Master Degree in Aerospace Engineering

A.A. 2020/2021

Iterative And Neural Network Based Methods To Solve A Model-Free Scheme For Flow Angles Estimation

Candidato
Pietro Mascolo Vitale
Matricola: 257370

Responsabili Scientifici
Ing. Angelo Lerro
Prof. Piero Gili

Ciao Papà, avrei voluto condividere con te questo giorno speciale, sentire ancora la tua voce, parlare con te e festeggiare con una bella discesa in kayak, come quelle belle che abbiamo sempre fatto insieme, quelle che mi hanno lasciato il più bel ricordo di te. Un giorno magari rifaremo tutto insieme e potrò raccontarti di questo momento, fino ad allora, buona linea Papà! Questo lavoro è dedicato a te.

Contents

List of Figures VII

List of Tables XII

Abstract XV

Introduction XVI

1 Model Free Scheme For AoA and AoS Estimation 1

1.1 Non Linear Equations Systems Solvers 1

1.1.1 Multi Dimensional Newton's Method 2

1.1.2 Trust Region Method 3

1.1.3 Trust Region Dogleg Method 4

1.2 Notation and Reference Frames 6

1.3 Rearrangement of Flight Mechanic Equations 8

1.4 Problem Formulation 9

1.5 Proposed Scheme 10

1.5.1 Zero-Order ASSE Approximation 10

1.5.2 Zero-Order ASSE Scheme 11

1.5.3 Solution Existence Conditions 12

2 Zero-Order ASSE Scheme Numerical Verification 13

2.1 Maneuver Definition 13

2.2 Linear Solution 15

2.2.1 Zero-Order ASSE Scheme Linearization 15

2.2.2 Numerical Results 16

2.3 Non-Linear Solution 20

2.3.1 Levenberg-Marquardt Algorithm 21

2.3.2 Numerical Results 22

2.3.3 Sensitivity 25

3 Neural Network Theoretical Background 29
3.1 Neural Network Characteristics . . . 29
3.1.1 Definition of a Neural Network . . . 29
3.1.2 Neuron Model . . . 30
3.1.3 Activation Functions . . . 31
3.1.4 Network Architectures . . . 35
3.2 Neural Network Learning Process . . . 40
3.2.1 Error-Correction Learning . . . 40
3.2.2 Supervised And Unsupervised Learning . . . 42
3.2.3 Online And Batch Learning . . . 44
3.2.4 MLP Learning Algorithms . . . 44
3.2.5 Neural Network Validation: Overfitting And Local Minima . 49

4 NN Applied To The Zero Order ASSE Scheme 52
4.1 Network Characteristics And Input Data Profile . . . 53
4.2 Presentation Of The Results . . . 53
4.3 Tests . . . 55
4.3.1 Test Group 1: Effects Of Training Epochs . . . 55
4.3.2 Test Group 2: Effects Of The Number Of Hidden Layers . . 64
4.3.3 Test Group 3: Effects Of The Number Of Neurons For Each Layer . . . 71
4.3.4 Test Group 4: Effects Of The Number Of Training Data . . 79
4.3.5 Combined Tests . . . 87
4.4 Best Network Performance . . . 95
4.4.1 manoeuvre_3211deltae . . . 98
4.4.2 manoeuvre_doublet_deltae_deltar_B . . . 101
4.4.3 manoeuvre_doublet_deltae_deltar_B bis . . . 102
4.5 Sensitivity . . . 104
4.5.1 manoeuvre_deltaedoublet_deltardoublet . . . 106
4.5.2 manoeuvre_3211deltae . . . 108
4.5.3 manoeuvre_doublet_deltae_deltar_B . . . 110
4.5.4 Critical Noisy Data . . . 111

5 Radial Basis Functions Neural Networks 121
5.1 Theoretical Background . . . 121
5.1.1 Radial Basis Functions . . . 122
5.1.2 Radial Basis Functions Networks Architecture . . . 124
5.1.3 Learning Process In RBF Networks . . . 124
5.2 RBF Networks Applied To The Zero Order ASSE Scheme . . . 126

5.2.1 manoeuvre_deltaedoublet_deltardoublet 128
5.2.2 manoeuvre_3211deltae 130
5.2.3 manoeuvre_doublet_deltae_deltar_B 132
5.3 Sensitivity 134
5.3.1 manoeuvre_deltaedoublet_deltardoublet 134
5.3.2 manoeuvre_3211deltae 137
5.3.3 manoeuvre_doublet_deltae_deltar_B 139

Conclusions 141

Appendix 144
A Linear ASSE Scheme Script 144
B Non-Linear ASSE Scheme Script 148

Bibliography 154

List of Figures

1.1 [1] Representation of a) inertial and control volume reference frames, and b) body and wind reference frames 6

2.1 True values of the Angle of Attack (a) and Angle of Sideslip (b) of maneuver_deltaedoublet_deltardoublet_nodrop 14
2.2 Comparison between the true and estimated value of AoA for the linear solution of the Zero Order ASSE Scheme 17
2.3 Comparison between the true and estimated value of AoS for the linear solution of the Zero Order ASSE Scheme 19
2.4 Comparison between the true and estimated value of AoA for the non-linear solution of the Zero Order ASSE Scheme 23
2.5 Comparison between the true and estimated value of AoS for the non-linear solution of the Zero Order ASSE Scheme 24
2.6 Comparison between the true and estimated value of AoA for the non-linear solution of the Zero Order ASSE Scheme with noisy flight data 26
2.7 Comparison between the true and estimated value of AoS for the non-linear solution of the Zero Order ASSE Scheme with noisy flight data 27

3.1 [2] Non-linear model of a neuron 31
3.2 [2] (a) Threshold function, (b) Piecewise-linear function, (c) Sigmoid function for varying slope parameter a. 34
3.3 [2] Fully connect feed forward or acyclic network with one hidden layer and one output layer 37
3.4 [2] Recurrent network with no self-feedback loops and no hidden neurons 38
3.5 [2] Recurrent network with hidden neurons 39
3.6 [2] Illustration of the error correction learning 40
3.7 [2] Block Diagram of Supervised Learning 45

3.8 [2] Block Diagram of Unsupervised Learning 46
3.9 [3] Flow-chart of the neural network training and validation process 50
3.10 [3] Non-linear curve-fitting problem with several local minima, highlighting the initial synaptic weight, w_{init}, a local minimum point, w_{min1}, and the absolute minima, w_{opt}. 51

4.1 Group 1-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 57
4.2 Group 1-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 58
4.3 Group 1-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 59
4.4 Group 1-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 60
4.5 Group 1-Test 5: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 61
4.6 Group 1 Statistic data 63
4.7 Group 2-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 65
4.8 Group 2-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 66
4.9 Group 2-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 67
4.10 Group 2-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 68
4.11 Group 2 Statistic data 70
4.12 Group 3-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 72
4.13 Group 3-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 73
4.14 Group 3-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 74
4.15 Group 3-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 75
4.16 Group 3-Test 5: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 76
4.17 Group 3 Statistic data 78
4.18 Group 4-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 80

4.19 Group 4-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 81
4.20 Group 4-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 82
4.21 Group 4-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 83
4.22 Group 4-Test 5: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 84
4.23 Group 4 Statistic data 86
4.24 Combined Group-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 88
4.25 Combined Group-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 89
4.26 Combined Group-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 90
4.27 Combined Group-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 91
4.28 Combined Group-Test 5: Comparison between the true and the network estimated value of AoA (a) and AoS (b) 92
4.29 Combined group Statistic data 94
4.30 Angle of Attack (a) and Angle of Sideslip (b) of manoeuvre_3211deltae 96
4.31 Angle of Attack (a) and Angle of Sideslip (b) of manoeuvre_doublet_deltae_deltar_B 97
4.32 Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_3211deltae 99
4.33 Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_doublet_deltae_deltar_B 101
4.34 Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_doublet_deltae_deltar_B 103
4.35 Sensitivity Test: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet 106
4.36 Sensitivity Test: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_3211deltae . . 108
4.37 Sensitivity Test: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_doublet_deltae_deltar_B 110

4.38 Noisy body accelerations: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet 112
4.39 Noisy Angular Rates: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet 114
4.40 Noisy TAS: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet 116
4.41 Noisy TASp: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet 118

5.1 Radial Basis Functions varying the parameter ε. Image taken from https://en.wikipedia.org/wiki/Radial_basis_function/media/File:Gaussian_function_shape_parameter.png. Author: Shawsa7 . . . 123
5.2 Radial Basis Functions Architecture. Image taken from: https://en.wikipedia.org/wiki/Radial_basis_function_network/media/File:Radial_funktion_network.svg. Author: SebDE 125
5.3 GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_deltaedoublet_deltardoublet 128
5.4 GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_3211deltae 130
5.5 GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_doublet_deltae_deltar_B 132
5.6 GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_deltaedoublet_deltardoublet with white noise 135
5.7 GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_3211deltae with white noise 137
5.8 GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_doublet_deltae_deltar_B with white noise 139

List of Tables

2.1 Linear Zero Order ASSE Scheme 2σ errors 20
2.2 Non-Linear Zero Order ASSE Scheme 2σ errors 25
2.3 Non-Linear Zero Order ASSE Scheme mean and max errors 25

4.1 Training and Test data for Group 1 55
4.2 Group 1 training characteristics 56
4.3 Statistic data for AoA of Group 1 tests 62
4.4 Statistic data for AoS of Group 1 tests 62
4.5 Training and Test data for Group 2 64
4.6 Group 2 training characteristics 64
4.7 Statistic data for AoA of Group 2 tests 69
4.8 Statistic data for AoS of Group 2 tests 69
4.9 Training and Test data for Group 3 71
4.10 Group 3 training characteristics 71
4.11 Statistic data for AoA of Group 3 tests 77
4.12 Statistic data for AoS of Group 3 tests 77
4.13 Training and Test data for Group 4 79
4.14 Group 4 training characteristics 79
4.15 Statistic data for AoA of Group 4 tests 85
4.16 Statistic data for AoS of Group 4 tests 85
4.17 Training and Test data for the combined group 87
4.18 Combined Group training characteristics 87
4.19 Statistic data for AoA of combined group tests 93
4.20 Statistic data for AoS of combined group tests 93
4.21 Statistic data for AoA and AoS of manoeuvre_3211deltae 100
4.22 Statistic data for AoA and AoS of manoeuvre_doublet_deltae_ deltar_B 102
4.23 New network parameters for testing maneuver deltae_ deltar_B . . 102

4.24 Statistic data for AoA and AoS of manoeuvre_doublet_deltae_ deltar_B . . . 104
4.25 Corrupted input data . . . 104
4.26 Training and Test data for Sensitivity test . . . 105
4.27 Sensitivity test training characteristics . . . 105
4.28 Sensitivity Test: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet . . . 107
4.29 Sensitivity Test: Statistic data for AoA and AoS of manoeuvre_3211deltae . . . 109
4.30 Sensitivity Test: Statistic data for AoA and AoS of manoeuvre_doublet_deltae_deltar_B . . . 111
4.31 Noisy body accelerations: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet . . . 113
4.32 Noisy Angular Rates: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet . . . 115
4.33 Noisy TAS: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet . . . 117
4.34 Noisy TASp: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet . . . 119

5.1 Statistic data for AoA and AoS of manouver_deltaedoublet_ deltardoublet . . . 129
5.2 Statistic data for AoA and AoS of manoeuvre_3211deltae . . . 131
5.3 Statistic data for AoA and AoS of manoeuvre_doublet_ deltae_deltar_B . . . 133
5.4 Statistic data for AoA and AoS of maneuver of manoeuvre_deltaedoublet_deltardoublet with white noise . . . 136
5.5 Statistic data for AoA and AoS of manoeuvre_3211deltae with white noise . . . 138
5.6 Statistic data for AoA and AoS of manoeuvre_doublet_deltae_ deltar_B with white noise . . . 140
5.7 Clear Data: Summary of the results for test maneuvers both for MLP and RBF network . . . 141
5.8 Noisy Data: Summary of the results for test maneuvers both for MLP and RBF network . . . 142

Abstract

Evaluation of aerodynamic, or flow, angles has always been a crucial topic since they are necessary to pilot and control the aircraft. These angles are usually measured using different probes attached to the vehicle fuselage surface which are surrounded with the flow field. However, new solutions have been explored in order to reduce this number of probes so that a better stealthiness and a less heavy impact on the airframe, especially for small UAVs, can be achieved. For this purpose, virtual software-based sensors, based on neural networks prediction techniques have been developed and proven to be suitable for the aerodynamic angles estimation of small UAVs. The aim of this work is to evaluate the accuracy in the estimation of neural network solvers for the angle of attack and the angle of sideslip of a model free scheme named ASSE [1]. In this work, all the tests have been carried out considering a null wind velocity but, for sake of clarity, conclusions of this work are also applicable when the wind speed is not zero. The suitability of the model-free ASSE scheme will be tested for both traditional solution methods (i.e. non-linear solver) and neaural network based methods.

Introduction

Since the dawn of digital computers in 1970 a lot of research has been done over virtual sensors based on soft computing techniques in order to substitute and support the traditional Air Data System (ADS). The ADS is essential for the flight control computer (FCC) to instruct the autopilot in order to guarantee stability and control during an automatic control flight. Among all the parameters provided by the ADS there are two in particular which are the focus of this work, the Angle-of-Attack, (AoA or α) and the Angle-of-Sideslip (AoS or β) also known as the aerodynamic angles. Traditional Air Data Systems are based on vanes, which were first described in detail by Ikard [4], in 1956, for both subsonic and supersonic applications. Another way is to use differential direction probes, as was well documented by Chue [5], Pankhurst and Holder [6] and Yajnik and Gupta [7], starting back in 1952. Modern flow direction Probes are integrated in multifunction probes who can detect both the aerodynamic angles and the static and dynamic pressure. One example can be the measurement of the Angle-of-Sideslip by sensing the difference in the static pressure in the two sides of the aircraft and differentiating between these values, which represents the basic principle of multihole probes. All these sensors are connected to an Air Data Computer (ADS) which provides the Flight Control System (FCS) with the required data. Only slight changements have occurred in the last decade and the basic principle of air data measurement methods have essentially remained unchanged. The main issue with traditional probes is that in order to perform correctly they must be placed in the front of the fuselage, and, as far as small UAVs are concerned, this can cause an interference with opto-electronic equipment and moreover it is also added weight to the structure, which is crucial for a UAVs. If also the required redundancy of the ADS is considered in order to comply with airworthiness regulations the situation can only get worse. All these problems related to the traditional ADS have slowly led, as said in the beginning, to the development of virtual sensors which allow to substitute the expensive, heavy and voluminous traditional probes with executable software codes. Virtual sensors have also introduced to the concept of analytical redundancy that is the process of replacing some of the actual sensors with virtual ones, which can be used as voters

in redundant or simplex sensor systems, to detect inconsistencies of the hardware sensors and can eventually be employed to provide alternative data. In the last two decades, many interesting algorithms have been produced, mostly related to a model-based approach. A practical example of a model-based algorithm is the one developed and patented by Wise [8], which is currently used on the on the Boeing X-45A aircraft. Using inertial data, in addition to an accurate aerodynamic aircraft model and a Kalman filter, the virtual sensor designed by Wise is able to predict the aerodynamic angles with good accuracy. However, the problem with model-based algorithms are the inevitable discrepancies between the mathematical model and the real one, which is very complex in the case of an aircraft. In order to overcome these problems, Neural Networks have been taken into account. Rohloff et al. [9] and Samy and Green [10] described virtual sensors, based on neural networks, that are able to reconstruct complete suite of air data parameters, starting from multiple static pressure measurements on an aircraft fuselage, without using inertial data. All the virtual sensors for aerodynamic angle estimation, whatever the technology on which they are built, they all share the use of dynamic pressure actual values, which is clearly a fundamental air data that is quite complex to estimate. A possible solution could be estimate the dynamic pressure with another virtual sensor. An innovative idea has been introduced in the work of A.Lerro and others [3] where the aerodynamic angles are estimated indirectly by means of neural systems which need inertial data from Attitude and Heading Reference system (AHRS), dynamic pressure from ADS and aircraft commands from FCS as input data. The virtual sensor solution allows one to save, or substitute, physical sensors with software-based ones and this leads to enormous benefits for the redundant systems of unmanned aircraft. Taking into account all the research and development over virtual sensors based on neural network predictions for estimating the Angle-of-Attack and the Angle-of-Sideslip described so far, the main purpose of this work is to evaluate the level of accuracy of Neural Networks in the estimation of the aerodynamic angles. In particular, the present work evaluates the feasibility of neurocomputing in solving a non-linear system of equations born from a rearrangement of flight mechanics equations, defined as the Zero-Order ASSE Scheme [1], which solutions are the Angle-of-Attack and the Angle-of-Sideslip. In Chapter 1 the Zero-Order ASSE Scheme is presented in order to show the equations of interest. In Chapter 2 the ASSE scheme is solved in two different situations (linear case and non-linear case) in order to show the accuracy of the scheme itself. For both the two cases a MatLab routine has been developed. In Chapter 3 Neural Networks characteristics are further discussed and eventually the Zero-Order ASSE scheme is put to test with a Multilayer Perceptron neural network developed in MatLab thanks to the Deep Learning Toolbox.

Chapter 1

Model Free Scheme For Angle-of-Attack and Angle-of-Sideslip Estimation

The aim of this work is to evaluate the accuracy of different methods for solving a non linear system of equation named "Angle of Attack and Sideslip Estimator" (ASSE). Such scheme is the result of a rearrangement of flight mechanics equations, and it has been described in detail in the work of Lerro A., Brandl A. and Gili P. : "Model-Free Scheme for Angle-of-Attack and Angle-of-Sideslip Estimation". This rearrangement leads to a non linear equations system which unknowns are the angle of attack (AoA) and the angle of sideslip (AoS). The first chapter of this work has the purpose of showing the mathematical steps, following the work of Lerro A., Brandl A. and Gili P., to get from the classical flight mechanics equations to the zero order ASSE scheme.

1.1 Non Linear Equations Systems Solvers

Since the main purpose of this work is to test neural network based methods to solve non linear systems of equations, before going in detail on the description of the work of Lerro A., Brandl A. and Gili P.: "Model-Free Scheme for Angle-of-Attack and Angle-of-Sideslip Estimation", a brief introduction on traditional mathematical methods to solve non linear systems of equations is given.

1.1.1 Multi Dimensional Newton's Method

Given a non linear system of n equations in n unknowns:

$$\begin{cases} f_1 = (x_1 ... x_n) = 0 \\ f_2 = (x_1 ... x_n) = 0 \\ \vdots \\ f_n(x_1 ... x_n) = 0 \end{cases} \tag{1.1}$$

Considering:

$$F(x) = \begin{bmatrix} \begin{cases} f_1 = (x_1 ... x_n) = 0 \\ f_2 = (x_1 ... x_n) = 0 \\ \vdots \\ f_n(x_1 ... x_n) = 0 \end{cases} \end{bmatrix} \tag{1.2}$$

The system can be written in the form: $F(x) = 0$. To get the iteration, $F(x)$ is approximated with the first order truncation of the Taylor series:

$$F(x) = F(x^k) + F'(x^k)(x - x^k) \tag{1.3}$$

In order to solve the system, the Jacobian matrix $F'(x^k)$ must be computed:

$$F'(x^k) = \begin{bmatrix} \frac{df_1}{x_1}(x_1^k, ...x_n^k) & \dots & \dots & \frac{df_1}{x_n}(x_1^k, ...x_n^k) \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \frac{df_n}{x_1}(x_1^k, ...x_n^k) & \dots & \dots & \frac{df_n}{x_n}(x_1^k, ...x_n^k) \end{bmatrix} \tag{1.4}$$

The iterative procedure can be generalized as follows:

$$x^{k+1} = x^k - (F'(x^k))^{-1} F(x^k) \tag{1.5}$$

Convergence is obtained when $F(x)$ tends to zero, that means it becomes smaller than an established value. The iterations start from an arbitrary starting point x_0. The choice of the starting point x_0 is critical to determine whether the result will converge or not, and if it does, whether it will converge to a particular solution among all the possible ones. Newton method has a high computational cost since the Jacobian matrix must be computed in each iteration. To solve this problem, there are other methods which allow to approximate the the Jacobian matrix, and these are the Newton-Jacobi and the Newton-Gauss-Seidel methods:

- Newton-Jacobi: $F'(x) = D(x)$
- Newton-Gauss-Seidel: $F'(x) = D(x) - L(x)$

Where $D(x)$ is the diagonal matrix and $L(x)$ is the lower triangular matrix.

1.1.2 Trust Region Method

Given a set of n nonlinear functions $F_i(x)$, where n is the number of components of the vector x, the goal of equation solving is to find a vector x that makes all $F_i(x) = 0$. The Trust Region Method is an optimization method used in many computational solvers (i.e. fsolve in MatLab) which are based on the minimization of the sum of squares of the components. If the sum of squares is zero, the system of equation is solved. To understand the trust-region approach to optimization, consider the unconstrained minimization problem, minimize $f(x)$, where the function takes vector arguments and returns scalars. Suppose you are at a point x in n-space and you want to improve, i.e., move to a point with a lower function value. The basic idea is to approximate f with a simpler function q, which reasonably reflects the behavior of function f in a neighborhood N around the point x. This neighborhood is the trust region. A trial step s is computed by minimizing (or approximately minimizing) over N. This is the trust-region subproblem, $min(q(s)),\ s \in N$. The current point is updated to be $x + s$ if $f(x + s) < f(x)$; otherwise, the current point remains unchanged and N, the region of trust, is shrunk and the trial step computation is repeated. The key questions in defining a specific trust-region approach to minimizing $f(x)$ are how to choose and compute the approximation q (defined at the current point x), how to choose and modify the trust region N, and how accurately to solve the trust-region subproblem. This section focuses on the unconstrained problem. In the standard trust-region method, the quadratic approximation q is defined by the first two terms of the Taylor approximation to F at x; the neighborhood N is usually spherical or ellipsoidal in shape. Mathematically the trust-region subproblem is typically stated:

$$\min_s \{\frac{1}{2} s^T H s + s^T g \ such\ that\ ||Ds|| \leq \Delta\} \tag{1.6}$$

where g is the gradient of f at the current point x, H is the Hessian matrix (the symmetric matrix of second derivatives), D is a diagonal scaling matrix, Δ is a positive scalar, and $||.||$ is the 2-norm. Good algorithms exist for solving 1.6; such algorithms typically involve the computation of a full eigensystem and a Newton process applied to the secular equation:

$$\frac{1}{\Delta} - \frac{1}{||s||} = 0 \tag{1.7}$$

Such algorithms provide an accurate solution to Eq. 1.6. However, they require time proportional to several factorizations of H. Therefore, for trust-region problems a different approach is needed. Several approximation and heuristic strategies, based on Eq. 1.6, have been proposed in the literature. The approximation approach is to restrict the trust-region subproblem to a two-dimensional subspace S. Once the subspace S has been computed, the work to solve Eq. 1.6 is trivial even if full eigenvalue/eigenvector information is needed (since in the subspace, the problem is only two-dimensional). The dominant work has now shifted to the determination of the subspace. The two-dimensional subspace S is determined with the aid of a preconditioned conjugate gradient process which is not described here. The solver defines S as the linear space spanned by $s1$ and $s2$, where $s1$ is in the direction of the gradient g, and $s2$ is either an approximate Newton direction, i.e., a solution to:

$$H * s_2 = -g \tag{1.8}$$

or a direction of negative curvature:

$$s_2^T * H * s_2 < 0 \tag{1.9}$$

The philosophy behind this choice of S is to force global convergence (via the steepest descent direction or negative curvature direction) and achieve fast local convergence (via the Newton step, when it exists). A sketch of unconstrained minimization using trust-region ideas is now easy to give:

- Formulate the two-dimensional trust-region subproblem
- Solve Equation 1.6 to determine the trial step s
- If $f(x + s) < f(x)$, then $x = x + s$
- Adjust Δ

These four steps are repeated until convergence. The trust-region dimension Δ is adjusted according to standard rules. In particular, it is decreased if the trial step is not accepted, i.e., $f(x + s) \geq f(x)$.

1.1.3 Trust Region Dogleg Method

Another approach is to solve a linear system of equations to find the search direction, namely, Newton's method says to solve for the search direction d_k such that:

$$J(x_k)d_k = -F(x_k)x_{k+1} = x_k + d_k \tag{1.10}$$

where $J(x_k)$ is the n-by-n Jacobian:

$$J(x_k) = \begin{bmatrix} \nabla F_1(x_k)^T \\ \nabla F_2(x_k)^T \\ \vdots \\ \nabla F_n(x_k)^T \end{bmatrix} \tag{1.11}$$

Newton's method can run into difficulties. $J(x_k)$ may be singular, and so the Newton step d_k is not even defined. Also, the exact Newton step d_k may be expensive to compute. In addition, Newton's method may not converge if the starting point is far from the solution. Using trust-region techniques improves robustness when starting far from the solution and handles the case when $J(x_k)$ is singular. To use a trust-region strategy, a merit function is needed to decide if x_{k+1} is better or worse than x_k. A possible choice is:

$$\min_d \{\frac{1}{2} F(x_k + d)^T F(x_k + d)\} \tag{1.12}$$

But a minimum of $f(d)$ is not necessarily a root of $F(x)$. The Newton step d_k is a root of:

$$M(x_k + d) = F(x_k) + J(x_k)d \tag{1.13}$$

and so it is also a minimum of $m(d)$, where:

$$\begin{aligned} \min_d m(d) =& \frac{1}{2}||M(x_k + d)||_2^2 = \frac{1}{2}||F(x_k) + J(x_k)d||_2^2 = \\ =& \frac{1}{2}F(x_k)^T F(x_k) + d^T J(x_k)^T F(x_k) + \frac{1}{2} d^T J(x_k)^T J(x_k) d \end{aligned} \tag{1.14}$$

Then $m(d)$ is a better choice of merit function than $f(d)$, and so the trust-region subproblem is:

$$\min_d m(d) = \frac{1}{2}F(x_k)^T F(x_k) + d^T J(x_k)^T F(x_k) + \frac{1}{2} d^T J(x_k)^T J(x_k) d \tag{1.15}$$

such that $||D * d|| \leq \Delta$. This subproblem can be efficiently solved using a dog-leg strategy. The key feature of this algorithm is the use of the Powell dogleg procedure for computing the step d, which minimizes Eq. 1.15. The step d is constructed from a convex combination of a Cauchy step (a step along the steepest descent direction) and a Gauss-Newton step for $f(x)$. The Cauchy step is calculated as $d_C = -\alpha J(x_k)^T F(x_k)$, where α is chosen to minimize Eq. 1.14. The Gauss-Newton step is calculated by solving $J(x_k) * d_{GN} = -F(x_k)$. The step d is chosen

so that $d = d_C + \lambda(d_{GN} - d_C)$, where λ is the largest value in the interval [0,1] such that $||d|| \leq \Delta$. If J_k is (nearly) singular, d is just the Cauchy direction. The dogleg algorithm is efficient since it requires only one linear solve per iteration (for the computation of the Gauss-Newton step). Additionally, it can be more robust than using the Gauss-Newton method with a line search.
Another algorithm used for solving non linear systems of equations is the **Levenberg-Marquardt** algorithm, which is the one used in this work. A detailed explanation of this algorithm is presented in subsection 2.3.1.

1.2 Notation and Reference Frames

In this section the reference frame used in the work "Model-Free Scheme for Angle-of-Attack and Angle-of-Sideslip Estimation" [1] is presented. Vectors are indicated with bold italic lower case letters (e.g. $\mathbf{v}$), vector components with lower case letters (e.g. v), and matrices with bold-italic capital letters (e.g. $\mathbf{A}$).

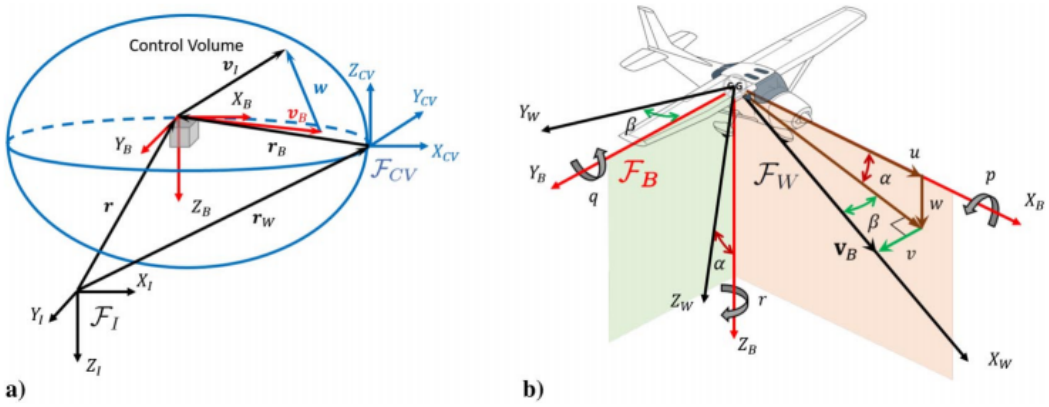


Figure 1.1: [1] Representation of a) inertial and control volume reference frames, and b) body and wind reference frames

In reference to Fig.1.1, there is an inertial reference frame $\mathcal{F}_I = \{X_I, Y_I, Z_I\}$ and two non inertial reference frames: $\mathcal{F}_B = \{X_B, Y_B, Z_B\}$ centered in the center of gravity (CG) and with axes oriented along fixed directions onboard (Fig. 1.1b), $\mathcal{F}_W = \{X_W, Y_W, Z_W\}$ with axes also centered in CG and the X axis aligned to the freestream velocity vector, the Z axis as the intersection of the plane normal to the trajectory and the (X_B, Z_B) plane of the aircraft and directed downward (i.e., from the upper to the lower wind surface). The aircraft is also considered surrounded by an air mass enclosed in a virtual control volume that moves together with its

own reference system $\mathcal{F}_{CV} = \{X_{CV}, Y_{CV}, Z_{CV}\}$. From Fig. 1.1a we can say that the distance $\mathbf{r}$ between the aircraft and $\mathcal{F}_I$ is $\mathbf{r} = \mathbf{r}_B + \mathbf{r}_W$. The angular velocity of $\mathcal{F}_B$ with respect to $\mathcal{F}_I$ is:

$$\boldsymbol{\omega} = p\hat{\boldsymbol{i}}_B + q\hat{\boldsymbol{j}}_B + r\hat{\boldsymbol{k}}_B$$

where $\hat{\boldsymbol{i}}_B$, $\hat{\boldsymbol{j}}_B$ and $\hat{\boldsymbol{k}}_B$ are the unit vectors in $\mathcal{F}_B$. Deriving with respect to time [11] it is possible to write:

$$\dot{\mathbf{r}} = \mathbf{v}_I = \dot{\mathbf{r}}_B + \mathbf{w} \tag{1.16}$$

$\dot{r} = \mathbf{v}_I$ is the inertial velocity, $\dot{r}_B$ is the relative velocity between the aircraft and the surrounding air, and $\mathbf{w}$ is the velocity of the control volume, or wind speed. In order to switch from $\mathcal{F}_I$ to $\mathcal{F}_B$ a vector transformation is applied considering the 3-2-1 ordered sequence of Euler angles: heading ψ, elevation θ, and bank ϕ.

$$\mathbf{C}_{I2B} = \begin{bmatrix} C_\theta C_\psi & C_\theta S_\psi & -S_\theta \\ S_\phi S_\theta C_\psi - C_\phi S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & S_\phi C_\theta \\ C_\phi S_\theta C_\psi + S_\phi S_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi & C_\phi C_\theta \end{bmatrix} \tag{1.17}$$

$\mathbf{C}_{I2B}$ is the full rotation matrix from $\mathcal{F}_I$ to $\mathcal{F}_B$ where C stands for the cosine function and S for the sine function and arguments are indicated as subscript. The full rotation matrix from $\mathcal{F}_W$ to $\mathcal{F}_B$ is:

$$\mathbf{C}_{W2B} = \begin{bmatrix} C_\alpha C_\beta & -C_\alpha S_\beta & -S_\alpha \\ S_\beta & C_\beta & 0 \\ S_\alpha C_\beta & -S_\alpha S_\beta & C_\alpha \end{bmatrix} \tag{1.18}$$

Furthermore, it is also important to highlight that [12]:

$$\mathbf{C}_{I2B}\dot{\mathbf{C}}_{B2I} = \boldsymbol{\Omega}_B \tag{1.19}$$

and

$$\boldsymbol{\Omega}_B = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \tag{1.20}$$

1.3 Rearrangement of Flight Mechanic Equations

Eq. (1.16) can be rewritten, recalling velocity definitions [13] as

$$\mathbf{v}_I = \mathbf{C}_{B2I}\mathbf{v}_B + \mathbf{w} \tag{1.21}$$

and from Eq.(1.18):

$$\mathbf{v}_B = V_\infty \hat{\boldsymbol{i}}_{WB} \tag{1.22}$$

where $V_\infty = |\mathbf{v}_B| = \sqrt{u^2+v^2+w^2}$, and $\hat{\boldsymbol{i}}_{WB} = \left(C_\beta C_\alpha\right)\hat{\boldsymbol{i}}_B + \left(S_\beta\right)\hat{\boldsymbol{j}}_B + \left(C_\beta S_\alpha\right)\hat{\boldsymbol{k}}_B$, i.e., the unit vector of the relative velocity in the body reference frame. Multiplying $\mathbf{C}_{I2B}$ to Eq. (1.21) and deriving with respect to time, considering Eq. (1.19) we get the the inertial acceleration projected on the body reference frame:

$$\mathbf{a}_B = \mathbf{C}_{I2B}\mathbf{a}_I = \dot{\mathbf{v}}_B + \boldsymbol{\Omega}_B\mathbf{v}_B + \mathbf{C}_{I2B}\dot{\mathbf{w}} \tag{1.23}$$

From this, it is then possible to write:

$$\dot{\mathbf{v}}_B = \mathbf{a}_B - \boldsymbol{\Omega}_B\mathbf{v}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}} \tag{1.24}$$

From Eq. (1.22) deriving with respect to time we get $\dot{V}_\infty = \frac{\mathbf{v}_B^T\dot{\mathbf{v}}_B}{V_\infty}$ and substituting $\dot{\mathbf{v}}_B$ with its expression of Eq. (1.24), the following is obtained:

$$\dot{V}_\infty V_\infty = \mathbf{v}_B^T\dot{\mathbf{v}}_B = \mathbf{v}_B^T\left(\mathbf{a}_B - \boldsymbol{\Omega}_B\mathbf{v}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}}\right) = \mathbf{v}_B^T\left(\mathbf{a}_B - \mathbf{C}_{I2B}\right)\dot{\mathbf{w}} \tag{1.25}$$

where $\mathbf{v}_B^T\boldsymbol{\Omega}_B\mathbf{v}_B$ is null, and all terms refer to the same time instant.

1.4 Problem Formulation

The main hypothesis of this scheme is that the relative velocity $\mathbf{v}_B$ and hence the aerodynamic angles at a certain time t can be modelled using information from the past. Consequently, by means of the integral definition, the relative velocity vector at time t can be expressed starting from at a generic time τ, with $t \geq \tau$, as

$$\mathbf{v}_B(t) = \mathbf{v}_B(\tau) + \int_\tau^t \dot{\mathbf{v}}_B(\mathcal{T})\, d\mathcal{T} \tag{1.26}$$

From Eq. (1.24) and Eq. (1.26) it can be derived that:

$$\mathbf{v}_{B,t} = \mathbf{v}_{B,\tau} + \int_\tau^t (\mathbf{a}_B - \mathbf{\Omega}_B \mathbf{v}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}})\, d\mathcal{T} \tag{1.27}$$

and

$$\mathbf{v}_{B,\tau} = \mathbf{v}_{B,t} - \int_\tau^t \mathbf{a}_B\, d\mathcal{T} + \int_\tau^t \mathbf{\Omega}_B \mathbf{v}_B\, d\mathcal{T} + \int_\tau^t \mathbf{C}_{I2B}\dot{\mathbf{w}}\, d\mathcal{T} \tag{1.28}$$

where $\mathbf{v}_{B,t}$ stands for $\mathbf{v}_B(t)$. Substituting Eq. (1.28) to Eq. (1.25) leads to:

$$\begin{aligned}
&V_{\infty,\tau}\dot{V}_{\infty,\tau} = \\
&= \left[\mathbf{v}_{B,t} - \int_\tau^t \mathbf{a}_B\, d\mathcal{T} + \int_\tau^t \mathbf{\Omega}_B \mathbf{v}_B\, d\mathcal{T} + \int_\tau^t \mathbf{C}_{I2B}\dot{\mathbf{w}}\, d\mathcal{T}\right]^T (\mathbf{a}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}})_\tau \Rightarrow \\
&\Rightarrow V_{\infty,\tau}\dot{V}_{\infty,\tau} + \left[\int_\tau^t \mathbf{a}_B\, d\mathcal{T} - \int_\tau^t \mathbf{C}_{I2B}\dot{\mathbf{w}}\, d\mathcal{T}\right]^T (\mathbf{a}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}})_\tau = \\
&= \left[\mathbf{v}_{B,t} + \int_\tau^t \mathbf{\Omega}_B \mathbf{v}_B\, d\mathcal{T}\right]^T (\mathbf{a}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}})_\tau
\end{aligned} \tag{1.29}$$

Where all the terms depending from the aerodynamic angles $\mathbf{v}_B$ are collected on the right hand side.

