Fault Tolerant Flight Control: An Application of the Fully Connected Cascade Neural Network

by

Saed Hussain

A thesis submitted in partial fulfilment for the requirements for the degree of Doctor of Philosophy at the University of Central Lancashire.

March 2015

Dedicated to my loving parents and beautiful sisters.

Declaration

I declare that whilst registered as a candidate for the research degree, I have not been a registered candidate or enrolled student for another award of the University or other academic or professional institution. No material contained in this thesis has been used in any other submission for an academic award and is solely my own work.

Saed Hussain

March 2015

"If we knew what we were doing, it wouldn't be called research."
– Albert Einstein

Abstract

The endurance of an aircraft can be increased in the presence of failures by utilising flight control systems that are tolerant to failures. Such systems are known as fault tolerant flight control systems (FTFCS). FTFCS can be implemented by developing failure detection, identification and accommodation (FDIA) schemes. Two of the major types of failures in an aircraft system are the sensor and actuator failures. In this research, a sensor failure detection, identification and accommodation (SFDIA); and an actuator failure detection, identification and accommodation (AFDIA) schemes are developed. These schemes are developed using the artificial neural network (ANN).

A number of techniques can be found in the literature that address FDIA in aircraft systems. These techniques are, for example, Kalman filters, fuzzy logic and ANN. This research uses the fully connected cascade (FCC) neural network (NN) for the development of the SFDIA and AFDIA schemes. Based on the study presented in the literature, this NN architecture is compact and efficient in comparison to the multi-layer perceptron (MLP) NN, which is a popular choice for NN applications. This is the first reported instance of the use of the FCC NN for fault tolerance applications, especially in the aerospace domain.

For this research, the X-Plane 9 flight simulator is used for data collection and as a test bed. This simulator is well known for its realistic simulations and is certified by the Federal Aviation Administration (FAA) for pilot training. The developed SFDIA scheme adds endurance to an aircraft in the presence of failures in the aircraft pitch, roll and yaw rate gyro sensors. The SFDIA scheme is able to replace a faulty gyro sensor with a FCC NN based estimate, with as few as 2 neurons. In total, 105 failure experiments were conducted, out of which only 1 went undetected.

In the developed AFDIA scheme, a FCC NN based roll controller is employed,

which uses just 5 neurons. This controller can adapt on-line to the post failure dynamics of the aircraft following a 66% loss of wing surface. With 66% of the wing surface missing, the NN based roll controller is able to maintain flight. This is a remarkable display of endurance by the AFDIA scheme, following such a severe failure. The results presented in this research validate the use of FCC NNs for SFDIA and AFDIA applications.

Acknowledgement

I remember at the beginning of this research, someone once told me - "*It's over before you know it!*". As I write these words, after 3 years of this journey, this is how it feels. I have experienced lots of new things, faced different challenges, learnt more about myself, made new friends and of course, did my research. Here is my attempt to acknowledge those who have influenced me to make this journey and helped me through it. This is not meant to be an exhaustive list. My sincere apologies to anyone I have missed and thank you for everything.

Firstly, I would like to thank my director of studies, Prof. Joe Howe, for guiding me through the PhD process. Your constant motivation, support and advice on personal development is greatly appreciated. I would also like to express my deepest gratitude to my supervisor, Dr. Maizura Mokhtar, for her incredible support and constant motivation. You have done more than what anyone could ask of their supervisor. Your kindness and moral support during various stages of this research, is something that I will always remember.

Additionally, this research would not have been possible without the backing of BAE Military, Air and Information (MAI). I would like to thank the wonderful personnel at MAI who took special interest in my work and guided me at various stages. In particular, I would like to thank Ian Harrington for all the support he has provided me when I needed something from BAE. I greatly appreciate everything you have done for me.

I decided to do this PhD at UCLan, due to the amazing experience I had in the final year of my BEng, doing a double module research project. I owe a lot to every member of the engineering department, who have constantly guided and supported me throughout my BEng. I would specially like to thank Phil Tranter and Dr. Geoff Hall for going beyond the role of a supervisor and taking personal interest in my life

and well being. Both of you were always supportive of me, not just in the aspect of my degree but also in terms of my personal and future ambitions. I will always remember you both. Additionally, I would also like to thank Clare Altham from the Graduate School Office, for her kindness and enthusiastic support. Clare, I cannot thank you enough for every time you have helped me and especially towards the end of my research.

Throughout my journey, I have been blessed to have come across many supportive and encouraging guides in my life, particularly a select few teachers to whom I am indebted. Some of which include: Mr. Martin, Mrs. Lobo, Mrs. Kavita, Mr. Deepak and Mr. Sanjay. I cannot thank you enough, you have been so greatly influential in shaping my character, without you this would not have been possible.

Aside from fantastic guides, I have been privileged to have many sincere and fun friends and colleagues. Particular mention must go to Ammar Majeed, Ajmal Ashraf, Ekta Gupte, Rakib Hossain, Mohiuddin Rahman, Clemence Pisira, Aidan Day, Larry Lei, Sarah Cherhabil, Funsho Adeolu, Kostas Papouis, Matt Timperley, Vanaja Rao, Swati Kumar, Sneha Subramanian, Nikhil Mahajan, Temba Mudariki and Rashed Alghafri. Matt, I must say, I am going to miss all the moments we created jotting down crazy ideas on the white board, and the times we were, for a lack of a better word, "*lazy-busy*". Dr. Adam Bedford, thank you for being a good friend and inspiration for crazy ideas. Furthermore, I would specially like to thank Saanya Sequeira for the times she was there and for her support.

Life is an interesting journey, filled with ups and downs. Sometimes, its during the downs, when you really need a good friend to help you walk out of it. I have been fortunate enough to cross paths with Dr. Banu Abdallah, Iftikhar Khan, Sakib Yousaf and Sasitaran Iyavoo. I am immensely grateful to have the privilege to call you my friends. Without you guys, I would have never completed this thesis. Your patience, support and constant motivation during my weakest moment, gave me the strength to keep moving forward with my dreams and ambition. I would also like to thank UCLan ISOC for giving me the opportunity to make great friends who made me laugh during tough times. A special thank you to Yousuf Bhikha, Tasneem

Vahed, Sana Iqbal, Aalisha Azam and Jowairiyah Shibli (Juju).

Finally, and most importantly, I would not be here doing this research without the constant support and sacrifice of my family. I will always remember the hardships and challenges we as a family had to face to get me to this point. Ammu and Abbu, I cannot express in words how grateful I am for everything you have done for me. Asma, Wafa and Rahma, I am lucky to have you lovely girls as my sisters. I am thankful to Allah for everything and to have you guys as my family. I love you all to bits.

Contents

List of Figures

List of Tables

List of Acronyms

UAS	Unmanned Aerial Systems
UAV	Unmanned Aerial Vehicle
UCLan	University of Central Lancashire
CEPM	Centre for Energy and Power Management
FTFCS	Fault Tolerant Flight Control Systems
FDIA	Failure Detection, Identification and Accommodation scheme
FA	Failure Accommodation
FDI	Failure Detection and Identification
SFDIA	Sensor Failure Detection, Identification and Accommodation scheme
AFDIA	Actuator Failure Detection, Identification and Accommodation scheme
NN	Neural Network
FCC	Fully Connected Cascade
MLP	Multi-Layer Perceptron
FAA	Federal Aviation Administration
FTCS	Fault Tolerant Control System
PFTCS	Passive Fault Tolerant Control System
AFTCS	Active Fault Tolerant Control System

GLR	Generalised Likelihood Ratio
CUSUM	Cumulative Sum
SPRT	Sequential Probability Ratio Test
MMAE	Multiple Model Adaptive Estimation
IMM	Interacting Multiple Model
MM	Multiple Model
MMST	Multiple Model Switching Tuning
TAFA	Tailless Advanced Fighter Aircraft
VTOL	Vertical Takeoff and Landing
PCA	Propulsion Controlled Aircraft
EKF	Extended Kalman Filter
SSME	Space Shuttle Main Engine
EBP	Error Back Propagation
EMRA	Extended Minimal Resource Allocation
RBF	Radial Basis Function
IFCS	Intelligent Flight Control System
OFC	Oscillatory Failure Case
BNN	Biological Neural Network
ANN	Artificial Neural Network
LM	Levenberg-Marquardt
NBN	Neuron by Neuron
SSE	Sum Squared Error

FCS	Flight Control System
AI	Artificial Intelligence
AHRS	Attitude/Heading Reference System
INS	Initial Navigation System
MSL	Mean Sea Level

List of Symbols

w	Weight of neural network connection
f	Activation function
sec	Second(s)
n	Index of the training iteration
$\boldsymbol{w_{n+1}}$	New weights vector
$\boldsymbol{w}$	Previous weights vector
J	Jacobian matrix
I	Identity matrix
$\boldsymbol{e}$	Error vector
μ	Combination coefficient for LM and NBN algorithm
p	Training pattern
j	Index of the neuron
$w_{j,x}$	x^{th} connection weight w to neuron j
m	Index of the neural network output neuron
$e_{p,m}$	Error for training pattern p at neural network output neuron m
$d_{p,m}$	Desired output for pattern p at neural network output neuron m
$o_{p,m}$	Actual output for pattern p at neural network output neuron m
M	Number of neural network output
P	Number of training patterns
C	Number of weights in the neural network
y_j	Output node of neuron j
$y_{j,i}$	i^{th} input node of neuron j
net_j	Sum of the weighted inputs to neuron j

s_j	Slope (or derivative) of the activation function f_j
$\delta_{m,j}$	Signal gain between neuron j and output neuron m
k	Neuron k
E	Sum squared error
Q	Quasi-Hessian matrix
$\boldsymbol{g}$	Gradient vector
$q_{p,m}$	Quasi-Hessian sub-matrix
$\eta_{p,m}$	Gradient sub-vector for pattern p and output neuron m
$\boldsymbol{j_{p,m}}$	Jacobian row for pattern p and output neuron m
x_n	Normalised value
x_o	Value to be normalised
a	Minimum value of the range to be normalised to
b	Maximum value of the range to be normalised to
x_{min}	Minimum value of the range to be normalised from
x_{max}	Maximum value of the range to be normalised from
F_A	Fault alarm
F_S	Fault switch
q	Pitch rate (deg/sec)
p	Roll rate (deg/sec)
r	Yaw rate (deg/sec)
q	Pitch acceleration (deg/sec^2)
p	Roll acceleration (deg/sec^2)
r	Yaw acceleration (deg/sec^2)
a_x	Acceleration along the X axis of the aircraft (deg/sec^2)
a_y	Acceleration along the Y axis of the aircraft (deg/sec^2)
a_z	Acceleration along the Z axis of the aircraft (deg/sec^2)
δ_A	Aileron demand (command)
δ_R	Rudder demand (command)
δ_E	Elevator demand (command)
g_x	Gravitational acceleration component along the aircraft X axis

g_y	Gravitational acceleration component along the aircraft Y axis
g_z	Gravitational acceleration component along the aircraft Z axis
t	Time
k	Time instance k
x_t	Sensor signal at time t
r_t	Useful signal from the sensor at time t
n_t	Noise signal from the sensor at time t
f_t	Failure signal from the sensor at time t
b	Constant bias
t_f	Time of failure
A	Magnitude of the additive fault
T_R	Duration of the ramp
d	Residual
n	Size of sliding data window
D	Residual
τ	Fault threshold
ϕ	Roll Euler angle (deg)
ψ	Yaw Euler angle (deg)
θ	Pitch Euler angle (deg)
c	Scalar multiple

"And, when you want something, all the universe conspires in helping you to achieve it."

– Paulo Coelho, *The Alchemist*

Chapter 1

Introduction

1.1 Unmanned Aircraft Systems and Endurance

Over the years, there has been a significant growth in the application of unmanned aerial systems (UAS). UAS are also commonly known as unmanned aerial vehicles (UAVs). UAS are used in applications such as border security, reconnaissance, aerial survey, search and rescue, to name a few, and military and scientific research. The growth in these types of aircraft systems can be attributed to a number of benefits, such as low cost, lack of risk to a human pilot in dangerous missions and the ability to conduct lengthy missions which may otherwise be cumbersome for a human pilot.

In 2009, a strategic research partnership agreement was signed between BAE Systems and the University of Central Lancashire (UCLan). As part of this partnership, the Centre for Energy and Power Management (CEPM) was setup and one of the research objectives of CEPM was to achieve longer endurance in UAS [5]. Long endurance is not just defined in terms of longer flight durations, such as continuous flights for days or months. Instead, the research also focused on achieving endurance from the following aspects:

1. **Intelligent Energy Management Systems**

   In this aspect, the research focused on the development of intelligent management algorithms/systems that could turn different parts of aircraft systems on or off to maintain the current state of the aircraft with minimum energy requirement. The goal here was to save energy by turning off systems that were

otherwise not required during that time. The energy saved could later be used to increase the duration of the flight, thereby adding endurance to the aircraft. These algorithms/systems were to be designed for seamless integration with existing aircraft systems.

2. **Fault Tolerant Control**

   Aircraft systems like any other system are prone to failure. This aspect of the research was to investigate the development of technology that can add endurance to an aircraft in case of failure. For example, consider an extreme case, where a section of an aircraft wing detaches from the aircraft body due to structural failure or battle damage. In this case, endurance could be added by developing systems that attempt to maintain flight following such a severe failure. In general, the goal here was to increase the endurance of the aircraft system in the presence of failures.

This thesis is part of the research effort by the CEPM in achieving longer endurance in UAS from the fault tolerant control aspect. To this end, this research aims to add fault tolerance capabilities to flight control systems, in order to add endurance to the aircraft in the presence of failures. In a nutshell, this research investigates the development of technology to obtain fault tolerant flight control.

1.2 Research Overview

Fault tolerant flight control systems (FTFCS) have the ability to tolerate component failures automatically while maintaining overall system stability and acceptable performance in the event of failures [7–9]. Their purpose is to detect, identify and accommodate any type of failure that may occur during flight [10,11]. Such systems can be implemented by developing failure detection, identification and accommodation (FDIA) schemes. There are two parts to an FDIA scheme. The first part is the failure detection and identification (FDI) and the second part is the failure accommodation (FA), where the necessary actions are taken to accommodate the failure. Two of the major types of failure in an aircraft system are sensor and actuator

failures [10, 12, 13]. Therefore in this research, two schemes are developed:

- Sensor failure detection, identification and accommodation (SFDIA)
- Actuator failure detection, identification and accommodation (AFDIA)

A number of techniques exist in the literature that address fault tolerance issues in manned and unmanned aircraft systems. These techniques are, for example, Kalman filters [14], neural networks (NNs) [11], fuzzy logic [15] and a combination of the previous techniques [16]. Over the past three decades there has been an increasing interest in the application of NN for SFDIA and AFDIA [11, 17–19]. This can be attributed to the following two properties of the NN [11]:

- **Learning ability and adaptation**

  NN can learn to represent a system using past data collected from the system. This is very useful where the mathematical model of the system might be limited. Furthermore, they have the added ability to adapt or learn on-line using the current data of the system to improve its representation on-line [20–22].

- **Application to non-linear systems**

  An aircraft system can be significantly non-linear during various phases of the flight. However, most of the solutions in the literature rely on a linearised model of the aircraft for fault tolerant control applications [10]. NNs have been shown to successfully represent highly non-linear systems, hence the suitability for this research [12, 23].

In this research, the developed SFDIA and AFDIA schemes are based on the fully connected cascade (FCC) NN. In this NN architecture, all possible forward connections are made, with the neurons arranged in a cascade. This architecture has been studied extensively by Wilamowski [24–28]. Compared to the popular NN architecture - the multi-layer perceptron (MLP) - the FCC architecture is more powerful, compact and efficient. This thesis details the first attempts at exploiting the benefits of the FCC NN architecture for SFDIA and AFDIA in aerospace applications.

The SFDIA scheme developed in this research addresses failures in the pitch, roll and yaw rate gyroscope sensors of an aircraft system. The endurance of an aircraft can be increased during failures of these three sensors, using the SFDIA scheme. In total, 7 failure cases are considered over 105 failure experiments. In this scheme, FCC NN based pitch, roll and yaw rate sensor estimators are developed which replace the faulty sensor, once a failure is detected.

The AFDIA scheme developed in this research, aims to increase the endurance of an aircraft following a 66% loss of wing surface. The scheme employs a FCC NN based roll controller, which adapts on-line to control the aircraft, and maintain flight with the wing surface missing. Due to the quick adaptation of the FCC NN based roll controller, the controller is able to adapt to the post failure dynamics of the aircraft and maintain flight. Note that the percentage of wing surface loss is defined by the aircraft model used in the simulator. Further discussion on this is presented in Chapter 5.

Table 1.1: Hardware specifications of the MacBook Pro laptop on which this research was conducted.

MacBook Pro Laptop Specifications	
Processor	2.3 GHz intel Core i7 Quad Core
Memory	8 GB 1333 MHz DDR3
Graphics	AMD Radeon HD 6750M 1024 MB

For this research, the X-Plane 9 flight simulator is used for data collection and as a test bed. This simulator was the main framework for testing the systems developed by the CEPM research team. This simulator produces realistic flight simulations due to which its professional version is certified by the Federal Aviation Administration (FAA) for pilot training [29, 30]. It is also used by the likes of NASA, Cessna and Japan Airlines, to train pilots, develop concept designs and flight testing [30, 31]. This research was conducted on a MacBook Pro laptop, the specifications of which are presented in Table 1.1. Additionally, the schemes developed here were coded using the C programming language and the LAPACK library [32] was used to implement the NN.

1.3 Research Contribution

The main contribution of this research can be summarised as follows:

1. The overall objective of this research is to add endurance to an aircraft system in the presence of failures. This is achieved by developing the FCC NN based SFDIA and AFDIA schemes, and is the first reported instance of FCC NN being applied to fault tolerance applications, especially in aircraft systems.

2. An SFDIA scheme based on the FCC NN is developed to add fault tolerance to an aircraft following a pitch, roll or yaw rate sensor failure. This scheme can add endurance to an aircraft system suffering said failures.

3. In the AFDIA scheme, the FCC NN is used to control the roll of an aircraft after a failure, which adds endurance to an aircraft following a 66% loss of wing surface. This is a severe case of failure where a major section of the aircraft wing breaks from the main structure, resulting in an extreme loss of lift. The scheme succeeds in flying the aircraft following a 66% wing surface loss.

1.4 Organisation of the Thesis

In this section, the structure of the thesis is outlined to provide an overview to the reader.

Chapter 2 provides an overview of the literature on fault tolerant flight control systems (FTCS). This covers the industry wide practice and various relevant methods presented in the open literature.

Chapter 3 reviews the literature behind the decision for the selection of the FCC NN architecture and the neuron by neuron (NBN) learning algorithm for this research.

Chapter 4 presents the development of the FCC NN based SFDIA scheme. It discusses the development process, experiments and the results obtained from the SFDIA scheme.

Chapter 5 discusses the challenges encountered while using the X-Plane simulator for this research.

Chapter 6 presents the development of the FCC NN based AFDIA scheme. It discusses the development process, experiments and the results obtained from the AFDIA scheme.

Chapter 7 provides a conclusive summary of the research conducted followed by proposed future work.

1.5 Publications by the Author

In this section, a list of articles published or submitted for publication is presented. The articles are based on the research presented in this thesis.

1. "Sensor Failure Detection, Identification and Accommodation using Fully Connected Cascade Neural Network", Saed Hussain, Maizura Mokhtar, Joe M. Howe. IEEE Transactions on Industrial Electronics (Impact: 6.5) (*Accepted for Publication on 6/09/2014, In Process for Publishing.*)

2. "Aircraft Sensor Estimation for Fault Tolerant Flight Control System using Fully Connected Cascade Neural Network", Saed Hussain, Maizura Mokhtar, Joe M. Howe, International Joint Conference on Neural Networks (IJCNN), Aug 4-9, 2013.

3. "Adaptive and Online Health Monitoring System for Autonomous Aircraft", Maizura Mokhtar, Sergio Z. Bayo, Saed Hussain, Joe M. Howe, AIAA Guidance, Navigation, and Control Conference, Aug 13, 2012.

"Your time is limited, so don't waste it living someone else's life."
– Steve Jobs

Chapter 2

Literature Review: Fault Tolerant Flight Control

2.1 Overview

The aim of this chapter is to provide the reader with the necessary background knowledge for the research presented in this thesis. In Section 2.2, the concept behind fault tolerant flight control systems (FTFCS) is introduced. In Section 2.3, an overview of the literature on FTFCS is presented. The view of the author on the current industry practice is presented in Section 2.4 and finally the chapter concludes with Section 2.5.

2.2 Fault Tolerant Flight Control System

Fault tolerant flight control systems (FTFCS) are systems that have the ability to tolerate component failures automatically while maintaining overall system stability and acceptable performance in the event of failures [7–9]. Generally speaking, any fault tolerant control system (FTCS) can be divided into two categories: passive (PFTCS) and active (AFTCS). PFTCS are fixed controllers that are designed to be robust against a class of failures. This category of controller has limited fault tolerant capability and lacks any mechanism to actively detect and identify developing faults [7, 33, 34].

In contrast to this, AFTCS actively monitors the system for the presence of faults and takes necessary actions to compensate for the failures [7, 33, 34]. The research presented in this thesis falls under the AFTCS category of controller. In the literature, such systems are also known as self-repairing, reconfigurable, re-structurable and self-designing control systems [7]. From the point of view of functionality, these controllers are also known as failure, detection, identification and accommodation (FDIA) schemes [7]. In this research, the FDIA schemes terminology is used to define the developed systems. Therefore, a FTFCS can be achieved by implementing an FDIA scheme [11, 13]. There are two part to an FDIA scheme [7, 11]:

- **Failure detection and identification (FDI)**, which detects significant abnormalities and identifies the cause.
- **Failure accommodation (FA)**, which takes the necessary action to reconfigure the control system to compensate for the impact of the failure.

Two of the major types of failure in an aircraft system are sensor and actuator failures [10, 12, 13]. Therefore in this research, two schemes are developed:

- **Sensor failure detection, identification and accommodation (SFDIA)**
- **Actuator failure detection, identification and accommodation (AFDIA)**

The key ingredient in any FTFCS is the availability of redundancy in the system, which plays an important role in the FDI stage [11, 34, 35]. There are two types of redundancy: hardware and analytical [11, 35]. In hardware redundancy, identical sensors are used to measure the same parameter. For example, consider an SFDIA scheme using hardware redundancy as shown in Figure 2.1. A voting scheme is employed to detect and identify any faulty sensor [12, 36, 37]. If the signal from one sensor differs significantly from the remaining two sensors, the sensor is declared as faulty. Sensor failure accommodation is achieved by replacing the faulty sensor with one of the two remaining sensors.

In the aircraft industry, the state of the art practice to achieve FTFCS is to implement high levels of hardware redundancy [7, 10, 11, 36]. For example, Airbus

A320/330/340/380 has triple or quadruple redundant actuation, sensor and flight control systems [36]. This is mainly because the failure detection and identification (FDI) mechanism is quick and reliable; and fault accommodation (FA) is easily achieved by switching to the fault free hardware. However, hardware redundancy has serious cost, power and weight implications, especially for aircraft such as UAVs. Due to these implications, analytical redundancy has become a far more appealing approach for FTFCS [11].

Generally in analytical redundancy, a model of the monitored system is used to generate signals that would otherwise be generated by redundant hardware (see Figure 2.2). In its simplest form, the difference between the model estimate and the measured reading is used to generate an error residual. This residual is then monitored to detect and identify faults [37].

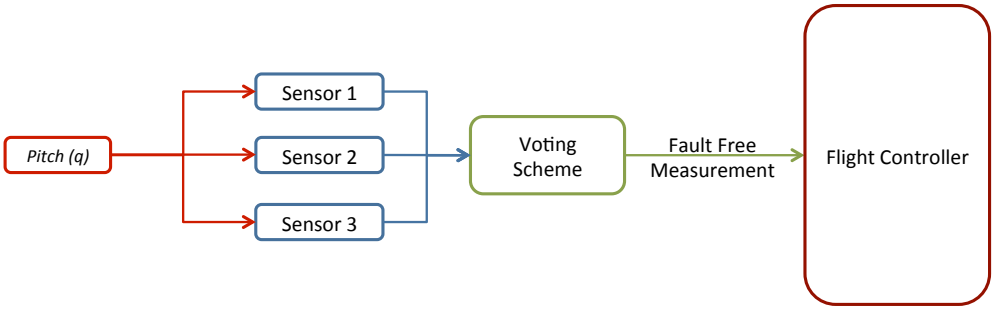


Figure 2.1: Block diagram of hardware redundancy.

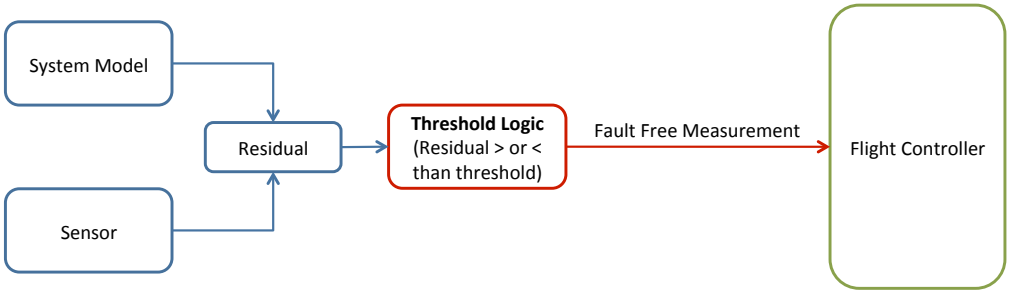


Figure 2.2: Block diagram of analytical redundancy.

2.3 Related Work

In Figure 2.3, a generic structure of the analytical redundancy based FDIA scheme for FTCS is presented. Based on this figure, the development of FTCS can be

divided into 3 separate steps [35]. The first step is to generate the residual signal using a mathematical model of the system. In some FDIA schemes, there are several analytical models, each of which is sensitive to different types of fault. Once the residual is generated, the next step is to evaluate the residual to decide if any failure has occurred. This is the step where the failure is detected (FD) and the source and type of failure are identified (FI). A simple way of implementing this step is to use a constant residual threshold. When a residual crosses a threshold, the fault corresponding to that residual is instantaneously detected and identified. More sophisticated methods of residual evaluation may consist of adaptive residual thresholds or based on statistical decision theories such as generalised likelihood ratio (GLR), cumulative sum (CUSUM) or sequential probability ratio test (SPRT) [7, 11, 35]. As is obvious, the two steps discussed so far are collectively known as the failure detection and identification (FDI) part of the FDIA scheme.

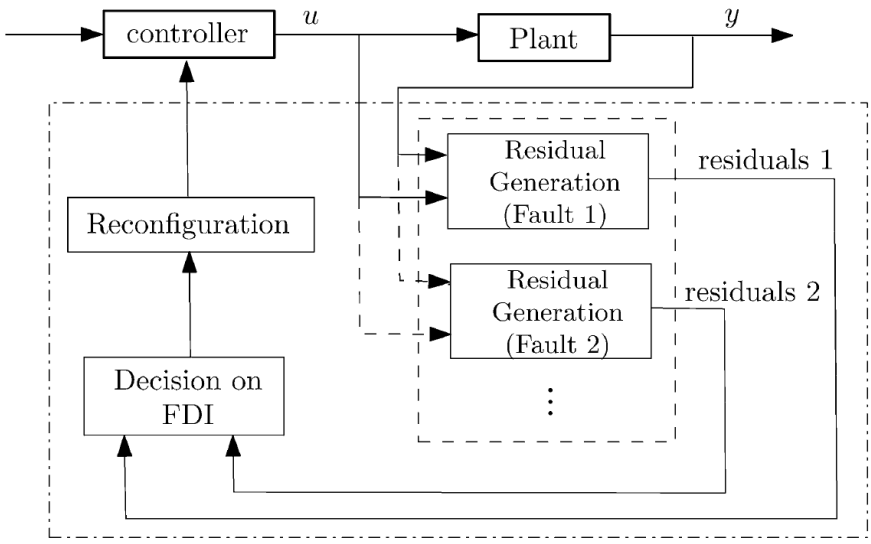


Figure 2.3: A generic structure of the analytical redundancy based failure detection identification and accommodation (FDIA) scheme. Note that this figure is taken from [35].

The final step to developing an FTCS is the failure accommodation (FA), where the controller is reconfigured on-line in response to the faults. This step could include just swapping to a different controller that can handle the fault or adapting the controller in response to the fault. In addition, the fault can be accommodated by reconfiguring the input signals and/or the output control of the controller. For example, a faulty sensor signal can be replaced by an analytical model and actu-

ator failures can be compensated by distributing the actuation on the remaining actuators.

Some of the popular approaches in the literature for the analytical models are the use of Kalman filters, Luenberger observers, fault detection filters and neural networks, to name a few [7, 11, 17, 35, 38]. Kalman filters are usually used in multiple model based FDI schemes. In these schemes, a bank of Kalman filters is generated off-line. Of these models, one of them represents the normal mode of the system, while the others are sensitive to different types of fault. Failure detection is achieved by using a hypothesis testing algorithm that monitors the residuals from each Kalman filter and assigns a probability to each of the fault hypotheses [13, 33, 35]. Some well known approaches to FDI schemes based on multiple models of Kalman filters are multiple model adaptive estimation (MMAE) and interacting multiple model (IMM) [7, 35].

With inspiration from the multiple model based FDI schemes, several multiple model (MM) based FDIA schemes for fault tolerant flight control systems have been developed [7, 11, 13, 33, 35]. Generally in the MM method for FTFCS, a bank of models is used to describe the system under normal operating mode and under various fault conditions, such as sensor or actuator failures. There is a model for every fault considered. For each of these models, a controller is designed (off-line) that can be used to accommodate the failure. Failure detection and identification is performed by using these fault models and, based on a suitable switching mechanism, the corresponding controller is selected for failure accommodation.

There are a number of variants to the MM method [33]. For example, in [39], Boskovic and Mehra applied the multiple model switching and tuning (MMST) method to add fault tolerance to the flight control system of a tailless advanced fighter aircraft (TAFA), in the presence of wing damage. As described earlier, in their system, there is a model for each fault scenario, which they refer to as identification models. And for each fault model, there is a corresponding controller. This formed a massively parallel architecture of identification models and their corresponding controllers. When a failure occurs, a switching mechanism quickly iden-

tifies the model that is the closest to the current damage mode, and switches to its corresponding controller. In addition, examples of IMM and MMAE based FDIA schemes for FTFCS can be found in [40–44]. For instance, in [43] and [44], the authors have developed an interacting multiple model (IMM) based FTFCS. They demonstrated the effectiveness of the system on a longitudinal vertical takeoff and landing (VTOL) aircraft model, which was subjected to single actuator, sensor and component failure at a given time.

A special case of the MM based FTFCS is the propulsion controlled aircraft (PCA). In this case, the only anticipated fault is complete hydraulic failure. The only way to control the aircraft is using the engine throttles. In 1995, NASA Dryden Flight Research Centre demonstrated the PCA method by successfully landing a MD-11 and an F-15 aircraft using just propulsion only control, following a complete hydraulic failure [33, 45–47].

Most of the early work on analytical redundancy is based on observers and Kalman filters [7, 11, 12, 18, 37, 48]. These techniques relied on the linear time invariant mathematical models of the systems. In aircraft systems, the assumption that the system is linear is not often valid throughout the entirety of the flight envelope [13, 17, 48]. Therefore, these techniques might perform inadequately in the non-linear regions of the flight envelope. In addition, these techniques can suffer from modelling discrepancies between real and mathematical models of the system [17]. Recent literature has seen efforts made to address these issues, especially with the linearity assumption of the Kalman filters [49]. Several improved versions of the Kalman filter have been developed and applied to various fault tolerance and state estimation problems in non-linear systems [14, 16, 17, 35, 50, 51]. For example in [17], an SFDIA scheme for the pitch rate sensor of the aircraft based on the extended Kalman filter (EKF) is presented, due to its applicability to non-linear problems and its popularity.

Over the past three decades, there has been an increasing interest in the application of neural networks (NN) for FTFCS [10, 11, 17, 18, 52–54]. This is mainly due to their innate ability to model both linear and non-linear systems [11, 52]. Unlike

Kalman filters, they do not require a detailed mathematical description of the system. They develop a structure based on training data instead. In addition, they can also be made to adapt on-line, whilst the system is in use; in order to adapt to the dynamic conditions of the environment and the system dynamics. On-line adaptation is provided by the on-line learning algorithm.

In [17], two SFDIA schemes for the pitch rate sensor of the aircraft are compared. Of the two schemes, one is based on an NN while the other uses EKF. From the comparison results, the authors concluded that the NN based SFDIA scheme outperformed the EKF based scheme. In [54], Guo and Musgrave presented a SFDIA scheme for sensors in the space shuttle main engine (SSME). Their scheme is based on the auto-associative multi-layer perceptron (MLP) NN, trained using the error back propagation learning algorithm (EBP).

Napolitano et al. presented an SFDIA and an AFDIA scheme in [11], which are based on the MLP NN. They studied the developed schemes using a mathematical model of the B747-200 aircraft model. The SFDIA scheme is capable of accommodating failures in the pitch, roll and yaw rate gyro sensors. In this scheme, MLP NN based estimators replace the respective faulty sensors. The developed AFDIA scheme is able to accommodate failures in the rudder, elevator and aileron control surfaces. In this case, MLP NN based controllers (for pitch, roll and yaw) are used to compensate for the failure by producing the appropriate control response. In [53], the performance of an MLP NN and an extended minimal resource allocating (EMRA) NN were evaluated for airspeed sensor failure. To evaluate the performance, real data collected from the jet-powered WVU YF-22 research UAV was used. They concluded that both the NNs were suitable to accommodate airspeed sensor failure, as part of an SFDIA scheme.

Samy, Postlethwaite and Gu proposed a SFDIA scheme using the radial basis function (RBF) NN in [10] and [37]. The scheme is aimed at accommodating failures in the pitch rate, normal acceleration and angle of attack sensors. Out of the 30 SFDIA experiments conducted, only 2 faults went undetected. In [55], the MLP NN was used again to develop an SFDIA scheme for the pitch, roll and yaw rate

sensors of a model UAV. Liu et al. developed an FTFCS using the MLP NN which is capable of handling an aircraft with primary control surfaces failures.

Additional examples of the application of NNs for FTFCS can be found in the following references [20, 38, 56–58]. A notable project on the application of an NN for flight control systems is the NASA Intelligent Flight Control System (IFCS) flight research project at NASA Dryden Flight Research Centre [22, 59]. The aim here is to use the learning ability of the NNs to develop software to aid the pilot with controlling the aircraft following a failure of a control surface or damage to the airframe. The developed NN based control systems were tested on a highly modified F-15B aircraft.

When it comes to NN based applications, including FTFCS problems, the popular choice for the NN architecture is the MLP NN [11, 24, 48, 60]. In this thesis, the schemes are based on the fully connected cascade (FCC) NN. This architecture is far more efficient than the MLP, as it requires fewer neurons to solve a problem [24, 60]. Therefore the efficiency of any MLP based FTFCS can be improved by updating it to use the FCC NN instead.

2.4 Industry Practice

To the knowledge of the author, the analytical redundancy based fault tolerant approach is still an idea limited to the literature. The practice in the aircraft industry is to use high levels of hardware redundancy [36]. However, an example of analytical redundancy can be found in the Airbus A380 aircraft, for a case of a failure called the oscillatory failure case (OFC) [36, 61]. One of the reasons for the limited use could be that the hardware redundancy based approach is much simpler to implement compared to developing an analytical redundancy based system.

The author believes that with the growing UAV sector of the aerospace industry, analytical approaches will eventually be common practice in the industry. Analytical redundancy will save cost, space, weight and power, which are especially limited in UAVs.

2.5 Conclusion

The aim of this chapter was to provide the reader with an overview of the literature on fault tolerant flight control systems (FTFCS). For additional reviews of the literature, interested readers are advised to refer to the following survey/introductory publications on FTFCS [7,9,33,62,63]. In the next chapter, a review of the literature on neural networks (NNs) is provided, to select the NN architecture and learning algorithm for this research.

"We must believe that we are gifted for something, and that this thing, at whatever cost, must be attained."

– Marie Curie

Chapter 3

Neural Network Architecture and Learning Algorithm

3.1 Overview

The human brain is a complex and magnificent organ, stemming from millions of years of evolution. Amongst other types of cell, the brain is comprised of ≈86 billion neurons, interconnected in a vast mesh [64,65], referred to as biological neural networks (BNNs). Scientist, doctors and engineers alike, have been captivated with comprehending the inner workings of the brain. This is not simply for the yielding of psychological and medical benefits, but also from the desire to design and construct machines which are essentially near human; or at the very minimum, possess the human ability to learn and adapt. To this end, the field of artificial neural networks (ANNs) was conceived, born and continues to grow. The research presented in this thesis utilises ANNs.

In this chapter, a brief introduction to ANNs is presented in Section 3.2. The ANN architecture used in this research is selected in Section 3.3 and in Section 3.4, the learning algorithm to train the ANN is decided. An insight into the number of neurons and the generalisation ability is presented in Section 3.5. A summary of the reasons behind the selected ANN architecture and the learning algorithm is presented in Section 3.6. In Section 3.7, a brief comparison between the neuron by neuron (NBN) and the Levenberg-Marquardt (LM) learning algorithms is presented.

Sections 3.8 to 3.10 detail the implementation of the NBN algorithm and the training steps it entails. The settings for the ANN used in this research are presented in Section 3.11 and finally the chapter concludes with Section 3.12.

3.2 Brief Introduction to Artificial Neural Networks

ANN are simple mathematical representation of the BNN. Similar to the BNN, the ANN consists of processing elements called neurons. These artificial neurons are simplistic representation of the biological neurons. Unlike the BNN, where the neurons are interconnected in a mesh, the neurons in the ANN are connected in an organised manner based on the architecture of the ANN. From this point on, ANN will be referred to as NN only. There are two main parts to consider when it comes to NNs: NN architecture and learning algorithm.

One of the most popular NN architectures is the multilayer perceptron (MLP) architecture [24, 60]. In this architecture, the neurons of the NN are arranged in layers. In the MLP NN presented in Figure 3.1 there are three layers of neurons. The first layer of neurons that receives the inputs to the NN is called the input layer. The layer of neurons that emits the output of the NN is called the output layer. And the layer between the input and output layers is referred to as the hidden layer.

The connections between these layers of neurons are weighted (w). Each of these neurons sum up the incoming weighted signals from the neurons or NN inputs from the previous layers and pass the sum through a function referred to as the activation function (f). Based on the summation value and the activation function, the neuron emits an output. It is these connection weights where the NN stores/learns its functionality. These weights can be adapted using a suitable learning algorithm to train the NN to perform a function. This learning can be off-line, when the NN is not in use, or on-line while the NN is actively in use.

To train the NN for a function, data representing that function must be available. For example, if the NN is to replicate a sensor, then input data along with the sensor

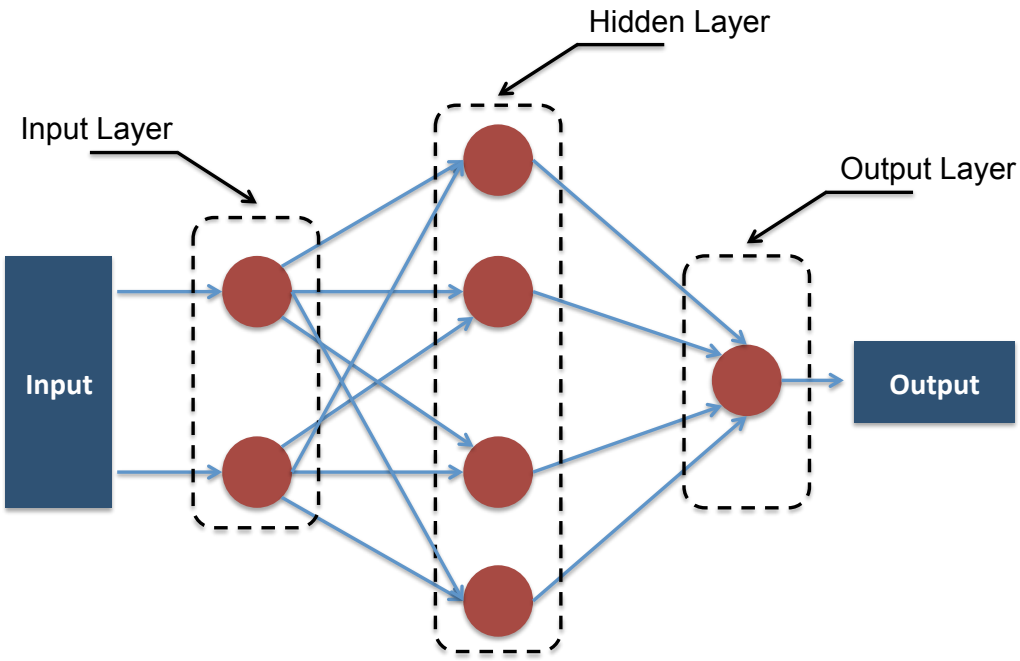


Figure 3.1: The multilayer perceptron (MLP) neural network (NN).

measurements are required to train the NN. The learning algorithm used in the training process maps the inputs to the desired sensor measurement. The dataset used by the NN for training is called the training dataset. The individual dataset (a pair of inputs and output) is know as a training pattern. Once the NN is trained, its functionality must be validated or verified. This is achieved by using a different dataset, known as the validation dataset.

When it comes to learning algorithms for NNs, it could be said that the popular choice is the error back-propagation algorithm (EBP) [60, 66]. This algorithm is often used with the MLP network topology. The process of training a network for a function using this algorithm, can be divided into two phases:

1. Forward propagation

2. Backward propagation

In the forward propagation phase, the input signal is propagated from the input neuron layer to the output neuron layer. No weights are adapted, therefore no learning takes place. The network generated outputs are compared against the desired output for that pattern of inputs. The difference between the desired and the actual NN output is the error. In the backward propagation phase, the errors

calculated are then propagated back from layer to layer and the weights connecting them are adapted to minimise the error using the learning algorithm. It is in this phase that the weights are adapted and learning takes place. The two phases are then repeated using new patterns or old patterns until the error is within an acceptable range. This is the process in which NNs learn their functionality.

Ever since the first NN model was proposed in 1943 by McCulloch and Pitts [67], NNs have been used in wide range of applications. For example, NNs have been used in forecasting weather in [68]. In [69], a review of NNs for pharmaceutical applications is presented. In [70], Atiya reviews NNs for corporate bankruptcies and developed a new NN based bankruptcy prediction model. NNs have also been used in the field of power electronics. An introductory review of the NN applications for power electronics is presented by Bose in [71]. These are just some of the examples of the extensive field of NN applications.

In this research, NNs are used for estimating aircraft sensor measurements and controlling aircraft attitude. In the remaining sections of this chapter, an NN architecture and learning algorithm is selected for the research presented in this thesis.

3.3 Selecting a Neural Network Architecture

Although the MLP architecture is the popular choice for NN applications, it is not very efficient when compared against architectures with connections across layers. The Fully Connected Cascade (FCC) architecture presented in Figure 3.2 is an example of an architecture with connections across layers. In this architecture, all possible forward connections are made with the neurons arranged in a cascade. This architecture has been widely studied by Professor Bogdan M. Wilamowski and his colleagues in [24,25,28,60,66,72–75]. They have shown that the FCC is more efficient than the MLP, as it requires fewer neurons to solve a problem.

A common benchmarking problem for NNs is the parity-N problem. The parity-N or N-bit parity function is a mapping defined on 2^N binary vectors that indicates whether the sum of the N elements of every binary vector is odd or even [75–77]. In its simplest form, the parity-2 is an exclusive-or (XOR) logic function, where

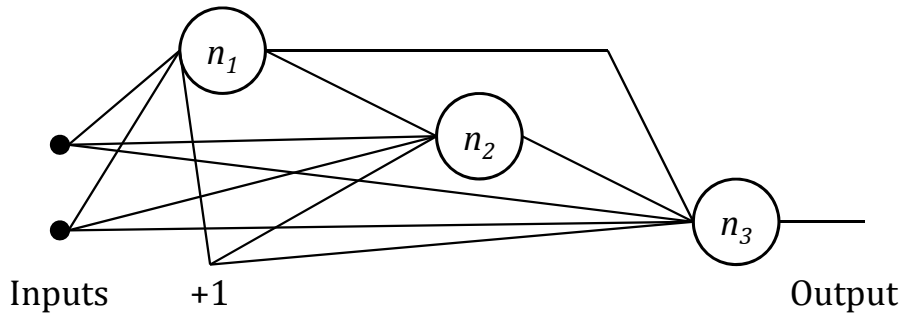


Figure 3.2: The Fully Connected Cascade (FCC) NN architecture. The presented NN has 2 inputs, 1 output and the bias input.

N represents the number of inputs. Depending on the NN architecture, different numbers of neurons and weights are required to solve the same parity problem. The larger the N, the more difficult it is to solve the problem.

Table 3.1: Comparison of FCC and MLP architecture to solve the Parity-N problem.

Parity	Architecture	No. Neurons	No. Weights
3	MLP	4	16
	FCC	2	9
7	MLP	8	64
	FCC	3	27
15	MLP	15	256
	FCC	4	70
31	MLP	32	1024
	FCC	5	170
63	MLP	64	4096
	FCC	6	399

Note: This table is adapted from [25, 60]. The MLP architecture is made of 1 hidden layer.

In Table 3.1, the minimum number of neurons and weights required for different parity problems using the FCC and MLP architectures are presented [25, 60]. With just 6 neurons and 399 weights, the FCC is able to solve the parity-63 problem. This is in comparison to the MLP which requires 64 neurons and 4096 weights. Similarly, to solve the parity-31 problem, the FCC requires 5 neurons and 170 weights, whereas the MLP requires 32 neurons and 1024 weights. It is clearly evident from this table

that the FCC is far more efficient than the popular MLP architecture. It is able to solve the same problem with significantly fewer neurons and weights.

In addition to the efficiency, another benefit of the FCC architecture is that it is relatively easy to find an optimal size of the NN to solve a problem. With the MLP architecture there is a large number of possibilities by varying the number of neurons and hidden layers. The FCC on the other hand, simply requires a search for the number of neurons in cascade.

Despite these benefits, the MLP architecture is far more popular. According to Wilamowski, one of the reasons for the popularity is the easy availability [25]. The MLP is one of the oldest architectures and is easy to write training software for. It is also readily available on popular research platforms such as MATLAB.

3.4 Selecting a Learning Algorithm

When it comes to learning algorithms for NNs, the error back-propagation (EBP) algorithm [60,78,79] is the most popular choice [25,27,60,80]. It is relatively simple to implement and can handle problems with an unlimited number of patterns. This algorithm is commonly used along with the popular MLP NN architecture. Due to its simplicity, it is easy to adapt the EBP algorithm for more efficient NN architectures, which allow connections across layers, such as the FCC NN. However, this algorithm is known to be slow and ineffective, especially when compared against the Levenberg-Marquardt (LM) or Neuron by Neuron (NBN) algorithm [25, 26, 60, 73, 74, 80, 81].

A comparison of the EBP, LM and NBN algorithms [60] is presented in Table 3.2. These algorithms are used to train the parity-4 problem using the MLP architecture. The MLP architecture has 1 hidden layer with 4 neurons. The maximum number of training iterations is set to 100 for both the LM and NBN algorithm. In the case of the EBP, this is set to 100000. The training process is repeated 100 times for each of the algorithms, with randomly generated initial weights in the range of ± 1. The sum squared error (SSE) is used to evaluate the training process and the expected SSE is set to 0.001.

From the results presented in Table 3.2, one can conclude that the LM and

Table 3.2: Comparison of the averages of different algorithms to solve the parity-4 problem using the MLP architecture. This table is taken from [60].

Algorithms	Success Rate (%)	No. Iterations	Training Time (msec)
EBP	68	12036.01	5348.52
LM	100	23.41	26.64
NBN	100	23.24	25.64

Note: For EBP momentum value: 0.5, learning constant: 1.

NBN algorithms require similar time and number of iterations to solve the parity-4 problem. However, the popular EBP algorithm requires 500 times more iterations and 200 times longer in training time for the same problem. In addition, the success rate of the EBP is 68% compared to 100% for the LM and NBN algorithms. The success rate of the EBP can be improved by using additional neurons to solve the problem. These results encapsulate how slow and inefficient the EBP algorithm is compared to algorithms such as the LM and NBN algorithm.

In the next section it will be shown why it is desirable to have an NN solution with a minimal number of neurons.

3.5 Number of Neurons and Generalisation Ability

It is quite often the case that too many neurons are used to solve a problem using a NN. With an increasing number of neurons, the NN converges to a solution faster and to smaller errors. However, this approach of increasing the number of neurons to quickly converge to a solution does have some issues. With an increasing number of neurons, the NN loses its generalisation ability [24, 25, 60, 73, 81]. This is the ability of the NN to perform well on patterns it has never seen before. If too many neurons are used, then the NN may overfit itself to the training patterns and respond poorly to new patterns. On the other hand if fewer neurons are used, the training error might not be very small, but the NN may produce much better results on new patterns. Therefore, in order to have good generalisation ability, the NN should use as few neurons as possible to obtain a reasonable training error [24, 25, 60, 73, 81].

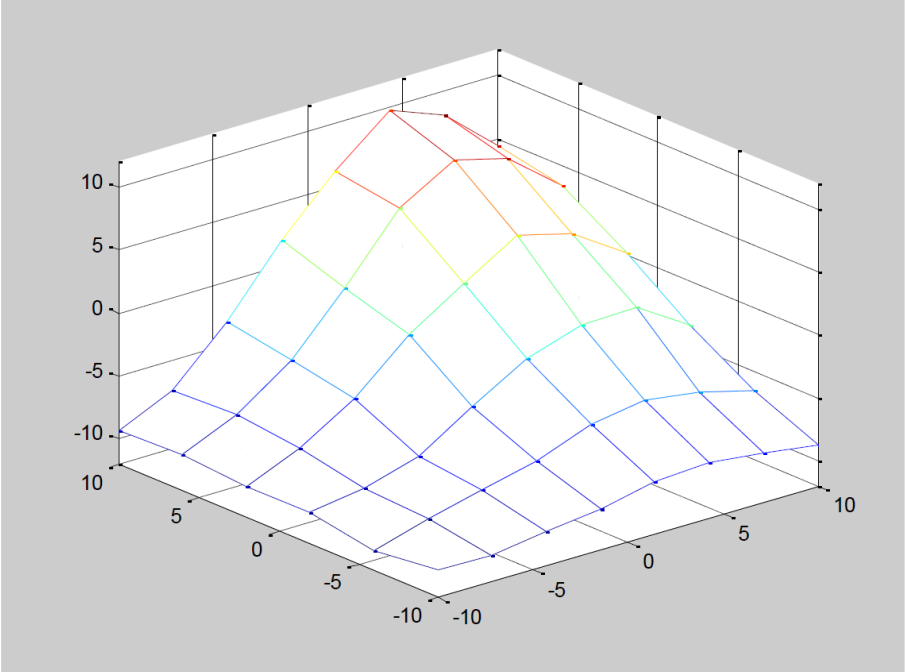
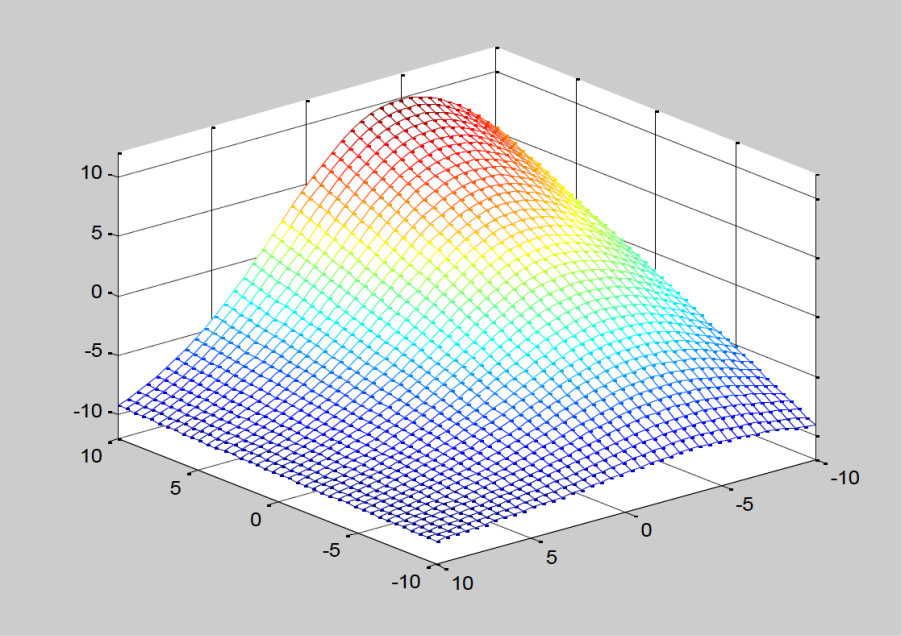


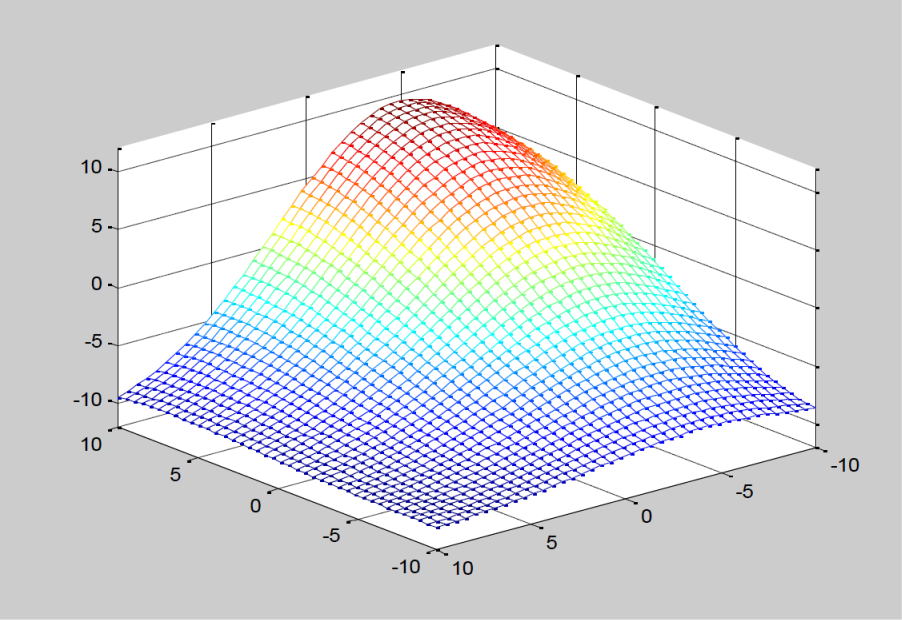
Figure 3.3: The TSK fuzzy controller control surface with $8 \times 6 = 48$ defuzzification rules. Note that this figure is taken from [24].

To demonstrate this, Wilamowski presented the results of finding the best NN solution to replace a fuzzy controller in [24, 25]. In Figure 3.3, the defuzzification rules and the required control surface for the Takagi, Sugeno and Kang or TSK fuzzy controller is illustrated. The defuzzification rules are used to train and develop the NN controller. The controller is based on the FCC architecture and to find a good solution to the controller, the number of neurons is varied. Figure 3.4 shows the results of an FCC NN controller using 3 (12 weights) and 4 (18 weights) neurons. In Figure 3.5, the results using 5 (25 weights) and 8 (52 weights) neurons are presented. In the captions of these figures, the training error for each of these FCC neural network controller designs is presented. As the number of neurons increases the training error decreases. For example, with 3 neurons the training error is 0.21049, whereas it is 1.118×10^{-5} using 8 neurons. However, it can be seen that with increasing size of the NN, the results become worse. According to Wilamowski, the best results were obtained using 4 neurons with a training error of 0.049061.

In conclusion, although with increasing number of neurons the training error can be decreased, the NN will lose its generalisation ability. For optimal performance, the NN must have as few neurons as possible.

