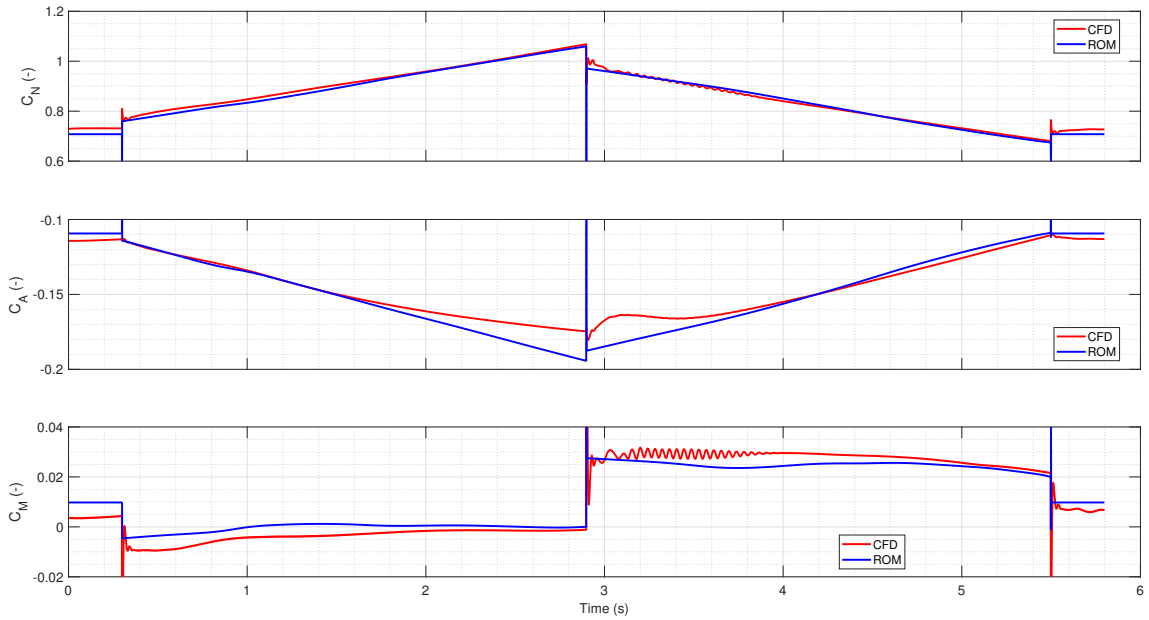


Figure 5.40: Comparison of CFD and ROM integral load coefficients for sharp pitch up maneuver.

5.5. MODEL PERFORMANCE: COSTS OF MODEL CONSTRUCTION AND APPLICATION

The essence of the surrogate model is that it is capable of predicting wing surface pressure distributions and corresponding loads much faster than conventional finite-volume methods do. The GPU-accelerated computations of neural networks allow for rapid evaluations of flowfields without the need of iterating single timesteps multiple times.

To compare the performance of the surrogate to CFD computations, the training maneuver is taken as reference. Although, simulation time of different maneuvers might differ using CFD solvers, the training maneuver being relatively complex serves a good example. The performance of the NN model is consistent and it shows little variance in performance when simulating different maneuvers. Moreover, if GPU acceleration and sufficiently powerful hardware are available, the execution time shows minor variation on different platforms. On the other hand, training greatly depends on the hardware. The on-clock-time costs of simulating the training maneuver are reported in Table 5.4 and the computing platforms are reported in Table 5.5.

Table 5.4: Comparison of computational costs incurred by CFD and surrogate model simulations, measured on-clock-time.

			Neural network	
Solver	ENSOLV	ROM	Primary	Secondary
Training	–	17 h 56 m	12 h 24 m	5 h 32 m
Complete simulation	9600 h / 120 h (@ 80 threads)	80 s	60 s	20 s
Average cost per time-step (w/o training)	6×10^6 ms / 76×10^3 ms (@ 80 threads)	15 ms	11 ms	4 ms

Table 5.5: Comparison of computational costs incurred by CFD and surrogate model simulations, measured on-clock-time.

Model	ENSOLV	Surrogate model
Platform (CPU)	NLR Computing Cluster @ 80 threads	Intel Xeon CPU E5-2690 v3 @ 2.60GHz
Platform (GPU)	–	Tesla k40m

The high simulation cost of 9600 hours reported in Table 5.4 is the accumulated time of the threads run

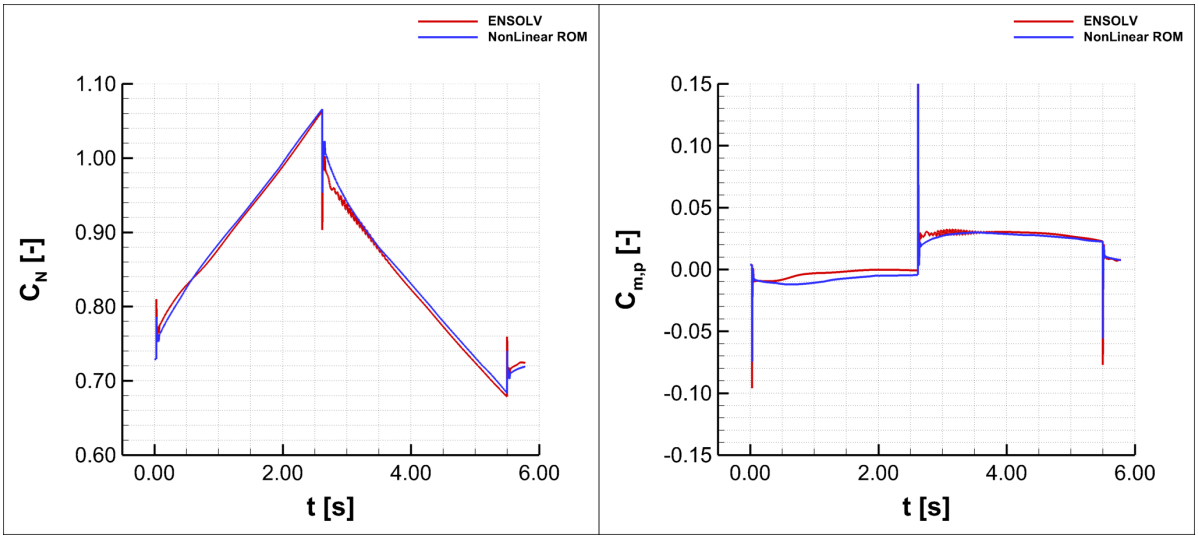


Figure 5.41: Results of indicial step response ROM for the sharp pitch up-down maneuver [65].

in parallel. The simulation was run on the computing cluster of NLR on 80 threads. In real on-clock-time, the simulation was finished in roughly 120 hours.

Network optimizations also incur relatively high costs, although, their overall contributions decrease as the trained networks allow for rapid simulations. If numerous maneuvers are to be simulated, their costs can almost be disregarded. If so, the surrogate model offers roughly 5000 times faster computational speed. On the other hand, not that in order to construct the CNN based surrogate model, first it is necessary to gather CFD data. It can be considered as part of the training cost. Similarly to the cost of network optimization, its relative value will decrease as the surrogate model is applied to more and more simulations.

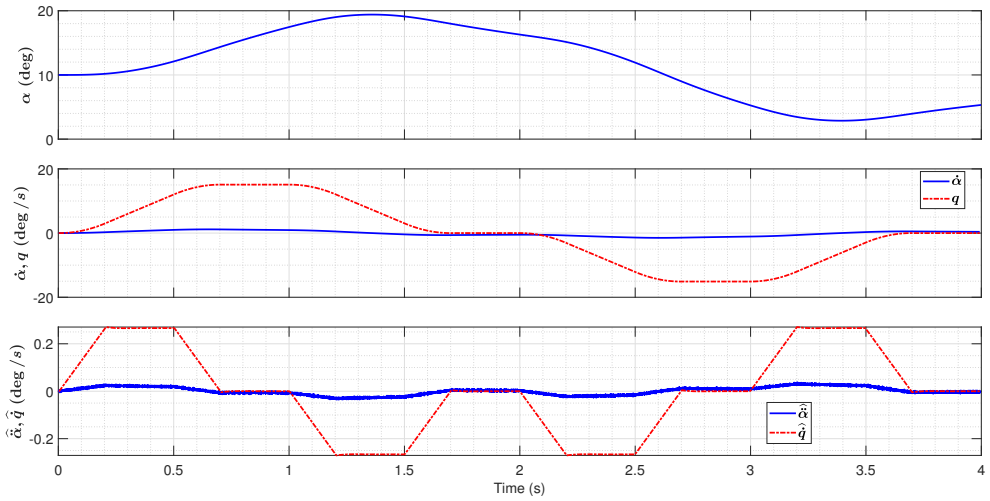


Figure 5.42: Smooth climbing maneuver input signals

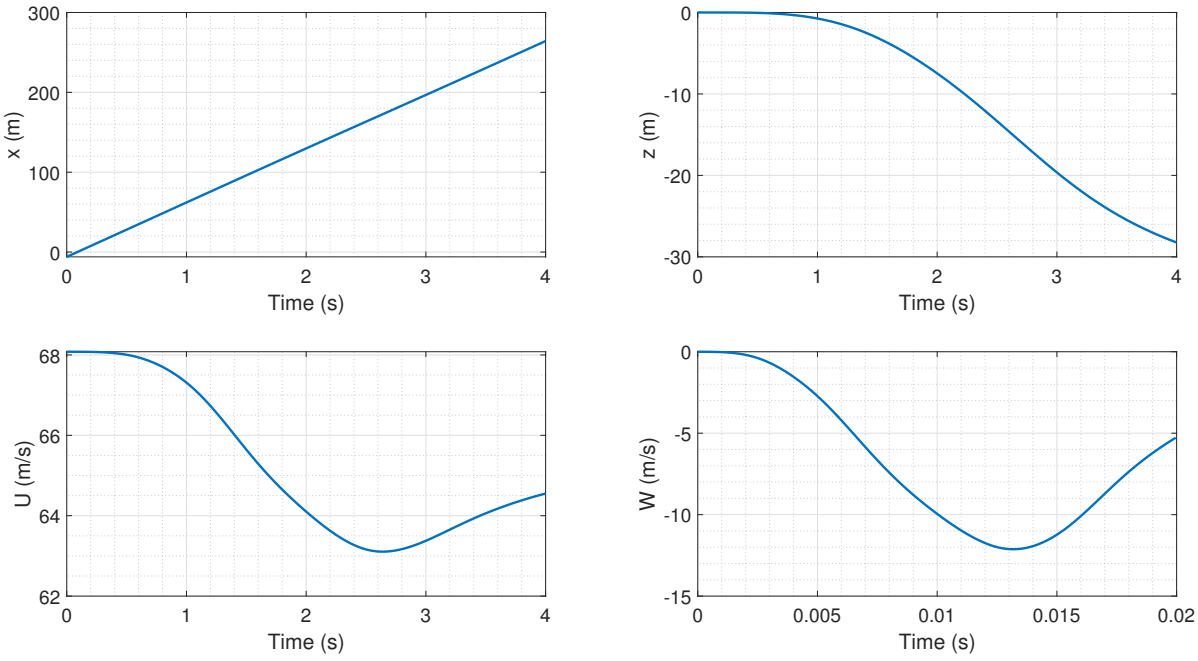


Figure 5.43: Smooth climbing maneuver motion variables resolved in the inertial earth reference frame.

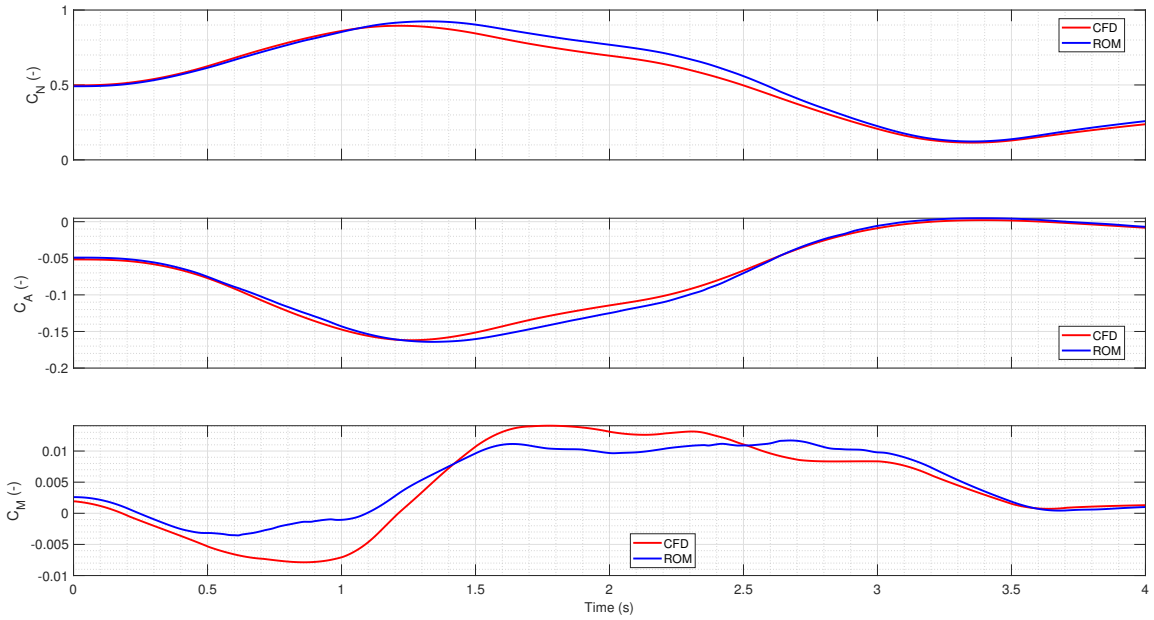
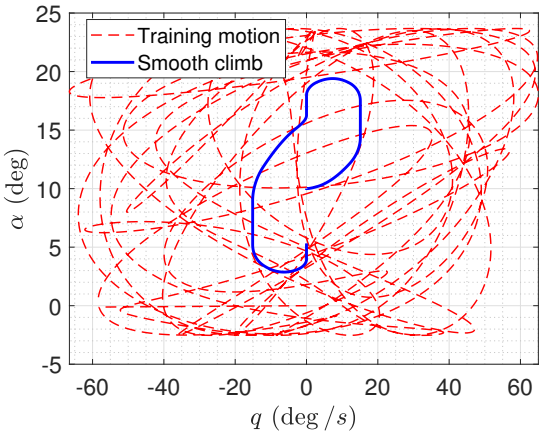
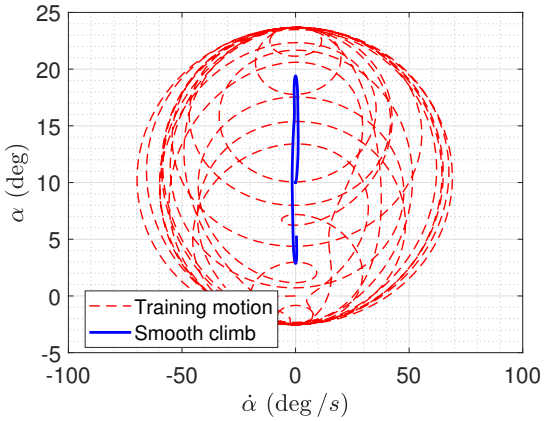


Figure 5.44: Comparison of CFD and ROM integral load coefficients for smooth climbing maneuver.



(a) Smooth climb motion compared to the training maneuver in the q α plane.



(b) Smooth climb motion compared to the training maneuver in the $\dot{\alpha}$ α plane.

Figure 5.45: Smooth climb motion compared to the training maneuver in the regressor space.

5.6. SUMMARY OF SIMULATIONS AND RESULTS

The input signals for incidence and pitch angles of the training maneuver are constructed by 25-25 harmonics with frequencies from 0.01 to 1 Hz. The lower bound of 0.01 Hz is desired to represent quasi-steady conditions [72]. The upper bound is a simple design choice. Both signals have uniform power distributions among the harmonics. Incidence and pitch excitations are decided to have equivalent amplitude for convenience. The common amplitude is defined as 12.5 degrees. The input of the incidence angle is centered around 10 degrees, so the prescribed range of incidence angles from 0 to 20 degrees (Table 4.3) is covered sufficiently. Lacking any requirements in that regard, the input signal of the pitch angle is centered around 0 degrees. Given the frequency range, the duration of the excitations is 25 seconds. The signals are optimized so they have minimum relative peak factors [68] and they start from their nominal values. The relative peak factors of the incidence and pitch angle inputs are 1.05 (-) and 1.02 (-) respectively. The cross correlation at zero lag between the signal is 0.05 (-). The signals achieve overall good coverages over the regressor spaces, although, as a drawback of the Schroeder sweeps, their powers are biased towards the perimeters.

Using the optimized input signals, both primary and secondary network are optimized over 5e05 iterations. The final values of moving exponential averages of the mean square errors are around $1e-04$ and $1e-05$ for the primary and secondary networks, respectively. The trained network have great accuracy over the test data set.

The performance of the model is evaluated using three different test cases. The first case considers a set of steady conditions from 0 to 20 degrees of angles of attack, going in a step of two degrees. The resulting force coefficients are in great agreement with the CFD predictions. For pitching moment coefficients, from low to moderate (0-10 degrees) angles of attack, model predictions are still accurate. However, from moderate to high angles of attack (10-20), pitching moment coefficients exhibit large errors compared to the CFD reference data. In the search for error sources, the surface pressure distributions are plotted at 14 and 20 degrees. Lower surfaces are solved with great accuracy. Concurrently, upper surfaces suffer from errors. The errors are seemingly concentrated towards the wingtips. Depending on the boundary conditions, the primary network may fail to (sufficiently) capture several phenomena over the upper surface. The strength of energetic vortices might be inaccurate, it might fail to predict the presence of smaller less energetic vertices farther outboard, or misjudges the strength of suction along and slightly behind the leading edge. However, errors in the pitching moment are also prominent when most of the upper surface agrees well with the CFD reference. To find out what actually causes the errors the pitching moments computed from the pressure values are accumulated along the leading edge. The investigation indicates two different sources of errors, for steady conditions at least. When the pressures are resolved inaccurately the actual corresponding error in the pitching moment should be far smaller then anticipated by the secondary network. Hence, it is concluded that among certain conditions the secondary network misjudges its inputs, and falsely identifies larger pitching moment coefficients then necessary. It is assumed that this type of error is due to insufficient representations of steady conditions among the training data. When the pressures are resolved accurately, the error among the accumulated pitching moment contributions account for the errors observed in the neural network prediction. In that case it is concluded that the neural network accurately predicts the pitching moment coefficient corresponding to the more accurate pressure distribution prediction. However, wingtips seemingly have a great contribution to the overall pitching moment and the neural network is very sensitive to the errors at the wingtip. A small patches of errors can cause significant difference it the pitching moment coefficient.