1.5 Proposed Scheme

The idea behind the "Angle of Attack and Sideslip Estimator" (ASSE) is to make the dependece from the term $\mathbf{v}_B$, which is further expressed in terms of the aerodynamic angles, explicit. For this purpose, the integral term $\int_\tau^t \mathbf{\Omega}_B \mathbf{v}_B \, d\mathcal{T}$ of Eq. (1.29) must be explicited in the variable $\mathbf{v}_B$. In order to achieve this, the work of Lerro A., Brandtl A. and Gili P. proposes an approximation identified as the zero order approximation, based on the hypothesis that the integrand function $\Omega_B \mathbf{v}_B$ is constant in the time interval $[\tau, t]$.

1.5.1 Zero-Order ASSE Approximation

If the hypothesis that $\Omega_B \mathbf{v}_B$ is constant in the time interval $[\tau, t]$ is considered, then

$$\int_\tau^t \mathbf{\Omega}_B \mathbf{v}_B \, d\mathcal{T} = (\mathbf{\Omega}_B \mathbf{v}_B)_t \, \Delta t, \tag{1.30}$$

where $\Delta t = t - \tau$. Substituting the latter expression into Eq. (1.29) and recalling matrix properties, Eq. (1.29) can be rewritten as

$$\begin{aligned} &V_{\infty,\tau}\dot{V}_{\infty,\tau} + \left[\int_\tau^t \mathbf{a}_B \, d\mathcal{T} - \int_\tau^t \mathbf{C}_{I2B}\dot{\mathbf{w}} \, d\mathcal{T}\right]^T (\mathbf{a}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}})_\tau = \\ &= V_{\infty,t}\hat{\mathbf{i}}_{WB,t}^T \left(\mathbf{I} - \mathbf{\Omega}_{B,t}\Delta t\right) (\mathbf{a}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}})_\tau \end{aligned} \tag{1.31}$$

Eq. (1.31) is the basic expression of the zero-order scheme referred to the generic time τ where the aerodynamic angles $\alpha(t)$ and $\beta(t)$ are the only unknowns and all other terms are supposed to be measured. Basing on this scheme, in order to calculate the aerodynamic angles, a direct measure of 1) true airspeed V_∞ and its time derivative $\dot{V}_\infty$, 2) the inertial body acceleration $\mathbf{a}_B$, 3) angular rates, and 4) the wind field, it's needed. The wind velocity is assumed to be known in order to be able to measure the wind acceleration term $\dot{\mathbf{w}}$ in Eq. (1.31). Conclusion of this work can always be applicable in the case of null, steady wind field or discrete wind change.

1.5.2 Zero-Order ASSE Scheme

To ease the notation, some terms of Equations (1.31) are grouped as follows

$$n_\tau = V_{\infty,\tau}\dot{V}_{\infty,\tau} + \left[\int_\tau^t \mathbf{a}_B\, d\mathcal{T} - \int_\tau^t \mathbf{C}_{I2B}\dot{\mathbf{w}}\, d\mathcal{T}\right]^T (\mathbf{a}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}})_\tau \tag{1.32}$$

and

$$\mathbf{m}_\tau = V_{\infty,t}\left(\mathbf{I} - \mathbf{\Omega}_{B,t}\Delta t\right)(\mathbf{a}_B - \mathbf{C}_{I2B}\dot{\mathbf{w}})_\tau = h_\tau\hat{\boldsymbol{\imath}}_B + l_\tau\hat{\boldsymbol{\jmath}}_B + m_\tau\hat{\boldsymbol{k}}_B \tag{1.33}$$

Eq. (1.31) can be rewritten in a more compact form

$$n_\tau = \hat{\mathbf{i}}_{WB,t}^T\mathbf{m}_\tau = h_\tau C_\beta C_\alpha + l_\tau S_\beta + m_\tau C_\beta S_\alpha \tag{1.34}$$

Eq. (1.34) represents a generic nonlinear scalar equation in two variables $\alpha(t)$ and $\beta(t)$. Eq. (1.34) can be expanded back in time starting from t to n-th generic τ_i with $i \in [0, 1, \ldots, n]$ where $\tau_0 \equiv t$. Therefore, the following system of $n+1$ nonlinear equations is obtained:

$$\begin{cases} n_t = \hat{\mathbf{i}}_{WB,t}^T\mathbf{m}_t = h_t C_\beta C_\alpha + l_t S_\beta + m_t C_\beta S_\alpha \\ n_{\tau_1} = \hat{\mathbf{i}}_{WB,t}^T\mathbf{m}_{\tau_1} = h_{\tau_1} C_\beta C_\alpha + l_{\tau_1} S_\beta + m_{\tau_1} C_\beta S_\alpha \\ \vdots \\ n_{\tau_n} = \hat{\mathbf{i}}_{WB,t}^T\mathbf{m}_{\tau_n} = h_{\tau_n} C_\beta C_\alpha + l_{\tau_n} S_\beta + m_{\tau_n} C_\beta S_\alpha \end{cases} \tag{1.35}$$

Eq. (1.35) is the generic form of the proposed zero-order ASSE scheme based on $n+1$ equations. Both an expansion in the past ($\tau_{i+1} < \tau_i$) and a forward expansion are equally feasible leading to the same conclusions. Moreover, it is worth highlighting that no hypothesis are assumed on time spacing of time steps considered here. In fact, even though very uncommon, nonuniform time spacing can be considered and two subsequent equations can also be written for two nonadjacent time steps. This latter aspect can be useful to improve the condition number of the system in Equation (1.35). The nonlinear system of Equation (1.35) can be rewritten in a more compact matrix form as

$$\mathbf{n}_n = \left(\hat{\mathbf{i}}_{WB,t}^T\boldsymbol{\mathcal{M}}_n^T\right)^T = \boldsymbol{\mathcal{M}}_n\hat{\mathbf{i}}_{WB,t} \tag{1.36}$$

where $\mathbf{n}_n = [n_t, n_{\tau_1}, \cdots, n_{\tau_n}]^T$ and $\boldsymbol{\mathcal{M}}_n = \left[\mathbf{m}_t^T, \mathbf{m}_{\tau_1}^T, \cdots, \mathbf{m}_{\tau_n}^T\right]^T$. Since the components of the unit vector are not independent, an extra condition is given by the unit magnitude constraint. Therefore, the nonlinear system of equations based on the zero-order ASSE scheme can be expressed as

$$\begin{cases} 1 = \hat{\mathbf{i}}_{WB,t}^T \hat{\mathbf{i}}_{WB,t} \\ \mathbf{n}_n = \boldsymbol{\mathcal{M}}_n \hat{\mathbf{i}}_{WB,t} \end{cases} \tag{1.37}$$

The most suitable solver can be adopted to solve the system of nonlinear Eq. (1.37) for AoA and AoS estimation.

1.5.3 Solution Existence Conditions

Considering the hypothesis that the system of Eq. (1.37) could be linear, it can be written as

$$\mathbf{n}_n^* = \boldsymbol{\mathcal{M}}_n^* \hat{\mathbf{i}}_{WB,t} \tag{1.38}$$

where $\mathbf{n}_n^* = \left[1, \mathbf{n}_n^T\right]^T$ and $\boldsymbol{\mathcal{M}}_n^* = \left[\hat{\mathbf{i}}_{WB,t}^T, \boldsymbol{\mathcal{M}}_n\right]^T$. If $\boldsymbol{\mathcal{M}}_n^*$ was invertible, and hence $n = 1$ in order to have a square matrix, the ASSE solution would be obtained as

$$\hat{\mathbf{i}}_{WB,t} = \boldsymbol{\mathcal{M}}_1^{*-1} \mathbf{n}_1^* \tag{1.39}$$

In order for the Eq. (1.38) to be solvable, at least two time steps are needed ($\tau_0 \equiv t$ and τ_1), since there are two unknowns $\alpha(t)$ and $\beta(t)$, thus two equations are necessary. For the non linear solution, this sets only the minimum number of equations required. Furthermore, in order to have a unique solution, the determinant of $\boldsymbol{\mathcal{M}}_n^*$ must not be zero. If we consider Eq. (1.32) and (1.33), it is clear that the zero order ASSE scheme is not suitable for uniform flight conditions, since the i-th time step τ_i would introduce an equation leading to a null determinant of $\boldsymbol{\mathcal{M}}_1^*$. So, in order for the matrix $\boldsymbol{\mathcal{M}}_1^*$ to have a full rank, it is needed that each step τ_i shall add an independent equation such that $\boldsymbol{\mathcal{M}}_1^*$ has only linearly independent rows. Hence, as also observed in [14], the analytical aerodynamic angle estimation (based on a model-free approach) cannot be performed in uniform flight (or trim) conditions. Therefore, assuming that Eq. (1.37) may be linearized, general conditions on the existence of AoA and AoS solutions based on the proposed ASSE scheme are 1) at least two time steps (τ_0 and τ_1) available, 2) not uniform flight conditions, and 3) two independent equations.

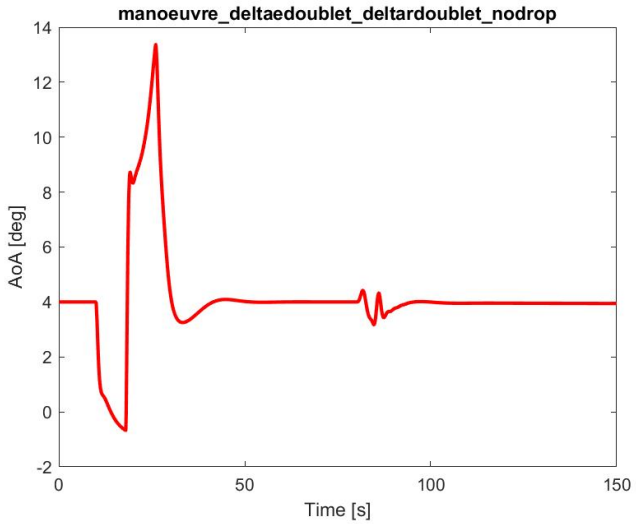
Chapter 2

Zero-Order ASSE Scheme Numerical Verification

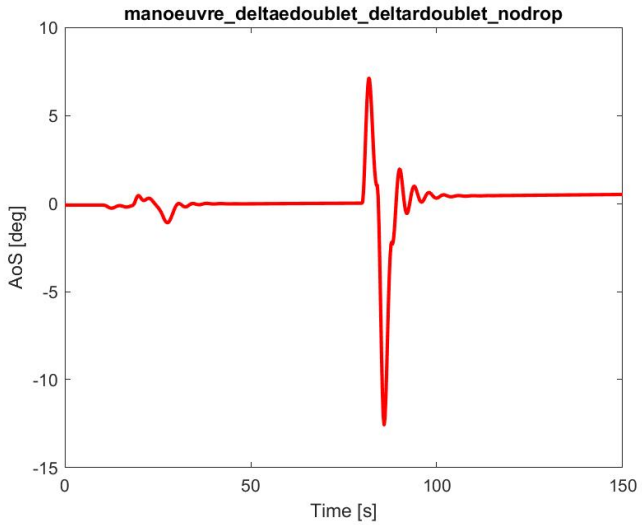
In order to test the level of accuracy of the Zero-Order ASSE scheme, different maneuvers have been tested. Knowing the values of the variables n_t, h_t,l_t and m_t of Eq. (1.35) for at least two time instants (section 1.5.3) of a certain maneuver, it is possible to solve Eq. (1.35) for $\alpha(t)$ and $\beta(t)$ and comparing this output with the real values of the AoA and AoS. For this purpose, two MatLab scripts have been developed: one for the linear case and the other for the non-linear case (Appendix A/B).

2.1 Maneuver Definition

The numerical validation is performed using a flight simulator, inspired to a two-seat light motorized aircraft. The simulator is based on a coupled six-degree-of-freedom nonlinear aircraft model equipped with nonlinear aerodynamic and thrust models designed accordingly to flight test results and the engine datasheet. The maneuver performed is a stall maneuver described in figure 2.1. The maneuver begins in trim conditions. Aileron commands are maintained to their trim positions even though, due to gyroscopic effects, the trim flight condition is not perfectly symmetric. For this latter reason, the aileron during the trim condition is nonzero and both longitudinal and lateral-directional modes are always slightly coupled. After a short dive, the stall maneuver is performed acting on the sole elevator command producing initially an increase of airspeed and then a smooth deceleration leading to high angle of attack with limited changes in the angle of sideslip.



(a) AoA



(b) AoS

Figure 2.1: True values of the Angle of Attack (a) and Angle of Sideslip (b) of maneuver_deltaedoublet_deltardoublet_nodrop

2.2 Linear Solution

The accuracy of the ASSE scheme has first been tested in the hypothesis of small values for the Angle-of-Attack and Angle-of-Sideslip. This hypothesis allows to consider i.e. $\cos(x) = 1$ and $\sin(x) = x$ thus changing Eq. (1.35) from non-linear to linear and making the solving process less complex.

2.2.1 Zero-Order ASSE Scheme Linearization

Under the assumption that $\alpha(t)$ and $\beta(t)$ are small, it is then possible to write that: $\cos(\alpha(t)) \simeq 1$, $\sin(\alpha(t)) \simeq \alpha(t)$ and $\cos(\beta(t)) \simeq 1$, $\sin(\beta(t)) \simeq \beta(t)$. Substituting this in Eq. (1.35) leads to:

$$\begin{cases} n_t = h_t + l_t\beta + m_t\alpha \\ n_{\tau_1} = h_{\tau_1} + l_{\tau_1}\beta + m_{\tau_1}\alpha \end{cases} \tag{2.1}$$

As already described in sub-section 1.5.3, in case the ASSE scheme is linearized, only two linear independent equations are needed to calculate the values of $\alpha(t)$ and $\beta(t)$. For this purpose, it is necessary to compute the values of the parameters n_t,h_t,l_t and m_t at two different time steps. The MatLab algorithm developed for this purpose (Appendix A) considers two subsequent time steps ($\tau = t - 1$) in the Time History of the maneuver. All the parameters are then computed, following the equations shown in chapter 1, for both the time steps, leading to:

$$\begin{bmatrix} l_t & m_t \\ l_{t-1} & m_{t-1} \end{bmatrix} \begin{bmatrix} \beta(t) \\ \alpha(t) \end{bmatrix} = \begin{bmatrix} n_t - h_t \\ n_{t-1} - h_{t-1} \end{bmatrix} \tag{2.2}$$

The aerodynamic angles are then easily calculated. This process is extended, two equations at a time, through all the Time History, thus providing the values of AoA and AoS for each time step t.

2.2.2 Numerical Results

In this subsection the results obtained for the linear solution of the Zero Order ASSE Scheme are presented. The computation of these results has been achieved with a MatLab routine (Appendix A) developed for this purpose. The following results are presented with four graphs:

1. Comparison between the true and estimated value of AoA
2. Error in the estimation of AoA, where the error is considered as $AoA_{err} = AoA_{Estimated} - AoA_{True}$
3. Comparison between the true and estimated value of AoS
4. Error in the estimation of AoS, where the error is considered as $AoS_{err} = AoS_{Estimated} - AoS_{True}$

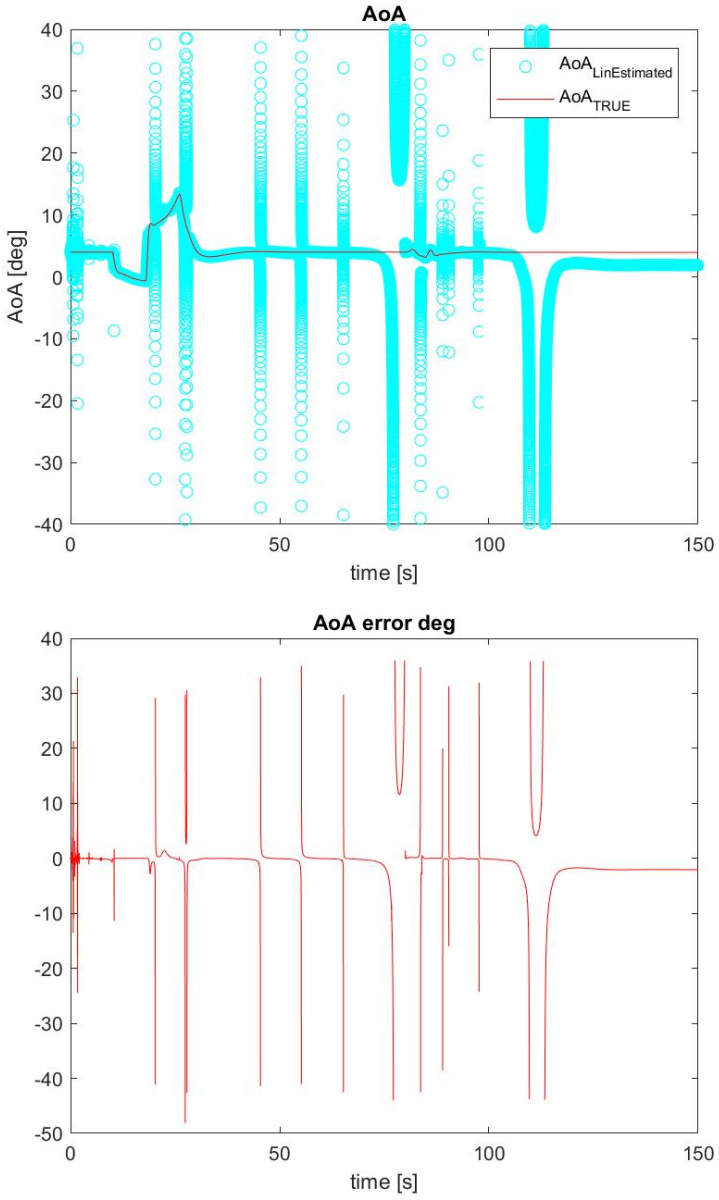


Figure 2.2: Comparison between the true and estimated value of AoA for the linear solution of the Zero Order ASSE Scheme

To evaluate these results, the requirements described in the work of Lerro A., Brandl A., Battipede M. and Gili P. [15] are taken into account. According to these requirements, the error probability of two standard deviations $Pr(-2\sigma \leq X \leq 2\sigma) = 95.4\%$ for the angle of attack in an extended flight envelope, must be less or equal to 1.5 deg. The results shown in 2.2 have a 2σ error of 8.4273 deg, thus making the linear solution for the angle of attack not suitable. The same conclusion applies for the angle of sideslip, shown in figure 2.3, with a 2σ error of 16.3904 deg over an expected value of 2.5 deg.

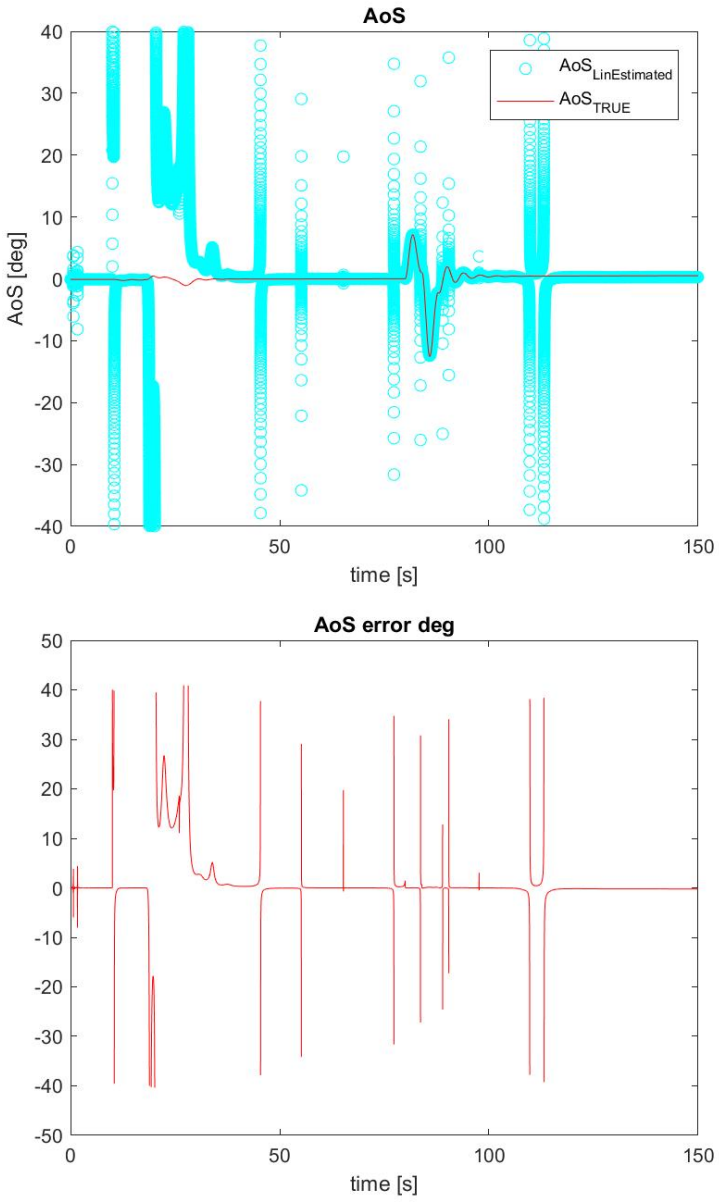


Figure 2.3: Comparison between the true and estimated value of AoS for the linear solution of the Zero Order ASSE Scheme

A summary of the results is given in the following table:

Aerodynamic Angle	2σ error[deg]	Required 2σ error[deg]
α	8.4273	1.5
β	16.3904	2.5

Table 2.1: Linear Zero Order ASSE Scheme 2σ errors

As expected, the simplifications introduced by the linearization of equation (1.35) produce not suitable results. For sake of clarity, the linear approximation holds for small values of $\alpha(t)$ and $\beta(t)$, which is not the case of the stall maneuver here tested. The linear scheme might result suitable for a different maneuver. Since the results produced with correct data do not satisfy the requirements, further analysis with white noise corrupted data haven't been carried out for the linear case.

2.3 Non-Linear Solution

This part of the work has been dedicated at experimenting the real level of accuracy of the Zero-Order ASSE Scheme without considering any hypothesis on the aerodynamic angles. The starting point in this case is equation (1.35). Since this time the problem is non-linear, given all the conditions described in subsection 1.5.3, two equations only represent the minimum number of equations needed to solve the system for $\alpha(t)$ and $\beta(t)$. However, it has been proven that the estimation accuracy is not significantly improved by increasing the number of equations and that for the zero-order scheme, the best trade-off is obtained using only two equations [1]. Because of this, the algorithm developed to test the behavior of the non-linear Zero-Order ASSE Scheme (Appendix B) considers only two equations at a time for two subsequent time steps, as for the linear case described in sub-section 2.2.1. The MatLab solver exploited for this purpose is the function **"fsolve"** which take as input parameters the system of equation to be solved, the starting point X_0 which is where the iterative algorithm chosen will start evaluating the solution and the "options" which is were such algorithm and different other parameters can be set. The equation system is passed in the form:

$$\begin{bmatrix} n_t \\ n_{t-1} \end{bmatrix} - \begin{bmatrix} h_t \\ h_{t-1} \end{bmatrix} \left[C_{\beta(t)} C_{\alpha(t)} \right] - \begin{bmatrix} l_t \\ l_{t-1} \end{bmatrix} \left[S_{\beta(t)} \right] - \begin{bmatrix} m_t \\ m_{t-1} \end{bmatrix} \left[C_{\beta(t)} S_{\alpha(t)} \right] = 0 \quad (2.3)$$

The iterative algorithm chosen for the solver is the Levenberg-Marquardt algorithm, which is further discussed later in this work.

2.3.1 Levenberg-Marquardt Algorithm

The scope of this subsection is to give a brief description of the Levenberg-Marquardt algorithm and it refers to the work of Lourakis, Manolis and others [16]. The Levenberg-Marquardt (LM) algorithm is an iterative technique that locates the minimum of a multivariate function that is expressed as the sum of squares of non-linear real-valued functions. LM can be thought of as a combination of steepest descent and the Gauss-Newton method. When the current solution is far from the correct one, the algorithm behaves like a steepest descent method: slow, but guaranteed to converge. When the current solution is close to the correct solution, it becomes a Gauss-Newton method. Let f be an assumed functional relation which maps a parameter vector $\mathbf{p} \in R^n$ to an estimated measurement vector $\hat{\mathbf{x}} = f(\mathbf{p})$, $\hat{\mathbf{x}} \in R^n$. An initial parameter estimate $\mathbf{p}_0$ and a measured vector $\mathbf{x}$ are provided and it is desired to find the vector $\mathbf{p}^+$ that best satisfies the functional relation f, i.e. minimizes the squared distance $\epsilon^T\epsilon$ with $\epsilon = \mathbf{x} - \hat{\mathbf{x}}$. The basis of the LM algorithm is a linear approximation to f in the neighborhood of $\mathbf{p}$. For a small $||\delta_p||$, a Taylor series expansion leads to the approximation

$$f(\mathbf{p} + \delta_p) \simeq f(\mathbf{p}) + \mathbf{J}\delta_p \tag{2.4}$$

Where $\mathbb{J}$ is the Jacobian matrix $\frac{\partial f(\mathbf{p})}{\partial \mathbf{p}}$. Like all non-linear optimization methods, LM is iterative: initiated at the starting point $\mathbf{p}_0$, the method produces a series of vectors $\mathbf{p}_1, \mathbf{p}_2, ...$, that converges to a local minimizer $\mathbf{p}^+$ for f. Hence, at each step, it is required to find the δ_p that minimizes the quantity $||\mathbf{x} - f(\mathbf{p} + \delta_p)|| \simeq ||\mathbf{x} - f(\mathbf{p}) - \mathbf{J}\delta_p|| \simeq ||\epsilon - \mathbf{J}\delta_p)||$. The sought δ_p is thus the solution to a linear least-squares problem: the minimum is attained when $\mathbf{J}\delta_p - \epsilon$ is orthogonal to the column space of $\mathbf{J}$. This leads to $\mathbf{J}^T(\mathbf{J}\delta_p - \epsilon) = 0$ which yields δ_p as the solution of the so-called normal equations:

$$\mathbf{J}^T\mathbf{J}\delta_p = \mathbf{J}^T\epsilon \tag{2.5}$$

The matrix $\mathbf{J}^T\mathbf{J}$ in the left hand side of Eq. 2.5, is the approximate Hessian matrix, i.e. an approximation to the matrix of second order derivatives. The LM method actually solves a slight variation of Eq. 2.5, known as augmented normal equations:

$$\mathbf{N}\delta_p = \mathbf{J}^T\epsilon \tag{2.6}$$

where the off-diagonal element of $\mathbf{N}$ are identical to the corresponding elements of $\mathbf{J}^T\mathbf{J}$ and the diagonal elements are given by $\mathbf{N}_{ii} = \mu + [\mathbf{J}^T\mathbf{J}]_{ii}$ for some $\mu > 0$. The strategy of altering the diagonal elements of $\mathbf{J}^T\mathbf{J}$ is called damping and μ is referred as damping term. If the updated parameter vector $\mathbf{p}+\delta_p$ with δ_p computed from Eq. 2.6 leads to a reduction in the error ϵ, the update is accepted and the process repeats with a decreased damping term. Otherwise, the damping term is increased, the augmented normal equations are solved again and the process iterates until a value of δ_p that decreases error is found. The process of repeatedly solving Eq. 2.6 for different values of the damping term until an acceptable update to the parameter vector is found corresponds to one iteration of the LM algorithm. In LM, the damping term is adjusted at each iterations to assure a reduction in the error ϵ. If the damping is set to a large value, matrix $\mathbf{N}$ in Eq. 2.6 is nearly diagonal and the LM update step δp is near the steepest descent direction. Moreover the magnitude of δp is reduced in this case. Damping also handles situations where the Jacobian is rank deficient and $\mathbf{J}^T\mathbf{J}$ is therefore singular. In this way, LM can defensively navigate a region of the parameter space in which the model is highly non-linear. If the damping is small, the LM step approximates the exact quadratic step appropriate for a fully linear problem. LM is adaptive because it controls its own damping: it raises the damping if a step fails to reduce ϵ; otherwise it reduces the damping. In this way LM is capable to alternate between a slow descent approach when being far from the minimum and a fast convergence when being at the minimum's neighborhood. The LM algorithm terminates when at least one of the following conditions is met:

- The magnitude of the gradient $\epsilon^T\epsilon$, i.e. $\mathbf{J}^T\epsilon$ in the right hand side of Eq. 2.5, drops below a threshold ε_1
- The relative change in the magnitude of δp drops below a threshold ε_2
- The error $\epsilon^T\epsilon$ drops below a threshold ε_3
- A maximum number of iterations k_{max} is completed

2.3.2 Numerical Results

Results for the non-linear solution of the Zero Order ASSE Scheme are here presented as described in subsection 2.2.2.

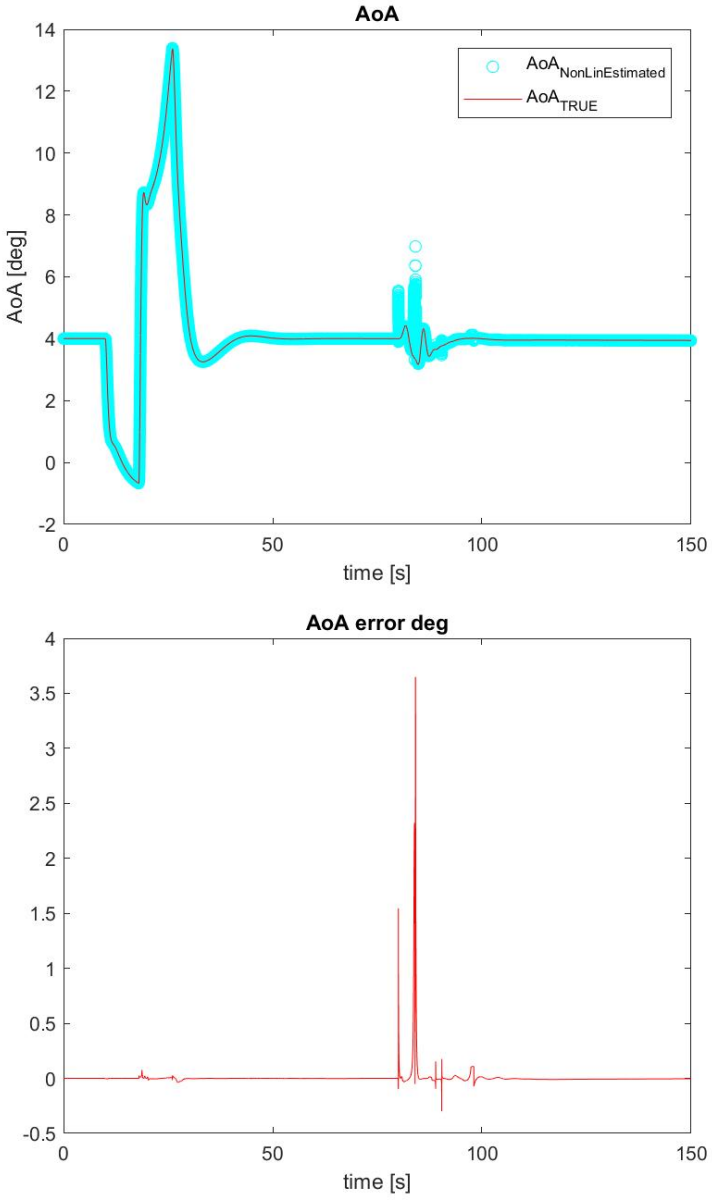


Figure 2.4: Comparison between the true and estimated value of AoA for the non-linear solution of the Zero Order ASSE Scheme

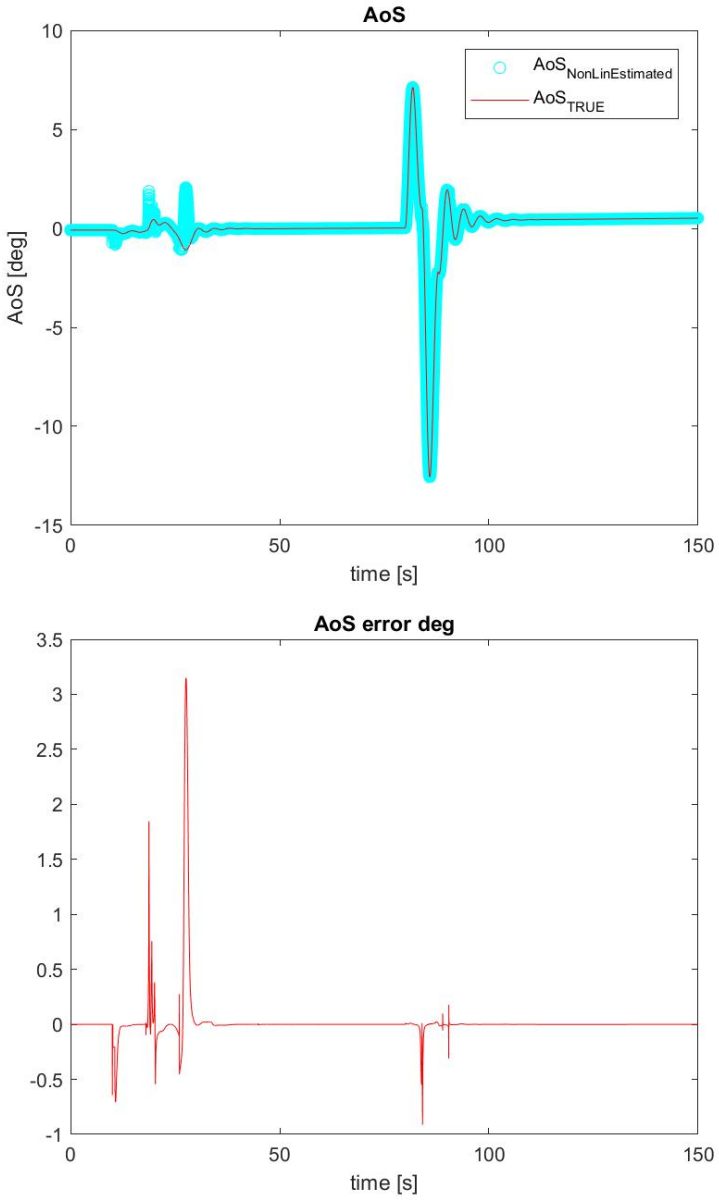


Figure 2.5: Comparison between the true and estimated value of AoS for the non-linear solution of the Zero Order ASSE Scheme

A summary of the results is presented in the following table:

Aerodynamic Angle	2σ error[deg]	Required 2σ error[deg]
α	0.0648	1.5
β	0.1182	2.5

Table 2.2: Non-Linear Zero Order ASSE Scheme 2σ errors

This results satisfy the prescribed requirements [15] and highlight the accuracy of the Zero Order ASSE Scheme proving it as a suitable method for the estimation of the aerodynamic angles. The following table gives further information on the errors committed during the computation of AoA and AoS:

Aerodynamic Angle	Mean error[deg]	Max error[deg]
α	-0.0078	3.6484
β	-0.0152	3.1475

Table 2.3: Non-Linear Zero Order ASSE Scheme mean and max errors

2.3.3 Sensitivity

Since the non-linear solution of the Zero Order ASSE Scheme with two equations for two subsequent time steps has proven to be a suitable method, a test with white noise corrupted data (which resembles real flight data) as been carried out in order to evaluate the method sensitivity. This topic is further discussed in more details in chapter 4 and the data is here corrupted as described in 4.25. Results are presented in the following figures.