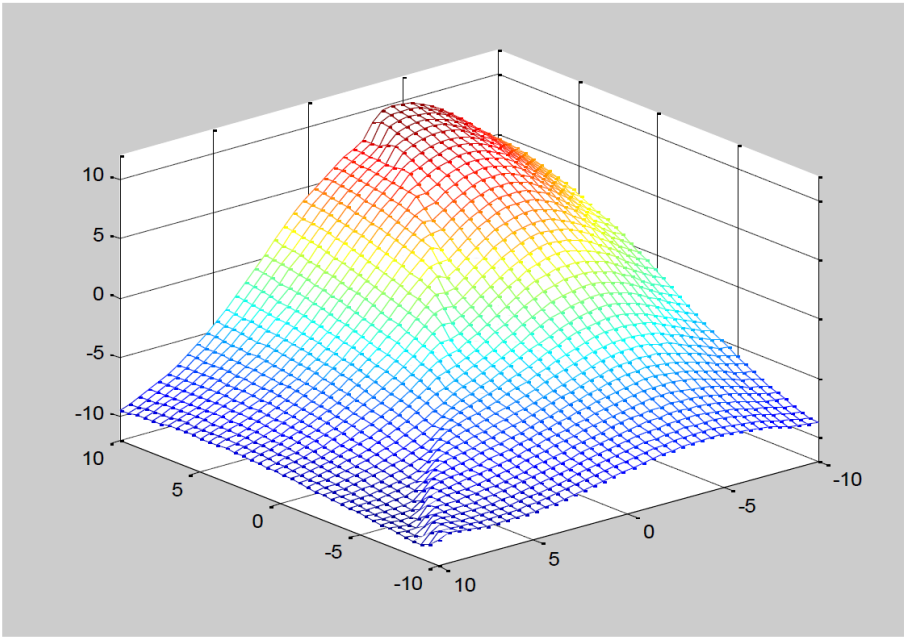


(a) Using 3 neurons (12 weights). Training Error = 0.21049

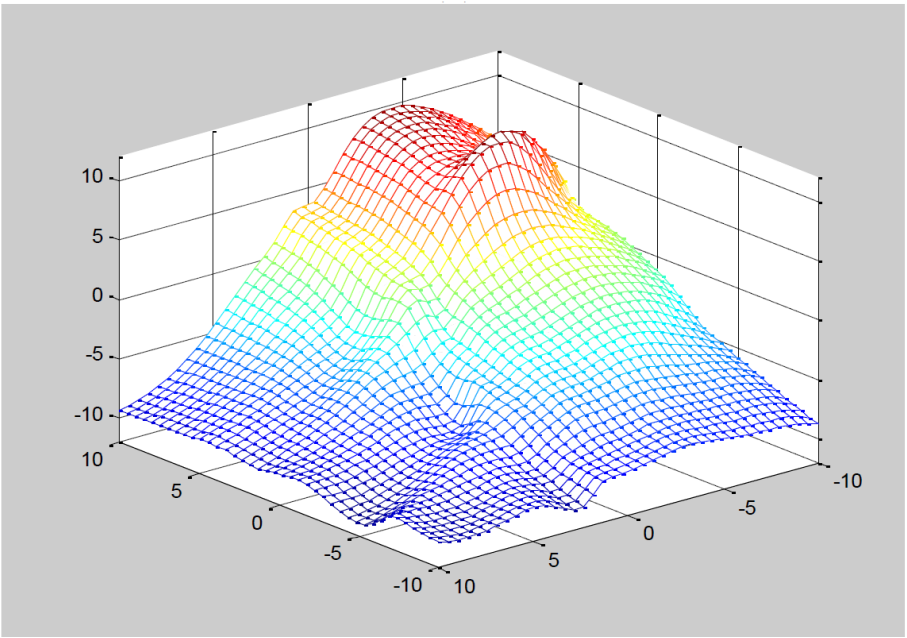


(b) Using 4 neurons (18 weights). Training Error = 0.049061

Figure 3.4: TSK Control Surface using 3 and 4 neuron FCC neural networks. Note that this figure is taken from [24].



(a) Using 5 neurons (25 weights). Training Error = 0.023973



(b) Using 8 neurons (52 weights). Training Error = 1.118×10^{-5}

Figure 3.5: TSK Control Surface using 5 and 8 neuron FCC neural networks. Note that this figure is taken from [24].

3.6 FCC NN architecture and NBN learning algorithm

In Section 1.3, the popular MLP NN architecture was compared against the FCC architecture. It was summarised that the MLP architecture is not very efficient when compared against the architecture with connections across layers, such as the FCC. The MLP requires significantly more neurons to solve a problem when compared against the FCC architecture. Therefore, using the FCC instead of the popular MLP, will have added benefits in terms of processing overhead. In addition, using the FCC can save development time when compared against the MLP. With the MLP, there is large number of possibility of designs (by varying the number of neurons and hidden layers) to experiment with. However, with the FCC, there is just the number of neurons to experiment with and usually a suitable solution is found within a couple of trials. Based on these conclusions, the FCC NN architecture is used for this research.

In addition to the NN architecture, the popular EBP algorithm was compared against the LM and NBN algorithm in Section 3.4. Table 3.2 shows that the EBP is not only slow, but is ineffective when compared against algorithms such as LM and NBN. It requires significantly more number of iteration to solve a problem compared to LM and NBN. In addition to being slow and ineffective, the EBP is not powerful enough to solve problems with limited neurons when compared against the NBN [24, 25, 60].

From the discussion in Section 3.5, it was concluded that in order to retain the generalisation ability of the NN, as few neurons as possible should be used. However, this cannot be achieved using the EBP. The EBP requires more neurons to solve a problem when compared against the NBN [24, 25, 60]. Therefore, in this research, the NBN algorithm is used to train the FCC NN based applications.

3.7 The LM and The NBN Algorithm

It is known that the LM algorithm is more efficient compared to the EBP learning algorithm [25, 27, 66, 80]. However, the LM algorithm does have some limitations that limits its applicability to wide range of NN applications. The LM algorithm is a combination of the gradient descent algorithm and the Gauss-Newton algorithm. In the LM algorithm, the weights are updated using the following update rule [81]:

$$w_{n+1} = w_n - (J^T J + \mu I)^{-1} J^T \boldsymbol{e} \tag{3.1}$$

where

n :	is the index of the iteration
$\boldsymbol{w_{n+1}}$:	is the new weights vector
$\boldsymbol{w_n}$:	is the previous weights vector
J :	is the Jacobian matrix
I :	is the identity matrix
$\boldsymbol{e}$:	is the error vector
μ :	is the combination coefficient (always positive)

$$J = \begin{array}{c} \overbrace{\hphantom{\frac{\partial e_{1,1}}{\partial w_{1,1}} \quad \frac{\partial e_{1,1}}{\partial w_{1,2}} \quad \cdots}}^{neuron\ 1} \quad \overbrace{\hphantom{\frac{\partial e_{1,1}}{\partial w_{j,1}} \quad \cdots}}^{neuron\ j} \\ \left[\begin{array}{ccccc} \frac{\partial e_{1,1}}{\partial w_{1,1}} & \frac{\partial e_{1,1}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{1,1}}{\partial w_{j,1}} & \cdots \\ \frac{\partial e_{1,2}}{\partial w_{1,1}} & \frac{\partial e_{1,2}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{1,2}}{\partial w_{j,1}} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{1,M}}{\delta w_{1,1}} & \frac{\partial e_{1,M}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{1,M}}{\partial w_{j,1}} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,1}}{\partial w_{1,1}} & \frac{\partial e_{P,1}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{P,1}}{\partial w_{j,1}} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,M}}{\partial w_{1,1}} & \frac{\partial e_{P,M}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{P,M}}{\partial w_{j,1}} & \cdots \end{array}\right] \end{array} \begin{array}{l} \\ \left.\begin{array}{l} m = 1 \\ m = 2 \\ \cdots \\ m = M \end{array}\right\} p = 1 \\ \quad \cdots \\ \left.\begin{array}{l} m = 1 \\ \cdots \\ m = M \end{array}\right\} p = P \end{array}$$

Figure 3.6: Jacobian Matrix J

$$\boldsymbol{e} = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ \cdots \\ e_{1,M} \\ \cdots \\ e_{P,1} \\ e_{P,2} \\ \cdots \\ e_{P,M} \end{bmatrix}$$

Figure 3.7: The Error Vector $\boldsymbol{e}$

The key to this weight update rule is solving the matrix J. The Jacobian matrix J is presented in Figure 3.6, where p is the training pattern; j is the index of the neuron; $w_{j,x}$ is the x^{th} connection weight w to neuron j; and m is the index of the network output neuron. The error $e_{p,m}$ for training pattern p at network output neuron m is calculated as follows:

$$e_{p,m} = d_{p,m} - o_{p,m} \tag{3.2}$$

where $d_{p,m}$ is the desired output and $o_{p,m}$ is the actual output for training pattern p at network output neuron m.

In the Jacobian matrix, notice that for every pattern p, there are M rows, where M is the number of the NN output. Therefore, there are $M \times P$ rows in the Jacobian matrix, where P is the number of training patterns. The elements in a row of the Jacobian matrix can be organised in terms of the neurons in the NN. For every neuron j, the number of elements corresponds to the number of weights connected to the neuron. In other words, there are C columns in the Jacobian matrix, where C is the number of weights in the NN. Therefore, the size of the Jacobian matrix can be described as $(P \times M) \times C$. The error vector $\boldsymbol{e}$ is presented in Figure 3.7, where for every pattern p there are M rows in the vector, therefore a total of $M \times P$ elements in the vector. Each of the elements in the error vector $\boldsymbol{e}$ is calculated using equation (3.2).

Usually the Jacobian matrix is calculated and stored for updating the weights using equation (3.1). For small to medium size training patters, the LM algorithm will work smoothly. However, if large training pattern sets are used, there will be a huge cost in terms of storage memory and computation time. This is because the number of elements in the Jacobian matrix J is proportional to the number of patterns P. This is one of the reasons why the LM algorithm is not always preferred for NN applications [25, 26, 66].

Apart from the memory requirement, the LM algorithm has several other limitations when compared against the NBN algorithm. The NBN algorithm is an improved version of the LM algorithm. The main advantages of the NBN algorithm over the LM algorithm can be summarised as follows [24, 82]:

1. **Train Neural Network with Connections Across Layers**

   The LM algorithm as presented in [83], can only handle the MLP architecture. As discussed in the previous sections, this architecture is inefficient when compared against architectures with connections across layers, such as the FCC. The NBN algorithm on the other hand can handle both the MLP and the efficient FCC NN architecture [74].

2. **Forward Propagation Only**

   In the LM algorithm, just like the popular EBP algorithm, there are two stages to training an NN, namely, forward propagation and backward propagation. In the LM algorithm [83], for a given pattern, the backward propagation has to be repeated for every output in the NN. Therefore, calculating the Jacobian matrix rows for those outputs. In the NBN algorithm, there is no need for backward propagation. All the information required is computed in the forward propagation phase of the NN training [80]. With a single forward propagation , all the Jacobian matrix rows corresponding to the NN outputs can be calculated. This makes the NBN algorithm efficient compared to LM, especially when an NN with multiple outputs is used [80].

3. **No Jacobian Matrix Calculation**

   Unlike the LM algorithm, with the NBN algorithm, there is no need to compute

and store the Jacobian matrix with a size that is proportional to the number of training patterns. This essentially means that the NBN algorithm can be applied to problems with unlimited patterns [66].

In the next section some basic concepts in NN training are presented, which will be used to describe the implementation of the NBN algorithm.

3.8 Basic Concepts in Neural Network Training

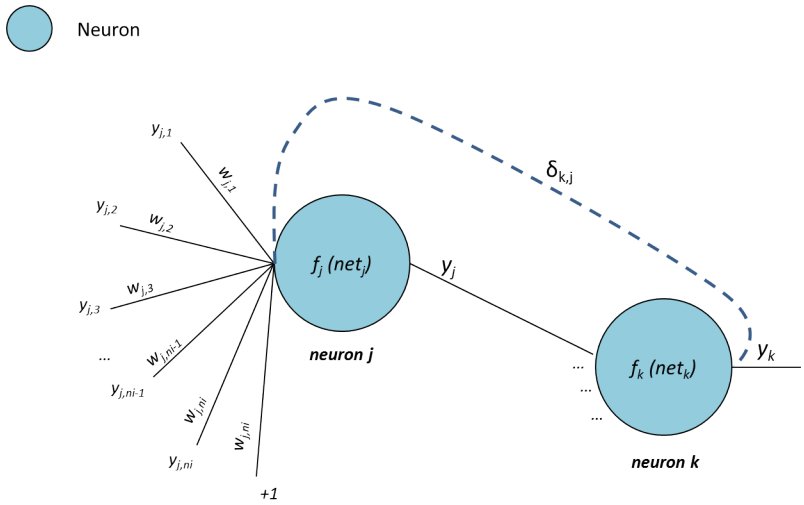


Figure 3.8: Concept of a neuron.

In Figure 3.8, a neuron j with ni inputs is depicted. If the neuron is in the first layer of the NN, all its inputs will be connected to the inputs of the NN. Otherwise, its inputs can be connected to the outputs of other neurons and to the NN inputs. In this figure, node y is used flexibly either side of the neuron j. It can be used with one index, y_j, to define the output node of neuron j. If two indices are used, $y_{j,i}$, it describes the i^{th} input node of neuron j. The output node of neuron j is calculated using

$$y_j = f_j(net_j) \tag{3.3}$$

where f_j it the activation function of neuron j and net_j is the sum of the weighted

input nodes (net input) of neuron j. The value of net_j is calculated as follows

$$net_j = \sum_{i=1}^{ni} w_{j,i} y_{j,i} + w_{j,0} \tag{3.4}$$

where $y_{j,i}$ is the i^{th} input of neuron j, weighted by $w_{j,i}$; and $w_{j,0}$ is the bias weight of neuron j. The slope s_j or the derivative of the activation function f_j is

$$s_j = \frac{\partial y_j}{\partial net_j} = \frac{\partial f_j(net_j)}{\partial net_j} \tag{3.5}$$

The elements of the Jacobian matrix can be calculated by

$$\frac{\partial e_m}{\partial w_{m,j}} = -y_{j,i}\delta_{m,j} \tag{3.6}$$

where $\delta_{m,j}$ is the signal gain between output neuron m and neuron j. In general, the signal gain between two neurons k and j, can be calculated as

$$\delta_{k,j} = \delta_{k,k} \sum_{i=j}^{k-1} w_{i,k}\delta_{i,j} \tag{3.7}$$

where $k \geq j$, $\delta_{k,k} = s_k$ is the slope of the activation function of neuron k, $w_{j,k}$ is the weight between neurons j and k, $\delta_{k,j}$ is the signal gain through $w_{j,k}$ and other part of the network connected to $w_{j,k}$.

Note that the training process of the NN is evaluated using the sum squared error (SSE) E. For all training patterns and network outputs, the SSE can be calculated using

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{m=1}^{M} e_{p,m}^2 \tag{3.8}$$

where $e_{p,m}$ is error at the neural network output m and is calculated using equation (3.2). In the next section, the implementation of the NBN algorithm is discussed.

3.9 Implementation of the NBN Algorithm

The implementation of the NBN algorithm can be divided into two main sections: forward propagation and quasi-Hessian matrix and gradient vector calculation. These sections are discussed below. Note that the derivation of the NBN algorithm is not presented in this chapter. Only the details required to implement the NBN for the FCC NN architecture are presented. Interested readers are recommended to read [1] for an introduction to the LM algorithm and [82] for the complete derivation of the NBN algorithm from the LM algorithm.

3.9.1 Forward Propagation

In forward propagation, for a pattern p, for each neuron in the NN, do the following:

1. Calculate the net input to the neuron using equation (3.4).
2. Calculate the output of the neuron using equation (3.3).
3. Calculate the slope of the neuron using equation (3.5).
4. Set the current slope of the neuron as the delta (i.e. $\delta_{k,k} = s_j$).
5. For simplicity, implement equation (3.7) in two steps:
    i. Multiply previous deltas through weights and sum.

$$x_{k,j} = \sum_{i=j}^{k-1} w_{i,j}\delta_{i,j} \tag{3.9}$$

    ii. Multiply this sum by the slope of the neuron.

$$\delta_{k,j} = \delta_{k,k}x_{k,j} = s_k x_{k,j} \tag{3.10}$$

At the end of this forward propagation process, the following variables will have been computed:

1. Output node values of each neuron. These output nodes are input to other neurons. Therefore, the output node values are stored in a vector $\boldsymbol{y}$, which stores the inputs to each of the neurons of the NN.

2. The slope of each of the neurons.

3. The delta values for the neurons in the NN. For example, in case of the NN presented in Figure 3.2, at the end of the forward propagation, the delta values for each of the neurons are:

   i. Neuron 1 (n_1): $\delta_{1,1}$

   ii. Neuron 2 (n_2): $\delta_{2,2}$, $\delta_{2,1}$

   iii. Neuron 3 (n_3): $\delta_{3,3}$, $\delta_{3,2}$, $\delta_{3,1}$

Note that for a pattern p and output neuron n_3 in Figure 3.2, the row elements of the Jacobian matrix can be calculated using the neuron 3 delta values and the $\boldsymbol{y}$ vector in equation (3.6). For example, the Jacobian row for pattern p and output neuron m can be described as

$$\boldsymbol{j}_{p,m} = \begin{bmatrix} \frac{\partial e_{p,1}}{\partial w_{1,1}} & \frac{\partial e_{p,1}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{p,1}}{\partial w_{j,1}} & \frac{\partial e_{p,1}}{\partial w_{j,2}} & \cdots \end{bmatrix} \tag{3.11}$$

Therefore using equation (3.6) and the delta values for neuron 3, the Jacobian row can be written as

$$\boldsymbol{j}_{p,m} = \begin{bmatrix} \delta_{3,1} \times \boldsymbol{y_1} & \delta_{3,2} \times \boldsymbol{y_2} & \delta_{3,3} \times \boldsymbol{y_3} \end{bmatrix} \tag{3.12}$$

where, $\boldsymbol{y_1}$, $\boldsymbol{y_2}$ and $\boldsymbol{y_3}$ are the neuron 1, 2 and 3 input vectors respectively.

3.9.2 Computation of the quasi-Hessian Matrix and Gradient Vector

In the LM algorithm, the entire Jacobian matrix must be calculated over all the patterns before the weights can be updated using equation (3.1). This has serious memory limitations, as the size of the Jacobian matrix increases with the number

of patterns (P). As mentioned earlier, the NBN algorithm is an improved version of the LM algorithm. It avoids the need to calculate and store the entire Jacobian matrix over all the patterns. To achieve this, the weight update rule for the NBN algorithm is as follows:

$$\boldsymbol{w_{n+1}} = \boldsymbol{w_n} - (Q + \mu I)^{-1}\boldsymbol{g} \tag{3.13}$$

where Q is the quasi-Hessian matrix and $\boldsymbol{g}$ is the gradient vector. This is just another form of the LM update rule [82] where

$$Q = J^T J \tag{3.14}$$

$$\boldsymbol{g} = J^T \boldsymbol{e} \tag{3.15}$$

The matrix Q is calculated by summing the quasi-Hessian sub-matrix $q_{p,m}$ for pattern p and network output neuron m:

$$Q = \sum_{p=1}^{P} \sum_{m=1}^{M} q_{p,m} \tag{3.16}$$

The gradient vector $\boldsymbol{g}$ is calculated by summing the gradient sub-vector $\boldsymbol{\eta_{p,m}}$ for pattern p and network output neuron m:

$$\boldsymbol{g} = \sum_{p=1}^{P} \sum_{m=1}^{M} \boldsymbol{\eta_{p,m}} \tag{3.17}$$

The size of the matrix Q is $C \times C$ and is independent of the number of patterns and outputs. Compared to the LM algorithm, the NBN algorithm calculates the matrix Q and vector $\boldsymbol{g}$ directly as the patterns are applied, there by removing the need to compute and store the Jacobian matrix (J) [82]. This is achieved by calculating the vector $\boldsymbol{j_{p,m}}$ as the patterns are applied. This vector is the Jacobian row for pattern p and network output neuron m. Using this vector, the matrix Q

and vector $\boldsymbol{g}$ can be updated as each pattern is applied using the following equations:

$$q_{p,m} = \boldsymbol{j}_{p,m}^T \boldsymbol{j}_{p,m} \tag{3.18}$$

$$\boldsymbol{\eta_{p,m}} = \boldsymbol{j}_{p,m} e_{p,m} \tag{3.19}$$

In essence, the entire computation of the quasi-Hessian matrix Q and gradient vector $\boldsymbol{g}$ is reduced to computing a vector $\boldsymbol{j_{p,m}}$ with C elements. The calculation of the $\boldsymbol{j_{p,m}}$ using equation (3.6) was shown in the forward propagation section. The matrix Q and vector $\boldsymbol{g}$ is directly calculated as the patterns are applied, without the need to store and multiply the Jacobian matrix J. The NBN algorithm with its forward propagation and the calculation of matrix Q and gradient vector $\boldsymbol{g}$ is presented in Figure 3.9.

3.10 The NBN Training Process

In the previous section, the implementation of the NBN algorithm was presented. This showed the calculations required to solve the weights update rule presented in equation (3.13). In this section, the NBN algorithm is organised into a repeatable training process which could be used to train the NN off-line (batch training) or on-line. This is similar to the LM training process. In general, the training process can be described as follows [1]:

1. Note that the NN is first initialised with the randomly generated weights.
2. Propagate the NN forward and calculate the total SSE E using equation (3.8).
3. Update the weights of the NN as directed by equation (3.13).
4. With the new weights, re-evaluate the total SSE E.
5. If the current SSE has increased compared to the previous SSE as a result of the weight update, then reset the weights to the previous values. Also, increase the combination coefficient μ by a factor of 10. Then repeat the process from step 3.

```
1: procedure INITIALIZATION(Q, g)
2:     Q ← 0
3:     g ← 0
4: end procedure
5:
6: for all patterns (p = 1 to p = P) do
7:     procedure FORWARD COMPUTATION
8:         for all neurons (nn) do
9:             for all weights of current neuron (j) do
10:                calculate net input (net_j)
11:            end for
12:            calculate neuron output
13:            calculate neuron slope (s_j)
14:            s_j = ∂Out_j(net_j) / ∂net_j
15:            set current slope as delta
16:            for weights to previous neurons (ny) do
17:                for previous neurons (nz) do
18:                    multiply delta through weights
19:                    then sum
20:                end for
21:                multiply sum by the slope
22:            end for
23:        end for
24:        for all outputs (m = 1 to m = M) do
25:            calculate error
26:        end for
27:    end procedure
28:
29:    procedure UPDATE(Q, g)
30:        for all outputs (m = 1 to m = M) do
31:            calculate vector j_{p,m}
32:            calculate sub matrix q_{p,m}
33:            calculate sub vector η_{p,m}
34:            Q = Q + q_{p,m}
35:            g = g + η_{p,m}
36:        end for
37:    end procedure
38: end for
39:
40: procedure IMPROVED LM TRAINING
41:    follow the LM algorithm training process
42:    update rule: W_{n+1} = W_n − (Q + μI)^{-1} g
43: end procedure
```

Figure 3.9: The Pseudo-Code for the NBN algorithm.

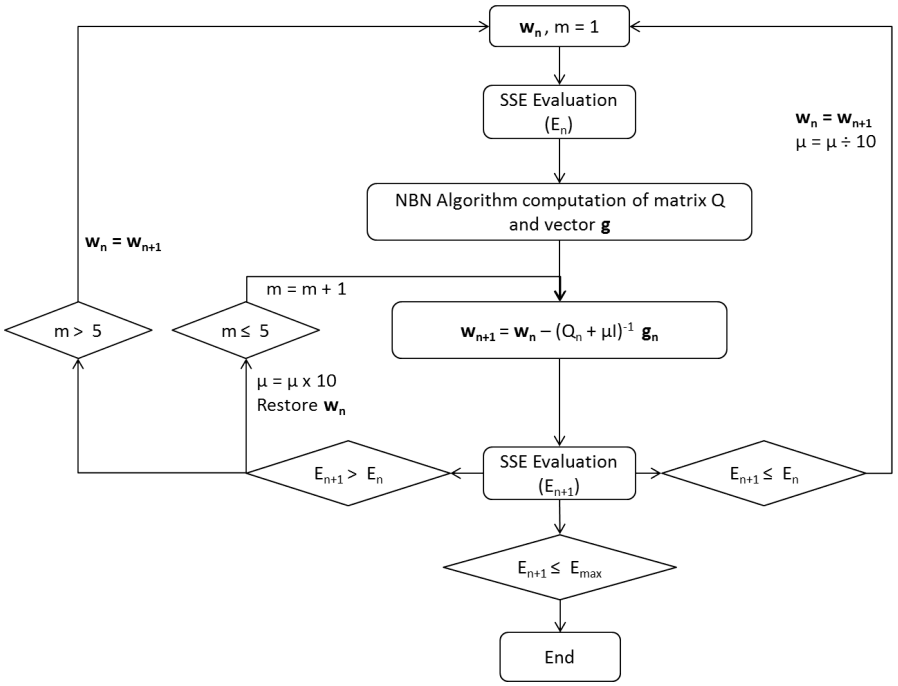


Figure 3.10: The NBN Training Process. This figure is adapted from [1].

6. If the current SSE has decreased compared to the previous SSE, due to the weights update, then keep the current weights and decrease the combination coefficient μ by a factor of 10.

7. Repeat the process from step 2 with the new weights, until the current total SSE is smaller than the required value.

In Figure 3.10, a flow chart for the training process is presented. In the next section, the general settings used for the development of the NN applications in this research is presented.

3.11 Neural Network Settings

In the research presented in this thesis, the NNs are initialised with random weights in the range of ± 1.5. The initial value of combination coefficient (μ), used in the weights update rule of the NBN algorithm, is set to 0.01. The activation function used by the neurons is the bipolar sigmoid [84], defined as follows:

$$Out_j = \frac{2}{1 + e^{-net_j}} - 1 \tag{3.20}$$

where net_j is the sum of the weighted inputs to neuron j and Out_j is the output of neuron j. This activation function produces an output in the range of ± 1.

Note that in the schemes presented in this research, the required output of the NNs developed might not be in the range of ± 1. In such cases, the required output is normalised to the range of ± 1 using the following

$$x_n = (b - a) \times \frac{x_o - x_{min}}{x_{max} - x_{min}} + a \tag{3.21}$$

where x_n is the normalised value and x_o is the value to be normalised. a and b are the minimum and maximum value of the range to be normalised to, which in this case is ± 1. x_{max} and x_{min} are the maximum and minimum values of the range from which x_o is been normalised.

3.12 Conclusion

The aim of this chapter was to provide an overview of the reasons behind choosing the FCC NN architecture and the NBN learning algorithm for the research. From the literature presented, it is clear that the FCC NN architecture is far more efficient than the popular MLP NN architecture. In addition, the NBN learning algorithm is an improved version of the LM algorithm. This algorithm is more effective compared to the popular EBP learning algorithm. Therefore, in the research presented in this thesis, the FCC NN trained using the NBN algorithm is used for the developed SFDIA and AFDIA schemes. In the next chapter, the developed SFDIA scheme is presented.

"A person who never made a mistake never tried anything new."
– Albert Einstein

Chapter 4

The SFDIA Scheme

4.1 Overview

Sensors are vital components for any control systems. They inform the controller about its environment and the state of the system. Any failure of the sensor could degrade the system's performance and possibly lead to total system failure. An aircraft system that can tolerate sensor failures would have added endurance in the presence of such failures. This chapter presents the development of the sensor failure, detection, identification and accommodation (SFDIA) scheme. The scheme utilises the fully connected cascade (FCC) neural network (NN). As described in Chapter 2, the SFDIA scheme can be divided into two stages:

- **Failure detection and identification (FDI)**: In this stage, a failure occurring or that has occurred is detected. Once the failure is detected, the failed sensor is then identified.

- **Failure accommodation (FA)**: In this stage, the failed sensor is replaced with a reliable alternative. In the case of the NN based SFDIA, the failed sensor is replaced with an NN based estimate.

The SFDIA scheme presented here can accommodate failure in the pitch, roll and yaw rate gyro sensors of an aircraft. This chapter is organised as follows: the outline of the developed SFDIA scheme is presented in Section 4.2. In Section 4.3, the aircraft simulation model and the sensor suite considered for this research are

presented. The development of the FCC NN based sensor estimators for the SFDIA is discussed in Section 4.4. During the development of the sensor estimators, some anomaly was observed with the pitch rate sensor, which is examined in Section 4.5. The setup of the SFDIA experiments and the results from the conducted experiments are discussed in Section 4.6. In Section 4.7, a summary of the results and related discussion is presented. Finally the chapter concludes in Section 4.8.

4.2 SFDIA Outline

A block diagram of the developed SFDIA scheme for the pitch, roll and yaw rate sensor is presented in Figure. 4.1. In this scheme, for every sensor considered, there is an NN based sensor estimator. As the name suggest, the output of this estimator is the sensor measurement it is estimating. Also associated with each sensor is a fault alarm signal (F_A), which could either be 0 or 1: where $F_A = 1$ indicates a fault and $F_A = 0$ if otherwise. Failure detection (FD) is performed by evaluating the residual between each sensor and its associated NN estimate. If the residual exceeds a certain threshold, the failure alarm for that sensor is triggered ($F_A = 1$). Failure identification (FI) is performed by identifying which sensor fault alarm is triggered. Once the failed sensor is identified, it remains in the failed state throughout the process. In other words, the sensor is assumed to remain faulty throughout. It does not recover from the fault.

In addition, the proposed scheme consists of a fault switch (F_S) for every sensor. The inputs to the fault switch are the fault alarm signal (F_A), sensor output and estimator output. This switch is controlled by the F_A signal. In fault free conditions ($F_A = 0$), the output of F_S is the sensor output. However, in the event of failure ($F_A = 1$), the F_S switches to the estimator output. This output is then used by the flight control system (FCS) to operate in a fault free state.

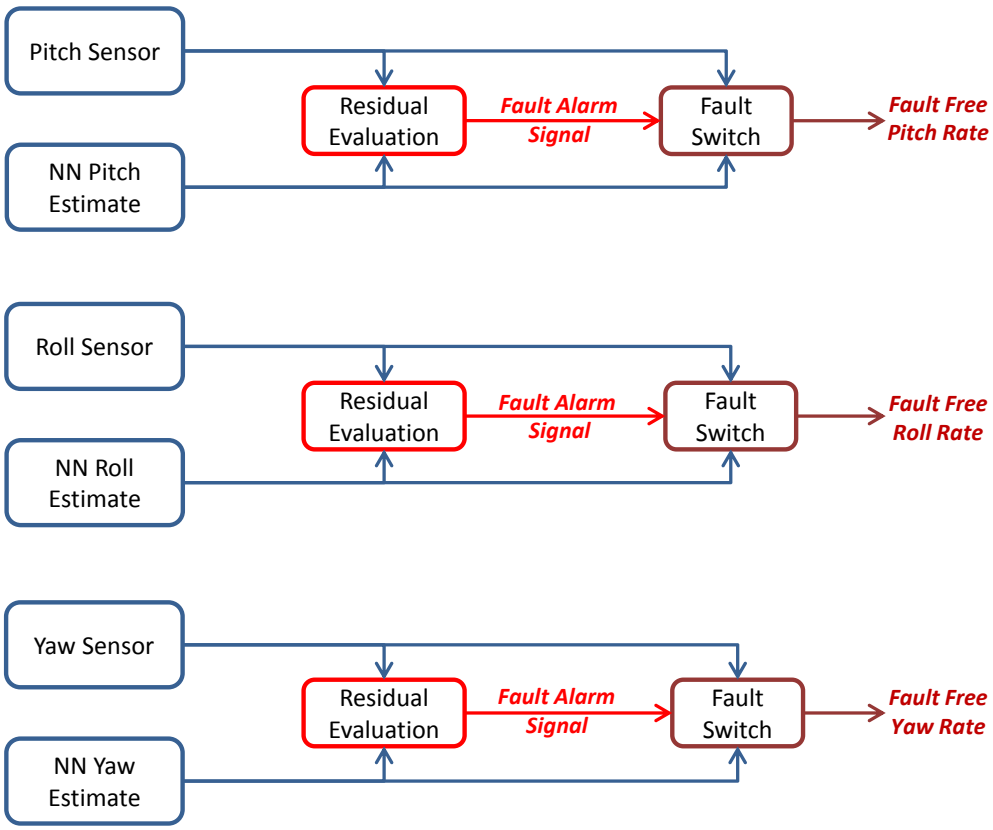


Figure 4.1: SFDIA scheme layout for pitch, roll and yaw rate sensors.

4.3 Aircraft Simulation and Sensor Suite

Aircraft data is collected using the X-Plane flight simulator [85]. For this research, the Cessna 172SP aircraft model in X-Plane is used to collect the flight data. This aircraft is flown by the artificial intelligence (AI) pilot in X-Plane. The aircraft is assumed to be equipped with six inertial sensors without any hardware redundancy. The inertial sensors are three gyroscopes (gyros) and three accelerometers. They are mounted along the x, y and z axis of the aircraft. These sensors are essential components of the attitude/heading reference system (AHRS) and the initial navigation system (INS) found in modern aircraft [86, 87]. The outputs of these sensors are as follows:

1. Gyros: pitch (q), roll (p) and yaw (r) rates.

2. Accelerometers: accelerations along the x (a_x), y (a_y) and z (a_z) axis.

Figure 4.2: The Cessna aircraft model in X-Plane 9.

4.4 Estimator Development

In this section, the structure of the NN based sensor estimators is presented. This is followed by the development of the NN based pitch, roll and yaw rate estimators.

4.4.1 Estimator NN Inputs, Outputs and Structure

The conducted research concentrates on the SFDIA of the gyro sensors. Therefore, three gyro sensor estimators are developed, one for each of the pitch, roll and yaw rate gyros. The outputs of these estimators are their respective estimated sensor rates. The inputs to the estimators are measurements from other sensors, excluding the one it is estimating. In addition, commanded control outputs by the flight control computer are also used as inputs. These inputs are taken at the current sample time t. Inputs to each of these estimators and their respective outputs are presented in Table 4.1.

These inputs are chosen because they can have an effect or can cause an effect on the parameter that the sensor is measuring. The relationship between the measured accelerations and the gyro rates can be derived from the linear acceleration

Table 4.1: Inputs to the sensor estimators.

Sensor Estimator	Inputs
Pitch (q)	a_z - Normal Acceleration a_x - Longitudinal Acceleration δ_E - Elevator Demand
Roll (p)	r - Yaw Rate δ_A - Aileron Demand δ_R - Rudder Demand
Yaw (r)	a_y - Lateral Acceleration δ_A - Aileron Demand δ_R - Rudder Demand

AXIS	Linear Velocity Component	Angular Velocity Component
Forward or Roll Axis, 0X	Forward Velocity U	Roll Rate p
Sideslip or Pitch Axis, 0Y	Sideslip Velocity V	Pitch Rate q
Vertical or Yaw Axis, 0Z	Vertical Velocity W	Yaw Rate r

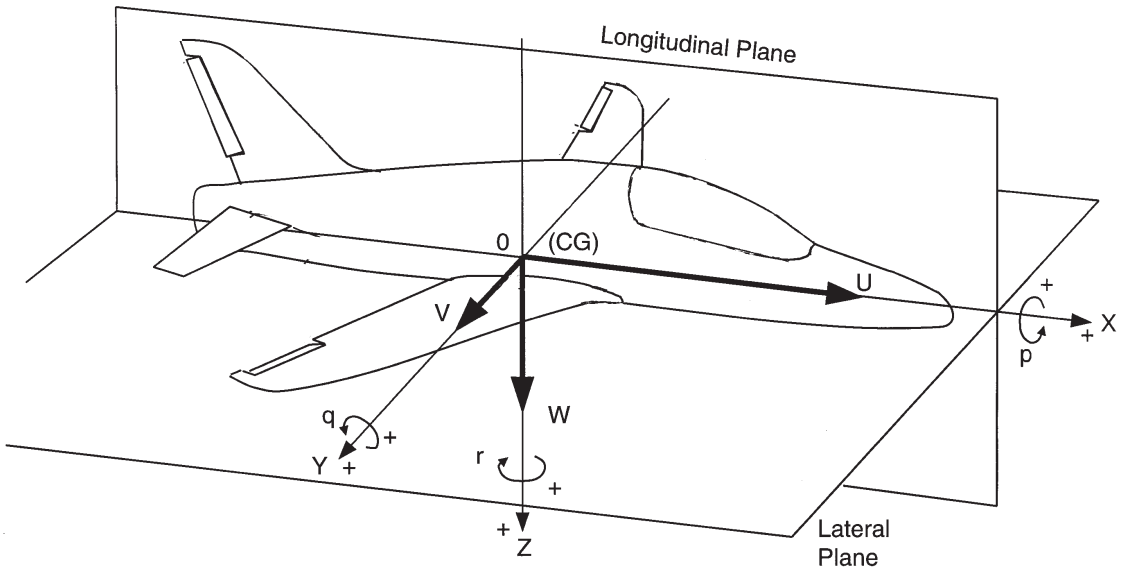


Figure 4.3: Aircraft X, Y and Z axis. This figure is adapted from [2].

equations [88], defined as follows:

$$\begin{aligned} a_x &= \dot{U} - rV + qW + g_x \\ a_y &= \dot{V} - pW + rU + g_y \\ a_z &= \dot{W} - qU + pV + g_z \end{aligned} \tag{4.1}$$

where (U, V, W) and (g_x, g_y, g_z) are the velocity and gravitational acceleration com-

ponents respectively, along the X, Y and Z axes, given in body fixed reference frame. The relationships between the remaining inputs and outputs of the estimators can be derived analytically from the aircraft's equations of motion described in [88].

To select the best structure (topology) for the FCC NN based sensor estimators, the number of neurons in each estimator is first explored experimentally; varying from 2 to 12 neurons. These estimators with different numbers of neurons are trained and validated using the process described in the following sections.

4.4.2 Training and Validation Data Sets

To train and evaluate the estimators, flight data from the Cessna 172SP aircraft in X-Plane is recorded for six different flight scenarios. In these scenarios, the aircraft takes off from different airports to capture different manoeuvres performed by the AI pilot in X-Plane. The manoeuvres include take-off, straight flight and randomly changing flight heading. These scenarios were simulated in turbulence-free weather conditions. Flight data was recorded once the aircraft reached its cruise altitude. These flight data contain various sensor readings and control inputs, recorded at every second.

In a practical system, sensor readings are updated at a higher frequency. In this case however, recording the flight data at every second allows the training data to capture more dynamic flight characteristics between each training sample. This helps prevent the NN estimator from over-fitting to less dynamic training data. Out of the six scenarios, one of them is randomly selected to train the NN based estimators for each of the three sensors. The remaining five are used for validating the estimators.

4.4.3 Estimator Training

For each sensor considered, estimators with 2 to 12 neurons are trained off-line (batch learning) using a fixed set of training data extracted from the training flight data mentioned in the previous section. The training set consists of data collected during the steady and transient states of flight. This ensures that the estimators

can produce good estimates during any state of flight. The estimators are trained until the Sum Squared Error (SSE) of the epoch is ≤ 0.01 or a maximum of 101 epochs is reached.

4.4.4 Simulation for Validation

Once trained, each of the estimators (ranging from 2 to 12 neurons) for a sensor is validated on the five different flight scenarios. These scenarios last for 1500 seconds, therefore containing 1500 samples. To assess the performance of the estimator on the scenario, the total SSE of all the samples in the scenarios is computed. Finally, the best estimator for a sensor is selected by calculating the average and the standard deviation of the SSE for all the scenarios.

4.4.5 Results and Discussion

Yaw Rate Estimator

The SSEs of the yaw rate estimators using different neurons on the five validation scenarios are presented in Table 4.2. From the average SSE, estimator networks with 2 and 5 neurons produced the least errors. For the estimator with 2 neurons, the SSE is 0.03290, while the SSE with 5 neurons is 0.03637. Using the standard deviation (σ), it is clear that the estimator with 2 neurons is best of the two, with $\sigma = 0.01979$. The output of this estimator on its best and worst scenarios are presented in Figure 4.4 and Figure 4.5 respectively. The best scenario is scenario 3 and the worst is scenario 1.

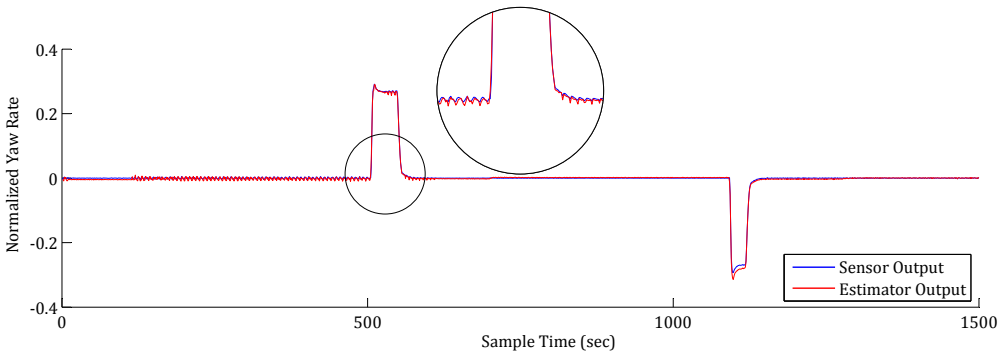


Figure 4.4: Normalised yaw rate using equation (3.21). Results using 2 neurons in scenario 3.

Table 4.2: Yaw rate estimator errors for the validation scenarios.

	SUM SQUARED ERRORS (SSE)										
Neurons	**2**	3	4	5	6	7	8	9	10	11	12
Scenario 1	**0.05403**	0.05790	0.05775	0.04916	0.14770	1.08820	0.08818	0.14520	0.41494	0.22531	0.74185
Scenario 2	**0.01678**	0.03428	0.02443	0.02183	0.23910	2.11596	0.13171	0.29886	1.26390	0.36817	1.23907
Scenario 3	**0.00923**	0.04650	0.03638	0.01692	0.25249	5.99460	0.25252	0.91540	2.30973	0.99688	3.57601
Scenario 4	**0.05017**	0.10001	0.04895	0.06144	0.11770	0.69716	0.08022	0.05969	0.27662	0.07500	0.47789
Scenario 5	**0.03428**	0.05799	0.04082	0.03250	0.44971	3.44615	0.35203	0.39369	2.05501	0.39984	2.04609
Average Error	**0.03290**	0.05934	0.04167	0.03637	0.24134	2.66841	0.18093	0.36257	1.26404	0.41304	1.61618
SD (σ)	**0.01979**	0.02475	0.01262	0.01869	0.13000	2.14222	0.11786	0.33528	0.92399	0.35091	1.24781

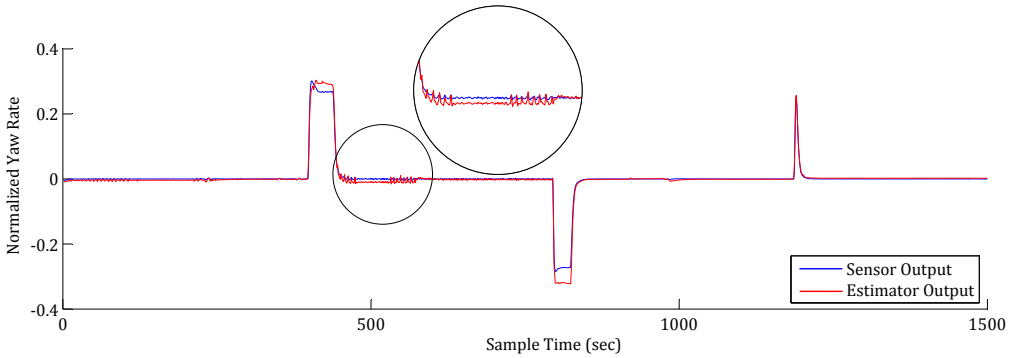


Figure 4.5: Normalised yaw rate using equation (3.21). Results using 2 neurons in scenario 1.

Pitch Rate Estimator

Table 4.3 presents the validation results for the pitch rate estimators. The estimator using 6 neurons has the least SSE among them, with SSE = 1.15978. The output of this estimator on its best and worst scenarios are presented in Figure 4.6 and Figure 4.7 respectively. The best scenario is scenario 4 and the worst is scenario 3.

As can be seen from Figure 4.6 and Figure 4.7, the output of the pitch rate sensor seems to oscillate rapidly over certain time frames. This is due to the aircraft being disturbed from its equilibrium state. These disturbances could be initiated by pilot control inputs, change in power settings and atmospheric influences such as gust and turbulence [89]. Since the scenarios were simulated in turbulence free weather conditions, in this case, the oscillations are caused by the control outputs from the AI pilot in X-Plane.

For a certain time frame, the oscillations are neither increasing nor decreasing in magnitude (see Figure 4.7). Once the aircraft is disturbed, it continues to oscillate without a significant increase or decrease in magnitude. The aircraft is said to be in a state of neutral dynamic stability [88, 90]. The magnitude and duration of these oscillations depends on the aircraft's aerodynamics and stability. The estimator follows these oscillations but does not follow the magnitude. This anomaly is further investigated in Section 4.5.

Table 4.3: Pitch rate estimator errors for the validation scenarios.

	SUM SQUARED ERRORS (SSE)										
Neurons	2	3	4	5	**6**	7	8	9	10	11	12
Scenario 1	0.87549	1.07204	0.77152	0.75686	**0.62664**	8.31579	1.07375	1.73602	1.60933	1.80456	4.34395
Scenario 2	1.30509	1.19663	1.06827	0.97039	**0.92628**	5.95501	1.15038	1.79250	2.21795	2.63000	6.52519
Scenario 3	3.09023	2.83352	2.80383	2.70440	**2.74835**	8.32024	2.88546	2.97336	3.03034	4.02907	6.08809
Scenario 4	0.64407	0.58858	0.46337	0.37710	**0.34452**	8.91346	0.50092	1.05716	1.20740	3.24782	4.86670
Scenario 5	1.78657	1.59820	1.35882	1.32644	**1.15308**	13.43923	1.87371	2.78344	2.85945	2.95253	7.94092
Average Error	1.54029	1.45779	1.29316	1.22704	**1.15978**	8.98874	1.49684	2.06850	2.18489	2.93280	5.95297
SD (σ)	0.96998	0.84925	0.90797	0.89468	**0.93901**	2.73477	0.91668	0.79678	0.78385	0.81640	1.42008

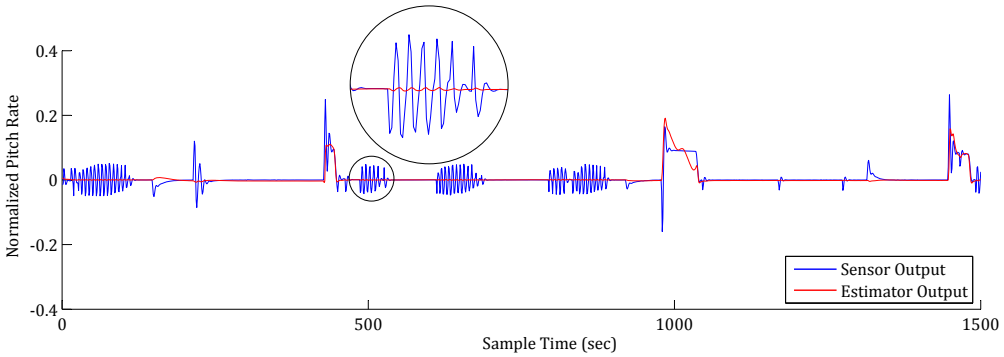


Figure 4.6: Normalised pitch rate using equation (3.21). Results using 6 neurons in scenario 4.

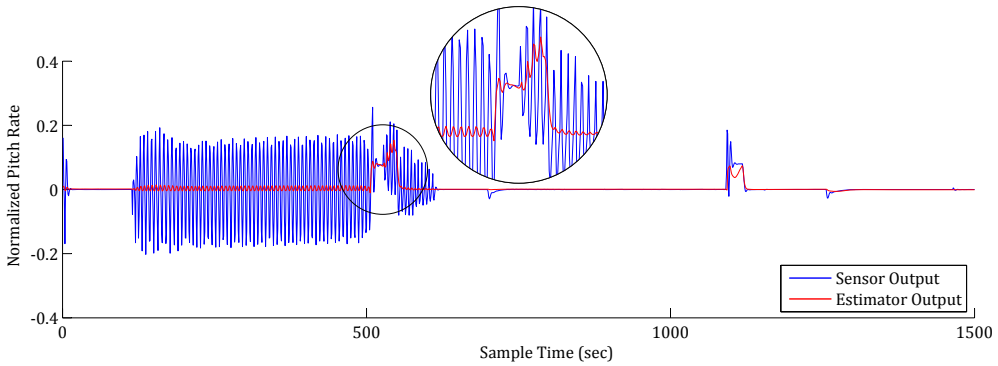


Figure 4.7: Normalised Pitch rate using equation (3.21). Results using 6 neurons in scenario 3.

Roll Rate Estimator

In Table 4.4, the results for the roll rate estimators are presented. As can be seen from the average SSE, there is a close tie between 2, 4 and 12 neurons, with SSE = 0.83437, 0.83610 and 0.83129, respectively. Using the standard deviations of their SSE, the estimator with 12 neurons has the best normal distribution among them (σ = 0.25250). However, it was decided to select the estimator with 4 neurons, keeping in line with the low neuron count of the previous gyro sensor estimators. The output of this estimator on its best and worst scenarios is presented in Figure 4.8 and Figure 4.9 respectively. The best scenario for this estimator is scenario 3 and the worst is scenario 5.

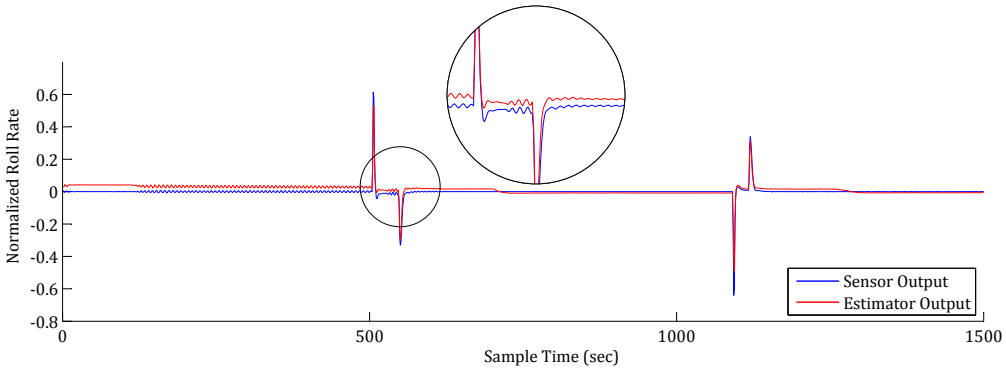


Figure 4.8: Normalised roll rate using equation (3.21). Results using 4 neurons in scenario 3.

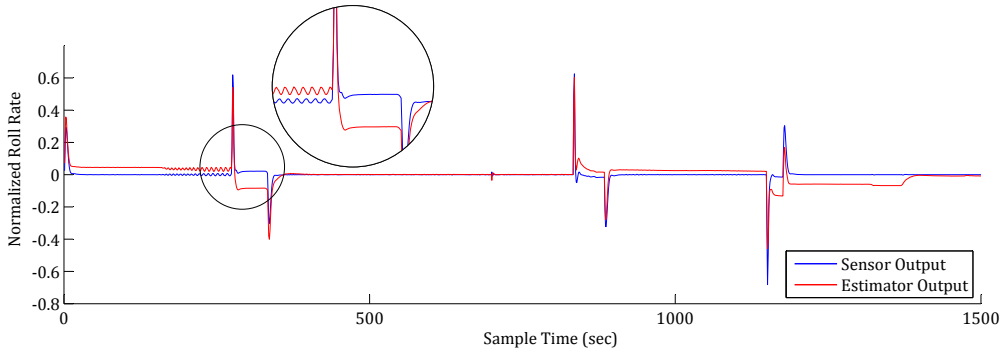


Figure 4.9: Normalised roll rate using equation (3.21). Results using 4 neurons in scenario 5.

Table 4.4: Roll rate estimator errors for the validation scenarios.

SUM SQUARED ERRORS (SSE)											
Neurons	2	3	**4**	5	6	7	8	9	10	11	12
Scenario 1	1.03861	1.71942	**0.99188**	1.26806	1.17937	1.23314	1.33951	1.06159	1.12756	1.03996	0.95751
Scenario 2	0.87189	0.71655	**0.62950**	0.74557	0.73817	0.80793	0.83962	0.90883	1.00083	0.75335	0.72139
Scenario 3	0.33534	0.48925	**0.41012**	0.44700	0.42922	0.43207	0.45478	0.64603	0.68607	0.56293	0.44613
Scenario 4	0.50070	0.92344	**0.82084**	1.07419	0.98542	1.01214	1.05378	1.36040	1.40734	1.21942	0.94279
Scenario 5	1.42533	1.46552	**1.32818**	1.53932	1.41908	1.58002	1.60261	1.20004	1.22841	1.27567	1.08864
Average Error	0.83437	1.06284	**0.83610**	1.01483	0.95025	1.01306	1.05806	1.03538	1.09004	0.97027	0.83129
SD (σ)	0.43380	0.51517	**0.35028**	0.42944	0.38405	0.43287	0.44369	0.27434	0.27036	0.30544	0.25250

4.5 The Pitch Rate Anomaly

As mentioned in Section 4.4.5, there is an anomaly with the pitch rate estimator. Although for the majority of the validation datasets, the pitch estimator with 6 neurons follows the oscillatory phase of the sensor output, the estimator it is not estimating the correct magnitudes. To address this anomaly, three main factors were considered:

- Sampling frequency
- Training data
- Estimator inputs

These factors are investigated below.

4.5.1 Sampling Frequency

With a sampling frequency of 1 Hz, it is possible that during the pitch rate oscillations important data could have been lost between samples. These lost data points could have helped train the estimators in matching the oscillation magnitude.

To investigate this cause, the validation dataset in scenario 3 was examined closely. Scenario 3 was chosen because, as indicated in Section 4.4.5 and Table 4.3, this scenario provides the worst results among the five scenarios. It was noted that the period of the pitch rate oscillation was ≈ 9 sec. If the period of the oscillations was less than 1 sec, it is possible for important information to be lost during training. Therefore, it is fair to conclude that increasing the sampling frequency would not have captured valuable data, which could have helped during the training of the estimators.

To confirm this further and rule out sampling frequency as a cause for the pitch rate anomaly, two flight datasets were collected from X-Plane at a sampling frequency of 10 Hz. One of the datasets was used to train the pitch rate estimators and the other to validate it. An experiment, similar to the one in Section 4.4, was conducted. However in this case, there is only one validation scenario.

From the results (see Figure 4.10), it is concluded that increasing the sampling frequency from 1 Hz to 10 Hz would not resolve the pitch rate anomaly.

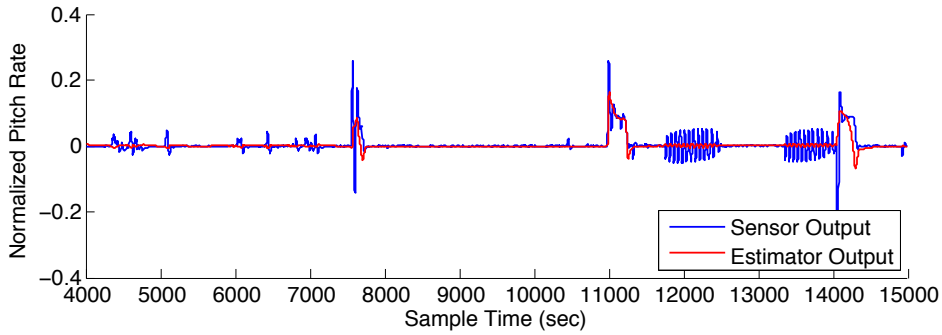


Figure 4.10: Normalised pitch rate using equation (3.21). Results using 5 neurons estimator trained using 10 Hz sampling frequency data.

4.5.2 Training Data

Training data that is incomplete or an insufficient representation of the problem could also be a good reason for the pitch rate anomaly. To eliminate training data as a cause of the pitch rate anomaly, the estimators must be trained using new training data, as conducted in Section 4.5.1. The new training data, collected at 10 Hz sampling frequency, is used to train the pitch rate estimators. These estimators are then validated on a scenario. As evident from Figure 4.10 and Table 4.5, changing the training data cannot resolve the anomaly observed in the pitch rate estimator.

It is concluded that the training data is not the cause of the pitch rate estimator failing to match the magnitude of the oscillatory pitch sensor output.

4.5.3 Inputs to Estimators

With the previous two factors excluded, it is concluded that the inputs to the pitch rate estimators are the cause for the anomaly. Having different and/or additional inputs could help capture additional information. This additional information could help in matching the magnitude of the oscillations.

Future work would focus on identifying other sensor suites that could be used to correct the anomaly observed in the pitch rate estimator. At the present stage,

Table 4.5: Pitch rate estimator errors using training data sampled at 10 Hz.

	SUM SQUARED ERRORS (SSE)										
Neurons	2	3	4	5	6	7	8	9	10	11	12
Scenario 1	6.77268	20.43585	6.64963	6.21719	11.46268	7.35264	12.74987	9.56275	9.01353	1501.12553	12.16450

since the majority of the pitch rate estimator outputs follows the pitch rate sensor values, the estimator is used as part of the SFDIA scheme. The pitch rate estimator anomaly can be accommodated in the failure detection stage, using a high residual threshold. This is discussed in greater detail in Section 4.6.3.

4.6 Sensor Failure Experiments

With the NN based sensor estimators developed, the next objective is the application of the estimators for SFDIA. In this section, the setup of the SFDIA experiments, the types of failures and the results from the experiments are presented.

4.6.1 Failure Detection and Identification Experiment Setup

The X-Plane 9 flight simulator does not support simulation of sensor faults. Therefore, faults have to be introduced manually once the flight data for a simulation is collected. To examine the performance of the proposed SFDIA scheme, the five validation scenarios used in Section 4.4 are reused. Faults are introduced manually at random locations into these five scenarios.

For every fault type considered, the faults are simulated on each of the scenarios for every sensor. This would allow the examination of the performance of the SFDIA scheme for each of the fault types in each sensor. The fault types considered in this research are discussed in the following section.