The second test case considers a set of harmonic pitch and plunge oscillations. The motions are commanded by sinusoidal angle of attack inputs. The remaining motion variables are defined by the boundary conditions. For the harmonic motions three different nominal values are considered, in particular 5,10 and 15 degrees of angles of attack. At 5 and 15 degrees three different frequencies are tested, 1, 0.5 and 0.25 Hz, all with an amplitude of 5 degrees. At 10 degrees, the same three frequencies are test again with an amplitude of 5 degrees. However, and additional set of frequencies is also considered, 0.5, 0.25 and 0.125 Hz, with an amplitude of 10 degrees. In total 24 oscillations are considered, 12 for pitch and 12 for plunge oscillations. After simulating the maneuvers the normalized mean square error [74] of the integral load coefficients are computed. For each oscillation a single error value is calculated for each integral coefficient representing the goodness of the fit of the predicted results. The results show that integral force coefficients usually enjoy good and accurate fits. Concurrently, pitching moment coefficients exhibit great variance, depending on the amplitude of the oscillation, the nominal angle of attack and on the frequency of the oscillation. Whilst oscillations with 5 degrees of amplitudes suffer greatly from errors and variance, oscillations with 10 degrees of amplitude preserve good fits throughout all conditions are also more accurate than the others. For pitch os-

cillations it is concluded that decreasing frequencies and increasing nominal values of angles of attack have adverse effects on prediction accuracy goodness of fit. The adverse effects intensify towards the extremes, i.e. 0.25 Hz and 15 degrees of nominal angle of attack. For plunge oscillations the worst fits and accuracy are recorded at 0.5 Hz and at 10 degrees of nominal angle of attack. Recalling the input signals of the training maneuver it is concluded that the performances of the simulations are related to the performance of training motion in terms of covering the regressor spaces. Actually, the worst fits and accuracy are achieved among conditions where the training maneuvers offers poorest coverages. Consequently, pressure field predictions suffer from errors. In return the pitching moment predictions become inaccurate too. To account for the primary source of data in the pitching moment coefficients. i.e. whether it is the primary or the secondary network that causes most of the issues, the pitching moment contributions are accumulated once again for a few examples. It is concluded from those examples that in case of the pitch oscillations the errors in the pressure field predictions do account for the error in the pitching moment prediction. Hence it is believed that the main source of the errors in the pitching moment predictions is the primary network.

The third test consists of two special maneuvers. The first is a more unrealistic sharp pitch up-down maneuver in which pitch rates are applied as instantaneous step inputs. Consequently, the angle of attack is basically a piece-wise linear signal. The motion starts with pitching up at constant rate until an incidence angle of 20 degrees is reached. Then, the pitch rate is reversed, again instantaneously, causing the angle of attack to decrease linearly. Performing the maneuver in the surrogate model yields accurate results that align well with reference CFD data. However, since the network only uses instantaneous inputs and lacks any recurrent or feedback connections it cannot reproduce the exact transients seen after the step inputs. Nevertheless, the simulation shows that the network is able to approximate results of maneuvers that are fundamentally different from the training motion. Additionally, the sharp pitch up-down maneuver allows to compare the devised model with another reduced order model based on indicial step responses [65]. Importantly, from the results of the two methods it can be concluded that the accuracy of the current model is comparable with other type reduced order model. The second special maneuver is a revised, realistic version of the first one where the rates are applied and revoked gradually. The model shows similar tendencies in the predictions as for the harmonic oscillations. It is concluded that it might be insufficient regressor space coverages that causes errors at high angle of attack conditions. Concurrently, the magnitudes of the pitching moment coefficients are very small, hence any small prediction errors result larger relative deviations. At regions that visit better explored regressor space domains the errors are mitigated for the pitching moment as well.

Computational efficiency of the the devised model is measured over the training maneuver and compared against the reference CFD simulation. Considering solely performing single simulations, the surrogate model is able to generate results in roughly 15 milliseconds for a single time-step, if necessary hardware support is available. Then, the model is able to simulate the complete training maneuver in roughly 80 seconds. In contrast, the training maneuver was solved approximately over 120 hours, running computations on 80 threads. With the given resources, the neural network based surrogate model is more than 5000 times faster the original CFD simulation. Additionally, whilst the costs of a CFD simulation might vary among different problems, the trained network shows consistent performance over many different examples. Moreover, if sufficient GPU-acceleration is available, the performance of the model is also consistent over different hardware. However, in order to construct the model, first the CFD simulation of the training maneuver is necessary, then, the networks must be trained as well. These procedures all add up to the complete cost. Nonetheless, as the model is employed over many examples and simulations the relative cost of constricting the model will decrease.

.

6

CONCLUSION & RECOMMENDATIONS

6.1. MAJOR CONCLUSION

In the current thesis a convolution neural network based surrogate model was devised for unsteady nonlinear aerodynamic simulations. The proposed method utilizes convolutional neural networks to map aircraft geometrical features into surface pressure distributions and pressure distributions into integral aerodynamic loads. The motivation was to provide computationally efficient accurate flow predictions that could substitute expansive CFD simulations, to promote aircraft control design and to enhance performance and structural analysis. A reduced-order CFD model was implemented as the computational domain, containing a filtered aircraft surface mesh. Pressure field predictions for the model are obtained via a deep encoding-decoding architecture, whilst integral aerodynamic load coefficients were derived by an additional encoding architecture. Replacing simple convolution layers with advanced gated residual blocks allowed to train the networks on relatively small datasets containing less than 6000 examples. Utilizing GPU-acceleration for neural network computations, the trained model was able to execute simulations roughly 5000 times faster than the employed CFD solver. Among favourable circumstances the model was able to retain high accuracy for all integral load variables and predict surface pressure distributions with small errors.

System behaviour was inferred from a single training maneuver in which the dynamic system was excited by multisine signals. Trained network performance was assessed via steady and dynamic motions. Steady cases considered a set of incidence angles whereas the dynamic motions investigated system performance over a set of harmonic pitch and plunge oscillations. The experiments indicated robust performance for normal and axial force coefficient predictions regardless of motion variables. Model predictions correlated well with CFD predictions. In contrast, pitching moment coefficient predictions were sensitive to input variables in terms of flow angles and motion frequencies. For input combinations from poorly explored regions of the regressor space, high errors were incurred.

Pressure field predictions on the lower surfaces proved to be easy, and generally high accuracy was attained. Upper surface predictions lost accuracy at higher angles of attack. Whilst inboard sections could still be resolved accurately, relevant flow phenomena towards the wingtips were missed. For instance, the model failed to capture pressure recovery, enhanced suction or fluctuations and possible low-energy vertices that occurred at the outer portions of the wings.

It was assumed that the inaccuracy of pressure predictions towards the outer portions of the wings were a major source of the pitching moment errors. Accumulated pitching moment distributions showed that the errors caused by missed phenomena at the wingtips can be accounted for the errors in pitching moment coefficients in many cases. Even when large portions of the wings were resolved accurately, small erroneous patches towards the wingtip were enough to cause considerable differences in the pitching moment.

Despite the faced issued, the model showed that it was applicable for motions of different type than the training maneuver. Besides, the different types of motions can be approximated with good accuracy. From a reduced order modelling aspect, the costs of building and running the model and its accuracy is comparable to other ROM techniques or may even surpass them. In light of all findings, the primary goals of the project are considered as accomplished.

6.2. MODEL LIMITATIONS

The current model was devised as a proof of concept and according to that, it only considered a simple setup. Only 2D planar, symmetric motions are investigated for which flow and ambient conditions are constant. Additionally, throughout model development, any sort of recurrent components were removed from the networks as those incurred adverse effects, e.g. diverging errors, that could not addressed effectively. Hence, the final architectures were dependent only on instantaneous variables. Consequently, the model had no real capability of capturing time history effects in the flow. Finally, the model relied on a single training maneuver that offered somewhat poor coverages for slower motions.

6.3. RECOMMENDATIONS FOR FUTURE WORK

Future research first should address model limitations. It is recommended to consider another, or additional training maneuvers to further assess and possibly exploit model capabilities. At the same time, flight conditions should be extended to investigate the generality of the method. Varying altitude, Mach number are of great interest in that regard, besides, 6 degrees of freedom motions could be considered as well.

From neural networks perspective incorporation of memory effects and temporal dependencies can be of great interest. Being successful, such a model could potentially capture small scale fluctuations that are missed in the current model. Furthermore, inclusion of additional flow variables, e.g. friction coefficient, could be considered, that could provide even better insight to flow behaviour around the wings.

Finally, considering the potentials of the model, i.e. its computational efficiency, the proposed model could be incorporated in a complete flight dynamics model to assess real time simulation capabilities of the networks.

A

HARMONIC EXCITATION RESULTS

A.1. PITCH OSCILLATIONS

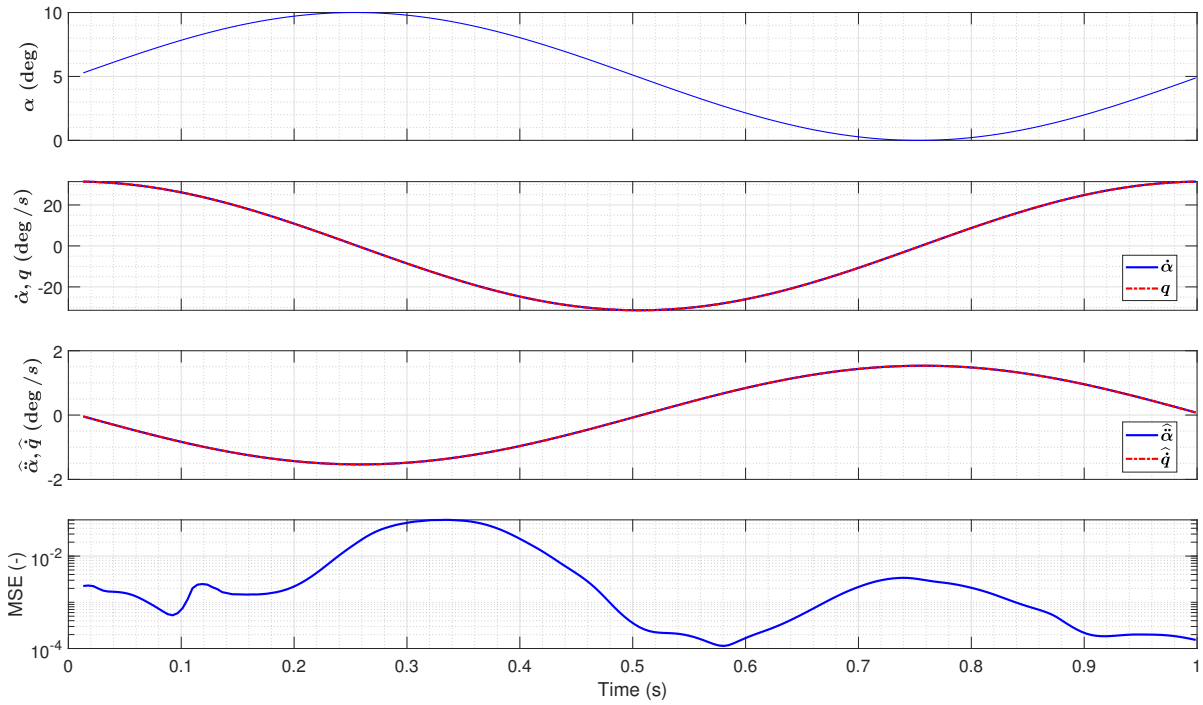


Figure A.1: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$.

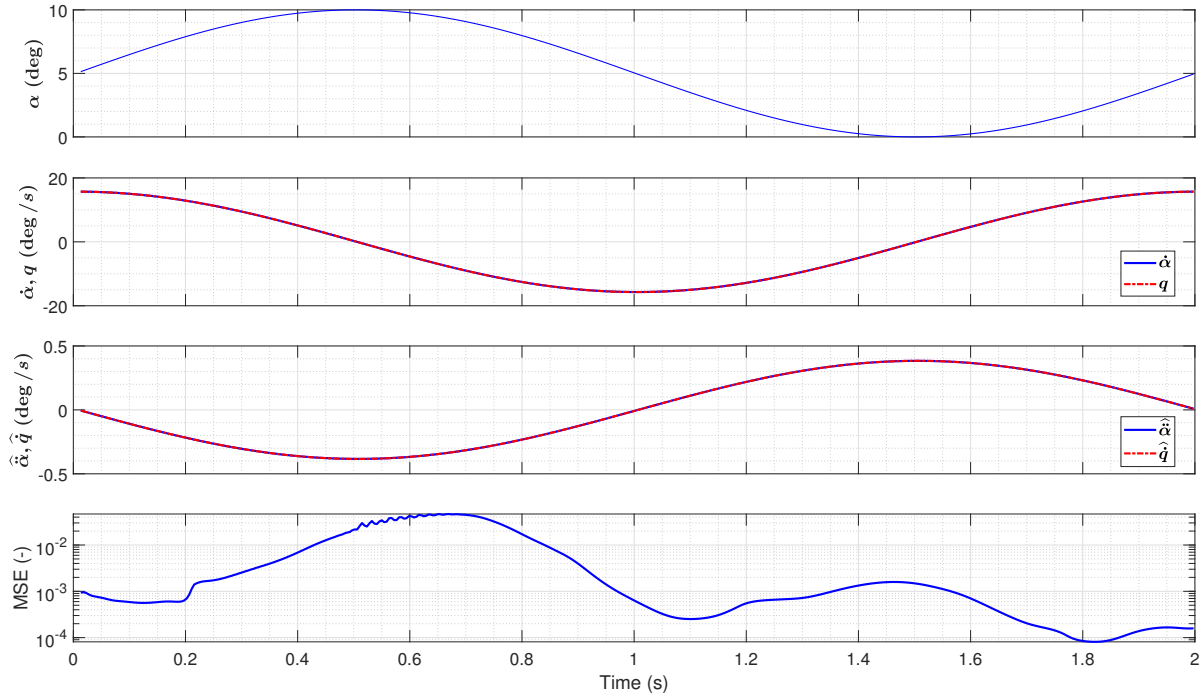


Figure A.2: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.

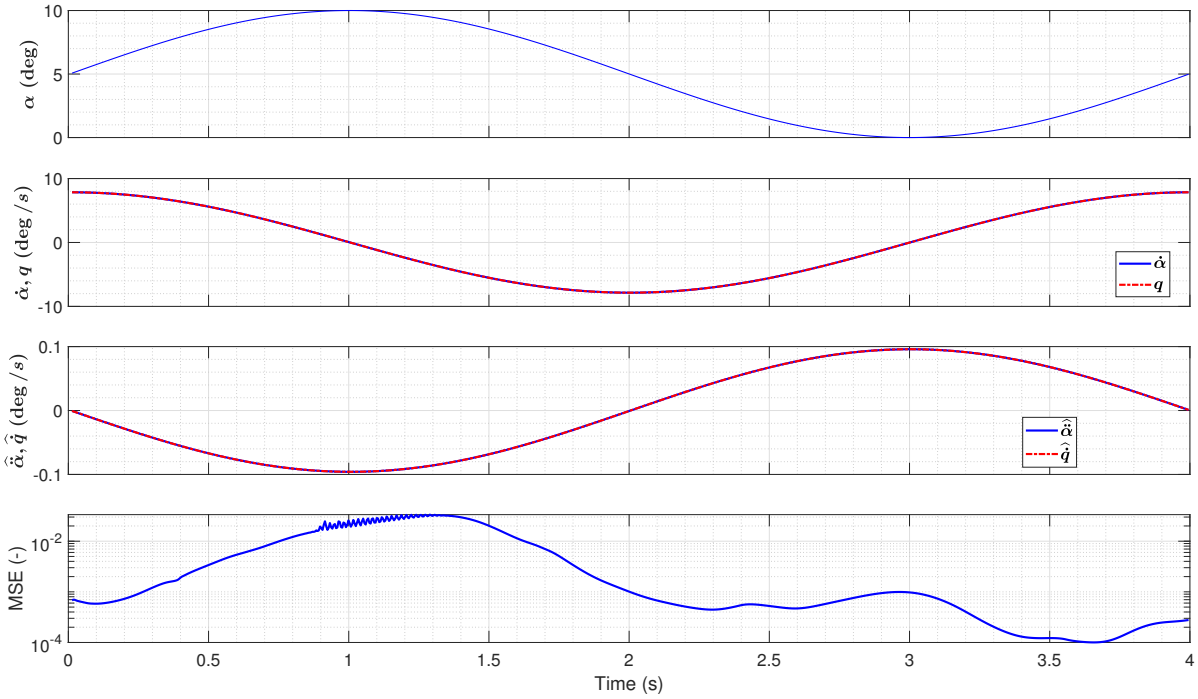


Figure A.3: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$.

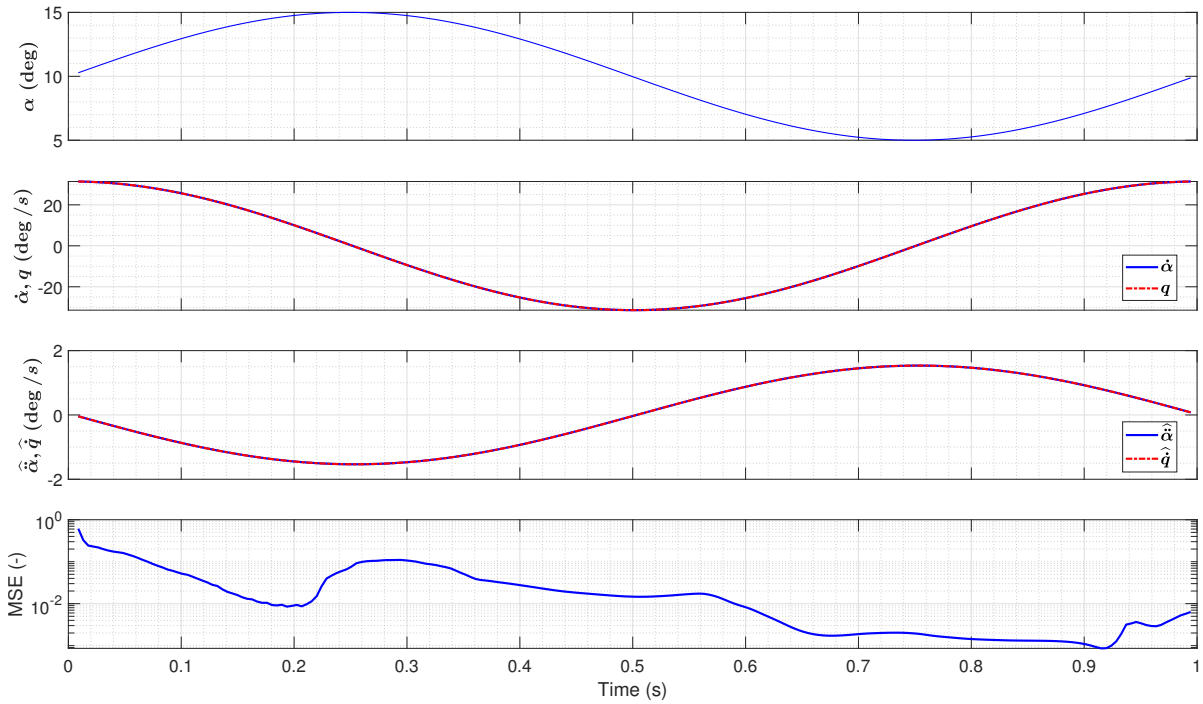


Figure A.4: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$.