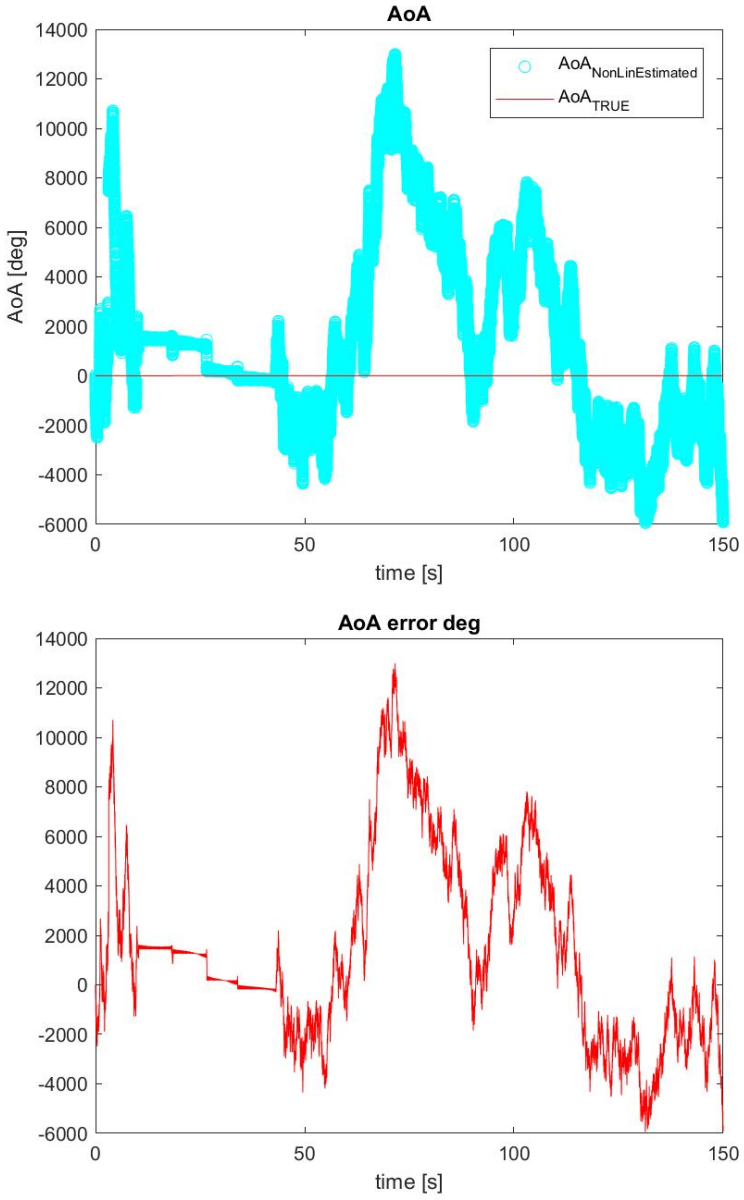


Figure 2.6: Comparison between the true and estimated value of AoA for the non-linear solution of the Zero Order ASSE Scheme with noisy flight data

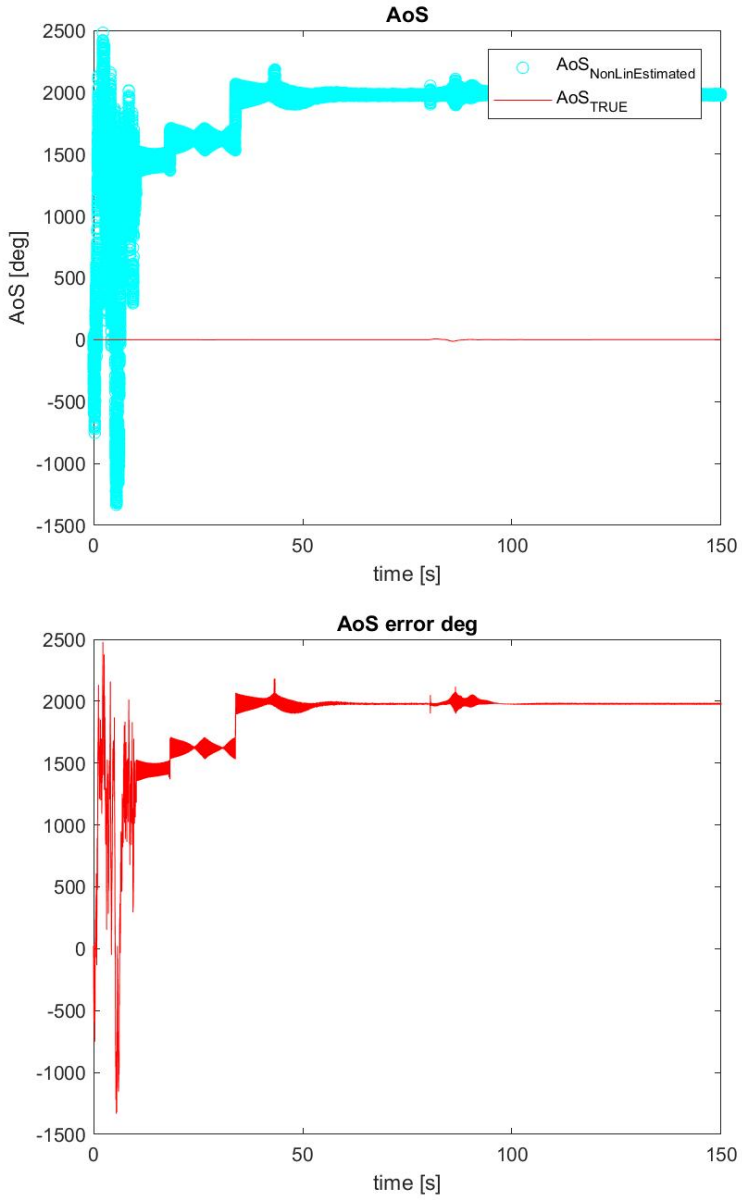


Figure 2.7: Comparison between the true and estimated value of AoS for the non-linear solution of the Zero Order ASSE Scheme with noisy flight data

As it is possible to see from these results, the non-linear solver with two equations is not suitable in the presence of noisy flight data with a 2σ error of magnitude 10^3. Because of this, no other maneuvers with noisy data are tested with the non-linear MatLab routine.

Chapter 3

Neural Network Theoretical Background

A lot of research has been conducted over virtual sensors which exploit soft computing techniques based on neural network predictions. The main advantage of virtual sensors is replacing traditional probes and vanes which cover part of the front surface of the fuselage and can cause, especially in small UAVs, interference with opto-electronic sensors which best placing location is the same of the traditional sensors. Furthermore, virtual sensors have been proven as accurate as traditional probes in measuring the aerodynamic angles for UAV applications [3]. This extended research has proven the importance and the power of neural networks in the process of estimation of the aerodynamic angles, and they are therefore further analyzed in this work. The aim of this part of the work is to evaluate the feasibility of using a neural network to solve a non-linear system of equations. The main goal is to check the level of accuracy that can be obtained by exploiting a neural network, which characteristics will be further discussed, to estimate the aerodynamic angles from the Zero-Order ASSE Scheme. For this purpose, a theoretical background over neural networks is given in the next sections.

3.1 Neural Network Characteristics

To start with, a brief description of what is a neural network is given. Information over the type of network used for this work will be later defined.

3.1.1 Definition of a Neural Network

According to the definition given by Simon Haykin [2], a neural network, sometimes also called neurocomputer, is a massively parallel distributed processor made up of simple processing units, called neurons, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in

two aspects:

1. Knowledge is acquired by the network from it's environment through a learning process.

2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

The power of neurocomputers is the fact that, once opportunely trained with a certain procedure, called the learning algorithm, they are able to generalize. Generalization is the ability to produce reasonable outputs for inputs not encountered during training. The learning ability and the parallel processing allows them to solve complex problems once they are decomposed in simple tasks, each one assigned to a specific network. More general features of neural network showed in the book of Hayakin [2] are not described here since they are beyond the scope of this work.

3.1.2 Neuron Model

The schematic representation of a neuron is depicted in Figure 3.1.
A neuron has the following characteristics:

1. Connecting Links or Synapses. Each input of the input layer of signals has it's specific strength related to that neuron. In fact the j^{th} signal is connected via synapses with the $\mathbf{k}^{th}$ neuron and it is multiplied by a factor W_{kj}, defined Weight of the synapse. The subscripts of the synaptic weight indicate that the particular value of the j^{th} weight for the j^{th} input, is related to the neuron k.

2. Adder. The adder of the neuron is a linear combiner which sums all the weighted inputs.

3. Activation Function. The output of a neuron is limited with a particular function, which are further discussed. The amplitude of the neuron output is usually limited in the interval [0,1] or [−1,1].

4. Bias. The term b_k of figure 3.1 is denoted as bias, and it has the effect of increasing or lowering the net input of the activation function.

In mathematical terms a neuron can be described with the following equations:

$$v_k = \sum_{j=0}^{m} W_{kj} x_j \tag{3.1}$$

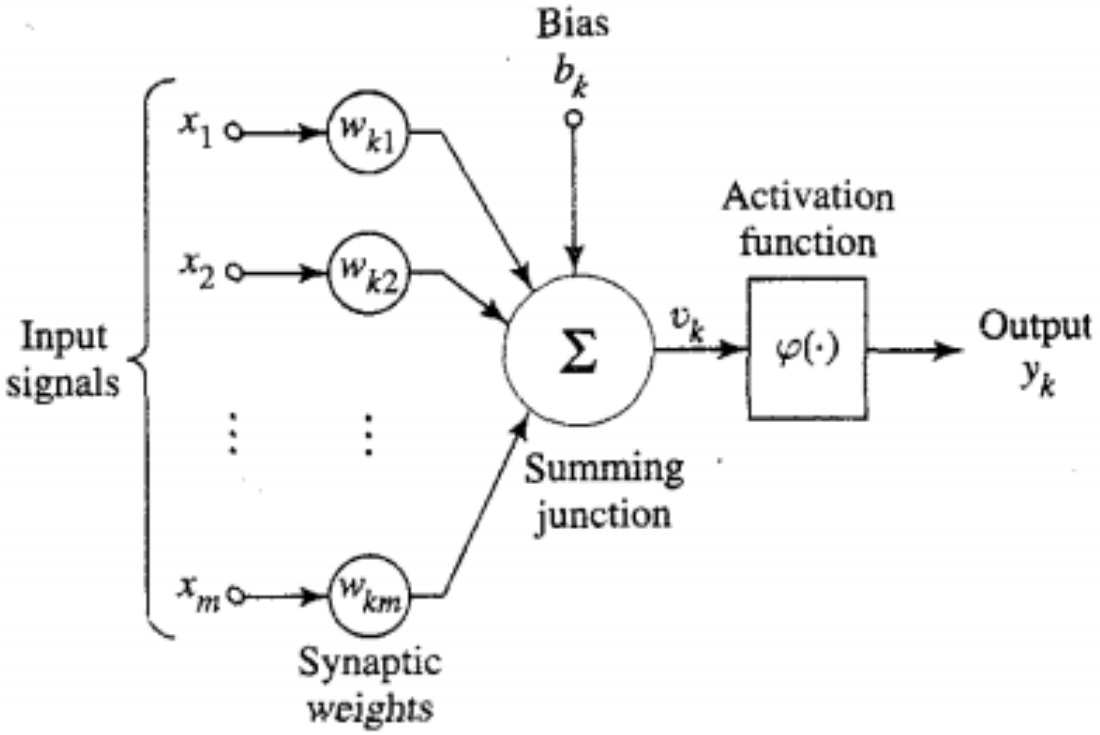


Figure 3.1: [2] Non-linear model of a neuron

where x_j is the j^{th} input, W_{kj} is the weight of the j^{th} input related to neuron k, and in the particular case of $j = 0$, the input $x_0 = 1$ and the weight $W_{k0} = b_k$, which is a compact way of including the term b_k in the equation.
Then, the activation function must be applied to produce the output of the k_{th} neuron:

$$y_k = \varphi(v_k) \tag{3.2}$$

Equations 3.1 and 3.2 represents the basic operations carried out from a neuron.

3.1.3 Activation Functions

Activation functions have the role of limiting the output of a neuron and to give the network the ability to generalize and extrapolate beyond the limit if the training patterns. The most common activation functions are:

1. Threshold function. Also known as Heaviside function, it gives an output value of 1 if the induced field of the neuron is non-negative, and it returns the value of 0 otherwise. The mathematical representation is:

$$\varphi(v_k) = \begin{cases} 1, & v_K \geqslant 0 \\ 0, & v_K < 0 \end{cases} \tag{3.3}$$

   where v_k is defined in Eq. 3.1.

2. Piecewise-linear function. The mathematical representation is the following:

$$\varphi(v_k) = \begin{cases} 1, & v_K \geqslant \frac{1}{2} \\ v_k, & -\frac{1}{2} < v_k < \frac{1}{2} \\ 0, & v_K \leqslant -\frac{1}{2} \end{cases} \tag{3.4}$$

   this form of an activation function can be viewed as an approximation to a non linear amplifier. It works as a linear-amplifier if the linear region of operation is maintained without running into saturation and it reduces to a threshold function if the amplification factor in the linear region is made infinitely large.

3. Sigmoid function. It is the most common used activation function in neurocomputing. It's a strictly increasing function and it's the perfect balance between a linear and non-linear behaviour. The mathematical expression is:

$$\varphi(v_k) = \frac{1}{1 + exp(-av_k)} \tag{3.5}$$

   the term a represents the slope parameters. The Sigmoid function assumes a continuous range of values between 0 and 1 and it behaves more and more like the Heaviside function as a approaches to infinity. A really important feature of the Sigmoid function is the fact that it is differentiable, which is also an important characteristic for neural networks.

All the activation functions discussed so far, range in the interval [0,1]. Sometimes it's preferable to have a function ranging in the interval $[-1,1]$. In this case, the threshold function becomes:

$$\begin{cases} 1, & v_K > 0 \\ 0, & v_K = 0 \\ -1, & v_K < 0 \end{cases} \tag{3.6}$$

which is also known as Signum function. The corresponding sigmoidal function is the Hyperbolic Tangent function:

$$\varphi(v_k) = \tanh(v_k) \tag{3.7}$$

which is a sigmoidal type function that can assume negative values. The following figure represents the main activation functions described above.

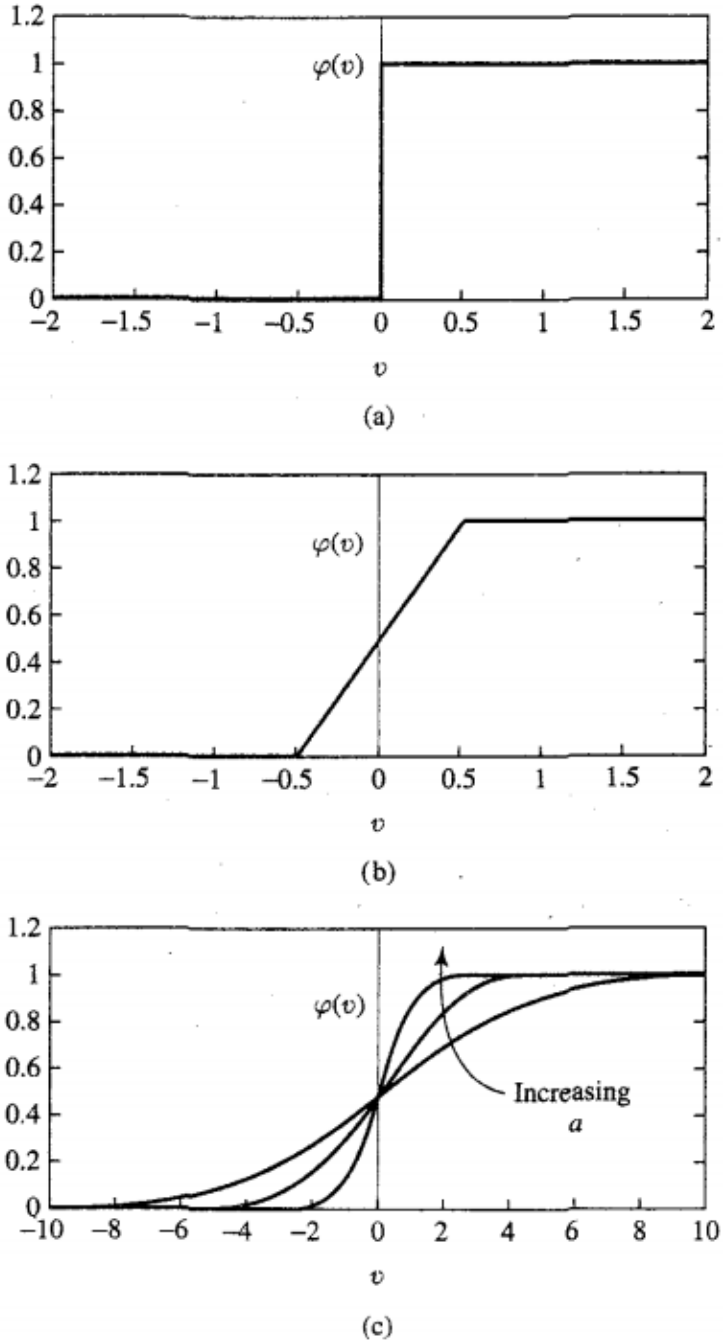


Figure 3.2: [2] (a) Threshold function, (b) Piecewise-linear function, (c) Sigmoid function for varying slope parameter a.

Another type of activation functions are the Radial-Basis-Functions (RBF), the most common in this new group of function is the Gaussian function:

$$\varphi(r_k) = \exp\left(\frac{r_k^2}{2\sigma^2}\right) \tag{3.8}$$

where $r_k = ||x - c_k||$, with c_k being the center of the k^{th} radial basis function of the network.

3.1.4 Network Architectures

In this subsection the neural networks architectures are presented. A group of neurons organized in a determined pattern becomes a neural network. Neurons are usually grouped in subsequent layers with the last being the output layer. The most common architectures are:

- Feed Forward Neural Networks
- Recurrent Networks

Feed forward neural networks are characterized by an absence of a feedback loop, and they can be divided in two categories:

1. Single Layer Perceptron (SLP). These networks feature an input layer of neurons which has only the purpose to collect the input data, but it doesn't perform any computation, and an output layer of computational neurons. It is also sometimes referred as acyclic network.

2. Multi Layer Perceptron (MLP). The difference with the former architecture is the fact that more hidden layers of computational neurons are present. The hidden units act as a useful intermediary between the external inputs and the output. The extra set of synaptic connections allows to extract higher order statistics which is very useful in the case of a great number of inputs. In this architecture, each layer of neurons is connected to the subsequent one, with the output of one layer being the input of the next layer. In addition, each neuron of a hidden layer can be connected with all the neurons of the next layer, in which case the networks is defined fully connected, or it can be connected with only a fraction of the neurons of the subsequent layer, in which case the network is defined partially connected. The input vector is applied to sensory nodes of input layers, and its effects propagates layer by layer to the output node. Here a set of output are calculated and compared with the

actual, or desired, target. In mathematical terms, for a two layer feed forward network, in which the subscripts i, j, k and l represent respectively the input layer, the first and the second hidden layer and the output layer, the output y_l is given by:

$$y_l = f_l(x_1, ...x_n) = \sum_{k=1}^{n} \sum_{j=1}^{n} \sum_{i=1}^{n} W_i x_i + b \tag{3.9}$$

a MLP network can be essentially interpreted as a non-linear mapping between the input and the target values. This architecture will be further used for the purpose of this work.

Beside the feed forward architectures, another type of networks are the Recurrent Networks. As described in the work of Medsker [17] recurrent neural networks have been an important focus of research and development during the 1990's. They are designed to learn sequential or time varying patterns. A recurrent net is a neural network with feedback (closed loop) connections. For example, a recurrent neural network may consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of all other neurons. Recurrent architectures with hidden neurons also exist. Recurrent neural network techniques have been applied to a wide variety of problems. Simple partially recurrent neural networks were introduced in the late 1980's by several researchers including Rumelhart, Hinton, and Williams to learn strings of characters. Many other applications have addressed problems involving dynamical systems with time sequences of events. Recurrent neural networks are being used to track water quality and minimize the additives needed for filtering water. Furthermore, the time sequences of musical notes have been studied with recurrent neural networks. The idea behind these type of networks is to consider the input data not individually but as part of a context in order to produce a sequence of inputs and making predictions considering the relationship among the inputs. Mathematically speaking, the output of the network at a certain time t is the result of a self-training of the network based on a sequence of tests at the previous time instants. To get the output at time t, not only the input vector at time t, x_t is considered, but also the input vector x_{t-1} and the output in $t-1$. In order to achieve this, the feedback loops involve the use of particular branches composed of unit delay elements (denoted z^{-1}) which result in a non-linear dynamical behaviour, assuming that the neural network contains non-linear units. The network is then able to change its behaviour in a dynamic way, based on the context in which is operating. In the next figures the basic architectures described above are presented.

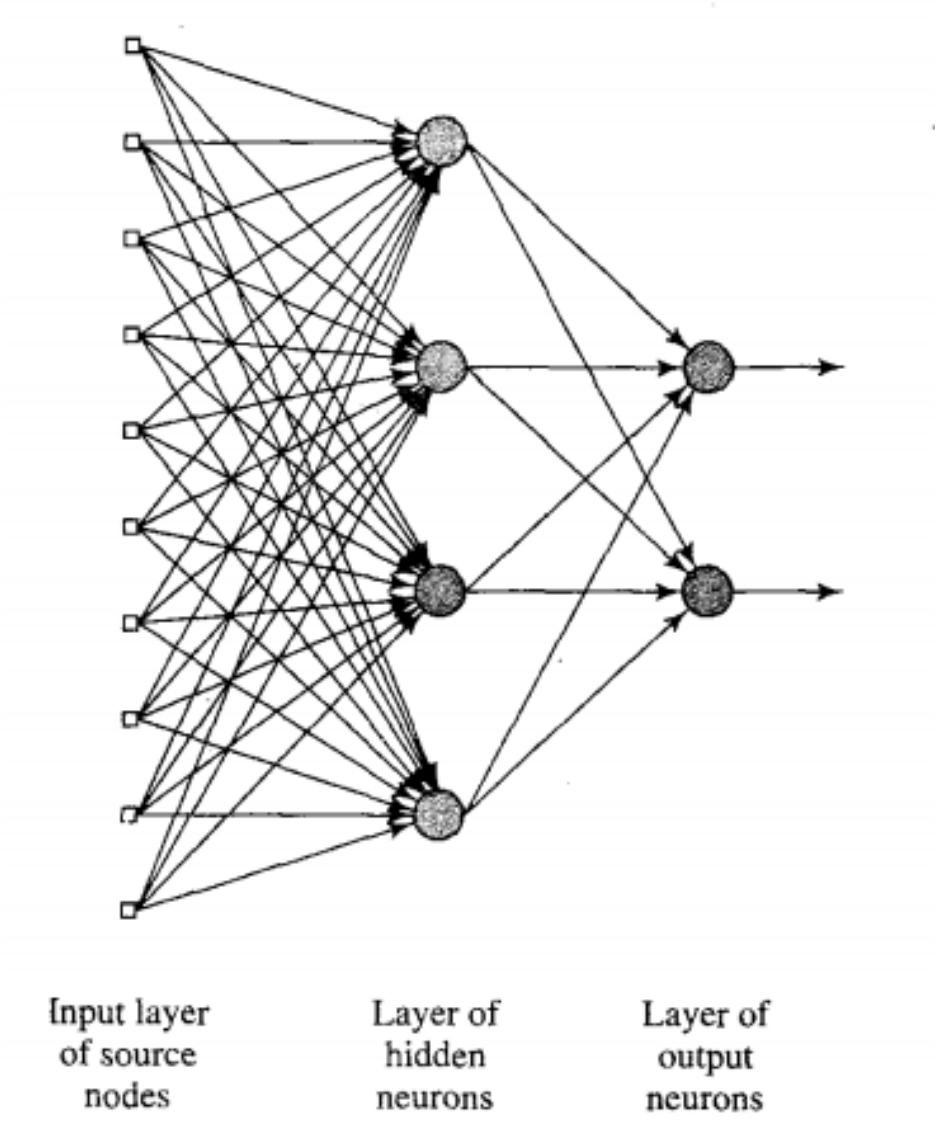


Figure 3.3: [2] Fully connect feed forward or acyclic network with one hidden layer and one output layer

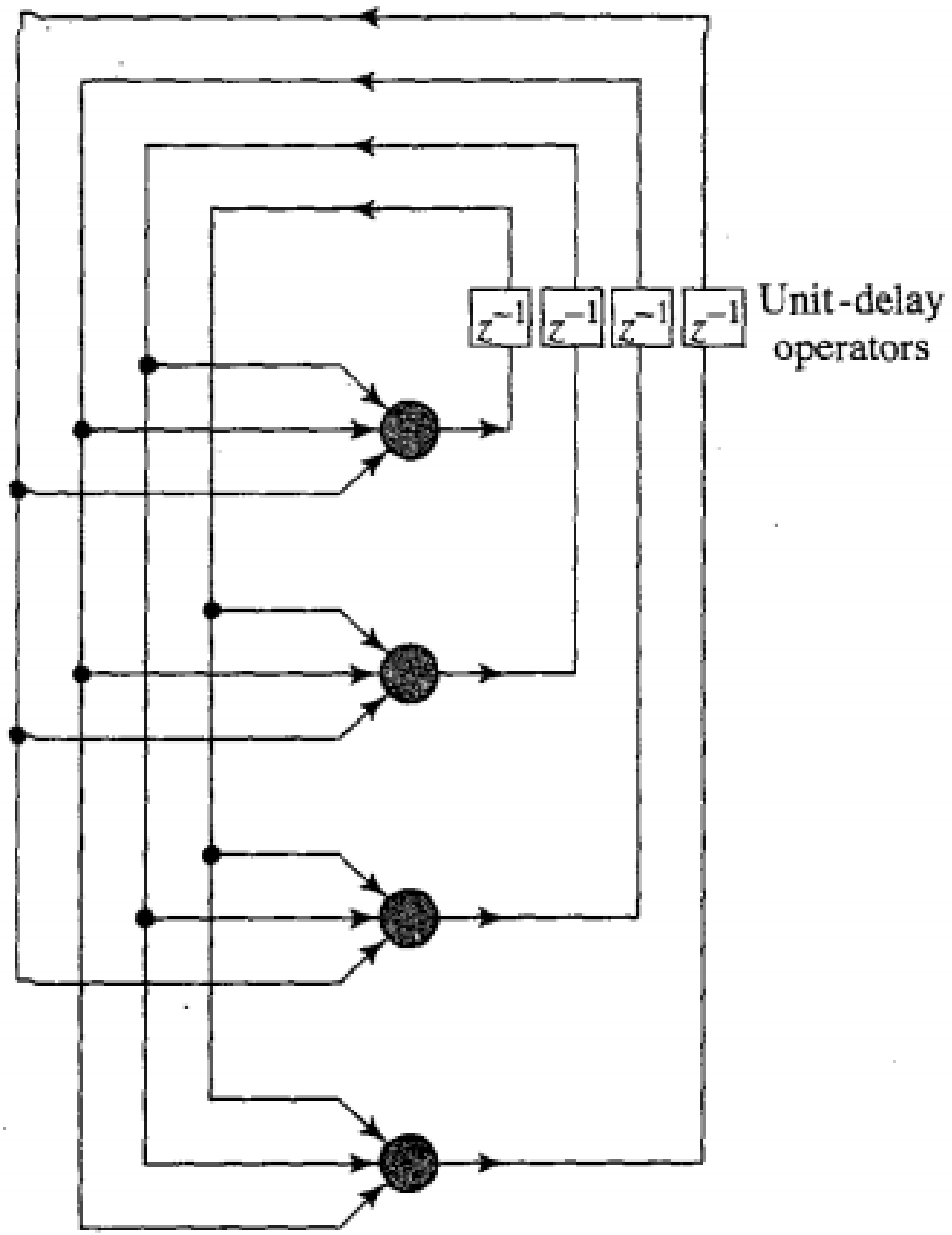


Figure 3.4: [2] Recurrent network with no self-feedback loops and no hidden neurons

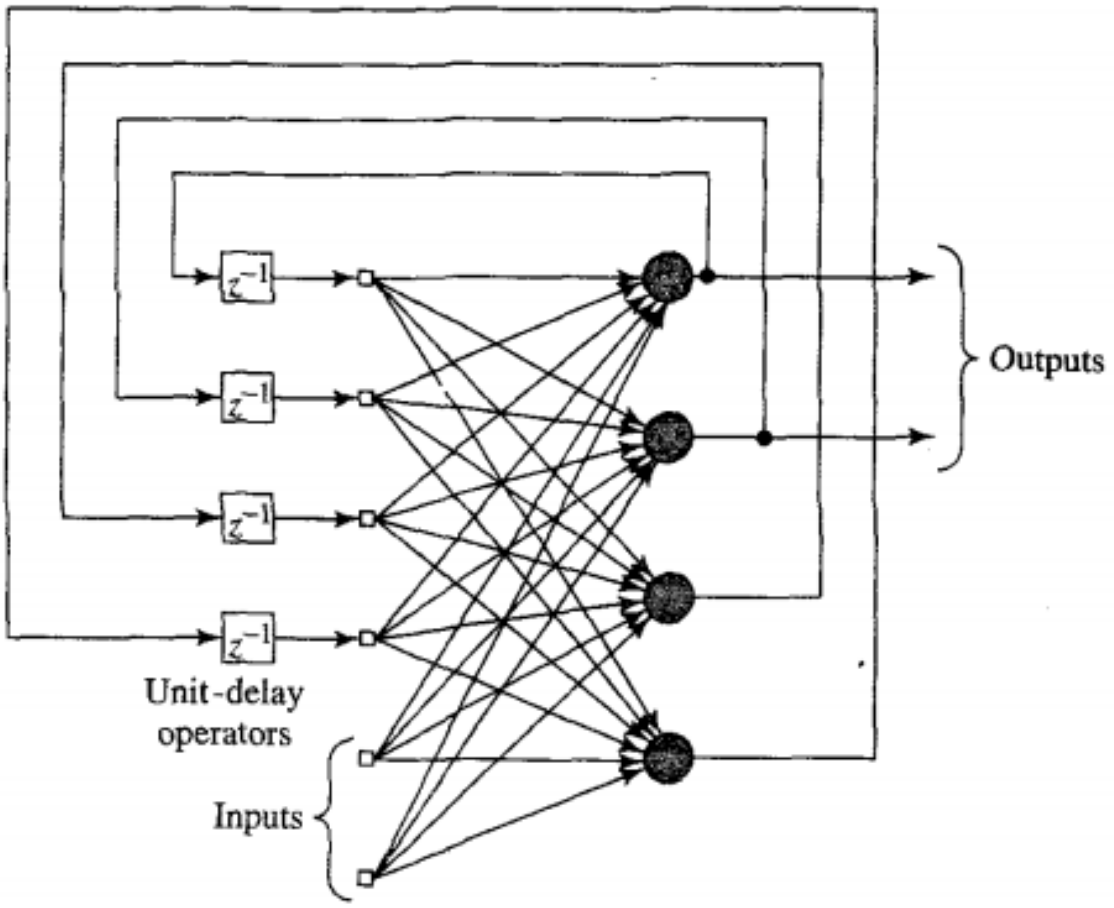


Figure 3.5: [2] Recurrent network with hidden neurons

3.2 Neural Network Learning Process

The learning process of a neurocomputer can be defined as [2] a procedure by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded and the different ways of learning are determined by the manner in which the parameters changes take place. According to this definition, there are three main steps involved in the act of learning:

1. Stimulation, which comes from an external environment.
2. Parameters Changes, which occur as a response to the external stimulation.
3. New Response of the network to a stimulus because of the changes occurred in its parameters.

3.2.1 Error-Correction Learning

One of the most used learning processes, which is also the one used in this work, is the error correction method. A scheme of how this process work is depicted in Fig. 3.6.

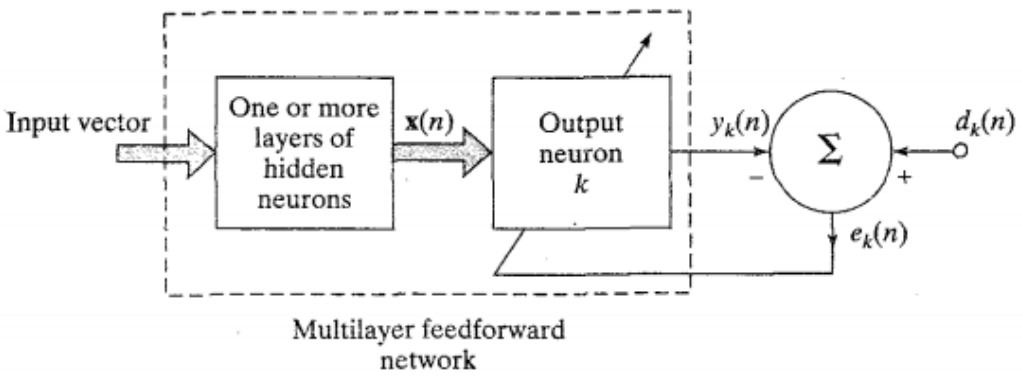


Figure 3.6: [2] Illustration of the error correction learning

Considering the simple case of a neuron k driven by a signal vector $\mathbf{x}(n)$, produced by one or more layers of hidden neurons which are themselves driven by an input vector applied to the input layer of the neural network. The argument n denotes discrete time, or more precisely, the time step of an iterative process involved in adjusting the synaptic weights of neuron k. The output signal of neuron k is denoted by $y_k(n)$. This output signal, representing the only output of the neural

network, is compared to a desired response or target output, denoted by $d_k(n)$. As a consequence, an error signal $e_k(n)$ is produced.

$$e_k(n) = d_k(n) - y_k(n) \tag{3.10}$$

the error signal $e_k(n)$ actuates as a control mechanism which purpose is to apply a sequence of corrective adjustments to the synaptic weights of neuron k in order to make the output $y_k(n)$ come closer to the desired response $d_k(n)$ in a step by step manner. The function to be minimized in order to get a better output is the cost function $\varepsilon(n)$ defined in terms of $e_k(n)$:

$$\varepsilon(n) = \frac{1}{2}e_k^2(n) = \frac{1}{2}\left(d_k(n) - y_k(n)\right)^2 \tag{3.11}$$

In the simple case of a linear model with m neurons:

$$y_k(x, n) = \sum_{k=1}^{m} w_k \varphi_k(x, n) \tag{3.12}$$

the cost function can be expressed as:

$$\varepsilon(n) = \frac{1}{2}\sum_{k=1}^{m} e_k^2(n) = \frac{1}{2}\sum_{k=1}^{m}\left(d_k(n) - y_k(n)\right)^2 \tag{3.13}$$

which represent the instantaneous value of the error energy. In both cases the step by step process is repeated until the system reaches a steady state (the weights are essentially stabilized). At this point, the learning process is terminated. In more complex situations there can be extra regulations terms in Eq. 3.13. In the case of batch learning or off-line learning, the relative errors are evaluated at different time instants and the cost function becomes:

$$\varepsilon = \frac{1}{2p}\sum_{n=1}^{p}\sum_{k=1}^{m} e_k^2(n) \tag{3.14}$$

where p is the total number of time instants. In addition, there are other different kinds of cost function that is possible to use:

1. Mean Squared Error (MSE)

$$\varepsilon = \frac{1}{2p}\sum_{n=1}^{p} e^2(n) \tag{3.15}$$

2. Root Mean Squared Error (RMSE)

$$\varepsilon = \sqrt{\frac{1}{2p}\sum_{n=1}^{p} e^2(n)} \tag{3.16}$$

3. Mean Absolute Error (MAE)

$$\varepsilon = \frac{1}{2p}\sum_{n=1}^{p} |e(n)| \tag{3.17}$$

Synaptic weights are updated in each iteration in order to minimize the cost function.

$$\Delta w_{kj}(n) = -\eta \frac{\partial \varepsilon}{\partial w_{kj}} \tag{3.18}$$

The instant difference in the value of the synaptic weight depends on the learning algorithm used. η is a constant, and $\Delta w_{kj}(n)$ is the variation of the synaptic weight of the j^{th} input at the time instant n related to the neuron k. The weight is then updated in the next instant as follows:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \tag{3.19}$$

The process of updating the weights, which is the training, stops when the network reaches a steady state in which the cost function does not change significantly anymore because a minimum has been reached. Such a method is also know as the Delta Rule.

3.2.2 Supervised And Unsupervised Learning

The learning process can occur with knowledge of the correct outputs or without it. This splits the learning process in two categories: supervised and unsupervised learning.

1. **Supervised Learning**. In Supervised Learning, the network is fed with a correct output given a determined set of inputs. The network has then to learn the relationship between the correct input-output couples. It is like the network is guided and controlled by a teacher which supervises its progresses. The environment is, however, unknown to the neural network of interest. Suppose now that the teacher and the neural network are both exposed to a training vector (i.e., example) drawn from the environment. By virtue of built-in knowledge, the teacher is able to provide the neural network with a desired response for that training vector. Indeed, the desired response represents the optimum action to be performed by the neural network. The network parameters are adjusted under the combined influence of the training vector and the error signal. The error signal is defined as the difference between the desired response and the actual response of the network. This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network emulate the teacher; the emulation is presumed to be optimum in some statistical sense. In this way knowledge of the environment available to the teacher is transferred to the neural network through training as fully as possible. When this condition is reached, we may then let the neural network deal with the environment completely by itself. As a performance measure for the system, the mean-square error or the sum of squared errors over the training sample, defined as a function of the free parameters of the system, can be considered. This function may be visualized as a multidimensional error-performance surface or simply error surface, with the free parameters as coordinates. The true error surface is averaged over all possible input—output examples. Any given operation of the system under the teacher's supervision is represented as a point on the error surface. For the system to improve performance over time and therefore learn from the teacher, the operating point has to move down successively toward a minimum point of the error surface; the minimum point may be a local minimum or a global minimum. A supervised learning system is able to do this with the useful information it has about the gradient of the error surface corresponding to the current behavior of the system. The gradient of an error surface at any point is a vector that points in the direction of steepest descent. Nevertheless, given an algorithm designed to minimize the cost function, an adequate set of input—output examples, and enough time permitted to do the training, a supervised learning system is usually able to perform such tasks as pattern classification and function approximation.

2. **Unsupervised Learning**. In unsupervised or self-organized learning there is no external teacher or critic to oversee the learning process. Rather, provision is made for a task-independent measure of the quality of the representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically (Becker, 1991). To perform unsupervised learning we may use a competitive learning rule. For example, we may use a neural network that consists of two layers, an input layer and a competitive layer. The input layer receives the available data while the competitive layer consists of neurons that compete with each other (in accordance with a learning rule) for the "opportunity" to respond to features contained in the input data. In its simplest form, the network operates in accordance with a "winner-takes-all" strategy. In such a strategy the neuron with the greatest total input "wins" the competition and turns on; all the other neurons then switch off.

3.2.3 Online And Batch Learning

Another classification of neural networks is based on the way it learns. If the network continuously learns at each time step, then it goes under the definition of an online training network. Online training is dynamic an changeable and it adapts to the examples given at each time step. Such a method is of a great advantage when it comes to computational cost since it is not necessary to memorize all the training patterns but once defined a single pattern, it will be later useless and hence discarded. If, on the contrary, the network is provided with all the examples needed in order to be trained in a single instant, it is defined Batch learning, or off-line learning. This method is much more expansive in terms of computational cost since more memory must be allocated for all the training patterns. Furthermore, once the training is concluded, the network is unchangeable, which can be an advantage since it avoids non-predictable behaviours.