4.6.2 Sensor Failure Types

Aircraft sensors can fail in several ways. Some failures are specific to a sensor, while others are general. The signal from a sensor could be described as follows [48,91,92]:

$$x_t = r_t + n_t + f_t \tag{4.2}$$

where at time t, x is the signal from the sensor, r is the useful signal, n is the noise and f is the sensor failure. The sensor data collected from X-Plane provides the

values for r and n. The f signal is injected manually for each fault type. In this research, the following fault types are considered [12, 92–94]:

- *Stuck constant bias failure*: At a given time, the sensor output gets stuck and outputs a constant bias (b).

$$x_t = b \tag{4.3}$$

- *Additive (drift) failure*: This type of failure is very common. It is usually caused by temperature changes or calibration problems. In this fault, a constant term (drift value) is added to the sensor output. An additive fault can be modelled by the ramp function, as follows [12, 48, 93]:

$$f(t) = \begin{cases} 0 & t < t_f \\ A(t - t_f)/T_R & t_f \leq t < t_f + T_R \\ A & t \geq t_f + T_R \end{cases} \tag{4.4}$$

  where t_f is the time when the fault is introduced, T_R is the duration of the ramp and A is the fault magnitude. The magnitude (A) of the additive fault can either be large or small. Depending on the duration of the ramp (T_R), the fault can be a step ($T_R \approx 0$ sec), soft ($T_R = 4$ sec) or hard ($T_R = 1$ sec) in nature [37, 93].

In this research, the outputs of the gyro sensors are assumed to be in the range of +10 deg/sec to −10 deg/sec. In the case of the additive fault type, large and small fault magnitudes are modelled using 30% and 15% of the maximum sensor value, respectively. In other words, for large faults, $A = 3$ deg/sec and for small faults, $A = 1.5$ deg/sec. In total, seven failure cases are considered, which can be summarised as follows:

1. Constant Bias
2. Hard Additive Large ($T_R = 1$ sec , $A = 3$ deg/sec)

3. Hard Additive Small ($T_R = 1$ sec , $A = 1.5$ deg/sec)

4. Soft Additive Large ($T_R = 4$ s , $A = 3$ deg/sec)

5. Soft Additive Small ($T_R = 4$ sec , $A = 1.5$ deg/sec)

6. Step Additive Large ($T_R = 0$ sec , $A = 3$ deg/sec)

7. Step Additive Small ($T_R = 0$ sec , $A = 1.5$ deg/sec)

In the next section, the technique to generate the sensor residual is discussed.

4.6.3 Residual Generation Technique

As described earlier, the SFDIA scheme presented here uses residuals d to detect and identify sensor failures. Generally, residuals are generated by squaring the difference between the real sensor measurement and the measurement from its model [37] as shown in equation (4.5). In equation (4.5), d is the residual at time t where x is the real sensor measurement and $\overline{x}$ is the estimator (model) measurement at time t.

$$d_t = (x_t - \overline{x}_t)^2 \tag{4.5}$$

Failure is detected when the residual d goes over a threshold (τ). Ideally the sensor measurement and the estimator output should be equal, therefore generating a residual $d = 0$, and $d \neq 0$ in case of failure. When the residual d crosses τ, the failure alarm is triggered. In this ideal condition, τ should be kept close to 0 for quick detection ($\tau \approx 0$).

However, in a practical system the sensor measurements are not equal to the estimator output due to sensor noise and modelling inaccuracies. This means that the residual d is not equal to 0 in fault free conditions. For this reason, in the absence of any faults, a false alarm ($F_A = 1$) could occur frequently when threshold $\tau \approx 0$. This could be resolved by raising the value of τ, but this risks not detecting the faults. Because of this, a balance between false alarms and fault detection is required.

In the proposed SFDIA scheme, the residual (d) is generated using a sliding window mechanism [37, 95]. In this mechanism, a window of size n data points keeps moving (sliding) with time. The window calculates the moving average of the n residuals calculated using equation (4.5). The result of the sliding average window is then weighted to produce the current residual [37]. The residual generation mechanism can therefore be described as follows:

$$D_t = \frac{\varpi}{n} \sum_{i=t-n-1}^{t} (x_i - \overline{x}_i)^2 \tag{4.6}$$

where D is the residual at time instant t and ϖ is the weight. Notice how equation (4.5) is utilised in equation (4.6). The sliding average window filters the residuals using equation (4.5) from noise and modelling inaccuracies. The weight allows us to magnify the residuals and have a high fault threshold (τ). In this research, the size of the sliding window is set to 5 ($n = 5$) and the weight is set to 40 ($\varpi = 40$). For the pitch, roll and yaw rate sensors, the threshold is set to $\tau = 0.8$, $\tau = 0.8$ and $\tau = 0.2$, respectively.

In the next section, the results of the sensor failure detection and identification for accommodation experiments are presented.

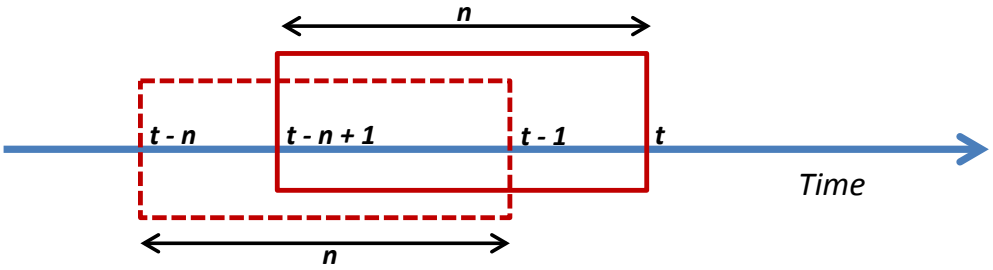


Figure 4.11: Sliding average window at time t.

4.6.4 Experiment Results

Yaw Sensor Failures

The results for the yaw rate sensor fault detection time are presented in Table 4.6. Generally, large magnitude faults are quicker to detect than small magnitude faults.

The greater the magnitude of the fault, the sooner the residual generated using equation (4.6) will cross the threshold τ. This observation is reflected in the results presented in Table 4.6, which compares the results for a sudden step fault of large and small magnitude. On average, the step fault of large magnitude is detected instantaneously compared to an average of 0.8 sec in sample time for the small magnitude step fault.

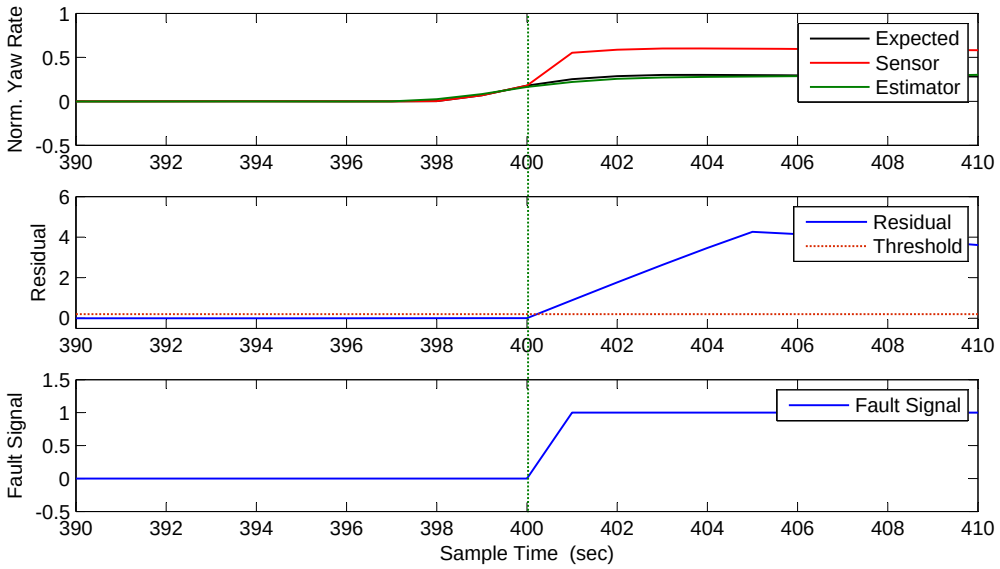
Similar results can be observed with the hard additive type faults with a ramp duration of $T_R = 1$ sec (see Section 4.6.2). Although the detection time is affected by the magnitude of the fault, it is also affected by the transient phase (ramp duration T_R) of the developing fault. Due to this, the detection time for hard faults is greater than that for step type additive faults.

In comparison to the step and hard additive type faults, soft faults have the longest detection time. These faults have the highest ramp duration ($T_R = 4$ sec) amongst the three types of additive fault. On average, the detection time for soft faults of large magnitude is 2.6 sec, in comparison to an average of 4 sec (sample time) for small magnitude fault. In the case of the constant bias fault, the average detection time is 1.6 sec.

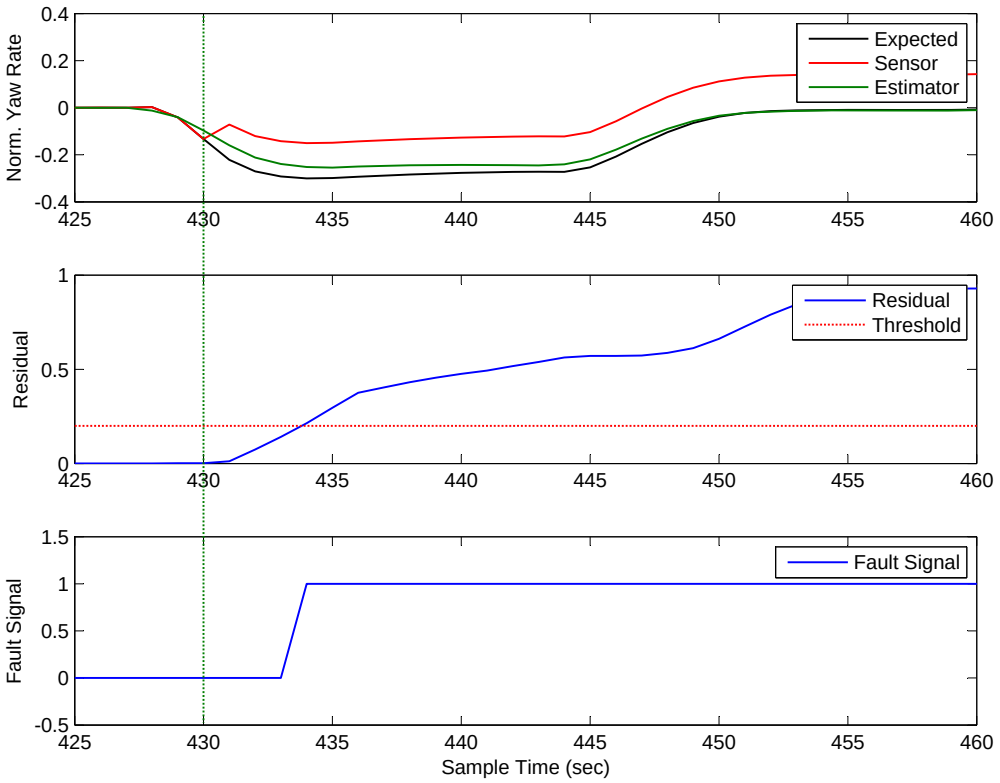
In Figure 4.12, the signals associated with the yaw rate sensor during the occurrence of a hard fault are presented. Figure 4.12a shows the response of various signals during the occurrence of a hard fault of large magnitude in scenario 1. In Figure 4.12b, the responses of various signals during a hard fault of small magnitude in scenario 4 are shown. Notice the response of the fault signal F_A in both cases after the occurrence of the fault. The time of fault is marked by the vertical green line running across the three plots.

Pitch Sensor Failures

The results for the pitch rate sensor fault detection time are presented in Table 4.7. The results reflect the observations made in the yaw rate sensor results. Large magnitude faults are quick to detect and additive faults with a ramp duration $T_R > 0$ sec take a longer time to detect. On average, the hard additive faults with small mag-



(a) Scenario: 1, Magnitude: Large, Fault Occurrence: 400 sec



(b) Scenario: 4, Magnitude: Small, Fault Occurrence: 430 sec

Figure 4.12: Yaw sensor hard fault simulations.

Table 4.6: Yaw FDI Results

	Detection Time for Fault Types in Sample Time (sec)					
Scenario	Const.Bias	Hard		Step		Soft	
-	-	L	S	L	S	L	S
1	1	1	1	0	0	2	3
2	1	1	2	0	1	3	5
3	1	1	2	0	1	2	3
4	2	1	3	0	2	3	5
5	3	1	2	0	0	3	4
Average	1.6	1	2	0	0.8	2.6	4

− : No Fault Detected, Threshold (τ) : 0.2, L : Large, S : Small

nitude are detected in 3 sec. In comparison, the hard faults with small magnitude are detected in an average of 1.4 sec.

Compared to the hard faults, the soft additive faults take on average 3 sec and 5.2 sec in sample time, for large and small magnitude respectively. Notice that the average detection time of soft faults is longer than that of hard faults. This is because the ramp duration is greater for soft faults, which is set at $T_R = 4$ sec, instead of $T_R = 1$ sec for hard faults. The step fault type has the lowest average of the additive fault types due to the ramp duration of $T_R = 0$ sec. Step faults with small and large magnitude have an average of 0.8 sec and 2.6 sec respectively. The constant bias fault type has an average of 0.8 sec.

Comparing Table 4.6 and Table 4.7 shows how the average detection time for the pitch rate sensor is greater than that of the yaw rate sensor, especially for the additive fault types. This is due to the higher fault residual threshold τ used for the pitch rate sensor. In comparison to the yaw rate sensor, the pitch rate estimator has a higher modelling error, as discussed in Section 4.5, therefore requiring a higher value for τ. The threshold τ is set to 0.8 for the pitch sensor whereas for the yaw sensor, $\tau = 0.2$. The modelling errors are reflected on the average SSE of the pitch and yaw rate estimators presented in Table 4.3 and Table 4.2 respectively.

In Figure 4.13, the signals associated with the pitch rate sensor during the occurrence of a step fault are presented. Figure 4.13a and Figure 4.13b show the response

of various signals during the occurrence of a step failure of large and small magnitude respectively. Note that Figure 4.13a presents the response of various signals in scenario 3 during a step fault. This is the scenario in which the pitch rate anomaly is significant. Although the anomaly between the sensor and estimator is significant, there is no false fault detection.

Table 4.7: Pitch FDI Reults

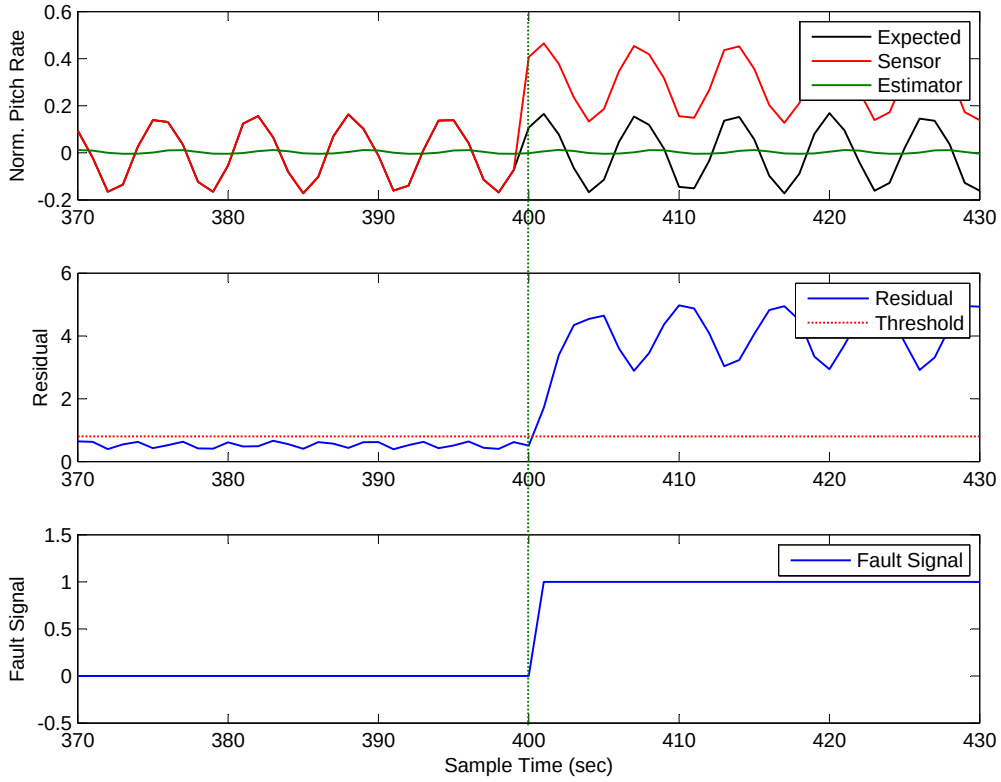
	Detection Time for Fault Types in Sample Time (sec)						
Scenario	Const.Bias	Hard		Step		Soft	
-	-	L	S	L	S	L	S
1	1	2	5	1	4	4	7
2	1	1	3	1	3	4	7
3	1	1	1	0	0	1	2
4	1	2	4	1	2	4	7
5	0	1	2	1	4	2	3
Average	0.8	1.4	3	0.8	2.6	3	5.2

$-$: No Fault Detected, Threshold (τ) = 0.8, L : Large, S : Small

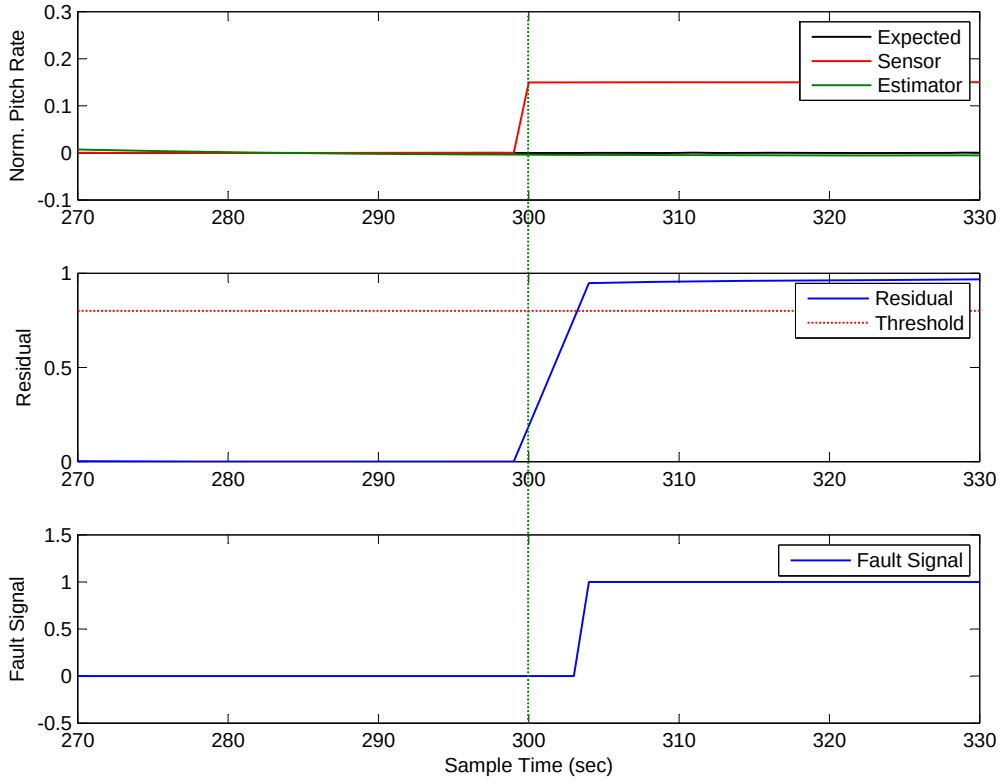
Roll Sensor Failures

In Table 4.8, the results for the roll rate sensor fault detection are presented. Similar to the pitch rate sensors, τ is set at a higher value: $\tau = 0.8$. This is to accommodate the difference between the estimator value and the sensor value. The least detection time is taken by the constant bias fault type with an average of 1 sec in sample time. For hard fault types the average is 2 sec and 4.6 sec in sample time for large and small magnitude. As expected, due to higher residual threshold, the fault detection time is longer.

The soft fault types take the most amount of time to be detected. For large magnitude soft faults, the average detection time is 4 sec in sample time. The average detection time is even higher for small magnitude soft failures, standing at an average of 7 sec in sample time. These results are considerably higher than the detection time in the yaw rate sensor. The longer detection time is caused by the higher residual threshold.



(a) Scenario: 3, Magnitude: Large, Fault Occurrence: 400 sec



(b) Scenario: 1, Magnitude: Small, Fault Occurrence: 300 sec

Figure 4.13: Pitch sensor step fault simulations.

In the case of the step type failures, the average is at 1.2 sec in sample time for large magnitude. However for the small magnitude, the average is at 3.75 sec with a fault going undetected in scenario 1. This is quite possible in scenarios where the magnitude of the fault is relatively small. The fault went undetected because the residual failed to trigger the threshold. This could be solved by reducing the threshold τ, but this risks false fault detection. Future work would consider additional inputs to the roll rate estimator to improve the estimate, and therefore improve the chances for detection.

In Figure 4.14, the signals associated with the roll rate sensor during the occurrence of a soft failure are presented. Figure 4.14a and Figure 4.14b, shows the response of various signals during the occurrence of soft failures of large and small magnitude respectively. Notice how the residual signal slowly rises over the threshold. This causes a delay in fault detection, as is evident from the fault signal response.

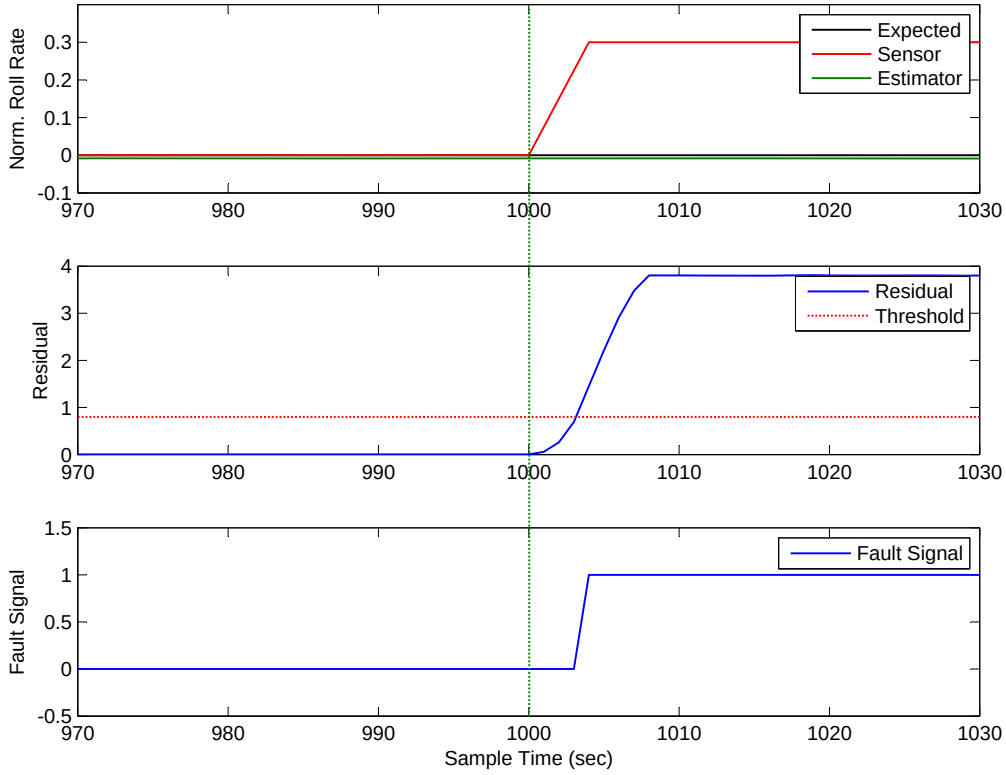
Table 4.8: Roll FDI Results

	Detection Time for Fault Types in Sample Time (sec)						
Scenario	Const.Bias	Hard		Step		Soft	
-	-	L	S	L	S	L	S
1	1	2	5	2	–	4	7
2	1	2	4	1	4	4	7
3	1	2	4	1	3	4	7
4	1	2	5	1	4	4	7
5	1	2	5	1	4	4	7
Average	1	2	4.6	1.2	3.75	4	7

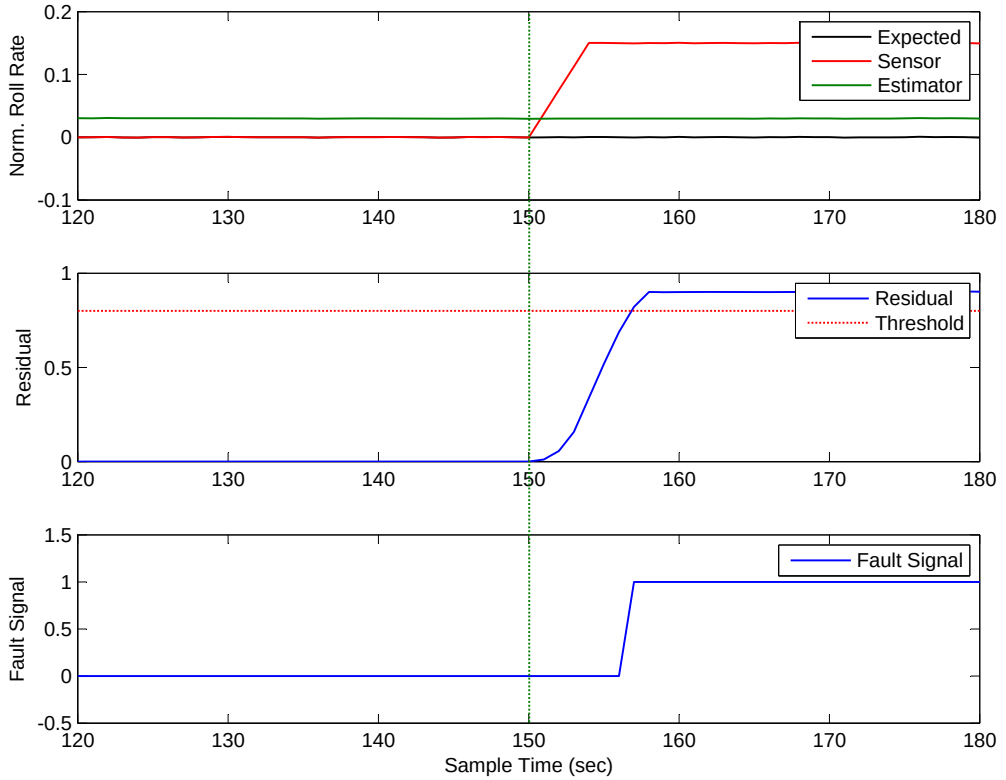
– : No Fault Detected, Threshold (τ) : 0.8, L : Large, S : Small

4.7 Summary of Results and Discussions

The FCC NN based SFDIA scheme is evaluated for failures in pitch, roll and yaw rate gyro sensors. Each sensor is manually injected with seven different faults at random locations on five different flight scenarios. The observations of the experiments can



(a) Scenario: 3, Magnitude: Large, Fault Occurrence: 1000 sec



(b) Scenario: 1, Magnitude: Small, Fault Occurrence: 150 sec

Figure 4.14: Roll sensor soft fault simulations.

be summarised as follows:

- Sudden fault types such as constant bias, hard additive and step additive are quicker to detect than faults that develop over time (e.g. soft additive faults).
- Faults with large magnitude are more easily detected than faults with small magnitude.
- Higher fault residual threshold to accommodate sensor estimator modelling errors and noise can increase the fault detection time.

These observations are consistent across the three gyro sensors. All faults were detected by the presented SFDIA scheme, except for one in the roll rate sensor. This undetected fault is a step fault with small magnitude. In this case, the fault went undetected because the residual failed to trigger the threshold. Due to the modelling inaccuracies of the developed pitch and roll estimators, the residual threshold for these estimators is set to $\tau = 0.8$, which is 4 times higher than the yaw residual threshold. This resulted in higher detection time as well as the roll fault going undetected. Nonetheless, the SFDIA scheme detected 104 faults out of the 105 cases evaluated.

The FDI results presented here can be compared to the SFDIA scheme presented in [37]. The scheme presented in [37] is based on the extended minimum resource allocating radial basis function (EMRAN-RBF) NN. The authors of [37], evaluated their SFDIA scheme on the pitch rate sensor for large magnitude ($A = 2.4$ deg/sec) additive faults. With their SFDIA scheme, pitch rate faults were detected in 1.24 sec, 1 sec and 1.86 sec for hard, step and soft faults respectively. In comparison, the SFDIA scheme presented here, detected the large magnitude ($A = 3$ deg/sec) faults in an average time of 1.4 sec, 0.8 sec and 3 sec for hard, step and soft faults respectively.

The results are fairly comparable, except for the case of soft failure, where the presented SFDIA scheme took 1.14 sec longer. This difference in performance can be accounted for by the fact that the SFDIA scheme presented in [37] uses a sampling time of 20 msec, compared to the 1 sec sampling time used in the scheme presented

here. The higher the sampling frequency, the quicker the faults are detected. Besides the sampling frequency, the SFDIA scheme presented here just uses 3 inputs compared to 4 inputs in [37].

One of the drawbacks of the presented SFDIA scheme is the fixed threshold based detection mechanism. Selecting a fixed fault threshold is a challenging task, especially in a dynamic system which is susceptible to noise and modelling inaccuracy. If the threshold is too high, the fault might take longer to be detected or worse, go undetected. Having a low threshold on the other hand might increase the rate of false alarms. The sliding averaging window mechanism does help reduce the effect of noise and modelling inaccuracy. However, if the dynamics of the system changes in the future, the thresholds would have to be evaluated and fixed again. An alternative to the fixed threshold based detection mechanism is an adaptive threshold. In this mechanism, the fault threshold adapts to the changes in the system dynamics with time. Such a mechanism, as presented in the [50] and [96], would increase the robustness of the SFDIA scheme presented here.

4.8 Conclusion

In this chapter, a FCC NN based SFDIA scheme was presented. The scheme was developed to address failures in the pitch, roll and yaw rate gyro sensors of an aircraft. This chapter presented the development of the FCC NN based pitch, roll and yaw rate gyro sensors. These estimators were used in the SFDIA scheme to replace the faulty sensors.

The results show that the FCC NN based estimators can produce good estimates of the sensor measurements, with as few as 2 neurons (see yaw rate results in Section 4.4.5). The pitch and roll rate estimators were able to produce good estimates with just 6 and 4 neurons respectively. However, the pitch rate estimator presented some anomaly. Upon further investigation it was conclude that the anomaly is due to the inputs to the estimator. Since the assumption made was that the aircraft is equipped with just the accelerometers and rate gyros, additional inputs to the estimator cannot be examined. Therefore the pitch estimator is limited by this as-

sumption. Further work needs to be conducted to identify additional sensor suites that could be used to correctly estimate the pitch sensor.

The SFDIA experiments covered 7 different failures over 105 experiments. Out of these experiments, only 1 failure went undetected. One of the shortcoming of the presented scheme is the detection mechanism. The scheme uses a fixed threshold based detection mechanism. While selecting a fault detection threshold, care must be taken to balance between the risk of false failure detection and no failure detection. The SFDIA scheme could be improved by implementing an adaptive fault detection threshold, which adapts with time.

To conclude, the results presented in this chapter show that the FCC NN can be used for SFDIA schemes. With as few as 2 neurons, the FCC NN was able to replicate the yaw rate sensor measurements. In the developed scheme, a faulty pitch, roll and yaw rate sensor is replaced by its respective NN estimators. Therefore, the scheme can add endurance to an aircraft system in the presence of failures in these sensors.

"Nothing is impossible, the word itself says 'I'm possible!"

– Audrey Hepburn

Chapter 5

Actuator Failures in the X-Plane Simulator

5.1 Overview

The next stage of the research is to develop the AFDIA scheme. Similar to the SFDIA, the X-Plane flight simulator is used. This is because of its realistic simulations of the aircraft dynamics, which is important for the AFDIA experiments. However, many challenges were faced while using this simulator for this research. This chapter attempts to highlight these challenges and explain how it affected the intended actuator failure research.

5.2 Initial Actuator Failure Study Objectives

In the initial stages of this research, the objective was to study actuator failures in the elevator, aileron and rudder flying control surfaces. The types of failure considered for the study are as follows:

1. **Stuck at failure**: where one of the flying control surface is stuck at a deflection angle.

2. **Loss of control surface**: where one of the flying control surface detaches from the aircraft.

3. **Combined stuck at failure**: where two different control surfaces are stuck at a deflection angle. For example a combination of elevator and aileron or aileron and rudder, stuck at a deflection angle.

The aim was to develop an actuator failure detection, identification and accommodation scheme (AFDIA) that can add endurance to an aircraft in the presence of the said failures.

5.3 Failure Simulation Constraints in the X-Plane Simulator

The X-Plane simulator is capable of simulating various types of failure, amongst which is the flying surface failure. This can be seen in Figure 5.1 which shows the X-Plane System Failures menu, with the Flying Surfaces failure tab selected. Using this menu, failures can be induced at any time during the simulation. To study the flying surfaces failure options, similar to the SFDIA scheme, the Cessna 172SP aircraft model in X-Plane 9 was used. Figure 5.1 shows that there is a drop down menu next to each of the aircraft parts simulated in X-Plane. These parts are linked to the various flying surfaces of the aircraft. The drop down menu gives the option of inducing a failure to its respective aircraft part. It does not however enable the specification of the type of failure being induced. The parts of interest to this research are listed in the first column of the menu in Figure 5.1.

VERT STAB stands for vertical stabiliser, which in this case is the rudder of the Cessna. Notice there are two parts to VERT STAB. Depending on the modelling of the aircraft, the rudder can be divided into two separate parts. In the case of the Cessna aircraft model, only the VERT STAB 1 is linked to an actual flying surface of the aircraft model. H STAB stands for horizontal stabiliser or the elevator of the aircraft. There are two of them, one each for the left and right elevators. The remaining parts in the first column are the four sections of the left and right wing of the aircraft. Once again, the number of sections, length of each section and how they link to the actual flying surfaces, depends on the modelling of the considered

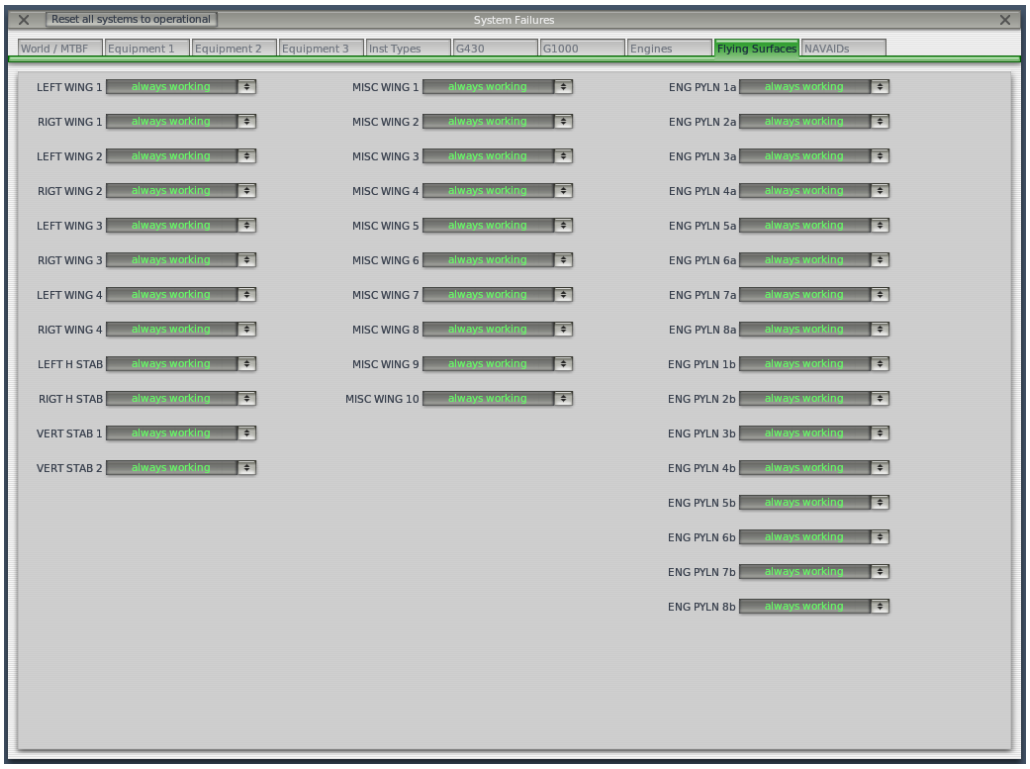


Figure 5.1: Flying surface failure options menu in X-Plane 9.

aircraft. Note that if an aircraft modelled in X-Plane just has two sections to a wing, LEFT/RIGHT WING 1 and 2, inducing failure on the remaining two parts option, LEFT/RIGHT WING 3 and 4, will have no effect in the simulation. This applies to all the flying surface parts listed in Figure 5.1. If one of these parts option is not modelled/linked in the aircraft model, inducing failure will have no effect at all. As part of the X-Plane simulator package, Laminar Research provides an application called the Plane Maker, which is used to develop or modify aircraft models for X-Plane. This application can be used to examine which parts of the modelled aircraft are actually linked to the flying surface parts listed in Figure 5.1.

Using the failure simulation menu in Figure 5.1, if a failure is simulated on the LEFT/RIGHT WING 1 parts, the ailerons on the Cessna visually appear to be stuck at the current deflection in the X-Plane simulation. This is because the ailerons of the Cessna are modelled on the WING 1 section of the aircraft. If a failure is simulated on VERT STAB 1, the rudder of Cessna visually appears to be stuck on the current deflection in X-Plane. Similarly, the left and right elevators of the Cessna aircraft model appear to be stuck at their current deflection if a failure is simulated on the respective H STAB. From these observations, it is concluded that

the only type of failure that can be simulated on the flying surfaces of an aircraft in X-Plane is the 'stuck at' type failure. Selecting the fail option in the drop down menu associate with the parts listed in Figure 5.1 simulates 'stuck-at' type failure. There is no option to simulate the loss of control surface type failure. Hence, it was decided to exclude the study of loss of control surface failure from the intended research.

Once the type of actuator failure simulated by X-Plane was identified, the effects of the 'stuck-at' type failure on the Cessna was investigated prior to the development of the AFDIA scheme. Failure simulations were conducted at random times during the flight of the Cessna. During the simulation it was observed that the aircraft did not behave as expected following the introduction of the failure. For example, during straight level flight of the aircraft, failure is simulated on the elevators of the Cessna. This should result in the Cessna elevators remaining stuck at about 0 degrees and the straight level flight should be maintained. Instead of maintaining flight, the aircraft pitches down and eventually crashes. It behaves as if the elevators were removed from the aircraft and affected the aerodynamics, which consequently resulted in the crash. Due to this observation, it was decided to try to simulate the flying surface failure available in X-Plane using a different aircraft model. Hence, the Airbus A320 model in X-Plane was selected to observe the failure simulated by X-Plane.

When a flying surface failure is simulated on the Airbus A320 using the menu in Figure 5.1, very different results are observed. For example, simulating the elevator failure on the Airbus A320 results in the disappearance or removal of the elevator from the aircraft in X-Plane. Simulating the failure on the WING 1 or WING 2 part, results in sections of the aircraft wing being removed. This is very different from the observations in the Cessna failures, where the elevator or ailerons appeared to be stuck at their most recent deflection. Unfortunately there is no published documentation of the X-Plane simulator that details the type of failure simulated by X-Plane through its built in features. Nonetheless, there is a web page to help developers and a forum for technical/non-technical discussions [97, 98]. The forum

Figure 5.2: Cessna 172SP in X-Plane with failure on the LEFT WING 1 part. The left aileron is stuck at a deflection.

is the only reliable way to clarify any problem in X-Plane by communicating with other engineers working with X-Plane and the X-Plane developers themselves.

After discussing this difference in observations with other developers and engineers in the forum, it was concluded that the only type of flying surface failure the simulator is capable of simulating is the loss or removal of the selected flying surface. Therefore, if left or right H STAB is put in a failed state, the respective elevator would be removed from the aircraft model, simulating a loss of the entire elevator surface. Similarly, with the WING part failure, the entire surface of a section of the wing is lost. The extent of the surface loss depends on the modelling of the aircraft. Each WING part (i.e. LEFT/RIGHT WING 1/2/3/4) may or may not be linked to different sections of the aircraft wings. In the case of the Airbus A320, the inner wing or the part of the wing that attaches to the aircraft body is linked to WING 1. The outer part of the wing, where the aileron is situated is linked to WING 2.

In the case of the Cessna 172SP, flying surface failure simulation also results in the loss of the flying surface of the aircraft. Although the failed surface remains visible and appears to be simulating the 'stuck-at' failure, the simulator is performing the aerodynamic calculations without the failed surface and is simulating the loss of the flying surface. The failed surface remains visible during the simulation in X-Plane

Figure 5.3: Airbus A320 in X-Plane with failure on the LEFT WING 2 part. The WING 2 section of the aircraft appears to be removed or destroyed.

because of the method used to develop the aircraft animation model. Due to the way the Cessna 172SP model is created, the animation of the failed surface remains visible following a failure. The Airbus A320 model on the other hand, was created using a method that ensures that the animation of the failed surface disappears when the failure is simulated. Therefore as soon as the failure is simulated using the menu in Figure 5.1, the wing surface disappears as expected. Nevertheless, in both cases the simulator is performing the aerodynamic calculations for an aircraft with the missing failed flying surface.

Since the X-Plane simulator is limited to simulating the loss of flying surface, the 'stuck at' type failure had to be excluded from the actuator failure study. However, attempts were made to explore whether the failure could be simulated using the X-Plane *datarefs* [99]. Datarefs are variables that publish information about the X-Plane simulation. They can be used to develop X-Plane plug-ins to manipulate the simulation.

Among the extensive list of *datarefs* available for X-Plane 9, the one of interest is the control surface override *dataref*: *sim/operation/override/override_control_surfaces*. This *dataref* allows the user to override the control surface deflections of any of the moving flying surfaces (e.g. the elevator, rudder and aileron) on the aircraft. The

aim was to use this *dataref* to override the deflection of the intended failed control surface and keep it stuck at a deflection angle, using their respective *dataref*. When this control surface deflection override *dataref* was set to ON, all the control surfaces on the aircraft were disabled. This means that the deflection values for each of the control surfaces must be individually programmed, including the intended failed surface, throughout the simulation.

In addition, there is no possibility in X-Plane, either built-in or through *dataref*, that would allow the simulator's calculations of each of the control surface deflections to be manually written to the respective surfaces, with the override *dataref* turned ON. This leaves just one option where each of the control surface deflections is calculated (using a control algorithm) manually and a 'stuck at' failure is simulated on the intended failed surface by programming a fixed deflection value throughout the simulation. In other words, flight control algorithms have to be developed to implement the 'stuck at' failures, which contradicts the need to use the X-Plane simulator. Due to these X-Plane limitations, the actuator failure study for this research is limited to the 'loss of flying surface' type of failure.

5.4 A Severe Case of Failure

As concluded earlier, the only type of failure simulated by X-Plane is the 'loss of flying surface'. Due to this limitation of the simulator, the objectives for the actuator failure research is revised to only address this type of failure. To that end, the aim is to develop an AFDIA scheme that can detect such a failure and accommodate it. Furthermore, it is decided to use the Airbus A320 instead of the Cessna 172SP model for the actuator failure simulations.

Note that there is a difference between a 'loss of control surface' and 'loss of flying surface' failure. The 'loss of flying surface' is a far more severe case of a failure in an aircraft. In such a failure you may not only lose a part of your control surface, but the entire flying surface. For example, simulating a loss of flying surface failure on the WING 2 part of the Airbus A320 results in an entire section of the wing surface being removed, including the aileron (control surface) on that wing. This

Figure 5.4: F-15 aircraft landed safely by Israeli pilot with just one wing. Taken from www.uss-bennington.org [3].

may not be a common type of failure, but such a failure was previously experienced. A well known case of such a failure was that encountered by the Israeli F-15 fighter pilot Ziv Nedivi in 1983 [3, 100]. During a training exercise, the F-15 had a mid air collision and lost an entire wing. Following this failure the aircraft went into an uncontrollable spin and headed for the ground. Fortunately, due to the quick thinking and experience of the pilot, the aircraft was bought under acceptable control and landed safely, using just one wing.

Due to the extreme nature of the failure and time limitations, the actuator failure studied in this research is limited to the loss of wing surface (which includes the aileron control surface) of the Airbus A320. This is an extreme case of a failure, similar to the F-15 incident. If this failure can be accommodated using an NN based AFDIA scheme, it is conceivable that the 'stuck at' type of failure can also be accommodated using an NN based AFDIA scheme.

In the conducted actuator failure research, the WING 2 part of the Airbus A320 is set to a failed state in Figure 5.1. This will result in the loss of wing surface for the entire section of the wing which is linked to WING 2. Using the Plane Maker

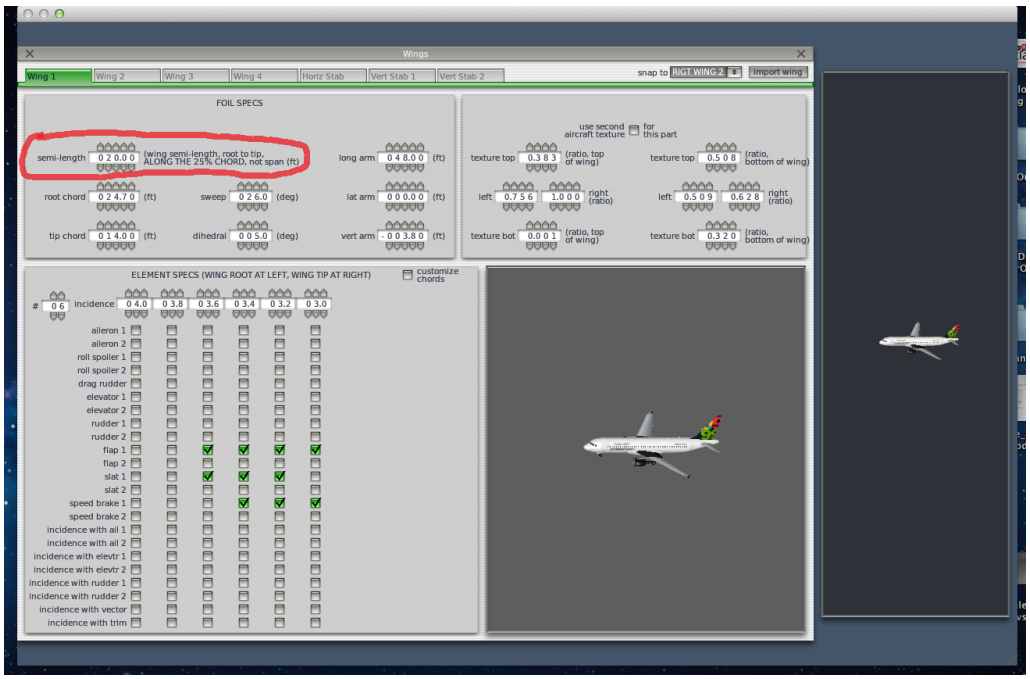


Figure 5.5: Airbus A320 in Plane Maker. WING 1 part linked to the inner section of the aircraft wing.

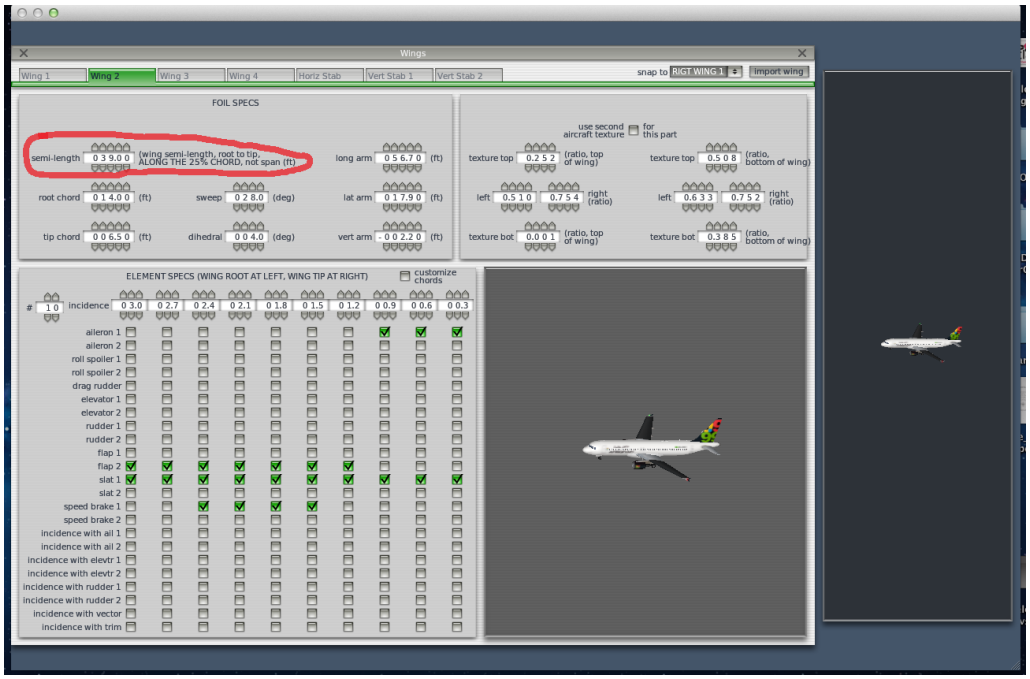


Figure 5.6: Airbus A320 in Plane Maker. WING 2 part linked to the outer section of the aircraft wing.

application, the length of each section of the wing linked to parts WING 1 and WING 2 can be determined. In Figures 5.5 and 5.6, a screenshot of the WING 1 and WING 2 section of the A320 aircraft wing in Plane Maker is presented, respectively. The length of each of the sections is marked in red. Notice that different sections of the wing are coloured in black based on which WING part is selected from the menu. The length of the wing of the modelled Airbus A320 is 59 meters, of which WING 2 represents a section of 39 meters. Therefore a loss of wing surface failure using WING 2 would result in about 66 % wing surface loss.

5.5 Challenges Faced Using X-Plane

X-Plane is well known for its almost realistic aircraft dynamics simulation due to which it is certified by the FAA for pilot training [29, 31, 97]. Due to its realistic simulations, it is used by the likes of NASA, Cessna and Japan Airlines, to train pilots, develop concept designs and flight testing [30, 31]. However, as is evident from the previous sections, there are certain challenges faced when using X-Plane for specific research purposes.

Although a manual is provided on using X-Plane [30], it is aimed at users who are interested in learning to fly. The technical information as to how the systems on the aircraft are implemented, what types of failures are simulated, amongst others, are not readily available. Information like this is only available through the forum [98]. In addition, the *datarefs* that can be used to manipulate the simulator to implement the AFDIA scheme are not well documented. There are occasions where the same property of the simulator can be manipulated using multiple *datarefs*, without any clear distinction. Therefore, one must experiment with the *datarefs* in order to find a suitable one for the intended purpose.

Such examples, among others, are some of the challenges faced while using the X-Plane simulator for this research.

5.6 Conclusion

In this chapter, a brief overview was given of how the actuator failure study and the development of the AFDIA scheme evolved during the course of this research. One of the objectives of this chapter was to highlight the challenges faced while using X-Plane for specific research purposes. The simulator is well known for its realistic simulations of the aircraft dynamics and therefore used for training pilots. However when it comes to a specific research study, the limited documentation can be challenging. The X-Plane technical forum is very helpful in such cases. But it can take a while before a solution is found. This is especially true if the simulator was rarely used in such a way, which is the case with this research; or in other research and development works where the simulator was used with proprietary information, therefore resulting in limited publicly available documentations.

Due to the limitation of the simulator, the actuator failure study is limited to the loss of flying surface type of failure. In the next chapter, the loss of the wing surface failure is studied and a AFDIA scheme is developed.

"There is only one thing that makes a dream impossible to achieve: the fear of failure."

– Paulo Coelho, *The Alchemist*

Chapter 6

The AFDIA Scheme

6.1 Overview

In this chapter, an actuator failure detection, identification and accommodation (AFDIA) scheme is presented. The aim of the scheme is to increase the endurance of the aircraft following a loss of 66% of the wing surface. To achieve this, a fully connected cascade (FCC) neural network (NN) based roll controller is implemented. This NN based roll controller has the ability to adapt on-line to the post failure dynamics of the aircraft. In the presented AFDIA scheme, the FCC NN based roll controller is used to control the aircraft in the case of failure. The AFDIA scheme is divided into two main stages:

1. **Failure detection and identification (FDI)**

   The purpose of this stage is to first detect a failure that has occurred or is occurring. Following a successful detection, the source of the failure needs to be identified.

2. **Failure accommodation (FA)**

   The objective of this stage is to take action to compensate for the failure. In this research, this action happens to be adapting the roll controller to try to bring the aircraft back to equilibrium by compensating for the rolling moment induced by the loss of wing surface.

This chapter begins with Section 6.2, which explores what happens to an aircraft following a loss of 66% of the wing surface. In Section 6.3, the concept behind the development of the NN based roll controller and its development process is presented. The AFDIA scheme is presented in Section 6.4. The setup for the AFDIA experiments is presented in Section 6.5. In Section 6.6, the results of the AFDIA experiments are presented and discussed. Based on the results of the conducted AFDIA experiments, improvements are made to the AFDIA scheme in Section 6.7. In Section 6.8, the setup and overview of the experiments conducted using the improved AFDIA scheme are explained. The results of the improved AFDIA scheme experiments are discussed in Section 6.9. Finally, the chapter concludes with Section 6.12.

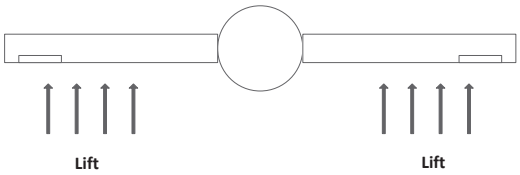
6.2 Loss of Wing Surface Failure

In this section, an overview of what happens to an aircraft following a loss of wing surface is presented. This is followed by the discussion on how to detect and identify this failure.

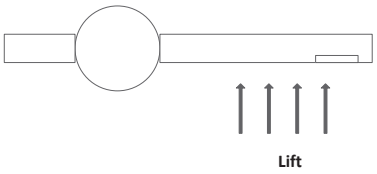
6.2.1 Overview

Wings are crucial to an aircraft as they provide most of the lift required to maintain flight. Any structural failure to the wing during flight will result in the decrease in lift. Such a failure can unbalance the lift distribution across the aircraft, which can create a prominent rolling moment. The magnitude of the rolling moment will depend on the extent of the damage. If the damage is significant and not quickly accommodated, the aircraft will go into an uncontrollable spin. The proposed AFDIA scheme aims to accommodate this severe case of failure.

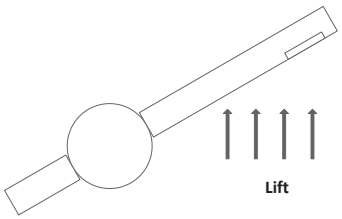
The X-Plane 9 simulator has the ability to simulate the loss of wing surface failure. The simulator can simulate complete loss of wing or loss of a section of wing, depending on how the model was made in X-Plane. For the purpose of this research, loss of a section of the wing on an Airbus A320 is simulated, which accounts for about 66% of the wing. To study the behaviour of the aircraft and investigate sensor



(a) Before Failure



(b) At Failure



(c) After Failure

Figure 6.1: Aircraft wings and lift force acting on them.