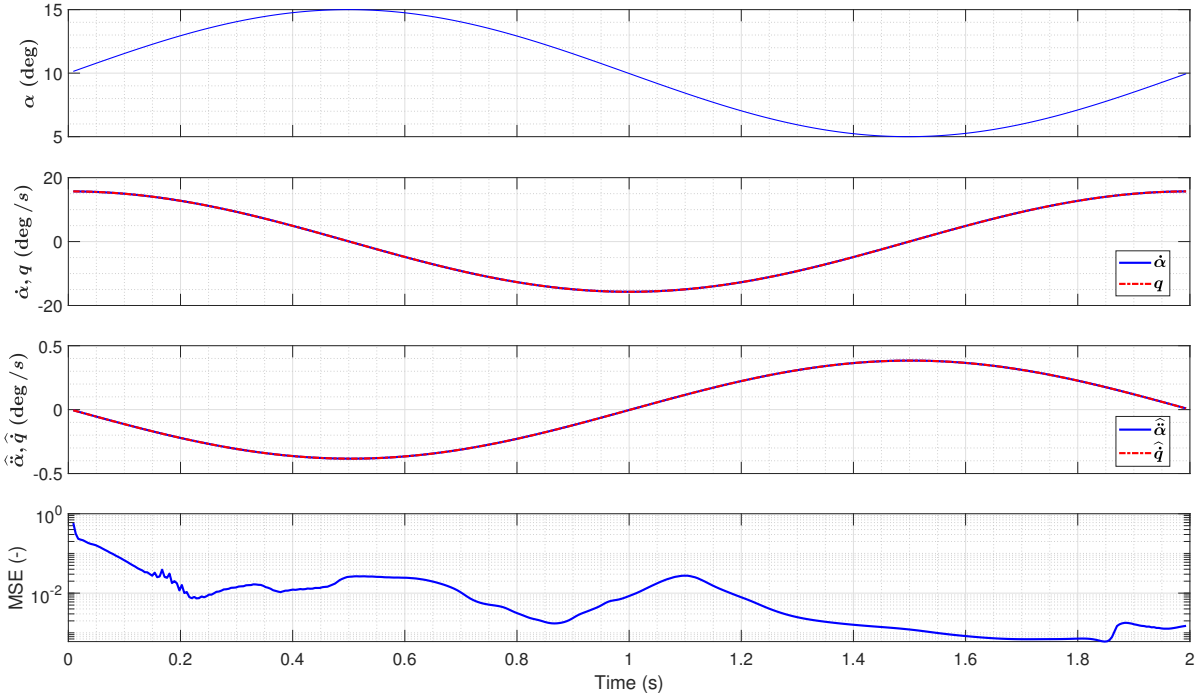


Figure A.5: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.

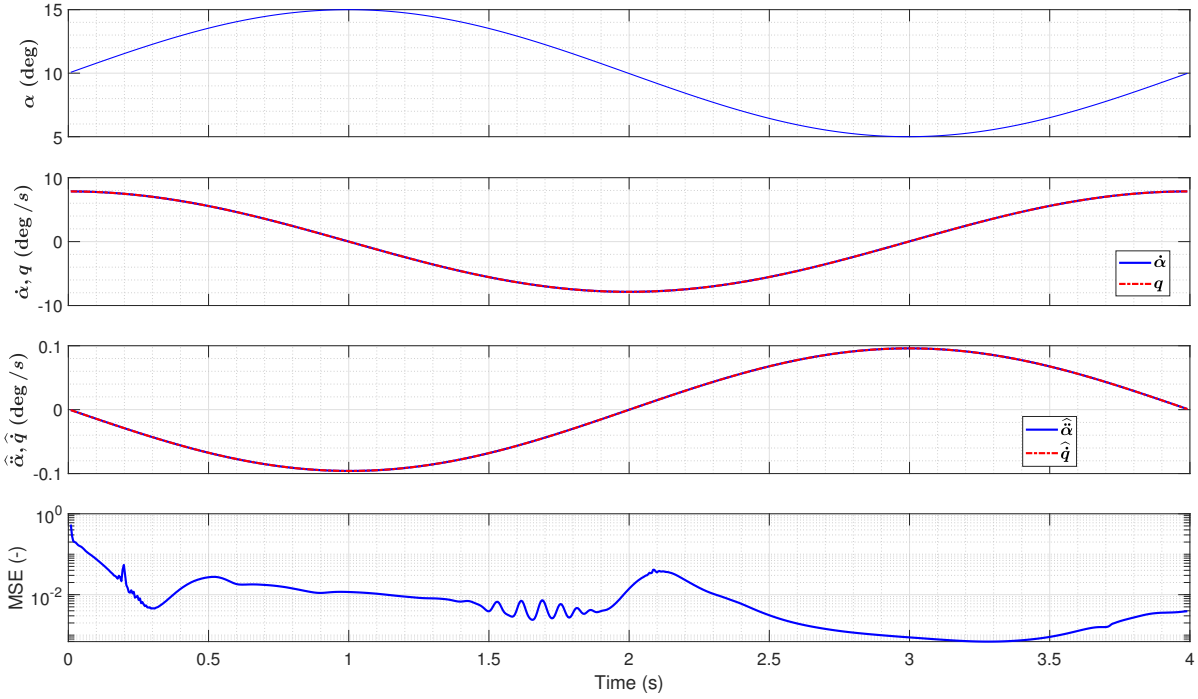


Figure A.6: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$.

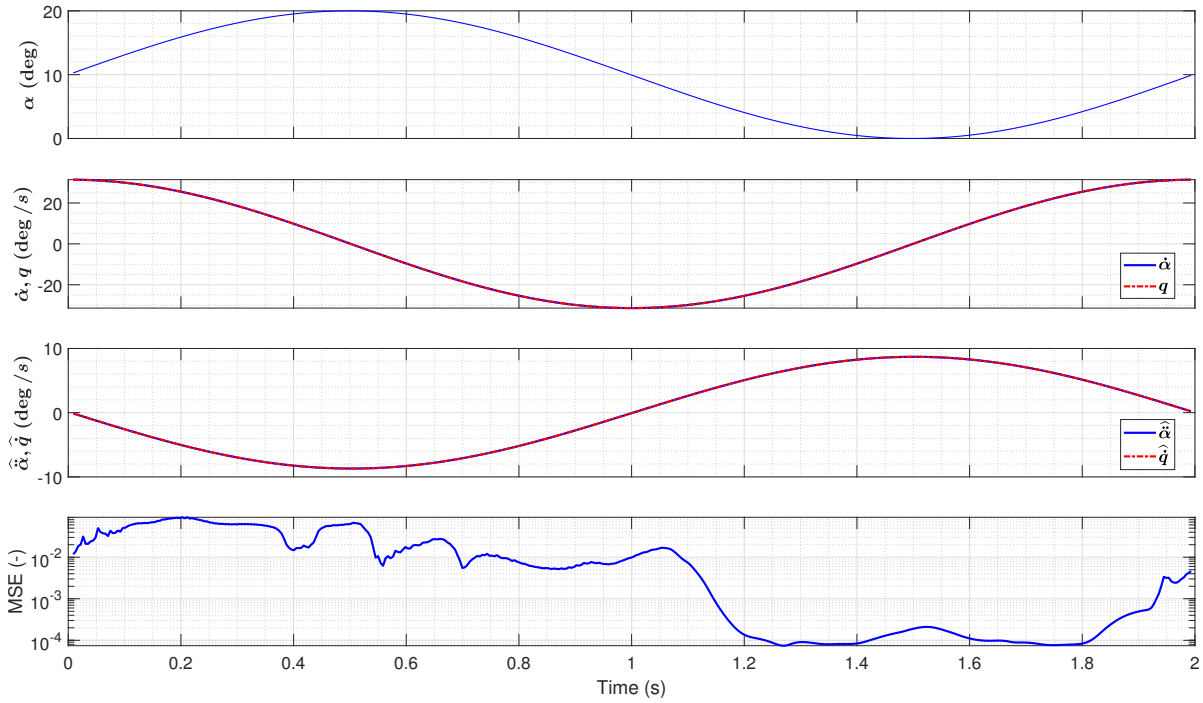


Figure A.7: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$, $f = 1\,(\text{Hz})$.

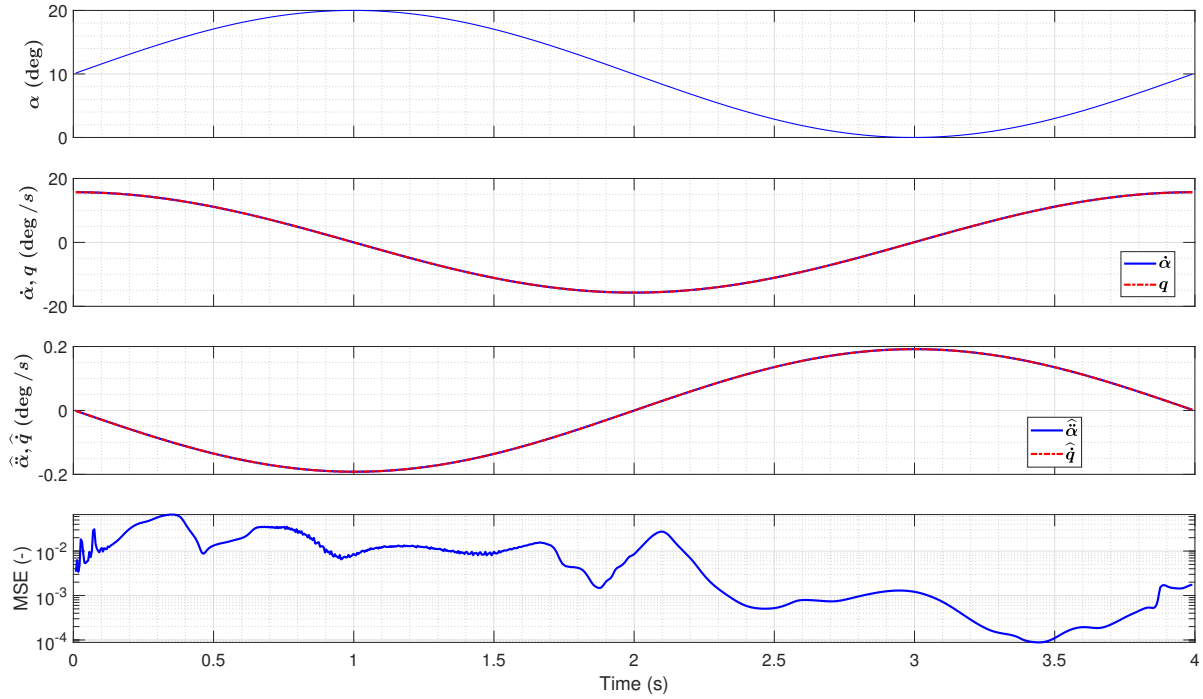


Figure A.8: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.

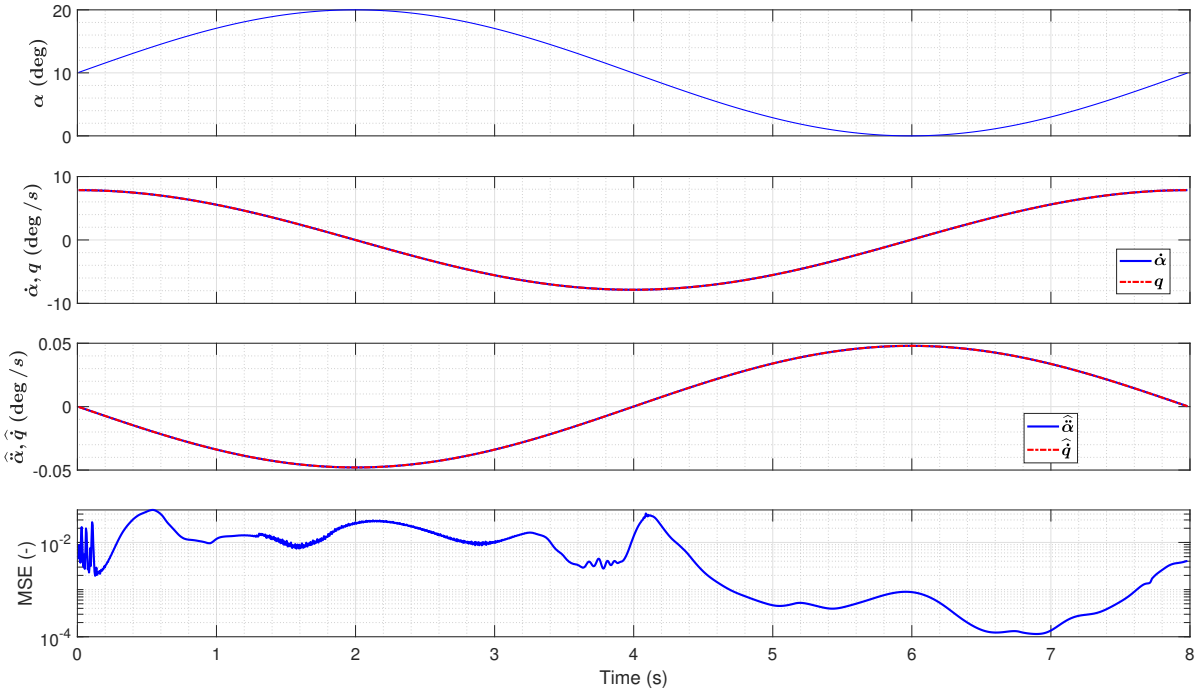


Figure A.9: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$, $f = 0.25\,(\text{Hz})$.

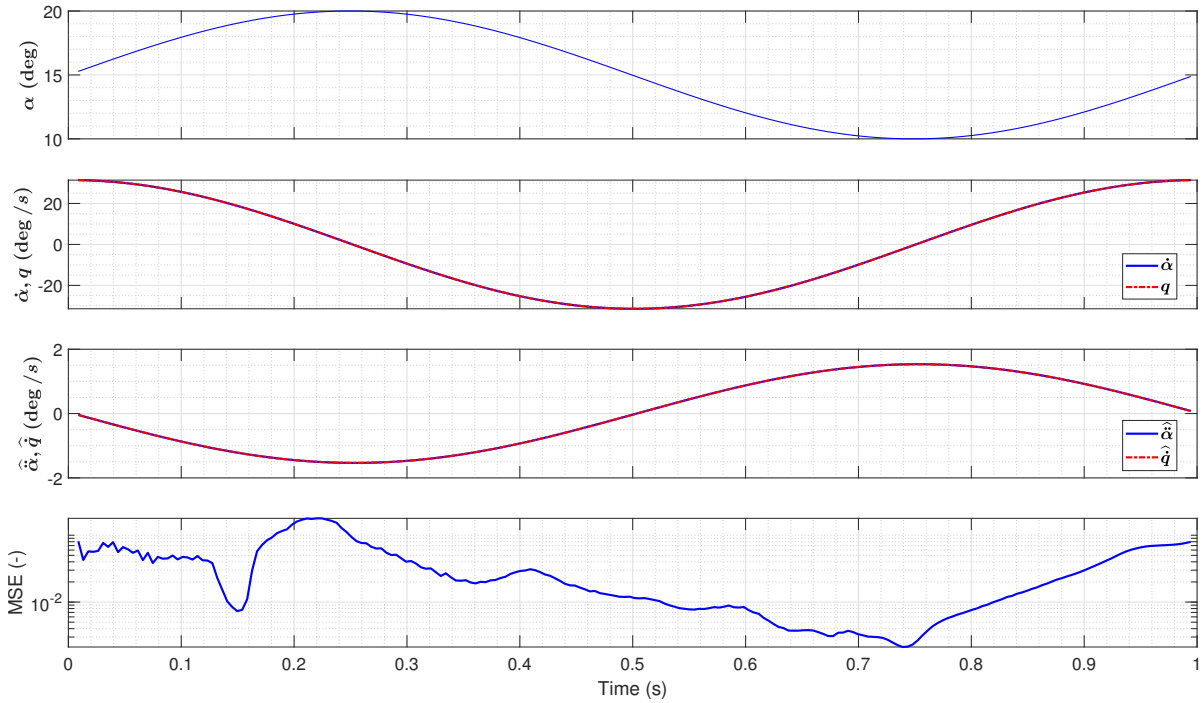


Figure A.10: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$.

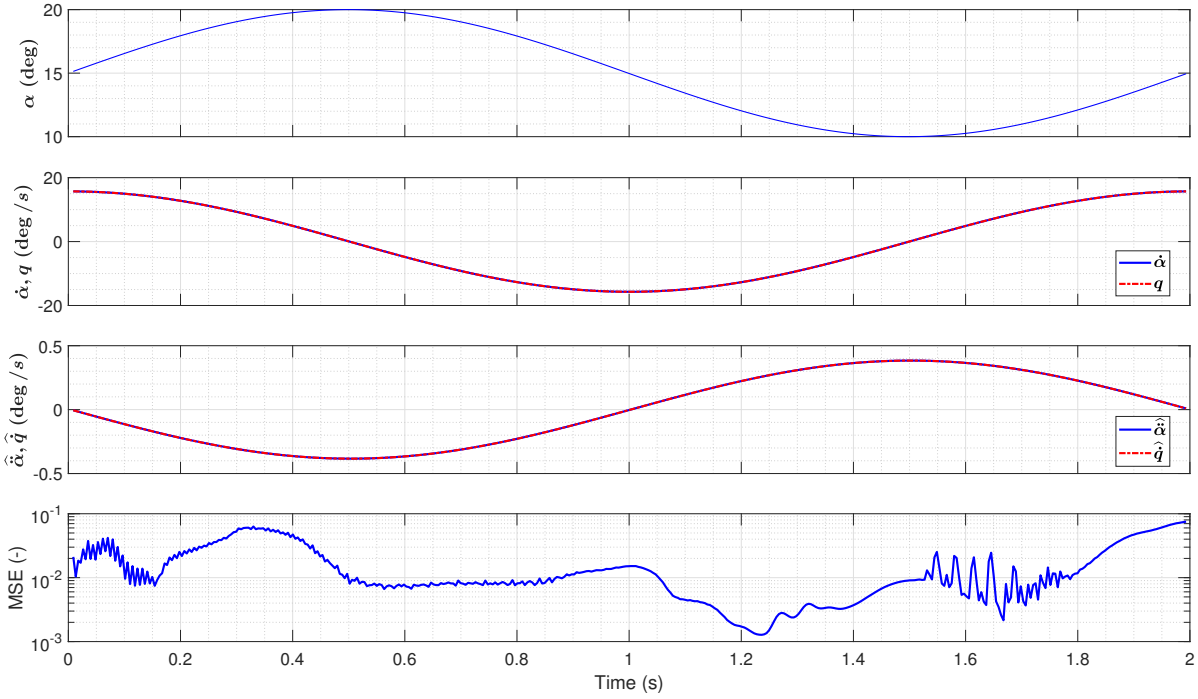


Figure A.11: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.