3.2.4 MLP Learning Algorithms

In this subsection the main learning algorithms for multi layer perceptrons are discussed. All this methods are based on the evaluation of the cost function gradient in order to update the weights to their new values. This kind of learning is hence supervised, since to evaluate the cost function, examples of correct input-output couples are needed. Considering the output of a generic neuron j, which inputs are

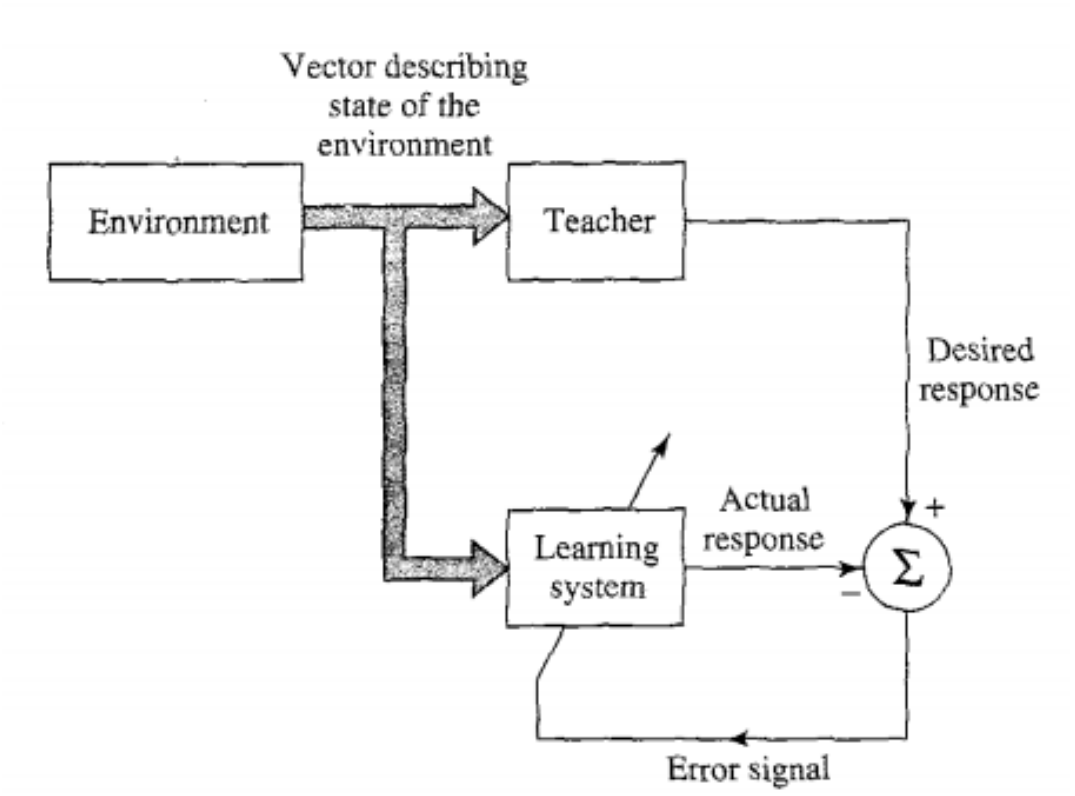


Figure 3.7: [2] Block Diagram of Supervised Learning

the outputs of other neurons in the context of a MLP, it is then possible to write:

$$f_j(v_j) = y_j = \varphi_j \left(\sum_{i=1}^{m} w_{ji} y_i + b \right) \tag{3.20}$$

where y_i refers to the outputs of the previous neurons. The list of algorithms for MLP is the following:

- Backpropagation algorithm.
- Descent or gradient based methods, such as:
  1. Steepest Descent method
  2. Newton's Method

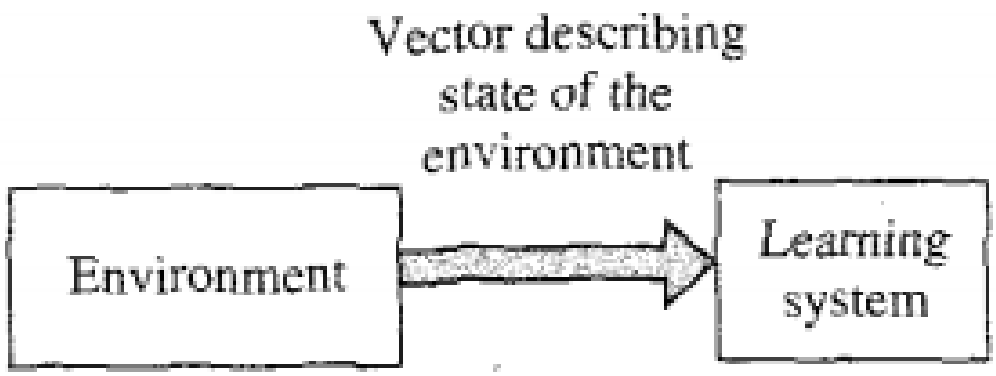


Figure 3.8: [2] Block Diagram of Unsupervised Learning

3. Levenberg-Marquardt Algorithm

The Back Propagation Algorithm (BP) applies a correction, Δw_{ji}, to the synaptic weight, w_{ji}, proportional to the gradient $\frac{\partial \varepsilon(t)}{\partial w_{ji}}$. According to the chain rule it may be written as:

$$\frac{\partial \varepsilon(t)}{\partial w_{ji}(t)} = \frac{\partial \varepsilon(t)}{\partial e_j(t)} \frac{\partial e_j(t)}{\partial y_j(t)} \frac{\partial y_j(t)}{\partial v_j(t)} \frac{\partial v_j(t)}{\partial w_{ji}(t)} \tag{3.21}$$

Differentiating Eq. (3.20) with respect to $v_j(t)$ and $w_{ji}(t)$, the following equations can be obtained

$$\frac{\partial y_j(t)}{\partial v_j(t)} = f_j'(v_j(t)) \tag{3.22}$$

and

$$\frac{\partial v_j(t)}{\partial w_{ji}} = y_i(t) \tag{3.23}$$

Differentiating both sides of Eq.(3.10) with respect to $y_j(t)$ and Eq.(3.13) with respect to $e_j(t)$, we get respectively

$$\frac{\partial e_j(t)}{\partial y_j(t)} = -1 \tag{3.24}$$

and

$$\frac{\partial \varepsilon(t)}{\partial e_j(t)} = e_j(t) \tag{3.25}$$

Therefore, the use of equations (3.22) to (3.25) in (3.21), yields

$$\frac{\partial \varepsilon(t)}{\partial w_{ji}(t)} = -e_j(t) f_j'\left(v_j(t)\right) y_i(t) = -\delta_j(t) y_i(t) \tag{3.26}$$

where $\delta_j(t)$ is commonly defined as the local gradient. The correction to the synaptic weight $w_{ji}(t)$ is established using the delta rule

$$\Delta w_{ji}(t) = -\eta \frac{\partial \varepsilon(t)}{\partial w_{ji}(t)} = \eta \delta_j(t) y_i(t) \tag{3.27}$$

where η is defined as the learning rate. Therefore, the updated weight is

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t) = w_{ji}(t) - \eta \frac{\partial \varepsilon(t)}{\partial w_{ji}(t)} = w_{ji}(t) + \eta \delta_j(t) y_i(t) \tag{3.28}$$

As shown by several authors there is not an optimum learning rate but according to the particular problem there is a η that assures fast and stable convergence. There are some algorithm provided with a variable learning algorithm rate according to the local gradient, or other parameters, to speed up the convergence. To summarise, the local gradient is equal to:

$$\begin{cases} \delta_j(t) = e_j(t) f_j'(v_j(t)) & j = \textit{output neuron} \\ \delta_j(t) = f_j'(v_j(t)) \sum_k \delta_k(t) w_{kj}(t) & j = \textit{hidden neuron} \end{cases} \tag{3.29}$$

In the second expression of 3.29 the index k refers to an output neuron, while index j refers to the hidden neuron. The Back Propagation algorithm described above adapts to an online learning type of network. In the case of Batch Learning, the procedure is the same, but the cost function assumes the expression of 3.14.

The Descent methods or gradient methods aim at finding the minimum of the cost function by solving the equation $\nabla \varepsilon(t) = 0$ and then updating the synaptic weights. The following discussion will refer to on-line training since it is very similar for batch training. The gradient of the cost function with respect to the synaptic weights is defined as:

$$\underline{g}(t) = \frac{\partial \varepsilon(t)}{\partial \underline{w}(t)} = \left[\frac{\partial \varepsilon(t)}{\partial w_1(t)}, \ldots, \frac{\partial \varepsilon(t)}{\partial w_n(t)} \right]^T \tag{3.30}$$

and hence the change in the synaptic weights can be written as:

$$\underline{w}(t+1) = \underline{w}(t) - \eta \underline{\underline{G}} \underline{g}(t) = \underline{w}(t) - \eta \frac{\partial \varepsilon(t)}{\partial \underline{w}(t)} \tag{3.31}$$

where G is a positive defined matrix. As already anticipated, there are three descent methods:

1. **Steepest Descent Method**. In this method, the positive defined matrix G is equal to the identity matrix I. Eq. 3.31 then becomes:

$$\underline{w}(t+1) = \underline{w}(t) - \eta \underline{g}(t) \tag{3.32}$$

this methods follows the steepest descent downhill, represented in Eq. 3.32 by the term $-g(t)$, and hence is really sensible to the initial conditions and can easily fall in a local minima without globally minimizing the function $\varepsilon(t)$.

2. **Newton Method**. This method uses the second order derivatives of the cost function if they are available. Considering the second order approximation of the Taylor expansion for the error energy:

$$\begin{aligned} \varepsilon(t+1) \approx & \\ & \varepsilon(t) + \underline{g}^T [\ \underline{w}(t+1) - \underline{w}(t)\ ] \\ & + \frac{1}{2} [\ \underline{w}(t+1) - \underline{w}(t)\ ]^T \underline{\underline{H}} [\ \underline{w}(t+1) - \underline{w}(t)\ ] \end{aligned} \tag{3.33}$$

higher order terms are omitted with the hypothesis that $[\underline{w}(t+1) - \underline{w}(t)]$ is small enough and the term $\underline{\underline{H}}(t)$ is the Hessian matrix:

$$\underline{\underline{H}}(t) = \begin{bmatrix} \dfrac{\partial^2 \varepsilon(t)}{\partial^2 w_1(t)} & \cdots & \dfrac{\partial^2 \varepsilon(t)}{\partial w_1(t) \partial w_n(t)} \\ \cdots & \cdots & \cdots \\ \dfrac{\partial^2 \varepsilon(t)}{\partial w_n(t) \partial w_1(t)} & \cdots & \dfrac{\partial^2 \varepsilon(t)}{\partial^2 w_n(t)} \end{bmatrix} \tag{3.34}$$

differentiating and considering equal to zero Eq. 3.33 the following is obtained:

$$0 = \underline{g}(t) + \underline{\underline{H}}(t) [\ \underline{w}_{MIN} - \underline{w}(t)\ ] \tag{3.35}$$

If the inverse matrix of $\underline{\underline{\mathrm{H}}}$ exists, the Newton's method is obtained

$$\underline{w}_{MIN} = \underline{w}(t) - \underline{\underline{H}}(t)^{-1} \underline{g}(t) \tag{3.36}$$

which in the case of a non quadratic function, it only represents one step towards the minimum:

$$\underline{w}(t+1) = \underline{w}(t) - \underline{\underline{H}}(t)^{-1}\underline{g}(t) \tag{3.37}$$

which is the general expression (3.31), where $\underline{\underline{G}} = -\underline{\underline{H}}^{-1}$ and $\eta = 1$.

3. **Levenberg-Marquardt Algorithm**. The LM algorithm has already been described in detail in subsection 2.3.1. Each step is described by the equation:

$$\underline{w}(t+1) = \underline{w}(t) - \left(\underline{\underline{H}}(t) + \mu\underline{\underline{I}}\right)^{-1}\underline{g}(t) \tag{3.38}$$

The LM method transits smoothly between Newton's method, as μ approaches 0, and the steepest descent method as μ grows infinitely. The steepest descent method is utilized at a large distance from the minimum of the considered function, to provide steady and convergent progress toward the solution. As the solution approaches the minimum, μ is adaptively decreased, the Levenberg-Marquardt method approaches Newton's method, and the solution usually converges rapidly to the local minima.

3.2.5 Neural Network Validation: Overfitting And Local Minima

Once the network has been trained, the validation phase starts. Input data, different from the ones given during training but within the training boundaries, are fed to the network in order to check if it is able to respond with a correct output. The training and validation loop process is represented in Fig. 3.9.
If the input-output patterns of the network during validation are reasonably acceptable for input test data within the training boundaries, then it is possible to say that the the network is able to generalize. However, when too much input-output examples are given for training, the network might end up memorizing the data instead of learning from it, thus loosing the ability to generalize. This problem is known as **overfitting** or **overtraining**. Since this problem is nothing but a deficiency stored inside the synaptic weights, there are two techniques to look for the most suitable number of neurons which allow to keep the network as general as possible. This techniques are:

1. Growing. The network is first built with an underestimated number of neurons, which is then gradually increased until improvements in the extrapolation and generalization ability of the network can be observed.

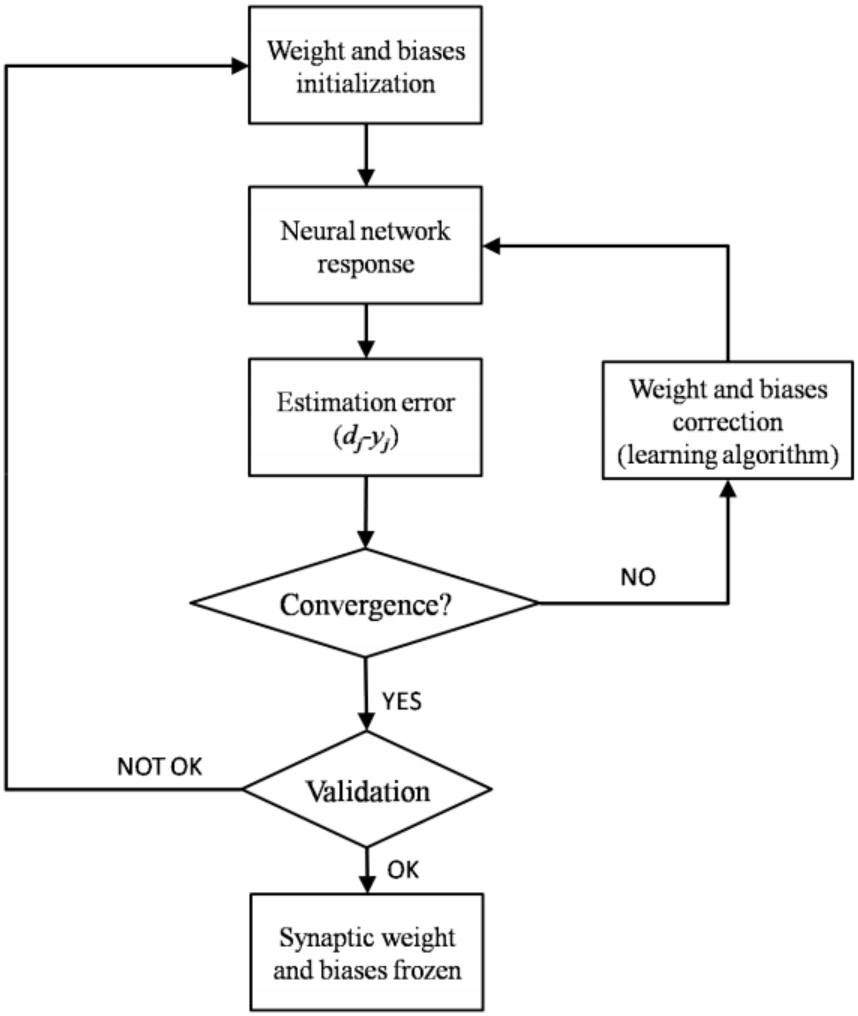


Figure 3.9: [3] Flow-chart of the neural network training and validation process

2. Pruning. On the contrary, the Pruning starts with an overestimated number of neurons which is then gradually decreased according to a regularization term which deletes the neurons that are less activated.

As far as the **Local Minima** problem is concerned, it is an issue related to the initial conditions of the network. Considering the simple example of figure 3.10 starting from the first weight attempt, w_{init}, any deterministic training algorithms, such as BP or LM algorithms, will always find the w_{min1} as the best weight to minimize the error, or optimize the neural network performance. This example can be translated into multi-dimensional space and the error profile becomes a complex hyper surface which depends on the NN free parameters, synaptic weights and

biases. The neural network training will often end up in local minima points of the hyper surface. To solve this problem, different algorithms have been developed but are not commonly used because of their complexity and computational costs. A solution to this problem is to carry on more parallel training of the network with different initial conditions and then choose the one which commits the smallest error.

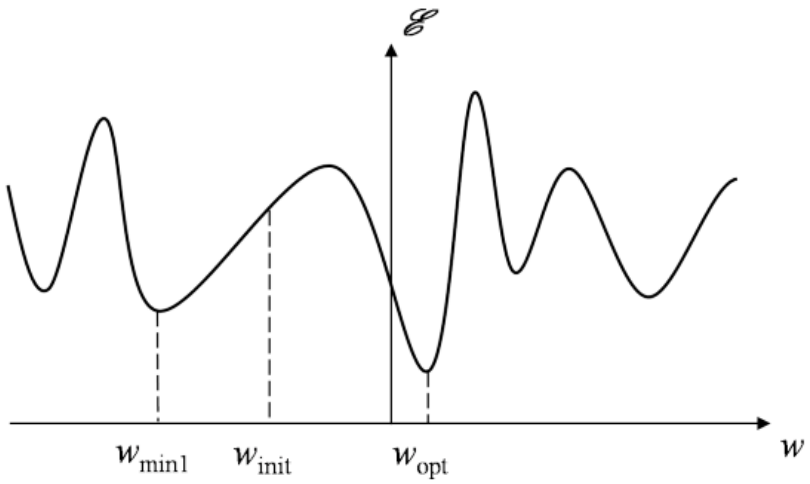


Figure 3.10: [3] Non-linear curve-fitting problem with several local minima, highlighting the initial synaptic weight, w_{init}, a local minimum point, w_{min1}, and the absolute minima, w_{opt}.

Chapter 4

Neural Network Applied To The Zero Order ASSE Scheme

This part of the work aims at testing the feasibility of using a neural network to solve the zero order ASSE scheme, which is essentially an application of neural networks to non-linear systems of equation. In order to achieve this, a MatLab routine which exploits the Deep Learning toolbox has been used. Network characteristics and further information about the MatLab routine are given in the next sections.

4.1 Network Characteristics And Input Data Profile

The Neural Network used for the purpose of this work is a Multi Layer Perceptron (sec. 3.1.4) which uses the Levenberg-Marquardt as the learning algorithm. Thanks to the MatLab routine, the following parameters have been modified during the different tests in order to find the most suitable solution:

1. Number of Training Epochs.
2. Number of Hidden Layers.
3. Number of Neurons for each Layer.
4. Number of input-output examples allocated for training .

The input data received by the network corresponds to values represented in equations 1.32 and 1.33 which are the coefficients of the Zero Order ASSE Scheme calculated for two subsequent time instants $[\tau, t]$, $\tau = t - 1$ since two equations at a time are considered. To summarize:

- $n_{t-1}, h_{t-1}, l_{t-1}, m_{t-1}$
- n_t, h_t, l_t, m_t

The computation of the input data is carried on through the same process used in the linear and non-linear routine (Appendix A and B). To conclude, a test with corrupted data has been carried out in order to evaluate the response of the network when data affected by white noise is given as input.

4.2 Presentation Of The Results

In order to present the results of this work, each different test is represented with four graphs and a table as already showed in chapter 2:

1. Comparison between the true and estimated value of AoA.
2. Error in the estimation of AoA, where the error is considered as: $AoA_{err} = AoA_{True} - AoA_{Estimated}$.
3. Comparison between the true and estimated value of AoS.

4. Error in the estimation of AoS, where the error is considered as: $AoS_{err} = AoS_{True} - AoS_{Estimated}$.

5. Table of statistic data:

   - 2σ Error [deg]
   - Mean Error [deg]
   - Max Error [deg]

The evaluation of the results quality is done following the work of Lerro A., Brandl A., Battipede M. and Gili P. [15] as already described in subsection 2.2.2.

4.3 Tests

The test phase of this work is aimed at evaluating the feasibility of using a neural network to solve the Zero Order ASSE Scheme. For this purpose, the maneuver described in section 2.1 has been used both for training and testing the network. Other tests on different maneuvers have also been carried out and are presented further in this chapter. In order to find the most suitable network, four different groups of tests have been done such that a best configuration of network parameters could be found. For sake of clarity, the best network parameters configuration found is specific for the purpose of this work and is limited to the number of tests carried out. The four groups are:

1. Group 1: Tests with an increasing number of training epochs.
2. Group 2: Tests with an increasing number of hidden layers.
3. Group 3: Tests with an increasing number of neurons for one layer.
4. Group 4: Tests with an increasing number of training data.

Further tests with a combination of the optimal parameters found in each group have been done and are later discussed.

4.3.1 Test Group 1: Effects Of Training Epochs

This first group of tests has been carried out in order to evaluate the effect of training epochs on the quality of the output results produced by the neural network. The maneuver used for training and test is described in subsection 2.2.2. Such simulated maneuver has a time history of 150 seconds, and the flight data is defined each millisecond, thus giving $1.5 * 10^4$ points to be distributed between test and training. The distribution between train and test data is summarized in the following table:

Phase	t_{init}	t_{step}	t_{end}
Train	100	100	length(**time_vect**)-100
Test	100	1	length(**time_vect**)-100

Table 4.1: Training and Test data for Group 1

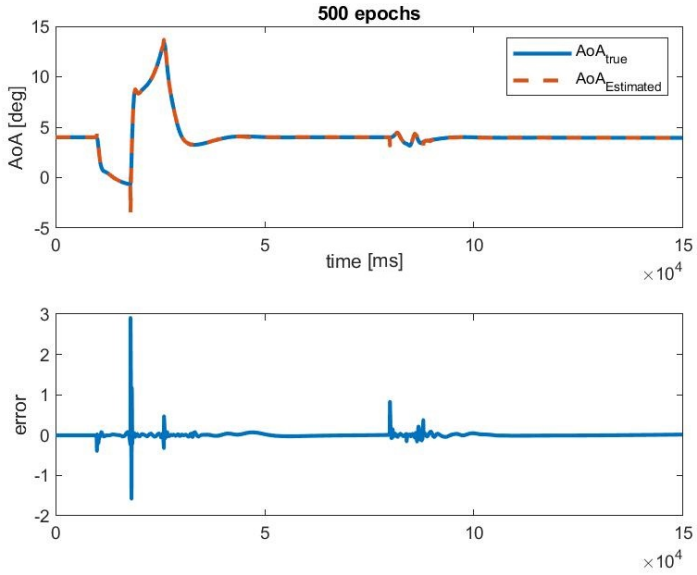
where t_{init} is an index which represents the starting index on the time vector (time_vect), t_{step} represents the step between two indexes on the time vector, and

t_{end} represents the final index to be considered for collecting training data. To summarize, all the data necessary, destined to training, is collected starting from an initial time corresponding to time_vect(t_{init}), and then continues with a step equal to t_{step} until the last value collected, corresponding to **time_vect**(length(**time_vect**)-100). The same principle applies for the test phase: The network collects all the points, starting from t_{init} all the way to t_{end} with a step equal to t_{step} (which is fixed to 1 in all the tests since the network is tested over the whole the time history) and estimates at each step the values of the aerodynamic angles. The results of this first group of tests are presented in the following pages.
The characteristic number of training epochs, hidden layers, neurons in each layer and the training time step t_{step} are summarized in the following table for each test:

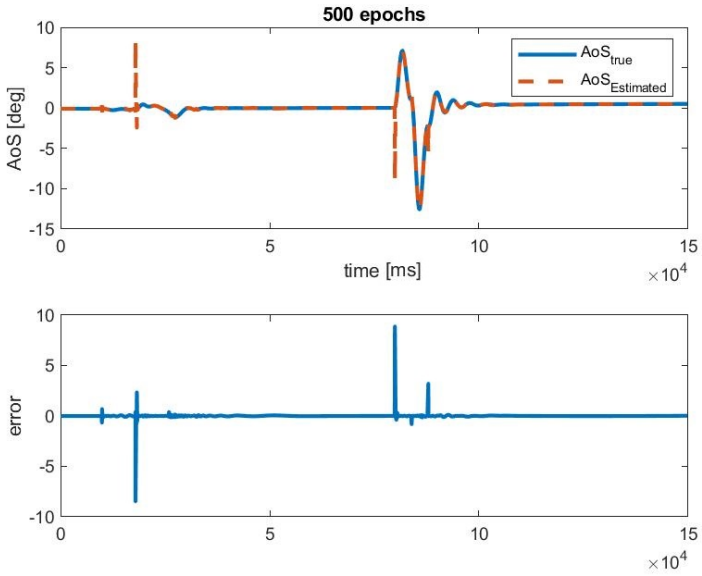
Test Number	training epochs	hidden layers	neurons/layer	t_{step}
Test 1	500	1	27	100
Test 2	1000	1	27	100
Test 3	2000	1	27	100
Test 4	4000	1	27	100
Test 5	6000	1	27	100

Table 4.2: Group 1 training characteristics

Test 1



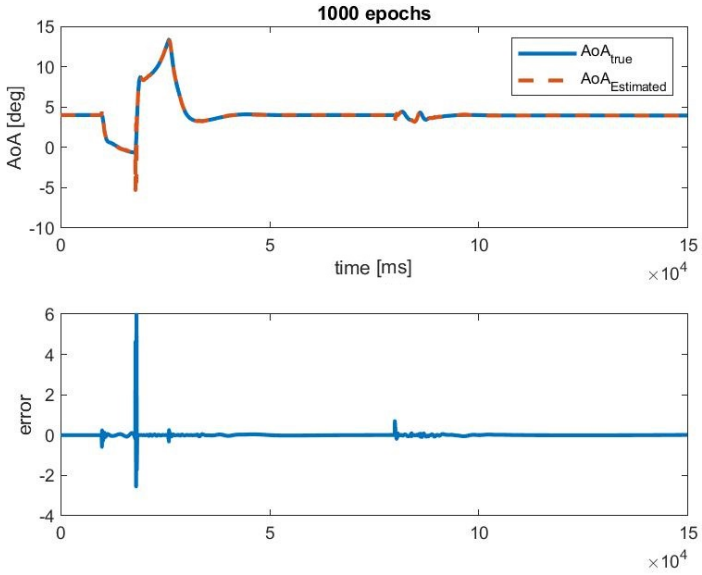
(a)



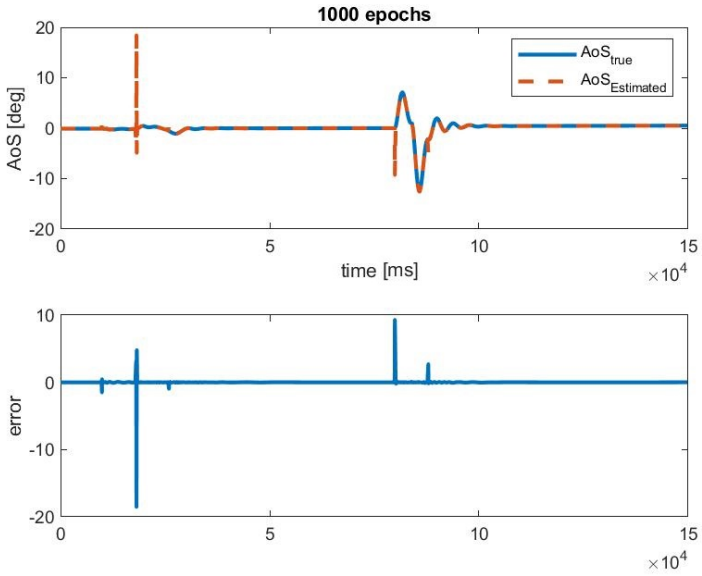
(b)

Figure 4.1: Group 1-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 2



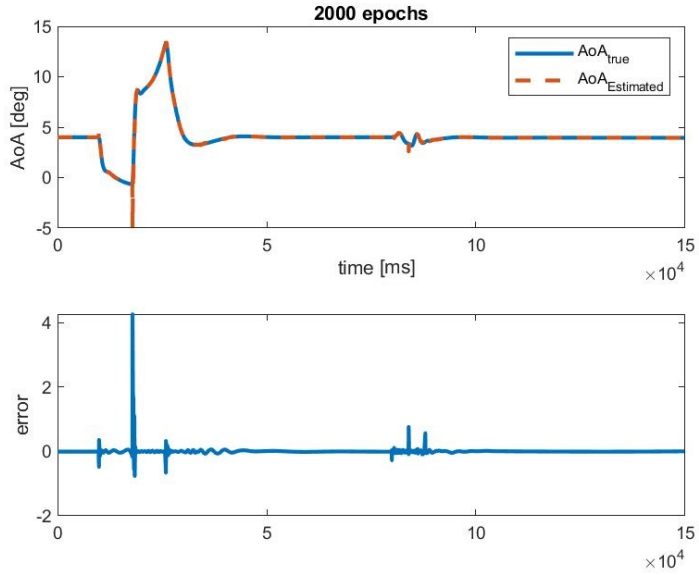
(a)



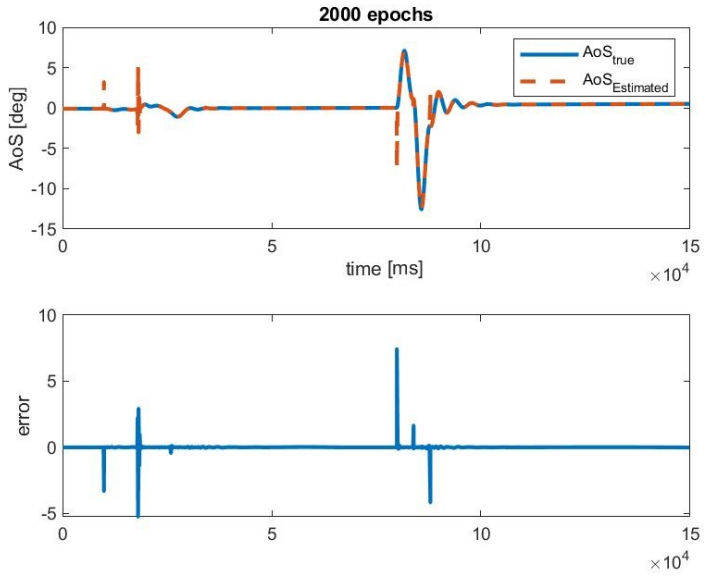
(b)

Figure 4.2: Group 1-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 3



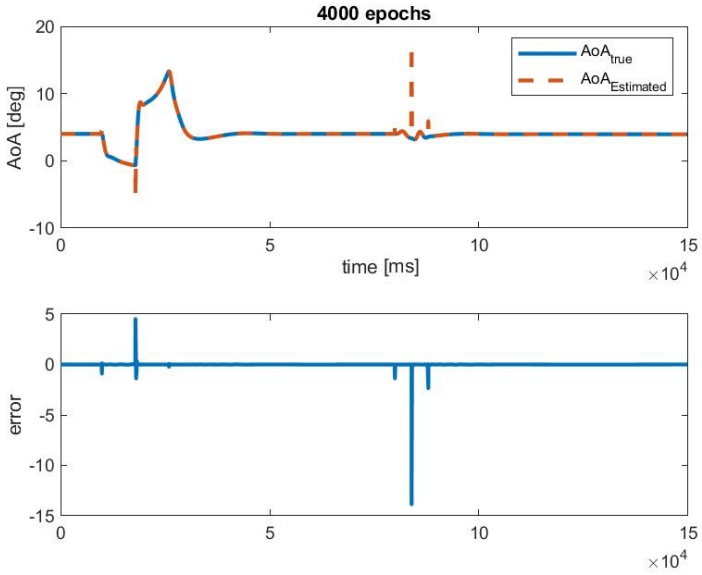
(a)



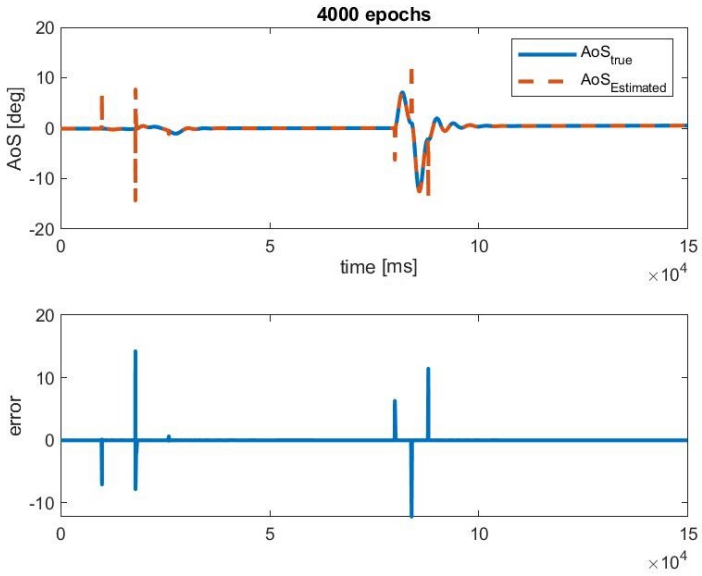
(b)

Figure 4.3: Group 1-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 4



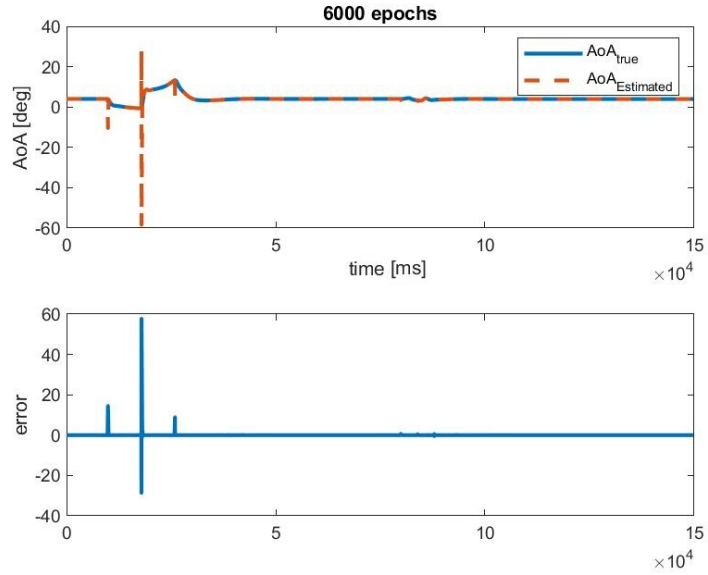
(a)



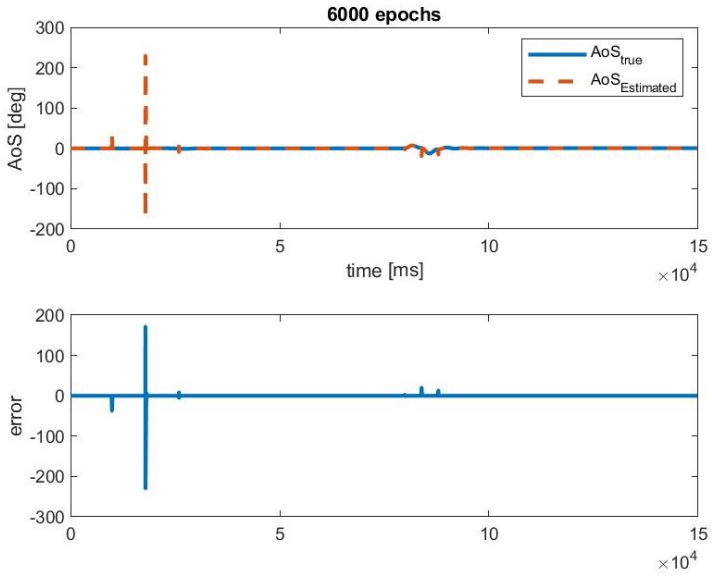
(b)

Figure 4.4: Group 1-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 5



(a)



(b)

Figure 4.5: Group 1-Test 5: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

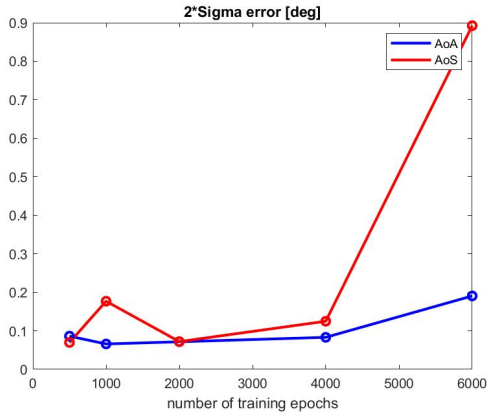
The statistic data of these results is presented in the following table and graphs:

AoA	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.0862	0.0007	2.906
Test 2	0.0659	0.0023	5.9741
Test 3	0.0716	0.0017	4.2715
Test 4	0.0832	-0.0026	13.876
Test 5	0.1906	0.0024	57.7647

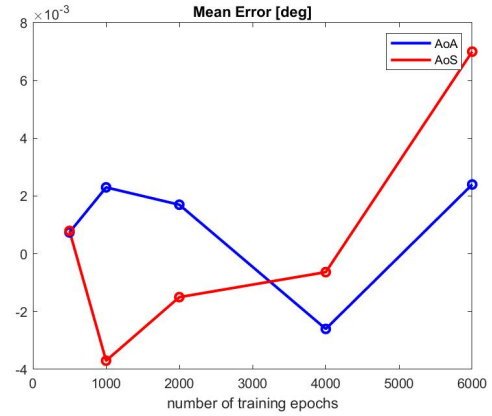
Table 4.3: Statistic data for AoA of Group 1 tests

AoS	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.0698	0.0008	8.8775
Test 2	0.1769	-0.0037	18.5335
Test 3	0.0721	-0.0015	7.4261
Test 4	0.1249	-0.0006	14.2399
Test 5	0.8924	0.007	229.6894

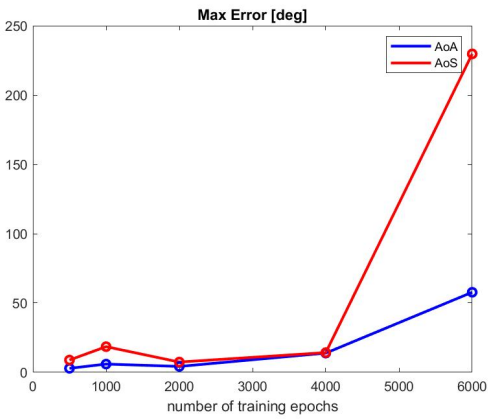
Table 4.4: Statistic data for AoS of Group 1 tests



(a)



(b)



(c)

Figure 4.6: Group 1 Statistic data

As it is possible to notice from these results, the first test of this group is the one that produced the most satisfying results among all the others. As a mater of fact, in **Test 1**, which corresponds to 500 training epochs, both the mean and max error are closer to zero than the other attempts, and both AoA and AoS satisfy the 2σ error requirements [15] with the best compromise. It is also possible to notice that increasing the number of training epochs , on average, leads to a worse prediction of the aerodynamic angles by the network.