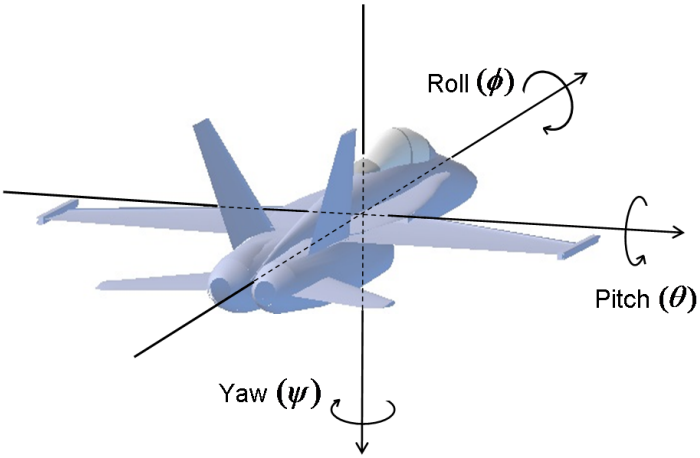
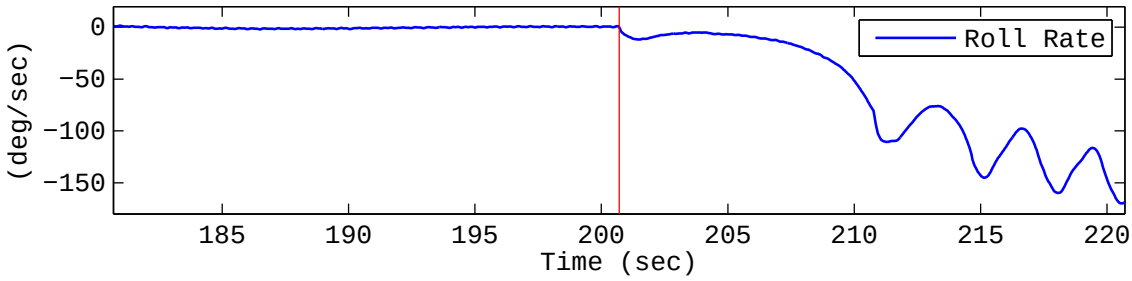
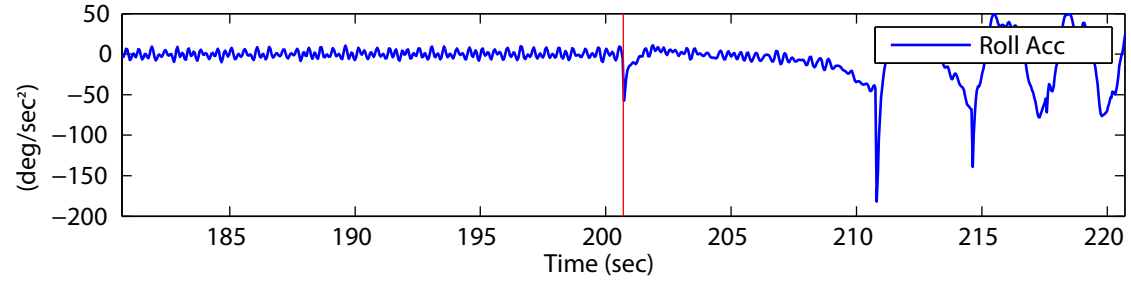


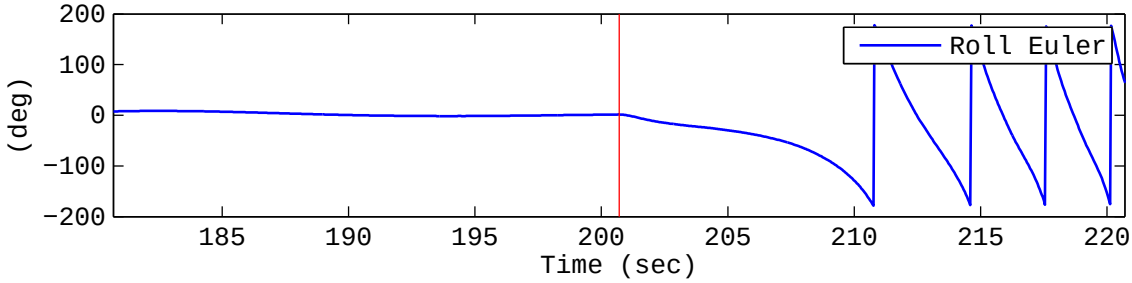
Figure 6.2: Aircraft attitude angles or Euler angles. This figure is adapted from [4].



(a) Roll Rate



(b) Roll Acceleration



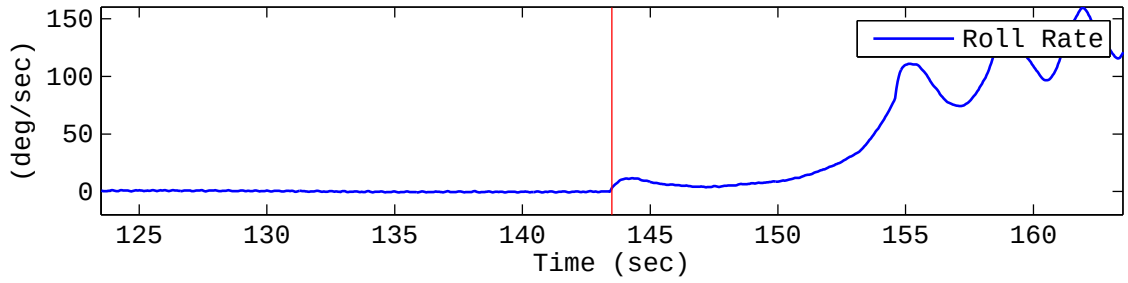
(c) Roll Euler Angle

Figure 6.3: Roll related sensor measurements from an aircraft following a left wing surface loss failure.

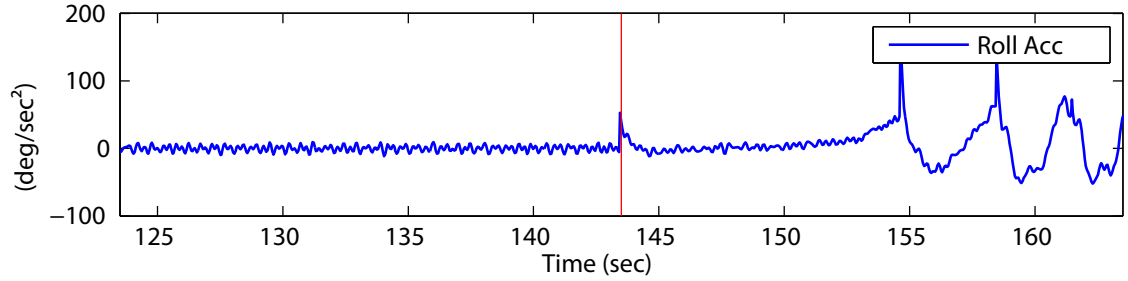
measurements following this failure, simulations are conducted with the aircraft under the control of the built-in autopilot. Some of the signals of interest from these simulations are presented in Figures 6.3 and 6.4.

Figure 6.3 shows the roll rate (p), acceleration ($\dot{p}$) and Euler angle (ϕ), 20 sec before and after the loss of left wing surface. The time of failure is denoted by the red vertical line. As expected, the aircraft goes into an uncontrollable spin following such a failure. This is clearly depicted by the fluctuating roll Euler angle, which represents the roll attitude of the aircraft. The roll Euler angle changes from one maximum end ($\phi = 180$ deg) to the other ($\phi = -180$ deg), depicting a continuous spin along the roll axis of the aircraft. Similar results can be seen following a right wing surface loss failure in Figure 6.4.

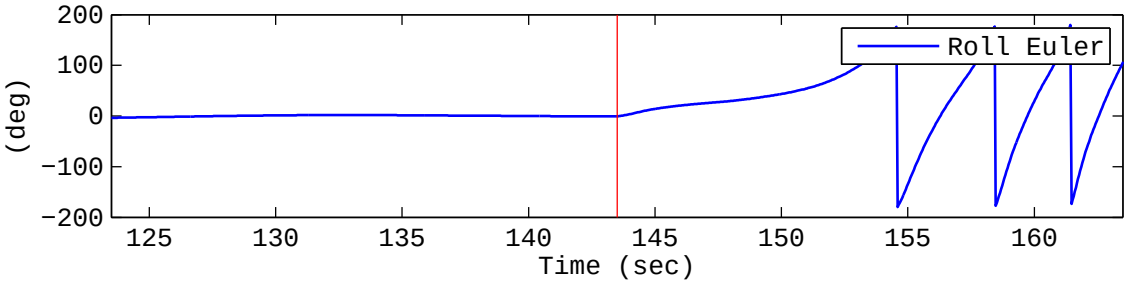
In both cases, the autopilot controller is unable to stop the aircraft from going



(a) Roll Rate



(b) Roll Acceleration



(c) Roll Euler Angle

Figure 6.4: Roll related sensor measurements from an aircraft following a right wing surface loss failure.

into an uncontrollable spin. The controller does resist the spin for a while, which can be seen from the gradual change of the Euler angle following the failure. However, once the aircraft reaches one maximum end of the roll Euler angle, the uncontrollable spin begins. This can be attributed to the fact that the autopilot controller is working under the fault free model assumption of the aircraft, which it was designed for. Following a loss of wing surface failure, the dynamics of the aircraft change and the autopilot controller is unaware of such an occurrence. Therefore, there is a need for a controller that can adapt to the new dynamics of the aircraft following such a failure. This is the motivation for the development and application of the NN based flight controllers. The NN is incorporated with on-line learning capabilities to enable the adaptation.

6.2.2 Detecting and Identifying Failure

As can be seen from Figures 6.3 and 6.4, at the time of failure, there is a sudden significant fluctuation from the norm in the roll acceleration measurements. This sudden fluctuation can be used to detect the loss of wing surface type of failure. Since the fluctuation in the roll acceleration is significant compared to the norm, this failure can simply be detected by a fixed threshold mechanism. In this mechanism, the failure is detected when the roll acceleration crosses a fixed threshold, similar to the SFDIA scheme.

In Table 6.1, the maximum roll acceleration following a wing surface loss failure is presented. The table presents the results from 5 separate simulations, for left and right wings each. The average maximum roll acceleration is above 50 deg/sec^2, for both left and right wing failure. The failure threshold could be set to this, but as can be seen from the first experiment for the right wing failure, the maximum roll acceleration is at 44.8 deg/sec^2.

While selecting a fixed threshold, two points need to be considered: false detection and detection time. If the threshold is too low (close to normal measurements), failure might be triggered in the absence of one. And if the threshold is set too high, it might take a while for the failure to be detected, or worse, go undetected. In this

research, after experimenting with various thresholds, it is decided to set the fixed threshold at $\tau = 35$ deg/sec^2.

Table 6.1: Maximum roll acceleration values immediately following a wing surface loss failure for left and right wings. Results from 5 different X-Plane simulations for each left and right wing are presented.

	Max. Roll Acc (deg/sec^2)	
Exp No.	Left Wing	Right Wing
1	-56.2557	44.8520
2	-57.4426	52.8902
3	-64.3698	49.7127
4	-47.8023	59.1092
5	-49.7121	52.3656
Average	-55.1165	51.7859
SD (σ)	6.6151	5.1856

Another important observation from Figures 6.3 and 6.4 is the direction of the sudden fluctuation in the roll acceleration immediately following a failure. The roll acceleration is negative ('$-ve$') if the failure is on left wing and positive ('$+ve$') if it is on the right wing. This can also be seen from the maximum roll acceleration measurements immediately following a failure; as presented in Table 6.1. This finding can therefore be used to identify the failed wing. In the next section, the concept behind the design of the NN based roll controller and its development process are presented.

6.3 Adaptive Neural Network Roll Controller

In this section, the concept behind the development of the roll controller is presented. This is followed by the development of the roll controller.

6.3.1 Balance the Moment

As discussed in the previous section, following a wing surface loss failure the aircraft goes into an uncontrollable spin. This is due to the rolling moment induced by

the imbalance of the lift force across the aircraft. The aircraft autopilot controller is unable to stop this spin, as it is still operating under the assumption that the aircraft is fault free, which is what it was designed for. It is unaware of the change in the aircraft dynamics following the failure. Therefore, to increase the endurance of the aircraft in case of such a failure, a controller with an adaptive capability is required.

In this case, an NN based roll controller is developed which can adapt to the new dynamics of the aircraft following a failure. Under normal conditions, the NN based adaptive roll controller must emulate the output of the autopilot roll control. Following a loss of wing surface, the objective of the adaptive controller is to adapt the use of the aileron on the healthy wing. This is to attempt to produce the compensating moment required to cancel the failure induced rolling moment and bring the aircraft back to equilibrium.

It must be noted that the failure induced rolling moment that needs to be cancelled can vary throughout the duration of the flight. The aircraft is flying in a dynamic environment, simulated by the X-Plane weather system. Environment factors such as the direction and speed of the wind, amongst others, will affect the moment forces acting on the aircraft. With a loss of 66% of the wing surface, it is conceivable that the aircraft will be highly unstable in these dynamic conditions.

Table 6.2: Inputs/Output of the NN based roll controller.

Output	Inputs
Roll control command ($\hat{\delta}_A$)	Roll rate (p)
	Actual roll Euler angle (ϕ_{act})
	Demanded roll Euler angle (ϕ_{dem})

The NN based roll controller has three inputs, namely: roll rate (p), actual roll Euler angle (ϕ_{act}) and demanded roll Euler angle (ϕ_{dem}). The actual and demanded Euler angles inform the controller about the error in the aircraft roll attitude, which is the difference between the two Euler angles ($\phi_{act} - \phi_{dem}$). The roll rate gives the controller a sense of the current roll motion. These inputs are chosen to provide the controller with all the information it needs to generate the roll command, in the

absence or presence of failure. The output of the NN based roll controller is the roll command denoted by $\hat{\delta}_A$. The output is a normalised value in the range of ± 1. The list of the inputs to the NN based roll controller is presented in Table 6.2.

In the next section, the roll controller development process is discussed. The aim of this is to identify the right number of neurons required to generate the roll command.

6.3.2 Roll Controller Development Process

The roll controller development process is similar to that of the NN based sensor estimator described in Section 4.4. This process is as follows:

1. **Training and Validation Data**

   The development of the controller can be divided into two main phases: the training and the validation phases. In the training phase, the controller is trained on a dataset to learn its functionality. Once the training process is completed, the functionality of the controller is validated over several datasets.

   These datasets are generated using flight data collected from X-Plane simulations. The X-Plane simulator is used to collect flight data for the Airbus A320 aircraft, recorded at 50 Hz (i.e. 0.02 sec). In total, 16 X-Plane simulations are conducted to collect data for the controller development. In these simulations, the aircraft takes off from various airports and is controlled by the aircraft autopilot system in X-Plane.

   Out of the 16 flight simulations, 1 is selected at random to generate the training dataset. The training dataset is generated such that it starts from the flight data recorded just after take-off. The dataset spans over 4 minutes, during which the aircraft climbs to the requested cruise altitude and keeps changing its heading as commanded at random times. This helps to encapsulate the steady and transient state behaviour of the aircraft autopilot roll controller. Since the data was recorded at 50 Hz (every 0.02 sec) and the training dataset spans over 4 minutes, there are 12000 individual samples in the training dataset.

The remaining 15 flight simulations are used to generate 15 validation datasets, each covering 10 minutes of flight data. These validation datasets help to verify and assess the functionality of the trained controller. Since the validation dataset spans over 10 minutes, there are 30000 individual samples in each validation dataset.

2. **Training the Controller**

   The controller is trained using the 12000 samples in the training dataset. This training dataset is used to train 10 different controller designs, ranging from 2 to 12 neurons. Each of these controllers is trained until the Sum Squared Error (SSE) of the training epoch is ≤ 0.001 or a maximum of 1000 epochs is reached. The training dataset is presented in Figure 6.5.

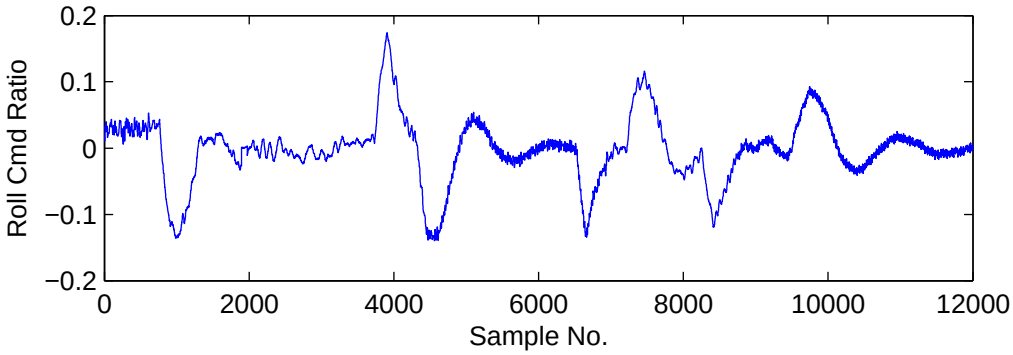
3. **Validating the Controller**

   Once trained, each controller design (ranging from 2 to 12 neurons) is validated on the 15 validation datasets. These datasets are 10 minutes long, containing 30000 samples. The performance of the different controller designs are assessed for each dataset by calculating the total SSE of all the samples in the dataset. The best controller design is then selected by calculating the average and the standard deviation of the SSE for all the datasets. The results of this validation process is presented in Table 6.3. From the table, the design with 5 neurons is the best option for the roll controller. The NN controller with 5 neurons has the lowest SSE standard deviation (5.161340) and average SSE as low as 9.77061. Note that the individual SSE presented in Table 6.3 is the total SSE of the 30000 samples. An example of the validation result using the 5 neuron based NN roll controller is presented in Figure 6.6.

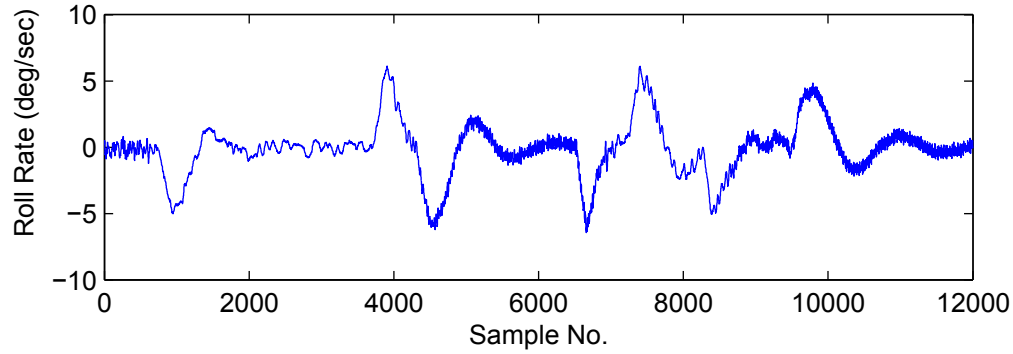
With the controller developed, the AFDIA scheme is presented in the next section.

Table 6.3: Roll controller errors for the validation datasets.

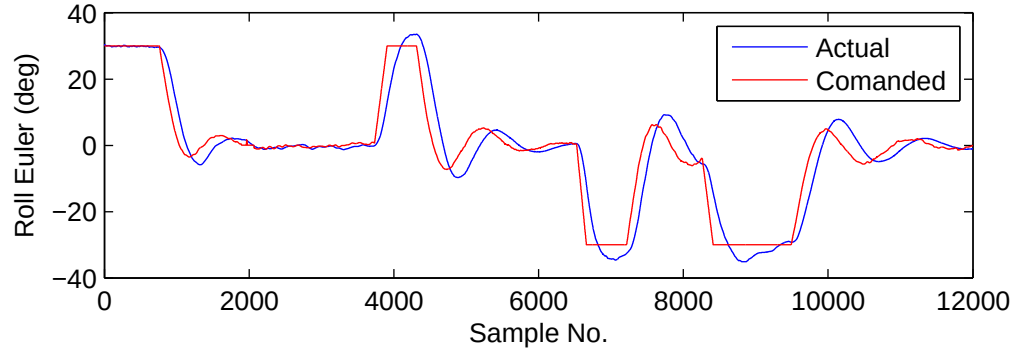
SUM SQUARED ERRORS (SSE)											
Neurons	2	3	4	**5**	6	7	8	9	10	11	12
Data 01	1.86780	2.00090	1.78750	**1.75700**	1.99710	1.83030	1.87100	1.81310	1.75450	1.67260	1.91320
Data 02	0.83990	1.22520	0.89150	**1.39100**	1.07500	1.61210	1.26470	1.50030	1.42280	1.67710	1.36840
Data 03	13.88270	12.33340	13.69940	**11.69820**	13.05050	11.15510	11.48640	11.35780	10.31770	11.48050	11.38620
Data 04	12.19850	13.50950	11.19970	**10.04490**	13.50660	10.62590	12.61350	10.79010	11.66770	9.53400	12.62980
Data 05	9.96100	11.15750	9.82130	**9.03730**	9.67260	9.59510	10.30510	9.37170	10.21010	9.34520	9.35620
Data 06	11.95280	12.46980	11.68920	**10.39980**	11.71330	10.09150	10.62050	9.87170	9.66060	9.84640	10.71530
Data 07	19.59420	20.22470	19.23700	**17.80840**	18.80870	16.56490	16.67490	16.56490	15.46220	16.18550	17.66540
Data 08	20.12000	22.49360	19.14640	**17.78380**	22.44340	19.09080	20.80910	18.70200	20.80110	19.63610	21.49560
Data 09	14.77530	14.82660	13.90790	**12.70920**	14.86480	12.99440	14.54710	13.79920	13.66480	12.32300	14.80790
Data 10	15.60090	16.32840	14.70970	**12.80460**	16.30110	12.29770	14.49810	14.64500	13.27830	13.42980	14.15150
Data 11	16.58730	16.91140	15.00410	**13.53750**	16.80660	13.72670	16.22020	15.39470	14.27710	12.61580	15.88620
Data 12	5.33470	4.87920	5.11510	**4.22530**	4.69760	3.63850	3.65960	3.92100	3.45360	3.57420	4.43240
Data 13	15.01850	15.76870	13.67460	**12.17090**	15.48610	13.19660	14.92270	13.45050	13.79220	12.27610	14.79780
Data 14	6.47850	6.16710	6.22110	**4.70910**	5.46930	3.67830	4.15090	4.08970	3.62250	3.40870	4.82570
Data 15	8.52830	8.79130	8.25770	**6.48210**	7.96040	6.02090	6.52040	6.01900	5.87360	5.14960	6.81420
Average	11.51603	11.93915	10.95748	**9.77061**	11.59021	9.74125	10.67761	10.08605	9.95059	9.47697	10.81639
SD (σ)	5.93317	6.30860	5.61135	**5.16134**	6.29879	5.33421	5.95675	5.51371	5.65981	5.38473	5.95526



(a) Actual roll command

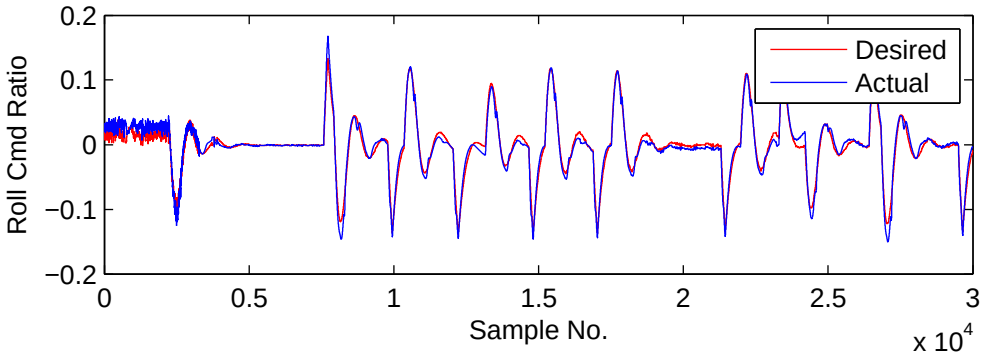


(b) Roll rate

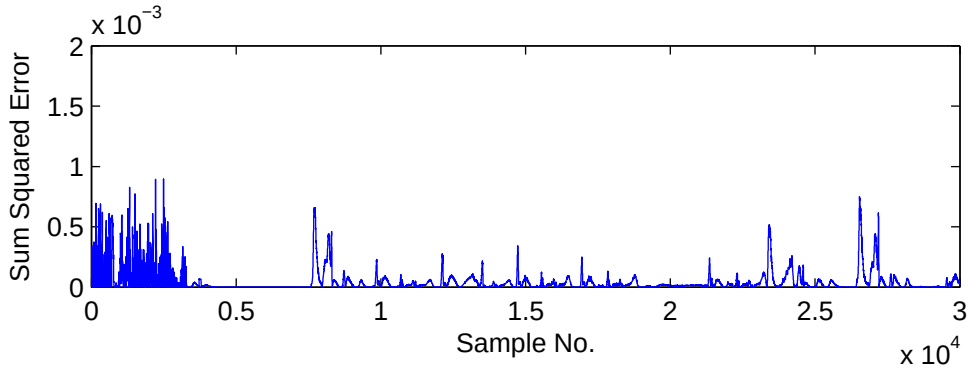


(c) Actual and commanded roll Euler angle

Figure 6.5: Training data for the neural network roll controller.



(a) NN controller output (actual) and the desired controller output.



(b) Sum squared error (SSE) between the NN controller output and actual output.

Figure 6.6: Validation results of 5 neuron based neural network roll controller on dataset 2.

6.4 The AFDIA Scheme

6.4.1 Overview

As mentioned at the beginning of the chapter, the developed AFDIA scheme can be divided into two main stages:

1. **Failure detection and identification (FDI)**

   The purpose of this stage is to first detect a failure that has occurred or is occurring. Following a successful detection, the source of the failure needs to be identified.

2. **Failure accommodation (FA)**

   The objective of this stage is to take action to compensate for the failure. In this research, this action happens to be adapting the roll controller to try to bring the aircraft back to equilibrium by compensating for the rolling moment induced by the loss of wing type of failure.

With this in mind, the operational outline of the developed AFDIA scheme is presented in the following section.

6.4.2 AFDIA Operational Outline

A flow chart for the AFDIA scheme is presented in Figure 6.7. The first stage of the scheme is the failure detection and identification (FDI). In this stage, the roll acceleration ($\dot{p}$) sensor measurements are monitored for detecting actuator failure and identifying its source. A fixed threshold based mechanism is used to detect the loss of wing surface. A failure is detected when the roll acceleration measurements cross a fixed threshold of $\tau = 35$ deg/sec^2. Once the failure is detected, the failed wing is identified using the direction of the fluctuation in the roll acceleration ($\dot{p}$) measurements. A negative fluctuation would indicate a left wing failure, while a positive fluctuation would indicate a right wing failure. The AFDIA scheme has two failure flags, F_L and F_R, for the left and right wing failure, respectively. If a failure is detected, the $F_{L/R}$ flag is set to 1, otherwise the flag remains at the default

value of 0. Therefore, the FDI stage of the AFDIA scheme can be as summarised as follows:

$$F_L = \begin{cases} 1, & \text{if } \dot{p} \leq -\tau \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

$$F_R = \begin{cases} 1, & \text{if } \dot{p} \geq +\tau \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

If the failure flag $F_{L/R}$ is set in the FDI stage, the next stage would be failure accommodation (FA). However, the presented scheme has an additional step, where a sliding data window is updated, regardless of the outcome of the FDI stage. This data window stores the previous n (window size) time steps of flight data variables which will be used to adapt the NN roll controller in case of failure. At each time step, the data window discards the oldest time step variables from the window and adds the current time step variables; effectively sliding the window through the data with time (t). Therefore, at time instance k, the sliding data window updates by storing the variables at time k and discarding the oldest variables from time instance $(k - n)$.

The purpose of this data window is to store data that will be used to adapt the NN roll controller in case of failure. Since the controller has 3 inputs and 1 output, at each time step (k), 4 variables are stored. At time k, the values of the NN controller inputs, namely, the roll rate (p), measured roll Euler angle (ϕ_{act}) and the demanded roll Euler angle (ϕ_{dem}), are stored in the data window regardless of the outcome in the FDI stage (i.e. $F_{L/R} = 0/1$).

The fourth variable (J_{roll}) depends on the FDI stage outcome. If no failure is detected in the FDI stage (i.e. $F_{L/R} = 0$), then the fourth variable stored in the window is the roll control command ($J_{roll} = \delta_A$) from the flight computer. Since no failure is detected, there is no need for adapting the NN roll controller. Hence, the data in the sliding window is not used. However, if a failure is detected in the FDI stage (i.e. $F_{L/R} = 1$), then a different variable is stored in the window, which

is used by the failure accommodation (FA) stage of the scheme. The variable stored is generated as follows:

$$J_{roll} = c \times m(p_{ref} - p) \tag{6.3}$$

where:

c : is a scalar multiple

p : is the roll rate in deg/sec

p_{ref} : is the desired roll rate in deg/sec

$m(x)$: $2 \times \frac{x + 10}{20} - 1$

In the FA stage, the NN roll controller needs to adapt to the post failure dynamics of the aircraft and try to achieve equilibrium. Following a loss of left or right wing surface, the aircraft goes into an uncontrollable spin induced by the rolling moment due to the failure. Therefore, the objective of the AFDIA scheme is to stabilise the aircraft by using the aileron on the healthy wing to compensate for the failure induced moment. This objective is achieved by adapting the NN roll controller.

To guide the on-line adaptation process of the NN roll controller, the function presented in equation (6.3) is used. The resultant J value from this function is considered as the desired output roll command by the on-line implementation of the NBN algorithm. The on-line implementation of the NBN algorithm is basically the NBN training process presented in Section 3.10, with the data in the sliding window considered as the training data. This training process is repeated once the sliding data window is updated, at every time step k (see Figure 6.7). At time k, the training process is executed only once (i.e. 1 epoch), using the data in the sliding window.

The guiding function presented in equation (6.3) uses the difference in the roll rate ($p_{ref} - p$) of the aircraft to guide the adaptation of the NN roll controller, based on the following assumption:

Assumption*: If the aircraft achieves an equilibrium state and stops spinning, the roll rate (p) of the aircraft must be approximately 0 deg/sec.*

Using this assumption, the desired roll rate of the aircraft (p_{ref}) in equation (6.3) is set to $p_{ref} = 0$ deg/sec. Following a failure, this difference in roll rate (i.e. $p_{ref} - p$) would be large compared to the controller output ratio that is in the range of ± 1. This range also happens to be the output range of the output neuron of the NN roll controller. If this difference is fed directly to the controller on-line learning algorithm, the controller might change more dramatically than required. Therefore, this difference is scaled using a scalar multiple (c) and the scaling equation (3.21), reproduced here:

$$m(x_o) = x_n = (b - a) \times \frac{x_o - x_{min}}{x_{max} - x_{min}} + a \tag{6.4}$$

Equation (3.21) scales the difference to the range of $\pm$ 1, assuming that the maximum and minimum value of the roll rate will be in the $\pm$ 10 deg/sec range. If the roll rate exceeds this range, then the equation will scale it accordingly. Notice when these range values are substituted in equation (3.21), it changes to $m(x)$ in equation (6.3).

In the case of a failure at time step (k), the fourth value stored in the sliding data window is the guiding function value J_{roll}, calculated using equation (6.3). The detection of the failure results in the on-line learning of the NN roll control being triggered. The data stored in the sliding window is then used by the on-line learning algorithm for adapting the controller. Once the learning is completed for the time step (k), the adapted NN roll controller is used to generate a new roll command. This command is then used by the flight control system to control the aircraft. Note that the presented AFDIA scheme just has an NN roll controller. Due to time constraints on the research, the pitch and yaw controllers were not developed. Therefore, the control for pitch and yaw is set to 0 following a failure. This has the added benefit of just studying the behaviour of the NN roll controller.

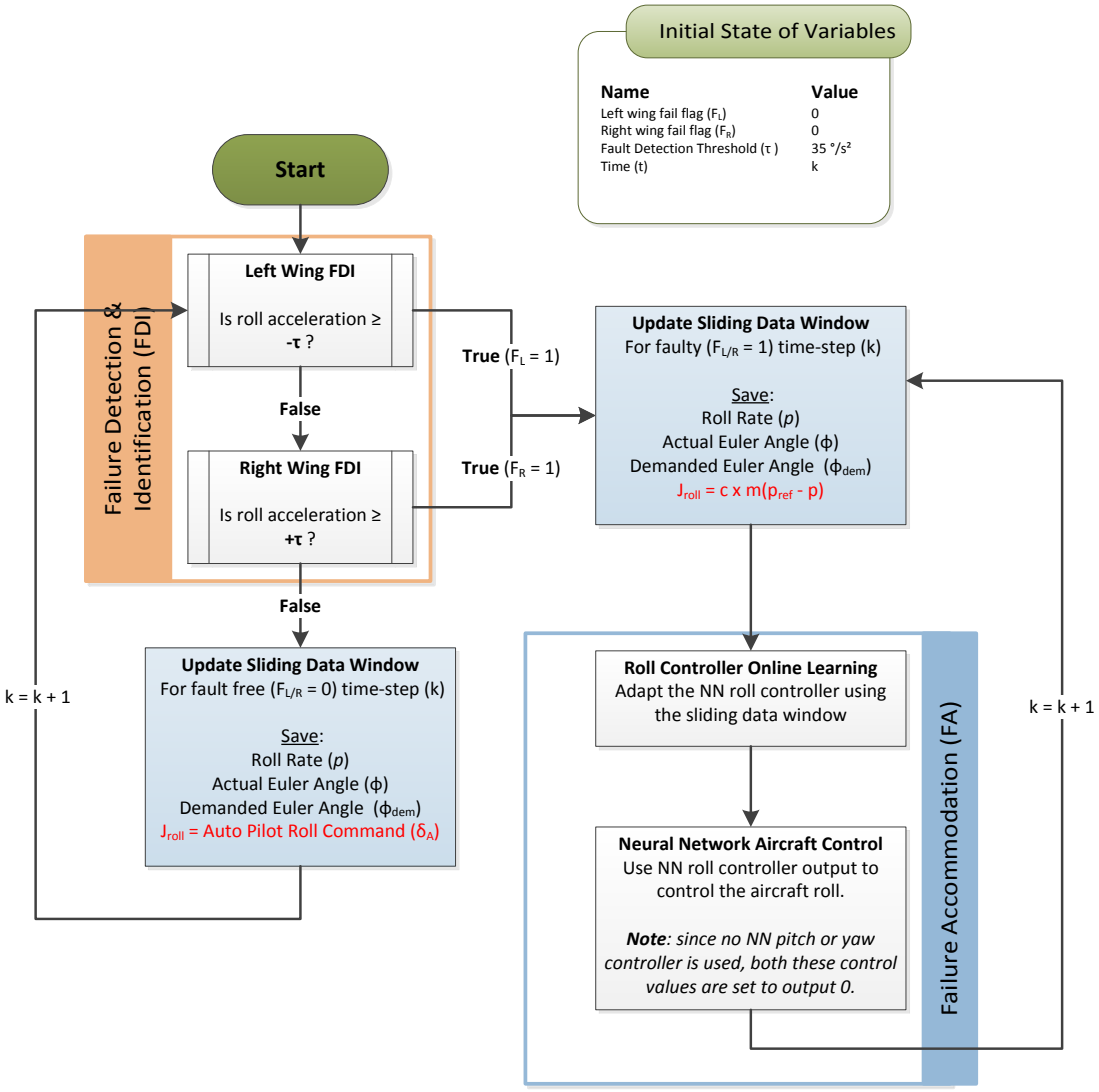


Figure 6.7: The AFDIA Scheme.

6.5 AFDIA Experimental Setup

In this section, an overview of the implementation of the AFDIA scheme in X-Plane, experiment conditions and overview are presented.

6.5.1 AFDIA Implementation in X-Plane

The AFDIA scheme is implemented in the X-Plane simulator as a plug-in. This plug-in contains a control window which allows the user to trigger the failure and display information about the state of the AFDIA scheme. During the development of the NN controller, the controller was trained and validated on datasets with a sampling frequency of 50 Hz. This was the intended frequency at which the AFDIA scheme was to be executed. Due to timing constraints imposed by X-Plane simulator execution demands and the need to log various simulation data for analysis, the execution frequency of the AFDIA scheme is decreased to 25 Hz. It will be shown later that the AFDIA scheme is capable of executing at 50 Hz.

6.5.2 Experimental Conditions

The X-Plane simulator is setup to simulate the flight of the aircraft under normal weather conditions. The Airbus A320 is controlled by the autopilot of the aircraft. The autopilot is programmed to maintain straight and level flight, to a set heading. The aircraft altitude is approximately 3000 ft (i.e. 941.4 m) above mean sea level (MSL), maintaining a speed of approximately 450 kn (i.e 833.4 km/h).

The wing surface loss failure is triggered at random times during the simulation. Following a failure, the autopilot of the aircraft is turned off to ensure that it does not interfere with the controller output from the AFDIA scheme.

6.5.3 Experiment Overview

Two main parameters of the AFDIA scheme, the sliding data window size (n) and the scalar multiple (c) in equation (6.3), are varied to perform a comparative study of the performance of the scheme. There are 3 different window sizes, namely, $n =$

5, 10 and 15. For each of the window sizes n, the scalar multiple varies from $c = 1$ to $c = 4$. For each combination of the window size n and scalar multiple c, 20 loss of wing surface failure experiments are conducted, 10 each for the left and the right wing. Therefore, in total 240 separate experiments are conducted as described by the equation below:

$$\begin{aligned} \text{No.Experiments} &= n \times c \times (10 \text{ Left} + 10 \text{ Right}) \text{ Failures} \\ 240 &= 3 \times 4 \times 20 \end{aligned} \tag{6.5}$$

The results from these experiments are analysed in the context of the following aspects:

i. **Post Failure Aircraft Behaviour**: The response of the NN controller and the behaviour of the aircraft following the failure are discussed.

ii. **Flight Duration**: The flight duration is used to quantify the endurance of the aircraft following a failure.

iii. **Failure Detection Time**: The time taken by the AFDIA scheme to detect the loss of wing surface is analysed.

iv. **AFDIA Scheme Execution Time**: The time taken by the NN based AFDIA scheme to execute in the presence or absence of failure is assessed.

In the next section, the results of the AFDIA scheme experiments are presented and discussed.

6.6 Results and Discussions

6.6.1 Post Failure Aircraft Behaviour

The AFDIA scheme is designed to accommodate the loss of 66% wing surface failure. The aim of the scheme following this failure is to try to control the aircraft and bring it back to a state where the moments are in equilibrium. If this is achieved, the

aircraft would maintain flight and therefore increase the endurance of the aircraft in the presence of such a severe failure. As discussed in the beginning of the chapter, under the control of the X-Plane autopilot, such a failure would cause the aircraft to spin uncontrollably along its roll axis and eventually crash. An ideal result for the AFDIA scheme would be if the aircraft manages to maintain flight following the failure.

Figures 6.8 and 6.9 present two of the simulation results from the AFDIA experiments conducted. Figure 6.8 represents the loss of left wing surface, while the loss of right wing surface is represented by Figure 6.9. These figures depict the roll acceleration ($\dot{p}$), roll rate (p), roll Euler angle (ϕ), both the NN controller and autopilot roll commands ($\hat{\delta}_A$ and δ_A respectively); and the altitude of the aircraft above mean sea level (MSL). These figures are plotted from 20 sec before the failure, where the time of failure is marked by the vertical red line. From the altitude plots in both the figures, it is clear that following a failure, the aircraft gradually looses altitude. The plots in these figures end when the aircraft crashes. Thus, the ideal result for the AFDIA scheme is not achieved. However, the aircraft does avoid going into an uncontrollable spin, as can be seen from the Euler angles. These observations are consistent in all the 240 experiments conducted.

In Figures 6.8 and 6.9, almost immediately after the failure, the roll acceleration crosses the fault detection threshold. The NN roll controller immediately adapts and responds to compensate for the induced rolling moment. In Figure 6.8, the NN roll controller output jumps to about + 0.7 in response to the failure. Notice that this is in the opposite direction to the jump in roll rate and Euler angle, where the measurements are negative. In an attempt to balance the rolling moment, the NN roll controller is trying to pull the aircraft in the opposite direction to which it is rolling. Similar behaviour can be observed in the case of right wing failure in Figure 6.9.

In both the figures, within seconds of the failure, the Euler angle gradually starts to change, indicating that the aircraft is gradually rolling to a side. In contrast to this, the altitude of the aircraft remains almost steady following a failure. The

altitude gradually climbs for a while before it starts to fall. The climb in altitude is due to the minor upward pitching of the aircraft following the failure. Eventually the aircraft cannot climb any further due to the extent of the aircraft roll attitude and the decrease in lift. At this point the aircraft starts to loose altitude and eventually crashes.

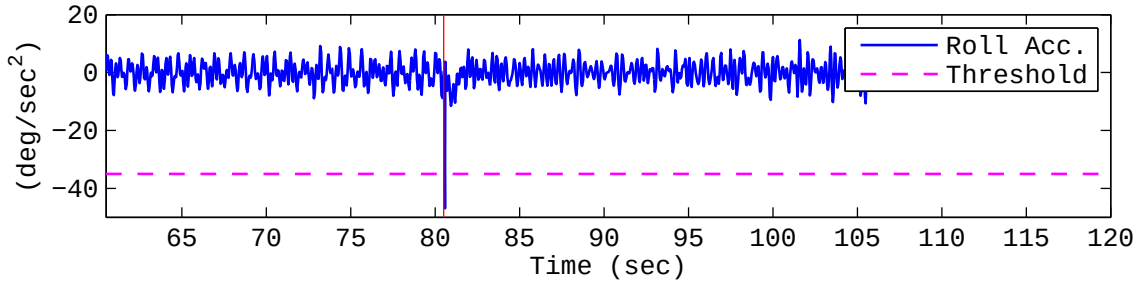
Notice the difference in the Euler angle plots using the AFDIA scheme in Figures 6.8 and 6.9, compared against the X-Plane autopilot only control in Figures 6.3 and 6.4. The aircraft avoids an uncontrollable spin in the presence of the AFDIA scheme. Using the AFDIA scheme, the aircraft behaves like it is resisting the rotation along the roll axis after the loss of wing surface. The end result of this resistance is the gradual turn of the aircraft along its roll axis, which increases the flight duration following failure when compared against the X-Plane autopilot. Although the ideal results for the AFDIA scheme is not achieved, the scheme managed to perform better than the X-Plane autopilot, by avoiding an uncontrollable spin.

The scheme successfully managed to increase the duration of the flight after the failure, therefore adding endurance to the aircraft. For the purpose of this discussion, the flight duration between the time of failure and crash of the aircraft is used to quantify the endurance following failure.

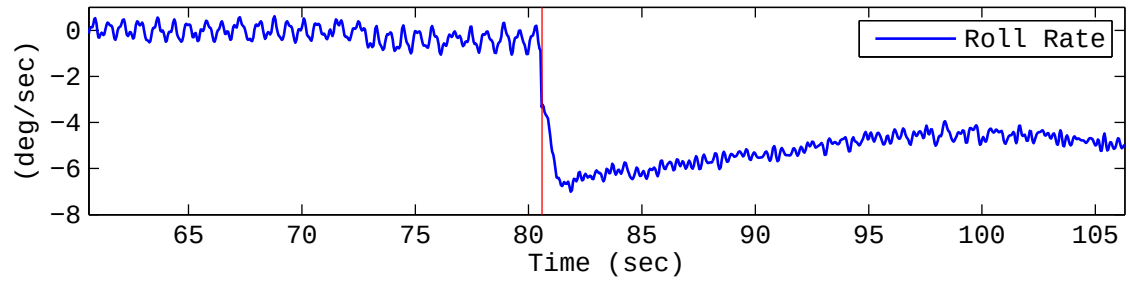
6.6.2 Flight Duration

As mentioned earlier, in the experiments conducted two parameters of the AFDIA scheme are varied to understand the effects on the performance of the scheme. The performance of the scheme is evaluated in terms of the flight duration following the failure. Flight duration is used as a measure of endurance added to aircraft following the failure. The parameters varied are namely, the scalar multiple (c) in equation (6.3) and the size (n) of the sliding data window used to store data for on-line adaptation of the NN roll controller. From the experiments conducted, it is clear that depending on the value of these parameters, the flight duration following failure can vary.

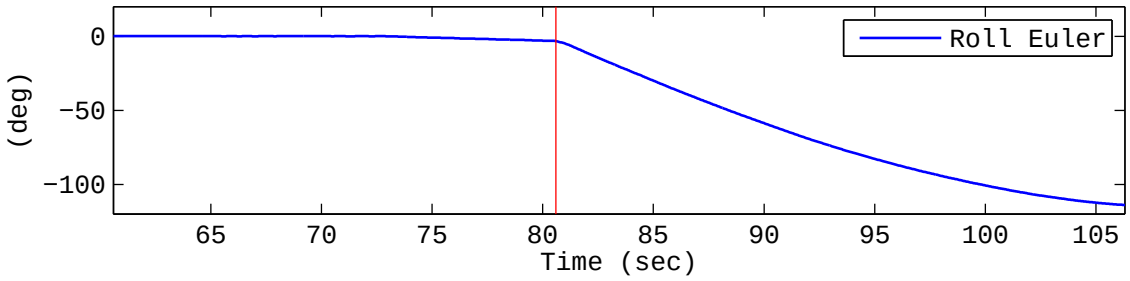
Table 6.4 presents a summary of the flight duration following failure for the



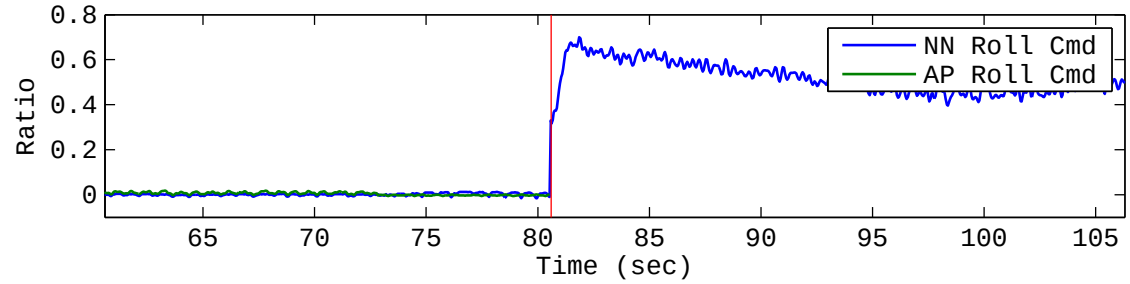
(a) Roll Acceleration



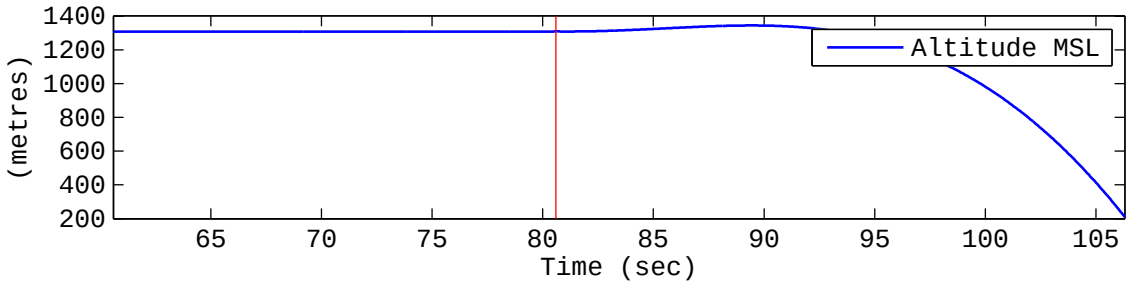
(b) Roll Rate



(c) Roll Euler

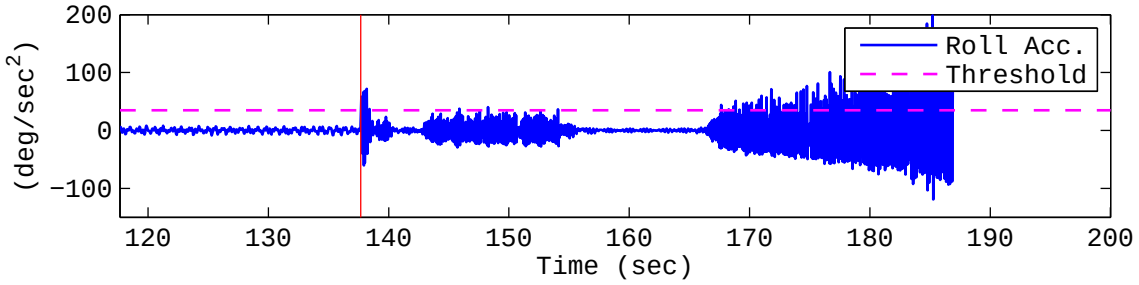


(d) Roll Command

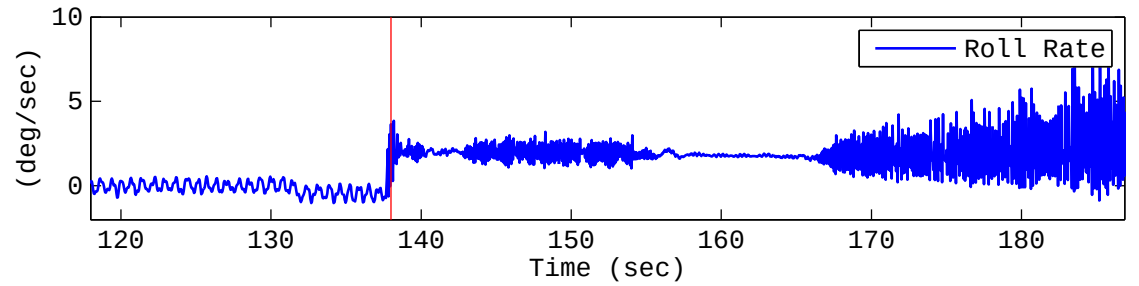


(e) Mean Sea Level (MSL) Altitude

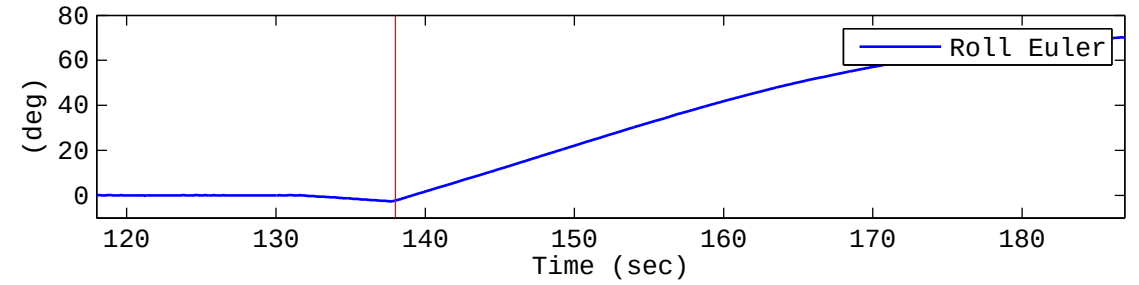
Figure 6.8: Aircraft Performance Results for Left Wing Failure. Window size $n = 10$, scalar multiple $c = 1$. The Red line marks when the failure was injected.



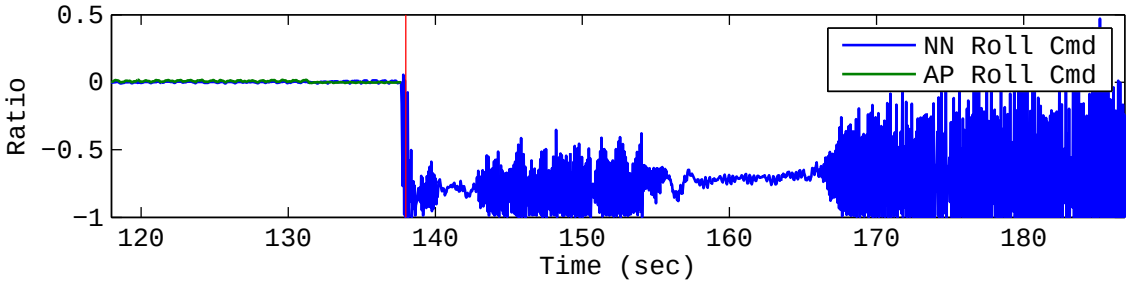
(a) Roll Acceleration



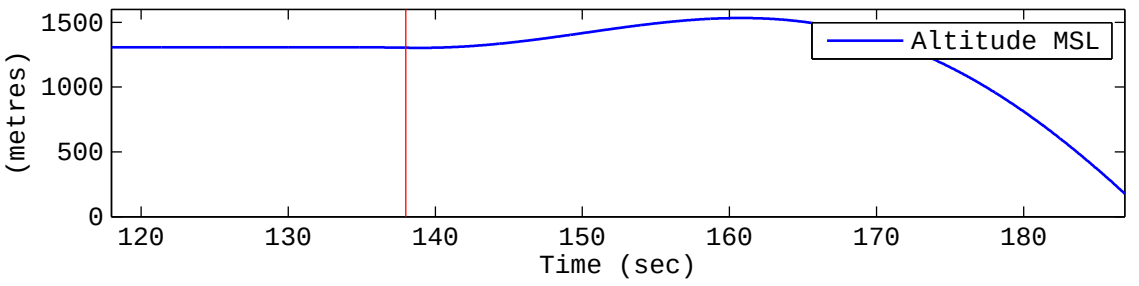
(b) Roll Rate



(c) Roll Euler



(d) Roll Command



(e) Mean Sea Level (MSL) Altitude

Figure 6.9: Aircraft Performance Results for Right Wing Failure. Window size $n = 15$, scalar multiple $c = 4$. The Red line marks the time of failure.

Table 6.4: Summary of the flight duration post wing surface loss failure. Each mean and standard deviation value presented here for the left and right wing failure is the average of 10 experiments for the set combination of the window size (n) and scalar multiple (c). Note that the colour coding highlights the values plotted in Figure 6.10.

		Duration of Flight (sec)					
		Left Wing Fail		Right Wing Fail		Average of Both Wings	
Win. Size	Scalar	Mean	SD	Mean	SD	Mean	SD
5	1	26.7328	0.7157	26.6202	0.9546	26.6765	0.8351
	2	35.6707	1.2559	36.3323	0.7429	36.0015	0.9994
	3	42.1959	1.8844	45.3088	0.4802	43.7523	1.1823
	4	45.9344	1.1483	50.2325	1.1866	48.0835	1.1675
	Average	37.6334	1.2511	39.6234	0.8411	38.6284	1.0461
10	1	25.1774	0.4396	27.0794	0.4672	26.1284	0.4534
	2	33.6245	0.4930	36.2687	0.3838	34.9466	0.4384
	3	41.2443	0.5774	44.0469	0.4948	42.6456	0.5361
	4	46.8332	0.5794	49.9794	0.5523	48.4063	0.5658
	Average	36.7198	0.5223	39.3436	0.4745	38.0317	0.4984
15	1	25.1754	0.2899	27.0285	0.3325	26.1019	0.3112
	2	34.3543	0.3700	36.1696	0.6446	35.2620	0.5073
	3	41.9601	0.4727	44.6866	0.6155	43.3233	0.5441
	4	46.6222	0.9746	50.0919	0.8963	48.3570	0.9355
	Average	37.0280	0.5268	39.4941	0.6222	38.2611	0.5745

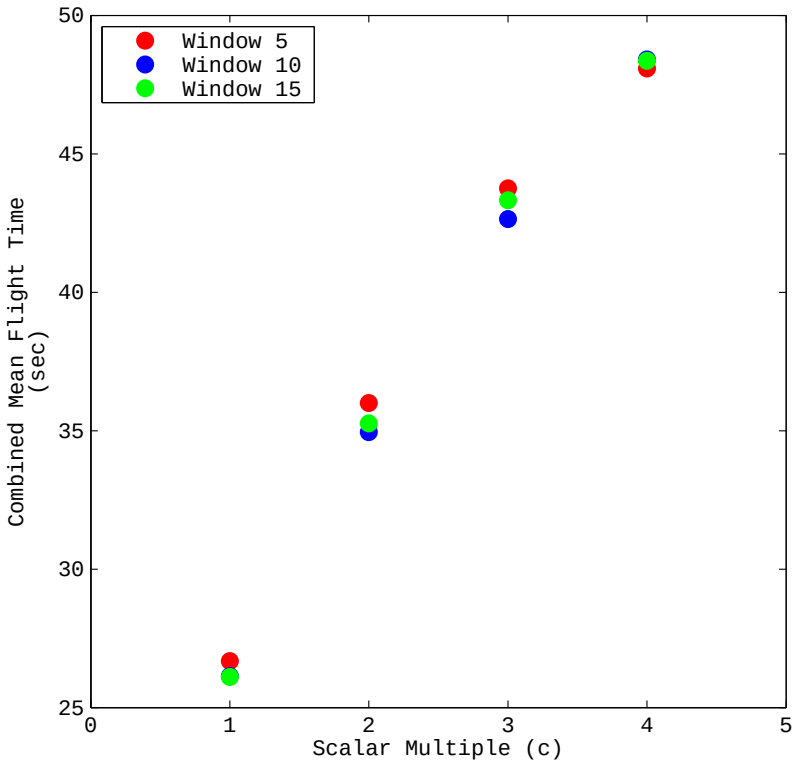


Figure 6.10: The relationship between the flight time and the parameters scalar multiple and window size.

experiments conducted. In this table, the mean and standard deviation of flight duration after the failure is organised based on the change in the scalar multiple (c) for every sliding data window size (n). Note that each mean and standard deviation value presented in the table is the average of 10 experiments for every combination of sliding data window size (n) and scalar multiple (c). The individual results of the 240 experiments are presented in Appendix A.

From the results in Table 6.4, it is clear that for a set combination of n and c parameters, the flight duration is almost similar regardless of the side of the failure. For example, for $n = 5$ and $c = 1$, the flight duration post failure is 26.7 sec and 26.6 sec, for left and right wing surface loss, respectively. Similarly, for $n = 15$ and $c = 3$, the flight duration post failure is 41.9 sec and 44.7 sec, for left and right wing surface loss, respectively. This is expected as both the wings lose the same amount of wing section in the failure simulation. Since the amount of surface loss is the same, both the wings experience similar loss of lift force and therefore similar failure induced rolling moment. The minor differences in the flight duration between the two wings can be attributed to environmental factors, such as the direction of the

wind.

Examining the results further reveals the extent of influence the c and n parameters have on the flight duration post failure. If the scalar multiple c is kept constant and the window size n is varied, the change in the flight duration is almost negligible. For example, for $c = 1$ and $n = \{5, 10, 15\}$, the combined average (left and right wing) flight durations are {26.8, 26.1, 26.1} sec, respectively. However, if the window size n remains constant and the scalar multiple c is varied, there is a significant increase in the flight duration from $c = 1$ to $c = 4$. For $n = 5$, the combined average (left and right wing) flight durations are {26.7, 36 , 43.7, 48} sec, for scalar multiple $c = \{1, 2, 3, 4\}$ respectively. As the scalar multiple increases, the flight duration increases. The durations are almost similar for the same scalar multiple across different window sizes. These observations are clearly represented in Figure 6.10. For different window sizes the flight duration remains almost the same, when the scalar multiple is constant. However, with the window size constant, increasing the scalar multiple results in an increase in the flight duration.