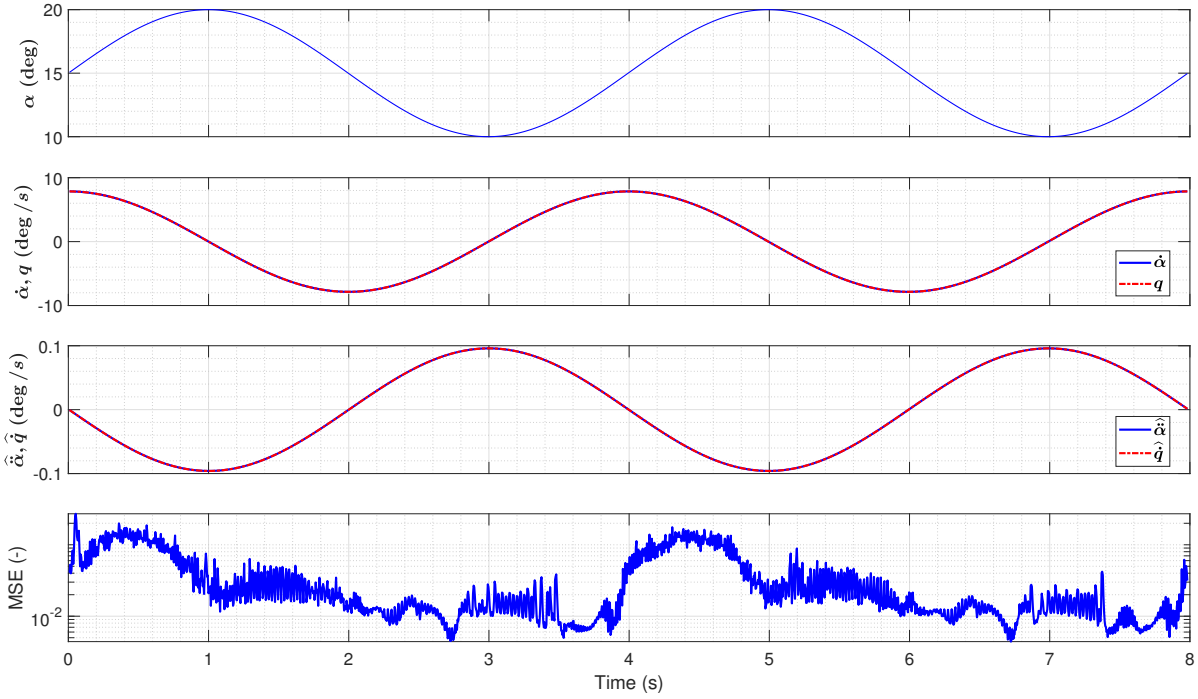


Figure A.12: Pitch oscillation inputs and errors in pressure distributions: $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$.

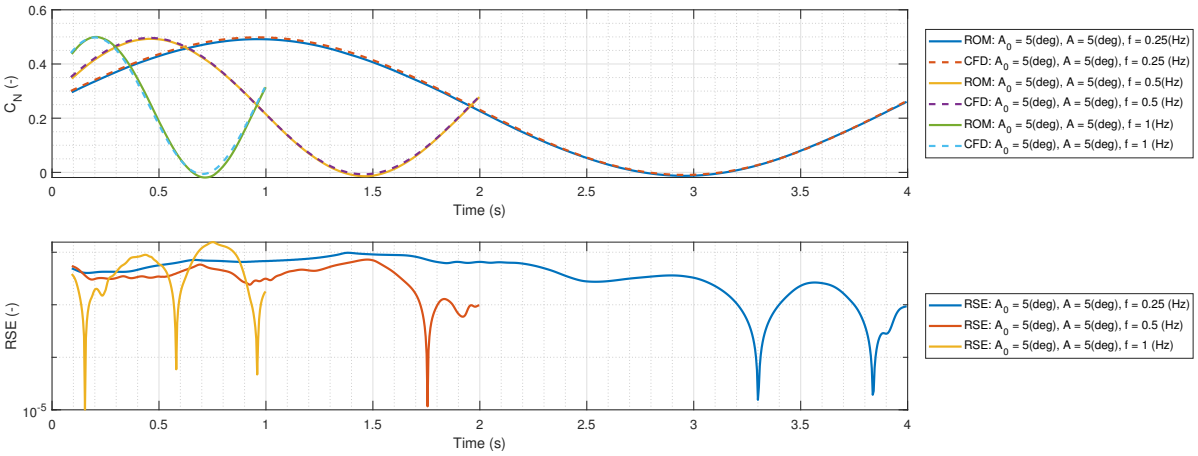


Figure A.13: Comparison of CFD and ROM results of C_N for plunge oscillation at $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$.

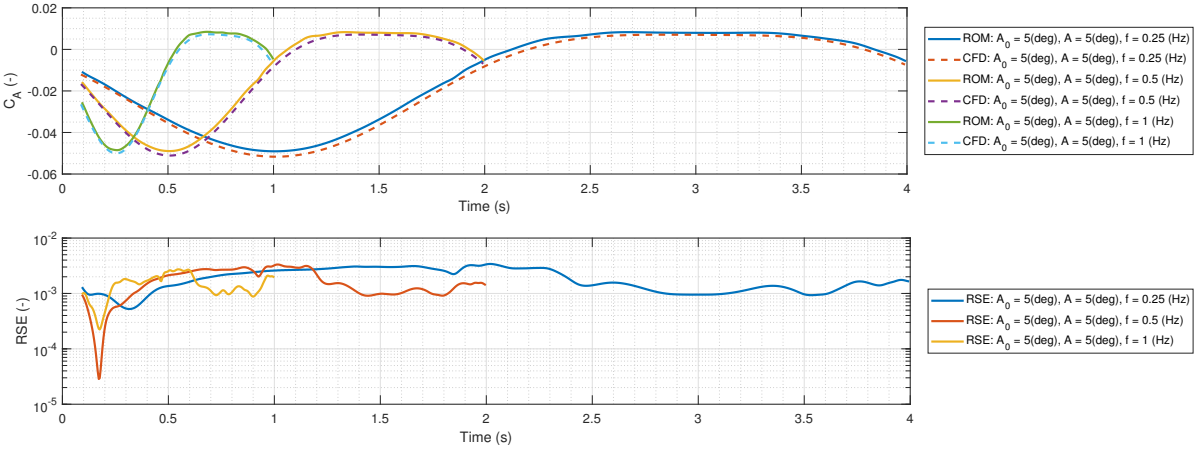


Figure A.14: Comparison of CFD and ROM results of C_A for plunge oscillation at $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$.

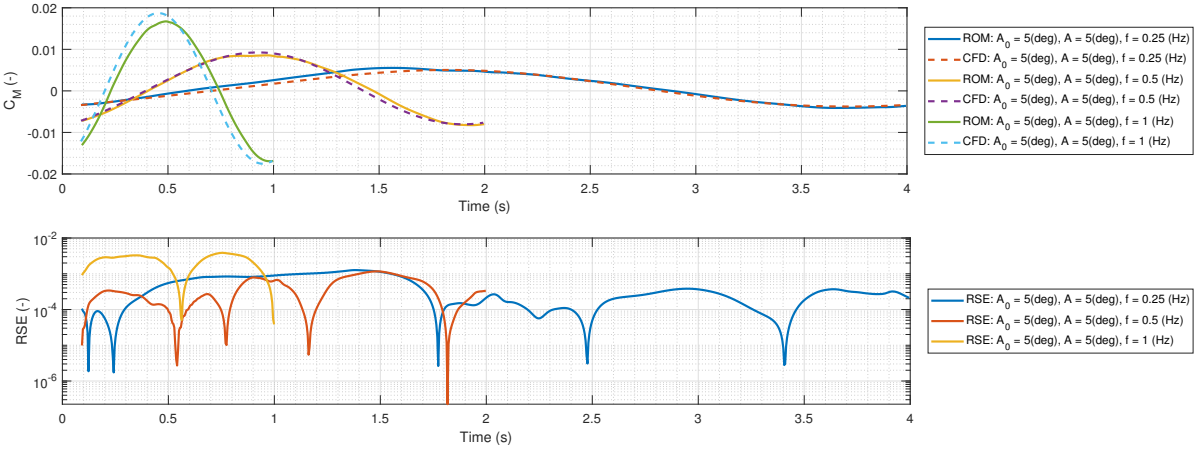


Figure A.15: Comparison of CFD and ROM results of C_T for plunge oscillation at $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$.

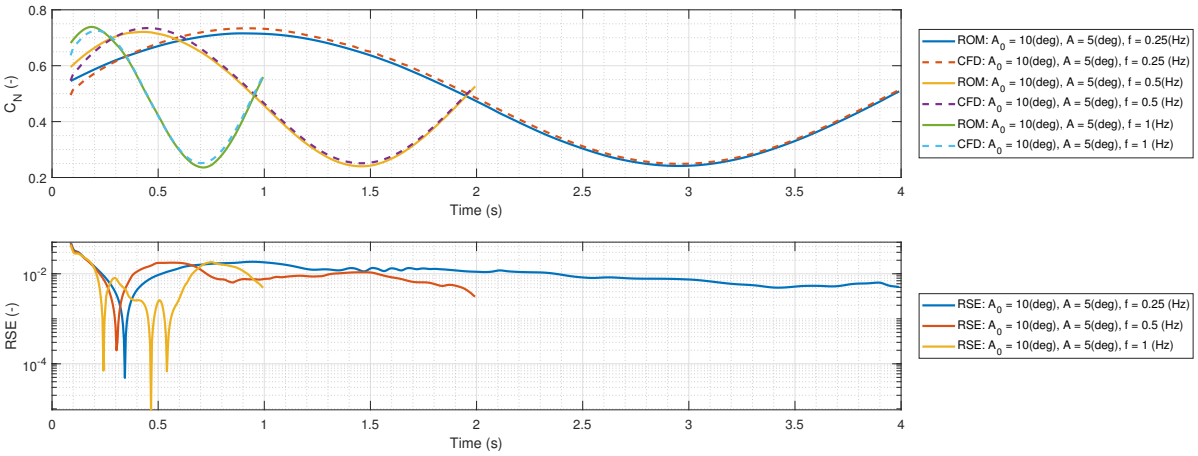


Figure A.16: Comparison of CFD and ROM results of C_N for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$.

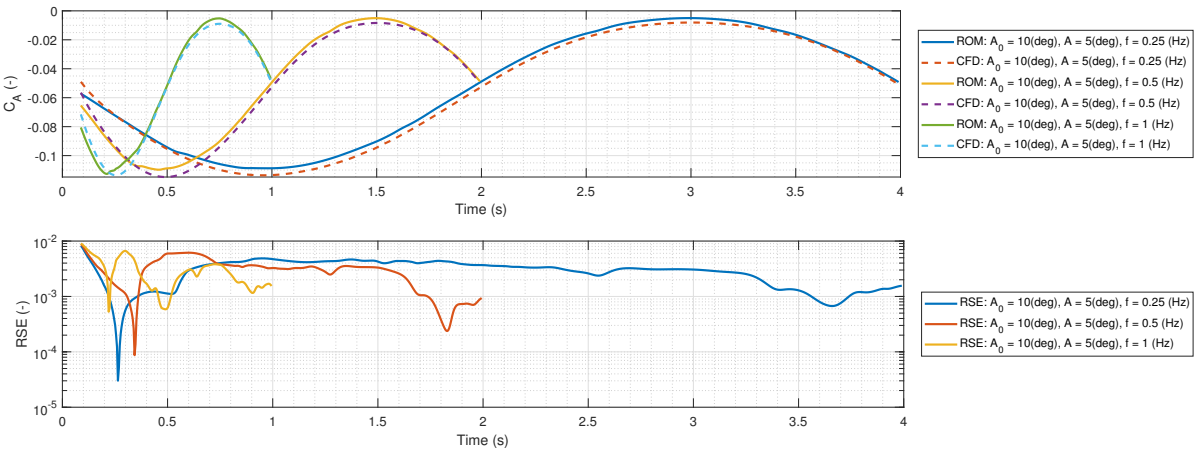


Figure A.17: Comparison of CFD and ROM results of C_A for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$.

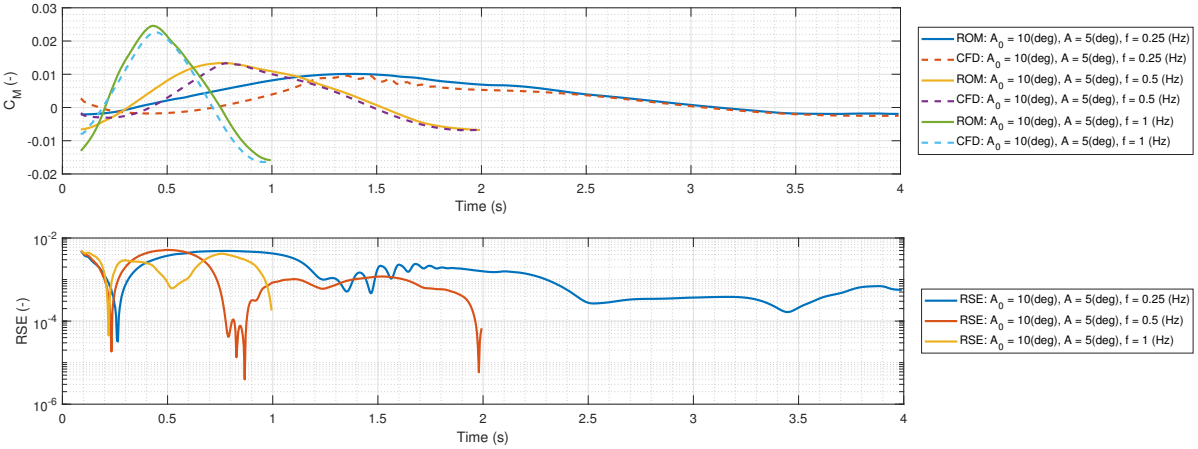


Figure A.18: Comparison of CFD and ROM results of C_M for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$.

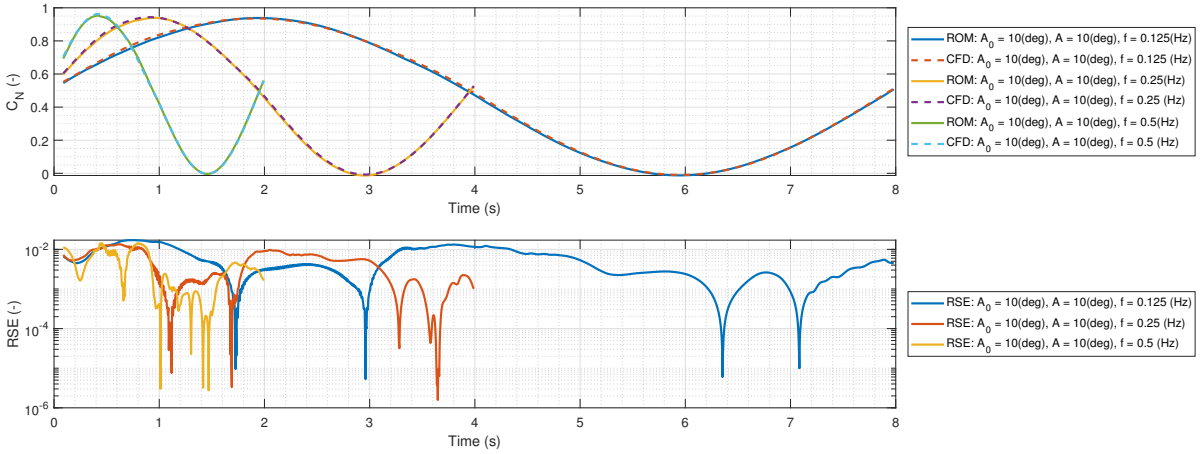


Figure A.19: Comparison of CFD and ROM results of C_N for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$.

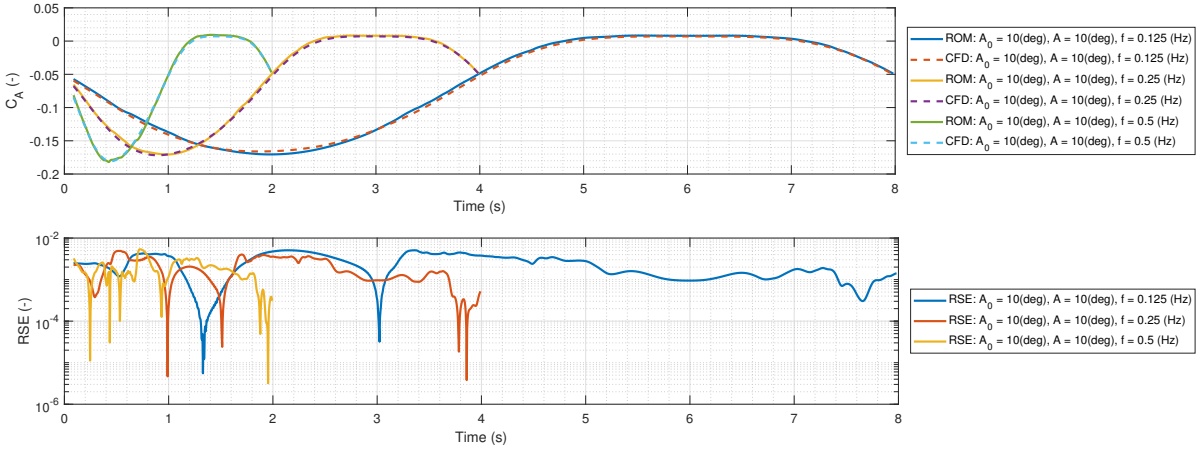


Figure A.20: Comparison of CFD and ROM results of C_A for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$.