4.3.2 Test Group 2: Effects Of The Number Of Hidden Layers

The second group of tests aims at evaluating the performance of MLP neural networks with a different number of hidden layers.

Phase	t_{init}	t_{step}	t_{end}
Train	100	100	length(**time_vect**)-100
Test	100	1	length(**time_vect**)-100

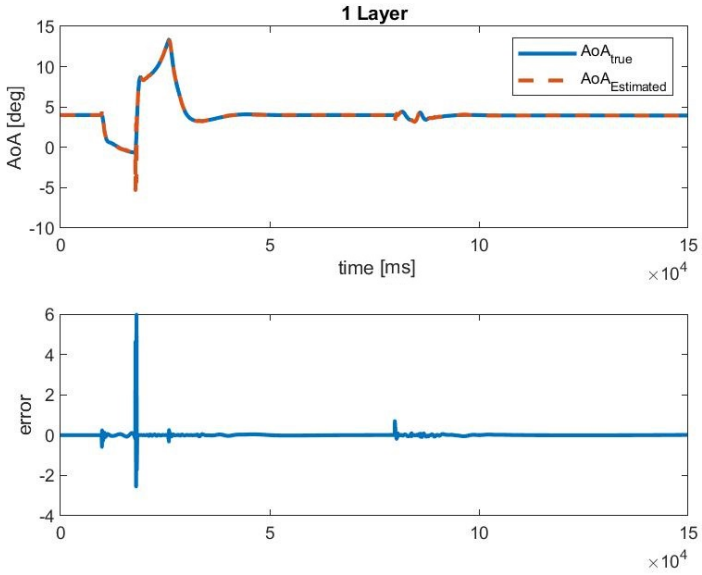
Table 4.5: Training and Test data for Group 2

The characteristic number of training epochs, hidden layers, neurons in each layer and the training time step t_{step} are summarized in the following table for each test:

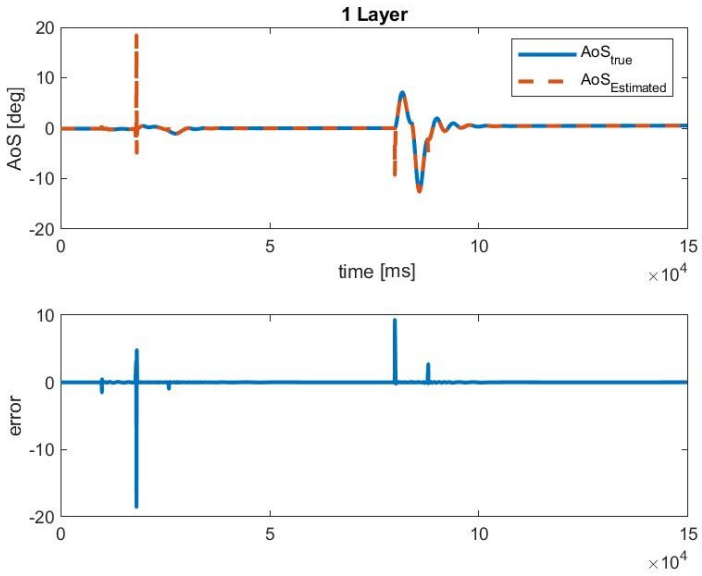
Test Number	training epochs	hidden layers	neurons/layer	t_{step}
Test 1	1000	1	27	100
Test 2	1000	2	27	100
Test 3	1000	3	27	100
Test 4	1000	4	27	100

Table 4.6: Group 2 training characteristics

Test 1



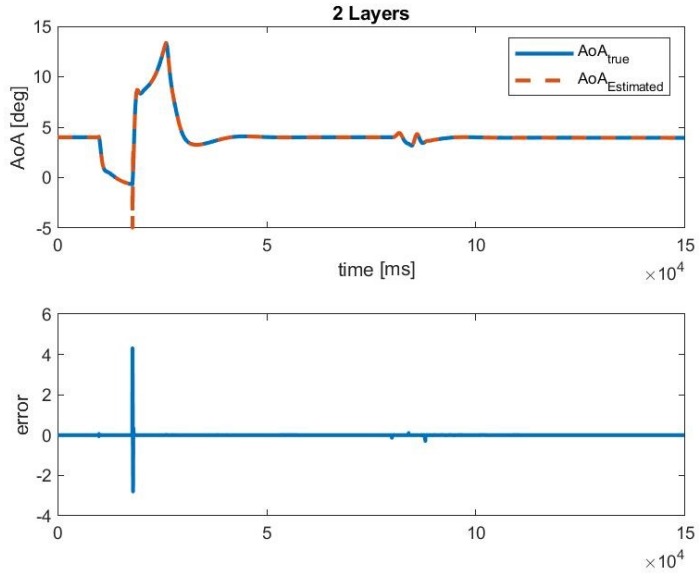
(a)



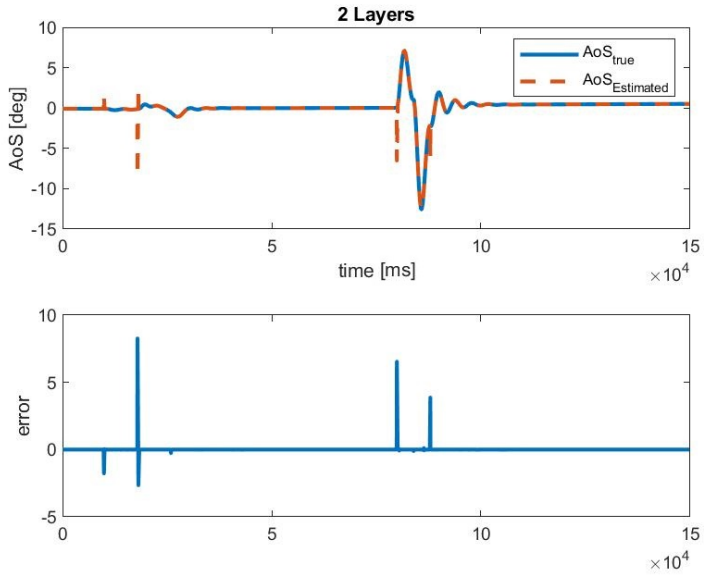
(b)

Figure 4.7: Group 2-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 2



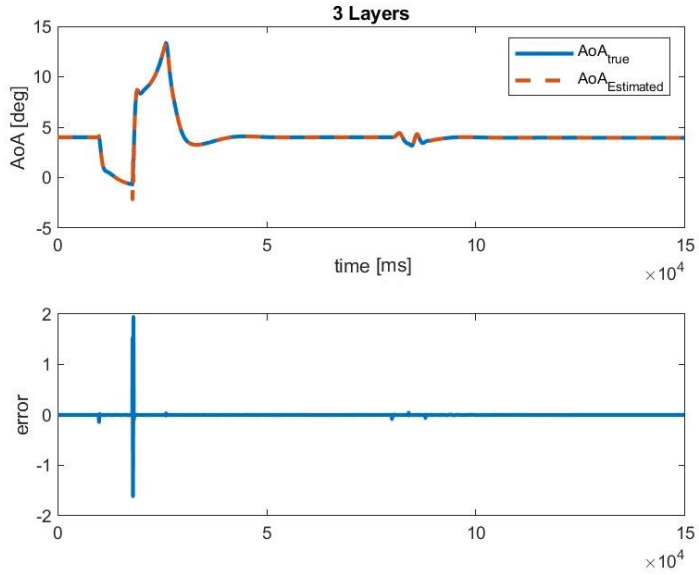
(a)



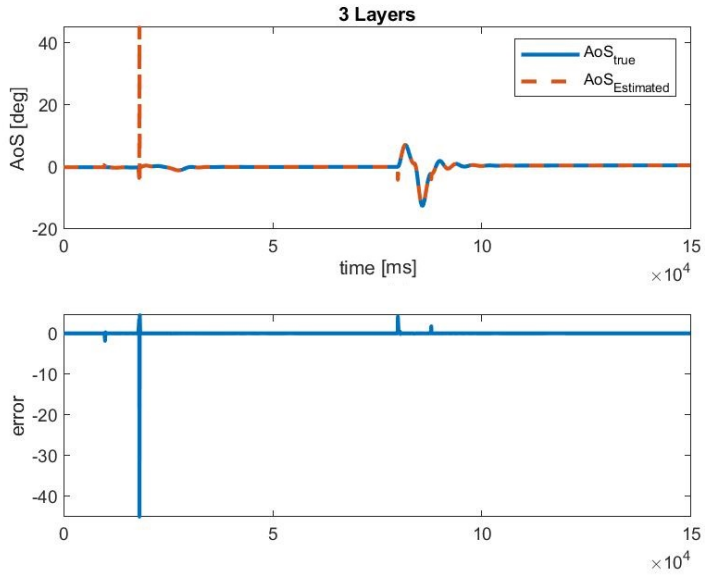
(b)

Figure 4.8: Group 2-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 3



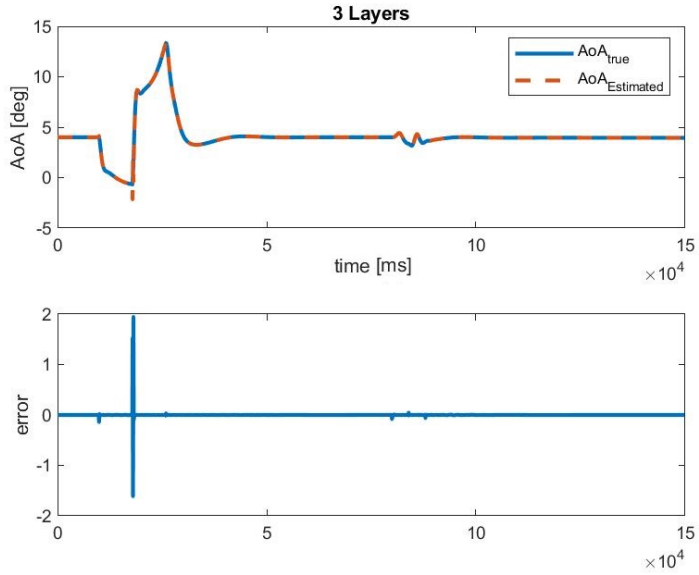
(a)



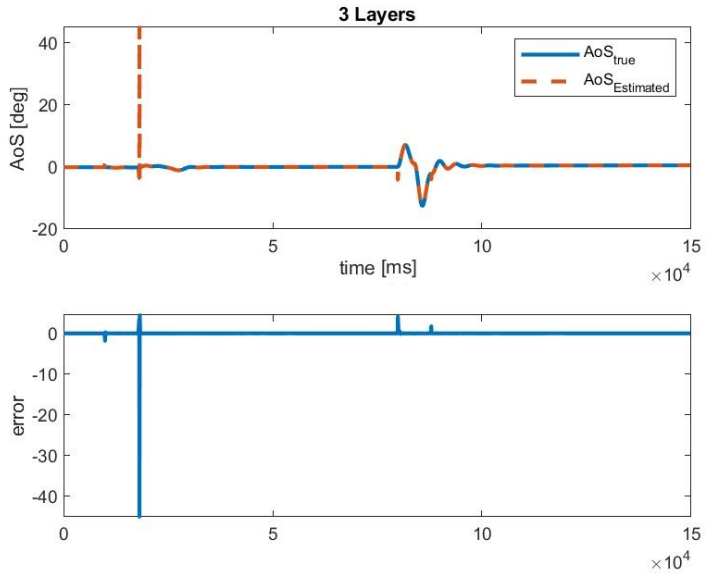
(b)

Figure 4.9: Group 2-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 4



(a)



(b)

Figure 4.10: Group 2-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

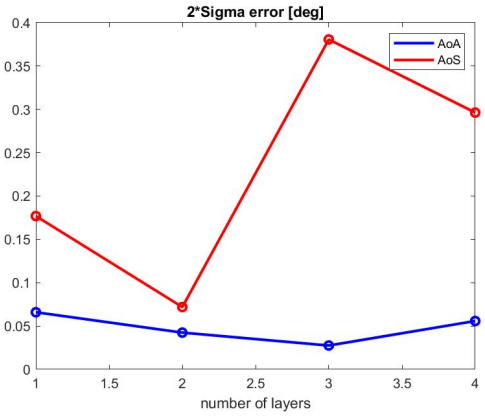
The statistic data of these results is presented in the following tables and graphs:

AoA	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.0659	0.0023	5.9741
Test 2	0.0424	0.0006	4.3094
Test 3	0.0275	0.0008	1.9468
Test 4	0.0558	-0.0035	6.4501

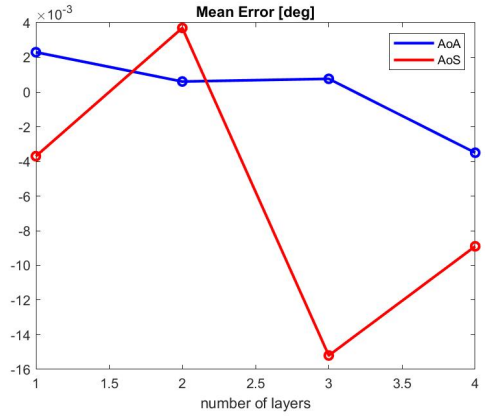
Table 4.7: Statistic data for AoA of Group 2 tests

AoS	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.1769	-0.0037	18.5335
Test 2	0.0719	0.0037	8.268
Test 3	0.3807	-0.0152	45.0436
Test 4	0.2963	-0.0089	35.6211

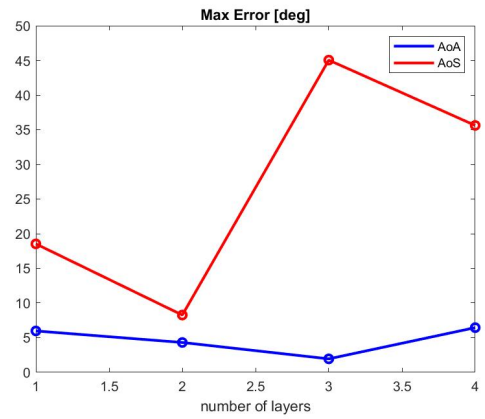
Table 4.8: Statistic data for AoS of Group 2 tests



(a)



(b)



(c)

Figure 4.11: Group 2 Statistic data

In the case of the second group of tests, the most suitable network can be identified in **Test 2**. In fact, the test with two hidden layers of 27 neurons each, shows acceptable values of the 2σ error, with the best compromise between AoA and AoS. The same can be noticed for the mean and max errors. In the case of these four tests related to the effects of the number of layers, no definite trend of the results quality can be identified when adding more hidden layers.

4.3.3 Test Group 3: Effects Of The Number Of Neurons For Each Layer

The third group of tests aims at evaluating the effects of a different number of neurons for one layer.

Phase	t_{init}	t_{step}	t_{end}
Train	100	100	length(**time_vect**)-100
Test	100	1	length(**time_vect**)-100

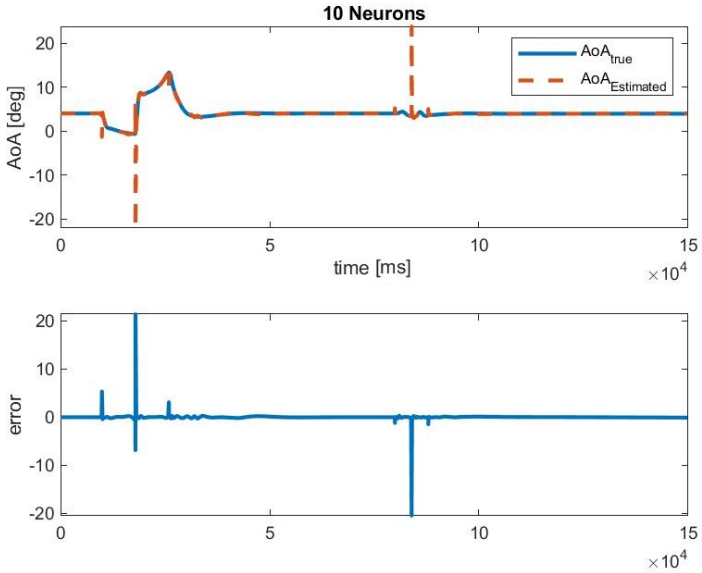
Table 4.9: Training and Test data for Group 3

The characteristic number of training epochs, hidden layers, neurons in each layer and the training time step t_{step} are summarized in the following table for each test:

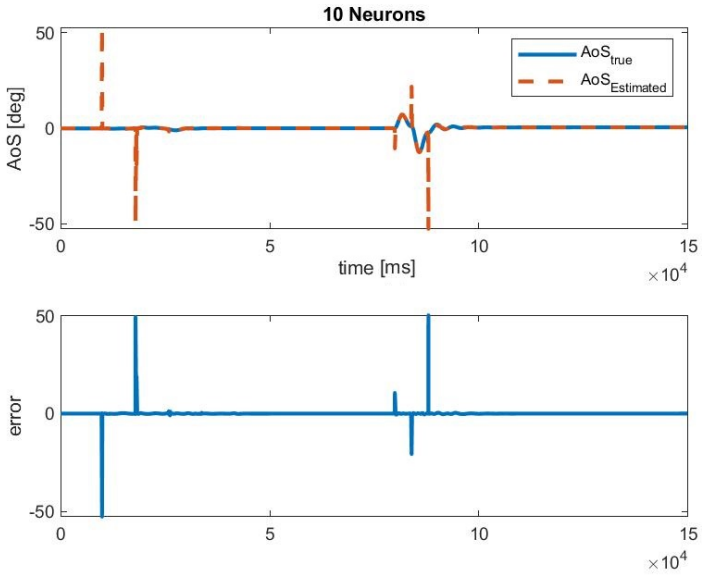
Test Number	training epochs	hidden layers	neurons/layer	t_{step}
Test 1	1000	1	10	100
Test 2	1000	1	20	100
Test 3	1000	1	27	100
Test 4	1000	1	35	100
Test 5	1000	1	50	100

Table 4.10: Group 3 training characteristics

Test 1



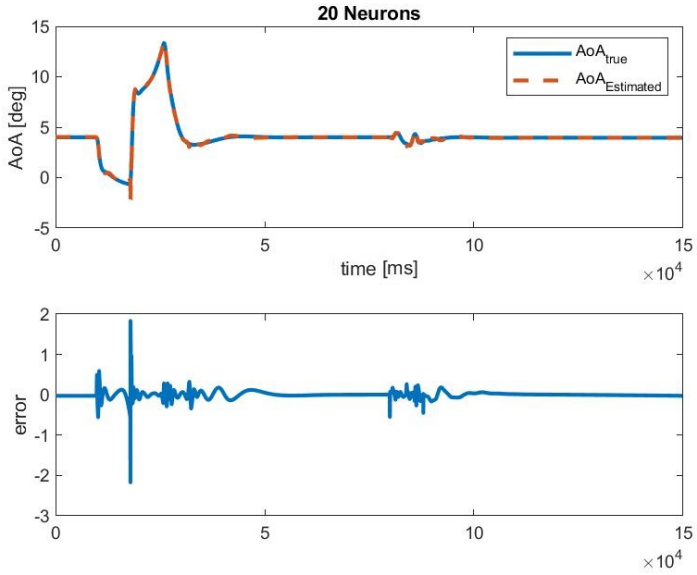
(a)



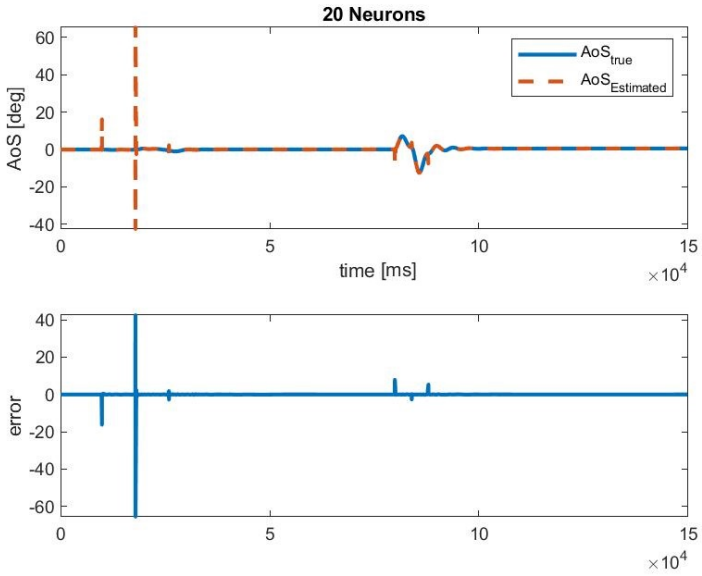
(b)

Figure 4.12: Group 3-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 2



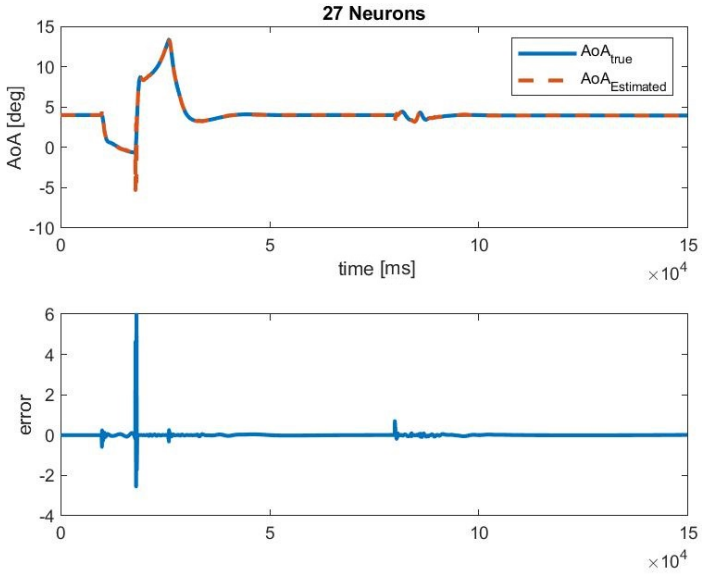
(a)



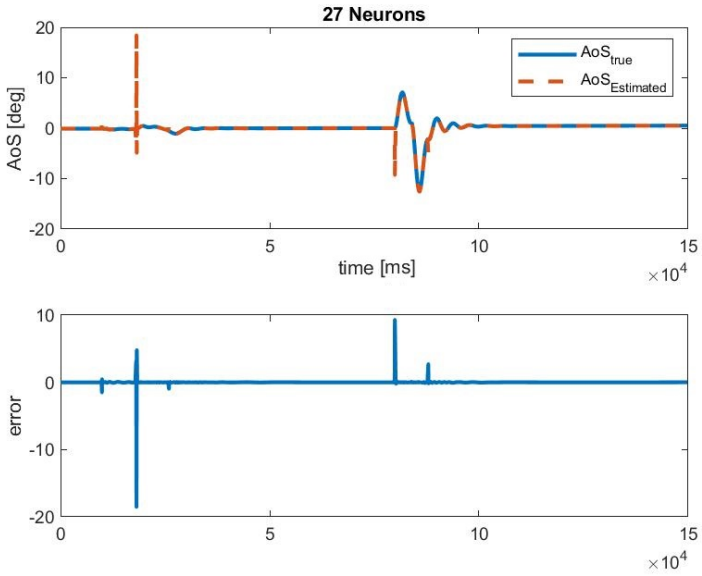
(b)

Figure 4.13: Group 3-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 3



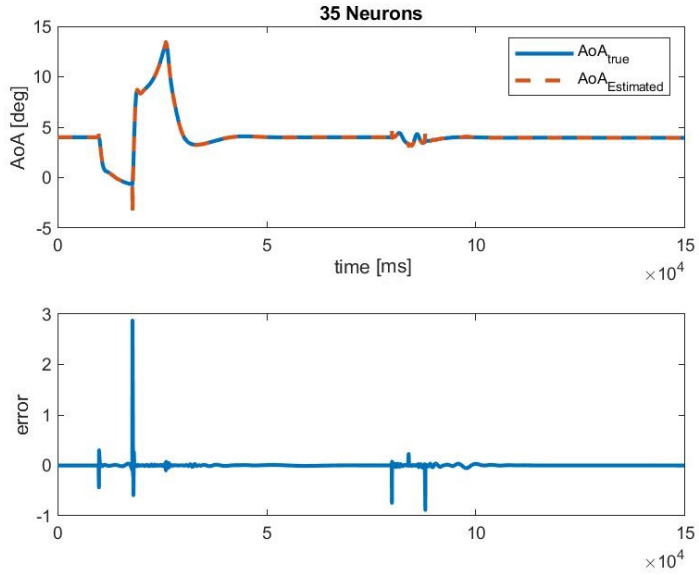
(a)



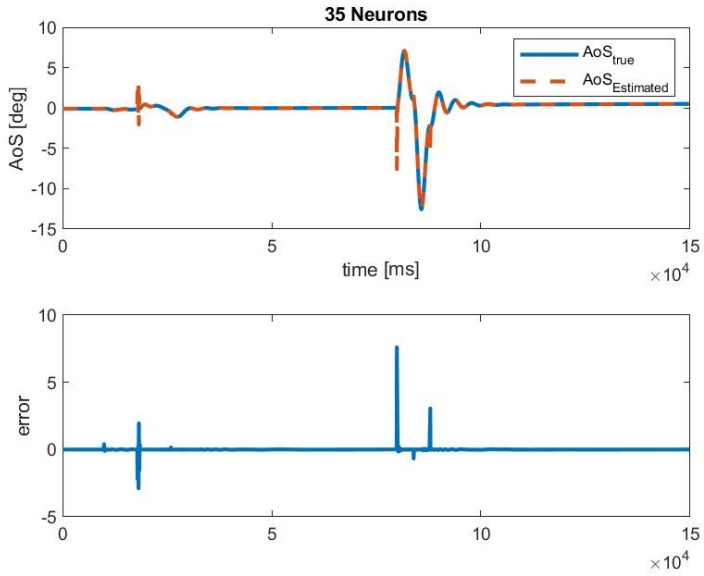
(b)

Figure 4.14: Group 3-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 4



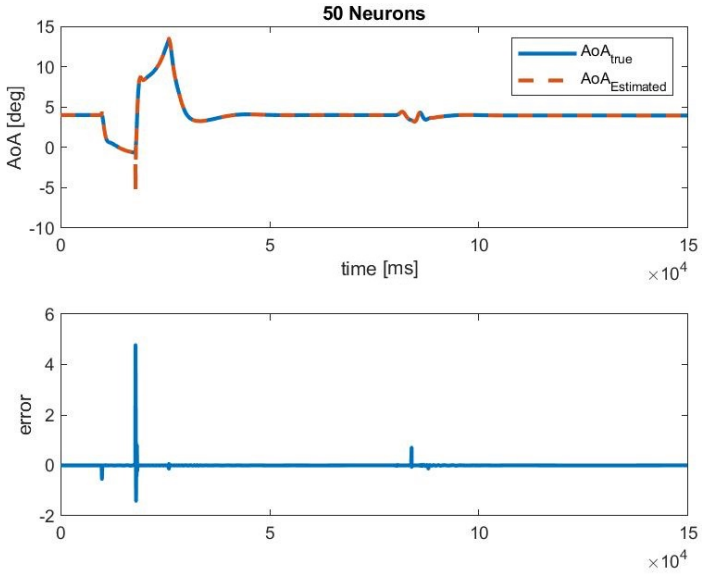
(a)



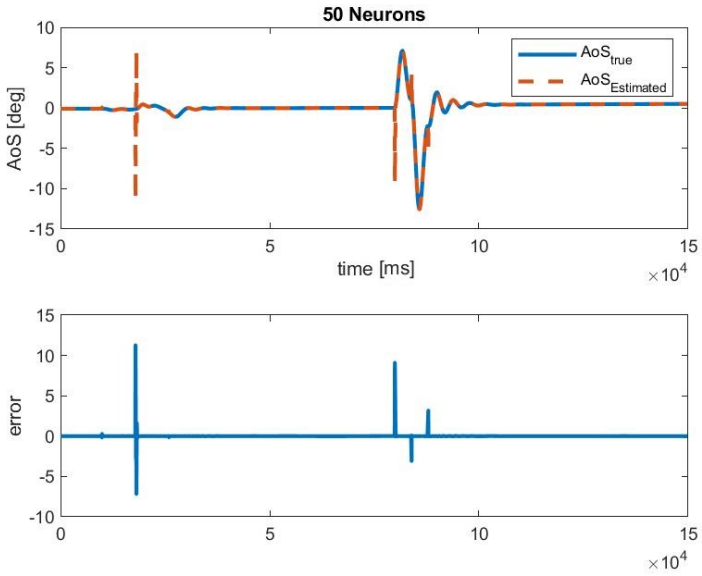
(b)

Figure 4.15: Group 3-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 5



(a)



(b)

Figure 4.16: Group 3-Test 5: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

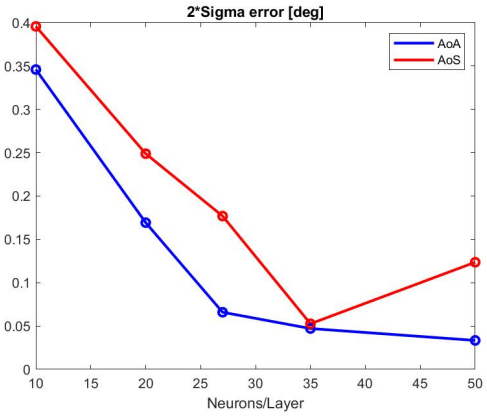
The statistic data of these results is presented in the following tables and graphs:

AoA	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.3465	-0.0025	21.4073
Test 2	0.1693	0.0001	2.1733
Test 3	0.0659	0.0023	5.9741
Test 4	0.0471	0.00003	2.8745
Test 5	0.0334	0.0009	4.7756

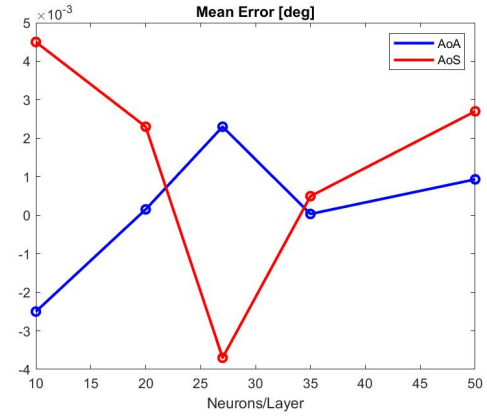
Table 4.11: Statistic data for AoA of Group 3 tests

AoS	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.3958	0.0045	52.8383
Test 2	0.2489	0.0023	65.5718
Test 3	0.1769	-0.0037	18.5335
Test 4	0.0526	0.0005	7.6148
Test 5	0.1236	0.0027	11.2857

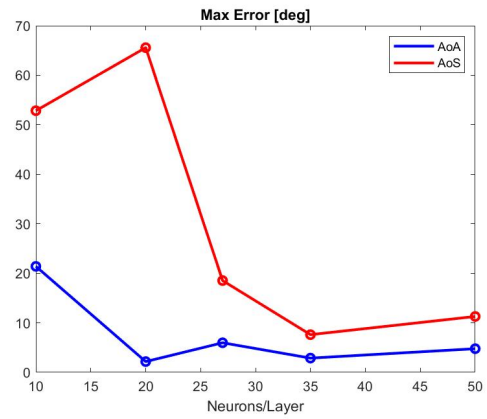
Table 4.12: Statistic data for AoS of Group 3 tests



(a)



(b)



(c)

Figure 4.17: Group 3 Statistic data

The best network in this third group of test is the one corresponding to **Test 4**, with 35 neurons for one hidden layer. The results show the best values for the 2σ error, which satisfies the requirements, and for mean and max error. As far as the number of neurons is concerned, neural network theory discussed in chapter 3, tells that there is an optimal number of neurons, since increasing this number too much leads to the problem of overfitting.

4.3.4 Test Group 4: Effects Of The Number Of Training Data

The fourth group of tests aims at evaluating the effects of an increasing number of training data.