The sliding data window stores variables for n time steps, which is used to adapt the NN controller on-line following a failure. The mechanism enables the use of historical data in the on-line adaptation process which could benefit the adapting controller. From the results presented in Table 6.4 and Figure 6.10, it is clear that changing the number of historical data points (i.e window size n), has negligible effect on the flight duration following the failure.

The purpose of the scalar multiple c is to magnify the error that is used to guide the on-line adaptation of the roll controller following failure. From Table 6.4 and especially from Figure 6.10, it can be seen that the flight duration following the failure is directly proportional to the scalar multiple c. In equation (6.3), the scalar multiple c is used to multiply the difference between the desired and actual roll rate ($p_{ref} - p$), to generate the J_{roll} value used to guide the adaptation of the roll controller on-line. For a set difference between the desired and actual roll rate, as the scalar multiple c increases, the value of J_{roll} increases. The output of the NN roll controller is proportional to the J_{roll} value. Therefore, the NN roll controller output

will increase proportionally to the value of the scalar multiple c. This is observable from the Figures 6.8 and 6.9.

In Figures 6.8 and 6.9, the scalar multiple c is 1 and 4, respectively. In both these figures, notice the maximum output generated by the NN roll controller immediately after the failure. In case of Figure 6.8, where $c = 1$, the maximum output is in the range of + 0.7. In comparison, in Figure 6.9, where $c = 4$, the output reaches the maximum possible controller output value of − 1. With the increase in the value of c, the maximum output of the NN roll controller increases. This increase in the controller output means that the aileron in the healthy wing is deflected further in Figure 6.9 than in Figure 6.8, to generate a greater resisting rolling moment. The result of this greater resistance is an increase in the flight duration following failure. Therefore, with increasing value of the scalar multiple c, the flight duration after failure increases.

6.6.3 Failure Detection Time

Failure detection is one of the important stages of any fault tolerant system. In case of severe failure such as the loss of wing surface, quick detection is very important. The sooner the failure is detected, the quicker the system can respond to compensate for the failure. In the presented AFDIA scheme, failure is detected when the roll acceleration ($\dot{p}$) measurements crosses a fixed threshold $\tau = 35$ deg/sec^2. In Table 6.5, the detection time for each of the 240 experiments conducted is summarised. Each mean and standard deviation value is the average of 10 experiments for a set window size (n) and scalar multiple (c). The results for each of the individual experiments is presented in Appendix B for reference.

Since the failure detection stage is independent from the failure accommodation (FA) stage, the detection time is not affected by the changing c and n values. From Table 6.5, it can be said that the combined average mean failure detection time of both the wings is around 0.04 sec In some instances, the mean detection time is slightly greater than 0.04 sec and less in others. Overall, the detection time is within 1 or 2 time steps of the AFDIA execution, which is set to execute every 0.04 sec or

Table 6.5: Summary of the failure detection time.

Failure Detection Time (sec)								
		Left		Right		Average of Both Wings	
Window	Scale	Mean	SD	Mean	SD	Mean	SD
5	1	0.0406	0.0145	0.0415	0.0194	0.0410	0.0170
	2	0.0415	0.0176	0.0389	0.0143	0.0402	0.0160
	3	0.0389	0.0112	0.0372	0.0142	0.0381	0.0127
	4	0.0477	0.0152	0.0468	0.0142	0.0473	0.0147
	Average	0.0422	0.0146	0.0411	0.0155	0.0416	0.0151
10	1	0.0457	0.0140	0.0410	0.0100	0.0434	0.0120
	2	0.0490	0.0105	0.0474	0.0154	0.0482	0.0130
	3	0.0452	0.0133	0.0391	0.0152	0.0421	0.0143
	4	0.0437	0.0138	0.0423	0.0126	0.0430	0.0132
	Average	0.0459	0.0129	0.0425	0.0133	0.0442	0.0131
15	1	0.0317	0.0095	0.0346	0.0128	0.0332	0.0111
	2	0.0457	0.0180	0.0438	0.0141	0.0448	0.0161
	3	0.0432	0.0120	0.0389	0.0122	0.0410	0.0121
	4	0.0430	0.0157	0.0441	0.0131	0.0436	0.0144
	Average	0.0409	0.0138	0.0403	0.0131	0.0406	0.0134

25 Hz.

In all the 240 experiments conducted, the loss of wing surface failure was quickly and successfully detected, using the simple fixed threshold based fault detection mechanism. However, the fault detection mechanism employed in the AFDIA does have some limitations. As with any fixed threshold based mechanism, the challenge is to identify a good fault detection threshold. If the threshold is too low (i.e. too close to the norm), the probability of false failure detection increases. On the other hand, if the threshold is too high, the failure might take longer to be detected or worse, go undetected. The failure detection threshold (τ) for the AFDIA scheme presented here is set to $\tau = 35$ deg/sec, based on the discussion in Section 6.2.2.

Additionally, in the experiments conducted, the failure was simulated when the aircraft was maintaining a straight and level flight. This limitation of the experiment is a good reason why no false failure detection was triggered in the 240 experiments conducted. For example the roll acceleration measurements ($\dot{p}$) could have crossed the failure threshold τ when the aircraft was making a steep turn or pitch or was affected by environmental factors. Crossing the threshold τ would result in a failure detection, in the absence of a failure.

Further study needs to be conducted to develop a robust mechanism to detect loss of wing surface failure during any phase of the flight. One possible solution is to develop a adapting threshold, which will adapt to the phase of the flight with time. Another approach could be the use of multiple thresholds monitoring various signals (not just $\dot{p}$) to identify a loss of wing surface failure.

6.6.4 AFDIA Execution Time Analysis

One of the important aspects to consider for an AFDIA scheme is the time it takes to execute. In Tables 6.6 and 6.7, the execution time of the AFDIA scheme for left and right wing failures respectively, are summarised. The tables compare the execution time before and after failure. To enable the comparison, the results are based on data 20 sec before and after failure. Each of the results presented is the average of 10 experiments for a set combination of window size (n) and scalar multiple (c).

The results for the individual experiments are presented in Appendix C.

From the results presented in these tables, it can be said that on average the execution time of the AFIDA scheme before failure is about 7.36 μsec. Following a failure, the execution time increases to about an average of 0.455 msec. This increase in execution time is due to the on-line adaptation of the NN roll controller. It is expected that the execution time will increase with increasing window size (n). This is because there are more data with increasing window size (n) that would need to be processed for adapting the controller. However, at these time resolution (around 0.1 msec), their differences are insignificant.

Table 6.6: Summary of AFDIA execution time before and after loss of left wing surface. Each mean and standard deviation value presented here is based on the average of 10 experiments for the set combination of window size (n) and scalar multiple (c).

		Before Fail (sec)		After Fail (sec)	
Window	Scalar	Mean	SD	Mean	SD
5	1	7.41 μ	4.62 μ	0.477 m	0.455 m
	2	7.69 μ	3.11 μ	0.599 m	0.403 m
	3	7.03 μ	2.89 μ	0.396 m	0.239 m
	4	7.65 μ	1.30 μ	0.463 m	0.261 m
Average		7.45 μ	2.43 μ	0.484 m	0.340 m
10	1	7.41 μ	5.50 μ	0.432 m	0.388 m
	2	7.32 μ	0.379 m	0.497 m	0.404 m
	3	7.21 μ	3.64 μ	0.389 m	0.197 m
	4	7.07 μ	3.68 μ	0.392 m	0.185 m
Average		7.25 μ	0.0980 m	0.428 m	0.294 m
15	1	7.27 μ	5.67 μ	0.440 m	0.376 m
	2	7.24 μ	4.41 μ	0.414 m	0.334 m
	3	7.33 μ	3.24 μ	0.450 m	0.204 m
	4	7.49 μ	5.06 μ	0.416 m	0.230 m
Average		7.33 μ	4.60 μ	0.430 m	0.286 m

At the beginning of the AFDIA experiment setup section (see Section 6.5.1), it was mentioned that the AFDIA scheme was set to execute at 25 Hz, although the NN roll controller was developed based on data collected at 50 Hz. This limit was applied due to the timing constrains imposed by the X-Plane simulator execution demands and the need to log various simulation data. From the results it is clear that the scheme can comfortability be implemented within 50 Hz (0.04 sec), even when the NN controller is adapting post failure.

Table 6.7: Summary of AFDIA execution time before and after loss of right wing surface. Each mean and standard deviation value presented here is based on the average of 10 experiments for the set combination of window size (n) and scalar multiple (c).

		Before Fail (sec)		After Fail (sec)	
Window	Scalar	Mean	SD	Mean	SD
5	1	7.49 μ	3.12 μ	0.387 m	0.321 m
	2	6.91 μ	2.24 μ	0.398 m	0.275 m
	3	7.44 μ	2.90 μ	0.527 m	0.279 m
	4	7.76 μ	1.91 μ	0.521 m	0.259 m
Average		7.40 μ	2.54 μ	0.458 m	0.284 m
10	1	7.41 μ	6.51 μ	0.367 m	0.205 m
	2	7.59 μ	3.10 μ	0.501 m	0.188 m
	3	7.66 μ	2.63 μ	0.552 m	0.302 m
	4	7.01 μ	2.65 μ	0.406 m	0.180 m
Average		7.42 μ	3.72 μ	0.457 m	0.219 m
15	1	7.54 μ	4.70 μ	0.570 m	0.328 m
	2	7.15 μ	2.22 μ	0.445 m	0.241 m
	3	7.37 μ	3.44 μ	0.473 m	0.195 m
	4	7.21 μ	3.18 μ	0.415 m	0.158 m
Average		7.32 μ	3.39 μ	0.476 m	0.231 m

6.7 Reflective Improvement of the AFDIA Scheme

Based on the observations made from the results of the 240 experiments conducted, improvements are made to the AFDIA scheme. These improvements are discussed in this section.

6.7.1 Euler angle based error function

In the AFDIA scheme, the J_{roll} function (see equation (6.3)) used to guide the on-line adaptation of the NN roll controller is based on the roll rate (p) of the aircraft. Following the loss of wing surface, the aircraft will go into an uncontrollable spin. The roll rate is used based on the assumption that if the aircraft stops spinning and is in equilibrium, the roll rate would be approximately 0 deg/sec. However, there are certain disadvantages of using the roll rate for guiding the adaptation process.

To understand the downside of using the roll rate, consider the roll rate and NN roll command ($\hat{\delta}$) plots in Figures 6.8 and 6.9. Notice that in Figure 6.8, for small fluctuations in the roll rate, the J_{roll} value would fluctuate accordingly, resulting in small fluctuations in the adapting NN roll controller output. When the fluctuations are small, the NN controller adaptation process fluctuates accordingly within small ranges. What happens if the fluctuations are large?

Such a case is observable in Figure 6.9, where the value of the scalar multiple c is 4. The scalar multiple c magnifies the fluctuations in the roll rate, resulting in large fluctuations in the J_{roll} value calculated using equation (6.3). These large changes in the J_{roll} value result in the adaptation process of the NN roll controller fluctuating significantly, resulting in significant fluctuations in the NN roll controller output. The significant fluctuation in adapting controller output produces proportional fluctuations in the roll rate, which then feeds back to the J_{roll} function and the outcome repeats. These fluctuations in the J_{roll} value (driven by the roll rate) are counter productive to the NN adaptation process and could further destabilise the aircraft. The importance of the scalar multiple c in the J_{roll} function was shown in the results discussion in Section 6.6.2 where, with increasing value of c, the flight duration following failure increased. However, from the discussion in this section, it

is clear that use of roll rate in the J_{roll} function could be counter productive to the objective of the AFDIA scheme.

Reflecting on these observations from the AFDIA experiments conducted, the J_{roll} function to guide the adaptation of the NN roll controller is modified to use the roll Euler angle (ϕ) instead. A J_{roll} function using the Euler angle (ϕ) is linear compared to one based on the roll rate (p). Following a loss of wing surface, the objective of the AFDIA scheme is to use the aileron on the healthy wing to compensate for the failure induced rolling moment and try to stabilise the aircraft. To achieve this using the Euler angle based J_{roll} function, the NN roll controller is adapted to try to level the aircraft where the roll Euler angle is 0 deg. Therefore the adaptation guiding function in equation (6.3) can be written as follows:

$$J_{roll} = c \times m(\phi_{ref} - \phi) \tag{6.6}$$

where:

c : is a scalar multiple

ϕ : is the roll Euler angle in deg

ϕ_{ref} : is the desired roll Euler angle, which is 0 deg in level flight

$m(x)$: $2 \times \frac{x + 30}{60} - 1$

The $m(x)$ function as described in equation (3.21) scales the difference between the desired and actual roll Euler angle ($\phi_{ref} - \phi$) to the range of $\pm$ 1. This function assumes that the maximum and minimum value of the roll Euler angle will in the $\pm$ 30 deg range. If the Euler angle exceeds this range, then the function will scale it accordingly.

6.7.2 On-line Learning and Stopping Condition

The FCC NN based roll controller is adapted on-line using the NBN learning algorithm and sliding data window technique. The on-line adaptation of the controller is similar to the off-line training. In the case of the off-line training, a large dataset is used to train the controller. Similarly in on-line learning, a sliding data window

is used to adapt the controller. However, the size of the sliding data window is significantly smaller than the dataset used to train the controller off-line.

When the controller is trained off-line, the error to guide the training process of the controller is derived as follows:

$$e_{sum} = \sum^{p} d_p - o_p \tag{6.7}$$

where e_{sum} is the error calculated by summing the error between the desired (d) and actual output (o), for every pattern (p) or samples in the dataset. In the off-line training phase, the NN controller is learning to mimic the output of the flight controller roll command. This is the desired output (d) and the NN controller output (o) is the actual output. However in the presented AFDIA scheme, when the controller is adapting on-line following a failure, the flight controller roll command cannot be used as the desired output (d), since the dynamics of the system have changed. Instead a desired value is derived from the post failure system using equation (6.3). Hence, equation (6.7) can be rewritten as

$$e_{sum} = \sum^{n} J_{roll_n} - o_n \tag{6.8}$$

where e_{sum} is the error calculated by summing the error between the function (J_{roll}) and actual output (o) of the NN controller, for every pattern or number of time steps (n) in the sliding data window. As mentioned earlier, J_{roll} is the guiding function for on-line adaptation of the NN controller. In the AFDIA experiments presented in Section 6.6, this function derived its value from the error between the reference (p_{ref}) and actual (p) roll rate. This function has since been updated in the previous section to derive its value from the difference between the desired (ϕ_{ref}) and actual (ϕ) roll Euler angle (see equation (6.6)).

In the J_{roll} function, what happens if the difference approaches 0, as the controller adapts to bring the aircraft to equilibrium? The guiding error function J_{roll} tends towards 0 and if the adaptation continues, the on-line training error value could be approximated as

$$e_{sum} \approx \sum^{n} \; - o \tag{6.9}$$

essentially feeding the adapted NN controller output as the error itself. This would degrade the adapting NN controller from the adapted controller state. Therefore a stopping condition is introduced in the AFDIA scheme to stop the adaptation at some point. To stop the adaptation process, the NN controller output ($\hat{\delta}_A$) is continuously monitored. The condition for stopping the adaptation process can be stated as follows:

*"**Stopping Condition:** If the controller output ($\hat{\delta}_A$) converges to a value and remains within a tolerance range for a set period of time (t), the adaptation process can be terminated."*

In essence, this condition checks if the controller output has converged to a value and therefore stop any further adaptation of the NN controller. In the improved AFDIA scheme, the time period for this condition to be satisfied is set to $t = 1$ sec. Since the scheme executes at 25 Hz, this condition has to be satisfied 25 times continuously before the adaptation is terminated.

This stopping condition is implemented by using an error value, $SCON_{err}$ and error tolerance range γ . To check if the controller output has converged and is within a tolerance error range, the error value $SCON_{err}$ is calculated after every adaptation of the controller, by subtracting the current output ($\hat{\delta}_A$) after adaptation from the average of the previous consecutive outputs when the condition was satisfied. For example, at time k, where $k >$ time of failure (T_{fail}), the error value $SCON_{err}$ is calculated as follows:

$$SCON_{err} = \hat{\delta}(k) - \frac{\sum_{i=1}^{m} \hat{\delta}(k-i)}{i} \tag{6.10}$$

where i is a counter value that tracks the number of consecutive times the value of $SCON_{err}$ was within the tolerance range γ. Based on limited experiments conducted

to find a suitable value for the tolerance range, γ is set to 0.1. The maximum value of i is $m = 25$, as the stopping condition has to be satisfied 25 times. If the difference between the two are within the defined tolerance range γ, the counter value i is incremented. If the count i reaches the maximum value, $i = m = 25$, indicating that the stopping condition is satisfied for the stated time period, the adaptation is terminated. However, if the error $SCON_{err}$ is not within the γ range, the count value (i) is reset to 1, and the sum of the previous output values is reset to the current output. This stopping condition process can also be described in a pseudo code form as presented in Figure 6.11.

```
1  if first time of execution then
2  |   i = 1;
3  |   sum of previous output = current output;
4  else
5  |   calculate the error between the current output and average of the previous
   |   output SCON_err = δ̂(k) − (Σ_{i=1}^{m} δ̂(k−i)) / i
6  |   if SCON_err <= γ then
7  |   |   i = i + 1;
8  |   |   if i = m then
9  |   |   |   Stop Adaptation;
10 |   |   |   Adapt = 2;
11 |   |   end
12 |   else
13 |   |   Reset the count;
14 |   |   i = 1
15 |   end
16 end
```

Figure 6.11: Pseudo code for the on-line adaptation stopping condition.

In the improved AFDIA scheme, a state flag called '*Adapt*' is used to implement the stopping condition. This state indicator flag has 3 possible values, namely, 0, 1 and 2. When no failure is detected by the FDI stage of the AFDIA scheme, *Adapt* is set to 0, indicating that there is no need for adapting the controller. If a failure is detected in the FDI stage ($F_{L/R} = 1$), the *Adapt* flag is set to 1 to indicate that the NN controller is adapting on-line following a failure. Once the stopping condition for terminating the adaptation process is satisfied, the state of the *Adapt* is set to 2.

In the next section, the operational outline of the improved AFDIA scheme is presented.

6.7.3 Improved AFDIA Scheme Operational Overview

In Figure 6.12 the flow chart for the improved AFDIA scheme is presented. As with the original scheme, the first step is failure detection and identification (FDI), where the roll acceleration ($\dot{p}$) sensor measurements are monitored. Failure is detected when the roll acceleration measurement crosses a fixed threshold of $\tau = 35$ deg/sec^2. Once the failure is detected, the failed wing is identified using the direction of the fluctuation in the roll acceleration ($\dot{p}$) measurements.

The AFDIA scheme has two failure flags, F_L and F_R, for the left and right wing failure, respectively. If a failure is detected, the $F_{L/R}$ flag sets to 1, otherwise the flag remains at the default value of 0. In addition, the scheme has a *Adapt* flag, which helps to implement the stopping condition. If no failure is detected, there is no need for the NN controller to adapt, therefore $Adapt = 0$. If a failure is detected, the need for adapting the NN controller is indicated by setting $Adapt = 1$. Therefore the FDI stage of the AFDIA scheme can be summarised as follows:

$$F_L, Adapt = \begin{cases} 1, & \text{if } \dot{p} \leq -\tau \\ 0, & \text{otherwise} \end{cases} \tag{6.11}$$

$$F_R, Adapt = \begin{cases} 1, & \text{if } \dot{p} \geq +\tau \\ 0, & \text{otherwise} \end{cases} \tag{6.12}$$

After the FDI stage of the scheme, the sliding data window is updated with the variables of the current time step. Similar to the original scheme, the fourth variable stored in the data window depends on the outcome of the FDI stage. If no failure is detected, (i.e. $F_{L/R} = 0$), then the fourth variable stored in the window is the roll control command ($J_{roll} = \delta_A$). However, if a failure is detected (i.e. $F_{L/R} = 1$), then the fourth variable stored is the J_{roll} value calculated using equation (6.6).

In the failure accommodation (FA) stage, the NN roll controller adapts to the

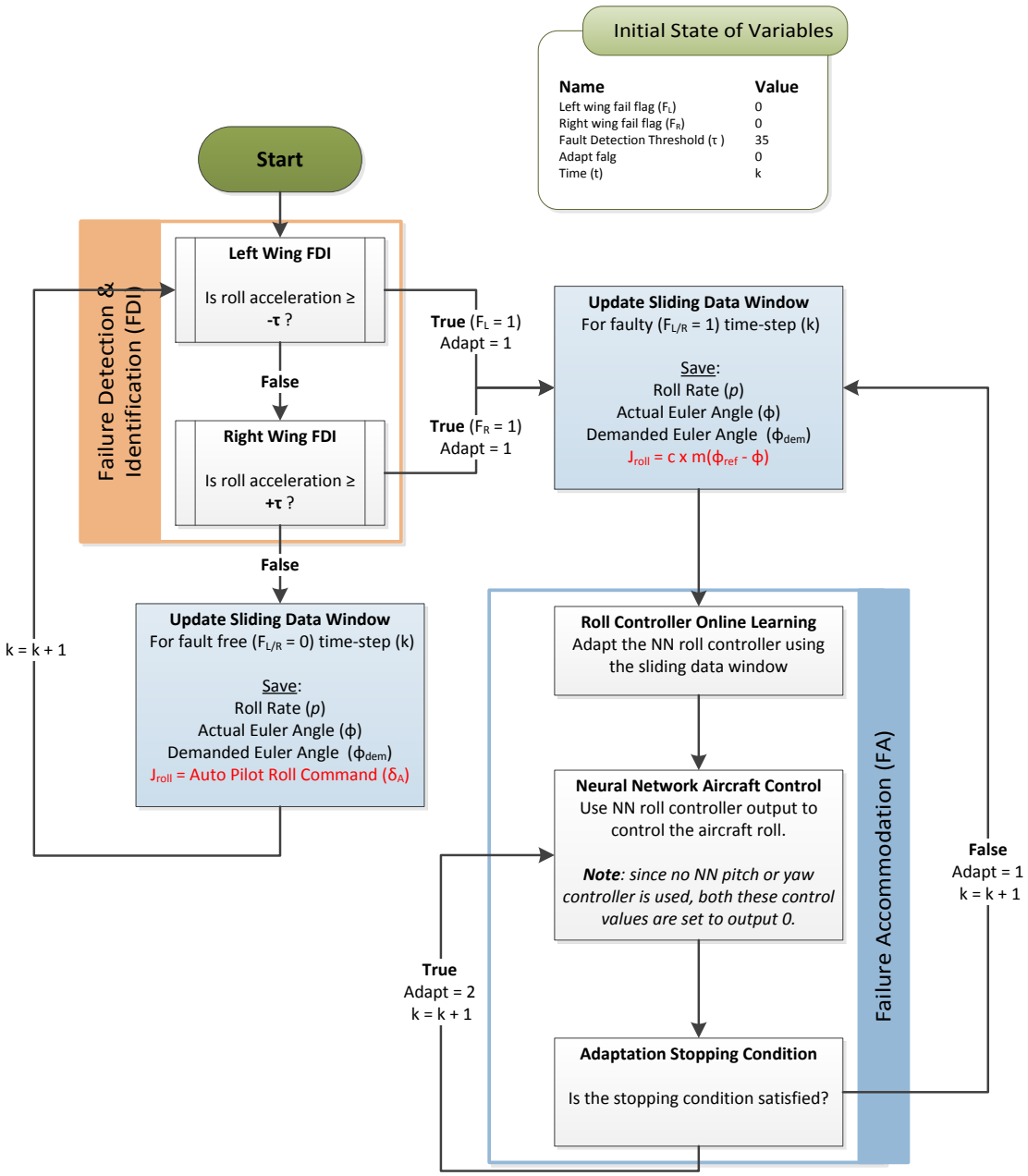


Figure 6.12: The Improved AFDIA Scheme.

post failure dynamics of the aircraft. The objective is to adapt to use the aileron on the healthy wing to balance the failure induced rolling moment and stabilise the aircraft. To achieve this objective the controller aims to return the aircraft to level flight condition, where $\phi = 0$ deg. The adaptation of the roll controller is indicated by $Adapt = 1$. Once the stopping condition is satisfied, the adaptation process is terminated, indicated by $Adapt = 2$.

6.8 Improved AFDIA Experimental Setup and Overview

The implementation of the improved AFDIA scheme in X-Plane and the simulation conditions are the same as the experiments for the original design of the AFDIA scheme. For the improved AFDIA scheme, only 20 experiments are conducted, 10 each for the loss of left and right wing surface. In these experiments, the sliding data window size (n) and the scalar multiple (c) are kept constant at 15 and 4, respectively. Each of these experiments were conducted for a duration of 5 minutes each.

6.9 Results and Discussion

6.9.1 Post Failure Aircraft Behaviour

Following the loss of wing surface, the AFDIA scheme attempts to bring the aircraft back to equilibrium and maintain flight. To do this the scheme uses the remaining aileron on the healthy wing and tries to achieve level flight (where $\phi = 0$ deg), to produce the required moment to compensate for the failure induced rolling moment. As mentioned during the development of the original AFDIA scheme, an ideal scenario would be if the aircraft maintains flight following the loss of wing surface. Using the improved AFDIA scheme, this ideal scenario is achieved.

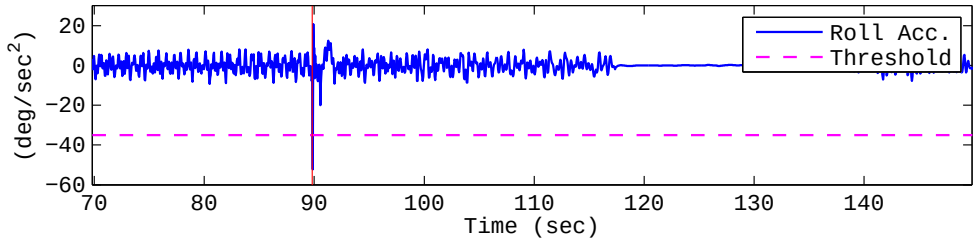
Figure 6.13 presents one of the AFDIA scheme results following the loss of left wing surface failure. Note that the plots in figure starts from 20 sec before failure and ends 60 sec after failure. The time of failure is indicated by the red vertical line. Immediately after the failure, the roll Euler angle starts to change rapidly, indicating a rapid turn along the roll axis of the aircraft. As is obvious, the aircraft rolls to the left side following the failure. The rapid turn along the roll axis would generate significant acceleration, which is visible from the roll acceleration plot. This significant jump in the roll acceleration crosses the fault detection threshold immediately after the failure. Notice the change in the *Adapt* flag from state 0 to 1,

confirming the detection of the failure and therefore the need to adapt the NN roll controller.

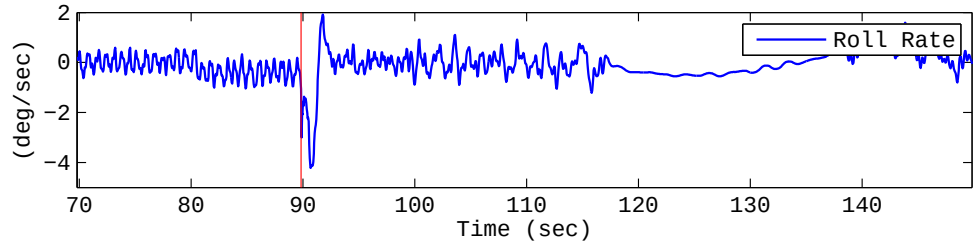
The NN controller adapts immediately following the failure, resulting in an increasing output command (output increases towards +1), until it settles at about +0.8. Notice the change in the state of the *Adapt* flag, which changes from 1 to 2, indicating that the NN controller has adapted to control the aircraft with the post failure dynamics. The result of the swift reaction from the NN controller is that the aircraft achieves a steady attitude and avoids an uncontrollable spin. This is observable from the roll Euler plot following the failure in Figure 6.13. From the roll Euler plot it is clear that although the roll attitude remains steady, the aircraft has not achieved level flight, where the roll Euler angle would be 0 deg. Instead, the aircraft maintains an almost steady roll attitude of about −6 deg.

The roll attitude stabilises when the failure induced rolling moment is balanced by the moment generated by the aileron on the healthy wing. In Figure 6.13, this is achieved when the roll Euler angle is about −6 deg. It must be mentioned that the aircraft is flying in a dynamic environment, simulated by the X-Plane weather system, where it is constantly effected by environmental conditions such as wind direction and speed among other weather phenomenon. With 66% of the wing surface missing, the aircraft is highly unstable to these dynamic changes in environmental forces which effect the rolling moment.

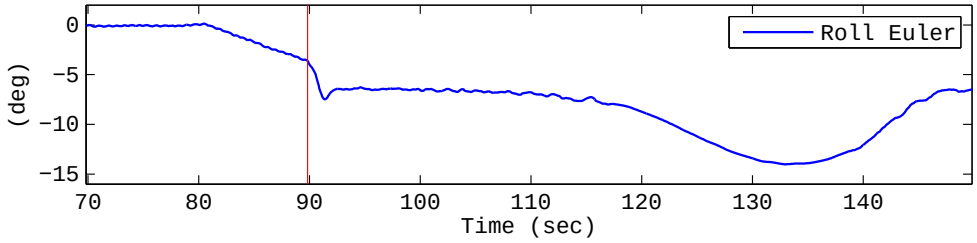
From the roll Euler plot in Figure 6.13, notice that after a while the Euler angle slowly drifts away from about −6 deg. To counteract this drift, the NN roll controller output slowly increases from about +0.8. At 120 sec, the Euler angle starts to gradually slide away, until a maximum of about −14 deg is achieved at approximately 135 sec. By 120 sec, the NN controller output is already at the maximum output value of +1, to counteract the change in Euler angle. This drift in the aircraft attitude and the subsequent changes are due to the change in the rolling moment that needs to be balanced. The dynamic environmental conditions are affecting the rolling moment that needs to be balanced to maintain a steady roll attitude. The NN controller deflects the aileron on the healthy wing to the



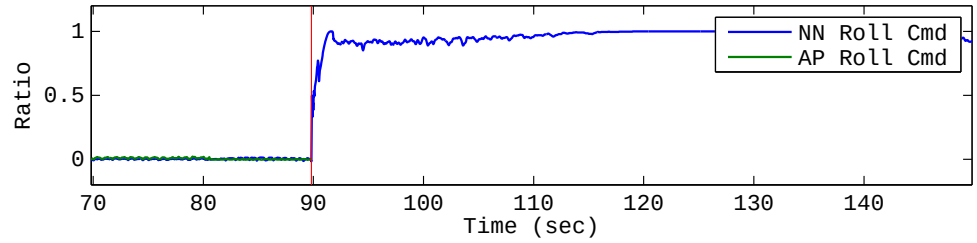
(a) Roll Acceleration



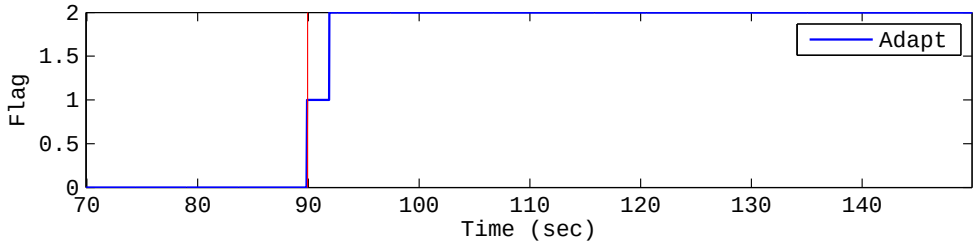
(b) Roll Rate



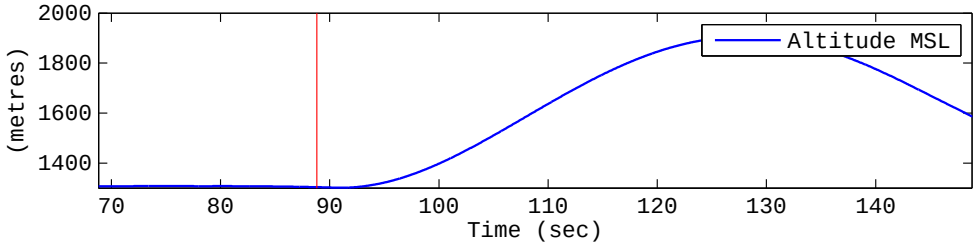
(c) Roll Euler



(d) Roll Command

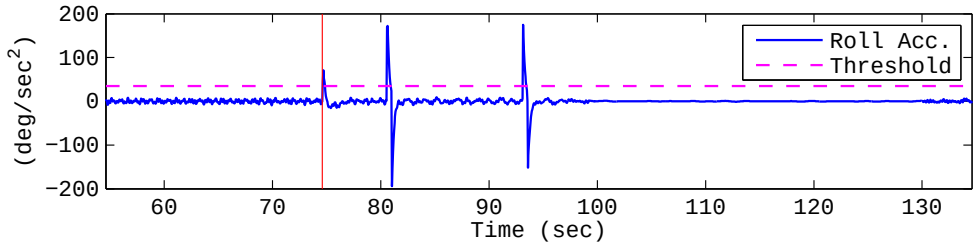


(e) Adapt Flag

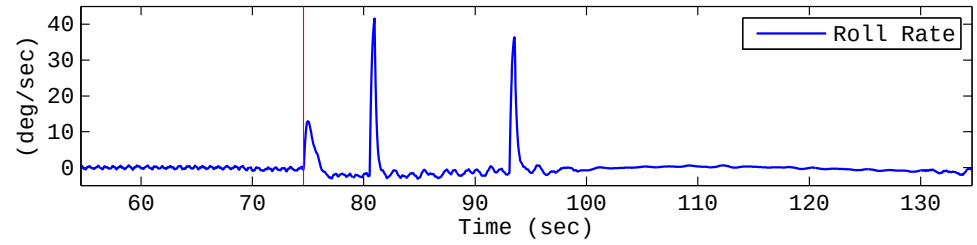


(f) Altitude

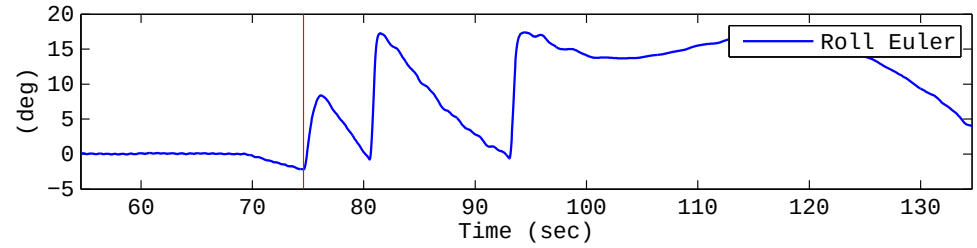
Figure 6.13: Aircraft Performance Results for Left Wing Failure. The Red line marks when the failure was injected.



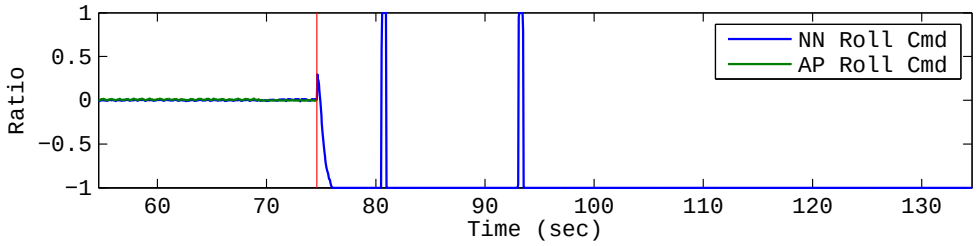
(a) Roll Acceleration



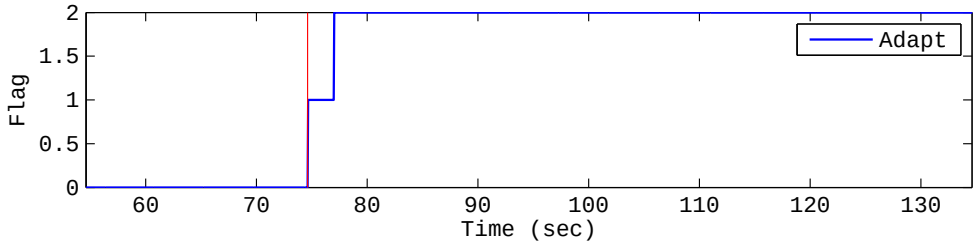
(b) Roll Rate



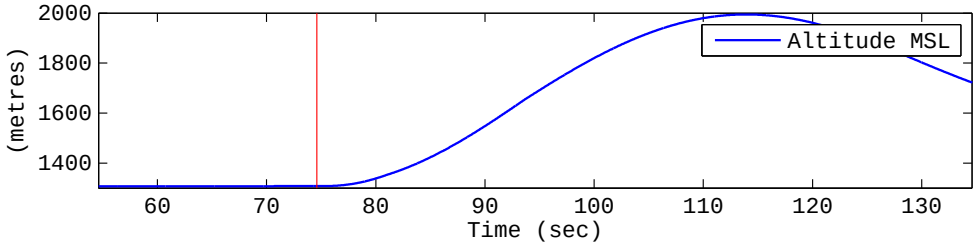
(c) Roll Euler



(d) Roll Command

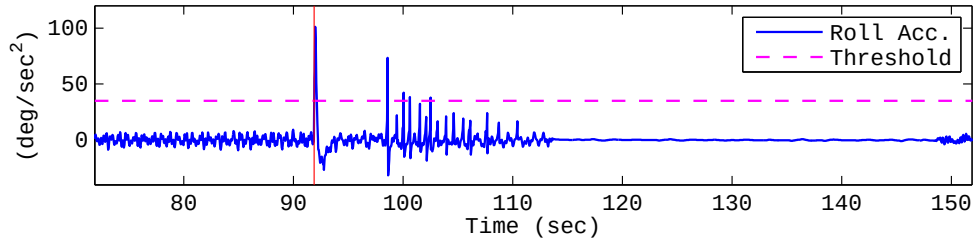


(e) Adapt Flag

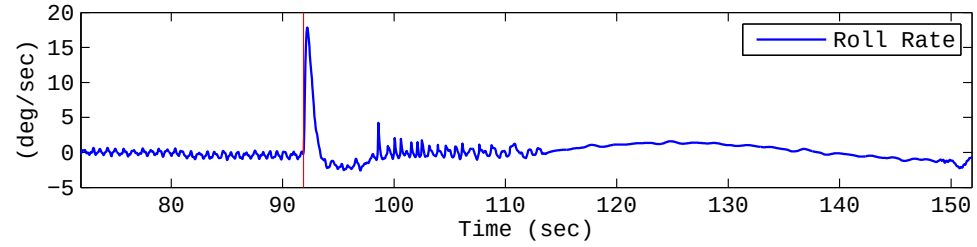


(f) Altitude

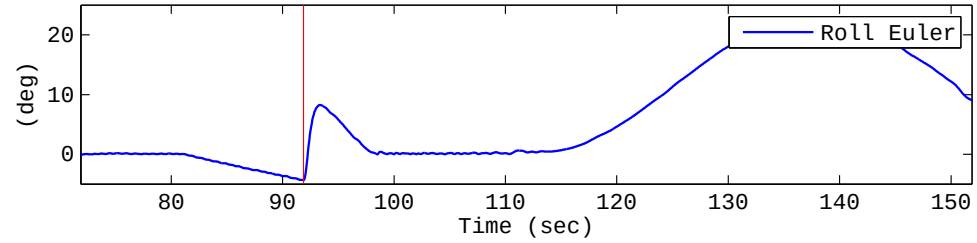
Figure 6.14: Aircraft Performance Results for Right Wing Failure. The Red line marks when the failure was injected.



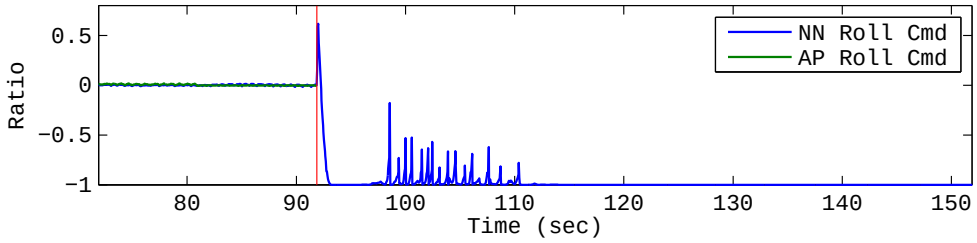
(a) Roll Acceleration



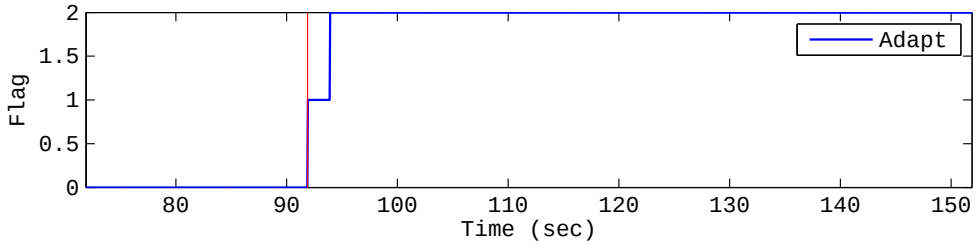
(b) Roll Rate



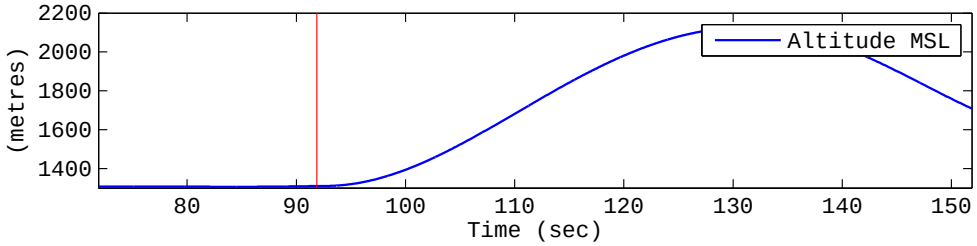
(c) Roll Euler Angle



(d) Roll Command



(e) Adapt Flag



(f) Altitude

Figure 6.15: Aircraft Performance Results for Right Wing Failure. The Red line marks when the failure was injected.

maximum, in order to compensate for the change in rolling moment. Eventually, the rolling moment that needs to be balanced changes and at about 140 sec, the Euler angle starts to return back towards the post failure steady state and settles at about −6 sec again.

In Figure 6.14 the simulation result following a loss of right wing surface is presented. In many ways the result is similar to the left wing failure in Figure 6.13. However, it does display some interesting behaviour, not present in Figure 6.13.

In Figure 6.14, the roll acceleration crosses the fault detection threshold (τ) and triggers the adaptation of the controller ($Adapt = 1$) immediately following the failure. The NN controller output rapidly increases to the maximum output of −1 in order to compensate for the failure induced rolling moment. Within a few seconds, the NN controller is adapted and any further adaptation is terminated, by setting $Adapt = 2$, as can be seen from the adapt flag plot.

From the Euler angle plot in Figure 6.14, notice that immediately following a failure, the Euler angle changes rapidly. Due to the response of the controller, the Euler angle peaks at about 8 deg, before returning towards level flight, where the roll Euler angle is 0 deg. Unlike the results presented in Figure 6.13, the aircraft approaches 0 deg roll attitude or level flight. However, an interesting behaviour is observed when the Euler angle approaches 0 deg.

Notice from the NN roll controller command plot in Figure 6.14 that as soon as roll attitude approaches 0 deg (at about 80 sec), the controller output flips to the maximum output of +1. This results in the Euler angle rapidly changing to a maximum of 17 deg, indicating a sudden rolling of the aircraft. The NN controller returns to the desirable output of −1, shortly after the flip. This results in the aircraft returning towards a level flight attitude. Once again, when the Euler angle approaches 0 deg (at about 93 sec), the NN controller outputs flips and the Euler angle changes rapidly.

This behaviour where the NN roll controller output flips when the Euler angle approaches 0 deg is what the stopping condition discussed in Section 6.7.2 was designed to avoid. This behaviour is a sign of the NN controller degradation from

the ideal adapted state of the controller. The degradation is the result of the on-line learning mechanism used to adapt the NN roll controller.

As described in Section 6.7.2, the on-line learning mechanism works on using the difference in the desired and the actual Euler angle (i.e. $\phi_{ref} - \phi$) in the J_{roll} function presented in equation (6.6). The NN roll controller is adapted on-line by using the difference between the J_{roll} value and the NN controller output as the learning error. After a failure, when the aircraft roll attitude approaches the Euler angle of 0 deg, the value of J_{roll} approaches 0. Around this time, if the on-line learning process continues, the output of the NN controller would be greater than J_{roll} and eventually feed itself as the learning error as described in equation (6.9). This would result in the degradation of the adapted controller that managed to bring the aircraft back to an Euler angle of 0 deg following a failure. Therefore a stopping condition was introduced (see Section 6.7.2) to terminate the learning process around this time. However, from the results presented in Figure 6.14, it is clear that the stopping condition was not effective in stopping the controller degradation. Although throughout the simulation the aircraft maintains flight following the failure, the ideal controller has been degraded to an extent.

Note that such a behaviour of controller degradation was not observed in the results presented in Figure 6.13 because the Euler angle never approached 0 deg. There are examples in the conducted experiments where the stopping condition was very effective in preventing the degradation of the adapted controller. As an example, consider the results presented in Figure 6.15 following a loss of right wing surface. At around 98 sec, the NN roll controller manages to bring the aircraft roll attitude to 0 deg, after the failure. Note that there are some signs of controller degradation from the fluctuations in the controller output. But these fluctuations are minor compared to the degradation in the results presented in Figure 6.14. In addition, the Euler angle remains steady at 0 deg during these minor fluctuations in controller output.

Figures 6.13, 6.14 and 6.15 are some of the results from the 20 experiments conducted using the improved AFDIA scheme. These results encapsulate the observed

behaviour of the aircraft post failure in all 20 experiments. Using the improved AFDIA scheme, during the duration of the experiments, the aircraft was able to maintain flight following about 66% loss of wing surface. This is a remarkable achievement compared to the original AFDIA scheme results presented in Section 6.6.

The improved AFDIA scheme implements a stopping mechanism to terminate the on-line adaptation of the NN roll controller, post failure. This has contributed to the success of the improved scheme. However, issues with this were highlighted in the above results discussion. Further study needs to be conducted to identify a better guiding function for the on-line adaptation of the controller. In conjunction with this, other mechanisms to terminate the on-line adaptation of the NN roll controller must be explored.

Although mentioned earlier, note that the AFDIA scheme only implements an NN roll controller. Following a failure, the pitch and yaw command output is set to 0. This decision was made due to the timing constraints on the research. In addition, implementing just the NN roll controller would enable the understanding of the behaviour of the aircraft post failure with just the adapting NN roll control. This knowledge could then be used in future work while developing the NN pitch and yaw controller.

As is obvious, setting the pitch and yaw control command output to 0 post failure does effect the behaviour of the aircraft following a failure. The aircraft does not maintain a fixed altitude or heading following a failure. Based on the effects of the environmental factors (e.g. wind direction) and with 66 % of the wing surface missing, the aircraft changes altitude and heading randomly. However, the NN roll controller keeps control of the aircraft roll post failure and avoids going into an uncontrollable spin.

6.9.2 AFDIA Execution and Adaptation Time

In the original AFDIA scheme presented in Section 6.4, the execution time was compared based on data 20 sec before and after failure. Such a comparison cannot be made here due to the addition of the stopping condition to terminate the adaptation

process following the failure. However, a fair comparison between the two AFDIA schemes can be made based on data before failure.

In Table 6.8, analysis of the AFDIA scheme execution time is presented based on 20 sec of data before the controller adaptation (i.e. before failure). This table presents the results for the 20 experiments conducted, 10 each for the left and right wing surface loss failure. On average, the execution time of the improved AFDIA scheme is about 7.45 μsec. This is not very different from the execution times of the original AFIDA scheme. At this time resolution, the change in the execution time based on the addition of the simple stopping condition is negligible.

Table 6.8: Analysis of the improved AFDIA execution time over 20 seconds before the controller adaptation.

	Left Wing		Right Wing	
Exp.No.	Mean (sec)	SD (sec)	Mean (sec)	SD (sec)
1	7.00 μ	5.00 μ	7.00 μ	1.00 μ
2	7.00 μ	2.00 μ	7.00 μ	1.00 μ
3	8.00 μ	9.00 μ	7.00 μ	2.00 μ
4	7.00 μ	1.00 μ	7.00 μ	1.00 μ
5	0.0120 m	0.130 m	7.00 μ	6.00 μ
6	7.00 μ	2.00 μ	7.00 μ	6.00 μ
7	7.00 μ	6.00 μ	7.00 μ	1.00 μ
8	8.00 μ	6.00 μ	7.00 μ	6.00 μ
9	7.00 μ	1.00 μ	7.00 μ	7.00 μ
10	8.00 μ	9.00 μ	8.00 μ	6.00 μ
Average	7.80 μ	0.0171 m	7.10 μ	3.70 μ

Tables 6.9 and 6.10 present the time taken for the controller to adapt following failure and the mean execution time of the AFDIA scheme during the adaptation period, for left and right wing respectively. It can be said that on average the controller adapts within 2.21 sec. The average execution time of the improved AFDIA scheme during this adaptation period is about 0.403 msec. This is very similar to the results of the original AFDIA scheme, following the failure. Once

again, these result confirm that the improved AFDIA scheme can execute at 50 Hz if necessary.

Table 6.9: Analysis of the improved AFDIA scheme execution and adaptation time following a loss of left wing surface.

Exp. No.	Mean Exe. Time (sec)	SD (sec)	Adaptation Time (sec)
1	0.415 m	0.208 m	3.475
2	0.412 m	0.136 m	1.857
3	0.419 m	0.177 m	1.843
4	0.443 m	0.185 m	1.989
5	0.355 m	0.190 m	1.822
6	0.425 m	0.197 m	2.194
7	0.427 m	0.167 m	2.404
8	0.490 m	0.395 m	2.147
9	0.431 m	0.143 m	1.850
10	0.433 m	0.199 m	2.017
Average	0.425 m	0.200 m	2.160
SD			0.499

6.10 Endurance Post Failure

In the original AFDIA scheme, evidence of endurance could be seen from the increase in flight duration post failure when compared against the X-Plane autopilot. The experiments conducted using the original scheme were terminated when the aircraft eventually crashed. With the improved AFDIA scheme, the aircraft managed to achieve the ideal goal of not crashing following a failure. With 66% of the wing surface missing, the aircraft maintained flight, which is a significant improvement in endurance when compared to the original AFDIA scheme.

In the interests of time, the 20 experiments conducted using the improved AFDIA scheme were limited to the duration of 5 minutes. During this duration the aircraft avoided going into an uncontrollable spin and crashing. In addition to the

Table 6.10: Analysis of the improved AFDIA scheme execution and adaptation time following a loss of right wing surface.

Exp. No.	Mean Exe. Time (sec)	SD (sec)	Adaptation Time (sec)
1	0.382 m	0.166 m	2.096
2	0.277 m	0.225 m	2.814
3	0.360 m	0.226 m	2.314
4	0.435 m	0.390 m	2.364
5	0.390 m	0.331 m	2.268
6	0.416 m	0.298 m	2.164
7	0.371 m	0.428 m	2.157
8	0.380 m	0.335 m	2.263
9	0.375 m	0.152 m	1.975
10	0.419E m	0.290 m	2.200
Average	0.381 m	0.284 m	2.262
SD			0.224

20 experiments conducted, 6 additional experiments were conducted using the improved AFDIA scheme.

In these experiments, 3 each are conducted for the loss of left and right wing surface. The aim of these experiments was to demonstrate the added endurance using the improved AFDIA scheme, following a 66% loss of wing surface. The results from these experiments are presented in Figures 6.16 through to 6.21.

In these additional experiments, the X-Plane simulation using the improved AFDIA scheme was left to run for a minimum of 2 hours following a failure. During this duration, the aircraft maintained flight with 66% of the wing surface missing. Hence, demonstrating the endurance added by the improved AFDIA over long duration, in the presence of severe loss of wing surface.

6.11 To Learn or Not to Learn

This section is independent of the results presented in this chapter. Instead, the aim here is to express the view of the author on how the schemes (SFDIA and AFDIA)

developed in this thesis could be considered for industrial implementation. These implementation issues are discussed from the perspective of the need to learn (or adapt on-line).

Before an aircraft system is in service, it has to be certified to add confidence in the system and meet strict safety requirements. This certification also applies to the control software on-board the aircraft systems. Certification is provided by authorities such as the Federal Aviation Administration (FAA) in the US or the Civil Aviation Authority (CAA) in the UK. The main document used as a guideline for the certification process is the DO-178 [101].

In accordance with this document, it is challenging to certify control software that is adaptive in nature. One of the major obstacles here is that adaptive control systems, such as the AFDIA scheme presented here, are considered non deterministic in nature and therefore it is harder to verify the stability of the control software. Due to this, the aerospace industry in general is reluctant to use such adaptive systems in practice. However, an argument can be made to use such systems, by considering when the system learns or adapts on-line.

In [37], the authors present an SFDIA scheme using NNs. Another NN based SFDIA is presented by the authors in [11]. In both of these schemes, the NN sensor estimators adapt on-line, even in the absence of failure. The main reasoning here is that the estimators improve their estimations by adapting to the dynamic environment of the aircraft. When a sensor failure occurs, the on-line learning is terminated to avoid degrading the sensor estimations. However, the question has to be raised on the necessity to learn in the absence of failure.

The SFDIA scheme presented in this thesis does not learn in the presence or absence of failure. Once the NN based estimators are developed, their structure remains fixed (i.e. no change in the weights or the number of neurons) throughout their lifetime. The reasoning for this is that if the NN based sensor estimators produce good estimates of the hardware sensors, there is no need to adapt the estimators on-line to the changing dynamics of the aircraft environment. The hardware sensors have already been certified and are implemented in practical systems, with-

out any need to adapt to the environment dynamics. Therefore, if the structure of the NN based sensor estimations are fixed, they can be classed as deterministic software and considered for certification using the existing practices.

Additionally in [11], the AFDIA scheme implements NN based pitch, roll and yaw controllers that adapt in the absence of failures. Once again, the argument here is to improve the controller estimation with the changing aircraft dynamics. Once a failure occurs, the NN controllers adapt using different guiding functions to accommodate the failures. Similar to the SFDIA scheme, an argument can be made that, if the NN controllers produce good estimates of the normal controls that have already been certified and which do not adapt on-line, there is no need to adapt (or learn) the NN based controllers in the absence of failures. Therefore, the NN based controllers with fixed structure can be certified using existing practices. The NN controllers will only adapt on-line when there is a failure (i.e. a need to learn the new dynamics of the aircraft).

In conclusion, although NNs have learning capabilities, one should consider when this is beneficial to the intended application.

6.12 Conclusion

In this chapter, an FCC NN based actuator failure detection, identification and accommodation (AFDIA) scheme was presented. The aim of this scheme was to add endurance to an aircraft following 66% loss of wing surface. This chapter presented the development of an FCC NN based roll controller, which was used by the AFDIA scheme. This FCC NN roll controller uses only 5 neurons to control the roll attitude of an aircraft.

The experiments were conducted using the Airbus A320 aircraft model in X-Plane. Following a 66% loss of wing surface, the aircraft under the control of X-Plane built-in autopilot would go into an uncontrollable spin and crash. This can be attributed to the fact that the autopilot is unaware of the change in dynamics of the aircraft following such a sever failure. An ideal scenario for the AFDIA scheme would be to maintain flight following a loss of the wing surface.

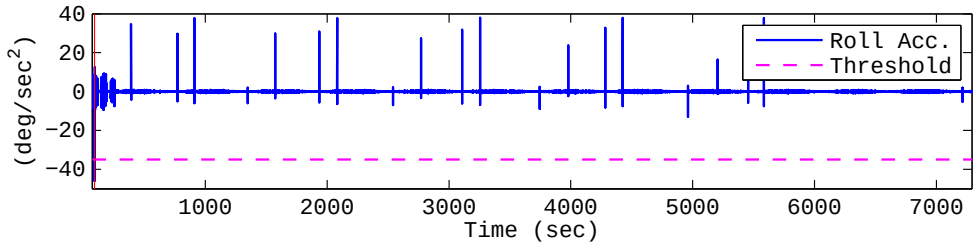
Initially, an AFDIA scheme was developed and studied on 240 experiments. Based on the observations from these experiments (Section 6.6), an improved version of the AFDIA scheme was developed. In the original AFDIA scheme (Section 6.4), the aircraft avoided going into an uncontrollable spin, like the results from the X-Plane built-in autopilot (Section 6.2). The aircraft did eventually crash, but the scheme increased the duration of the flight following a failure, when compared against the X-Plane autopilot results. Hence, adding endurance to the aircraft in presence of the failure. However, the ideal scenario of maintaining flight following failure was not achieved.

With the improved AFDIA scheme (Section 6.7.3), not only did the aircraft manage to avoid an uncontrollable spin, but also maintained flight following such a severe failure. This is remarkable, considering the fact that the aircraft maintains flight with 66% of the wing surface missing. Therefore, with the improved AFDIA scheme, the ideal scenario of maintaining flight following this failure is achieved. This is an exceptional addition of endurance to the aircraft system in the presence of such an extreme failure. Due to timing constraints on the research, the improved AFDIA scheme was evaluated on 20 experiments only, each for 5 minutes duration. During the duration of these experiments, the aircraft did not crash following the failure. However, to highlight the endurance added by the improved AFDIA scheme, 6 experiments were conducted, each of which spanned a duration of 2 hours. During these experiments, the aircraft maintained flight with 66% of the wing surface missing. This demonstrated the endurance added by the improved AFDIA scheme over long duration flight.