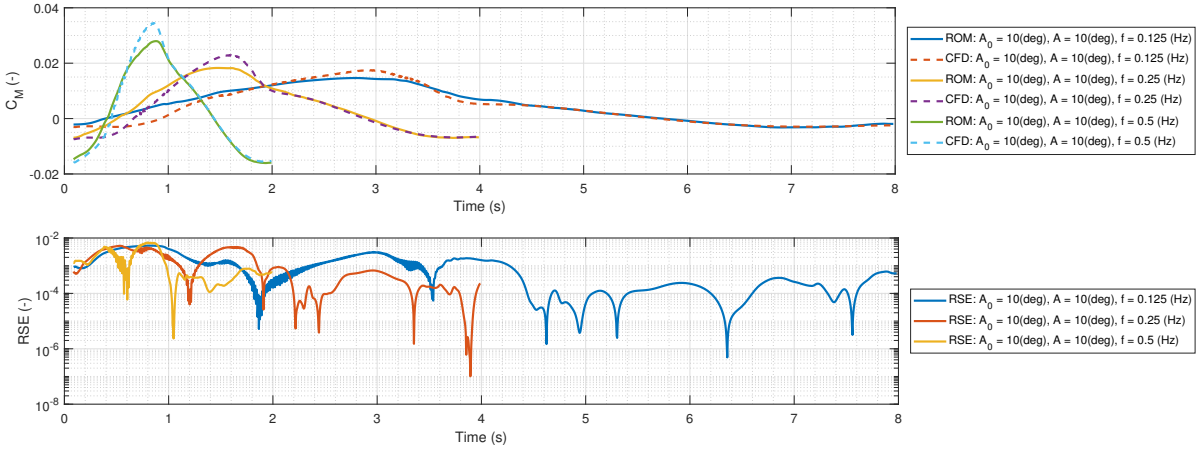


Figure A.21: Comparison of CFD and ROM results of C_T for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$.

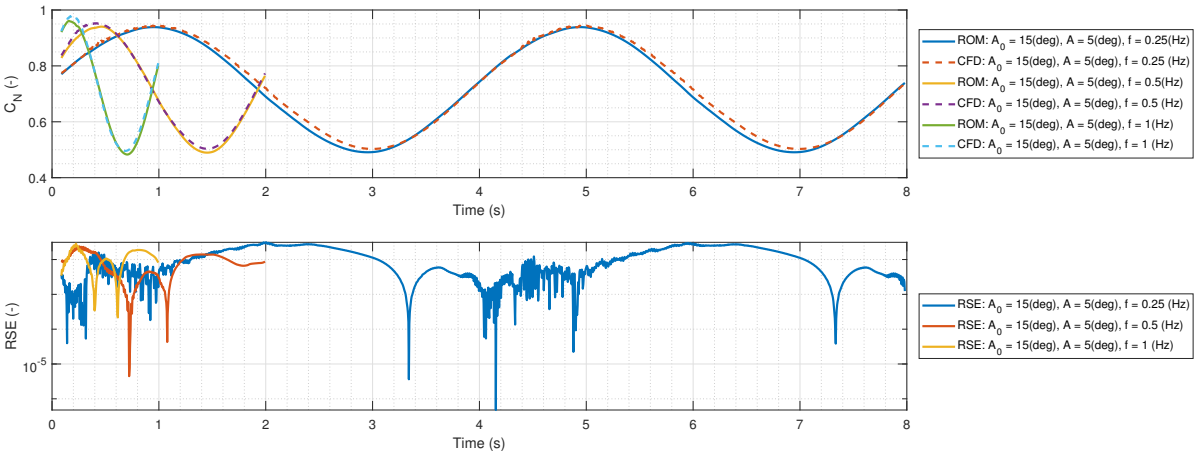


Figure A.22: Comparison of CFD and ROM results of C_N for plunge oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$.

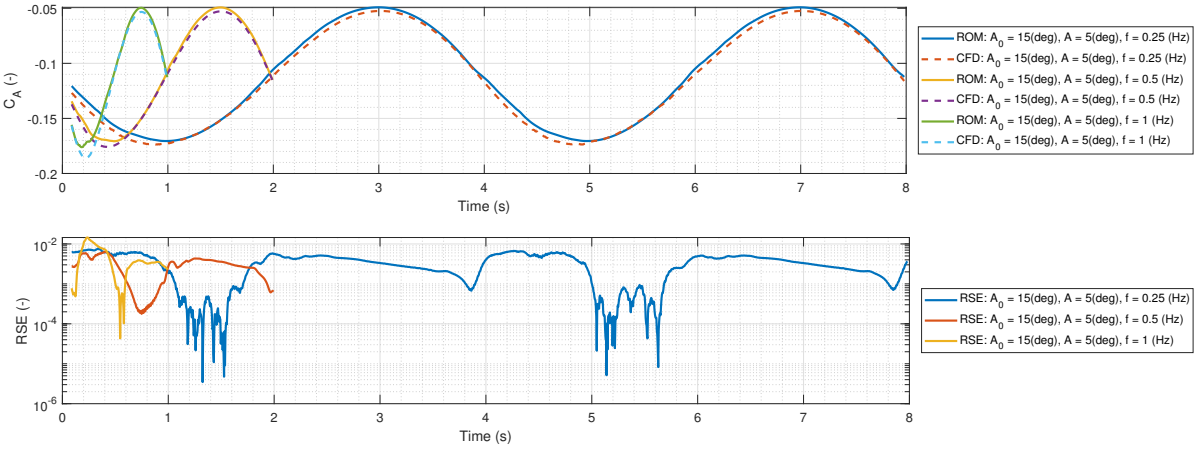


Figure A.23: Comparison of CFD and ROM results of C_A for plunge oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$.

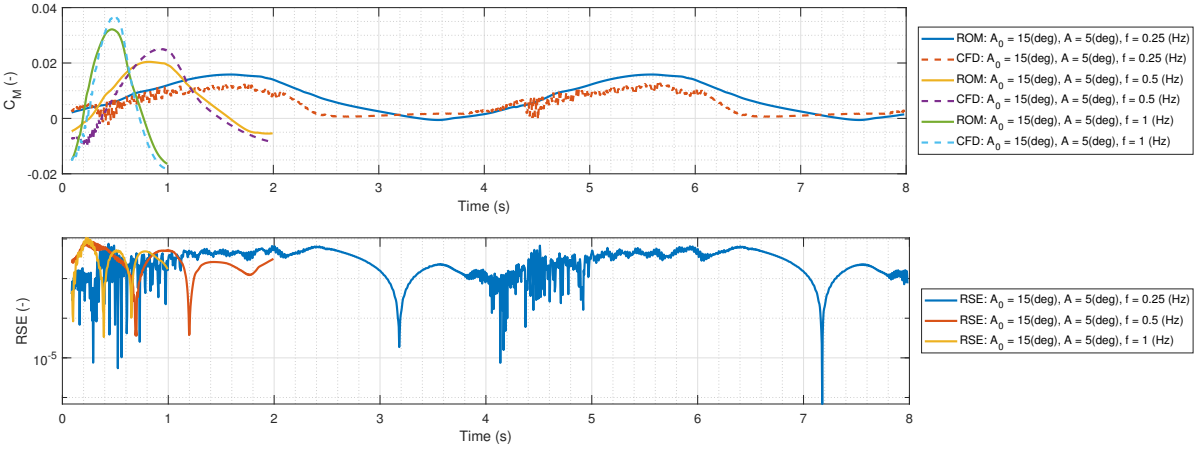


Figure A.24: Comparison of CFD and ROM results of C_M for plunge oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$.

A.2. PLUNGE OSCILLATIONS

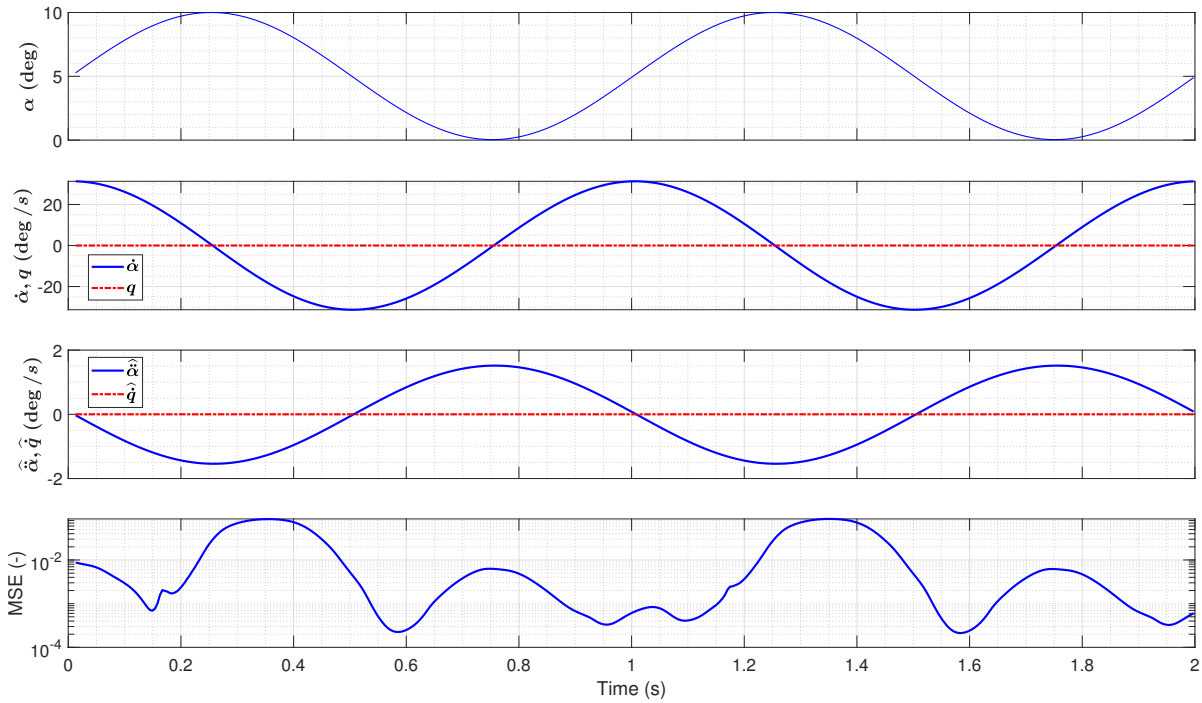


Figure A.25: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$.

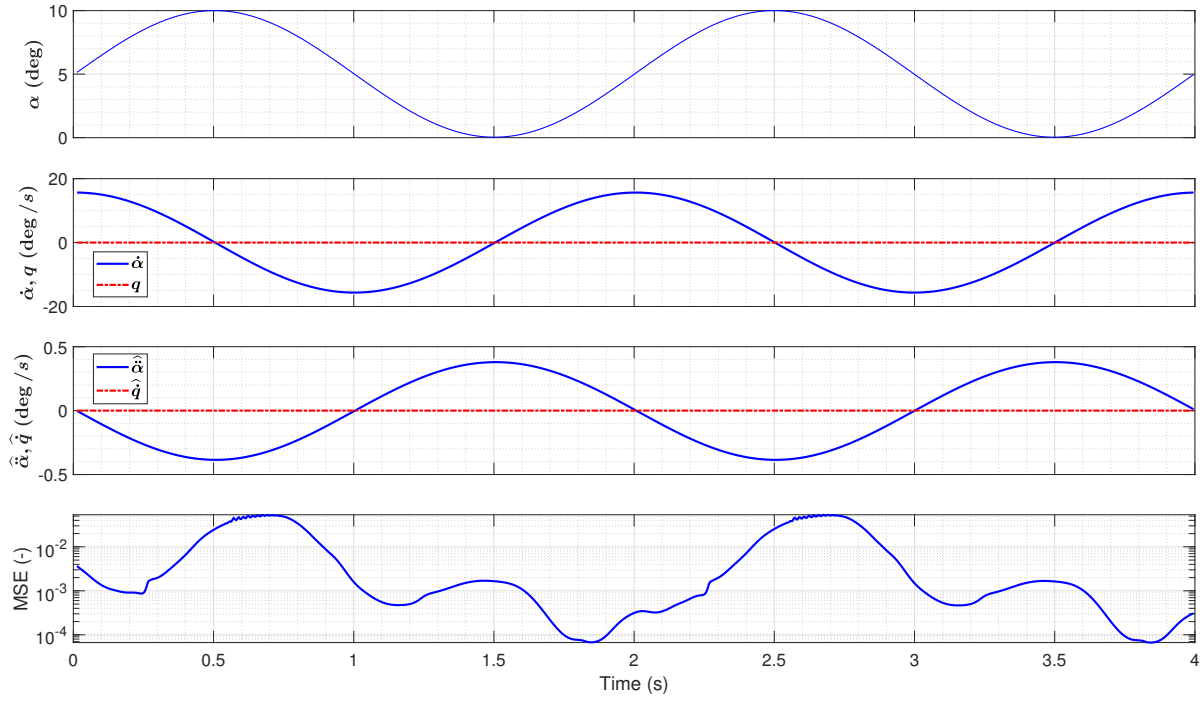


Figure A.26: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.

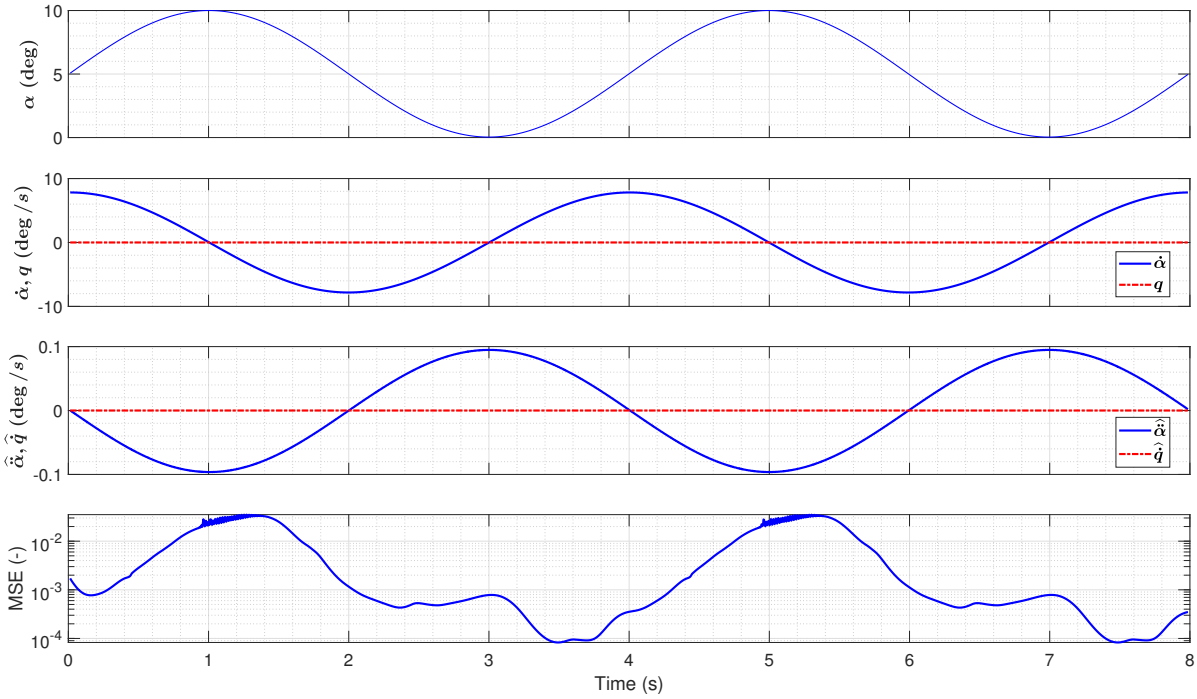


Figure A.27: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$.

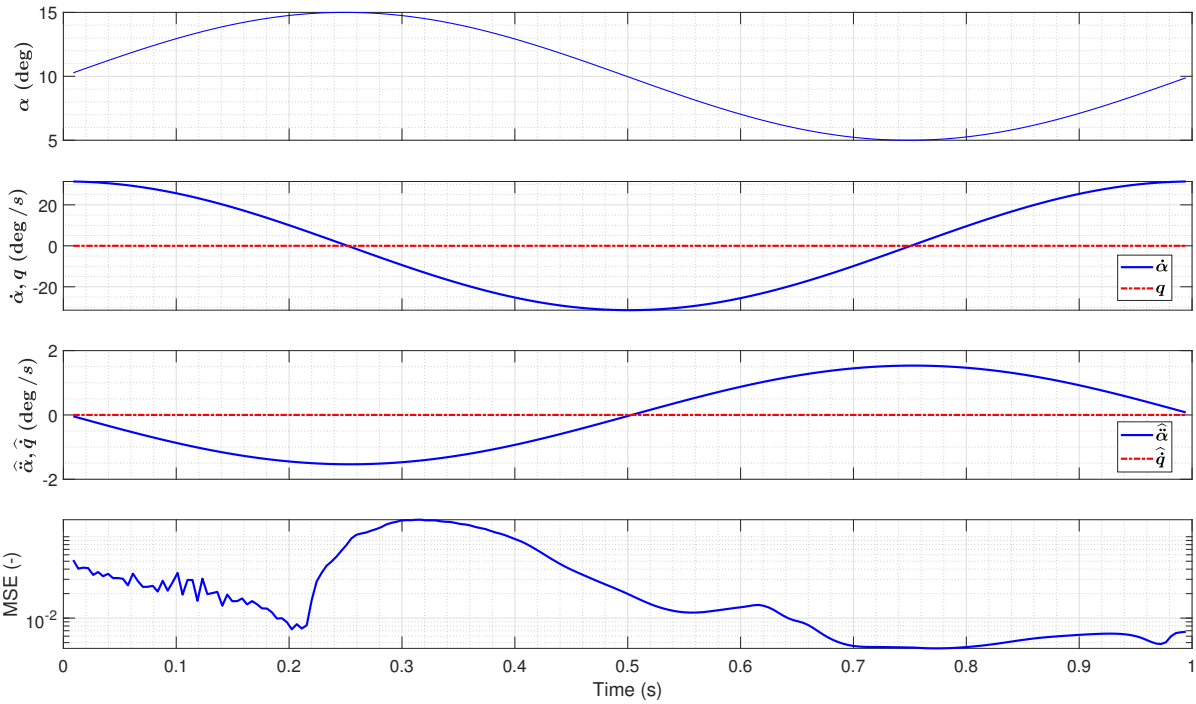


Figure A.28: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$.

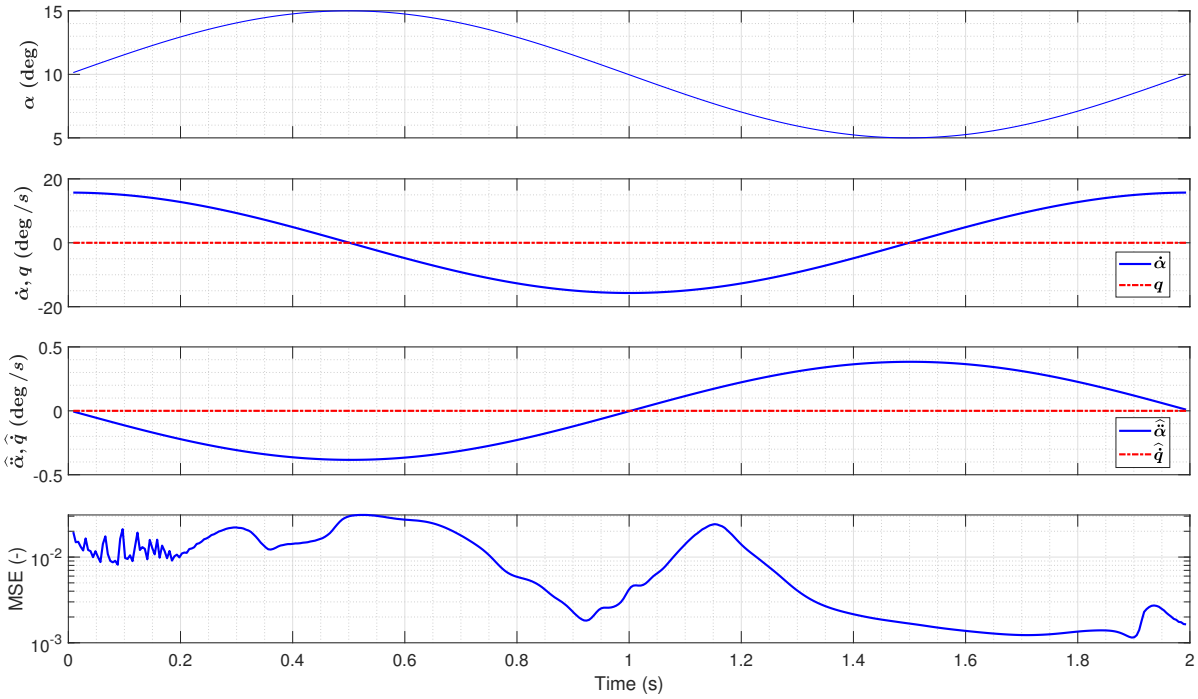


Figure A.29: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.