Phase	t_{init}	t_{step}	t_{end}
Train	100	variable	length(**time_vect**)-100
Test	100	1	length(**time_vect**)-100

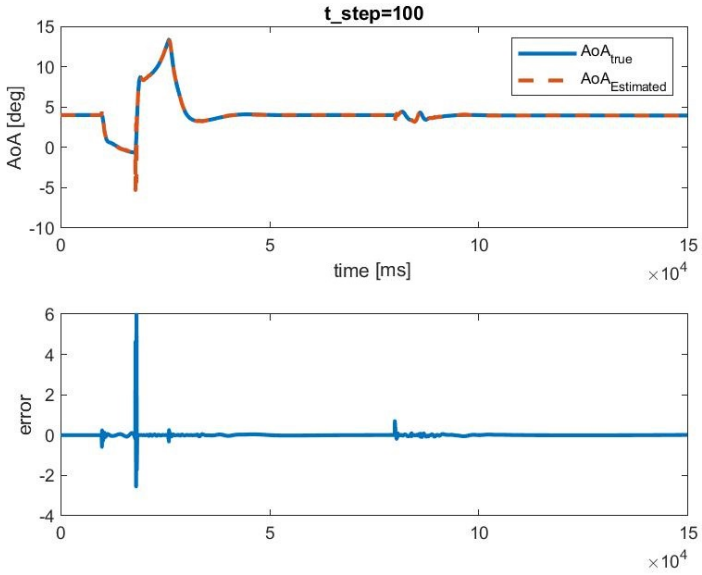
Table 4.13: Training and Test data for Group 4

The characteristic number of training epochs, hidden layers, neurons in each layer and the training time step t_{step} are summarized in the following table for each test:

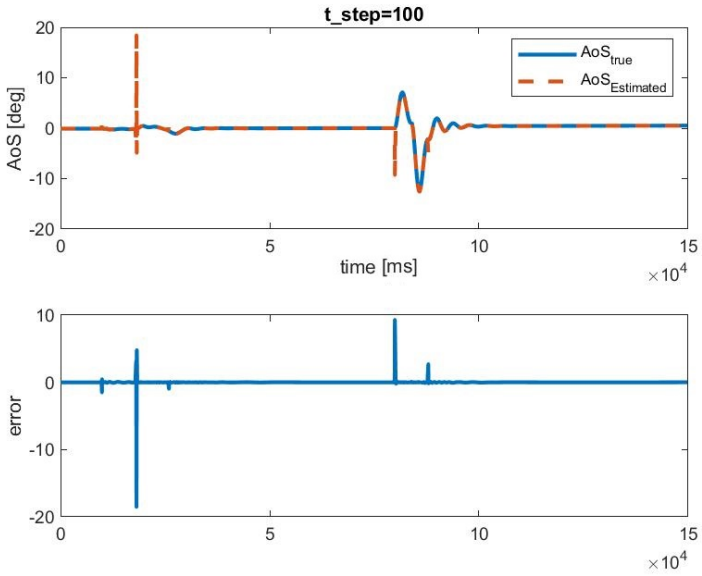
Test Number	training epochs	hidden layers	neurons/layer	t_{step}
Test 1	1000	1	27	100
Test 2	1000	1	27	75
Test 3	1000	1	27	50
Test 4	1000	1	27	25
Test 5	1000	1	27	20

Table 4.14: Group 4 training characteristics

Test 1



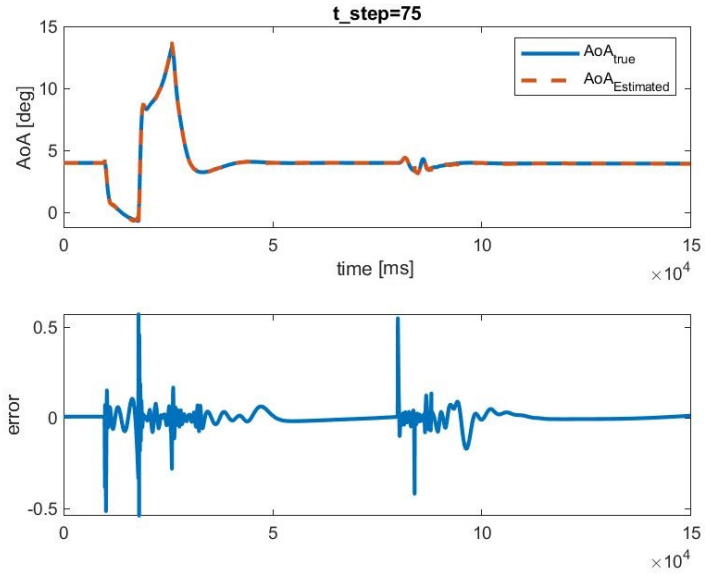
(a)



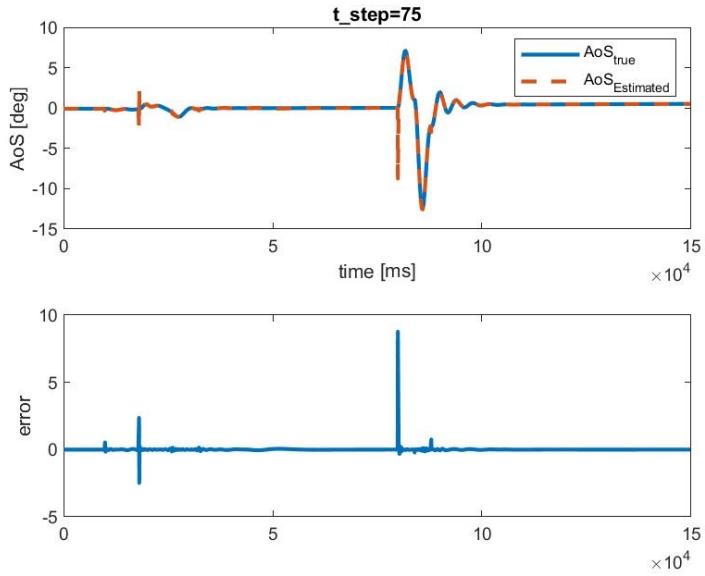
(b)

Figure 4.18: Group 4-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 2



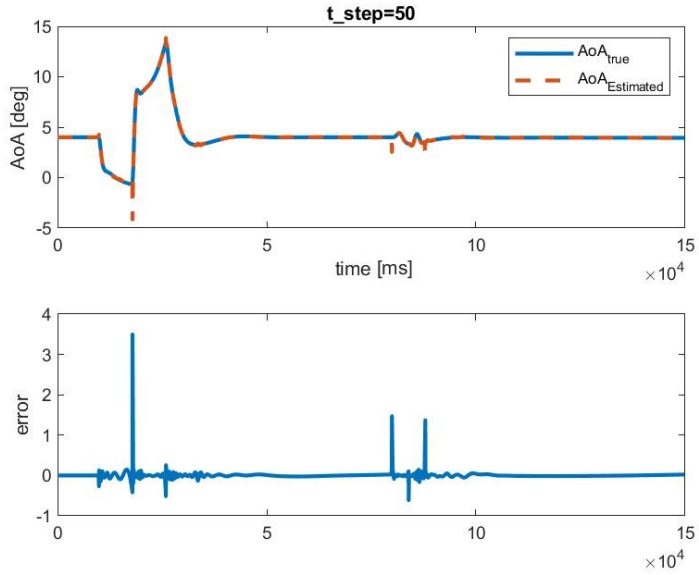
(a)



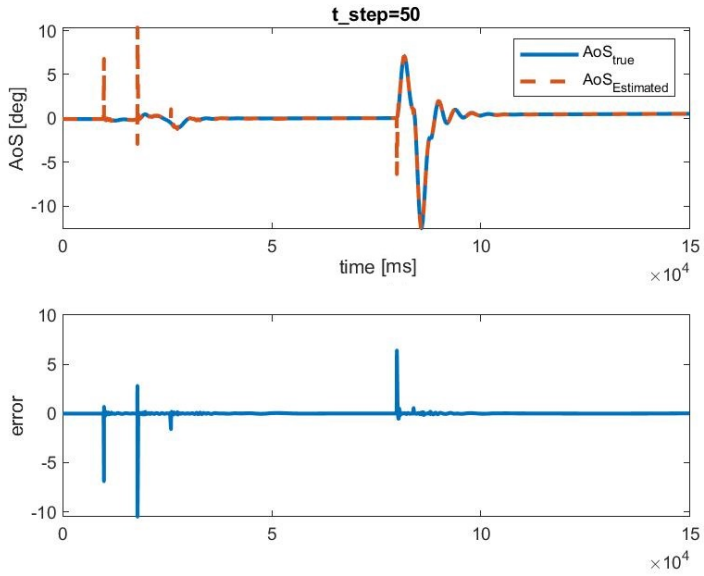
(b)

Figure 4.19: Group 4-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 3



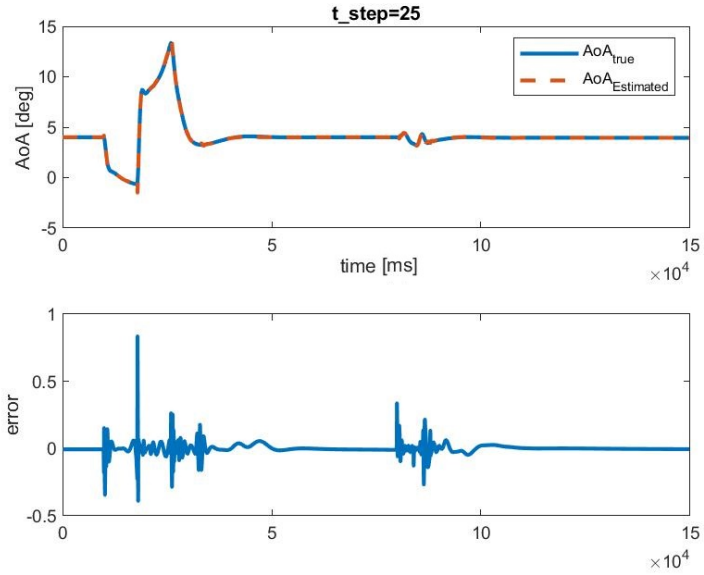
(a)



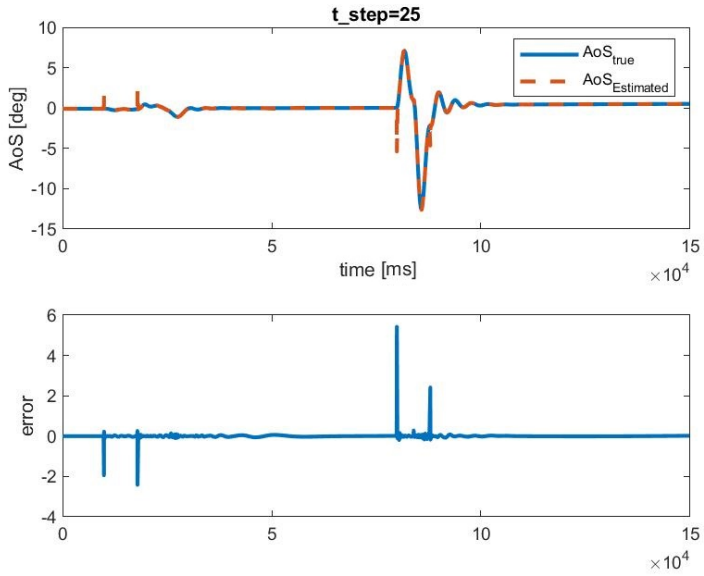
(b)

Figure 4.20: Group 4-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 4



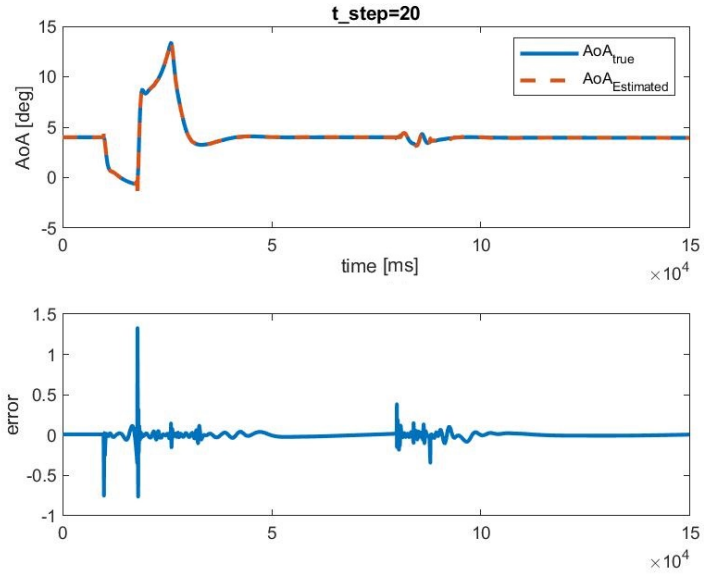
(a)



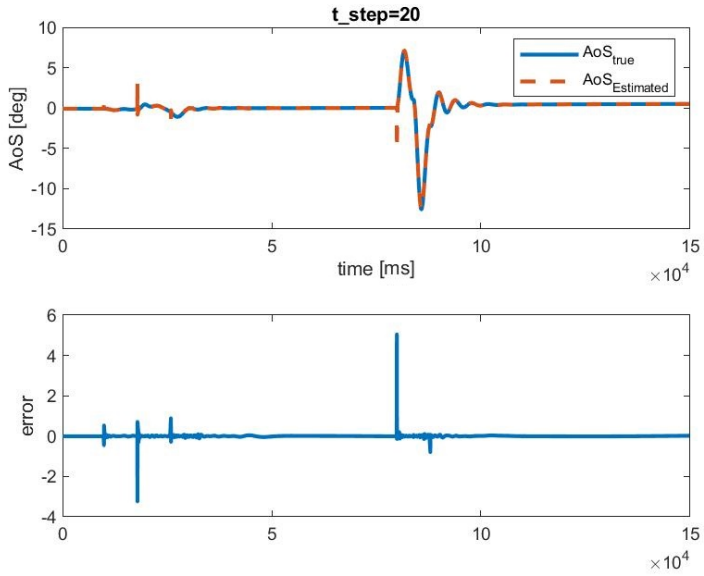
(b)

Figure 4.21: Group 4-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 5



(a)



(b)

Figure 4.22: Group 4-Test 5: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

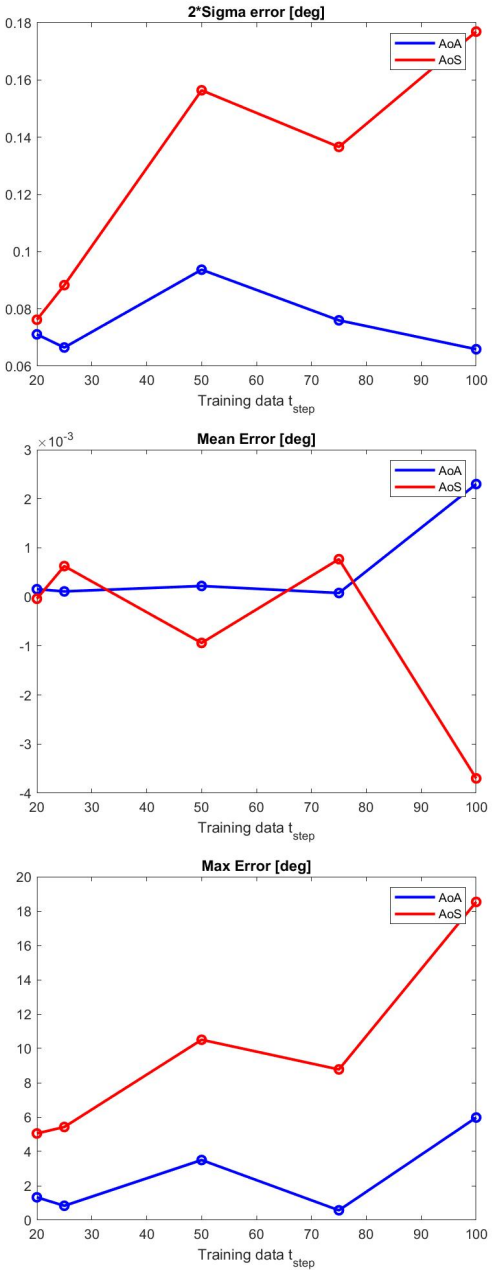
The statistic data of these results is presented in the following tables and graphs:

AoA	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.0711	0.0001	1.3275
Test 2	0.0665	0.0001	0.8354
Test 3	0.0937	0.0002	3.4995
Test 4	0.0764	0.00008	0.5718
Test 5	0.0659	0.0023	5.9741

Table 4.15: Statistic data for AoA of Group 4 tests

AoS	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.0762	-0.00004	5.0465
Test 2	0.0883	0.0006	5.4246
Test 3	0.1564	-0.0009	10.5051
Test 4	0.1366	0.0008	8.7754
Test 5	0.1769	-0.0037	18.5335

Table 4.16: Statistic data for AoS of Group 4 tests



(a)

(b)

(c)

Figure 4.23: Group 4 Statistic data

The most suitable network is, as expected, **Test 5**. When the number of data given for training increases, the network has more input-output examples on which it can rely on to adjust the synaptic weights, thus making more accurate predictions.

4.3.5 Combined Tests

This last group of tests has been carried out in order to find the best possible network for the sake of this work. The best characteristics showed in the results of the former groups of tests have been combined to train a network which could make the best predictions. Results and network characteristics are presented for each test the same way as for the previous groups of tests.

Phase	t_{init}	t_{step}	t_{end}
Train	100	100	length(**time_vect**)-100
Test	100	1	length(**time_vect**)-100

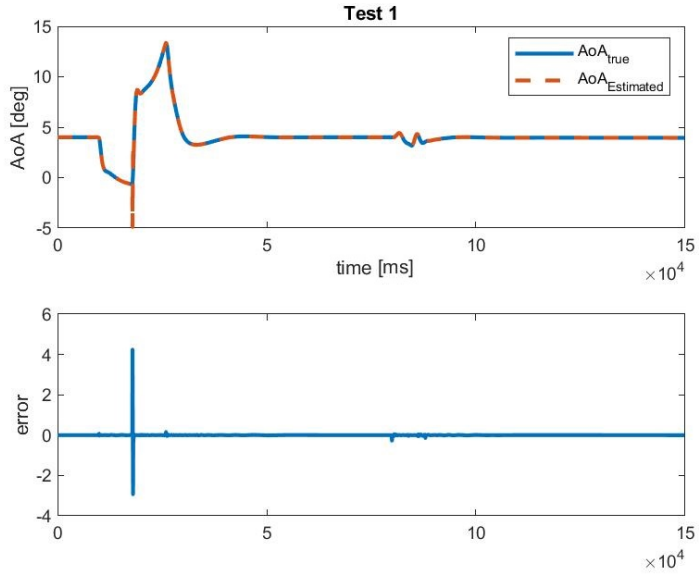
Table 4.17: Training and Test data for the combined group

The characteristic number of training epochs, hidden layers, neurons in each layer and the training time step t_{step} are summarized in the following table for each test:

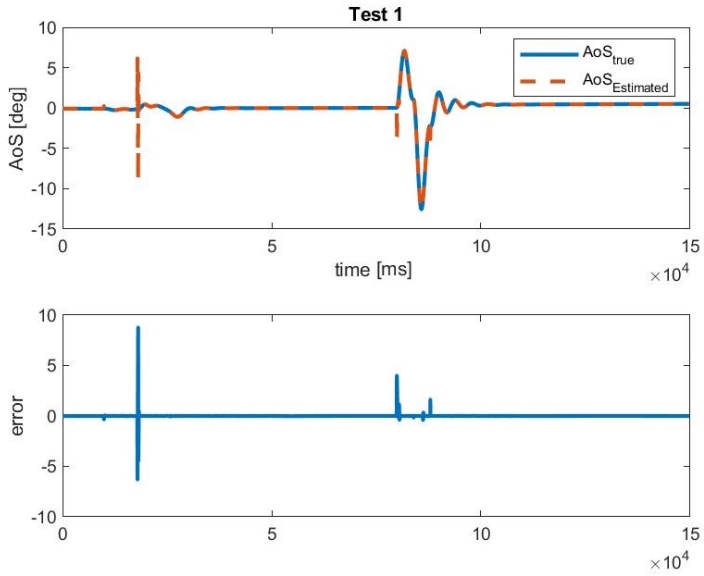
Test Number	training epochs	hidden layers	neurons/layer	t_{step}
Test 1	500	2	[35 35]	100
Test 2	500	2	[25 15]	100
Test 3	500	2	[18 17]	100
Test 4	500	2	[20 20]	100
Test 5	500	2	[25 20]	100

Table 4.18: Combined Group training characteristics

Test 1



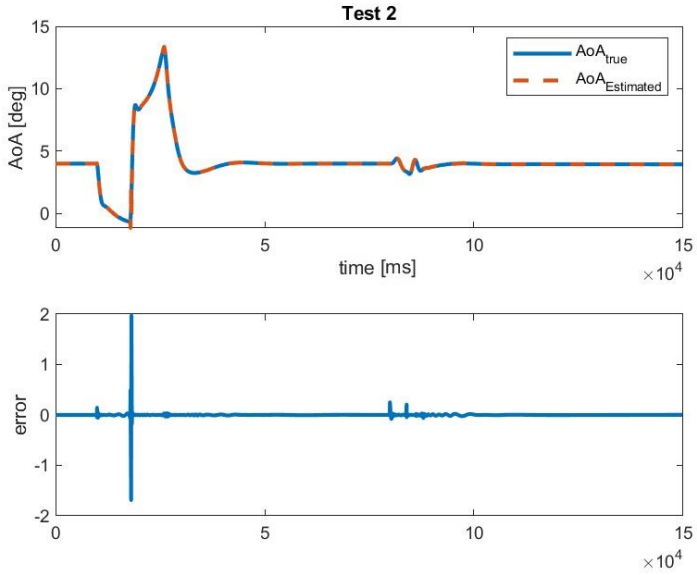
(a)



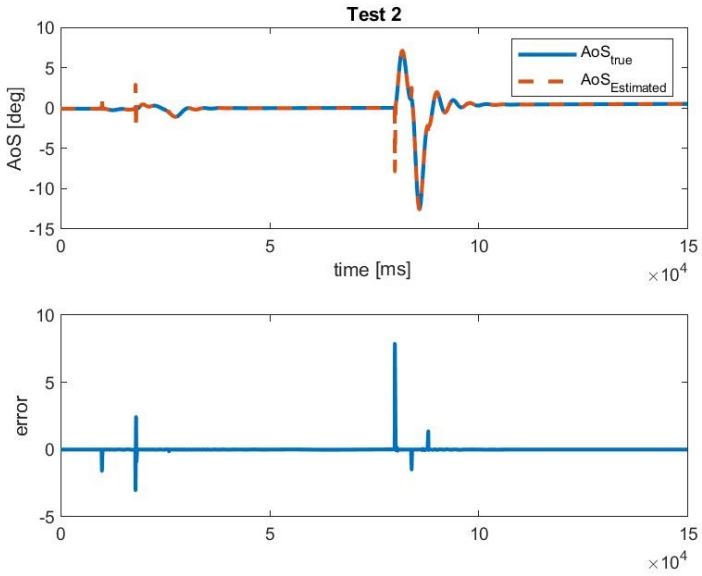
(b)

Figure 4.24: Combined Group-Test 1: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 2



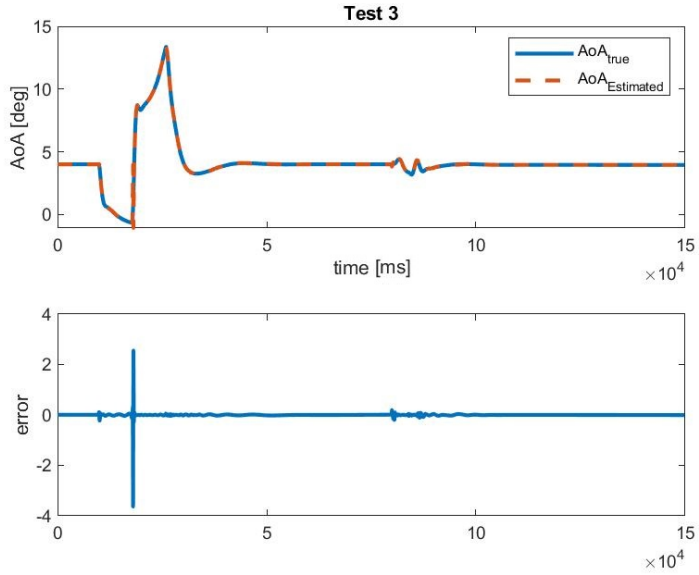
(a)



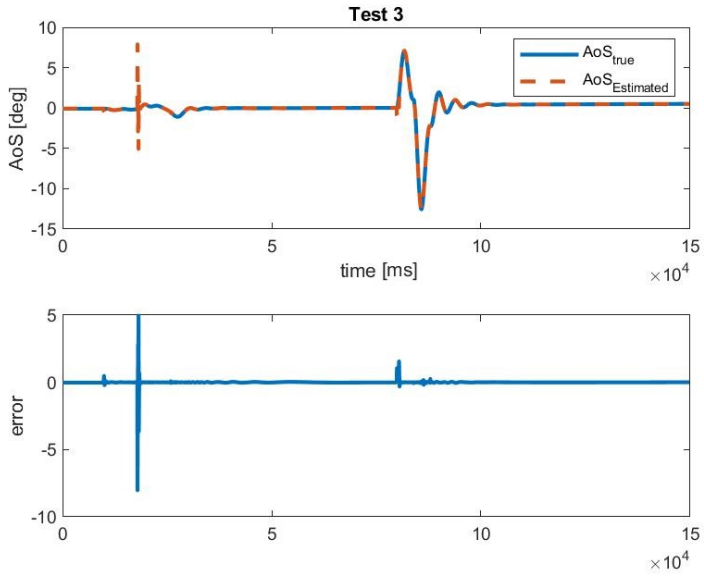
(b)

Figure 4.25: Combined Group-Test 2: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 3



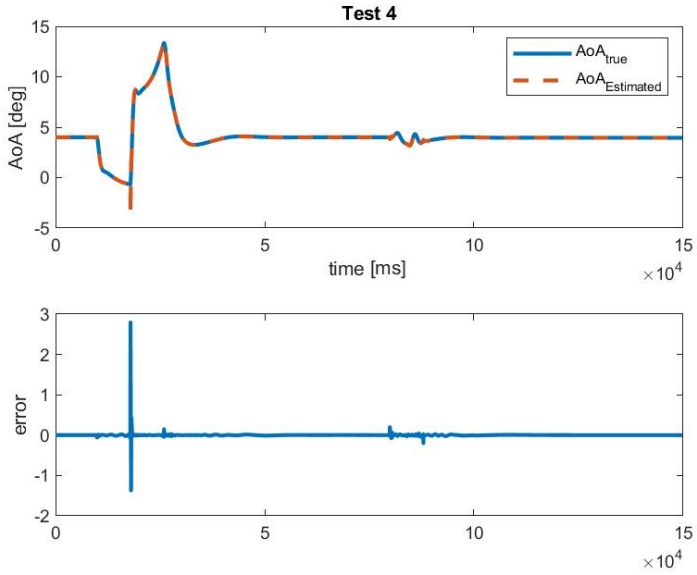
(a)



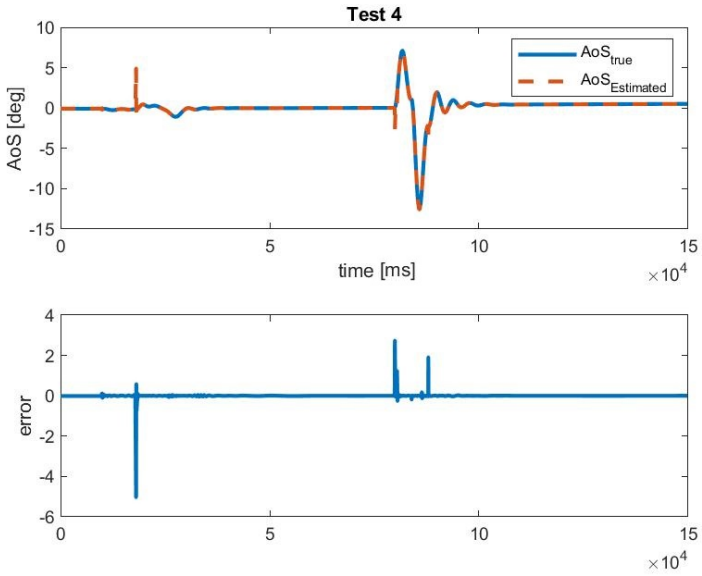
(b)

Figure 4.26: Combined Group-Test 3: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 4



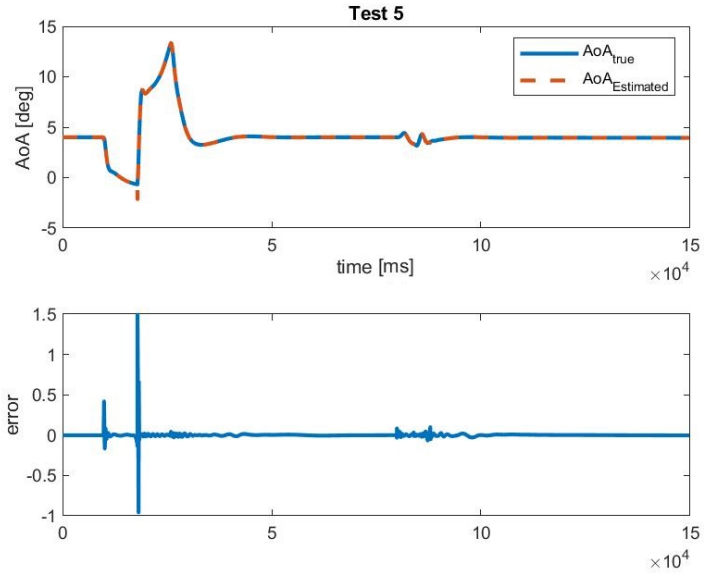
(a)



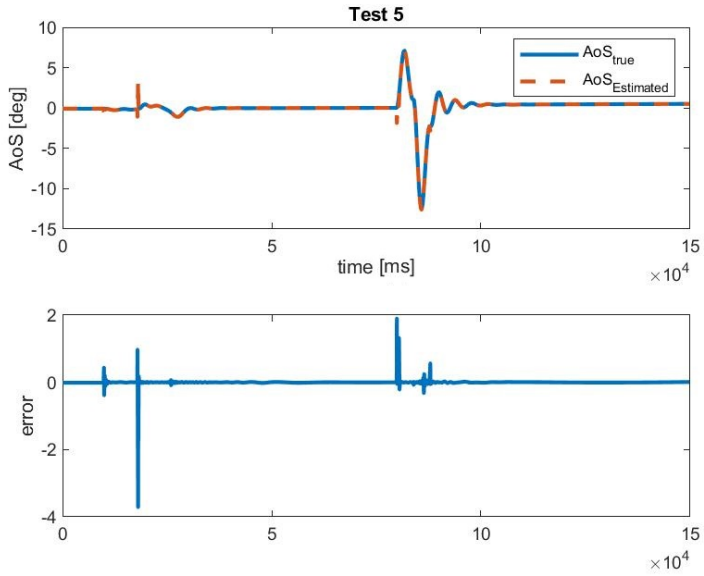
(b)

Figure 4.27: Combined Group-Test 4: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

Test 5



(a)



(b)

Figure 4.28: Combined Group-Test 5: Comparison between the true and the network estimated value of AoA (a) and AoS (b)

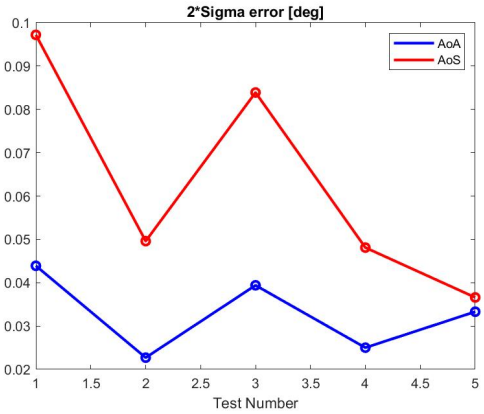
The statistic data of these results is presented in the following tables and graphs:

AoA	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.0439	0.0005	4.2442
Test 2	0.0227	0.0005	1.9741
Test 3	0.0394	-0.0006	3.6356
Test 4	0.0253	0.0005	2.7973
Test 5	0.0333	0.0006	1.4939

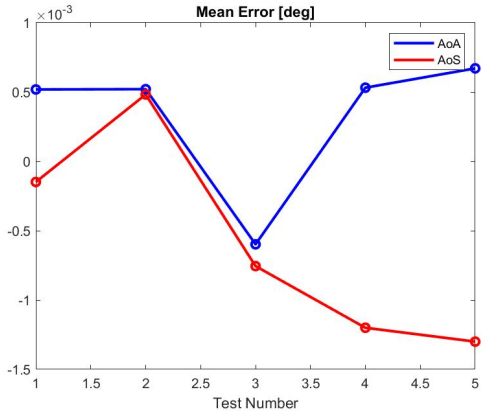
Table 4.19: Statistic data for AoA of combined group tests

AoS	2σ error [deg]	mean error [deg]	max error [deg]
Test 1	0.0972	-0.0001	8.779
Test 2	0.0496	0.0005	7.8709
Test 3	0.0839	-0.0007	8.0382
Test 4	0.0481	-0.0012	5.0278
Test 5	0.0366	-0.0013	3.7133

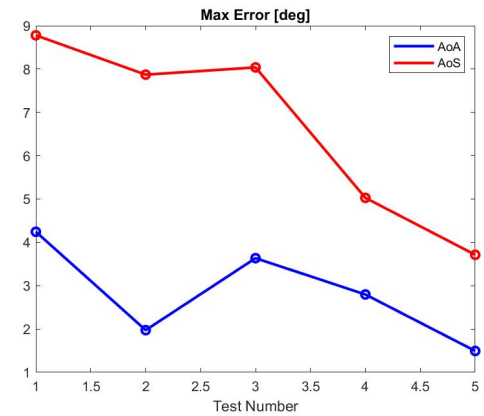
Table 4.20: Statistic data for AoS of combined group tests



(a)



(b)



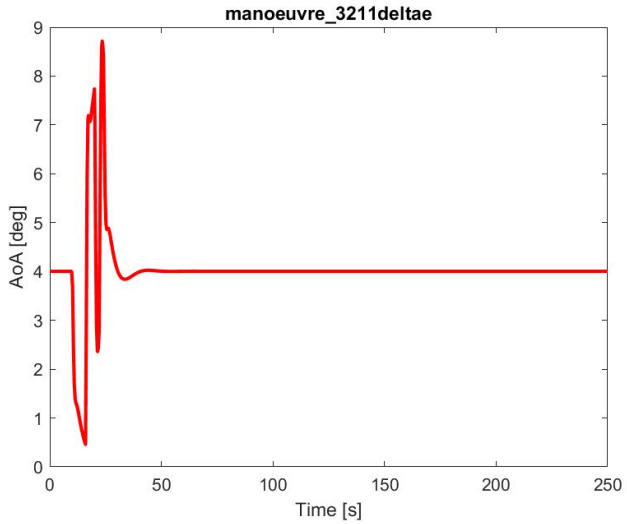
(c)

Figure 4.29: Combined group Statistic data

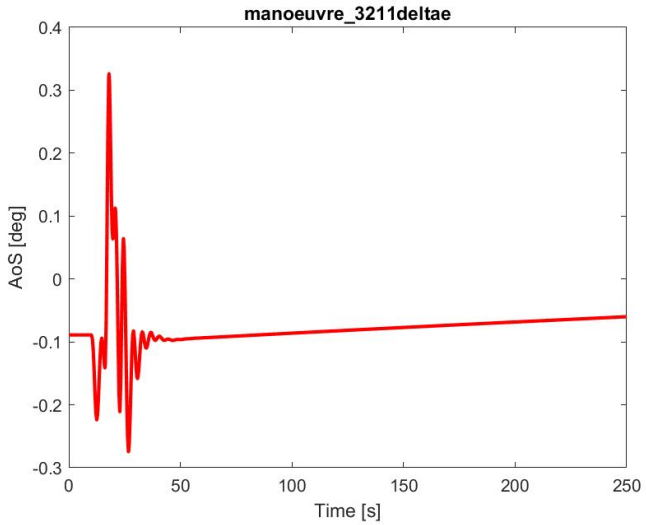
The most performing network, as it is possible to observe from these results, can be identified in **Test 5**. In fact, Test 5 shows low and acceptable values for the 2σ error and the lowest values of the max error. Because of this, for the sake of this work, the neural network trained in Test 5 will be used to conduct further tests on other maneuvers in order to evaluate its response in the prediction of the aerodynamic angles.

4.4 Best Network Performance

The neural network trained in **Test 5** (Fig. 4.28) has proven to be the best performing network when tested with data from maneuver_deltaedoublet_deltardoublet (Fig. 2.1), which is the same maneuver with whom it has been trained. In the section, results obtained with network from Test 5 applied to different maneuvers are presented. The characteristics in terms of Angle of Attack and Angle of Sideslip of the maneuvers are reported in the following figures:

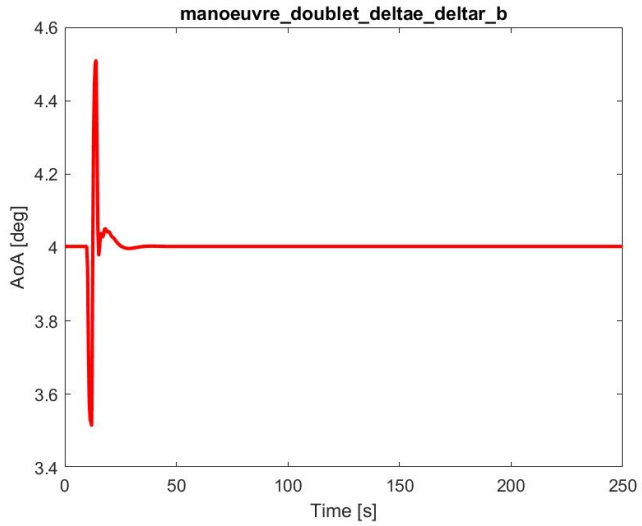


(a) AoA

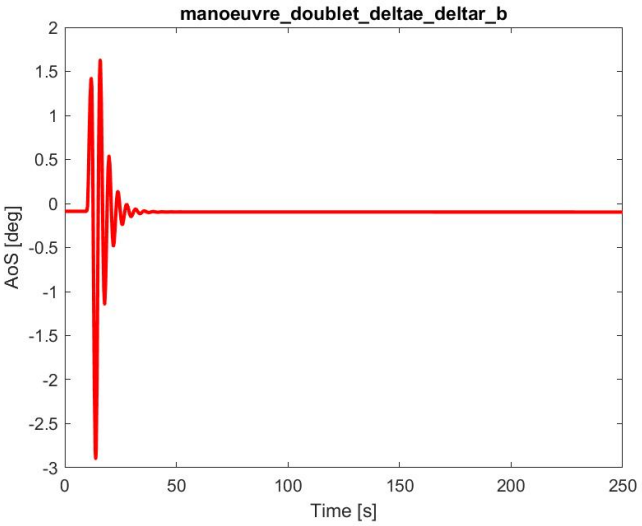


(b) AoS

Figure 4.30: Angle of Attack (a) and Angle of Sideslip (b) of manoeuvre_3211deltae



(a) AoA

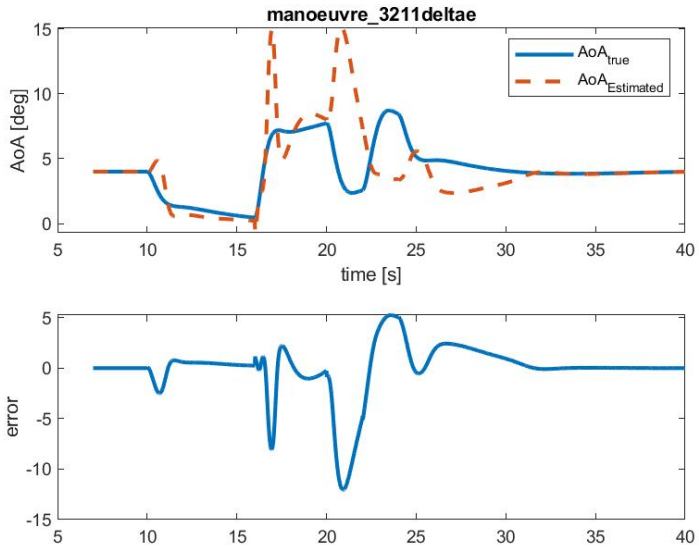


(b) AoS

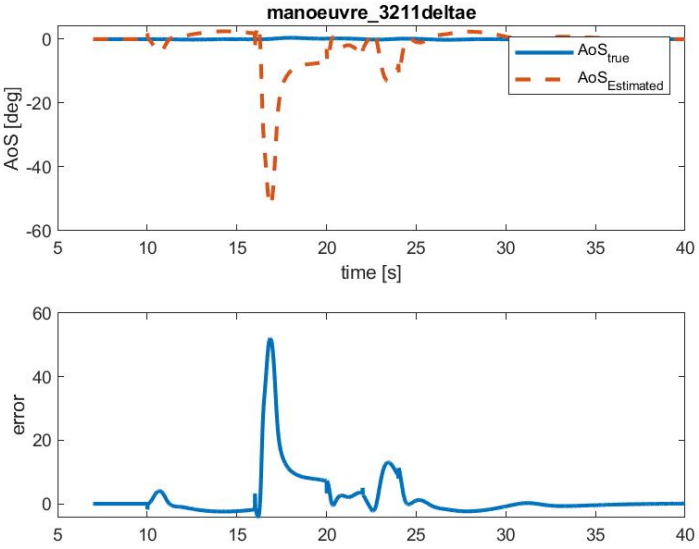
Figure 4.31: Angle of Attack (a) and Angle of Sideslip (b) of manoeuvre_doublet_deltae_deltar_B

Results of the predictions made by the best network (Test 5) are presented in the following subsections.