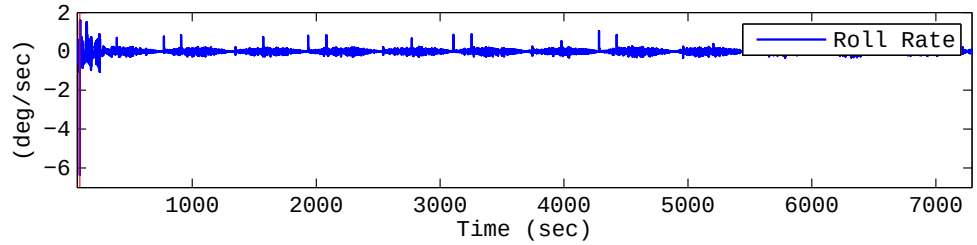
One of the drawbacks of the scheme was the use of the fixed threshold based mechanism for fault detection. The threshold must be predetermined, taking into account the probability of false detection and longer detection time. It must be noted that, in all the experiments conducted in this chapter, the failure was promptly detected and there was no false detection. However, this 100 % detection of failure can be attributed to the fact that the failure simulations were limited to the straight level flight manoeuvre phase of the aircraft. Further work needs to be conducted to

improve on the detection mechanism.

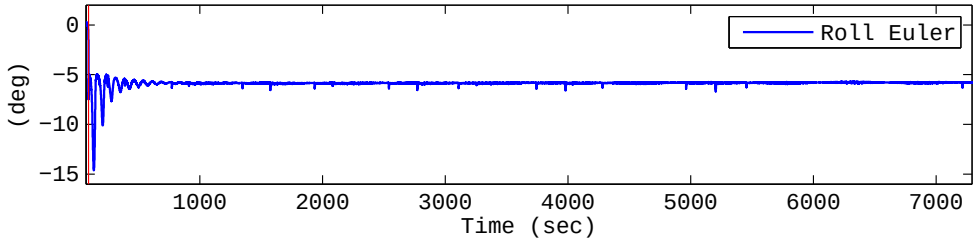
In conclusion, an AFDIA scheme is presented in this chapter that can add endurance to an aircraft with 66% of the wing surface missing. The AFIDA scheme manages to maintain flight in the presence of such a severe failure. The results presented in this chapter also validate the use of an FCC NN for AFDIA applications. The AFDIA scheme is able to add such remarkable endurance with just 5 neurons in the FCC NN based roll controller.



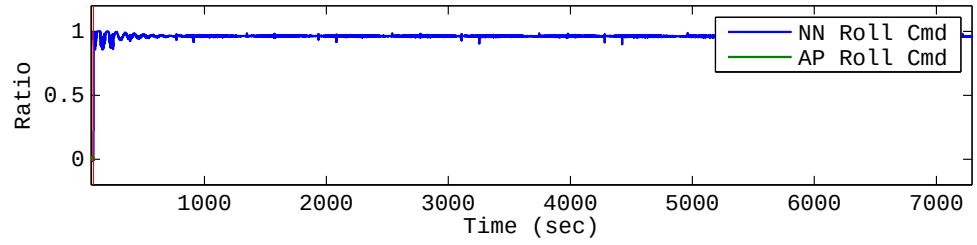
(a) Roll Acceleration



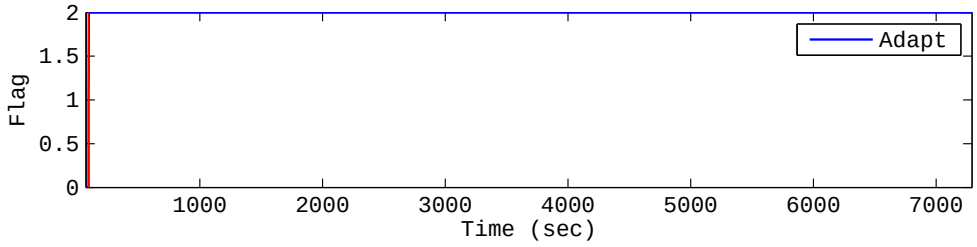
(b) Roll Rate



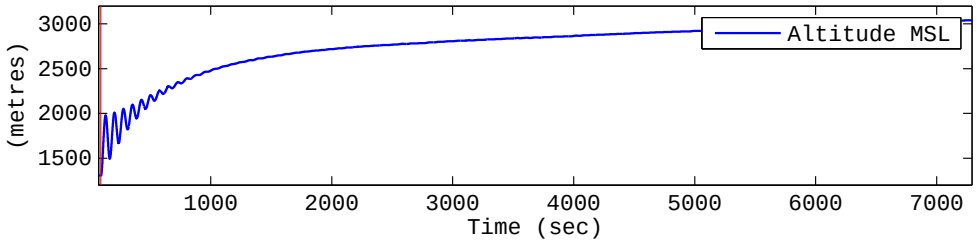
(c) Roll Euler Angle



(d) Roll Command

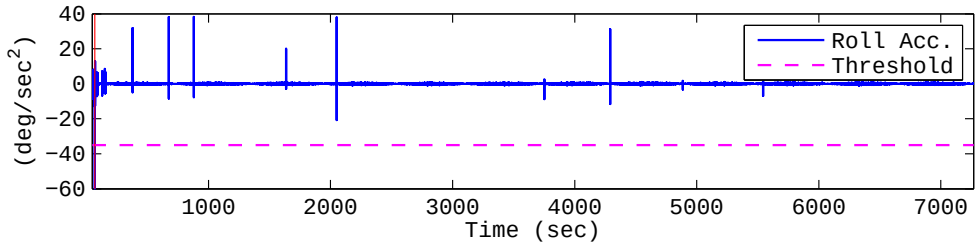


(e) Adapt Flag

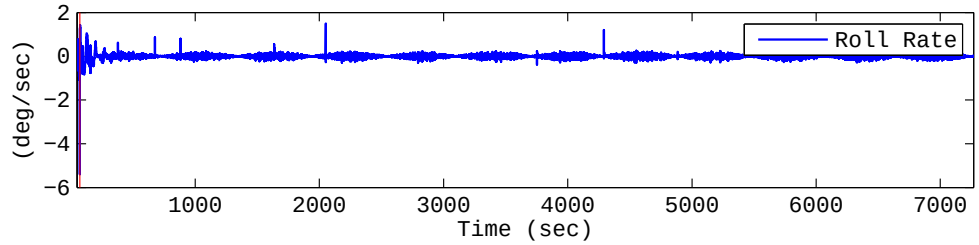


(f) Altitude

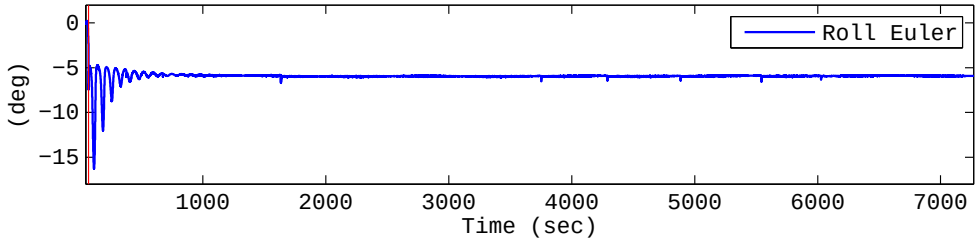
Figure 6.16: Aircraft flight data over 2 hours following left wing failure. The Red line marks when the failure was injected.



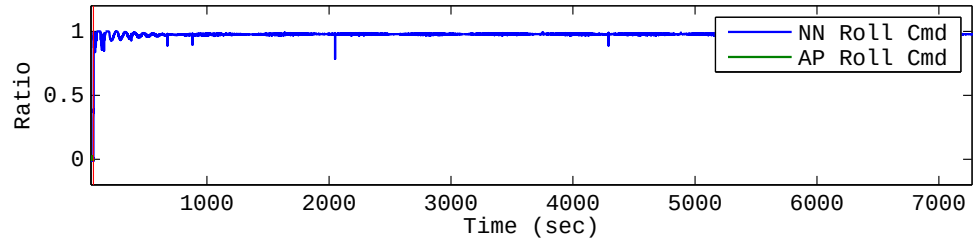
(a) Roll Acceleration



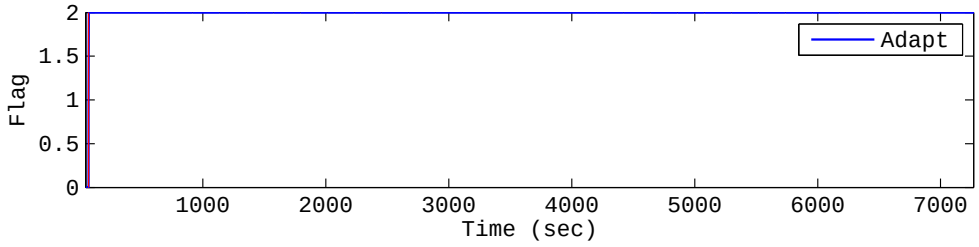
(b) Roll Rate



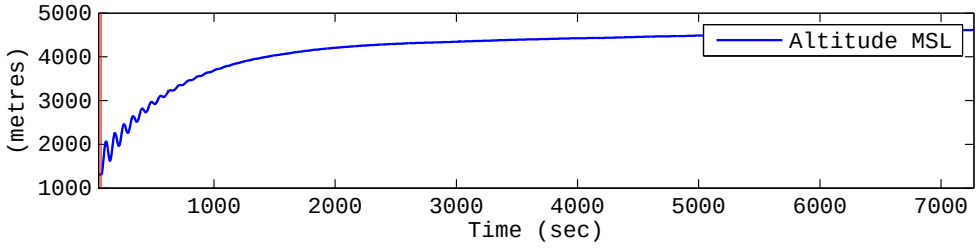
(c) Roll Euler Angle



(d) Roll Command

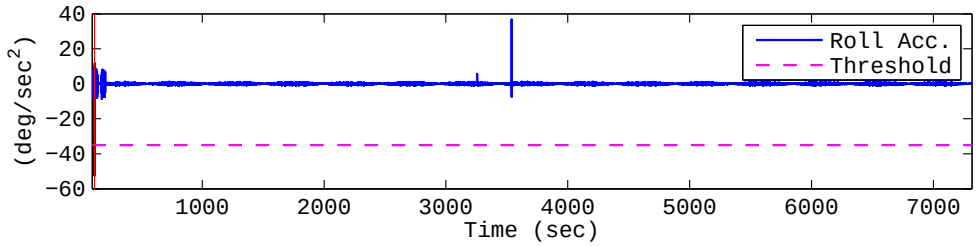


(e) Adapt Flag

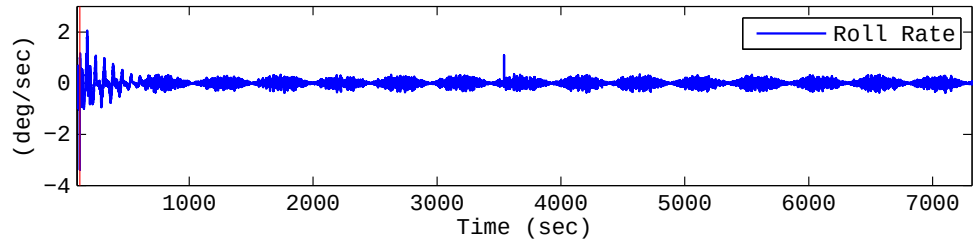


(f) Altitude

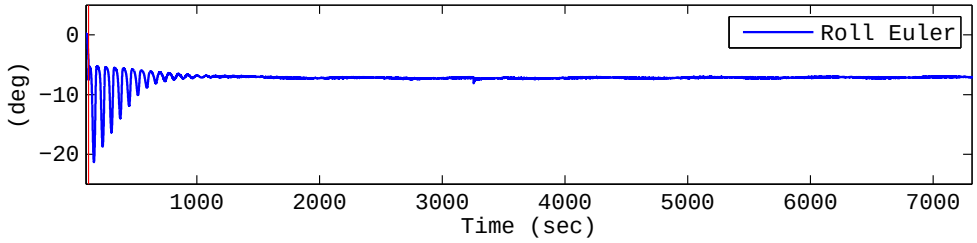
Figure 6.17: Aircraft flight data over 2 hours following left wing failure. The Red line marks when the failure was injected.



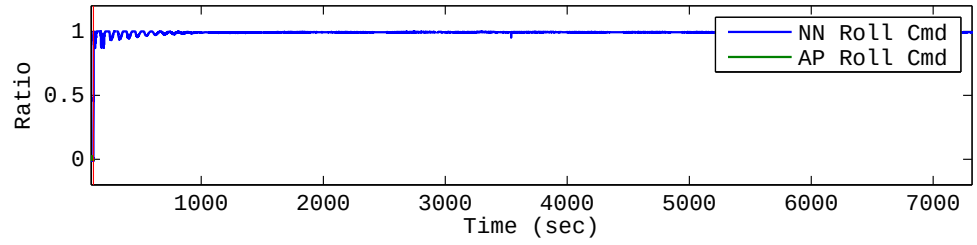
(a) Roll Acceleration



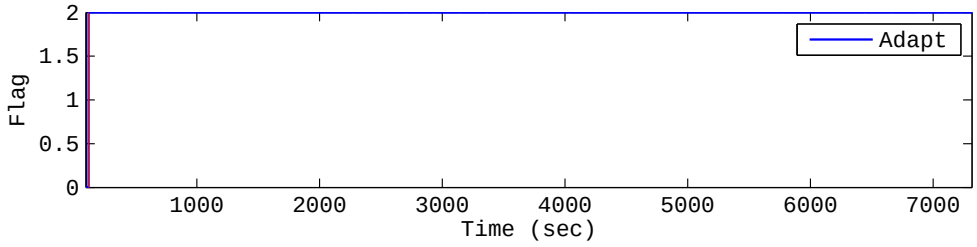
(b) Roll Rate



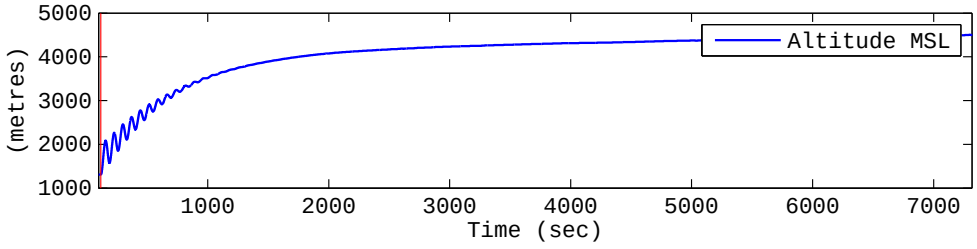
(c) Roll Euler Angle



(d) Roll Command

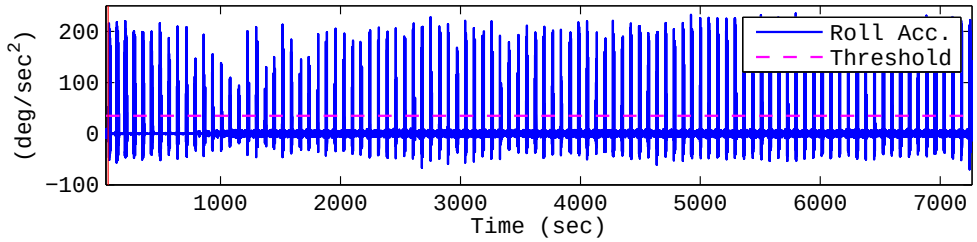


(e) Adapt Flag

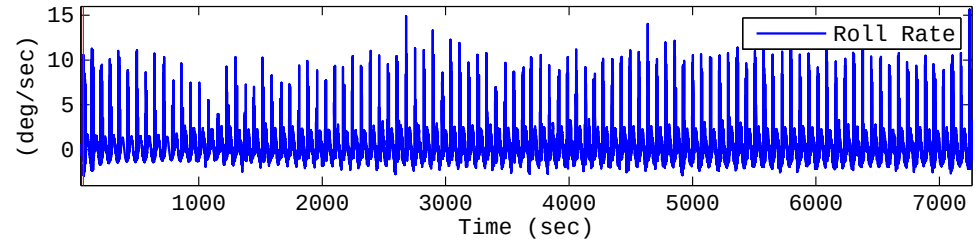


(f) Altitude

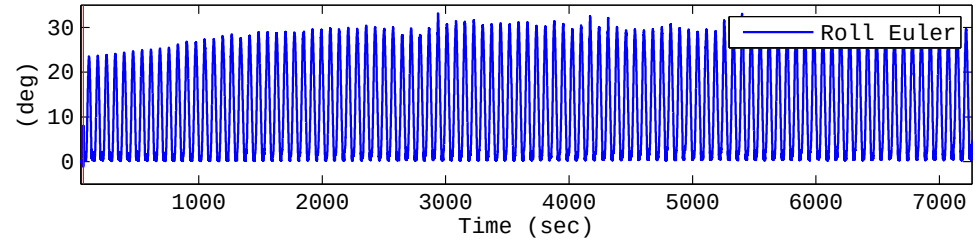
Figure 6.18: Aircraft flight data over 2 hours following left wing failure. The Red line marks when the failure was injected.



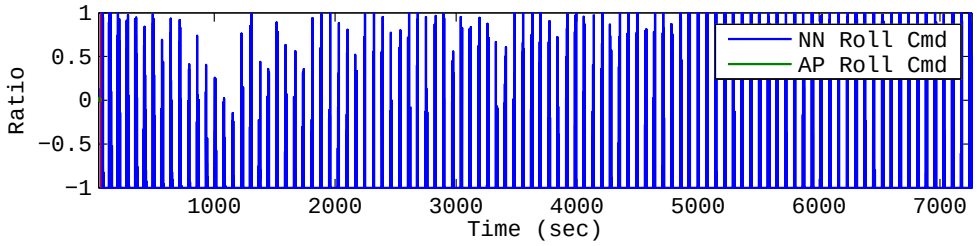
(a) Roll Acceleration



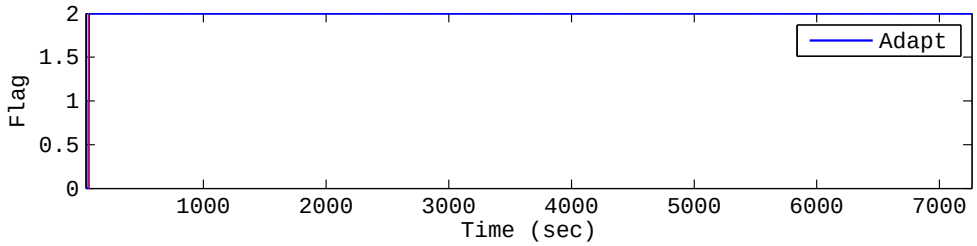
(b) Roll Rate



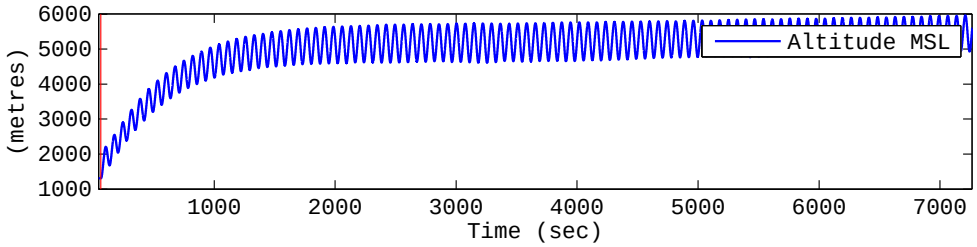
(c) Roll Euler Angle



(d) Roll Command

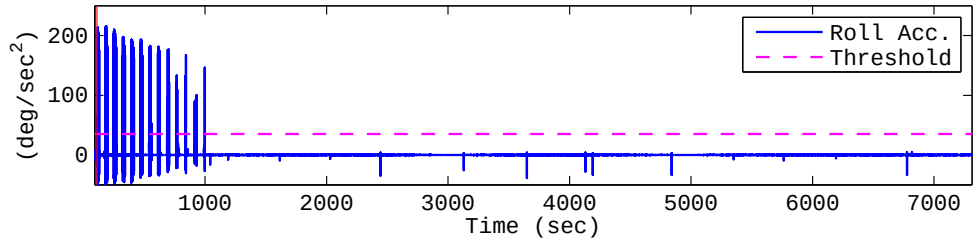


(e) Adapt Flag

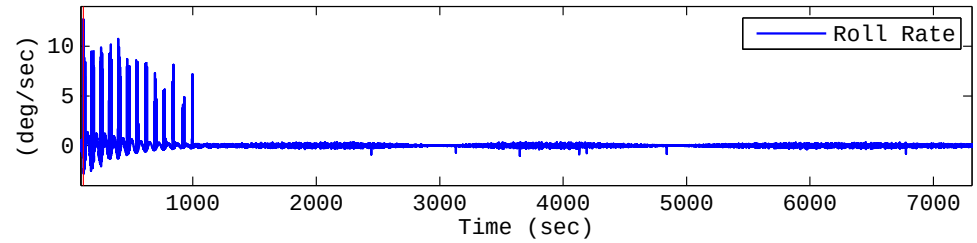


(f) Altitude

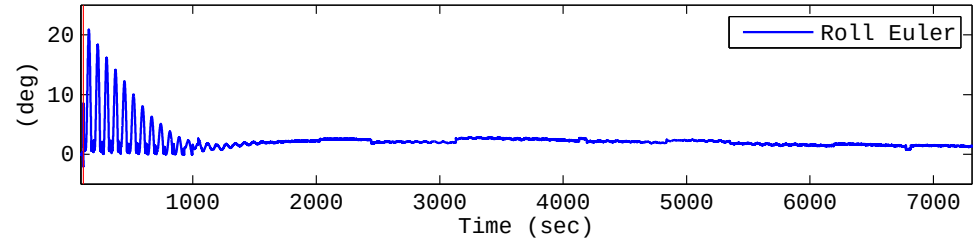
Figure 6.19: Aircraft flight data over 2 hours following right wing failure. The Red line marks when the failure was injected.



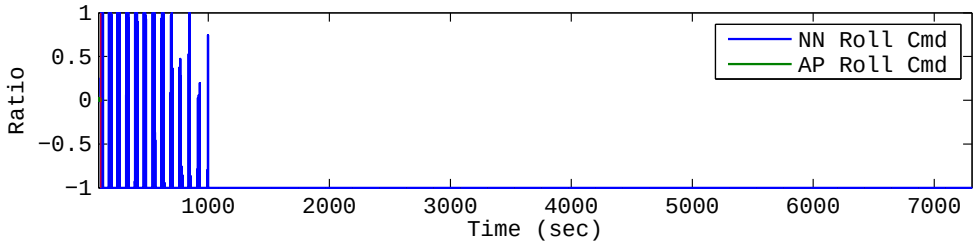
(a) Roll Acceleration



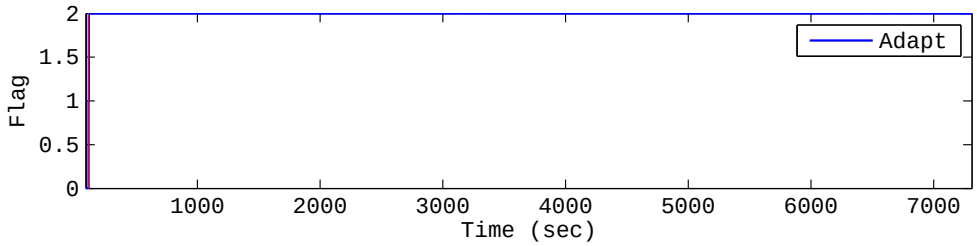
(b) Roll Rate



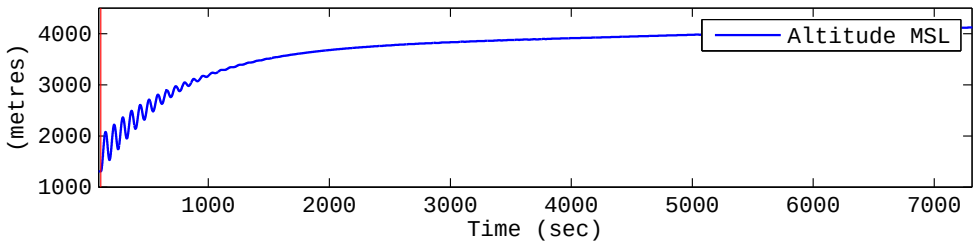
(c) Roll Euler Angle



(d) Roll Command

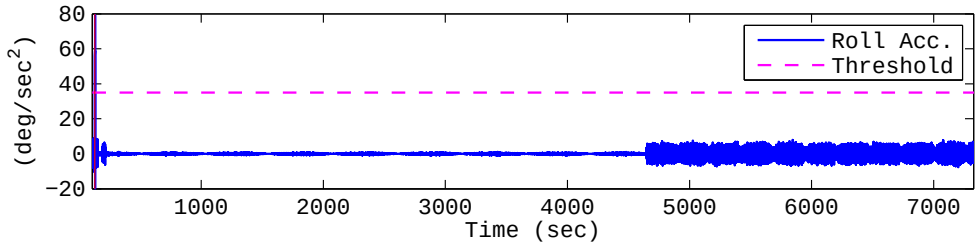


(e) Adapt Flag

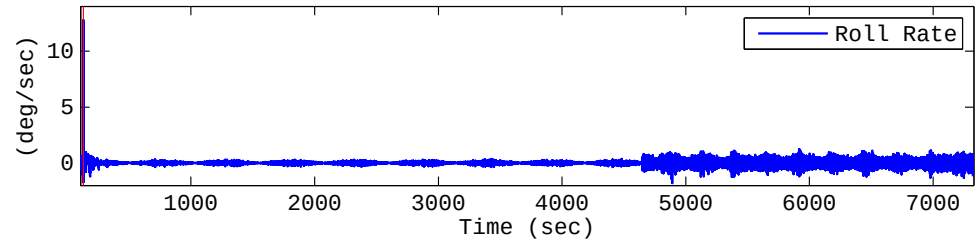


(f) Altitude

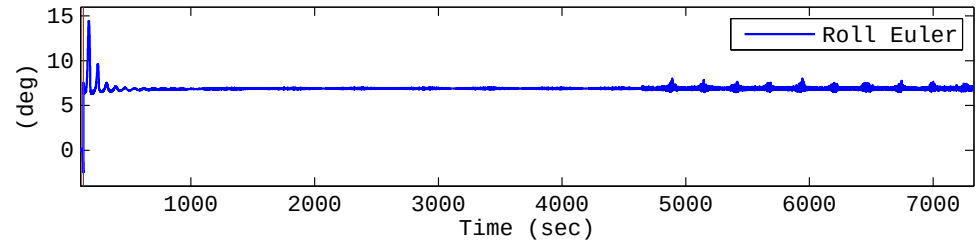
Figure 6.20: Aircraft flight data over 2 hours following right wing failure. The Red line marks when the failure was injected.



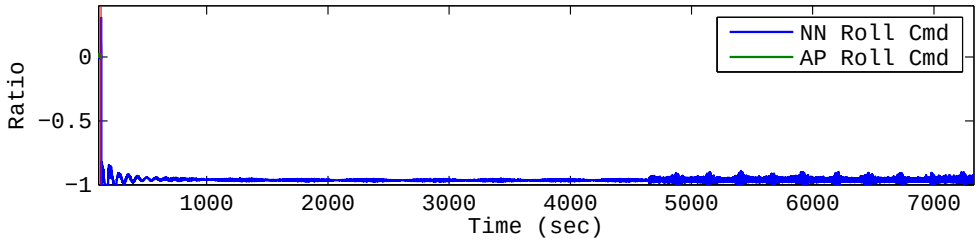
(a) Roll Acceleration



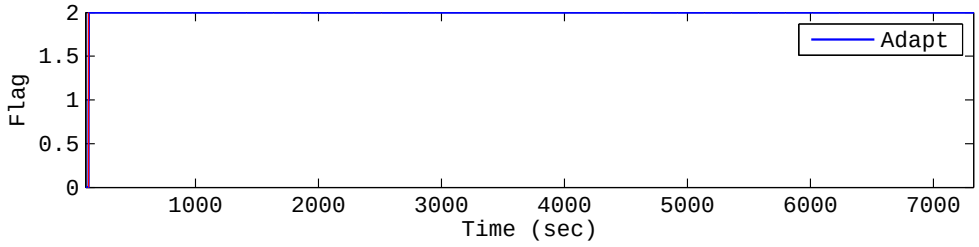
(b) Roll Rate



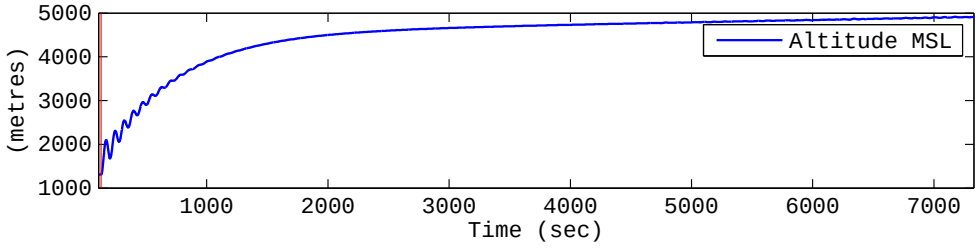
(c) Roll Euler Angle



(d) Roll Command



(e) Adapt Flag



(f) Altitude

Figure 6.21: Aircraft flight data over 2 hours following right wing failure. The Red line marks when the failure was injected.

"Success is going from failure to failure without losing your enthusiasm."
– Winston Churchill

Chapter 7

Conclusions and Future Work

7.1 Research Overview

The endurance of an aircraft in presence of failures can be improved if the flight control system is tolerant to failures. Two of the major types of failure in aircraft systems are sensor and actuator failures. In this research, schemes for fault tolerant flight control systems (FTFCS) were developed to add endurance to an aircraft in the presence of failures. To that end, two schemes were developed:

1. Sensor failure, detection, identification and accommodation scheme (SFDIA)

2. Actuator failure, detection, identification and accommodations scheme (AFDIA)

These schemes are based on the fully connected cascade (FCC) neural network (NN). The FCC NN architecture has been shown to be far more efficient and powerful compared to the popular multilayer perceptron (MLP) NN architecture [25, 60]. To the best knowledge of the author, this is first time the benefits of the FCC NN are harnessed for SFDIA and AFDIA in aircraft systems. The SFDIA and AFDIA scheme developed are discussed in the following section, where the contributions of this research are summarised.

7.2 Research Contribution

From the research conducted in this thesis, 3 main contributions can be identified. These are as follows:

1. **FCC NN for SFDIA and AFDIA**

   This research presented the development and application of SFDIA and AFDIA scheme based on the FCC NN. This is the first time the FCC NN architecture has been used for such application. Usually the popular choice for NN applications is the MLP NN architecture [25,60]. However, this architecture is not efficient and powerful when compared with the FCC NN. For example in the results presented by the authors in [60] and [25], to solve a parity-15 problem, the MLP NN with 1 hidden layer required 15 neurons. In comparison, the FCC NN was able to solve the problem using just 5 neurons. Additionally, to solve a party-63 problem the FCC NN required 6 neurons compared to 63 by the MLP architecture.

   In this research, FCC NN based pitch, roll and yaw rate sensor estimators were developed. These estimators are part of the SFDIA scheme. Following a failure in these sensors, the FCC NN based estimators replace the fault sensor measurements. With as few as 2 neurons, the FCC NN was able to estimate the yaw sensor measurements. The developed AFDIA scheme depends on an FCC NN based roll controller. This controller is able to mimic the output of the aircraft roll controller with 5 neurons. Following a severe loss of wing surface, the FCC controller adapts to maintain flight. The results achieved in this thesis validate the use of this NN architecture for SFDIA and AFDIA applications in aircraft systems.

2. **The SFDIA Scheme**

   One of the major types of failure in an aircraft system is sensor failure. The endurance of an aircraft can be increased in the presence of sensor failure if the aircraft system is capable of tolerating or accommodating it. This added tolerance can be achieved by implementing a sensor failure detection, iden-

tification and accommodation (SFDIA) scheme. In this research an SFDIA scheme based on an FCC NN was developed.

In this scheme, FCC NN based pitch, roll and yaw rate sensor estimators were developed. These estimators could replace the faulty physical sensor in case of a failure. One notable outcome of the SFDIA scheme development was the ability of the FCC NN to accurately estimate the yaw sensor measurements with just 2 neurons. In total, 105 failure experiments were conducted to analyse the developed SFDIA scheme. In these experiments, only 1 failure went undetected. These results validate the use of the FCC NN based SFDIA scheme, however, the scheme does have some shortcomings. These shortcomings were explored in the relevant sections and solutions were proposed as part of further work on improving the scheme.

One noteworthy limitation of the presented SFDIA scheme is the fixed threshold based failure detection mechanism. Selecting a fixed fault threshold is a challenging task especially in a dynamic system which is susceptible to noise and modelling inaccuracy. If the threshold is too high, the fault might take longer to be detected or worse, go undetected. Having a low threshold on the other hand might increase the rate of false detection. Additionally, if the dynamics of the system change in the future, the thresholds would have to be evaluated and fixed again. An alternative to the fixed threshold based detection mechanism is an adaptive threshold. In this mechanism, the fault threshold adapts to the changes in the system dynamics with time. Such a mechanism, as presented in the [50] and [96], would increase the robustness of the SFDIA scheme developed in this research.

3. **The AFDIA Scheme**

   Another major type of failure in an aircraft system is the failure of the actuators. In this research an FCC NN based actuator failure detection, identification and accommodation (AFDIA) scheme was developed. This scheme aimed at accommodating a severe case of failure, where about 66% of the wing surface is lost during flight. Successfully accommodating this would re-

sult in a remarkable addition of endurance to an aircraft system. An ideal scenario would be if the aircraft maintains flight with the loss of 66% of the wing surface. To achieve this scenario, the scheme implemented a an FCC NN based roll controller that could adapt on-line following a failure in order to accommodate the failure.

Initially, 240 AFDIA experiments were conducted based on which the final AFDIA scheme was proposed. The final AFDIA scheme was able to successfully accommodate such a severe failure, and achieved the ideal scenario of maintaining flight following a 66% loss of wing surface. 6 simulation results covering 2 hours of flight following the failure were presented to highlight the remarkable display of added endurance using the final design of the AFDIA scheme. The aircraft maintained flight over the 2 hours of simulation following a 66% loss of wing surface.

Regardless of this success, the scheme does have some limitations which were discussed in the relevant sections. One noteworthy limitation of the AFDIA scheme was the use of fixed threshold based fault detection mechanism. This is similar to the limitation of the SFDIA scheme discussed earlier, and a possible alternative to this is the use of the adaptive threshold based mechanism. In addition to this, in all the AFDIA experiments conducted the failure was successfully detected.

However, theses results could have been favoured by the limitation of the experiments to straight level flight. If the aircraft was making a turning manoeuvre and/or was in a turbulent environment, it is possible that the roll acceleration ($\dot{p}$) signal monitored for fault detection would cross the fixed threshold. This would result in false failure detection. Additional study needs to be conducted to develop detection mechanisms, that would take into account the flight phase of the aircraft, to ensure no false alarms are triggered. One possible solution is to monitor multiple signals to ensure the robustness of the detection mechanism.

7.3 Future Work

In this section, future work based on the research presented in this thesis is proposed. Note that future work specific to improving the presented SFDIA and AFDIA schemes has been discussed in the relevant sections of the thesis. The aim of this section is to provide the reader with some possible future work to expand on the research presented here.

1. **Multiple Sensor Failure**

   The SFDIA scheme developed addressed failures in pitch, roll and yaw rate sensors. The scheme however was limited to single sensor failure at a time. Future work could focus on extending the scheme to addresses multiple sensor failure at a time, similar to the research conducted by Samy [37].

2. **Complete AFDIA Scheme**

   The developed AFDIA scheme added endurance to an aircraft in the presence of 66% loss of wing surface. Due to timing constraints on the research, the scheme only implemented an FCC NN based roll controller. Future work could focus on developing an NN based pitch and yaw controller. These controllers could further enhance the results achieved using just the roll controller. For example, following a 66% loss of wing surface, the aircraft is able to maintain flight using the NN based roll controller which is part of the AFDIA scheme. Although the aircraft maintains flight following such a severe failure, it does not maintain a fixed altitude or heading. This is due to the lack of pitch or yaw control which is set to 0 following a failure.

   The AFDIA scheme could be expanded to implement an FCC NN based pitch and yaw controller, similar to the roll controller. This would further enhance the capability of the AFDIA scheme by enabling the ability to control the pitch and yaw attitude of the aircraft following a failure. In addition, implementing the pitch and yaw controller could allow the AFDIA scheme to accommodate loss of surface failures in the elevators and rudder of the aircraft.

3. **Integration of the AFDIA and SFDIA Scheme**

The SFDIA and the AFDIA scheme presented were developed independently of each other. However in practical applications one must consider how these two schemes will interact with each other. This problem was not addressed in this body of research. Future work needs to consider how these two schemes could be integrated to function together in a harmonious fashion.

4. **Using a Different Simulator**

   As mentioned earlier, the X-Plane simulator which is well known for its realistic simulations was used for this research. However, some challenges were encountered while using the simulator. These challenges and the intended research for different types of actuator failures were discussed in Chapter 5. Since the simulator only simulates loss of flying surface failures, additional control actuator failures could not be explored. This is a key limitation of the research presented here.

   Changing the simulator would allow the exploration of the further development of the AFDIA scheme, which accommodates a wide range of actuator failures. In addition, the FCC NN could be used to develop a robust detection mechanism that can detect a wide range of failures.

7.4 Summary

To conclude, the endurance of an aircraft can be increased in the presence of failures if the aircraft implements a fault tolerant flight control system (FTFCS). FTFCS can be achieved by implementing a failure detection, identification and accommodation (FDIA) schemes. In this research a sensor failure detection, identification and accommodation (SFDIA) and an actuator failure detection, identification and accommodation (AFDIA) schemes were developed. These schemes are based on the fully connected cascade (FCC) neural network (NN) architecture.

The SFDIA scheme can add endurance to an aircraft, following a pitch, roll or yaw rate sensor failure. The AFDIA scheme only addresses a severe failure of 66% loss of wing surface. The scheme manages to add endurance to an aircraft by

maintaining flight following a 66% loss of wing surface. The results presented in this research validate the use of the FCC NN for SFDIA and AFDIA applications, especially in aircraft systems.

"If you're absent during my struggle, don't expect to be present during my success."

– Will Smith

Bibliography

[1] H. Yu and B. Wilamowski, "Levenberg-Marquardt Training," in *Intelligent Systems*, ser. Electrical Engineering Handbook. CRC Press, Feb. 2011, pp. 1–16.

[2] R. Collinson, *Introduction to Avionics Systems*. Dordrecht: Springer Netherlands, 2011.

[3] U. BENNINGTON, "USS Bennington - crew stories - No Wing F15." [Online]. Available: http://www.uss-bennington.org/phz-nowing-f15.html

[4] CHRobotics, "Understanding Euler Angles," 2014. [Online]. Available: http://www.chrobotics.com/library/understanding-euler-angles

[5] UCLan, "Centre For Energy and Power Management." [Online]. Available: http://www.uclan.ac.uk/research/explore/groups/centre_for_energy_and_power_management.php

[6] S. Hussain, M. Mokhtar, and J. M. Howe, "Aircraft sensor estimation for fault tolerant flight control system using fully connected cascade neural network," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Aug. 2013, pp. 1–8.

[7] Y. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *Annual Reviews in Control*, vol. 32, no. 2, pp. 229–252, Dec. 2008.

[8] F. Bateman, H. Noura, and M. Ouladsine, "Fault Diagnosis and Fault-Tolerant Control Strategy for the Aerosonde UAV," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 3, pp. 2119–2137, Jul. 2011.

[9] M. Steinberg, "Historical Overview of Research in Reconfigurable Flight Control," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 219, no. 4, pp. 263–275, Jun. 2005.

[10] I. Samy, I. Postlethwaite, and D. Gu, "Neural network based sensor validation scheme demonstrated on an unmanned air vehicle (UAV) model," in *2008 47th IEEE Conference on Decision and Control*. IEEE, 2008, pp. 1237–1242.

[11] M. R. Napolitano, Y. An, and B. A. Seanor, "A fault tolerant flight control system for sensor and actuator failures using neural networks," *Aircraft Design*, vol. 3, no. 2, pp. 103–128, Jun. 2000.

[12] G. Campa, M. Fravolini, M. Napolitano, and B. Seanor, "Neural networks-based sensor validation for the flight control system of a B777 research model," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 1. American Automatic Control Council, 2002, pp. 412–417.

[13] Y. An, "A design of fault tolerant flight control systems for sensor and actuator failures using on-line learning neural network," PhD Thesis, West Virginia University, US, 1998.

[14] G. Heredia and A. Ollero, "Sensor fault detection in small autonomous helicopters using observer/Kalman filter identification," in *2009 IEEE International Conference on Mechatronics*, vol. 00, no. April. IEEE, 2009, pp. 1–6.

[15] H.-J. Rong, G.-B. Huang, N. Sundararajan, and P. Saratchandran, "Fuzzy Fault Tolerant Controller for Actuator Failures during Aircraft Autolanding," in *2006 IEEE International Conference on Fuzzy Systems*. IEEE, 2006, pp. 1200–1204.

[16] S. Kim, J. Choi, and Y. Kim, "Fault detection and diagnosis of aircraft actuators using fuzzy-tuning IMM filter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 3, pp. 940–952, Jul. 2008.

[17] I. Samy, I. Postlethwaite, and D.-W. Gu, "Detection and accommodation of sensor faults in UAVs- a comparison of NN and EKF based approaches," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, Dec. 2010, pp. 4365–4372.

[18] R. Isermann and P. Ballé, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control Engineering Practice*, vol. 5, no. 5, pp. 709–719, May 1997.

[19] M. R. Khosravani, "Application of Neural Network on Flight Control," *International Journal of Machine Learning and Computing*, vol. 2, no. 6, pp. 882–885, 2012. [Online]. Available: http://www.ijmlc.org/show-34-378-1.html

[20] G. Chowdhary and E. Johnson, "Adaptive Neural Network Flight Control Using both Current and Recorded Data," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, ser. Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, Aug. 2007.

[21] M. Innocenti and M. Napolitano, "Neural Networks and other Techniques for Fault Identification and Isolation of Aircraft Systems," PISA UNIV (ITALY) DEPT OF ELECTRICAL SYSTEMS AND AUTOMATION, Tech. Rep. May, 2003.

[22] Craig R. Bomben, J. W. Smolka, J. T. Bosworth, P. S. Williams-Hayes, J. J. Burken, R. R. Larson, Mark J. Buschbacher, and Heather A. Maliska, "Development and Flight Testing of a Neural Network Based Flight Control System on the NF-15B Aircraft," *NASA*, 2006.

[23] M.-Y. Chow, R. Sharpe, and J. Hung, "On the application and design of artificial neural networks for motor fault detection. I," *IEEE Transactions on Industrial Electronics*, vol. 40, no. 2, pp. 181–188, Apr. 1993.

[24] B. M. Wilamowski, "How to not get frustrated with neural networks," *2011 IEEE International Conference on Industrial Technology*, pp. 5–11, Mar. 2011.

[25] B. Wilamowski, "Neural network architectures and learning algorithms," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 56–63, Dec. 2009.

[26] H. Yu and W. Auburn, "Fast and efficient and training of neural networks," in *3rd International Conference on Human System Interaction*. IEEE, May 2010, pp. 175–181.

[27] B. M. Wilamowski, N. Cotton, J. Hewlett, and O. Kaynak, "Neural Network Trainer with Second Order Learning Algorithms," in *Intelligent Engineering Systems, 2007 International Conference on*. IEEE, Jun. 2007, pp. 127–132.

[28] B. Wilamowski, D. Hunter, and A. Mabnowski, "Solving parity-N problems with feedforward neural networks," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 4. IEEE, 2003, pp. 2546–2551.

[29] Laminar Research, "FAA-Certified X-Plane." [Online]. Available: http://www.x-plane.com/pro/certified/

[30] ——, "X-Plane 10 Manual," 2012. [Online]. Available: http://www.x-plane.com/files/manuals/X-Plane_10_Desktop_manual.pdf

[31] ——, "X-Plane Pro Website." [Online]. Available: http://www.x-plane.com/pro/landing/

[32] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[33] M. Verhaegen, S. Kanev, R. Hallouzi, C. Jones, J. Maciejowski, and H. Smail, "Fault tolerant flight control-a survey," in *Fault Tolerant Flight Control*. Springer, 2010, pp. 47–89.

[34] J. Jiang, "Fault-tolerant control systems-an introductory overview," *Acta Automatica Sinica*, vol. 31, no. 1, pp. 161–174, 2005.

[35] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A Survey of Fault Detection, Isolation, and Reconfiguration Methods," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 3, pp. 636–653, May 2010.

[36] P. Goupil, "AIRBUS state of the art and practices on FDI and FTC in flight control system," *Control Engineering Practice*, vol. 19, no. 6, pp. 524–539, Jun. 2011.

[37] I. Samy, I. Postlethwaite, and D.-W. Gu, "Survey and application of sensor fault detection and isolation schemes," *Control Engineering Practice*, vol. 19, no. 7, pp. 658–674, Jul. 2011.

[38] C. Lin and C. Liu, "Failure detection and adaptive compensation for fault tolerable flight control systems," *Industrial Informatics, IEEE Transactions on*, vol. 3, no. 4, pp. 322–331, Nov. 2007.

[39] J. Boskovic and R. Mehra, "A multiple model-based reconfigurable flight control system design," in *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, vol. 4, no. December. IEEE, 1998, pp. 4503–4508.

[40] P. Maybeck and R. Stevens, "Reconfigurable flight control via multiple model adaptive control methods," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 27, no. 3, pp. 470–480, May 1991.

[41] T. Menke and P. Maybeck, "Sensor/actuator failure detection in the Vista F-16 by multiple model adaptive estimation," *Aerospace and Electronic Systems, ...*, vol. 31, no. 4, pp. 1218–1229, Oct. 1995.

[42] P. Maybeck and D. Pogoda, "Multiple model adaptive controller for the STOL F-15 with sensor/actuator failures," in *Proceedings of the 28th IEEE Conference on Decision and Control*. IEEE, 1989, pp. 1566–1572. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=70412http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=70412

[43] Y. Zhang and J. Jiang, "Integrated active fault-tolerant control using IMM approach," *Aerospace and Electronic Systems, IEEE ...*, vol. 37, no. 4, pp. 1221–1235, 2001.

[44] ——, "An interacting multiple-model based fault detection, diagnosis and fault-tolerant control approach," in *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No.99CH36304)*, vol. 4, no. December. IEEE, 1999, pp. 3593–3598.

[45] NASA, "NASA Dryden Past Projects: Propulsion Controlled Aircraft (PCA)," 2009. [Online]. Available: http://www.nasa.gov/centers/dryden/history/pastprojects/PCA/\#.U-0cDvldVEI

[46] J. J. Burken and F. W. Burcham, "Flight-Test Results of Propulsion-Only Emergency Control System on MD-11 Airplane," *Journal of Guidance, Control, and Dynamics*, vol. 20, no. 5, pp. 980–987, Sep. 1997.

[47] F. Burcham, J. Burken, T. Maine, and J. Bull, "Emergency flight control using only engine thrust and lateral center-of-gravity offset: A first look," NASA, Tech. Rep. July, 1997. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.1997-3189

[48] G. Campa, M. L. Fravolini, B. Seanor, M. R. Napolitano, D. D. Gobbo, G. Yu, and S. Gururajan, "On-line learning neural networks for sensor validation for the flight control system of a B777 research scale model," *International Journal of Robust and Nonlinear Control*, vol. 12, no. 11, pp. 987–1007, Sep. 2002.

[49] S. Julier and J. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," *Int. symp. aerospace/defense ...*, 1997.

[50] M. Mrugalski, "An unscented Kalman filter in designing dynamic GMDH neural networks for robust fault detection," *International Journal of Applied Mathematics and Computer Science*, vol. 23, no. 1, pp. 157–169, Jan. 2013.

[51] C. Hajiyev and H. E. Soken, "Robust Estimation of UAV Dynamics in the Presence of Measurement Faults," *Journal of Aerospace Engineering*, vol. 25, no. 1, pp. 80–89, Jan. 2012.

[52] H. Guo-jian, L. Gui-xiong, C. Geng-xin, and C. Tie-qun, "Self-recovery method based on auto-associative neural network for intelligent sensors," in *2010 8th World Congress on Intelligent Control and Automation*, no. 2007. IEEE, Jul. 2010, pp. 6918–6922.

[53] S. Gururajan, M. L. Fravolini, H. Chao, M. Rhudy, and M. R. Napolitano, "Performance evaluation of neural network based approaches for airspeed Sensor Failure Accommodation on a small UAV," in *21st Mediterranean Conference on Control and Automation*. IEEE, Jun. 2013, pp. 603–608.

[54] T.-H. Guo and J. Musgrave, "Neural network based sensor validation for reusable rocket engines," in *Proceedings of 1995 American Control Conference - ACC'95*, vol. 2. American Autom Control Council, 1995, pp. 1367–1372.

[55] L. Cork, R. Walker, and S. Dunn, "Fault Detection, Identification and Accommodation Techniques for Unmanned Airborne Vehicles," *Australian International Aerospace Congress*, 2005.

[56] M. Fravolini, G. Campa, and K. Napolitano, "Minimal resource allocating networks for aircraft SFDIA," in *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. Proceedings (Cat. No.01TH8556)*, vol. 2, no. July. IEEE, 2001, pp. 1251–1256.

[57] N. Sundararajan, "Neural Networks for Intelligent Aircraft Fault Tolerant Controllers," *Signal Processing, Communications and ...*, pp. 8–14, 2008.

[58] L. Xiaoxiong, S. Liyuan, C. Kang, and G. Wei, "A neural network-based direct adaptive fault tolerance flight control for control surface damage," *Procedia Engineering*, vol. 15, pp. 147–151, Jan. 2011.

[59] NASA, "NASA Armstrong Fact Sheet: Intelligent Flight Control System," 2014. [Online]. Available: http://www.nasa.gov/centers/armstrong/news/FactSheets/FS-076-DFRC.html\#.VB1xb_ldVEI

[60] D. Hunter, H. Yu, and M. Pukish, "Selection of Proper Neural Network Sizes and Architectures—A Comparative Study," *Industrial Informatics, ...*, vol. 8, no. 2, pp. 228–240, May 2012.

[61] P. Goupil, "Oscillatory failure case detection in the A380 electrical flight control system by analytical redundancy," *Control Engineering Practice*, vol. 18, no. 9, pp. 1110–1119, Sep. 2010.

[62] I. Sadeghzadeh and Y. Zhang, "A review on fault-tolerant control for unmanned aerial vehicles (UAVs)," *Infotech@ Aerospace, St. Louis, MO*, no. March, pp. 1–12, 2011. [Online]. Available: http://arc.aiaa.org/doi/pdf/10.2514/6.2011-1472

[63] R. Patton, "Fault detection and diagnosis in aerospace systems using analytical redundancy," *Computing & Control Engineering Journal*, vol. 2, no. 3, pp. 127–136, 1991.

[64] F. A. C. Azevedo, L. R. B. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. L. Ferretti, R. E. P. Leite, W. Jacob Filho, R. Lent, and S. Herculano-Houzel, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain." *The Journal of comparative neurology*, vol. 513, no. 5, pp. 532–41, Apr. 2009.

[65] S. Herculano-Houzel, "The human brain in numbers: a linearly scaled-up primate brain." *Frontiers in human neuroscience*, vol. 3, no. November, p. 31, Jan. 2009.

[66] B. M. Wilamowski and H. Yu, "Improved computation for Levenberg-Marquardt training." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 21, no. 6, pp. 930–7, Jun. 2010.

[67] D. Kriesel, *A Brief Introduction to Neural Networks*, 2007. [Online]. Available: availableathttp://www.dkriesel.com

[68] K. Abhishek, M. Singh, S. Ghosh, and A. Anand, "Weather Forecasting Model using Artificial Neural Network," *Procedia Technology*, vol. 4, pp. 311–318, Jan. 2012.

[69] V. B. Sutariya, A. Groshev, and Y. V. Pathak, "Artificial Neural Networks in Pharmaceutical Research, Drug Delivery and Pharmacy Curriculum," in *2013 29th Southern Biomedical Engineering Conference*, vol. 4749. IEEE, May 2013, pp. 91–92.

[70] A. F. Atiya, "Bankruptcy prediction for credit risk using neural networks: a survey and new results." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 12, no. 4, pp. 929–35, Jan. 2001.

[71] B. K. Bose, "Neural Network Applications in Power Electronics and Motor Drives An Introduction and Perspective," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 1, pp. 14–33, Feb. 2007.

[72] B. M. Wilamowski, "C++ implementation of neural networks trainer," in *2009 International Conference on Intelligent Engineering Systems*. IEEE, Apr. 2009, pp. 257–262.

[73] B. Wilamowski, "Challenges in applications of computational intelligence in industrial electronics," *Industrial Electronics (ISIE), 2010 IEEE*, pp. 15–22, Jul. 2010.

[74] B. Wilamowski, N. Cotton, O. Kaynak, and G. Dundar, "Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 10, pp. 3784–3790, Oct. 2008.

[75] B. Wilamowski, H. Yu, and K. Chung, "Parity-N Problems as a Vehicle to Compare Efficiencies Parity-N Problems as a Vehicle to Compare Efficiencies," in *Intelligent Systems*, ser. Electrical Engineering Handbook. CRC Press, Feb. 2011, pp. 1–8.

[76] D. Liu, M. E. Hohil, and S. H. Smith, "N-bit parity neural networks: new solutions based on linear programming," *Neurocomputing*, vol. 48, no. 1–4, pp. 477–488, 2002.

[77] M. E. Hohil, D. Liu, and S. H. Smith, "Solving the N-bit parity problem using neural networks," *Neural Networks*, vol. 12, no. 9, pp. 1321–1323, 1999.

[78] P. Werbos, "Backpropagation: past and future," in *IEEE International Conference on Neural Networks*. IEEE, 1988, pp. 343–353 vol.1.

[79] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[80] B. M. Wilamowski and H. Yu, "Neural network learning without backpropagation." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 21, no. 11, pp. 1793–803, Nov. 2010.

[81] B. M. Wilamowski, "Advanced learning algorithms," in *2009 International Conference on Intelligent Engineering Systems*. IEEE, Apr. 2009, pp. 9–17.

[82] B. Wilamowski, H. Yu, and N. Cotton, "NBN Algorithm," in *Intelligent Systems*, ser. Electrical Engineering Handbook. CRC Press, Feb. 2011, pp. 1–24.

[83] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 5, no. 6, pp. 989–993, Jan. 1994.

[84] B. Wilamowski, "Understanding Neural Networks," in *Intelligent Systems*, ser. Electrical Engineering Handbook. CRC Press, Feb. 2011, pp. 1–11.

[85] Laminar Research, "X-Plane 10 Global — The World's Most Advanced Flight Simulator — X-Plane.com." [Online]. Available: http://www.x-plane.com

[86] R. Collinson, "Navigation Systems," in *Introduction to Avionics Systems*. Dordrecht: Springer Netherlands, 2011, pp. 303–376.

[87] R. P. G. Collinson, "Inertial Sensors and Attitude Derivation," in *Introduction to Avionics Systems*. Springer Netherlands, 2011, pp. 255–302.

[88] R. Collinson, "Aerodynamics and Aircraft Control," in *Introduction to Avionics Systems*. Dordrecht: Springer Netherlands, 2011, pp. 101–117.

[89] M. V. Cook, *Flight Dynamic Principles*, 2nd ed. Oxford: Butterworth-Heinemann, 2007.

[90] T. A. Talay, "Stability and Control." [Online]. Available: http://history.nasa.gov/SP-367/chapt9.htm

[91] A. Kozionov, M. Kalinkin, A. Natekin, and A. Loginov, "Wavelet-based sensor validation: Differentiating abrupt sensor faults from system dynamics," in *2011 IEEE 7th International Symposium on Intelligent Signal Processing*. IEEE, Sep. 2011, pp. 1–5.

[92] E. Balaban, A. Saxena, P. Bansal, K. F. Goebel, and S. Curran, "Modeling, Detection, and Disambiguation of Sensor Faults for Aerospace Applications," *IEEE Sensors Journal*, vol. 9, no. 12, pp. 1907–1917, Dec. 2009.

[93] I. Samy, "Development and Evaluation of Neural Network Models For Cost Reduction in Unmanned Air Vehicles," PhD Thesis, University of Leicester, UK, May 2009.

[94] G. Heredia, a. Ollero, M. Bejar, and R. Mahtani, "Sensor and actuator fault detection in small autonomous helicopters," *Mechatronics*, vol. 18, no. 2, pp. 90–99, Mar. 2008.

[95] M. Kim, S. H. Yoon, P. a. Domanski, and W. Vance Payne, "Design of a steady-state detector for fault detection and diagnosis of a residential air con-

ditioner," *International Journal of Refrigeration*, vol. 31, no. 5, pp. 790–799, Aug. 2008.