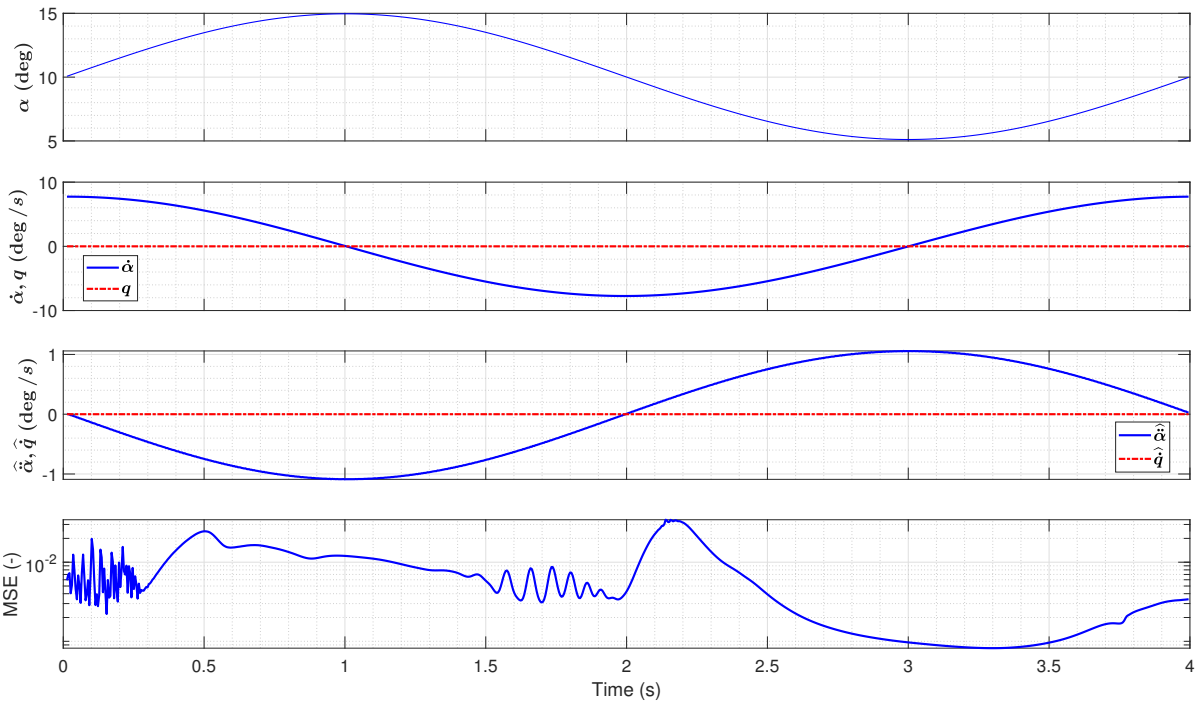


Figure A.30: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$.

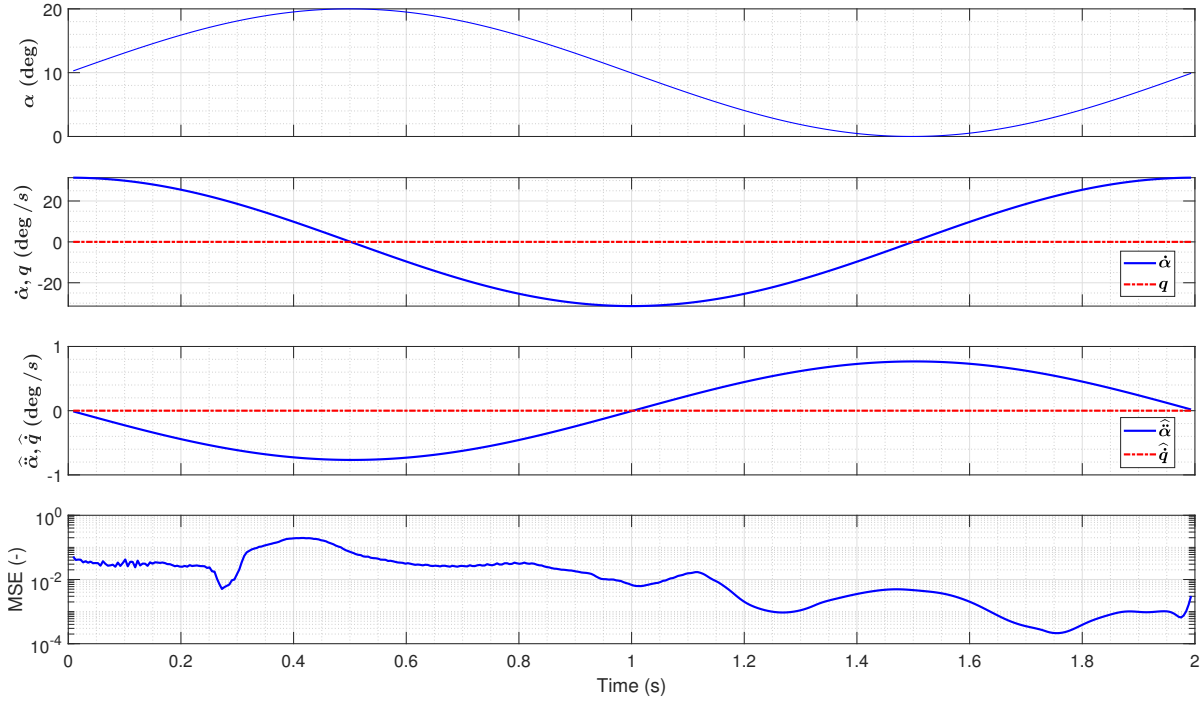


Figure A.31: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$, $f = 1\,(\text{Hz})$.

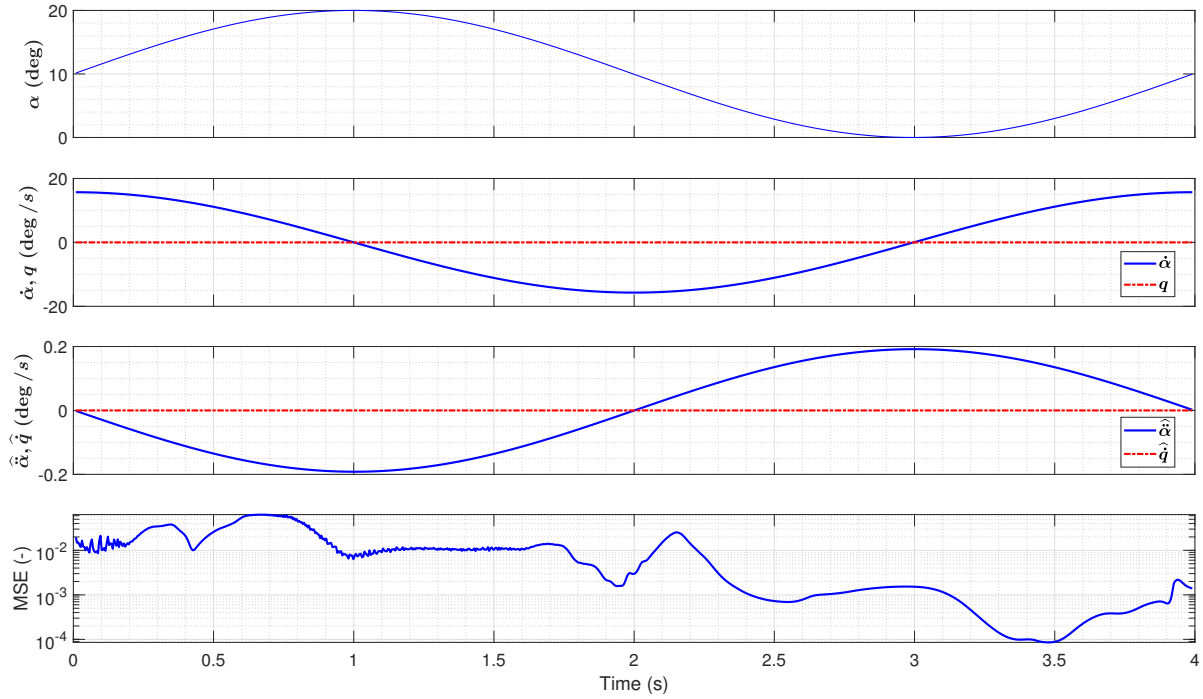


Figure A.32: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.

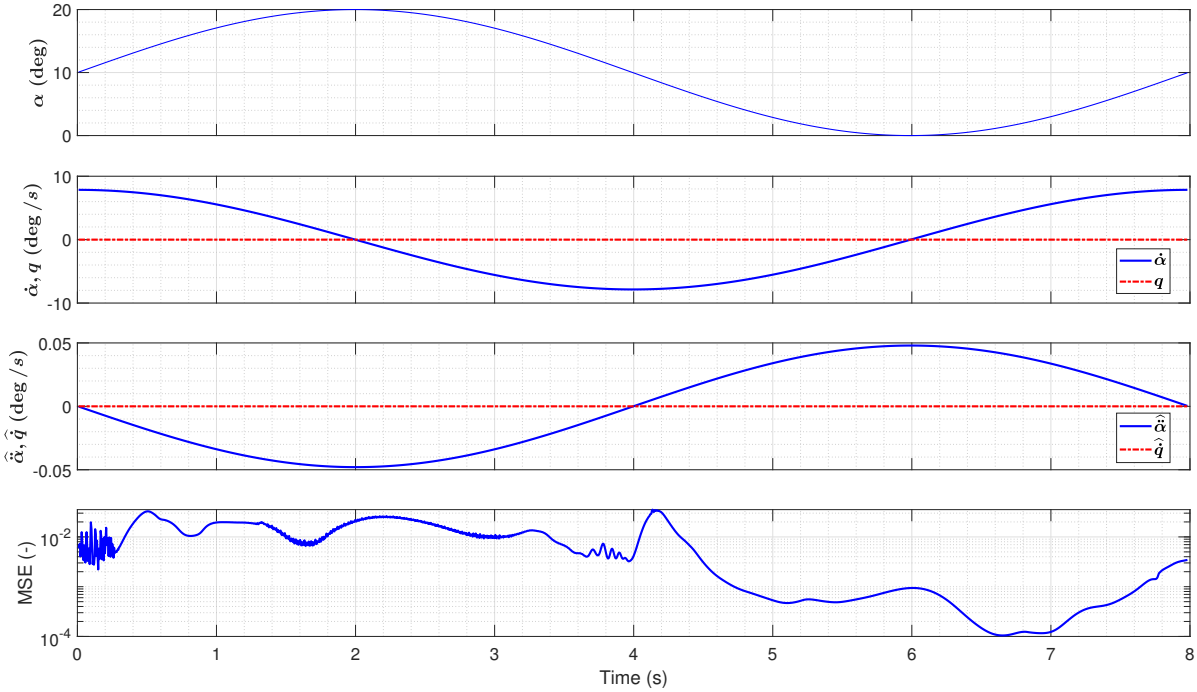


Figure A.33: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$, $f = 0.25\,(\text{Hz})$.

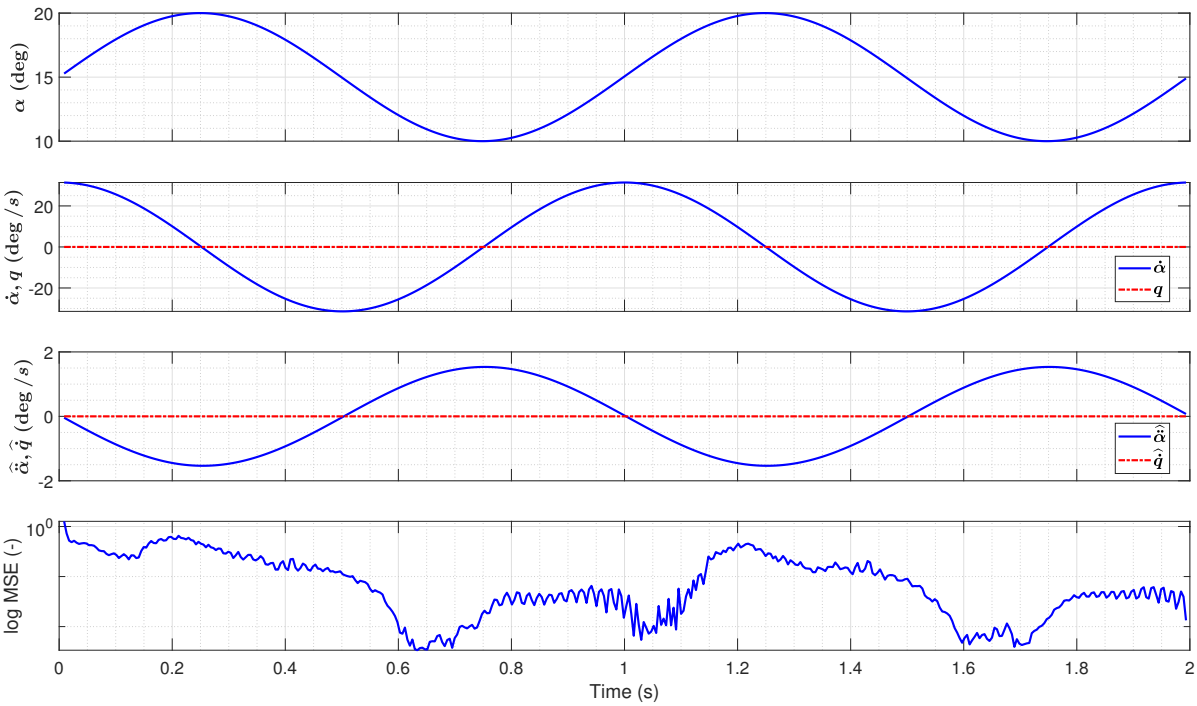


Figure A.34: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 1\,(\text{Hz})$.

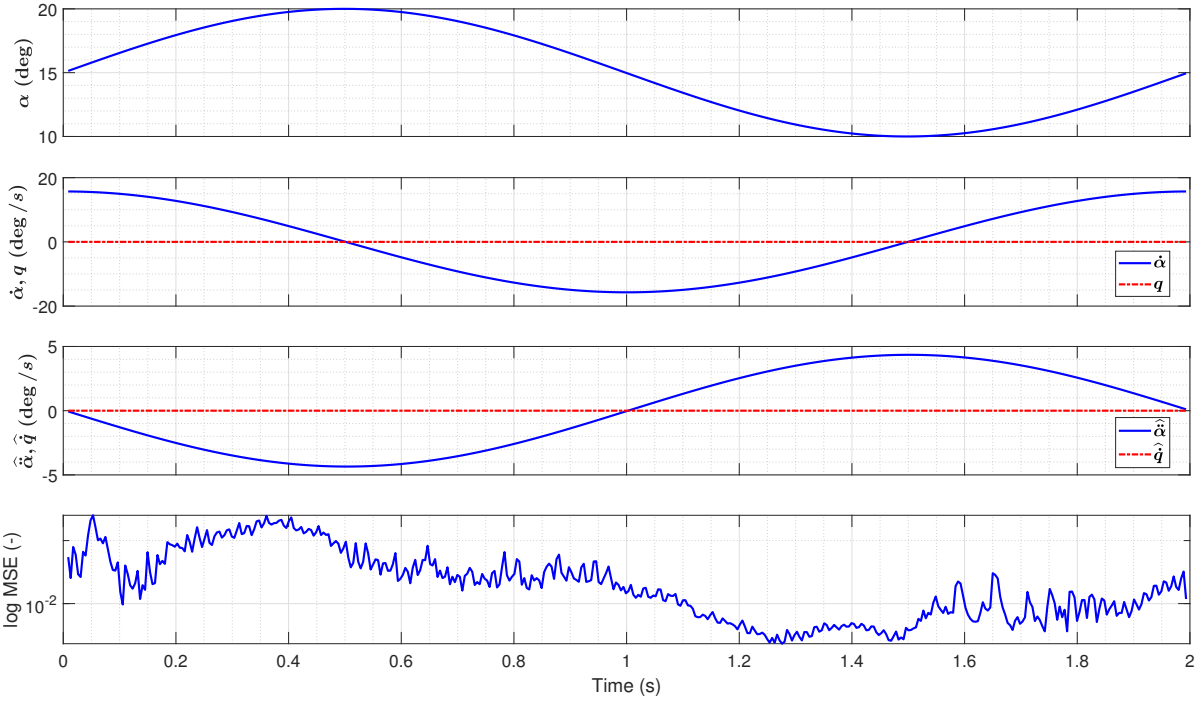


Figure A.35: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.5\,(\text{Hz})$.