4.4.1 manoeuvre_3211deltae



(a)



(b)

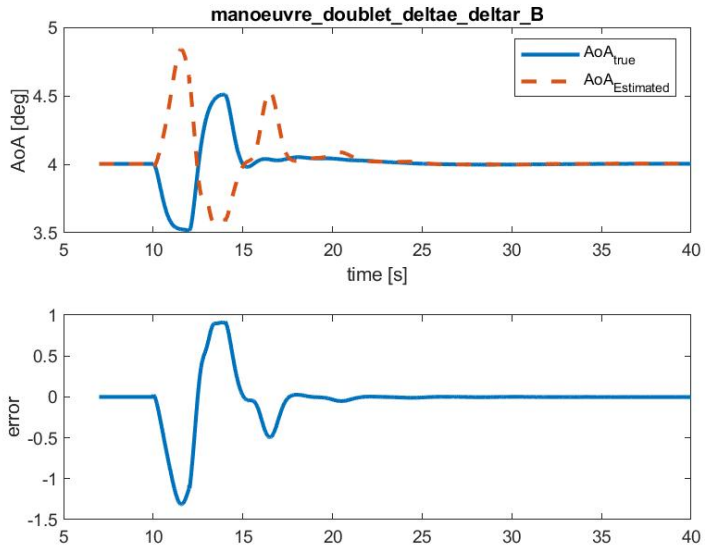
Figure 4.32: Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_3211deltae

3211deltae	2σ error [deg]	mean error [deg]	max error [deg]
AoA	3.2489	-0.1211	12.0505
AoS	9.9768	2.9671	52.1190

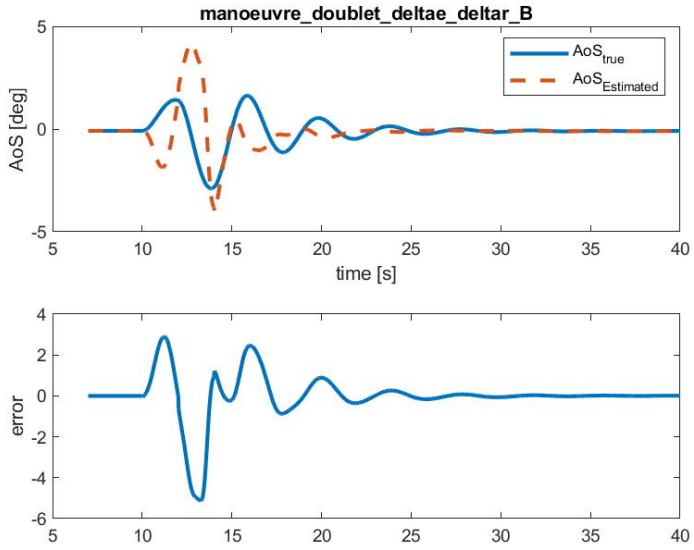
Table 4.21: Statistic data for AoA and AoS of manoeuvre_3211deltae

The solution predicted by the best network when tested with manoeuvre_3211deltae does not satisfy the requirements of the 2σ error for the Angle of Attack and the Angle of Sideslip.

4.4.2 manoeuvre_doublet_deltae_deltar_B



(a)



(b)

Figure 4.33: Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_doublet_deltae_deltar_B

deltae_deltar_B	2σ error [deg]	mean error [deg]	max error [deg]
AoA	1.1054	-0.0719	1.3129
AoS	2.7885	-0.0079	5.1327

Table 4.22: Statistic data for AoA and AoS of manoeuvre_doublet_deltae_ deltar_B

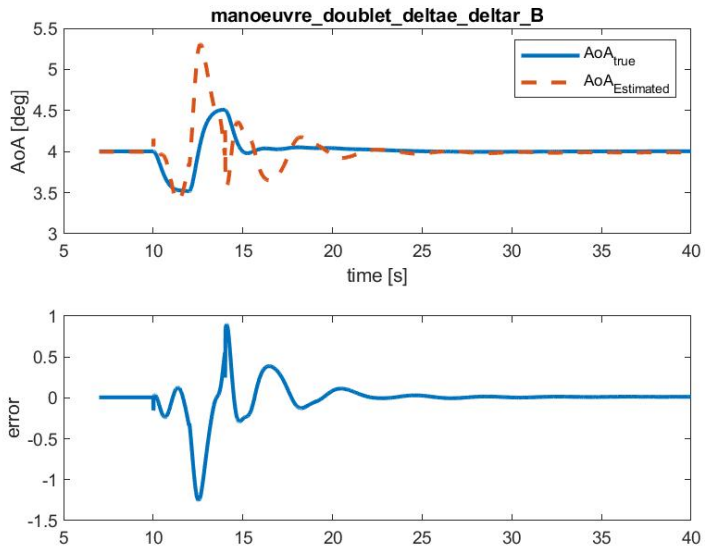
The solution predicted by the best network when tested with manoeuvre_doublet_deltae_deltar_B satisfies the requirements of the 2σ error for the Angle of Attack but not for the Angle of Sideslip.

4.4.3 manoeuvre_doublet_deltae_deltar_B bis

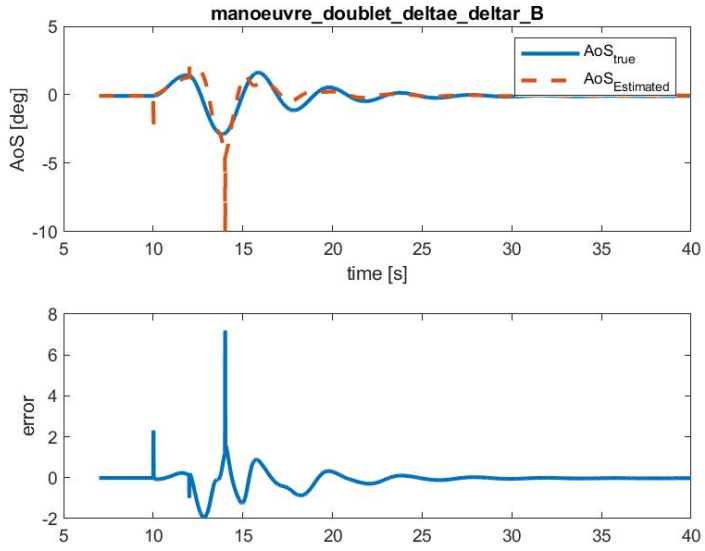
In this subsection maneuver (Fig.4.31) is put to test with a new network trained with manoeuvre_3211deltae (Fig.4.30) and manoeuvre_deltaedoublet_deltardoublet (Fig.2.1). Results are here presented.

training epochs	hidden layers	neurons/layer	t_{step}
300	1	25	30

Table 4.23: New network parameters for testing maneuver deltae_ deltar_B



(a)



(b)

Figure 4.34: Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_doublet_deltae_deltar_B

deltae_deltar_B	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.6509	-0.0159	1.2538
AoS	1.2378	-0.1089	7.2009

Table 4.24: Statistic data for AoA and AoS of manoeuvre_doublet_deltae_ deltar_B

The solution predicted by the new network, trained with manoeuvre_3211deltae and manoeuvre_deltaedoublet_deltardoublet, when tested with manoeuvre_doublet_deltae_deltar_B satisfies the requirements of the 2σ error both for the Angle of Attack and the Angle of Sideslip.

4.5 Sensitivity

Since the network obtained in **Test 5** of the combined tests group (subsection 4.3.5) has been proven as the most accurate network in the prediction of the aerodynamic angles (in the context of the tests done in this work), it has then been tested with data corrupted with white noise. The aim of this test is to check the accuracy of the predictions of the neural network when the provided input data has intrinsic errors due to the white noise. To make this test, the input data provided to the network trained in Test 5 of the combined group has been modified as follows.

Variable	Corruption	measure unit
p	p + randn(size(p))*0.01	deg/s
q	q + randn(size(q))*0.01	deg/s
r	r + randn(size(r))*0.01	deg/s
ax	ax + randn(size(ax))*0.01	m/s^2
ay	ay + randn(size(ay))*0.01	m/s^2
az	az + randn(size(az))*0.01	m/s^2
TAS	TAS + randn(size(TAS))*0.1	m/s
TASp	TASp + randn(size(TASp))*0.1	m/s^2

Table 4.25: Corrupted input data

where $[p, q, r]$ is the vector of the angular velocities and $[ax, ay, az]$ the vector of the components of the body acceleration. The command "randn" is a MatLab command which generate a random value for all the values of the vector chosen (size(.)). The maneuvers tested with the corrupted data are the same maneuvers used in the previous subsection (sec. 4.4). The network characteristics are the one corresponding to Test 5 of subsection 4.3.5 and are summarized in the following tables:

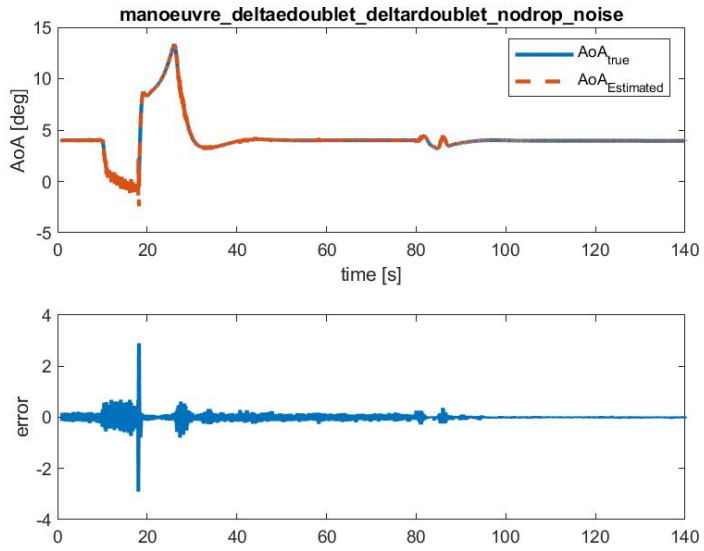
Phase	t_{init}	t_{step}	t_{end}
Test	100	1	length(**time_vect**)-100

Table 4.26: Training and Test data for Sensitivity test

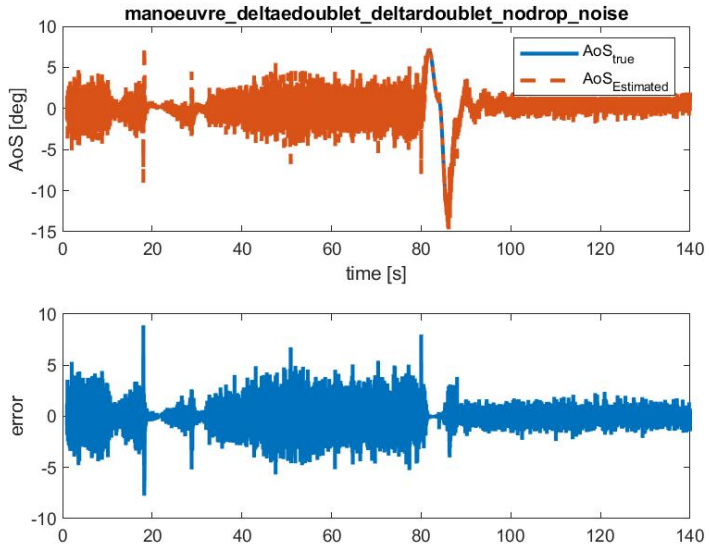
Sensitivity Test	training epochs	hidden layers	neurons/layer	t_{step}
Test 5	500	2	[25 20]	100

Table 4.27: Sensitivity test training characteristics

4.5.1 manoeuvre_deltaedoublet_deltardoublet



(a)



(b)

Figure 4.35: Sensitivity Test: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet

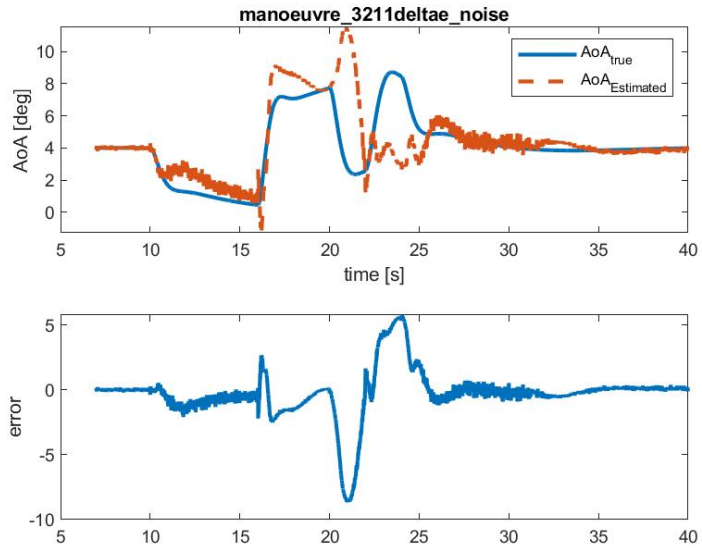
Statistic data are presented in the following table:

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.2021	0.0014	2.9062
AoS	2.5948	-0.0073	8.9006

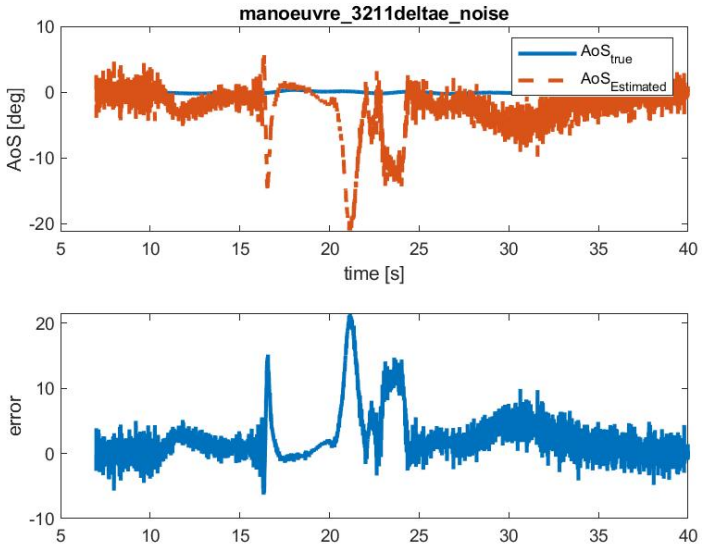
Table 4.28: Sensitivity Test: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet

The results show an oscillatory behaviour in the predictions of the network when provided with corrupted input data. The solution predicted by the network satisfies the requirements of the 2σ error for the Angle of Attack but not for the Angle of Sideslip.

4.5.2 manoeuvre_3211deltae



(a)



(b)

Figure 4.36: Sensitivity Test: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_3211deltae

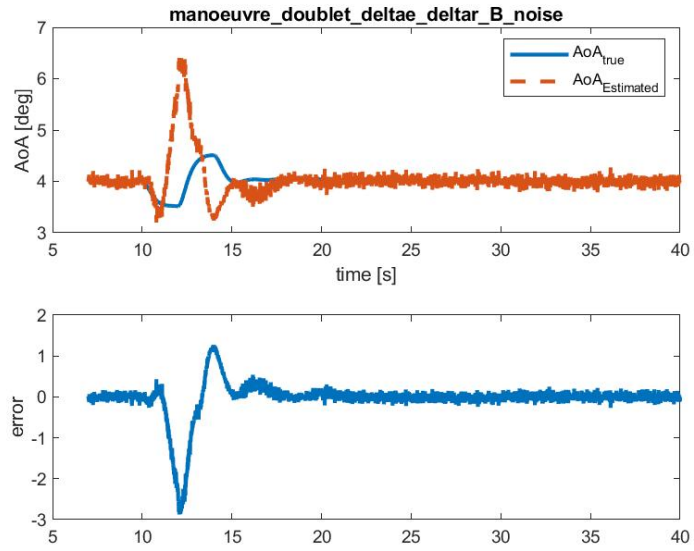
Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	5.3145	-0.2779	8.6605
AoS	11.3197	2.3135	21.3524

Table 4.29: Sensitivity Test: Statistic data for AoA and AoS of manoeuvre_3211deltae

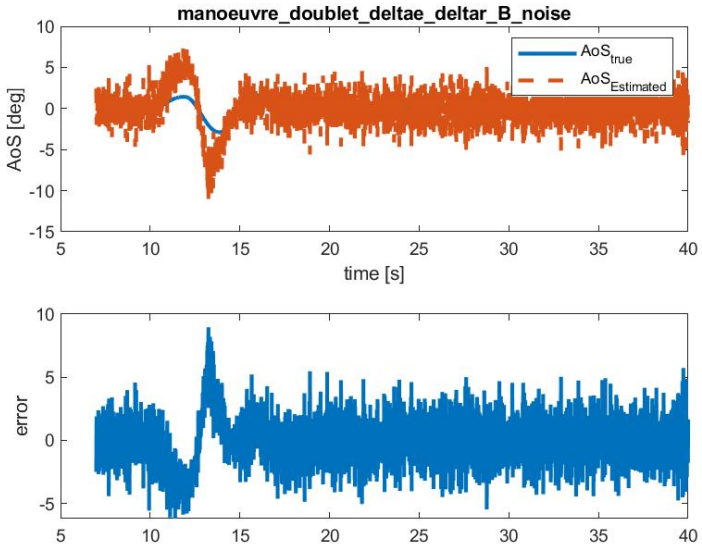
The results show an oscillatory behaviour in the predictions of the network when provided with corrupted input data. The solution predicted by the network does not satisfy the requirements of the 2σ error for the Aerodynamic angles.

4.5.3 manoeuvre_doublet_deltae_deltar_B

The network described in subsection 4.4.3 has been here tested with data of manoeuvre_doublet_deltae_deltar_B corrupted with white noise (tab. 4.25).



(a)



(b)

Figure 4.37: Sensitivity Test: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_doublet_deltae_deltar_B

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	1.1070	-0.0315	2.8758
AoS	4.1259	-0.0632	8.9518

Table 4.30: Sensitivity Test: Statistic data for AoA and AoS of manoeuvre_doublet_deltae_deltar_B

The results show an oscillatory behaviour in the predictions of the network when provided with corrupted input data. The solution predicted by the network satisfies the requirements of the 2σ error for the Angle of Attack but not for the Angle of Sideslip.

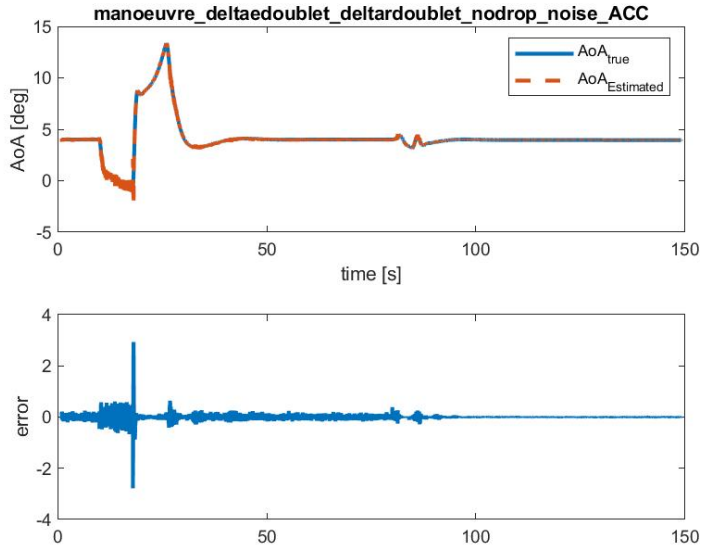
4.5.4 Critical Noisy Data

Since the results with noisy data have proven to be not adequate, other tests have been carried out in order to identify the critical variables which influence the quality of the network predictions the most. In order to achieve this, only a few variables at a time have been corrupted with white noise and used to test the network. Referring to table 4.25, four test have been carried out:

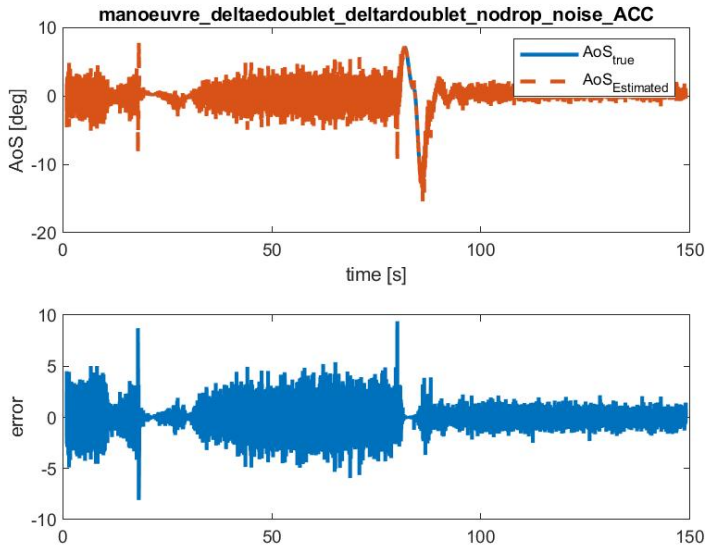
- Corrupted Body Accelerations (a_x, a_y, a_z).
- Corrupted Angular Rates (p, q, r).
- Corrupted True Air Speed (TAS).
- Corrupted Derivative Of True Air Speed (TASp).

This analysis has been carried out on manoeuvre_deltaedoublet_
deltardoublet with the best network obtained in section 4.3. Results are presented in the following pages.

Corrupted Body Accelerations (a_x, a_y, a_z)



(a)



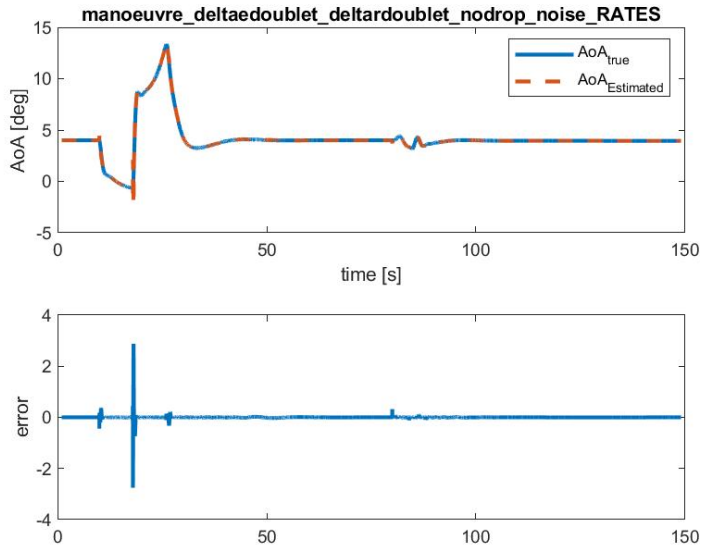
(b)

Figure 4.38: Noisy body accelerations: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet

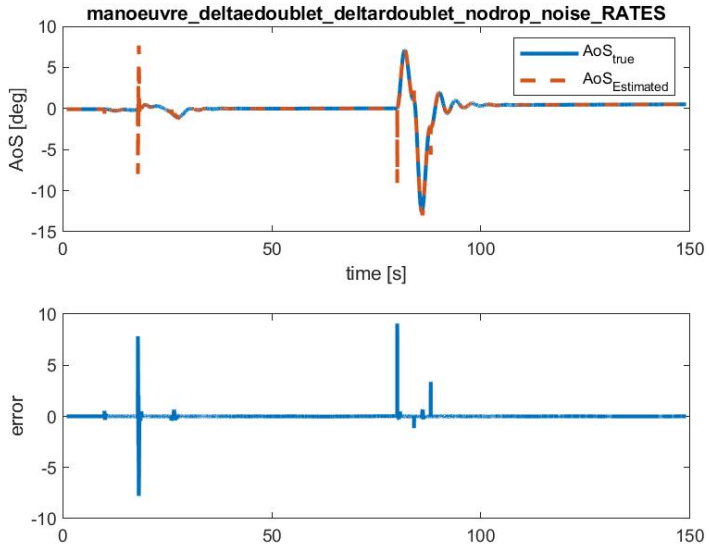
Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.1766	0.0008	3.0637
AoS	2.5906	-0.0008	8.2622

Table 4.31: Noisy body accelerations: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet

Corrupted Angular Rates (p, q, r)



(a)



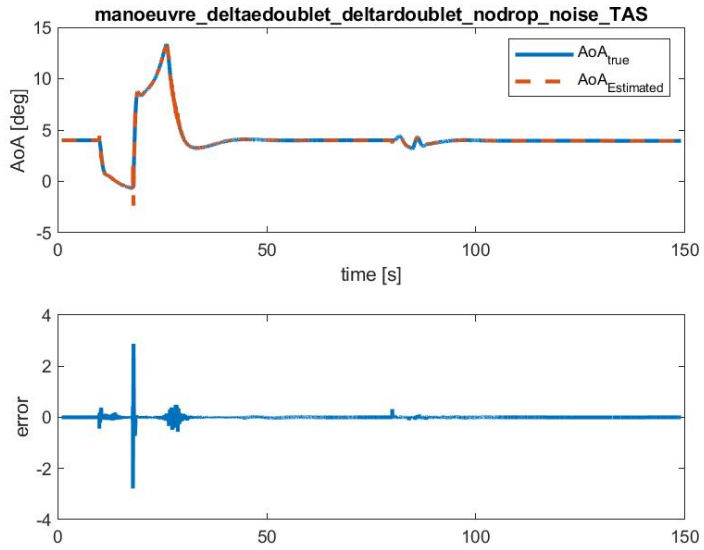
(b)

Figure 4.39: Noisy Angular Rates: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet

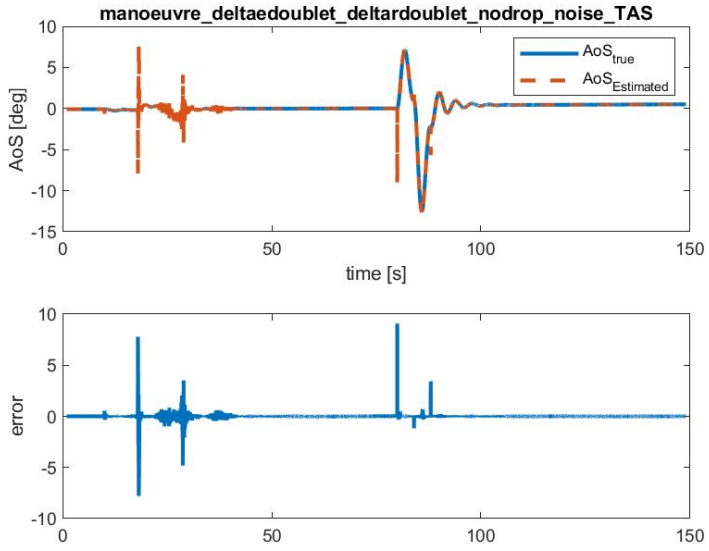
Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.0416	0.0006	2.8779
AoS	0.1291	-0.0016	9.0724

Table 4.32: Noisy Angular Rates: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet

Corrupted True Air Speed (TAS)



(a)



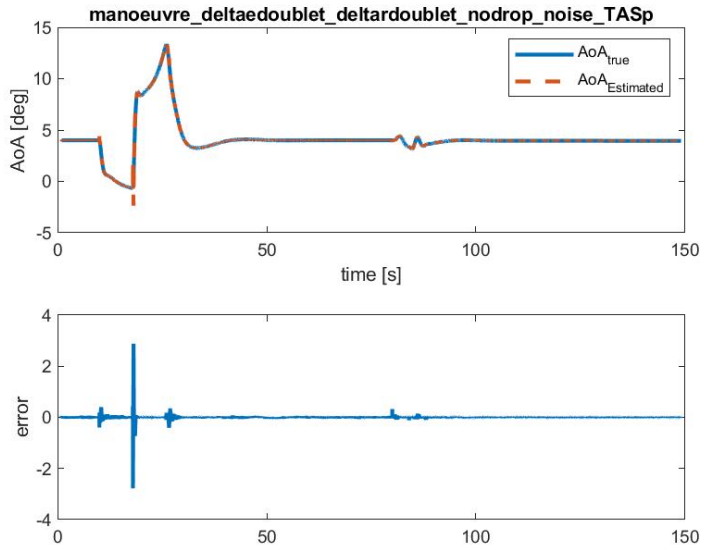
(b)

Figure 4.40: Noisy TAS: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet

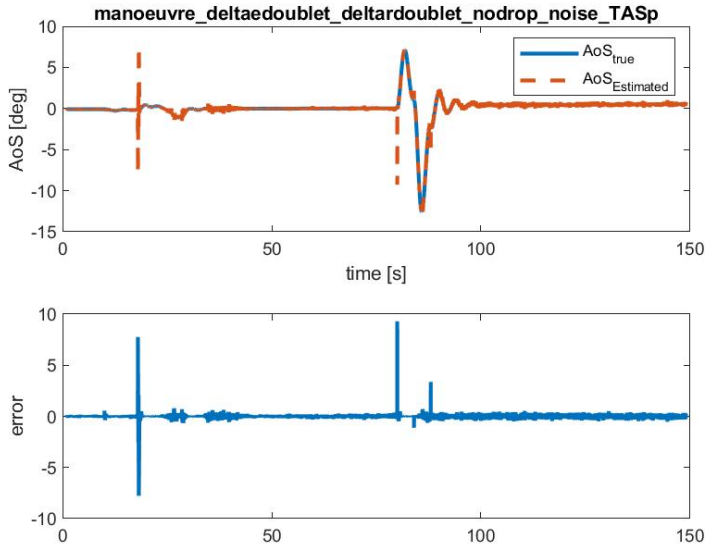
Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.0721	0.0009	2.8781
AoS	0.2331	-0.0012	9.0724

Table 4.33: Noisy TAS: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet

Corrupted True Air Speed Derivative ($TASp$)



(a)



(b)

Figure 4.41: Noisy TASp: Comparison between the true and the network estimated value of AoA (a) and AoS (b) for manoeuvre_deltaedoublet_deltardoublet

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.0711	0.0006	2.8801
AoS	0.3214	-0.0011	8.9902

Table 4.34: Noisy TASp: Statistic data for AoA and AoS of manoeuvre_deltaedoublet_deltardoublet

As it is possible to notice from these results, the body accelerations have proven to be the most critical variables when it comes to the quality of the predictions of the aerodynamic angles, while the angular rates, True Air Speed and it's derivative seem to produce no significant errors.

Chapter 5

Radial Basis Functions Neural Networks

In this chapter neural networks based on Radial Basis Functions, introduced in chapter 3, are shortly described with more theoretical details and are then tested with the Zero Order ASSE Scheme.

5.1 Theoretical Background

While Multi Layer Perceptrons networks under supervised learning are based on stochastic approximation, which means that the network looks for the solution by making statistically optimal choices based on techniques of statistical inference, Radial Basis Functions (RBF) networks are based on a different approach by viewing the design of a neural network as a curve-fitting (approximation) problem in a high-dimensional space. According to this view point, learning is equivalent to finding a surface in a multidimensional space that provides a best fit to the training data, with the criterion for "best fit" being measured in some statistical sense. Correspondingly, generalization is equivalent to the use of this multidimensional surface to interpolate the test data. In the context of a neural network, the hidden units provide a set of "functions" that constitute an arbitrary "basis" for the input patterns (vectors) when they are expanded into the hidden space; these functions are called Radial Basis Functions. The construction of a radial basis function network, in its most basic form, involves three layers with entirely different roles. The input layer is made up of source nodes (sensory units) that connect the network to its environment. The second layer, the only hidden layer in the network, applies a non-linear transformation from the input space to the hidden space; in most applications the hidden space is of high dimensionality. The output layer is linear, supplying the response of the network to the activation pattern (signal) applied to the input layer. A mathematical justification for the rationale of a non-linear transformation followed by a linear transformation may be traced back to an early

paper by Cover (1965). According to this paper, a pattern-classification problem cast in a high-dimensional space is more likely to be linearly separable than in a low dimensional space, hence the reason for frequently making the dimension of the hidden space in an RBF network high. Another important point is the fact that the dimension of the hidden space is directly related to the capacity of the network to approximate a smooth input-output mapping (Mhaskar, 1996; Niyogi and Girosi, 1996); the higher the dimension of the hidden space, the more accurate the approximation will be. When a radial basis function (RBF) network is used to perform a complex pattern-classification task, the problem is basically solved by transforming it into a high-dimensional space in a non-linear manner. The underlying justification is found in Cover's theorem on the separability of patterns, which in qualitative terms, may be stated as follows (Cover, 1965): A complex pattern-classification problem cast in a high-dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space. Further explanations on this theorem can be found in the work of Haykin [2].

5.1.1 Radial Basis Functions

The following description is taken from the work of Simone Giannattasio [18]. A radial basis function is a particular real function ϕ which depends on the distance between an input value x and a fixed value x_i which is defined as the center of the radial function. In the case of multiple input patterns, x is a vector and the distance between x and the center of the radial basis function corresponds to the Euclidean distance:

$$\phi(x) = \phi(||x - \mu||) \tag{5.1}$$

Given two n-dimensional vectors $\mathbf{p}$,$\mathbf{q}$, their euclidean distance is calculated as follows

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{k=1}^{n}(p_k - q_k)^2} \tag{5.2}$$

The distance between the x vector and the center μ can be also intended as the radius of the radial function.

$$r = ||x - \mu|| \tag{5.3}$$

Some of the most common radial basis functions are:

- Gaussian

$$\varphi(r) = e^{(-\varepsilon r)^2} \tag{5.4}$$

- Multiquadric

$$\varphi(r) = \left(1 + (r\varepsilon)^2\right)^{\frac{1}{2}} \tag{5.5}$$

- Tin Plate Spline

$$\varphi(r) = r^2 \log r \tag{5.6}$$

- Inverse Multiquadric

$$\varphi(r) = \frac{1}{(1 + (r\varepsilon)^2)^{\frac{1}{2}}} \tag{5.7}$$

where ε is a shape parameter.

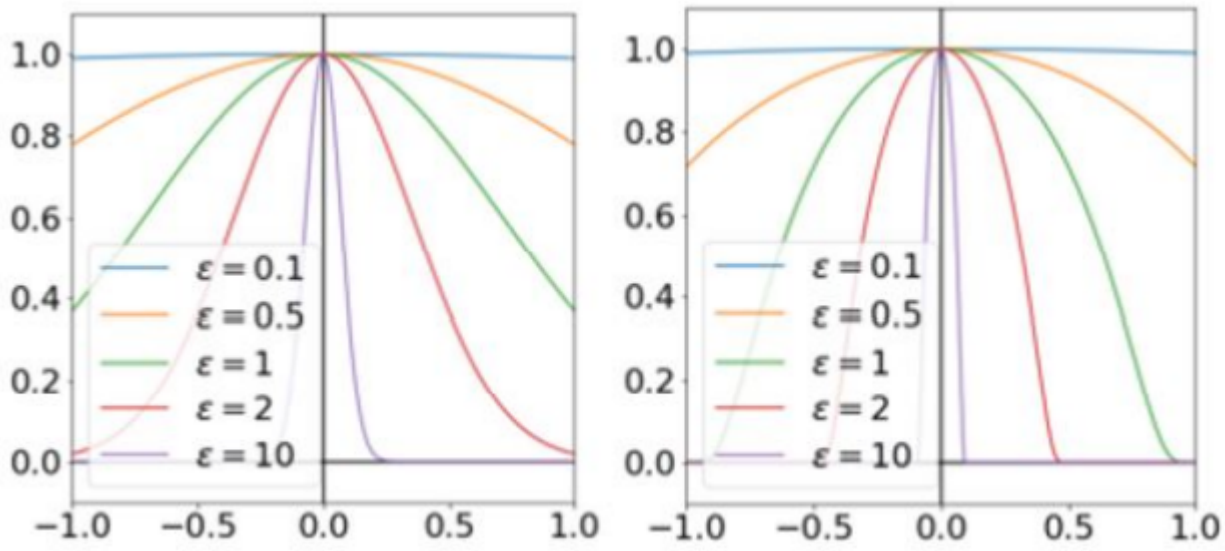


Figure 5.1: Radial Basis Functions varying the parameter ε. Image taken from https://en.wikipedia.org/wiki/Radial_basis_function/media/File:Gaussian_function_shape_parameter.png.
Author: Shawsa7

these functions are usually strictly positive, hence the use of the shape parameter ε. The most used RBF is the Gaussian.