[96] M. Perhinschi, M. Napolitano, G. Campa, B. Seanor, J. Burken, and R. Larson, "An adaptive threshold approach for the design of an actuator failure detection and identification scheme," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 3, pp. 519–525, May 2006.

[97] Laminar Research, "X-Plane Website." [Online]. Available: http://www.x-plane.com/desktop/home/

[98] ——, "X-Plane Forum." [Online]. Available: http://forums.x-plane.org/

[99] X-Plane, "X-Plane Datarefs." [Online]. Available: http://www.xsquawkbox.net/xpsdk/docs/DataRefs.html

[100] Military.com, "F-15 One Wing Miracle Landing, Militay.com." [Online]. Available: http://www.military.com/video/military-aircraft-operations/crash-landings/f-15-one-wing-miracle-landing/660534011001/

[101] S. Jacklin, "Closing the Certification Gaps in Adaptive Flight Control Software," in *AIAA Guidance, Navigation and Control Conference and Exhibit*. Reston, Virigina: American Institute of Aeronautics and Astronautics, Aug. 2008, pp. 1–14.

Appendix A

Flight Duration Post Failure

Table A.1: Flight duration after failure for window = 5, scalar = 1.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	150.2445	177.3502	27.1057	153.4713	178.6182	25.1469
2	170.5796	198.1014	27.5219	184.5694	212.3194	27.7500
3	143.5789	169.8939	26.3150	194.0901	220.9019	26.8118
4	189.1915	216.2050	27.0135	228.3212	255.3141	26.9929
5	132.7407	159.7128	26.9721	176.4010	203.1687	26.7677
6	142.0849	169.1844	27.0995	155.0645	180.4173	25.3527
7	170.0213	196.9459	26.9246	176.2685	203.8789	27.6104
8	170.4354	197.7496	27.3141	169.6324	195.8537	26.2213
9	157.0884	182.8824	25.7940	155.7398	183.4530	27.7131
10	161.7281	186.9957	25.2677	176.5608	202.3955	25.8347
Average			26.7328			26.6202
SD			0.7157			0.9546

Table A.2: Flight duration after failure for window = 5, scalar = 2.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	145.7096	179.5242	33.8145	167.0497	203.7273	36.6776
2	148.8662	183.3794	34.5133	189.6095	226.5344	36.9249
3	187.2029	222.1616	34.9586	164.9783	200.8098	35.8316
4	194.2512	228.5379	34.2867	173.7484	208.9580	35.2096
5	189.5934	226.3197	36.7263	170.7298	206.9873	36.2575
6	171.8881	208.4742	36.5860	181.5931	217.6323	36.0391
7	238.7613	275.9999	37.2387	145.0559	181.4273	36.3714
8	173.7456	210.8386	37.0930	163.8484	199.7397	35.8914
9	149.1726	185.4382	36.2656	154.9047	191.0589	36.1542
10	141.9942	177.2180	35.2238	187.2708	225.2362	37.9653
Average			35.6707			36.3323
SD			1.2559			0.7429

Table A.3: Flight duration after failure for window = 5, scalar = 3.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	161.0635	203.2530	42.1895	80.7080	126.8291	46.1211
2	159.1147	203.6044	44.4897	90.7395	135.3193	44.5798
3	154.6768	199.0921	44.4153	90.6498	135.8056	45.1558
4	160.1172	205.4547	45.3376	90.7891	135.7188	44.9297
5	69.2600	110.8725	41.6124	66.7080	111.4631	44.7551
6	67.7430	109.1845	41.4415	91.8022	137.2820	45.4798
7	72.9330	113.9629	41.0299	78.8304	124.7404	45.9099
8	94.7963	134.8373	40.0409	75.7257	121.0817	45.3560
9	78.3706	118.5630	40.1924	74.3511	119.7682	45.4170
10	79.6932	120.9028	41.2095	76.8040	122.1879	45.3839
Average			42.1959			45.3088
SD			1.8844			0.4802

Table A.4: Flight duration after failure for window = 5, scalar = 4.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	68.2892	113.3526	45.0634	73.5902	123.9375	50.3472
2	85.9587	131.0649	45.1062	76.3741	125.2705	48.8964
3	90.4226	134.8861	44.4635	75.5907	126.3641	50.7735
4	80.7038	126.3148	45.6110	102.5846	155.4214	52.8368
5	80.9104	126.4647	45.5543	81.0407	132.2083	51.1675
6	76.2323	122.3747	46.1424	70.9449	121.3476	50.4027
7	85.5884	130.8004	45.2120	72.8508	122.0661	49.2153
8	80.6946	127.7934	47.0988	86.3819	136.0234	49.6415
9	76.0650	124.2801	48.2151	74.4946	124.5176	50.0230
10	73.8876	120.7645	46.8769	73.5384	122.5598	49.0214
Average			45.9344			50.2325
SD			1.1483			1.1866

Table A.5: Flight duration after failure for window = 10 scalar = 1

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	80.8262	106.0094	25.1832	86.8268	114.7391	27.9123
2	78.2050	102.8790	24.6739	87.3009	114.7200	27.4191
3	80.5769	106.2989	25.7220	71.6604	97.8969	26.2365
4	75.8973	101.1028	25.2055	75.4064	102.1480	26.7416
5	81.2491	106.1602	24.9111	81.3398	108.6644	27.3246
6	91.1470	116.0349	24.8879	77.7716	104.7764	27.0048
7	75.9793	101.3240	25.3447	75.5770	102.7294	27.1524
8	76.1527	100.6861	24.5334	86.8205	113.9030	27.0825
9	81.2922	107.2055	25.9133	77.1313	103.7514	26.6201
10	70.7834	96.1824	25.3990	75.7793	103.0799	27.3006
Average			25.1774			27.0794
SD			0.4396			0.4672

Table A.6: Flight duration after failure for window = 10 Scale = 2

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	70.7666	104.4930	33.7264	70.6163	107.2747	36.6584
2	73.0821	106.9959	33.9138	78.2068	114.8550	36.6482
3	75.5196	109.6357	34.1161	73.7480	109.8802	36.1322
4	71.4888	105.6731	34.1844	70.9855	106.9730	35.9875
5	78.5277	111.4495	32.9218	71.0631	106.4558	35.3927
6	88.9978	122.0260	33.0282	72.6033	109.0806	36.4773
7	77.0519	111.0535	34.0016	75.9452	112.5334	36.5882
8	73.6499	107.1377	33.4879	71.0266	107.3914	36.3647
9	88.5676	121.5234	32.9558	71.5365	107.8051	36.2686
10	70.5463	104.4549	33.9086	74.4938	110.6633	36.1695
Average			33.6245			36.2687
SD			0.4930			0.3838

Table A.7: Flight duration after failure for window = 10, scalar = 3.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	86.0173	126.5839	40.5666	67.6742	111.5790	43.9049
2	74.1014	116.1228	42.0215	70.9553	113.9949	43.0396
3	71.0777	112.4925	41.4149	78.4502	123.0322	44.5821
4	81.7859	122.2867	40.5008	70.4791	114.5335	44.0543
5	71.6949	112.6291	40.9343	72.6733	116.8146	44.1413
6	75.8087	117.0467	41.2381	70.5229	114.0637	43.5408
7	69.5850	110.8656	41.2806	69.3710	113.5539	44.1829
8	75.7718	116.4661	40.6944	70.2260	114.5232	44.2973
9	69.4104	111.5372	42.1268	77.0922	121.0451	43.9529
10	70.4685	112.1334	41.6649	73.4069	118.1799	44.7730
Average			41.2443			44.0469
SD			0.5774			0.4948

Table A.8: Flight duration after failure for window = 10, scalar = 4.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	74.6836	121.6101	46.9265	72.8235	122.3057	49.4822
2	75.2630	121.0634	45.8004	71.4826	121.0383	49.5556
3	79.4281	125.9985	46.5704	72.1018	121.4621	49.3603
4	73.6363	120.2051	46.5688	72.0176	122.4469	50.4292
5	73.8631	120.1802	46.3171	74.2060	124.4421	50.2361
6	73.5682	120.9842	47.4159	76.9707	126.6231	49.6524
7	73.1208	119.6456	46.5248	77.2314	127.2411	50.0097
8	76.4675	124.1347	47.6672	76.6922	126.1791	49.4869
9	73.9334	121.3218	47.3884	79.5569	130.3563	50.7994
10	76.6600	123.8124	47.1525	72.6414	123.4234	50.7820
Average			46.8332			49.9794
SD			0.5794			0.5523

Table A.9: Flight duration after failure for window = 15, scalar = 1.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	71.6822	97.0784	25.3961	76.9183	103.3374	26.4191
2	77.9546	102.7312	24.7766	83.4707	110.2156	26.7449
3	73.4495	98.6797	25.2301	76.1448	103.8072	27.6624
4	73.4059	98.7338	25.3278	78.6977	105.6913	26.9936
5	71.9201	97.3336	25.4135	79.5368	106.3074	26.7706
6	76.0673	101.3196	25.2523	82.1777	109.3321	27.1545
7	76.9751	101.5026	24.5275	79.1834	106.3010	27.1175
8	75.6794	101.0231	25.3437	75.1643	102.3878	27.2235
9	80.1249	105.3305	25.2056	73.7069	100.7741	27.0672
10	82.7283	108.0087	25.2804	73.5675	100.6987	27.1313
Average			25.1754			27.0285
SD			0.2899			0.3325

Table A.10: Flight duration after failure for window = 15, scalar = 2.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	74.3977	108.7223	34.3246	85.0803	121.5570	36.4767
2	75.2234	109.9334	34.7100	76.9829	112.2583	35.2754
3	85.7886	119.9593	34.1707	75.1888	111.1923	36.0035
4	84.4855	118.9568	34.4714	82.1491	118.7213	36.5723
5	77.8985	113.0402	35.1417	76.0607	110.8754	34.8147
6	86.7035	120.7780	34.0745	73.7009	110.4544	36.7534
7	71.9979	105.8106	33.8127	78.6740	115.1637	36.4898
8	77.4074	111.7339	34.3265	80.5827	116.7085	36.1259
9	77.9525	112.3584	34.4059	81.1147	117.8025	36.6878
10	85.0730	119.1783	34.1053	82.5470	119.0437	36.4967
Average			34.3543			36.1696
SD			0.3700			0.6446

Table A.11: Flight Duration after failure for window = 15, scalar = 3.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	72.4737	114.9053	42.4316	99.0782	142.7486	43.6703
2	73.9911	116.3881	42.3971	77.8251	121.7319	43.9068
3	82.5983	124.8254	42.2271	81.9747	126.7976	44.8229
4	78.9535	120.7761	41.8226	83.7316	129.0528	45.3212
5	78.9571	120.4503	41.4931	87.8847	132.5038	44.6191
6	78.6863	120.3292	41.6429	88.9720	133.9358	44.9637
7	77.5709	120.3126	42.7418	85.6161	130.2778	44.6616
8	85.4169	126.7389	41.3219	88.1161	133.0484	44.9323
9	80.8487	122.3957	41.5470	88.8281	133.0952	44.2671
10	84.1517	126.1272	41.9756	75.2922	120.9930	45.7009
Average			41.9601			44.6866
SD			0.4727			0.6155

Table A.12: Flight duration after failure for window = 15, scalar = 4.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Fail Detection	Time of Crash	Duration	Fail Detection	Time of Crash	Duration
1	81.0975	127.9159	46.8184	81.0166	130.9166	49.9000
2	81.5167	128.1190	46.6023	107.9082	157.2625	49.3543
3	78.3048	126.1952	47.8903	137.7240	186.8901	49.1661
4	85.5500	131.6093	46.0593	86.0942	136.5985	50.5043
5	85.1090	131.2361	46.1272	104.6387	154.2023	49.5636
6	78.9648	127.0427	48.0779	122.9080	175.1441	52.2362
7	79.0158	126.3583	47.3425	128.0069	178.4787	50.4718
8	81.3281	128.0856	46.7574	148.3754	198.0058	49.6303
9	80.2931	125.7166	45.4236	109.3023	158.9104	49.6081
10	89.1957	134.3188	45.1232	122.5085	172.9926	50.4841
Average			46.6222			50.0919
SD			0.9746			0.8963

Appendix B

Wing Loss Failure Detection Time

Table B.1: Wing loss failure detection for window = 5, scalar = 1.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	150.2093	150.2445	0.0352	153.4508	153.4713	0.0204
2	170.5392	170.5796	0.0404	184.5443	184.5694	0.0251
3	143.5146	143.5789	0.0644	194.0704	194.0901	0.0197
4	189.1687	189.1915	0.0229	228.3029	228.3212	0.0182
5	132.6779	132.7407	0.0628	176.3582	176.4010	0.0427
6	142.0612	142.0849	0.0238	155.0044	155.0645	0.0601
7	169.9746	170.0213	0.0467	176.2123	176.2685	0.0563
8	170.4048	170.4354	0.0307	169.5892	169.6324	0.0431
9	157.0428	157.0884	0.0455	155.6791	155.7398	0.0607
10	161.6944	161.7281	0.0337	176.4925	176.5608	0.0683
Average			0.0406			0.0415
SD			0.0145			0.0194

Table B.2: Wing loss failure detection for window = 5, scalar = 2.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	145.6527	145.7096	0.0569	166.9955	167.0497	0.0542
2	148.8450	148.8662	0.0211	189.5925	189.6095	0.0170
3	187.1584	187.2029	0.0446	164.9295	164.9783	0.0488
4	194.2176	194.2512	0.0336	173.7009	173.7484	0.0475
5	189.5728	189.5934	0.0206	170.6874	170.7298	0.0425
6	171.8127	171.8881	0.0754	181.5386	181.5931	0.0546
7	238.7266	238.7613	0.0346	145.0285	145.0559	0.0274
8	173.7110	173.7456	0.0346	163.8123	163.8484	0.0360
9	149.1387	149.1726	0.0339	154.8599	154.9047	0.0448
10	141.9343	141.9942	0.0599	187.2548	187.2708	0.0160
Average			0.0415			0.0389
SD			0.0176			0.0143

Table B.3: Wing loss failure detection for window = 5, scalar = 3.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	161.0309	161.0635	0.0326	80.6845	80.7080	0.0235
2	159.0683	159.1147	0.0463	90.7037	90.7395	0.0357
3	154.6511	154.6768	0.0258	90.6158	90.6498	0.0339
4	160.0643	160.1172	0.0529	90.7653	90.7891	0.0239
5	69.2255	69.2600	0.0345	66.6714	66.7080	0.0366
6	67.7085	67.7430	0.0345	91.7399	91.8022	0.0622
7	72.8978	72.9330	0.0352	78.7934	78.8304	0.0371
8	94.7390	94.7963	0.0573	75.6629	75.7257	0.0628
9	78.3245	78.3706	0.0461	74.3234	74.3511	0.0277
10	79.6691	79.6932	0.0242	76.7755	76.8040	0.0285
Average			0.0389			0.0372
SD			0.0112			0.0142

Table B.4: Wing loss failure detection for window = 5, scalar = 1.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	68.2297	68.2892	0.0595	73.5369	73.5902	0.0534
2	85.9019	85.9587	0.0568	76.3093	76.3741	0.0648
3	90.3830	90.4226	0.0395	75.5405	75.5907	0.0502
4	80.6632	80.7038	0.0406	102.5565	102.5846	0.0281
5	80.8449	80.9104	0.0655	81.0155	81.0407	0.0253
6	76.1817	76.2323	0.0506	70.9078	70.9449	0.0371
7	85.5400	85.5884	0.0484	72.8010	72.8508	0.0498
8	80.6698	80.6946	0.0248	86.3452	86.3819	0.0367
9	76.0401	76.0650	0.0249	74.4326	74.4946	0.0620
10	73.8208	73.8876	0.0668	73.4776	73.5384	0.0607
Average			0.0477			0.0468
SD			0.0152			0.0142

Table B.5: Wing loss failure detection for window = 10, scalar = 1.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	80.7792	80.8262	0.0470	86.7793	86.8268	0.0476
2	78.1572	78.2050	0.0478	87.2424	87.3009	0.0585
3	80.5175	80.5769	0.0594	71.6131	71.6604	0.0473
4	75.8739	75.8973	0.0233	75.3718	75.4064	0.0347
5	81.2256	81.2491	0.0235	81.3047	81.3398	0.0351
6	91.0892	91.1470	0.0578	77.7362	77.7716	0.0354
7	75.9325	75.9793	0.0468	75.5302	75.5770	0.0469
8	76.1179	76.1527	0.0348	86.7858	86.8205	0.0348
9	81.2339	81.2922	0.0582	77.0848	77.1313	0.0465
10	70.7253	70.7834	0.0581	75.7555	75.7793	0.0238
Average			0.0457			0.0410
SD			0.0140			0.0100

Table B.6: Wing loss failure detection for window = 10, scalar = 2.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	70.7193	70.7666	0.0473	70.5580	70.6163	0.0583
2	73.0351	73.0821	0.0470	78.1478	78.2068	0.0590
3	75.4610	75.5196	0.0586	73.7115	73.7480	0.0365
4	71.4530	71.4888	0.0357	70.9606	70.9855	0.0249
5	78.4922	78.5277	0.0355	71.0135	71.0631	0.0496
6	88.9384	88.9978	0.0594	72.5785	72.6033	0.0248
7	76.9918	77.0519	0.0601	75.8810	75.9452	0.0642
8	73.6007	73.6499	0.0492	70.9752	71.0266	0.0515
9	88.5065	88.5676	0.0611	71.4699	71.5365	0.0667
10	70.5100	70.5463	0.0363	74.4551	74.4938	0.0387
Average			0.0490			0.0474
SD			0.0105			0.0154

Table B.7: Wing loss failure detection for window = 5, scalar = 3.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	85.9712	86.0173	0.0461	67.6392	67.6742	0.0350
2	74.0665	74.1014	0.0348	70.9082	70.9553	0.0471
3	71.0192	71.0777	0.0584	78.3881	78.4502	0.0620
4	81.7382	81.7859	0.0476	70.4545	70.4791	0.0247
5	71.6353	71.6949	0.0596	72.6344	72.6733	0.0390
6	75.7505	75.8087	0.0581	70.4863	70.5229	0.0366
7	69.5616	69.5850	0.0234	69.3459	69.3710	0.0250
8	75.7232	75.7718	0.0486	70.1988	70.2260	0.0272
9	69.3601	69.4104	0.0503	77.0252	77.0922	0.0670
10	70.4438	70.4685	0.0247	73.3797	73.4069	0.0272
Average			0.0452			0.0391
SD			0.0133			0.0152

Table B.8: Wing loss failure detection for window = 5, scalar = 4.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	74.6240	74.6836	0.0596	72.7998	72.8235	0.0237
2	75.2279	75.2630	0.0351	71.4469	71.4826	0.0357
3	79.4044	79.4281	0.0236	72.0546	72.1018	0.0472
4	73.5883	73.6363	0.0480	71.9588	72.0176	0.0588
5	73.8048	73.8631	0.0584	74.1592	74.2060	0.0469
6	73.5208	73.5682	0.0475	76.9473	76.9707	0.0234
7	73.0975	73.1208	0.0233	77.1843	77.2314	0.0471
8	76.4080	76.4675	0.0595	76.6338	76.6922	0.0584
9	73.8869	73.9334	0.0465	79.5101	79.5569	0.0468
10	76.6249	76.6600	0.0351	72.6063	72.6414	0.0351
Average			0.0437			0.0423
SD			0.0138			0.0126

Table B.9: Wing loss failure detection for window = 15, scalar = 1.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	71.6469	71.6822	0.0353	76.8948	76.9183	0.0236
2	77.9197	77.9546	0.0349	83.4240	83.4707	0.0468
3	73.4262	73.4495	0.0233	76.1094	76.1448	0.0353
4	73.3592	73.4059	0.0468	78.6395	78.6977	0.0582
5	71.8964	71.9201	0.0237	79.5126	79.5368	0.0243
6	76.0207	76.0673	0.0466	82.1412	82.1777	0.0364
7	76.9514	76.9751	0.0237	79.1593	79.1834	0.0241
8	75.6557	75.6794	0.0237	75.1402	75.1643	0.0241
9	80.1010	80.1249	0.0239	73.6826	73.7069	0.0243
10	82.6930	82.7283	0.0353	73.5186	73.5675	0.0489
Average			0.0317			0.0346
SD			0.0095			0.0128

Table B.10: Wing loss failure detection for window = 15, scalar = 2.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	74.3368	74.3977	0.0609	85.0228	85.0803	0.0575
2	75.1422	75.2234	0.0811	76.9478	76.9829	0.0351
3	85.7658	85.7886	0.0228	75.1421	75.1888	0.0467
4	84.4270	84.4855	0.0585	82.0908	82.1491	0.0583
5	77.8633	77.8985	0.0352	76.0015	76.0607	0.0591
6	86.6682	86.7035	0.0353	73.6773	73.7009	0.0237
7	71.9630	71.9979	0.0349	78.6380	78.6740	0.0359
8	77.3724	77.4074	0.0349	80.5242	80.5827	0.0584
9	77.8936	77.9525	0.0589	81.0764	81.1147	0.0383
10	85.0382	85.0730	0.0348	82.5224	82.5470	0.0246
Average			0.0457			0.0438
SD			0.0180			0.0141

Table B.11: Wing loss failure detection for window = 15, scalar = 3.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	72.4499	72.4737	0.0238	99.0213	99.0782	0.0570
2	73.9562	73.9911	0.0348	77.8016	77.8251	0.0236
3	82.5639	82.5983	0.0345	81.9279	81.9747	0.0468
4	78.9039	78.9535	0.0496	83.7080	83.7316	0.0236
5	78.8983	78.9571	0.0588	87.8392	87.8847	0.0455
6	78.6266	78.6863	0.0597	88.9238	88.9720	0.0482
7	77.5355	77.5709	0.0353	85.5678	85.6161	0.0484
8	85.3709	85.4169	0.0460	88.0931	88.1161	0.0230
9	80.8130	80.8487	0.0357	88.7900	88.8281	0.0382
10	84.0983	84.1517	0.0534	75.2576	75.2922	0.0345
Average			0.0432			0.0389
SD			0.0120			0.0122

Table B.12: Wing loss failure detection for window = 15, scalar = 4.

	Left Wing Failure Time (sec)			Right Wing Failure Time (sec)		
No.	Injection	Detection	Elapsed	Injection	Detection	Elapsed
1	81.0614	81.0975	0.0360	80.9811	81.0166	0.0355
2	81.4565	81.5167	0.0602	107.8848	107.9082	0.0234
3	78.2466	78.3048	0.0582	137.6862	137.7240	0.0378
4	85.5152	85.5500	0.0348	86.0602	86.0942	0.0340
5	85.0523	85.1090	0.0566	104.5917	104.6387	0.0470
6	78.9414	78.9648	0.0235	122.8475	122.9080	0.0604
7	78.9922	79.0158	0.0236	127.9466	128.0069	0.0603
8	81.2929	81.3281	0.0352	148.3316	148.3754	0.0438
9	80.2570	80.2931	0.0361	109.2654	109.3023	0.0369
10	89.1300	89.1957	0.0657	122.4466	122.5085	0.0620
Average			0.0430			0.0441
SD			0.0157			0.0131

Appendix C

Controller Run Time

Table C.1: AFDIA run time for left wing failure, window = 5, scalar = 1.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.00 μ	0.365 m	5.50 μ	0.430 m
2	7.10 μ	0.370 m	8.40 μ	0.411 m
3	7.30 μ	0.520 m	4.70 μ	0.339 m
4	8.40 μ	0.345 m	7.10 μ	0.325 m
5	7.10 μ	0.943 m	5.00 μ	0.830 m
6	7.60 μ	0.375 m	6.10 μ	0.370 m
7	8.10 μ	0.243 m	1.19 μ	0.229 m
8	6.80 μ	0.368 m	5.00 μ	0.373 m
9	7.20 μ	0.370 m	8.00 μ	0.403 m
10	7.50 μ	0.873 m	7.00 μ	0.836 m
Average	7.41 μ	0.477 m	4.62 μ	0.455 m

Table C.2: AFDIA run time for right wing failure, window = 5, scalar = 1.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
2	7.50 μ	0.446 m	5.80 μ	0.341 m
3	7.40 μ	0.445 m	0.700 μ	0.240 m
4	7.50 μ	0.397 m	0.700 μ	0.393 m
5	7.00 μ	0.286 m	0.500 μ	0.233 m
6	6.90 μ	0.247 m	0.500 μ	0.146 m
7	7.70 μ	0.419 m	6.30 μ	0.482 m
8	8.10 μ	0.489 m	9.30 μ	0.49 m
9	7.80 μ	0.444 m	1.30 μ	0.199 m
10	7.40 μ	0.266 m	0.700 μ	0.175 m
Average	7.49 μ	0.387 m	3.12 μ	0.321 m

Table C.3: AFDIA run time for left wing failure, window = 5, scalar = 2.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.80 μ	0.338 m	0.900 μ	0.227 m
2	7.80 μ	0.816 m	0.800 μ	0.771 m
3	7.40 μ	0.456 m	1.20 μ	0.311 m
4	7.70 μ	1.21 m	6.60 μ	0.878 m
5	8.00 μ	0.601 m	0.900 μ	0.224 m
6	7.60 μ	0.492 m	5.00 μ	0.480 m
7	7.70 μ	0.526 m	4.70 μ	0.249 m
8	7.80 μ	0.517 m	5.10 μ	0.235 m
9	7.50 μ	0.517 m	5.20 μ	0.352 m
10	7.60 μ	0.513 m	0.700 μ	0.299 m
Average	7.69 μ	0.599 m	3.11 μ	0.403 m

Table C.4: AFDIA run time for right wing failure, window = 5, scalar = 2.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	6.90 μ	0.383 m	1.20 μ	0.185 m
2	6.80 μ	0.454 m	1.40 μ	0.537 m
3	6.80 μ	0.404 m	5.00 μ	0.232 m
4	6.90 μ	0.419 m	1.60 μ	0.288 m
5	6.70 μ	0.413 m	1.10 μ	0.267 m
6	7.30 μ	0.395 m	6.80 μ	0.181 m
7	6.90 μ	0.408 m	1.30 μ	0.344 m
8	7.10 μ	0.404 m	1.40 μ	0.282 m
9	7.00 μ	0.287 m	1.40 μ	0.214 m
10	6.70 μ	0.415 m	1.20 μ	0.221 m
Average	6.91 μ	0.398 m	2.24 μ	0.275 m

Table C.5: AFDIA run time for left wing failure, window = 5, scalar = 3.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	6.90 μ	0.404 m	1.60 μ	0.199 m
2	7.00 μ	0.413 m	1.50 μ	0.258 m
3	6.50 μ	0.428 m	1.10 μ	0.201 m
4	6.80 μ	0.425 m	4.60 μ	0.197 m
5	7.30 μ	0.394 m	2.20 μ	0.235 m
6	6.90 μ	0.355 m	1.40 μ	0.154 m
7	6.90 μ	0.347 m	1.60 μ	0.194 m
8	7.50 μ	0.388 m	7.70 μ	0.287 m
9	6.90 μ	0.311 m	1.40 μ	0.271 m
10	7.60 μ	0.491 m	5.80 μ	0.396 m
Average	7.03 μ	0.396 m	2.89 μ	0.239 m

Table C.6: AFDIA run time for right wing failure, window = 5, scalar = 3.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.50 μ	0.482 m	7.90 μ	0.293 m
2	7.10 μ	0.450 m	6.70 μ	0.268 m
3	6.90 μ	0.461 m	1.70 μ	0.264 m
4	7.10 μ	0.496 m	1.30 μ	0.221 m
5	7.30 μ	0.516 m	1.20 μ	0.266 m
6	7.70 μ	0.578 m	0.900 μ	0.373 m
7	7.90 μ	0.541 m	6.40 μ	0.201 m
8	7.60 μ	0.607 m	1.00 μ	0.279 m
9	7.60 μ	0.568 m	0.900 μ	0.262 m
10	7.70 μ	0.570 m	1.00 μ	0.366 m
Average	7.44 μ	0.527 m	2.90 μ	0.279 m

Table C.7: AFDIA run time for left wing failure, window = 5, scalar = 4.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	6.80 μ	0.383 m	1.50 μ	0.171 m
2	6.90 μ	0.252 m	1.60 μ	0.261 m
3	7.90 μ	0.382 m	0.900 μ	0.381 m
4	8.10 μ	0.514 m	0.250 μ	0.208 m
5	7.90 μ	0.519 m	0.900 μ	0.222 m
6	8.00 μ	0.514 m	1.20 μ	0.173 m
7	7.60 μ	0.471 m	1.10 μ	0.175 m
8	7.70 μ	0.519 m	1.20 μ	0.207 m
9	7.90 μ	0.488 m	1.00 μ	0.204 m
10	7.70 μ	0.591 m	1.10 μ	0.609 m
Average	7.65 μ	0.463 m	1.30 μ	0.261 m

Table C.8: AFDIA run time for right wing failure, window = 5, scalar = 4.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	8.00 μ	0.552 m	5.50 μ	0.213 m
2	7.70 μ	0.543 m	1.00 μ	0.467 m
3	8.00 μ	0.517 m	4.30 μ	0.204 m
4	7.90 μ	0.507 m	0.900 μ	0.201 m
5	8.00 μ	0.545 m	1.00 μ	0.251 m
6	7.70 μ	0.513 m	1.20 μ	0.238 m
7	7.50 μ	0.524 m	1.50 μ	0.199 m
8	8.00 μ	0.521 m	1.30 μ	0.195 m
9	7.50 μ	0.496 m	1.30 μ	0.360 m
10	7.30 μ	0.494 m	1.10 μ	0.264 m
Average	7.76 μ	0.521 m	1.91 μ	0.259 m

Table C.9: AFDIA run time for left wing failure, window = 10, scalar = 1.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.30 μ	0.600 m	1.60 μ	0.683 m
2	8.20 μ	0.903 m	0.0104 m	0.823 m
3	7.40 μ	0.389 m	6.80 μ	0.376 m
4	6.80 μ	0.232 m	1.40 μ	0.226 m
5	8.20 μ	0.190 m	0.0127 m	0.109 m
6	7.60 μ	0.220 m	7.60 μ	0.167 m
7	7.20 μ	0.309 m	6.60 μ	0.272 m
8	7.20 μ	0.760 m	4.50 μ	0.798 m
9	7.00 μ	0.393 m	1.90 μ	0.170 m
10	7.20 μ	0.326 m	1.50 μ	0.255 m
Average	7.41 μ	0.432 m	5.50 μ	0.388 m

Table C.10: AFDIA run time for right wing failure, window = 10, scalar = 1.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.80 μ	0.379 m	0.0112 m	0.151 m
2	7.50 μ	0.369 m	6.40 μ	0.376 m
3	6.90 μ	0.377 m	1.20 μ	0.146 m
4	7.20 μ	0.381 m	4.70 μ	0.157 m
5	8.00 μ	0.393 m	0.0130 m	0.220 m
6	8.00 μ	0.393 m	0.0130 m	0.220 m
7	7.30 μ	0.382 m	6.30 μ	0.207 m
8	7.00 μ	0.201 m	1.50 μ	0.189 m
9	7.30 μ	0.380 m	6.20 μ	0.148 m
10	7.10 μ	0.410 m	1.60 μ	0.231 m
Average	7.41 μ	0.367 m	6.51 μ	0.205 m

Table C.11: AFDIA run time for left wing failure, window = 10, scalar = 2.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.00 μ	0.393 m	4.60 μ	0.146 m
2	7.10 μ	0.585 m	1.70 μ	0.628 m
3	7.00 μ	0.380 m	1.50 μ	0.152 m
4	7.00 μ	0.456 m	1.70 μ	0.410 m
5	7.10 μ	0.543 m	5.10 μ	0.616 m
6	7.20 μ	0.573 m	0.90 μ	0.621 m
7	7.70 μ	0.600 m	6.40 μ	0.595 m
8	7.40 μ	0.498 m	5.10 μ	0.209 m
9	8.20 μ	0.467 m	9.80 μ	0.155 m
10	7.50 μ	0.476 m	1.10 μ	0.512 m
Average	7.32 μ	0.497 m	3.79 μ	0.404 m

Table C.12: AFDIA run time for right wing failure, window = 10, scalar = 2.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.00 μ	0.405 m	2.00 μ	0.181 m
2	6.90 μ	0.376 m	1.90 μ	0.133 m
3	7.50 μ	0.486 m	1.70 μ	0.193 m
4	7.80 μ	0.549 m	4.50 μ	0.284 m
5	7.70 μ	0.473 m	1.30 μ	0.129 m
6	8.10 μ	0.556 m	7.30 μ	0.185 m
7	7.40 μ	0.506 m	1.10 μ	0.136 m
8	7.40 μ	0.506 m	1.10 μ	0.136 m
9	8.00 μ	0.585 m	5.10 μ	0.285 m
10	8.10 μ	0.573 m	5.00 μ	0.216 m
Average	7.59 μ	0.501 m	3.10 μ	0.188 m

Table C.13: AFDIA run time for left wing failure, window = 10, scalar = 3.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	6.90 μ	0.390 m	1.40 μ	0.157 m
2	7.70 μ	0.402 m	0.0111 m	0.214 m
3	7.20 μ	0.389 m	4.50 μ	0.157 m
4	7.40 μ	0.385 m	6.40 μ	0.161 m
5	7.40 μ	0.412 m	1.80 μ	0.195 m
6	7.00 μ	0.463 m	5.80 μ	0.284 m
7	6.80 μ	0.275 m	1.30 μ	0.178 m
8	6.90 μ	0.412 m	1.50 μ	0.206 m
9	7.40 μ	0.453 m	1.30 μ	0.237 m
10	7.40 μ	0.313 m	1.30 μ	0.182 m
Average	7.21 μ	0.389 m	3.64 μ	0.197 m

Table C.14: AFDIA run time for right wing failure, window = 10, scalar = 3.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.50 μ	0.408 m	6.90 μ	0.141 m
2	6.90 μ	0.430 m	1.50 μ	0.279 m
3	7.60 μ	0.497 m	1.10 μ	0.178 m
4	7.50 μ	0.488 m	1.10 μ	0.179 m
5	8.10 μ	0.558 m	4.20 μ	0.310 m
6	7.80 μ	0.819 m	6.70 μ	0.853 m
7	7.60 μ	0.536 m	1.00 μ	0.166 m
8	8.00 μ	0.618 m	1.50 μ	0.490 m
9	7.90 μ	0.552 m	1.20 μ	0.150 m
10	7.70 μ	0.616 m	1.10 μ	0.272 m
Average	7.66 μ	0.552 m	2.63 μ	0.302 m

Table C.15: AFDIA run time for left wing failure, window = 10, scalar = 4.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	6.70 μ	0.390 m	1.40 μ	0.177 m
2	6.80 μ	0.403 m	1.30 μ	0.162 m
3	7.10 μ	0.393 m	1.50 μ	0.152 m
4	7.10 μ	0.407 m	1.50 μ	0.184 m
5	7.00 μ	0.414 m	1.60 μ	0.168 m
6	6.60 μ	0.406 m	1.40 μ	0.199 m
7	7.40 μ	0.392 m	9.20 μ	0.174 m
8	7.40 μ	0.404 m	6.30 μ	0.211 m
9	7.20 μ	0.289 m	6.50 μ	0.214 m
10	7.40 μ	0.417 m	6.10 μ	0.207 m
Average	7.07 μ	0.392 m	3.68 μ	0.185 m

Table C.16: AFDIA run time for right wing failure, window = 10, scalar = 4.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.00 μ	0.395 m	4.50 μ	0.192 m
2	6.80 μ	0.415 m	1.50 μ	0.185 m
3	7.10 μ	0.418 m	1.70 μ	0.176 m
4	6.80 μ	0.421 m	1.60 μ	0.260 m
5	6.80 μ	0.405 m	1.70 μ	0.171 m
6	7.60 μ	0.376 m	8.80 μ	0.126 m
7	7.00 μ	0.403 m	1.40 μ	0.194 m
8	7.10 μ	0.399 m	2.00 μ	0.142 m
9	7.00 μ	0.415 m	1.60 μ	0.162 m
10	6.90 μ	0.416 m	1.70 μ	0.194 m
Average	7.01 μ	0.406 m	2.65 μ	0.180 m

Table C.17: AFDIA run time for left wing failure, window = 15, scalar = 1.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.20 μ	0.383 m	6.80 μ	0.230 m
2	7.40 μ	0.275 m	0.0104 m	0.225 m
3	7.60 μ	0.180 m	7.10 μ	0.142 m
4	7.30 μ	0.449 m	1.70 μ	0.481 m
5	7.20 μ	0.441 m	5.10 μ	0.243 m
6	7.10 μ	0.265 m	1.50 μ	0.189 m
7	7.20 μ	0.558 m	6.80 μ	0.602 m
8	7.00 μ	0.584 m	1.50 μ	0.675 m
9	7.20 μ	1.01 m	6.50 μ	0.815 m
10	7.50 μ	0.249 m	9.30 μ	0.162 m
Average	7.27 μ	0.440 m	5.67 μ	0.376 m

Table C.18: AFDIA run time for right wing failure, window = 15, scalar = 1.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.60 μ	0.832 m	7.80 μ	0.823 m
2	7.10 μ	0.342 m	4.40 μ	0.367 m
3	7.30 μ	0.396 m	7.00 μ	0.168 m
4	7.00 μ	0.413 m	1.80 μ	0.177 m
5	7.70 μ	0.511 m	6.10 μ	0.165 m
6	7.90 μ	0.494 m	1.30 μ	0.149 m
7	7.20 μ	1.26 m	1.20 μ	0.948 m
8	8.00 μ	0.502 m	4.70 μ	0.164 m
9	7.8 μ	0.481 m	6.30 μ	0.161 m
10	7.80 μ	0.470 m	6.40 μ	0.161 m
Average	7.54 μ	0.570 m	4.70 μ	0.328 m

Table C.19: AFDIA run time for left wing failure, window = 15, scale = 2.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.10 μ	0.465 m	2.00 μ	0.554 m
2	7.50 μ	0.440 m	9.00 μ	0.244 m
3	7.50 μ	0.455 m	8.00 μ	0.240 m
4	7.20 μ	0.258 m	1.70 μ	0.197 m
5	6.90 μ	0.439 m	1.60 μ	0.340 m
6	7.60 μ	0.409 m	6.40 μ	0.430 m
7	7.00 μ	0.488 m	1.40 μ	0.419 m
8	7.30 μ	0.288 m	6.50 μ	0.223 m
9	7.10 μ	0.445 m	1.80 μ	0.372 m
10	7.20 μ	0.453 m	5.70 μ	0.327 m
Average	7.24 μ	0.414 m	4.41 μ	0.334 m

Table C.20: AFDIA run time for right wing failure, window = 15, scalar = 2.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.10 μ	0.398 m	1.50 μ	0.156 m
2	7.20 μ	0.457 m	4.50 μ	0.251 m
3	6.90 μ	0.440 m	1.70 μ	0.339 m
4	7.10 μ	0.439 m	6.30 μ	0.316 m
5	6.90 μ	0.442 m	1.40 μ	0.298 m
6	6.90 μ	0.324 m	1.40 μ	0.267 m
7	6.90 μ	0.431 m	1.50 μ	0.251 m
8	7.20 μ	0.448 m	1.60 μ	0.116 m
9	7.40 μ	0.503 m	1.00 μ	0.182 m
10	7.90 μ	0.571 m	1.30 μ	0.234 m
Average	7.15 μ	0.445 m	2.22 μ	0.241 m

Table C.21: AFDIA run time for left wing failure, window = 15, scale = 3.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	6.80 μ	0.767 m	1.30 μ	0.502 m
2	7.40 μ	0.432 m	6.20 μ	0.153 m
3	7.00 μ	0.285 m	3.80 μ	0.219 m
4	7.60 μ	0.411 m	8.10 μ	0.159 m
5	7.00 μ	0.406 m	1.40 μ	0.230 m
6	7.20 μ	0.384 m	1.70 μ	0.152 m
7	7.20 μ	0.423 m	6.40 μ	0.168 m
8	7.20 μ	0.414 m	1.30 μ	0.165 m
9	7.30 μ	0.456 m	1.20 μ	0.145 m
10	7.50 μ	0.525 m	1.00 μ	0.151 m
Average	0.722 m	0.450 m	3.24 μ	0.204 m

Table C.22: AFDIA run time for right wing failure, window = 15, scale = 3.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	6.90 μ	0.423 m	1.20 μ	0.153 m
2	7.10 μ	0.402 m	1.50 μ	0.143 m
3	7.00 μ	0.415 m	1.60 μ	0.156 m
4	7.30 μ	0.480 m	6.50 μ	0.217 m
5	7.10 μ	0.451 m	1.60 μ	0.174 m
6	7.60 μ	0.540 m	1.40 μ	0.179 m
7	8.10 μ	0.526 m	8.80 μ	0.253 m
8	7.20 μ	0.455 m	7.50 μ	0.158 m
9	8.00 μ	0.561 m	0.90 μ	0.320 m
10	7.00 μ	0.448 m	6.80 μ	0.294 m
Average	7.37 μ	0.473 m	3.44 μ	0.195 m

Table C.23: AFDIA run time for left wing failure, window = 15, scale = 4.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.60 μ	0.389 m	8.90 μ	0.147 m
2	7.60 μ	0.404 m	7.90 μ	0.130 m
3	7.20 μ	0.386 m	1.60 μ	0.127 m
4	7.40 μ	0.409 m	4.30 μ	0.186 m
5	7.20 μ	0.330 m	6.40 μ	0.291 m
6	7.30 μ	0.329 m	1.70 μ	0.261 m
7	6.80 μ	0.406 m	1.30 μ	0.159 m
8	7.40 μ	0.382 m	1.60 μ	0.134 m
9	8.50 μ	0.723 m	7.90 μ	0.727 m
10	7.90 μ	0.399 m	9.00 μ	0.142 m
Average	7.49 μ	0.416 m	5.06 μ	0.230 m

Table C.24: AFDIA run time for right wing failure, window = 15, scalar = 4.

	Run Time Mean (sec)		Run Time SD (σ) (sec)	
No.	Before Fail	After Fail	Before Fail	After Fail
1	7.80 μ	0.394 m	8.70 μ	0.157 m
2	7.20 μ	0.431 m	1.50 μ	0.138 m
3	7.00 μ	0.419 m	1.60 μ	0.199 m
4	7.70 μ	0.414 m	8.70 μ	0.122 m
5	7.50 μ	0.398 m	9.00 μ	0.167 m
6	7.00 μ	0.405 m	4.80 μ	0.149 m
7	6.70 μ	0.419 m	1.00 μ	0.131 m
8	7.10 μ	0.441 m	0.50 μ	0.165 m
9	7.10 μ	0.416 m	1.40 μ	0.167 m
10	7.00 μ	0.415 m	0.90 μ	0.158 m
Average	7.21 μ	0.415 m	3.81 μ	0.155 m

Appendix D

Publications by the Author

Aircraft Sensor Estimation for Fault Tolerant Flight Control System using Fully Connected Cascade Neural Network

Saed Hussain, *Student Member, IEEE,* Maizura Mokhtar, *Member, IEEE,* and Joe M. Howe

***Abstract*—Flight control systems that are tolerant to failures can increase the endurance of an aircraft in case of a failure. The two major types of failure are sensor and actuator failures. This paper focuses on the failure of the gyro sensors in an aircraft. The neuron by neuron (NBN) learning algorithm, which is an improved version of the Levenberg-Marquardt (LM) algorithm, is combined with the fully connected cascade (FCC) neural network architecture to estimate an aircraft's sensor measurements. Compared to other neural networks and learning algorithms, this combination can produce good sensor estimates with relatively few neurons. The estimators are developed and evaluated using flight data collected from the X-Plane flight simulator. The developed sensor estimators can replicate a sensor's measurements with as little as 2 neurons. The results reflect the combined power of the NBN algorithm and the FCC neural network architecture.**

I. Introduction

In recent years there has been a significant growth in the development of unmanned aerial vehicles (UAVs) for various applications (e.g. search and rescue, survey, border control). UAVs are most commonly used in applications that are considered dangerous, dull, impractical or unreachable by manned vehicles. These applications have contributed to an increasing importance for UAVs and the need to improve their endurance. Increasing the endurance of a UAV allows for:

- Longer flight hours without the need to refuel/recharge.
- Autonomously maintaining stability despite varying environmental conditions.
- Autonomously maintaining stability in case of failure.

Long endurance can be achieved by developing UAV systems that incorporate:

- Intelligent energy management systems.
- Intelligent flight behavior.
- Adaptive fault tolerance.

This research considers the development of a fault tolerant flight control system (FTFCS) to increase the endurance of a UAV in case of failure. In particular, this paper focuses on the development of neural network based sensor estimators to replace faulty sensors in case of sensor failure.

The paper investigates, for each sensor, the optimal architecture of the neural network based estimator. The type of neural network used is the fully connected cascade (FCC) neural network. The FCC neural network is chosen because it is able to achieve its objective with small number of neurons in the network [1]–[5]. Therefore, the paper presents the optimal number of neurons within the neural network to be used as the sensor estimator. These FCC neural network based sensor estimators will be used in future experiments to replace faulty sensors.

This paper is organized as follows: Section II provides background information on FTFCS, as well as the use of neural networks for FTFCS. Section III describes the FCC neural network and learning algorithm used to develop the sensor estimators. The aircraft simulator used for this research is briefly presented in Section IV. In Section V, the estimator development process is explained. Finally the results are discussed in Section VI and the conclusion is presented in Section VII.

II. Fault Tolerant Flight Control System

FTFCS are systems that have the ability to tolerate component failures automatically while maintaining overall system stability and acceptable performance in the event of errors and failures. Their purpose is to detect, identify and accommodate any type of failure that may occur during a flight. Two major classes of failure are sensor and actuator failures [6], [7]. In general a fully FTFCS needs to perform:

- Sensor failure detection, identification and accommodation (SFDIA) [8]
- Actuator failure detection, identification and accommodation (AFDIA) [8]

These tasks could be further divided into [8]:

- Failure detection and identification (FDI), which detects significant abnormalities and identifies the cause.
- Failure accommodation (FA), which in the case of sensors, replaces the faulty sensor with an appropriate estimation. In case of the actuators, it determines what actions need to be taken to recover the impaired aircraft.

This paper focuses on the failure accommodation stage (FA) of the SFDIA scheme. SFDIA schemes are particularly important when failed sensor measurements are used in the feedback loop of an aircraft's control laws. This could result in closed loop instability, possibly leading to unrecoverable flight conditions if the failure is not detected and accommodate for [6], [8].

S. Hussain, M. Mokhtar and J. M. Howe are with the School of Computing, Engineering and Physical Sciences, University of Central Lancashire (UCLan), Preston, PR1 2HE, United Kingdom. E-mail: saed@ieee.org,{MMokhtar, JMHowe}@uclan.ac.uk.

A. Redundancy for FDIA

Traditionally, fault detection, identification and accommodation (FDIA) is achieved through high levels of hardware redundancy. This is still the state-of-the-art practice in the aircraft manufacturing industry [6], [8]–[10]. For example, Airbus A320/330/340/380 has triple or quadruple redundant actuation, sensor and flight control computer systems [9].

In hardware redundancy for SFDIA, identical sensors are used to measure the same parameter; and fault tolerance is achieved based on a voting scheme [11]. For example, in a system with three redundant sensors, if one of the redundant signals differs significantly from the other two, the differing signal is eliminated.

However, hardware redundancy has serious cost, power and weight implications, especially for small aircraft's like UAVs. Due to these implications, analytical redundancy is a far more appealing approach for FTFCS. Analytical redundancy uses a model of the monitored system to generate signals that would otherwise be generated by redundant hardware. In its simplest form, the difference between the model estimate and the measured reading is used to generate an error residual. This residual is then monitored to detect and identify faults [12].

B. Neural Networks for Analytical Redundancy

Over the past two decades, there has been an increasing interest in the application of neural networks for SFDIA schemes [8], [13], [14]. For example, Guo and Musgrave [15] presented a SFDIA scheme for sensors in the space shuttle main engine (SSME). Their scheme is based on the auto-associative multi-layer perceptron (MLP) neural network, trained using the error back propagation learning algorithm (BPA). Napolitano et al. [8] developed a SFDIA scheme using the MLP neural network trained using the extended back propagation algorithm (EBPA). Samy et al. [6], [12] proposed a SFDIA scheme using the radial basis function (RBF) neural network, trained using the extended minimum resource allocating network (EMRAN) algorithm.

In this paper, a combination of the FCC neural network and neuron by neuron (NBN) learning algorithm is proposed for sensor accommodation. This can be part of any neural network based SFDIA scheme. Once a sensor failure is detected and isolated in the failure detection and identification (FDI) stage, the accommodation stage replaces the faulty sensor reading with a reliable estimate. SFDIA schemes based on neural network replaces the faulty sensor reading with a neural network generated estimate. It other words, the neural network works as an estimator.

III. Fully Connected Cascade (FCC) Neural Network and Neuron by Neuron (NBN) Learning Algorithm

A. Neural Network Architecture

It could be argued that the MLP neural network architecture is the most popular choice for neural network applications [1]–[4], [16]. However, this architecture is neither powerful nor efficient. MLP architectures requires more neurons, to solve a problem, than other architectures in which connection across layers is allowed [1]–[5]. Although increasing the number of neurons converges the neural network faster, the network loses its generalization ability [1], [2]. Therefore, the neural network responds poorly to patterns never used in the training.

As a comparison between the two architectures, the authors of [5] state that: to solve the parity-7 problem, the MLP architecture using one hidden layer required 8 neurons; whereas the fully connected cascade (FCC) architecture (Fig. 1), which allows connection across layers, managed to solve this problem using just 3 neurons. To solve the parity-64 problem, 64 neurons were required by the MLP architecture in comparison to 6 neurons by the FCC architecture.

This shows that the FCC architecture is better and more efficient in comparison to the MLP architecture. Therefore, the FCC neural network is chosen for use as the sensor estimator neural network.

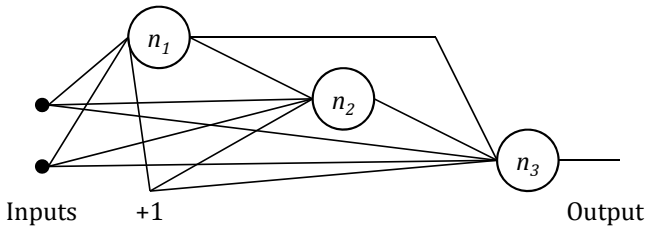


Fig. 1. FCC Neural Network Architecture

B. Learning Algorithm

The error back propagation (EBP) algorithm is popularly used along with the MLP neural network architecture. However this algorithm is slow and inefficient [1]–[3], [5], [17]. Many improvements have been made to help speed up the EBP algorithm (e.g. momentum, EBPA), but as long as first order algorithms are used, improvements are not dramatic [17]. Instead of the EBP algorithm, advance second order algorithms like the Levenberg-Marquardt (LM) or the neurons by neuron (NBN) algorithms should be used. These algorithms can not only train fast, but also efficiently, with small number of neurons within the neural network [1]–[3], [5], [16], [18]–[20].

The NBN algorithm is an improved version of the second order LM algorithm [2], [5]. In the LM algorithm, the weights are updated using the following update rule [20]:

$$W_{n+1} = W_n - (J^T J + \mu I)^{-1} J^T \boldsymbol{e} \tag{1}$$

where W_{n+1} is the new weights vector; W_n is the previous weights vector; J is the Jacobian matrix; I is the identity matrix; $\boldsymbol{e}$ is the error vector; and μ is the combination coefficient. The size of the Jacobian matrix and the error vector are $(P \times M) \times N$ and $(P \times M) \times 1$ respectively, where P is the number of training patterns, M is the number of network outputs and N is the number of weights [21].

$$J = \begin{bmatrix} \overbrace{}^{neuron\ 1} & & & \overbrace{}^{neuron\ j} & & \\ \frac{\partial e_{1,1}}{\partial w_{1,1}} & \frac{\partial e_{1,1}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{1,1}}{\partial w_{j,1}} & \cdots & m=1 \\ \frac{\partial e_{1,2}}{\partial w_{1,1}} & \frac{\partial e_{1,2}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{1,2}}{\partial w_{j,1}} & \cdots & m=2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{1,M}}{\delta w_{1,1}} & \frac{\partial e_{1,M}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{1,M}}{\partial w_{j,1}} & \cdots & m=M \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,1}}{\partial w_{1,1}} & \frac{\partial e_{P,1}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{P,1}}{\partial w_{j,1}} & \cdots & m=1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,M}}{\partial w_{1,1}} & \frac{\partial e_{P,M}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{P,M}}{\partial w_{j,1}} & \cdots & m=M \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ p=1 \\ \\ \\ \\ p=P \\ \\ \end{matrix}$$

Fig. 2. The Jacobian Matrix J

The Jacobian matrix is presented in Fig. 2, where, p is the training pattern; j is the index of the neuron; $w_{j,x}$ is the x^{th} connection weight w to neuron j; and m is the index of the network output neuron. The error $e_{p,m}$ for training pattern p at network output neuron m is calculated as follows:

$$e_{p,m} = d_{p,m} - o_{p,m} \tag{2}$$

where $d_{p,m}$ is the desired output and $o_{p,m}$ is the actual output for training pattern p at network output neuron m.

Usually, the Jacobian matrix is calculated and stored for updating the weights using (1). This is fine with problems that require small number of training patterns. However, for problems with a large number of training patterns, memory limitation may become a major concern. This is due to the size for the Jacobian matrix [21], [22].

In the NBN algorithm, the weights are updated using the following update rule:

$$W_{n+1} = W_n - (Q + \mu I)^{-1}\boldsymbol{g} \tag{3}$$

where Q is the quasi-Hessian matrix and $\boldsymbol{g}$ is the gradient vector. This is just another form of the LM update rule [23] where

$$Q = J^T J \tag{4}$$

$$\boldsymbol{g} = J^T \boldsymbol{e} \tag{5}$$

However, in the NBN algorithm, the matrix Q is calculated by summing the quasi-Hessian sub-matrix $q_{p,m}$ for pattern p and network output neuron m:

$$Q = \sum_{p=1}^{P} \sum_{m=1}^{M} q_{p,m} \tag{6}$$

The gradient vector $\boldsymbol{g}$ is calculated by summing the gradient sub-vector $\boldsymbol{\eta_{p,m}}$ for pattern p and network output neuron m:

$$\boldsymbol{g} = \sum_{p=1}^{P} \sum_{m=1}^{M} \boldsymbol{\eta_{p,m}} \tag{7}$$

The size of the matrix Q is $N \times N$ and is independent of the number of patterns and outputs. Compared to the LM algorithm, the NBN algorithm calculates the matrix Q and vector $\boldsymbol{g}$ directly as the patterns are applied. Therefore removing the need to compute and store the Jacobian matrix (J) [23]. This is achieved by calculating the vector $\boldsymbol{j_{p,m}}$ as the patterns are applied. This vector is the Jacobian row for pattern p and network output neuron m. Using this vector, the matrix Q and vector $\boldsymbol{g}$ can be updated as each pattern is applied using the following equations:

$$q_{p,m} = j_{p,m}^T j_{p,m} \tag{8}$$

$$\boldsymbol{\eta_{p,m}} = j_{p,m} e_{p,m} \tag{9}$$

The main advantages of the NBN algorithm over the LM algorithm can be summarized as follows [1], [23]:

1) It can train arbitrarily connected, feed forward neural network (i.e. it can be used with the FCC network unlike the LM algorithm).
2) Error derivatives are calculated in the forward propagation therefore no need for back propagation. This makes it more efficient compared to LM algorithm, especially for networks with multiple outputs [17].
3) It does not need to compute and store large Jacobian matrix, therefore it can be used with unlimited patterns [21].

Due to these benefits, the NBN algorithm is selected to train the FCC neural network based sensor estimators. The pseudo code for this algorithm, adapted from [23], is given in Fig. 3. In the next subsection, the settings of various parameters used to create the neural networks for the estimators is presented.

C. Neural Network Settings

The neural networks are initialized with random weights in the range of $+1.5$ to -1.5. The activation function used by the neurons is the bipolar sigmoid [24], defined as follows:

$$Out_j = \frac{2}{1 + e^{-net_j}} - 1 \tag{10}$$

where net_j is the sum of the weighted inputs to neuron j and Out_j is the output of neuron j. This activation function produces an output in the range of $+1$ to -1.

To match the output range of the neurons, the sensor measurements that should be the output of the estimators are normalized using [25]:

$$x_n = (b - a) \times \frac{x_o - x_{min}}{x_{max} - x_{min}} + a \tag{11}$$

where x_n is the normalized value and x_o is the value to be normalized. a and b are the minimum and maximum value of the range to be normalized to, which in this case is $+1$ to -1. x_{max} and x_{min} are the maximum and minimum values of the range from which x_o is been normalized. This range is set to be $+10$ to -10.

```
procedure INITIALIZATION(Q, g)
    Q ← 0
    g ← 0
end procedure

for all patterns (p = 1 to p = P) do
    procedure FORWARD COMPUTATION
        for all neurons (nn) do
            for all weights of current neuron (j) do
                calculate net input (net_j)
            end for
            calculate neuron output
            calculate neuron slope (s_j)
            s_j = ∂Out_j(net_j) / ∂net_j
            set current slope as delta
            for weights to previous neurons (ny) do
                for previous neurons (nz) do
                    multiply delta through weights
                    then sum
                end for
                multiply sum by the slope
            end for
        end for
        for all outputs (m = 1 to m = M) do
            calculate error
        end for
    end procedure

    procedure UPDATE(Q, g)
        for all outputs (m = 1 to m = M) do
            calculate vector j_{p,m}
            calculate sub matrix q_{p,m}
            calculate sub vector η_{p,m}
            Q = Q + q_{p,m}
            g = g + η_{p,m}
        end for
    end procedure
end for

procedure IMPROVED LM TRAINING
    follow the LM algorithm training process
    update rule: W_{n+1} = W_n − (Q + μI)^{-1} g
end procedure
```

Fig. 3. NBN Algorithm Pseudo Code

The initial value of combination coefficient (μ), used in the weights update rule of the NBN algorithm, is set to 0.01. The factor by which to increase or decrease this value of μ is 10.