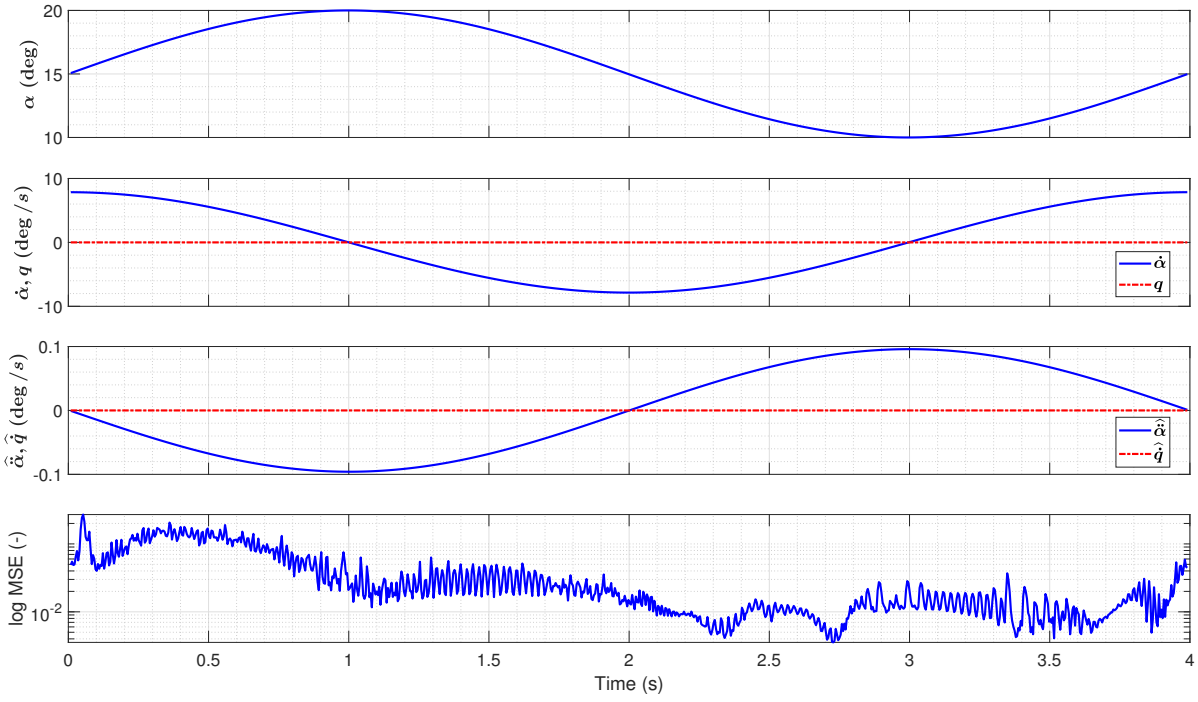


Figure A.36: Plunge oscillation inputs and errors in pressure distributions: $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$, $f = 0.25\,(\text{Hz})$.

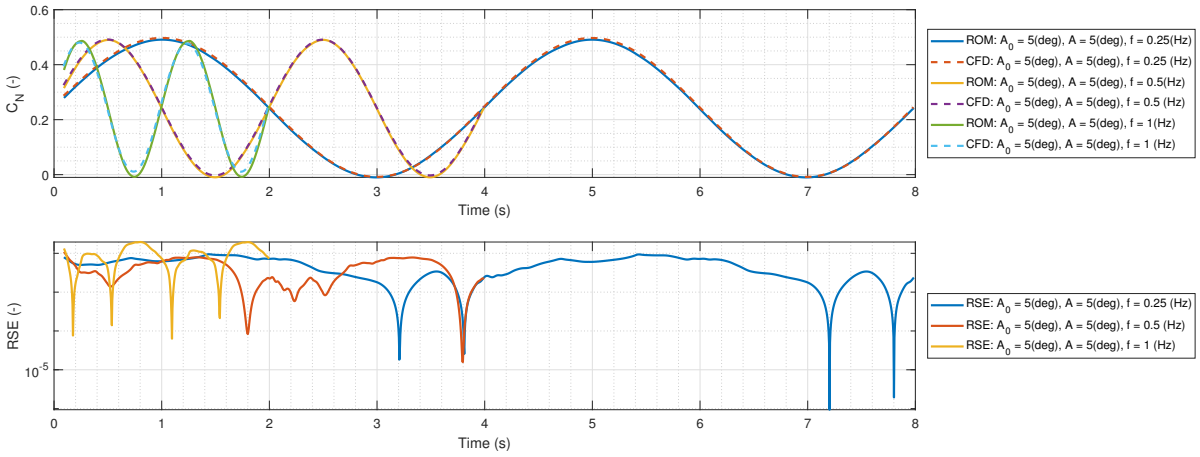


Figure A.37: Comparison of CFD and ROM results of C_N for plunge oscillation at $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$.

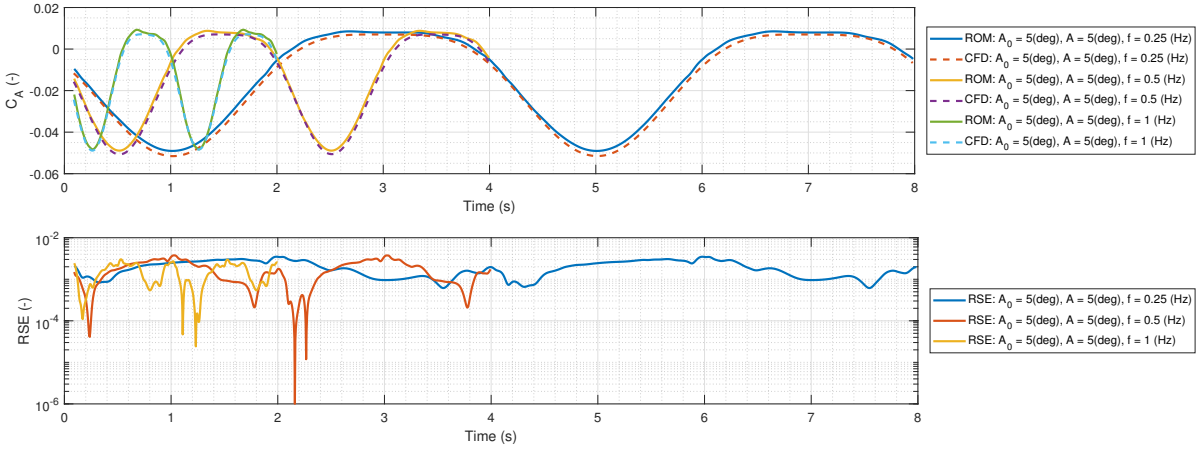


Figure A.38: Comparison of CFD and ROM results of C_A for plunge oscillation at $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$.

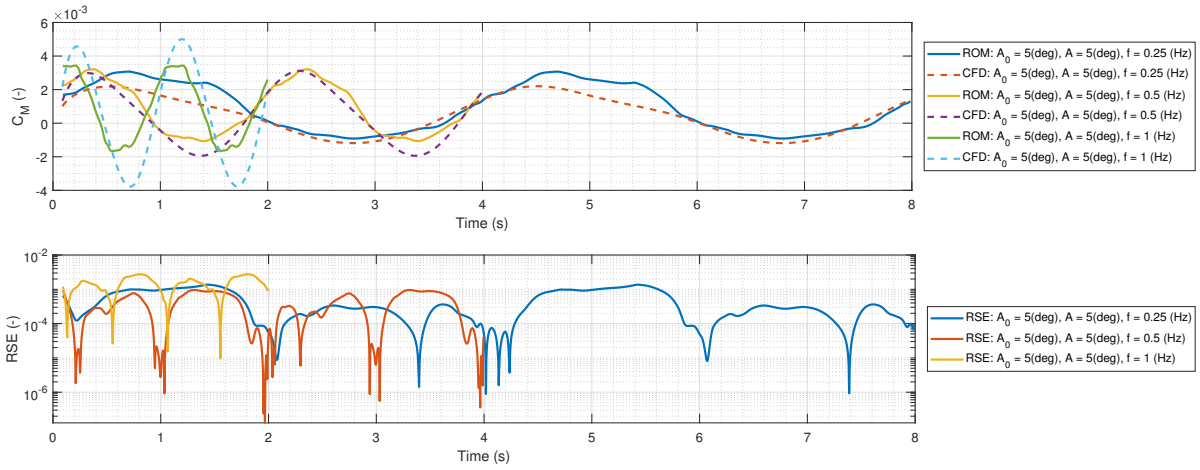


Figure A.39: Comparison of CFD and ROM results of C_M for plunge oscillation at $A_0 = 5\,(\text{deg})$, $A = 5\,(\text{deg})$.

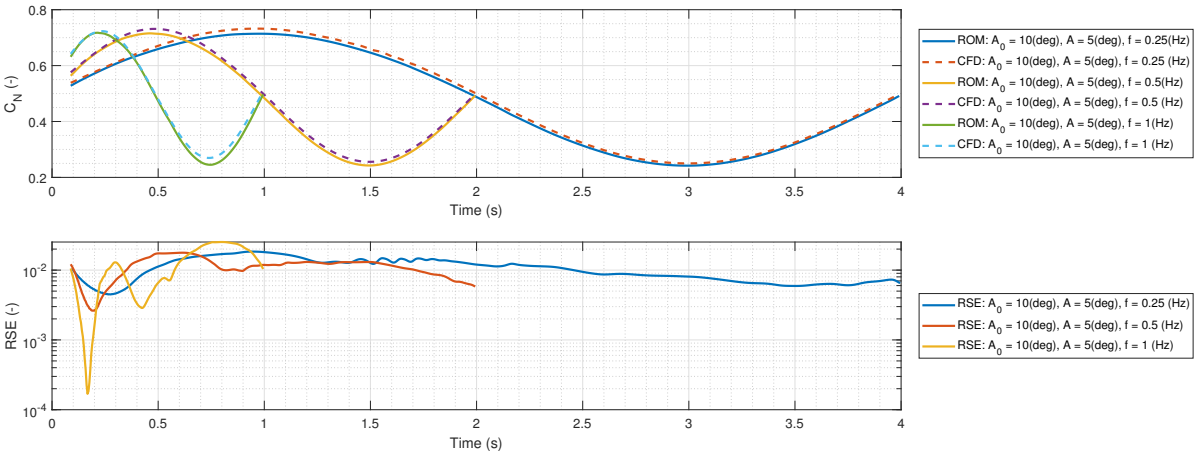


Figure A.40: Comparison of CFD and ROM results of C_N for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$.

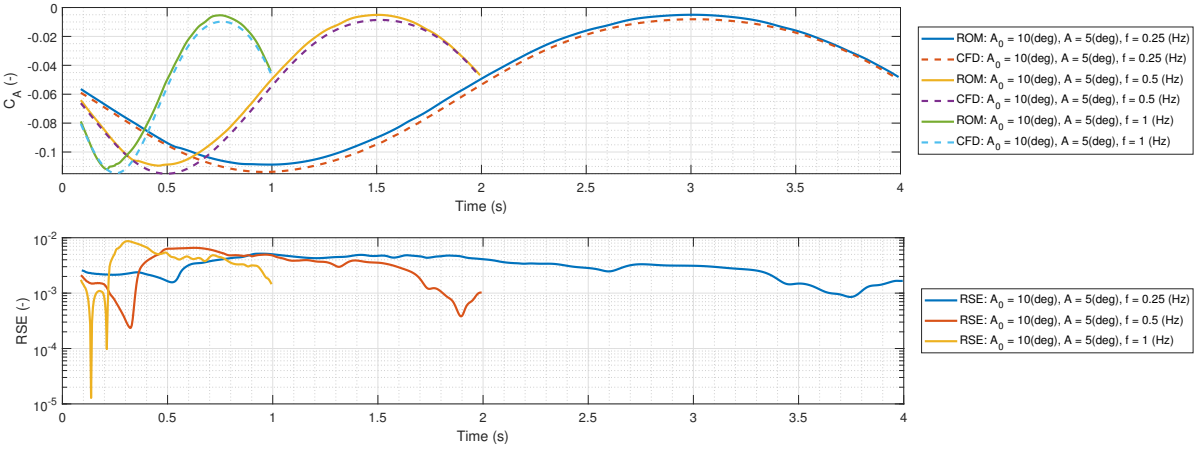


Figure A.41: Comparison of CFD and ROM results of C_A for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$.

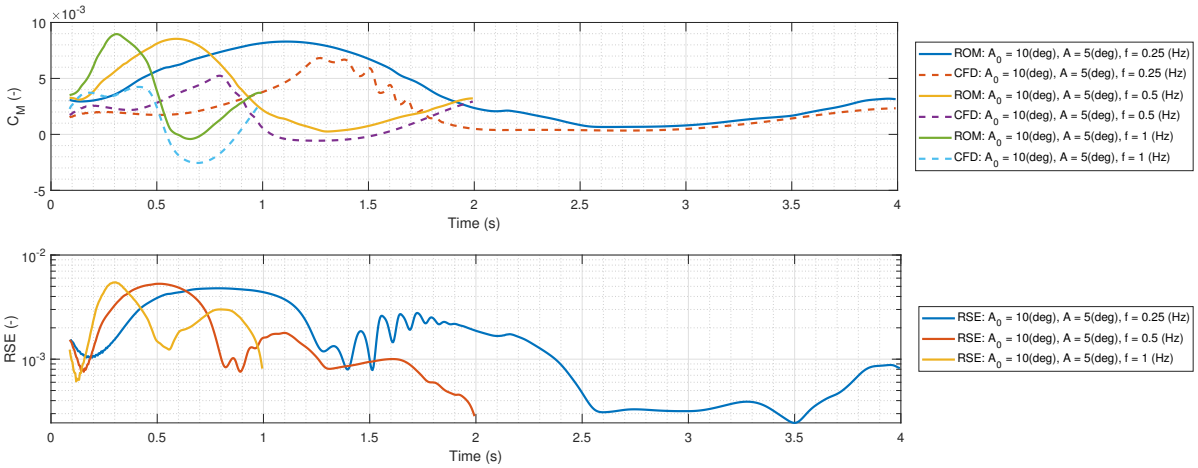


Figure A.42: Comparison of CFD and ROM results of C_M for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 5\,(\text{deg})$.

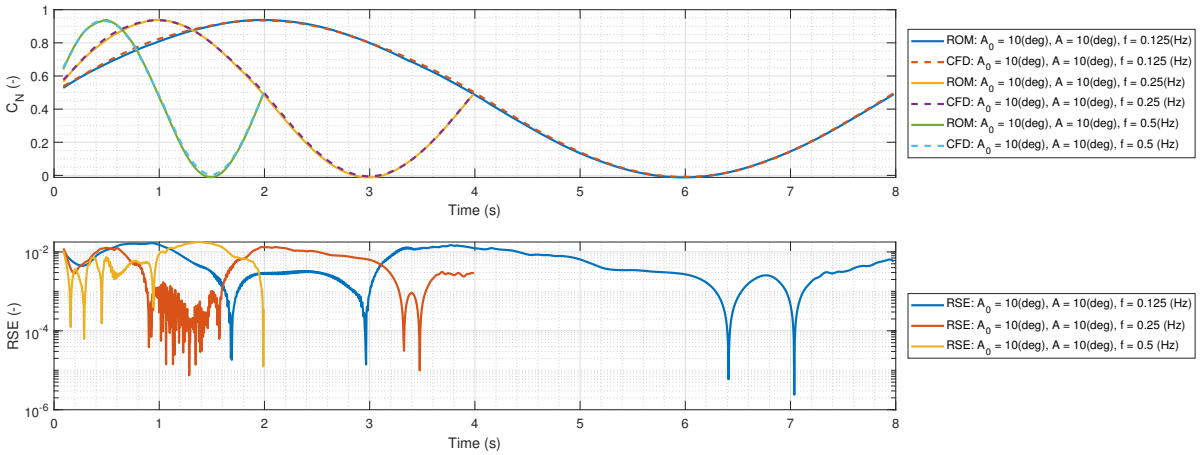


Figure A.43: Comparison of CFD and ROM results of C_N for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$.

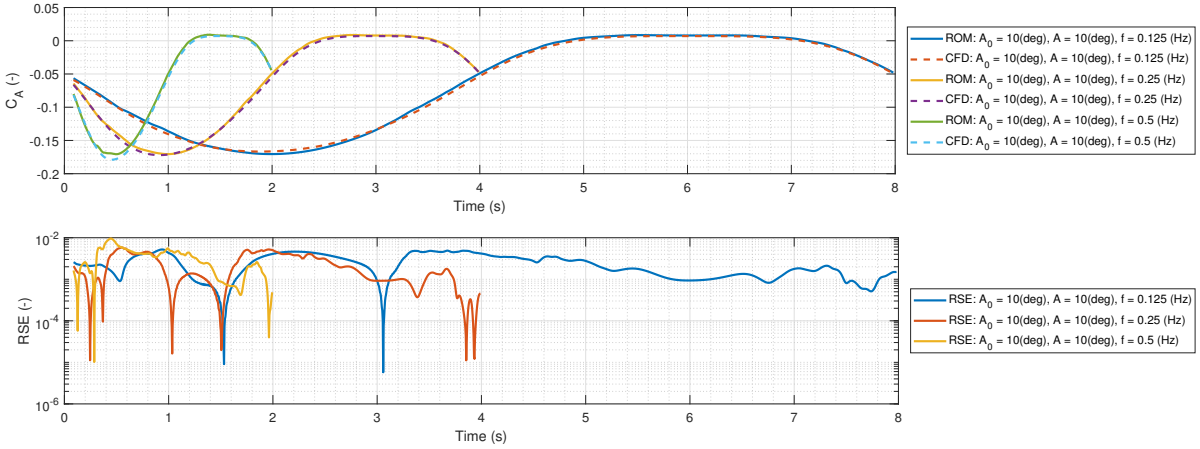


Figure A.44: Comparison of CFD and ROM results of C_A for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$.

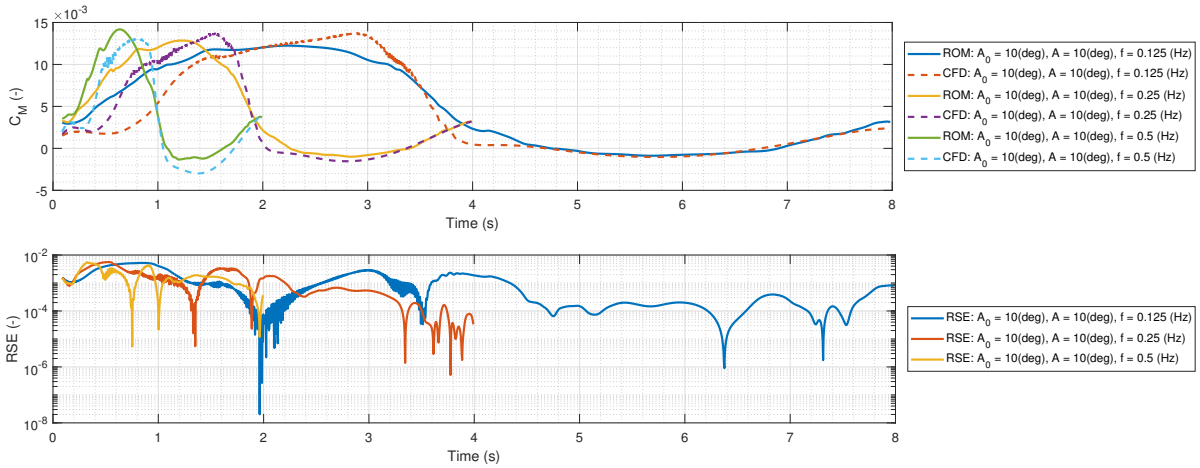


Figure A.45: Comparison of CFD and ROM results of C_M for plunge oscillation at $A_0 = 10\,(\text{deg})$, $A = 10\,(\text{deg})$.