5.1.2 Radial Basis Functions Networks Architecture

The problem of multivariate interpolation in a highly dimensional space has been a subject of many studies. Referring to David's theory of 1963, an exact interpolation occurs when the interpolating surface goes through all the training points. This solution is only theoretical because it implies to have as many radial functions and hidden units as the number of input patterns given to the network. To avoid this problem, an approximate interpolation method will be further presented. Considering $\hat{f}$ as the interpolating function, it follows that:

$$\hat{f}(x_i) = d_i \tag{5.8}$$

where x_i is a generic value of the input vector, and d_i is the corresponding desired output. The interpolation technique of the radial basis functions implies that the estimated output of the network is calculated as a linear combination of the multiplication between the synaptic weights and the radial basis functions (which depends from the radius, hence the distance between the input vector and the center of the radial function). In this approach the input vector data are chosen as the centers of the radial basis functions. However, there are also other criteria for the choice of the RBF centers.

$$\hat{f}(x) = \sum_{i=1}^{n} w_i \phi(||x - x_i||) \tag{5.9}$$

What has been shown so far equals to create a network with a single hidden layer. This network has the purpose of making a non-linear mapping of the input data (inside the hidden layer) by applying the radial basis functions, together with a linear mapping (which consists in multiplying the output of the j^{th} neuron of the hidden layer with a synaptic weight). The radial basis functions are different for each neuron and they are distributed such that they cover the entire input hyperspace.

5.1.3 Learning Process In RBF Networks

As already showed in general for neural networks, even for the RBF networks the learning process is based on minimizing the error, hence the cost function. The parameters that characterize the network learning process are the following:

- Number of neurons inside the hidden layer.
- The center coordinates of each radial basis function inside the hidden layer.
- The radius of every radial basis function in each dimension.

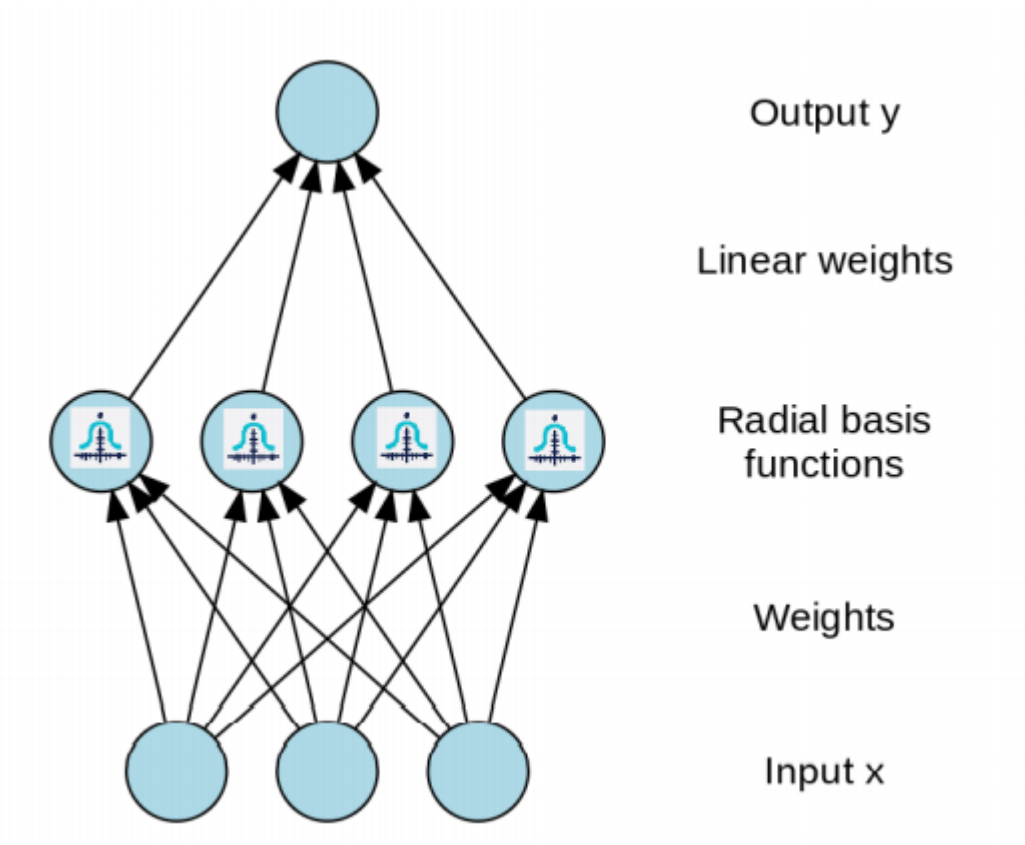


Figure 5.2: Radial Basis Functions Architecture. Image taken from: https://en.wikipedia.org/wiki/Radial_basis_function_network/media/File:Radial_funktion_network.svg. Author: SebDE

- The synaptic weights applied to the outputs of the radial basis function when they transfer to the summation layer.

The RBF can be trained with different techniques. The most common approach is to use an hybrid algorithm which implies a non-supervised learning for the determination of the centers and a supervised learning for determining the optimal synaptic weights of the network. The main steps to follow are:

- To fix the number of hidden neurons.
- To assign center and radius to the radial basis functions of each unit.
- To calculate the optimal synaptic weights (which minimize the cost function).
- To verify the generalization performance of the network and possibly to apply optimization techniques of the architecture.

As far as the non-supervised determination of the centers and variances of the radial functions are concerned, the algorithm usually used is called Kmeans [19]. The purpose is to find the clusters centers first, which are then used as centers for the radial basis functions. Anyway, the clustering K-means has a high computational cost and it often does not generate the optimal number of centers. Another approach is to use a random subset of training data as centers. The computation of the optimal synaptic weights among the neurons of the hidden layer and of the summation layer is done by using the Least Squares or the Ordinary Least Squares algorithm. The number of neurons inside the hidden layer can vary depending on the approach used. The most common choice, which optimizes the network in the best way, consists in using a training algorithm which relies on an evolutionary approach based on a forward or backward selection logic. This methods checks the performance of the network by adding or subtracting a single neuron each time. The process of adding neurons (in the case of a forward logic) stops when the error starts increasing because of the overfitting.
More theory about Radial Basis Functions can be found in the work of Simone Giannattasio [18] and in the book of Haykin [2] and it is not discussed here since it is beyond the purpose of this work.

5.2 RBF Networks Applied To The Zero Order ASSE Scheme

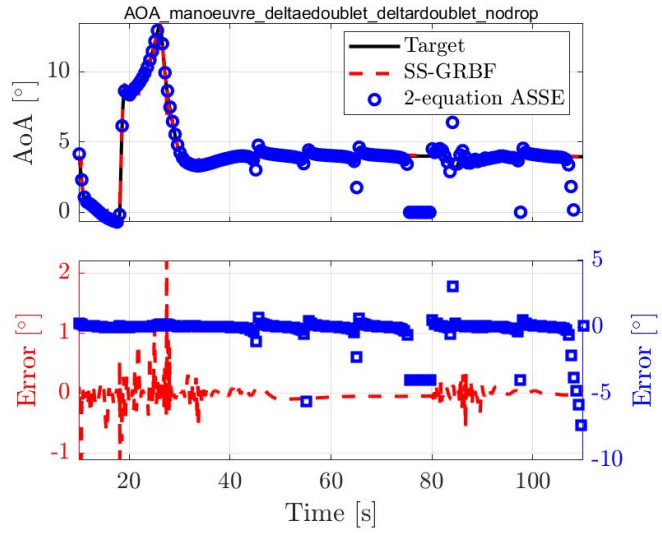
In this section the maneuvers previously tested in chapter 4 with MLP networks are here tested with a GRBF neural network. Results are presented and evaluated as already described for MLP networks in chapter 4. The GRBF network

has been developed with the Deep Learning MatLab Toolbox and it has different characteristics for the estimation of AoA and AoS:

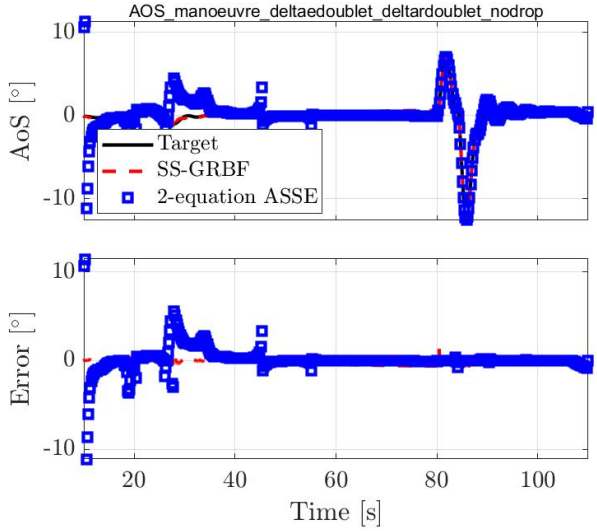
- AoA: number of neurons in the hidden layer is 200
- AoS: number of neurons in the hidden layer is 145

The aim of this chapter is to compare the performance of MLP and GRBF networks, especially in the case of white noise corrupted data. Results are presented in the following subsections:

5.2.1 manoeuvre_deltaedoublet_deltardoublet



(a)



(b)

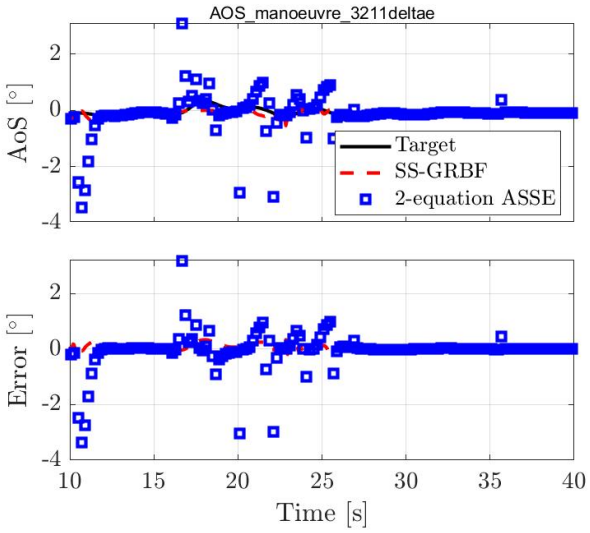
Figure 5.3: GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_deltaedoublet_deltardoublet

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.2743	-0.0221	2.2135
AoS	0.5932	-0.1224	1.3246

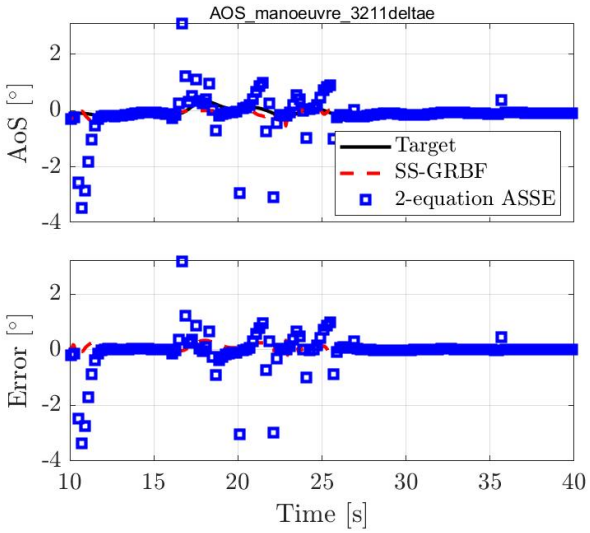
Table 5.1: Statistic data for AoA and AoS of manouver_deltaedoublet_ deltardoublet

The result predicted by the GRBF network satisfies the 2σ requirements both for AoA and AoS.

5.2.2 manoeuvre_3211deltae



(a)



(b)

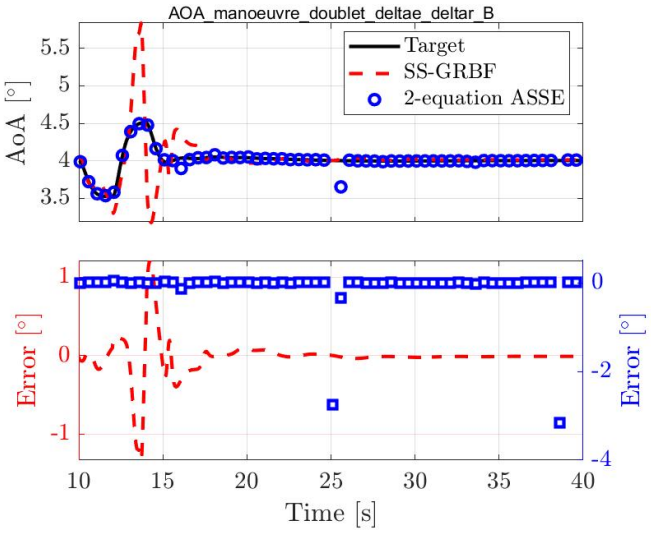
Figure 5.4: GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_3211deltae

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.2411	-0.0513	5.0621
AoS	0.2512	0.0414	0.3422

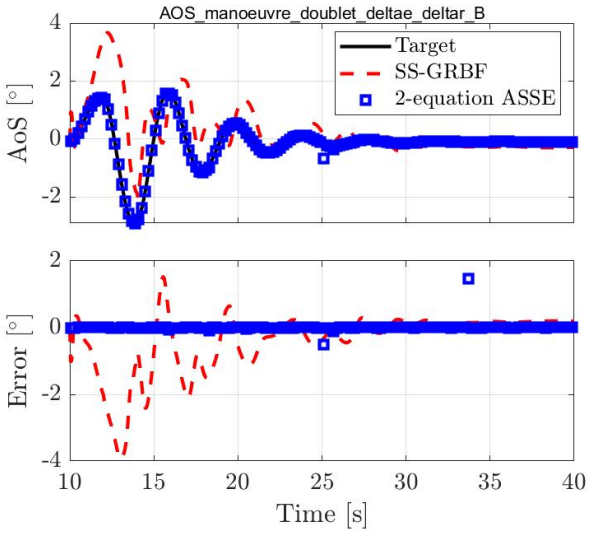
Table 5.2: Statistic data for AoA and AoS of manoeuvre_3211deltae

The result predicted by the GRBF network satisfies the 2σ requirements both for AoA and AoS.

5.2.3 manoeuvre_doublet_deltae_deltar_B



(a)



(b)

Figure 5.5: GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_doublet_deltae_deltar_B

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.4432	-0.0223	1.3331
AoS	2.4211	0.3942	3.9413

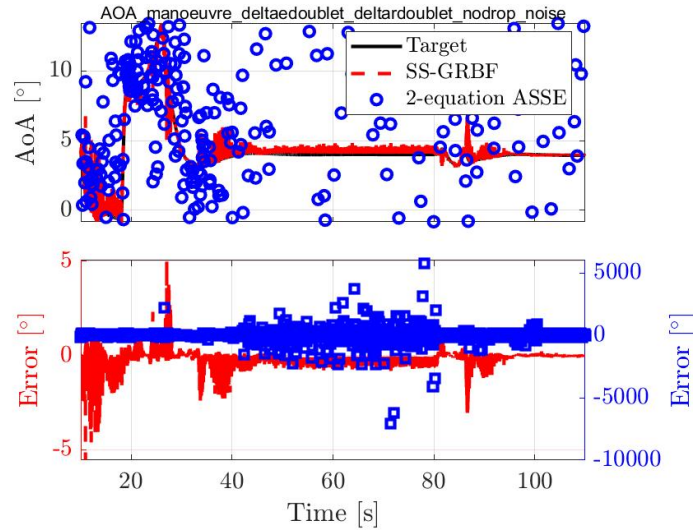
Table 5.3: Statistic data for AoA and AoS of manoeuvre_doublet_ deltae_deltar_B

The result predicted by the GRBF network satisfies the 2σ requirements both for AoA and AoS.

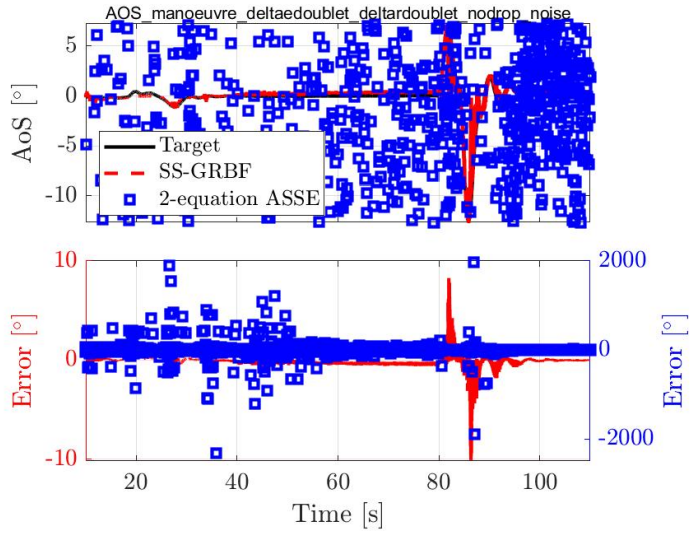
5.3 Sensitivity

In this section the maneuvers presented in the previous section are tested with data corrupted with white noise in order to evaluate the performance of GRBF networks. Test data are corrupted as described in table 4.25. Results are presented in the following subsections.

5.3.1 manoeuvre_deltaedoublet_deltardoublet



(a)



(b)

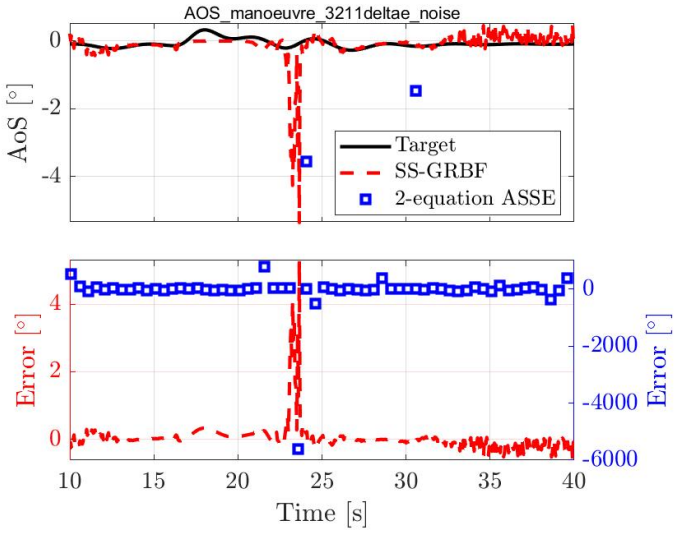
Figure 5.6: GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_deltaedoublet_deltardoublet with white noise

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.8613	-0.1624	5.5516
AoS	0.7746	-0.1136	10.1734

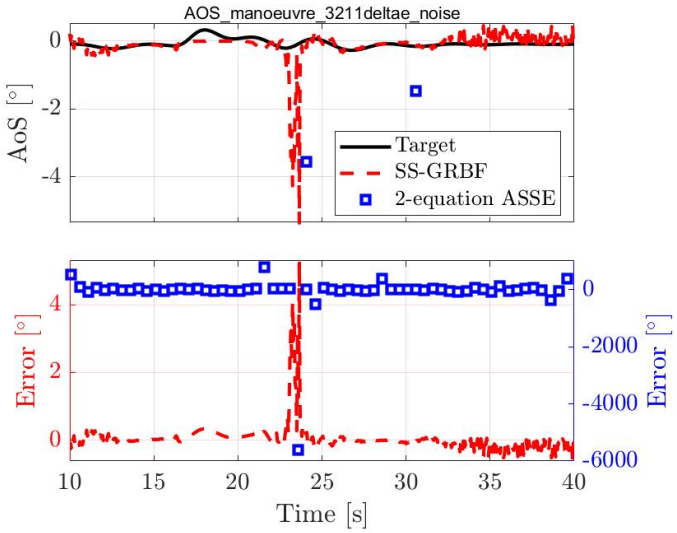
Table 5.4: Statistic data for AoA and AoS of maneuver of manoeuvre_deltaedoublet_deltardoublet with white noise

The result predicted by the GRBF network satisfies the 2σ requirements both for AoA and AoS.

5.3.2 manoeuvre_3211deltae



(a)



(b)

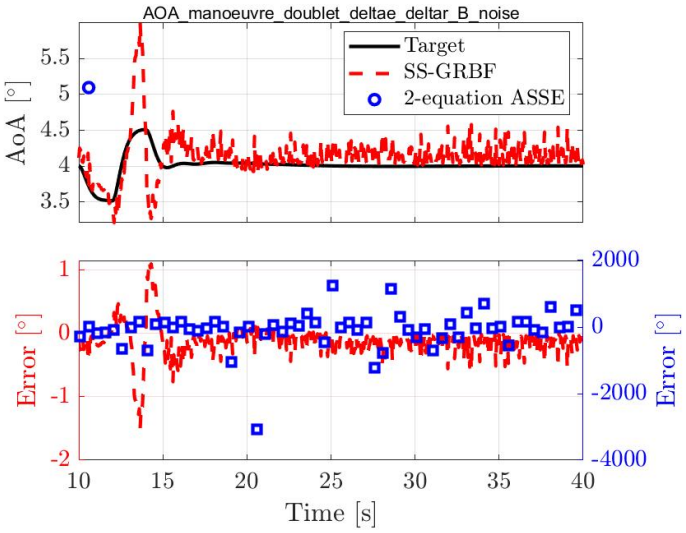
Figure 5.7: GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_3211deltae with white noise

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	1.2232	-0.1264	4.8211
AoS	0.4445	0.0431	5.2912

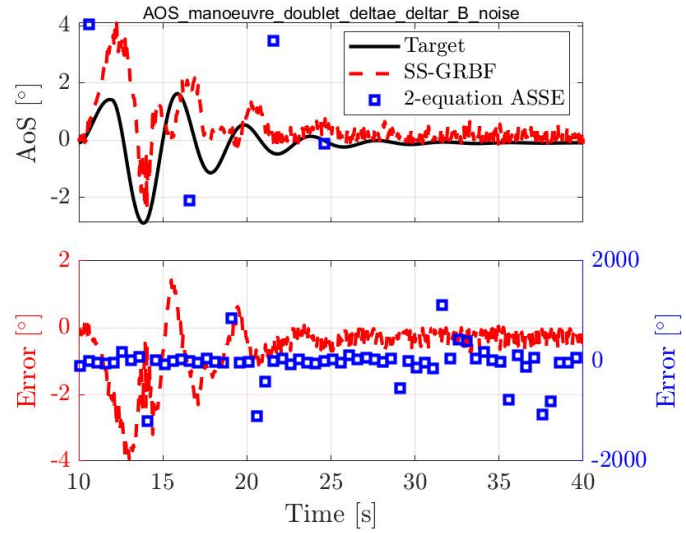
Table 5.5: Statistic data for AoA and AoS of manoeuvre_3211deltae with white noise

The result predicted by the GRBF network satisfies the 2σ requirements both for AoA and AoS.

5.3.3 manoeuvre_doublet_deltae_deltar_B



(a)



(b)

Figure 5.8: GRBF:Comparison between the true and the network estimated value of AoA (a) and AoS (b) of manoeuvre_doublet_deltae_deltar_B with white noise

Aerodynamic Angle	2σ error [deg]	mean error [deg]	max error [deg]
AoA	0.5818	-0.1227	1.4921
AoS	2.4453	-0.6047	3.9435

Table 5.6: Statistic data for AoA and AoS of manoeuvre_doublet_deltae_ deltar_B with white noise

The result predicted by the GRBF network satisfies the 2σ requirements both for AoA and AoS.

Conclusions

A summary table with the results obtained both with MLP and RBF networks for test maneuvers: maneuver_3211deltae, maneuver_doublet_deltae_deltar_B is here presented.

CLEAR DATA

		maneuver_3211deltae	
		MLP	
	AoA		AoS
2σ error [deg]	3.2489		9.9768
mean error [deg]	-0.1211		2.9671
max error [deg]	12.0505		52.1190
		RBF	
	AoA		AoS
2σ error [deg]	0.2411		0.2512
mean error [deg]	-0.0513		0.0414
max error [deg]	5.0621		0.3422
		maneuver_doublet_deltae_deltar_B	
		MLP	
	AoA		AoS
2σ error [deg]	1.1054		2.7885
mean error [deg]	-0.0719		-0.0079
max error [deg]	1.3129		5.1327
		RBF	
	AoA		AoS
2σ error [deg]	0.4432		2.4211
mean error [deg]	-0.0223		0.3942
max error [deg]	1.3331		3.9413

Table 5.7: Clear Data: Summary of the results for test maneuvers both for MLP and RBF network

NOISY DATA

		maneuver_3211deltae	
		MLP	
	AoA		AoS
2σ error [deg]	5.3145		11.3197
mean error [deg]	-0.2779		2.3135
max error [deg]	8.6605		21.3524
		RBF	
	AoA		AoS
2σ error [deg]	1.2232		0.4445
mean error [deg]	-0.1264		0.0431
max error [deg]	4.8211		5.2912
		maneuver_doublet_deltae_deltar_B	
		MLP	
	AoA		AoS
2σ error [deg]	1.1070		4.1259
mean error [deg]	-0.0315		-0.0632
max error [deg]	2.8758		8.9518
		RBF	
	AoA		AoS
2σ error [deg]	0.5818		2.4453
mean error [deg]	-0.1227		-0.6047
max error [deg]	1.4921		3.9435

Table 5.8: Noisy Data: Summary of the results for test maneuvers both for MLP and RBF network

As it is possible to notice from these results, radial basis function networks seem to produce more accurate results in terms of the 2σ error, according to [15], compared to the multi layer perceptron. Both the MLP and RBF network have been trained with the training maneuver deltaedoublet_deltardoublet (Fig. 2.1). Furthermore, it is also possible to observe that the RBF network seems to be more tolerant to white noise input data since the the 2σ does not increase as much as it does for the MLP network.

Appendix

Linear ASSE Scheme Script

```
clear all
close all
clc

filename='manoeuvre_deltaedoublet_deltardoublet_nodrop.mat'
load(filename)


t_init=sync.time(2);
t_step=0.001;
t_end=sync.time(end);


time_vect = t_init:t_step:t_end;
np=length(time_vect);


H1=0.*time_vect;
L1=H1;
M1=H1;
N1=H1;
H2=H1;
L2=H1;
M2=H1;
N2=H1;
```

```
%% COEFFICIENTS COMPUTATION

for ii=2:np

t = ii %index of the variables of the first equation, subscript "t"
tau = t - 1; %index of the variables of the second equation,
                    %subscript "tau"


% EQUATION 1
I=eye(3);


a_b1=[sync.ax(t) sync.ay(t) sync.az(t)]';
Vinf1=sync.TAS(t);
Vinfdot1=(sync.u(t)*sync.udot(t) + sync.v(t)*sync.vdot(t)...
                +sync. w(t)*sync.wdot(t)) / Vinf1;


% OMEGA
OM_B1=[0 -sync.r(t) sync.q(t); sync.r(t) 0 -sync.p(t);...
               -sync.q(t) sync.p(t) 0 ];


dt1=0;


% coeff ASSE
N1(tau) = Vinf1 * Vinfdot1/Vinf1 ;
m1 = Vinf1 * (I-OM_B1*dt1) * a_b1/Vinf1 ;


H1(tau) = m1(1);
L1(tau) = m1(2);
M1(tau) = m1(3);

% EQUATION 2
I= eye(3);
a_b2 = [sync.ax(tau) sync.ay(tau) sync.az(tau)]';
```

```
Vinf2 = sync.TAS(tau);
Vinfdot2 = (sync.u(tau)*sync.udot(tau) + sync.v(tau)*sync.vdot(tau)...
            +sync. w(tau)*sync.wdot(tau)) / Vinf2;


% OMEGA (tau)
OM_B2=[0 -sync.r(tau) sync.q(tau); sync.r(tau) 0 -sync.p(tau);...
          -sync.q(tau) sync.p(tau) 0 ];


dt2 = sync.time(t)-sync.time(tau);


ix = trapz( sync.time(tau:t), sync.ax(tau:t), 1);
iy = trapz( sync.time(tau:t), sync.ay(tau:t), 1);
iz = trapz( sync.time(tau:t), sync.az(tau:t), 1);


int_a_b=[ix iy iz];


N2(tau) = ( Vinf2*Vinfdot2 + int_a_b*a_b2)/Vinf1;
m2 = Vinf1 * (I - OM_B1*dt2)*a_b2/Vinf1;


H2(tau) = m2(1);
L2(tau) = m2(2);
M2(tau) = m2(3);


end



%% LINEAR SYSTEM SOLUTION


np2=length(H1);
```

```
for ii=1:np2


ii


A=[L1(ii) M1(ii);L2(ii) M2(ii)];
B=[N1(ii)-H1(ii);N2(ii)-H2(ii)];
x=A\B;


x_prova=[sync.beta(ii)*pi/180,sync.alpha(ii)*pi/180];
difff=(x-x_prova).*180/pi


Beta(ii)=x(1)*180/pi;
Alfa(ii)=x(2)*180/pi;


Alfa_max = 40;
if abs(Alfa(ii)) > Alfa_max
Alfa(ii) = NaN;
end


Beta_max = 40;
if abs(Beta(ii)) > Bet_max
Beta(ii) = NaN;
end


AoA_err(ii)=Alfa(ii)-sync.alpha(ii);
AoS_err(ii)=Beta(ii)-sync.beta(ii);


end
```

```
figure(1)
plot(sync.time(1:t),Alfa(1:end),'co')
hold on
plot(sync.time(1:t),sync.alpha(1:t),'r')
title('AoA')
legend('AoA_{LinEstimated}','AoA_{TRUE}')


figure(2)
plot(sync.time(1:t),Beta(1:end),'co')
hold on
plot(sync.time(1:t),sync.beta(1:t),'r')
title('AoS')
legend('AoS_{LinEstimated}','AoS_{TRUE}')


figure(3)
plot(sync.time(1:t),AoA_err,'r')
title('AoA error deg')


figure(4)
plot(sync.time(1:t),AoS_err,'r')
title('AoS error deg')
```

Non-Linear ASSE Scheme Script

```
clear all
close all
clc
```

```
filename='manoeuvre_deltaedoublet_deltardoublet_nodrop.mat'
load(filename)


t_init=sync.time(2);
t_step=0.001;
t_end=sync.time(end);


time_vect = t_init:t_step:t_end;
np=length(time_vect);


H1=0.*time_vect;
L1=H1;
M1=H1;
N1=H1;
H2=H1;
L2=H1;
M2=H1;
N2=H1;


%% COEFFICIENTS COMPUTATION

for ii=2:np

t = ii %index of the variables of the first equation, subscript "t"
tau = t - 1; %index of the variables of the second equation,
                    %subscript "tau"


% EQUATION 1
I=eye(3);


a_b1=[sync.ax(t) sync.ay(t) sync.az(t)]';
Vinf1=sync.TAS(t);
Vinfdot1=(sync.u(t)*sync.udot(t) + sync.v(t)*sync.vdot(t)...
```

```
                +sync. w(t)*sync.wdot(t)) / Vinf1;


% OMEGA
OM_B1=[0 -sync.r(t) sync.q(t); sync.r(t) 0 -sync.p(t);...
              -sync.q(t) sync.p(t) 0 ];


dt1=0;


% coeff ASSE
N1(tau) = Vinf1 * Vinfdot1/Vinf1 ;
m1 = Vinf1 * (I-OM_B1*dt1) * a_b1/Vinf1 ;


H1(tau) = m1(1);
L1(tau) = m1(2);
M1(tau) = m1(3);

% EQUATION 2
I= eye(3);
a_b2 = [sync.ax(tau) sync.ay(tau) sync.az(tau)]';
Vinf2 = sync.TAS(tau);
Vinfdot2 = (sync.u(tau)*sync.udot(tau) + sync.v(tau)*sync.vdot(tau)...
              +sync. w(tau)*sync.wdot(tau)) / Vinf2;


% OMEGA (tau)
OM_B2=[0 -sync.r(tau) sync.q(tau); sync.r(tau) 0 -sync.p(tau);...
              -sync.q(tau) sync.p(tau) 0 ];


dt2 = sync.time(t)-sync.time(tau);


ix = trapz( sync.time(tau:t), sync.ax(tau:t), 1);
iy = trapz( sync.time(tau:t), sync.ay(tau:t), 1);
iz = trapz( sync.time(tau:t), sync.az(tau:t), 1);
```

```
int_a_b=[ix iy iz];


N2(tau) = ( Vinf2*Vinfdot2 + int_a_b*a_b2)/Vinf1;
m2 = Vinf1 * (I - OM_B1*dt2)*a_b2/Vinf1;


H2(tau) = m2(1);
L2(tau) = m2(2);
M2(tau) = m2(3);


end



%% NON LINEAR SYSTEM SOLUTION


np2=length(H1);
x0=[sync.beta(1)*pi/180,sync.alpha(1)*pi/180];


for ii=1:np2


ii


H=[H1(ii);H2(ii)];
L=[L1(ii);L2(ii)];
M=[M1(ii);M2(ii)];
N=[N1(ii);N2(ii)];


asse=@(x)...
        [N-H.*cos(x(1)).*cos(x(2))-L.*sin(x(1))-M.*cos(x(1)).*sin(x(2))];
```

```
options =optimoptions(...
                'fsolve','Algorithm','levenberg-marquardt',...
                'Display','none','SpecifyObjectiveGradient',false,...
                'OptimalityTolerance',1E-10,'StepTolerance', 1e-10,...
                'MaxIterations', 1000,'MaxFunctionEvaluations', 1000);


x=fsolve(asse,x0,options);


x_prova=[sync.beta(ii)*pi/180,sync.alpha(ii)*pi/180];
difff=(x-x_prova).*180/pi


Beta(ii)=x(1)*180/pi;
Alfa(ii)=x(2)*180/pi;


AoA_err(ii)=Alfa(ii)-sync.alpha(ii);
AoS_err(ii)=Beta(ii)-sync.beta(ii);


x0=x;


end



figure(1)
plot(sync.time(1:t),Alfa(1:end),'co')
hold on
plot(sync.time(1:t),sync.alpha(1:t),'r')
title('AoA')
legend('AoA_{NonLinEstimated}','AoA_{TRUE}')
```

```
figure(2)
plot(sync.time(1:t),Beta(1:end),'co')
hold on
plot(sync.time(1:t),sync.beta(1:t),'r')
title('AoS')
legend('AoS_{NonLinEstimated}','AoS_{TRUE}')



figure(3)
plot(sync.time(1:t),AoA_err,'r')
title('AoA error deg')



figure(4)
plot(sync.time(1:t),AoS_err,'r')
title('AoS error deg')
```

Bibliography

[1] A. Lerro, A. Brandl, and P. Gili, "Model-free scheme for angle-of-attack and angle-of-sideslip estimation,"
Journal of Guidance, Control, and Dynamics, 2020.

[2] S. Haykin, "A comprehensive foundation," Neural networks, vol. 2, no. 2004, p. 41, 2004.

[3] A. Lerro, P. Gili, and M. S. Caselle, "Development and evaluation of neural network-based virtual air data sensor for estimation of aerodynamic angles," Politecnico di Torino, 2012.

[4] W. L. Ikard, An Air-Flow-Direction Pickup Suitable for Telemetering Use on Pilotless Aircraft. National Advisory Committee for Aeronautics, 1956.

[5] S. Chue, "Pressure probes for fluid measurement," Progress in aerospace sciences, vol. 16, no. 2, pp. 147–223, 1975.

[6] R. C. Pankhurst and D. W. Holder, Wind-tunnel technique: an account of experimental methods in low-and high-speed wind tunnels. Pitman, 1952.

[7] K. Yajnik and R. Gupta, "A new probe for measurement of velocity and flow direction in separated flows," Journal of Physics E: Scientific Instruments, vol. 6, no. 1, p. 82, 1973.

[8] K. A. Wise, "Computational air data system for angle-of-attack and angle-of-sideslip," Aug. 9 2005. US Patent 6,928,341.

[9] T. J. Rohloff, S. A. Whitmore, and I. Catton, "Air data sensing from surface pressure measurements using a neural network method," AIAA journal, vol. 36, no. 11, pp. 2094–2101, 1998.

[10] I. Samy, I. Postlethwaite, D.-W. Gu, and J. Green, "Neural-network-based flush air data sensing system demonstrated on a mini air vehicle," Journal of aircraft, vol. 47, no. 1, pp. 18–31, 2010.

[11] B. K, "Inertial navigation systems analysis,"
Wiley Canada,Norwood, 1971.

[12] O. Salychev, "Applied inertial navigation:problems and solutions," BMSTU Press, Moscow, Russia, 2004.

[13] R. C. Nelson et al., Flight stability and automatic control, vol. 2. WCB/McGraw Hill New York, 1998.

[14] K. Sun, C. D. Regan, and D. G. Egziabher, "Gnss/ins based estimation of air data and wind vector using flight maneuvers," in 2018 IEEE/ION Position, Location and Navigation Symposium (PLANS), pp. 838–849, IEEE, 2018.

[15] A. Lerro, A. Brandl, M. Battipede, and P. Gili, "Preliminary design of a model-free synthetic sensor for aerodynamic angle estimation for commercial aviation," Sensors, vol. 19, no. 23, 2019.

[16] M. I. Lourakis et al., "A brief description of the levenberg-marquardt algorithm implemented by levmar," Foundation of Research and Technology, vol. 4, no. 1, pp. 1–6, 2005.

[17] L. R. Medsker and L. Jain, "Recurrent neural networks," Design and Applications, vol. 5, 2001.

[18] S. Giannattasio, "Sviluppo di un sensore sintetico per la stima dell'angolo d'attacco tramite rete neurale generalized radial basis function. = development of a synthetic sensor for the estimation of the angle of attack using the generalized radial basis function neural network.," 2020.

[19] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," Pattern recognition, vol. 36, no. 2, pp. 451–461, 2003.