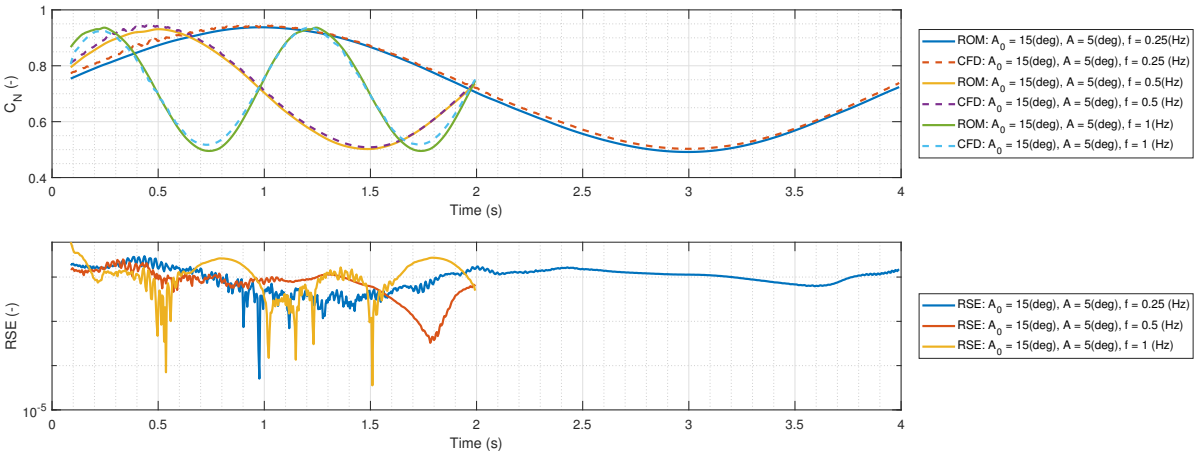


Figure A.46: Comparison of CFD and ROM results of C_N for plunge oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$.

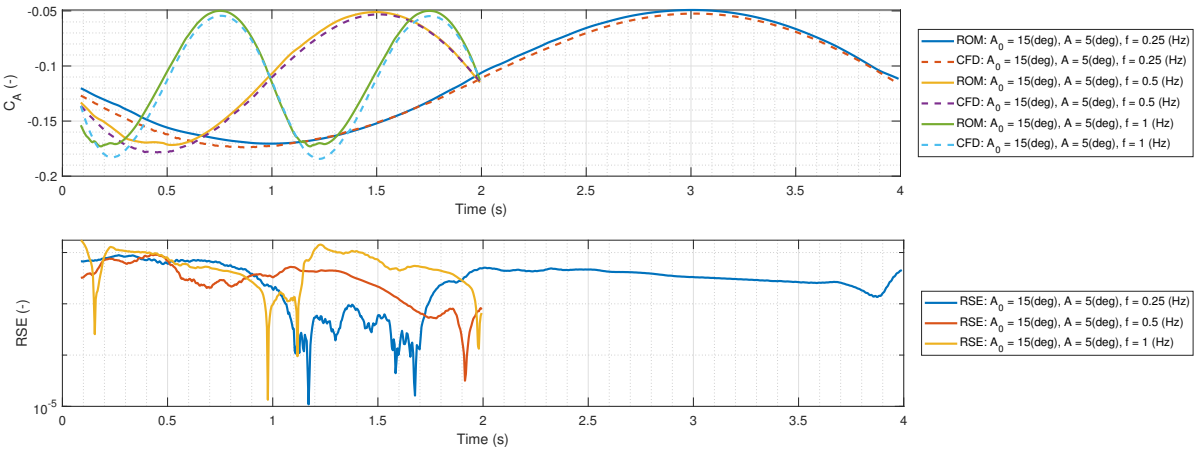


Figure A.47: Comparison of CFD and ROM results of C_A for plunge oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$.

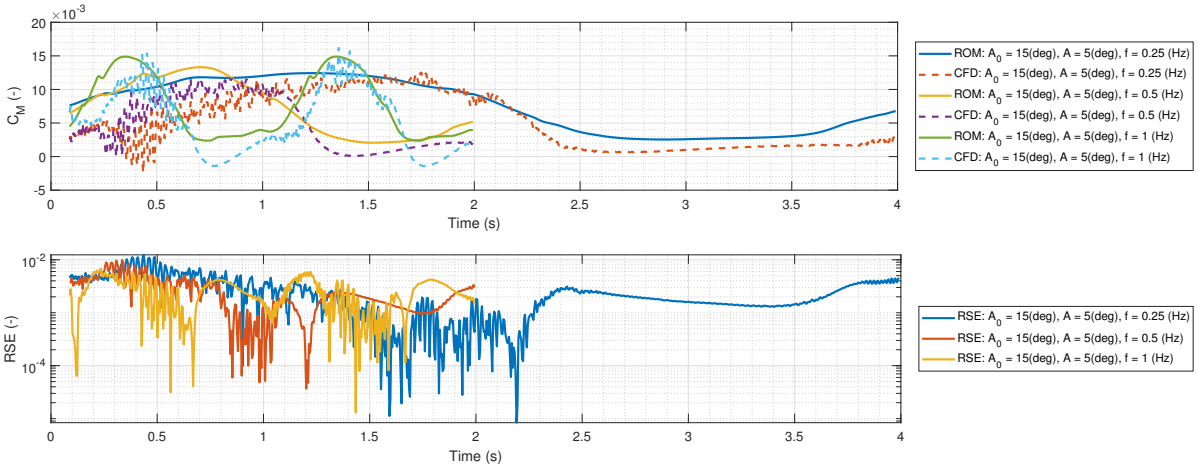


Figure A.48: Comparison of CFD and ROM results of C_M for plunge oscillation at $A_0 = 15\,(\text{deg})$, $A = 5\,(\text{deg})$.

BIBLIOGRAPHY

[1] *Dassault neuron UCAV,* (n.d.), [Online; accessed 15-Jul-2018], available at https://www.turbosquid.com/3d-models/unmanned-neuron-rigged-3d-model/879529.

[2] *Neural network cover image,* (n.d.), [Online; accessed 15-Jul-2018], available at https://becominghuman.ai/artificial-neural-networks-and-deep-learning-a3c9136f2137.

[3] *Abstract network stock photo,* (2016), [Online; accessed 15-Jul-2018], available at https://www.istockphoto.com/nl/foto/abstract-network-gm508917664-85503939.

[4] P. Reisenthel, *Development of a nonlinear indicial model for maneuvering fighter aircraft,* in *34th Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings (American Institute of Aeronautics and Astronautics, 1996) doi: 10.2514/6.1996-896.

[5] M. Ghoreyshi, A. Jirásek, and R. M. Cummings, *Reduced order unsteady aerodynamic modeling for stability and control analysis using computational fluid dynamics,* Progress in Aerospace Sciences **71**, 167 (2014), doi: 10.1016/j.paerosci.2014.09.001.

[6] Q. Wang, W. Qian, and K. He, *Unsteady aerodynamic modeling at high angles of attack using support vector machines,* Chinese Journal of Aeronautics **28**, 659 (2015).

[7] M. Winter and C. Breitsamter, *Efficient unsteady aerodynamic loads prediction based on nonlinear system identification and proper orthogonal decomposition,* Journal of Fluids and Structures **67**, 1 (2016).

[8] D. J. Lucia, P. S. Beran, and W. A. Silva, *Reduced-order modeling: new approaches for computational physics,* Progress in Aerospace Sciences **40**, 51 (2004), doi: 10.1016/j.paerosci.2003.12.001.

[9] W. Silva, *Identification of nonlinear aeroelastic systems based on the volterra theory: Progress and opportunities,* Nonlinear Dynamics **39**, 25 (2005), doi: 10.1007/s11071-005-1907-z.

[10] S. J. Schreck, W. E. Faller, and M. W. Luttges, *Neural network prediction of three-dimensional unsteady separated flowfields, Journal of Aircraft,* Journal of Aircraft **32**, 178 (1995).

[11] D. I. Ignatyev and A. N. Khrabrov, *Neural network modeling of unsteady aerodynamic characteristics at high angles of attack,* Aerospace Science and Technology **41**, 106 (2015).

[12] D. I. Ignatyev and A. N. Khrabrov, *Application of neural networks in the simulation of dynamic effects of canard aircraft aerodynamics,* TsAGI Science Journal **42**, 817 (2011).

[13] L. Planckaert, *Model of unsteady aerodynamic coefficients of a delta wing aircraft at high angles of attack,* Tech. Rep. (OFFICE NATIONAL D'ETUDES ET DE RECHERCHES AERONOSPATIALES CEDEX FRANCE SYSTEMS CONTROL/FLIGHT DYNAMICS, 2003).

[14] A. Spentzos, G. Barakos, K. Badcock, and B. Richards, *Modelling of 3d dynamic stall using cfd and neural networks,* (American Institute of Aeronautics and Astronautics, 2018).

[15] Y. Lyu, W. Zhang, J. Shi, X. Qu, and Y. Cao, *Unsteady aerodynamic modeling of biaxial coupled oscillation based on improved elm,* Aerospace Science and Technology **60**, 58 (2017), doi: 10.1016/j.ast.2016.10.029.

[16] X. Guo, W. Li, and F. Iorio, *Convolutional neural networks for steady flow approximation,* in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16 (ACM, New York, NY, USA, 2016) pp. 481–490.

[17] O. Hennigh, *Automated Design using Neural Networks and Gradient Descent,* ArXiv e-prints (2017), arXiv:1710.10352 [stat.ML] .

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) http://www.deeplearningbook.org.

[19] M. A. Nielsen, *Neural Networks and Deep Learning* (Determination Press, 2015).

[20] M. Rouse and J. Burke, *Definition: neural network,* TechTarget (2016), [Online; accessed 11-Nov-2017], available at http://searchnetworking.techtarget.com/definition/neural-network.

[21] A. Karpathy, *Convolutional neural networks for visual recognition,* lecture notes, Stanford University (2017), [Online; accessed 10-Jul-2018], available at http://cs231n.github.io/.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors,* Nature **323**, 533 (1986).

[23] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning,* Nature **521**, 436 (2015).

[24] Wikipedia, *Logistic function — Wikipedia, the free encyclopedia,* (2018), [Online; accessed 17-July-2018], available at https://en.wikipedia.org/wiki/Logistic_function#Derivative.

[25] *How neural networks are trained,* ml4a (2018), [Online; accessed 20-06-2018], available at https://ml4a.github.io/ml4a/how_neural_networks_are_trained/.

[26] M. D. Zeiler, *ADADELTA: an adaptive learning rate method,* CoRR **abs/1212.5701** (2012), arXiv:1212.5701 .

[27] I. Dabbura, *Gradient descent algorithm and its variants,* (2017), [Online; accessed 20-June-2018], available at https://towardsdatascience.com/.

[28] *An overview of gradient descent optimization algorithms,* [online lecture], Coursera (2018), [Online; accessed 20-06-2018], available at https://www.coursera.org/lecture/deep-neural-network/understanding-mini-batch-gradient-descent-1BXu8.

[29] C. Fahey, *Neural network with learning by backward error propagation,* (n.d.), [Online; accessed 20-June-2018].

[30] S. Ruder, *Understanding mini-batch gradient descent,* (2016), [Online; accessed 20-06-2018], available at http://ruder.io/optimizing-gradient-descent/.

[31] Wikipedia, *Stochastic gradient descent — Wikipedia, the free encyclopedia,* (2018), [Online; accessed 01-Jul-2018], available at https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[32] Wikipedia, *Convolution — Wikipedia, the free encyclopedia,* (2017), [Online; accessed 18-Nov-2017], available at https://en.wikipedia.org/wiki/Convolution.

[33] E. W. Weisstein, *Convolution — From MathWorld–A Wolfram Web Resource,* (2017), [Online; accessed 18-Nov-2017] available at http://mathworld.wolfram.com/Convolution.html.

[34] A. Deshpande, *A beginner's guide to understanding convolutional neural networks,* (2017), [Online; accessed 19-Nov-2017], available at https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.

[35] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning,* ArXiv e-prints (2016).

[36] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, *Striving for simplicity: The all convolutional net,* CoRR **abs/1412.6806** (2014), arXiv:1412.6806 .

[37] N. Shibuya, *Up-sampling with transposed convolution,* (2017), [Online; accessed 26-June-2018], available at https://towardsdatascience.com/.

[38] C. Fannjiang and M. Fang, *Nonlinear activations for convolutional neural network acoustic models,* (2016).

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks,* in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12 (Curran Associates Inc., USA, 2012) pp. 1097–1105.

[40] Wikipedia, *Rectifier (neural networks) — Wikipedia, the free encyclopedia,* (2018), [Online; accessed 01-Jul-2018], available at https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[41] *Derivatives of activation functions,* [online lecture], Coursera (2018), [Online; accessed 20-06-2018], available at https://www.coursera.org/lecture/neural-networks-deep-learning/derivatives-of-activation-functions-qcG1j.

[42] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks,* in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, Vol. 15, edited by G. Gordon, D. Dunson, and M. Dudík (PMLR, Fort Lauderdale, FL, USA, 2011) pp. 315–323.

[43] *Rectified linear unit,* TinyMind (2018), [Online; accessed 20-06-2018], available at https://www.tinymind.com/learn/terms/relu.

[44] D. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus),* CoRR **abs/1511.07289** (2015), arXiv:1511.07289 .

[45] S.-I. Amari, *Natural gradient works efficiently in learning,* Neural Comput. **10**, 251 (1998).

[46] Wikipedia, *CIFAR-10 — Wikipedia, the free encyclopedia,* (2018), [Online; accessed 01-Jul-2018], available at https://en.wikipedia.org/wiki/CIFAR-10.

[47] J. Le, *How to do semantic segmentation using deep learning,* (2018), [Online; accessed 01-Jul-2018], available at https://medium.com/nanonets/.

[48] G. Chen, Q. Weng, G. J. Hay, and Y. He, *Geographic object-based image analysis (GEOBIA): emerging trends and future opportunities,* GIScience & Remote Sensing **55**, 159 (2018).

[49] Wikipedia, *Lattice boltzmann methods — Wikipedia, the free encyclopedia,* (2017), [Online; accessed 17-July-2018], available at https://en.wikipedia.org/wiki/Lattice_Boltzmann_methods.

[50] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation,* CoRR **abs/1505.04597** (2015), arXiv:1505.04597 .

[51] T. Salimans, A. Karpathy, X. Chen, D. P. Kingma, and Y. Bulatov, *Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications,* in *Submitted to ICLR 2017* (2016).

[52] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition,* ArXiv e-prints (2015), arXiv:1512.03385 .

[53] C. Olah, *Understanding lstm networks,* (2015), [Online; accessed 20-Jun-2018], available at http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[54] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, *Conditional image generation with pixelcnn decoders,* CoRR **abs/1606.05328** (2016), arXiv:1606.05328 .

[55] W. Shang, K. Sohn, D. Almeida, and H. Lee, *Understanding and improving convolutional neural networks via concatenated rectified linear units,* CoRR **abs/1603.05201** (2016), arXiv:1603.05201 .

[56] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* CoRR **abs/1412.6980** (2014), arXiv:1412.6980 .

[57] *Multi-disciplinary design and performance assessment of effective, agile nato air vehicles,* Technical Activity Proposal (2016), [Online; accessed 04-Dec-2017], available at https://www.sto.nato.int/Lists/test1/activitydetails.aspx?ID=16080.

[58] C. Liersch, *MULDICON design requirements*, Tech. Rep. (NATO STO AVT-251, 2016) unpublished.

[59] R. M. Cummings and A. Schütte, *Integrated computational/experimental approach to unmanned combat air vehicle stability and control estimation, Journal of Aircraft*, Journal of Aircraft **49**, 1542 (2012).

[60] P. Aref, S. T. McGlone, J. Allen, M. Ghoreyshi, A. Jirasek, and A. J. Lofthouse, *Preliminary computational aerodynamic investigation of the nato avt-251 multi-disciplinary configuration,* in *35th AIAA Applied Aerodynamics Conference*, AIAA AVIATION Forum (American Institute of Aeronautics and Astronautics, 2017).

[61] *Glossary of definitions, ground rules, and mission profiles to define air vehicle performance capability,* [Online; accessed 01-Dec-2017], available at http://quicksearch.dla.mil/qsDocDetails.aspx?ident_number=212831 (2008), MIL-STD-3013A, Fig B-12, p.91.

[62] M. Cook, *Flight dynamics principles: a linear systems approach to aircraft stability and control*, third edition ed. (Butterworth-Heinemann, 2013).

[63] T. van Holten, J. Holierhoek, and J. Melkert, *Advanced dynamics,* [Course reader], AE4315 (2008).

[64] F. R. Menter, *Two-equation eddy-viscosity turbulence models for engineering applications,* AIAA Journal **32**, 1598 (2018).

[65] M. v. Rooij and R. M. Cummings, *Aerodynamic design of a ucav in a collaborative framework,* AIAA Conference Paper (2018), (To be submitted for publication).

[66] A. Jirasek, T. Jeans, M. Martenson, R. Cummings, and K. Bergeron, *Improved methodologies for maneuver design of aircraft stability and control simulations,* in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Aerospace Sciences Meetings (American Institute of Aeronautics and Astronautics, 2010) doi: 10.2514/6.2010-515.

[67] M. Schroeder, *Synthesis of low-peak-factor signals and binary sequences with low autocorrelation (corresp.),* IEEE Transactions on Information Theory **16**, 85 (1970).

[68] E. A. Morelli, *Multiple input design for real-time parameter estimation in the frequency domain,* IFAC Proceedings Volumes **36**, 639 (2003).

[69] J. D. Anderson Jr, *Fundamentals of aerodynamics* (Tata McGraw-Hill Education, 2010).

[70] W. Faller, S. Schreck, and M. Luttges, *Real-time prediction and control of three-dimensional unsteady separated flow fields using neural networks,* (American Institute of Aeronautics and Astronautics, 1994).

[71] Wikipedia, *Moving average — Wikipedia, the free encyclopedia,* (2018), [Online; accessed 10-Jul-2018], available at https://en.wikipedia.org/wiki/Moving_average.

[72] D. I. Greenwell, *Frequency effects on dynamic stability derivatives obtained from small-amplitude oscillatory testing, Journal of Aircraft*, Journal of Aircraft **35**, 776 (1998).

[73] S. Hickel, *Cfd for aerospace engineers,* lecture notes, TU Delft (2016).

[74] MathWorks, *Goodness of fit between test and reference data,* (n.d.), [Online; accessed 20-Jun-2018], available at https://nl.mathworks.com/help/ident/ref/goodnessoffit.html.

[75] P. Reisenthel, *Application of nonlinear indicial modeling to the prediction of a dynamically stalling wing,* in *14th Applied Aerodynamics Conference*, Fluid Dynamics and Co-located Conferences (American Institute of Aeronautics and Astronautics, 1996).

[76] M. Ketelaars, *Reduced-order modelling for prediction of aircraft flight dynamics,* Master's thesis, Delft University of Technology (2017), [Online; accessed 01-Jul-2018], available at https://repository.tudelft.nl/.