IV. Aircraft Simulator and Sensors

Aircraft data is collected using the X-Plane flight simulator [26]. This simulator produces realistic flight simulations due to which its professional version is certified by FAA (Federal Aviation Administration) for pilot training [27], [28]. It is also used by leading defence contractors, air forces and space agencies for applications of flight training, concept design and testing [28].

For this research, the Cessna 172SP aircraft model in X-Plane is used to collect the flight data. Since the main emphasis of the work is on sensor estimation, the aircraft is flown by the provided AI pilot in X-Plane.

It is assumed that the aircraft is equipped with 6 inertial sensors without any hardware redundancy. The inertial sensors are 3 gyroscopes (gyros) and 3 accelerometers. They are mounted along the x, y and z axis of the aircraft. These sensors are essential components of the attitude/heading reference system (AHRS) and the inertial navigation system (INS) found in today's aircrafts [29], [30].

The outputs of these sensors are as follows:

1) Gyros: pitch (q), roll (p) and yaw (r) rates.
2) Accelerometers: accelerations along the x (a_x), y (a_y) and z (a_z) axis.

V. Estimator Development

A. Estimator Neural Network Input/Output and Structure

The paper concentrates on the gyro sensors of the aircraft. Therefore three gyro sensor estimators are developed, one each for the (i) pitch, (ii) roll and (iii) yaw rate gyro sensors. The outputs of these estimators are their respected estimated sensor rates.

The inputs to the estimators are other sensors' measurements (excluding the one it is estimating) and the commanded control values provided by flight control computers. Inputs to each of these estimators and their respected outputs are presented in Table I. These inputs are taken at $t-1$, where t is the current sample time.

These inputs are chosen because they can have an effect or cause an effect on the parameter that the sensor is measuring. The relationship between the measured accelerations and the gyro rates can be derived from the linear acceleration equations [31] defined as follows:

$$\begin{aligned} a_x &= \dot{U} - rV + qW \\ a_y &= \dot{V} - pW + rU \\ a_z &= \dot{W} - qU + pV \end{aligned} \tag{12}$$

where (U, V, W) are the velocity along the X, Y and Z axes, given in body fixed reference frame. The relationships between the control inputs and the gyro rates can be derived from the aircraft's linearized equations of motion [31]. From the aircraft's longitudinal equations of motion, the equation relevant to this paper is as follows [31]:

$$\begin{bmatrix} \dot{q} \end{bmatrix} = \begin{bmatrix} \frac{M_w}{I_y} & \frac{M_q}{I_y} \end{bmatrix} \begin{bmatrix} w \\ q \end{bmatrix} + \begin{bmatrix} \frac{M_{\delta_E}}{I_y} \end{bmatrix} \begin{bmatrix} \delta_E \end{bmatrix} \tag{13}$$

where, $\dot{q}$ is the rate of change of q, w is the vertical velocity increment and δ_E is the elevator demand. M_w, M_{δ_E} and M_q are the pitching moment derivatives due to w, δ_E and

q, respectively. I_y is the moment of inertia of aircraft about the pitch axis. The relevant equations from the lateral motion of the aircraft are as follows [31]:

$$\begin{bmatrix} \dot{p} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{L_v}{I_x} & \frac{L_p}{I_x} & \frac{L_r}{I_x} \\ \frac{N_v}{I_z} & \frac{N_p}{I_z} & \frac{N_r}{I_z} \end{bmatrix} \begin{bmatrix} v \\ p \\ r \end{bmatrix} + \begin{bmatrix} \frac{L_{\delta_R}}{I_x} & \frac{L_{\delta_A}}{I_x} \\ \frac{N_{\delta_R}}{I_z} & \frac{N_{\delta_A}}{I_z} \end{bmatrix} \begin{bmatrix} \delta_R \\ \delta_A \end{bmatrix} \quad (14)$$

where, $\dot{p}$ and $\dot{r}$ are the rate of change of p and r, respectively; L is the rolling moment; N is the yawing moment; v is the side-slip velocity increment; δ_R is the rudder demand; and δ_A is the aileron demand. L_v, L_p, L_r, L_{δ_R} and L_{δ_A} are the rolling moment derivatives due to v, p, r, δ_R and δ_A, respectively. N_v, N_p, N_r, N_{δ_R} and N_{δ_A} are the yawing moment derivatives with respect to v, p, r, δ_R and δ_A. I_z is the moment of inertia of aircraft about the yaw axis.

These equations are derived assuming the aircraft is in steady, straight and level trimmed conditions, with small disturbances. Writing the aircraft equations using these assumptions and linearizing them is a common practice. This helps to simplify the equations and analyze the behavior of the aircraft in response to the control inputs.

In order to select the best structure (topology) for the FCC neural network based sensor estimators, the number of neurons in each estimator is first experimented; varying from 2 to 12 neurons. These estimators with different number of neurons are trained and validated using the process described in the following subsections.

TABLE I
INPUTS TO THE SENSOR ESTIMATORS

Sensor Estimator	Inputs
Pitch (q)	a_z - Normal Acceleration a_x - Longitudnal Acceleration δ_E - Elevator Demand
Roll (p)	r - Yaw Rate δ_A - Aileron Demand δ_R - Rudder Demand
Yaw (r)	a_y - Lateral Acceleration δ_A - Aileron Demand δ_R - Rudder Demand

B. Training and Validation Data Sets

To train and evaluate the estimators, flight data from the Cessna 172SP aircraft in X-Plane is recorded for 6 different flight scenarios. In these scenarios, the aircraft takes off from different airports to capture different maneuvers performed by the AI pilot in X-Plane. These flight data contain various sensor readings and control inputs, recorded every second.

In a practical system, sensor readings are updated at a higher frequency. In this case, however, recording the flight data at every second allows greater dynamics of the data to be captured within a single and short time window for the training data.

These scenarios were simulated in turbulent free weather conditions. The data was recorded once the aircraft reached its cruise altitude. Out of these scenarios, 1 of them is used for training and the remaining 5 are used for validating the estimators. Only 1 training set is used to train the FCC neural network. This is to test the capabilities of the NBN algorithm in training the neural network.

C. Estimator Training

The estimators are trained offline (batch learning) using a fixed set of training data extracted from the training flight data mentioned in the previous section. The training set consists of data collected during the steady and transient state of flight. This ensures that the estimators can produce good estimates during any state of flight. The estimators are trained until the Sum Squared Error (SSE) of the epoch is ≤ 0.01 or a maximum of 101 epochs is reached.

D. Simulation for Validation

Once trained, each of the estimators (ranging from 2 to 12 neurons) for a sensor are validated on the 5 different flight scenarios. These scenarios last for 1500 seconds, therefore containing 1500 patterns. To assess the performance of the estimator on the scenario, the total Sum Squared Error (SSE) of all the patterns in the scenarios is computed. Finally, the best estimator for a sensor is selected by calculating the average and the standard deviation of the SSE for all the scenarios.

VI. Results and Discussion

A. Yaw Rate Estimator

The Sum Squared Errors (SSE) of the yaw rate estimators using different neurons on the 5 validation scenarios are presented in Table II. From the average SSE, estimator networks with 2 and 5 neurons produce the least errors. Using the standard deviation, it is clear that the estimator with 2 neurons is the best among the two.

The output of this estimator on its best and worst scenarios is presented in Fig. 4 and Fig. 5 respectively. The best scenario is scenario 3 and the worst is scenario 1.

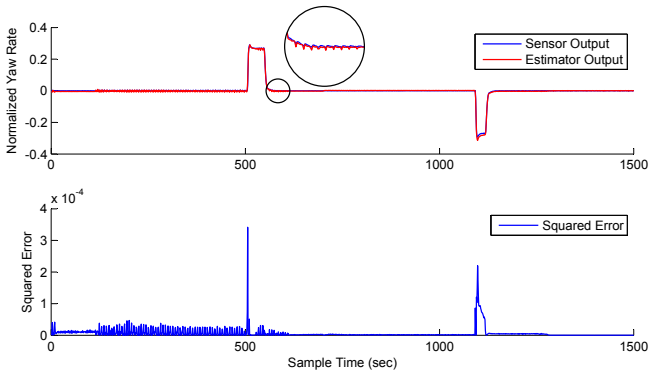


Fig. 4. Normalized yaw rate using (11) and the associated SSE. Results using 2 neurons in scenario 3.

TABLE II
YAW RATE ESTIMATOR ERRORS FOR THE VALIDATION SCENARIOS

	SUM SQUARED ERRORS (SSE)										
Neurons	2	3	4	5	6	7	8	9	10	11	12
Scenario 1	0.05403	0.05790	0.05775	0.04916	0.14770	1.08820	0.08818	0.14520	0.41494	0.22531	0.74185
Scenario 2	0.01678	0.03428	0.02443	0.02183	0.23910	2.11596	0.13171	0.29886	1.26390	0.36817	1.23907
Scenario 3	0.00923	0.04650	0.03638	0.01692	0.25249	5.99460	0.25252	0.91540	2.30973	0.99688	3.57601
Scenario 4	0.05017	0.10001	0.04895	0.06144	0.11770	0.69716	0.08022	0.05969	0.27662	0.07500	0.47789
Scenario 5	0.03428	0.05799	0.04082	0.03250	0.44971	3.44615	0.35203	0.39369	2.05501	0.39984	2.04609
Average Error	0.03290	0.05934	0.04167	0.03637	0.24134	2.66841	0.18093	0.36257	1.26404	0.41304	1.61618
SD	0.01979	0.02475	0.01262	0.01869	0.13000	2.14222	0.11786	0.33528	0.92399	0.35091	1.24781

TABLE III
PITCH RATE ESTIMATOR ERRORS FOR THE VALIDATION SCENARIOS

	SUM SQUARED ERRORS (SSE)										
Neurons	2	3	4	5	6	7	8	9	10	11	12
Scenario 1	0.87549	1.07204	0.77152	0.75686	0.62664	8.31579	1.07375	1.73602	1.60933	1.80456	4.34395
Scenario 2	1.30509	1.19663	1.06827	0.97039	0.92628	5.95501	1.15038	1.79250	2.21795	2.63000	6.52519
Scenario 3	3.09023	2.83352	2.80383	2.70440	2.74835	8.32024	2.88546	2.97336	3.03034	4.02907	6.08809
Scenario 4	0.64407	0.58858	0.46337	0.37710	0.34452	8.91346	0.50092	1.05716	1.20740	3.24782	4.86670
Scenario 5	1.78657	1.59820	1.35882	1.32644	1.15308	13.43923	1.87371	2.78344	2.85945	2.95253	7.94092
Average Error	1.54029	1.45779	1.29316	1.22704	1.15978	8.98874	1.49684	2.06850	2.18489	2.93280	5.95297
SD	0.96998	0.84925	0.90797	0.89468	0.93901	2.73477	0.91668	0.79678	0.78385	0.81640	1.42008

TABLE IV
ROLL RATE ESTIMATOR ERRORS FOR THE VALIDATION SCENARIOS

	SUM SQUARED ERRORS (SSE)										
Neurons	2	3	4	5	6	7	8	9	10	11	12
Scenario 1	1.03861	1.71942	0.99188	1.26806	1.17937	1.23314	1.33951	1.06159	1.12756	1.03996	0.95751
Scenario 2	0.87189	0.71655	0.62950	0.74557	0.73817	0.80793	0.83962	0.90883	1.00083	0.75335	0.72139
Scenario 3	0.33534	0.48925	0.41012	0.44700	0.42922	0.43207	0.45478	0.64603	0.68607	0.56293	0.44613
Scenario 4	0.50070	0.92344	0.82084	1.07419	0.98542	1.01214	1.05378	1.36040	1.40734	1.21942	0.94279
Scenario 5	1.42533	1.46552	1.32818	1.53932	1.41908	1.58002	1.60261	1.20004	1.22841	1.27567	1.08864
Average Error	0.83437	1.06284	0.83610	1.01483	0.95025	1.01306	1.05806	1.03538	1.09004	0.97027	0.83129
SD	0.43380	0.51517	0.35028	0.42944	0.38405	0.43287	0.44369	0.27434	0.27036	0.30544	0.25250

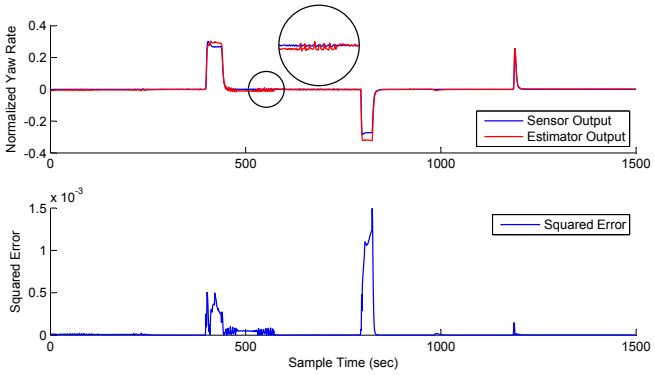


Fig. 5. Normalized yaw rate using (11) and the associated SSE. Results using 2 neurons in scenario 1.

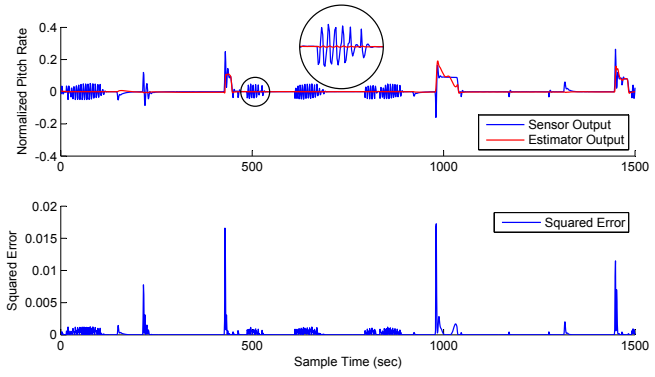


Fig. 6. Normalized pitch rate using (11) and the associated SSE. Results using 6 neurons in scenario 4.

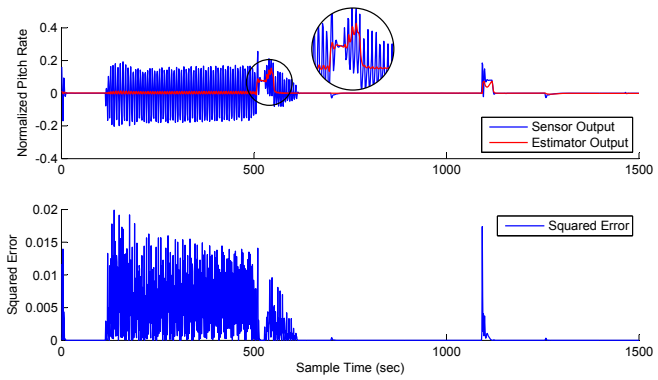


Fig. 7. Normalized Pitch rate using (11) and the associated SSE. Results using 6 neurons in scenario 3.

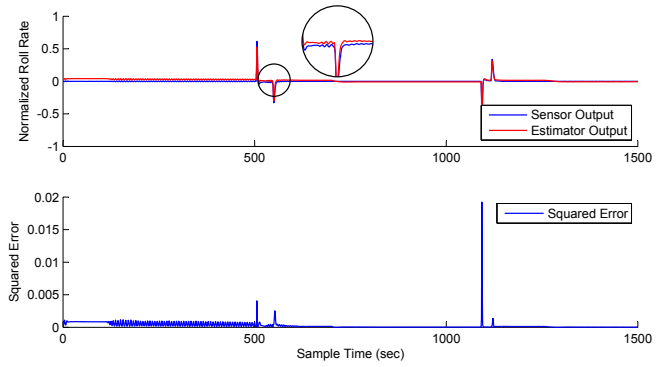


Fig. 8. Normalized roll rate using (11) and the associated SSE. Results using 4 neurons in scenario 3.

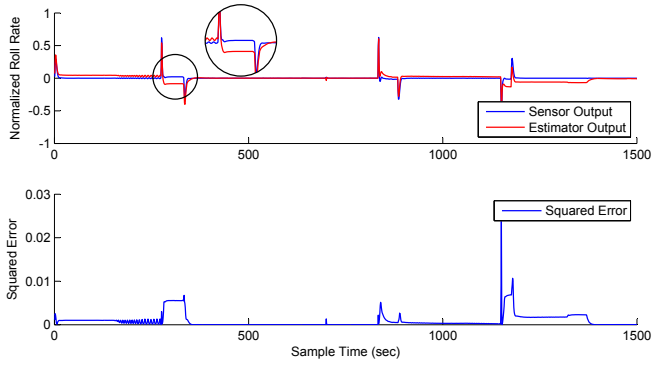


Fig. 9. Normalized roll rate using (11) and the associated SSE. Results using 4 neurons in scenario 5.

B. Pitch Rate Estimator

Table III presents the validation results for the pitch rate estimators. The estimator using 6 neurons has the least SSE among them. The output of this estimator on its best and worst scenarios is presented in Fig. 6 and Fig. 7 respectively. The best scenario is scenario 4 and the worst is scenario 3.

As can be seen from Fig. 6 and Fig. 7, the output of the pitch rate sensor seems to oscillate rapidly over certain time frames. This is due to the aircraft being disturbed from its equilibrium state. These disturbances could be initiated by pilot control inputs, change in power settings and atmospheric influences like gust and turbulence [32]. Since the scenarios where simulated in turbulent free weather conditions, in this case, the oscillations are caused by the control outputs from the AI pilot in X-Plane. For a certain time frame, the oscillations are neither increasing nor decreasing in magnitude (Fig. 7). Once the aircraft is disturbed, it continues to oscillate without a significant increase or decrease in magnitude. The aircraft is said to be in a state of neutral dynamic stability [31], [33]. The magnitude and duration of these oscillations depends on the aircraft's aerodynamics and stability.

The estimator follows these oscillations but does not follow the magnitude. To investigate this anomaly, the estimator with 6 neurons was validated on the training data itself. The estimator output followed all the training data points, except the points with rapid oscillations. This leads to the conclusion that the anomaly is due to the lack of inputs.

Additional inputs could help capture the aircraft's aerodynamics within the neural network, which would then help to follow the magnitude of the sensor during the oscillatory phase of the aircraft. Future work would focus on identifying the inputs to the estimator to follow the magnitude of the oscillatory phase.

C. Roll Rate Estimator

In Table IV the results for the roll rate estimators are presented. As can be seen from the average SSE, there is a close tie between neurons 2, 4 and 12. Using the standard deviations of their SSE, the estimator with 12 neurons has the best normal distribution among them with 3 being the worst. However, it was decided to select the estimator with 4 neurons, keeping in line with the low neuron count of the previous gyro sensor estimators.

The output of this estimator on its best and worst scenarios is presented in Fig. 8 and Fig. 9 respectively. The best scenario is scenario 3 and the worst is scenario 5.

VII. Conclusion

Fault tolerant flight control system (FTFCS) can increase the endurance of an aircraft in case of failures. As part of the FTFCS, the sensor failure detection, identification and accommodation (SFDIA) scheme, must detect any faulty sensor and replace it with a reliable estimate.

The neuron by neuron (NBN) learning algorithm is an improved version of the Levenberg-Marquardt (LM) algorithm. This algorithm is combined with the fully connected cascade (FCC) neural network to develop the sensor estimators for the pitch, roll and yaw rate gyros of an aircraft. These estimators can be used in any SFDIA scheme to provide the failure accommodation (FA).

The results show that the proposed algorithm and neural network architecture can produce good estimates of the sensor measurements, with as little as 2 neurons (see yaw rate results in Section VI). The pitch and roll rate estimators, were able to produce good estimates with just 6 and 4 neurons respectively;

in comparison to the SFDIA scheme presented in [8]. In the scheme presented in [8], the estimators are based on the multilayer perceptron (MLP) neural network using 1 hidden layer. They are trained using the extended back propagation learning algorithm (EBPA). These estimators require 20, 30 and 18 neurons in their hidden layer, to produce reliable pitch, roll and yaw rate estimates, respectively. This scheme, presented in [8], also uses more inputs to its estimators, compared to 3 inputs each to the pitch, roll and yaw rate estimators, presented in this paper.

On-going research is aimed at using the NBN algorithm with the FCC neural network architecture to develop a fully FTFCS; incorporating a SFDIA scheme and a actuator failure detection, identification and accommodation (AFDIA) scheme. Future work also aims to identify additional inputs that can solve the limitation of the pitch estimator.

ACKNOWLEDGMENT

This research is part-funded by the University of Central Lancashire (UCLan) and Military Air and Information (MAI), BAE Systems, UK. The authors thank Adam Bedford of UCLan and Mohiuddin Rahman of University of Glasgow, UK, for their advice with this research. In addition, we acknowledge the supportive reviews from our colleagues at BAE MAI.

REFERENCES

[1] B. M. Wilamowski, "How to not get frustrated with neural networks," *2011 IEEE International Conference on Industrial Technology*, pp. 5–11, Mar. 2011.

[2] B. Wilamowski, "Neural network architectures and learning algorithms," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 56–63, Dec. 2009.

[3] H. Yu and W. Auburn, "Fast and efficient and training of neural networks," in *3rd International Conference on Human System Interaction*. IEEE, May 2010, pp. 175–181.

[4] B. Wilamowski, D. Hunter, and A. Mabnowski, "Solving parity-N problems with feedforward neural networks," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 4. IEEE, 2003, pp. 2546–2551.

[5] D. Hunter, H. Yu, and M. Pukish, "Selection of Proper Neural Network Sizes and ArchitecturesA Comparative Study," *Industrial Informatics, ...*, vol. 8, no. 2, pp. 228–240, May 2012.

[6] I. Samy, I. Postlethwaite, and D. Gu, "Neural network based sensor validation scheme demonstrated on an unmanned air vehicle (UAV) model," in *2008 47th IEEE Conference on Decision and Control*. IEEE, 2008, pp. 1237–1242.

[7] G. Campa, M. Fravolini, M. Napolitano, and B. Seanor, "Neural networks-based sensor validation for the flight control system of a B777 research model," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 1. American Automatic Control Council, 2002, pp. 412–417.

[8] M. Napolitano, "A fault tolerant flight control system for sensor and actuator failures using neural networks," *Aircraft Design*, vol. 3, no. 2, pp. 103–128, Jun. 2000.

[9] P. Goupil, "AIRBUS state of the art and practices on FDI and FTC in flight control system," *Control Engineering Practice*, vol. 19, no. 6, pp. 524–539, Jun. 2011.

[10] Y. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *Annual Reviews in Control*, vol. 32, no. 2, pp. 229–252, Dec. 2008.

[11] A. S. Willsky, "A survey of design methods for failure detection in dynamic systems," *Automatica*, vol. 12, no. 6, pp. 601–611, Nov. 1976.

[12] I. Samy, I. Postlethwaite, and D.-W. Gu, "Survey and application of sensor fault detection and isolation schemes," *Control Engineering Practice*, vol. 19, no. 7, pp. 658–674, Jul. 2011.

[13] ——, "Detection and accommodation of sensor faults in UAVs- a comparison of NN and EKF based approaches," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, Dec. 2010, pp. 4365–4372.

[14] R. Isermann and P. Ballé, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control Engineering Practice*, vol. 5, no. 5, pp. 709–719, May 1997.

[15] T.-H. Guo and J. Musgrave, "Neural network based sensor validation for reusable rocket engines," in *Proceedings of 1995 American Control Conference - ACC'95*, vol. 2. American Autom Control Council, 1995, pp. 1367–1372.

[16] B. M. Wilamowski, N. Cotton, J. Hewlett, and O. Kaynak, "Neural Network Trainer with Second Order Learning Algorithms," in *Intelligent Engineering Systems, 2007 International Conference on*. IEEE, Jun. 2007, pp. 127–132.

[17] B. M. Wilamowski and H. Yu, "Neural network learning without backpropagation." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 21, no. 11, pp. 1793–803, Nov. 2010.

[18] B. M. Wilamowski, "C++ implementation of neural networks trainer," in *2009 International Conference on Intelligent Engineering Systems*. IEEE, Apr. 2009, pp. 257–262.

[19] B. Wilamowski, "Challenges in applications of computational intelligence in industrial electronics," *Industrial Electronics (ISIE), 2010 IEEE*, pp. 15–22, Jul. 2010.

[20] B. M. Wilamowski, "Advanced learning algorithms," in *2009 International Conference on Intelligent Engineering Systems*. IEEE, Apr. 2009, pp. 9–17.

[21] B. M. Wilamowski and H. Yu, "Improved computation for Levenberg-Marquardt training." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 21, no. 6, pp. 930–7, Jun. 2010.

[22] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 5, no. 6, pp. 989–993, Jan. 1994.

[23] B. Wilamowski, H. Yu, and N. Cotton, "NBN Algorithm," in *Intelligent Systems*, ser. Electrical Engineering Handbook. CRC Press, Feb. 2011, pp. 1–24.

[24] B. Wilamowski, "Understanding Neural Networks," in *Intelligent Systems*, ser. Electrical Engineering Handbook. CRC Press, Feb. 2011, pp. 1–11.

[25] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, Mar. 1998.

[26] Laminar Research, "X-Plane 10 Global — The Worlds Most Advanced Flight Simulator — X-Plane.com." [Online]. Available: http://www.x-plane.com

[27] ——, "FAA-Certified X-Plane." [Online]. Available: http://www.x-plane.com/pro/certified/

[28] ——, "X-Plane 10 Manual," 2012. [Online]. Available: http://www.x-plane.com/files/manuals/X-Plane_10_Desktop_manual.pdf

[29] R. Collinson, "Navigation Systems," in *Introduction to Avionics Systems*. Dordrecht: Springer Netherlands, 2011, pp. 303–376.

[30] R. P. G. Collinson, "Inertial Sensors and Attitude Derivation," in *Introduction to Avionics Systems*. Springer Netherlands, 2011, pp. 255–302.

[31] R. Collinson, "Aerodynamics and Aircraft Control," in *Introduction to Avionics Systems*. Dordrecht: Springer Netherlands, 2011, pp. 101–117.

[32] M. V. Cook, *Flight Dynamic Principles*, 2nd ed. Oxford: Butterworth-Heinemann, 2007.

[33] T. A. Talay, "Stability and Control." [Online]. Available: http://history.nasa.gov/SP-367/chapt9.htm

Sensor Failure Detection, Identification and Accommodation Using Fully Connected Cascade Neural Network

Saed Hussain, *Student Member, IEEE,* Maizura Mokhtar, *Member, IEEE,* and Joe M. Howe

***Abstract*—Modern control systems rely heavily on their sensors for reliable operation. Failure of a sensor could destabilize the system, which could have serious consequences to the system's operations. Therefore there is a need to detect and accommodate such failures, especially if the system in question is of a safety critical application. In this paper, a sensor failure detection, identification and accommodation (SFDIA) scheme is presented. This scheme is based on the fully connected cascade (FCC) neural network (NN) architecture. The NN is trained using the neuron by neuron (NBN) learning algorithm. This NN architecture is chosen because of its efficiency in terms of the number of neurons and the number of inputs required to solve a problem. The SFDIA scheme considers failures in pitch, roll and yaw rate gyro sensors of an aircraft. A total of 105 experiments were conducted; out of which, only one went undetected. The SFDIA scheme presented here is efficient, compact and computationally less expensive, in comparison to schemes using, for example, the popular multi-layer perceptron (MLP) NN. These benefits are inherited from the FCC NN architecture.**

***Index Terms*—Sensors, neural networks, fault tolerance, failure detection, analytical redundancy.**

I. Introduction

Sensors are vital components of any control system. They inform the controller about its environment and the state of the system. With increasing safety, performance and automation requirements, control systems are increasingly sophisticated and are heavily reliant on their sensors. However, sensors are often considered as the weak link in these systems [1], [2].

Any sensor failure could degrade the system's performance and possibly lead to total system failure. The impact of the failure depends on the application domain. In safety critical applications, any failure could result in damage to property or environment and in worst case scenario, result in loss of life. Therefore sensor failure detection, identification and accommodation (SFDIA) is an important area of research in the safety critical systems domain.

Manuscript received April 28, 2013; revised September 23, 2013, March 6, 2014 and August 7, 2014; accepted September 6, 2014.

Saed Hussain is with the School of Computing, Engineering and Physical Sciences (CEPS), University of Central Lancashire (UCLan), PR1 2HE, United Kingdom (UK) (e-mail: SHussain23@uclan.ac.uk)

Maizura Mokhtar is with the Department of Automatic Control and Systems Engineering (ACSE), University fo Sheffield, S1 3JD, UK, and was with the School of CEPS, UCLan, UK (email: M.Mokhtar@sheffield.ac.uk)

Joe M. Howe is with the Thornton Energy Institute, University of Chester, Thornton Science Park, Chester, CH2 4NU, UK, and was with the School of CEPS, UCLan, UK (e-mail: j.howe@chester.ac.uk)

An aircraft system is a good example of a safety critical system. Sensor failures are particularly important to an aircraft, due to their role in the feedback control loop. If measurements from a faulty sensor enter the control loop, it can lead to closed loop instability which can eventually result in undesirable, or worst, unrecoverable flight conditions [3], [4]. Therefore these systems must have robust sensor fault tolerance mechanisms.

This paper presents a neural network (NN) based SFDIA scheme, with an aircraft system as the application domain. Neural networks have been used in various applications, including fault diagnosis and detection [5]–[11]. The popular architecture for NN based applications is the multi-layer perceptron (MLP) NN [12]–[14]. However, the scheme presented here is based on the fully connected cascade (FCC) NN architecture. This architecture is selected due to its ability to solve a problem with a small number of neurons [13], [15]–[17]. It should be noted that this paper is based on the research presented by the authors in [18]. In [18], FCC NN based aircraft pitch, roll and yaw rate sensor estimators are developed. These estimators are utilized by the SFDIA scheme presented here.

This paper is organized as follows: Section II provides a brief review of the SFDIA methods detailed in the literature. Section III provides an overview of the FCC NN architecture and also presents the settings used for this research. The outline of the SFDIA scheme developed is discussed in Section IV. The sensor suite used to collect data for this research is briefly described in Section V. The FCC NN based sensor estimator development process presented in [18] is summarized in Section VI. In Section VII, the failure types considered is discussed. The experimental results are presented in Section VIII and summarized in Section IX. And finally, Section X, concludes the paper.

II. Brief Review of SFDIA Methods

The state of the art practice for fault detection, identification and accommodation (FDIA) is to implement high levels of hardware redundancy [3], [10], [19], [20]. For example, Airbus A320/330/340/380 has triple or quadruple redundant actuation, sensor and flight control systems [19]. In hardware redundancy (see Fig. 1), identical sensors are used to measure the same parameter. A voting scheme is then employed to detect and identify any faulty sensor [4], [11], [19]. For example, in a system with three redundant sensors, if the signal from one sensor differs significantly from the remaining two sensors,

the sensor is declared as faulty. Sensor failure accommodation is achieved by replacing the faulty sensor with one of the two remaining sensors.

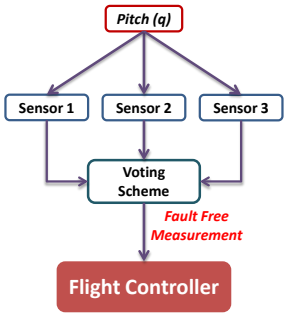


Fig. 1. Block diagram of hardware redundancy

However, for small aircraft's like UAVs, this method has serious implications in terms of cost, power and weight. Due to these implications, analytical redundancy has become a far more appealing approach for SFDIA.

Generally, in analytical redundancy (see Fig. 2), a model of the monitored system is used to generate signals that would otherwise be generated by redundant hardware. In its simplest form, the difference between the model estimate and the measured reading is used to generate an error residual. This residual is then monitored to detect and identify faults [11].

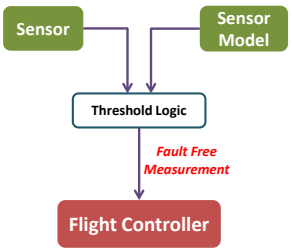


Fig. 2. Block diagram of analytical redundancy

In the literature, early works on analytical redundancy are mostly based on observers and Kalman filters [4], [10], [11], [20]–[22]. These techniques relied on the linear time invariant mathematical model of the systems. In an aircraft system, the assumption that the system is linear, is not often valid throughout the entirety of the flight envelope [22], [23]. Therefore, these techniques might perform inadequately, in the non-linear regions of the flight envelope. In addition, these techniques can suffer from modeling discrepancies between real and mathematical model of the system [24]. Recent literature, has seen efforts been made to address these issues, especially with the linearity assumption of the Kalman filters [25]. Several versions of the Kalman filter has been developed and applied to various fault tolerance and state estimation problems in non-linear systems [7], [24], [26]–[30].

Over the past two decades, there has been an increasing interest in the application of NN for SFDIA schemes [3], [10], [21], [24], [31]–[33]. This is mainly due to their innate ability to model both linear and non-linear systems [5], [34]. Unlike the Kalman filters, they do not require a detailed mathematical description of the system. They develop a structure based on training data instead. In addition, they can also be made to adapt on-line, whilst the system is in use; in order to adapt to the dynamic conditions of the environment and the system dynamics. On-line adaptation is provided by the on-line learning algorithm.

Example applications of NN based SFDIA schemes include, an SFDIA scheme for the space shuttle main engine using the auto-associative MLP NN, presented in [33]. This NN architecture has also been used for fault detection in intelligent sensors [8], [31]. The authors of [10] developed a SFDIA scheme using the hetero-associative MLP NN. This scheme was evaluated on the pitch, roll and yaw rate gyro sensors. Samy, Postlethwaite and Gu proposed a SFDIA scheme using the radial basis function (RBF) NN in [3] and [11].

In this paper, a SFDIA scheme using the FCC NN [13] is presented. This NN is trained using the neuron by neuron (NBN) learning algorithm [13]. In an SFDIA scheme, once a sensor failure is detected and identified (FDI), the faulty sensor reading is replaced with a reliable estimate, a process known as failure accommodation (FA). SFDIA schemes based on NN replaces the faulty sensor reading with a NN generated estimate. It other words, the NN works as the estimator.

III. Neural Network

A. Fully Connected Cascade Neural Network

In the literature, the MLP NN architecture is the popular choice for NN applications [12]–[14]. This architecture however is neither powerful nor efficient, in comparison to other architectures with connections across layers [13], [15], [17], [35].

The FCC NN architecture, presented in Fig. 3, allows connections across layers. Compared to the popular MLP architecture, this architecture is compact and efficient as it requires less neurons to solve a problem. For example Table I, which is adapted from [15], compares the two architecture to solve the parity-n problem. This is a common benchmarking problem for neural networks [13].

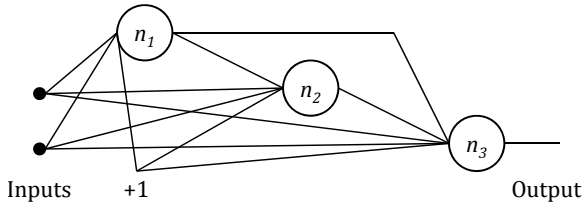


Fig. 3. FCC Neural Network Architecture

As can be seen from Table I, to solve a parity-7 problem, the MLP architecture requires 8 neurons compared to 3 by the FCC. The FCC architecture is capable of solving a parity 63 problem with just 6 neurons, compared to 64 by the MLP. With increasing number of neurons, the computational expense increases due to an increase in the number of network weights.

In Table I, to solve the parity-31 problem, the MLP architecture with 32 neurons has 1024 weights; whereas the FCC

TABLE I
COMPARISON OF FCC AND MLP ARCHITECTURE TO SOLVE PARITY-N PROBLEM

Parity	Architecture	No. Neurons	No. Weights
3	MLP	4	16
	FCC	2	9
7	MLP	8	64
	FCC	3	27
15	MLP	15	256
	FCC	4	70
31	MLP	32	1024
	FCC	5	170
63	MLP	64	4096
	FCC	6	399

Note: This table is adapted from [15]. The MLP architecture is made of 1 hidden layer.

architecture with just 5 neurons has 170 weights. The FCC architecture requires significantly low number of weights and neurons compared to MLP due to its unique architecture. It is clear from Table I, that a signification saving in computation expense can be made, if a system based on MLP is updated to use the FCC architecture. Based on these results, the FCC NN is chosen for use as the NN based sensor estimator.

B. Training Algorithm

To train the FCC NN based sensors estimators, the NBN learning algorithm is used. The NBN algorithm is an improved version of the second order Levenberg-Marquardt (LM) algorithm [15]–[17]. Using the NBN algorithm, the weights are updated as follows:

$$W_{n+1} = W_n - (Q + \mu I)^{-1} g \tag{1}$$

where W_{n+1} is the new weights vector, W_n is the previous weights vector, Q is the quasi-Hessian matrix, g is the gradient vector and μ is the combination coefficient.

In comparison to the popular error back propagation (EBP) training algorithm, this algorithm can not only train fast, but also efficiently, with small number of neurons within the NN [13], [15]–[17], [35]–[38].

C. Neural Network Settings

The neural networks are initialized with random weights in the range of $+1.5$ to -1.5. The activation function used by the neurons is the bipolar sigmoid [39], defined as follows:

$$Out_j = \frac{2}{1 + e^{-net_j}} - 1 \tag{2}$$

where net_j is the sum of the weighted inputs to neuron j and Out_j is the output of neuron j. This activation function produces an output in the range of $+1$ to -1.

To match the output range of the neurons, the sensor measurements that should be the output of the estimators are normalized using [40]:

$$x_n = (b - a) \times \frac{x_o - x_{min}}{x_{max} - x_{min}} + a \tag{3}$$

where x_n is the normalized value and x_o is the value to be normalized. a and b are the minimum and maximum value of the range to be normalized to, which in this case is $+1$ to -1. x_{max} and x_{min} are the maximum and minimum values of the range from which x_o is normalized. This range is set to be $+10$ to -10.

The initial value of combination coefficient (μ), used in the weights update rule of the NBN algorithm, is set to 0.01. The factor by which to increase or decrease this value of μ is 10.

IV. SFDIA OUTLINE

For every sensor considered, there is a NN based sensor estimator. As the name suggest, the output of this estimator is the sensor measurement it is estimating. Also associated with each sensors, is a fault alarm signal (F_A), which could either be '0' or '1': where $F_A = 1$ indicates a fault and $F_A = 0$ if otherwise. Failure detection (FD) is performed by evaluating the residual between each sensor and its associated NN estimate. If the residual exceeds a certain threshold, the failure alarm for that sensor is triggered $(F_A = 1)$. Failure identification (FI) is performed by identifying which sensor fault alarm is triggered. Once the failed sensor is identified, it remains in the failed state throughout the process.

In addition, the proposed scheme consists of a fault switch (F_S) for every sensor. The inputs to the fault switch are the fault alarm signal (F_A), sensor output and estimator output. This switch is controlled by the F_A signal. In fault free conditions $(F_A = 0)$, the output of F_S is the sensor output. However in the event of failure $(F_A = 1)$, the F_S switches to the estimator output. The block diagram of the SFDIA scheme for the pitch rate sensor is presented in Fig. 4. Note that the SFDIA scheme and the experiments presented here, only addresses single sensor failure at a time. However, this scheme could be extended to address multiple sensor failures (simultaneously or in series), similar to the research presented in [11].

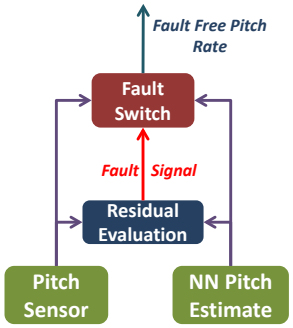


Fig. 4. SFDIA scheme layout for pitch rate sensor

V. THE SENSOR SUITE

The SFDIA scheme is applied to an aircraft's sensor suite. To test the functionality of the SFDIA scheme, flight data (which includes sensor readings) is collected from the X-Plane flight simulator [41]. This simulator is known for its realistic flight simulations, due to which its professional version is

certified by the FAA (Federal Aviation Administration) for pilot training [42], [43]. The aircraft model used for the simulations is the Cessna 172SP. This aircraft is flown by the artificial intelligence (AI) pilot in X-Plane.

For this research, it is assumed that the aircraft is equipped with six inertial sensors without any hardware redundancy. The inertial sensors are three gyroscopes (gyros) and three accelerometers. They are mounted along the x, y and z axis of the aircraft. These sensors are essential components of the attitude/heading reference system (AHRS) and the inertial navigation system (INS) found in modern aircraft [44], [45]. The outputs of these sensors are as follows:

1. Gyros: pitch (q), roll (p) and yaw (r) rates.
2. Accelerometers: accelerations along the x (a_x), y (a_y) and z (a_z) axis.

VI. Estimator Development

A. Estimator NN Inputs, Outputs and Structure

In total three estimators are developed, one each for the pitch, roll and yaw rate gyros. The inputs to the estimators are measurements from other sensors, excluding the one it is estimating. In addition, commanded control outputs by the flight control computer is also used as inputs. These inputs are taken at the current sample time t. Inputs to each of these estimators and their respected outputs are presented in Table II. These inputs are chosen because they can have an effect or cause an effect on the parameter that the sensor is measuring.

TABLE II
Inputs to the Sensor Estimators [18]

Sensor Estimator	Inputs
Pitch (q)	a_z - Normal Acceleration a_x - Longitudnal Acceleration δ_E - Elevator Demand
Roll (p)	r - Yaw Rate δ_A - Aileron Demand δ_R - Rudder Demand
Yaw (r)	a_y - Lateral Acceleration δ_A - Aileron Demand δ_R - Rudder Demand

The relationship between the measured accelerations and the gyro rates are as follows [46]:

$$\begin{aligned} a_x &= \dot{U} - rV + qW + g_x \\ a_y &= \dot{V} - pW + rU + g_y \\ a_z &= \dot{W} - qU + pV + g_z \end{aligned} \tag{4}$$

where (U, V, W) and (g_x, g_y, g_z) are the velocity and gravitational acceleration components respectively, along the X, Y and Z axes, given in body fixed reference frame. Other input relationships to the outputs are indicated in [47].

B. Estimator Training and Validation Data

Using the X-Plane simulator, six different flight scenarios are recorded to train and evaluate the FCC NN based SFDIA scheme. Out of the 6, 1 is chosen at random to train the NN based estimators for each of the sensors; and the remaining 5 are used to validate the estimators. The results of the training and validation process are presented in [18]. The 5 validation scenarios are used again to evaluate the SFDIA scheme.

In the simulations, the aircraft takes off from different airports to capture different maneuvers performed by the AI pilot in X-Plane. The maneuvers include, take-off, straight flight and randomly changing flight heading. The flight data contains various sensor readings and control inputs, recorded every second. Although in a practical system, sensor readings are updated at a higher frequency; recording the flight data at every second allows the training data to capture more dynamic flight characteristics between each training pattern. This helps to prevent the estimator NN from over-fitting to less dynamic training data.

C. Estimator Development Summary

The development process of the NN based sensor estimators is presented in [18]. The process can be summarized into the following steps:

1. *Estimator Training*: The aim of the estimator development process is to develop and select the best sensor estimator in terms of size and error. In this step, for each sensor considered, estimators with 2 to 12 neurons are trained offline (batch learning). The training data is extracted from the training data scenario described in the previous subsection. The estimators are trained until the Sum Squared Error (SSE) of the epoch is ≤ 0.01 or a maximum of 101 epochs is reached.
2. *Estimator Validation*: In this step, the estimators with varying number of neurons, trained for each sensor are validated on the 5 flight scenarios. Each scenario has a duration of $1500s$, therefore containing 1500 patterns. The performance of the estimator on the scenario is assessed by calculating the total Sum Squared Error (SSE) of all the patterns in the scenario. The best estimator for a sensor is then selected by calculating the average and the standard deviation of the SSE for all the scenarios.

The results of the best number of neurons for the estimators are presented in Table III.

TABLE III
Structure of the Sensor Estimators [18]

Sensor Estimator	No. Neurons
Pitch	6
Roll	4
Yaw	2

D. Estimator Size Comparison

The SFDIA scheme presented here is computationally less expensive in comparison to that of [10] and [11]. This is due to the low neuron count and number of inputs to each estimators when compared against [10] and [11]. For example, Table IV compares the parameters used in this SFDIA scheme with the NN based sensor estimators presented in [10] and [11].

The estimators in [10] are based on the MLP NN using 1 hidden layer. As can be seen from the table, the estimators in [10] require 20, 30 and 18 neurons for the pitch, roll and yaw rate sensors, respectively. In comparison, the FCC NN based estimators require 6, 4 and 2 neurons for the pitch, roll and yaw rate sensors, respectively. In addition, the estimators in [10] have considerably higher number of inputs compared to the estimators presented here. This is due to the use of historical values (previous time (t) instances) as inputs to these estimators [10].

In [11], the SFDIA scheme was developed for the pitch rate (q), normal acceleration (a_z) and angle of attack (α) sensors. Therefore only the pitch rate (q) sensor is compared in Table IV. This SFDIA scheme [11] is based on the extended minimum resource allocating radial basis function (EMRAN RBF) NN. Using the EMRAN algorithm, the number of neurons can vary between 0 and 10, based on the performance of the estimator. Since no information on the average number of neurons is presented in [11], the maximum value of 10 is considered for comparison. This is 4 additional neurons compared to the FCC NN based estimators.

TABLE IV
COMPARISONS OF THE SENSOR ESTIMATOR NEURAL NETWORKS

Sensor	Parameters	MLP [a]	FCC	EMRAN RBF [b]
Pitch (q)	No. Neurons	20	6	10
	No. Input Variables	4	3	4
	Input Pattern[c]	5	1	1
	Total Inputs[d]	20	3	4
Roll (p)	No. Neurons	30	4	-
	No. Input Variables	6	3	-
	Input Pattern[c]	5	1	-
	Total Inputs[d]	30	3	-
Yaw (r)	No. Neurons	18	2	-
	No. Input Variables	6	3	-
	Input Pattern[c]	5	1	-
	Total Input[d]	30	3	-

[a] Ref [10]. The estimator architecture consist of 1 hidden layer.
[b] Ref [11]. No. of neurons represent the maximum value.
[c] No. of time (t) instances.
[d] No. of time instances (t) $\times$ No. of input variables.

VII. SENSOR FAILURE EXPERIMENTS

A. Failure Detection and Identification Experiment Setup

The X-Plane 9 flight simulator does not support simulation of sensor faults. Therefore, faults have to be introduced manually once the flight data for a simulation is collected. The 5 scenarios used to validate the NN based estimators (see VI-B), are used to evaluate the SFDIA scheme. Faults are introduced manually at random locations into these 5 scenarios. Note that although, the flight data contains data collected during the take-off, straight flight and flight heading changes, faults are not introduced during the take-off phase.

For every fault type considered, the faults are simulated on each of the scenarios for every sensor. This would allow the examination of the performance of the SFDIA scheme for each of the fault type for every sensor. The fault types considered in this research are discussed in the following subsection.

B. Sensor Failure Types

Sensors can fail in several ways. Some failures are specific to a sensor, while others are general. The signal from a sensor could be described as follows [1], [22], [48]:

$$x_t = s_t + n_t + f_t \tag{5}$$

where at time t, x is the signal from the sensor, s is the useful signal, n is the noise and f is the sensor failure. The sensor data collected from X-Plane consists of $s + n$ value. The f signal is injected manually for each fault type. In this research, the following fault types are considered [1], [4], [49], [50]:

- *Stuck constant bias failure*: At a given time, the sensor output gets stuck and outputs a constant bias b.

$$x_t = b \tag{6}$$

- *Additive (drift) failure*: This type of failure is very common. They are usually caused due to temperature changes or calibration problems. In this fault, a constant term (drift value) is added to the sensor output. Additive fault can be modeled using the following equation [4], [22], [49]:

$$f_t = \begin{cases} 0 & t < t_f \\ A(t - t_f)/T_R & t_f \leq t < t_f + T_R \\ A & t \geq t_f + T_R \end{cases} \tag{7}$$

where t_f is the time when the fault is introduced, T_R is the duration of the ramp and A is the fault magnitude. The magnitude A of the additive fault can either be large or small. Depending on the duration of the ramp (T_R), the fault can be step ($T_R \approx 0s$), soft ($T_R = 4s$) or hard ($T_R = 1s$) [11], [49].

In this research, the output of the gyro sensors are assumed to be in the range of $+10\ deg/s$ to $-10\ deg/s$. In case of the additive fault type, large and small magnitude faults are modeled using $A = 3\ deg/s$ and $A = 1.5\ deg/s$, respectively. In total, seven failure cases are considered, which can be summarized as follows:

1. Constant Bias
2. Hard Additive Large ($T_R = 1\ s$, $A = 3\ deg/s$)
3. Hard Additive Small ($T_R = 1\ s$, $A = 1.5\ deg/s$)
4. Soft Additive Large ($T_R = 4\ s$, $A = 3\ deg/s$)
5. Soft Additive Small ($T_R = 4\ s$, $A = 1.5\ deg/s$)
6. Step Additive Large ($T_R = 0\ s$, $A = 3\ deg/s$)
7. Step Additive Small ($T_R = 0\ s$, $A = 1.5\ deg/s$)

In the next subsection, the technique to generate the sensor residual is discussed.

C. Residual Generation Technique

As described earlier, the SFDIA scheme presented here uses residuals (d) to detect and identify sensor failures. Generally, residuals are generated by squaring the difference between the real sensor measurement and the measurement from its model [11]. This is as shown in (8). In (8), d is the residual

at time t where, x is the real sensor measurement and $\overline{x}$ is the estimator (model) measurement at time t.

$$d_t = (x_t - \overline{x}_t)^2 \tag{8}$$

Failure is detected when the residual d goes over a threshold τ. Ideally the sensor measurement and the estimator output must be equal, therefore generating a residual $d = 0$ and $d \neq 0$ in case of failure. When the residual d crosses τ, the failure alarm is triggered. In this ideal condition, τ should be kept close to 0 for quick detection ($\tau \approx 0$).

However, in a practical system, the sensor measurements are not equal to the estimator (model) output due to sensor noise and modeling inaccuracies. This means that the residual d is not equal to zero in fault free conditions. Due to this reason, in the absence of any faults, a false alarm ($F_A = 1$) could occur frequently when threshold $\tau \approx 0$. This could be resolved by raising the value of τ, however this risks the non detection of faults. Therefore, there is a need to have a balance between false alarms and fault detection.

In the SFDIA scheme proposed in this research, the residual d is generated using a sliding window mechanism [11]. In this mechanism (see Fig. 5), a window of size n data points keeps moving (sliding) with time. The window calculates the moving average of the n residuals generated using (8). The result of the sliding average window is then weighted to produce the current residual [11]. The residual generation mechanism can therefore be described as follows:

$$D_t = \frac{\varpi}{n} \sum_{i=t-n-1}^{t} (x_i - \overline{x}_i)^2 \tag{9}$$

where D is the residual at time instant t and ϖ is the weight. Notice how (8) is substituted in (9). The sliding average window filters the residuals using (8) from noise and modeling inaccuracies. The weight allows the magnification of the residuals and a high fault threshold τ. In this research, the size of the sliding window is set to 5 ($n = 5$) and the weight is set to 40 ($\varpi = 40$). For the pitch, roll and yaw rate sensors, the threshold is set to $\tau = 0.8$, $\tau = 0.8$ and $\tau = 0.2$, respectively.

In the next section, the results of the sensor failure detection and identification for accommodation experiments are presented.

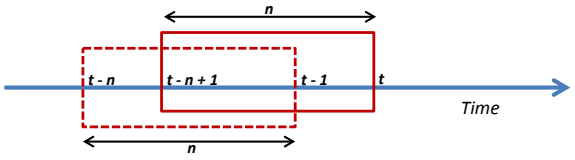


Fig. 5. Sliding average window at time t

VIII. Experimental Results

A. Yaw Sensor Failures

The results for the yaw rate sensor failure detection time are presented in Table V. Generally, large magnitude faults are quicker to detect, in comparison to the small magnitude faults. The greater the magnitude of the fault, the sooner the residual generated using (9) will cross the threshold τ. This observation is reflected in the results presented in Table V, which compares the results for a sudden step fault of large and small magnitude. On average, the step fault of large magnitude is detected instantaneously compared to an average of $0.8s$ in sample time for small magnitude step fault.

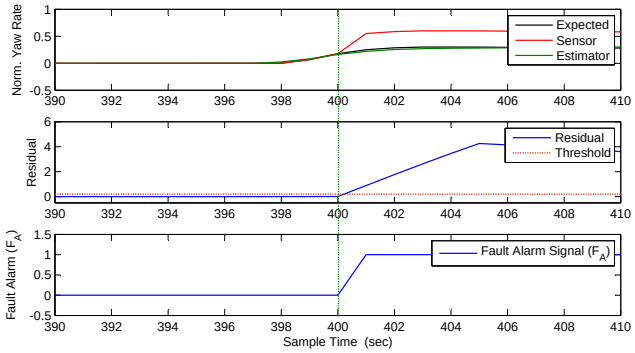


Fig. 6. Yaw sensor hard fault simulation of large magnitude. This result is from scenario 1 where the fault occurs at 400s.

Similar results can be observed with the hard additive type faults with a ramp duration of $T_R = 1s$ (Section VII-B). Although the detection time is affected by the magnitude of the fault, it is also affected by the transient phase (ramp duration T_R) of the developing fault. Due to this, the detection time for hard faults is greater than step type additive faults. In Fig. 6, the signals associated with the yaw rate sensor during the occurrence of hard fault of large magnitude is presented. Notice the fast response of the fault signal F_A after the occurrence of the fault. The time of fault is marked by a green line running across the three plots.

In comparison to the step and hard additive type faults, soft faults have the longest detection time. These faults have the highest ramp duration ($T_R = 4s$) amongst the three types of additive faults. On average, the detection time for soft faults of large magnitude is $2.6s$, in comparison to an average of $4s$ (sample time) for small magnitude fault. In the case of the constant bias fault, the average detection time is $1.6s$.

TABLE V
Yaw FDI Results

Detection Time for Fault Types in Sample Time							
Scn.	Bias	Hard		Step		Soft	
-	-	L	S	L	S	L	S
1	1	1	1	0	0	2	3
2	1	1	2	0	1	3	5
3	1	1	2	0	1	2	3
4	2	1	3	0	2	3	5
5	3	1	2	0	0	3	4
Avg.	1.6	1	2	0	0.8	2.6	4

−: No Fault Detected, Threshold (τ): 0.2, Scn: Scenario, L: Large, S: Small

B. Pitch Sensor Failures

The results for the pitch rate sensor failure detection time are presented in Table VI. The results reflect the observations made in the yaw rate sensor results. Large magnitude faults are quick to detect and additive faults with a ramp duration $T_R > 0s$ takes a longer time to detect. On average, the hard additive faults with large magnitude are detected in $1.4s$ sample time. In comparison, the hard faults with small magnitude are detected in average of $3s$ sample time.

Compared to the hard faults, the soft additive faults take an average of $3s$ and $5.2s$ in sample time, for large and small magnitude respectively. Notice that the average detection time of soft faults is longer compared to hard faults. This is because the ramp duration is greater for soft faults, which is set at $T_R = 4s$, instead of $T_R = 1s$ for hard faults. The step fault type has the lowest average of the additive fault types due to the zero ramp duration ($T_R = 0$). Step faults with small and large magnitude have an average of $0.8s$ and $2.6s$ respectively. The constant bias fault type has an average of $0.8s$.

In Fig. 7, the signals associated with the pitch rate sensor during the occurrence of step fault is presented. It shows the response of various signals during the occurrence of step fault of small magnitude. Notice how the residual gradually crosses the fault threshold τ and triggers the fault alarm F_A.

Comparing Table V and Table VI shows how the average detection time for the pitch rate sensor is greater compared to the yaw rate sensor, especially for the additive fault types. This is due to the higher fault residual threshold τ used for the pitch rate sensor. In comparison to the yaw rate sensor, the pitch rate estimator has a higher modeling error, therefore requiring a higher value for τ. The threshold τ is set to 0.8 for the pitch sensor whereas for the yaw sensor, $\tau = 0.2$.

TABLE VI
PITCH FDI REULTS

Detection Time for Fault Types in Sample Time							
Scn.	Bias	Hard		Step		Soft	
-	-	L	S	L	S	L	S
1	1	2	5	1	4	4	7
2	1	1	3	1	3	4	7
3	1	1	1	0	0	1	2
4	1	2	4	1	2	4	7
5	0	1	2	1	4	2	3
Avg.	0.8	1.4	3	0.8	2.6	3	5.2

– : No Fault Detected, Threshold (τ) = 0.8, Scn: Scenario, L : Large, S : Small

C. Roll Sensor Failures

In Table VII, the results for the roll rate sensor failure detection are presented. Similar to the pitch rate sensors, τ is set at a higher value: $\tau = 0.8$. This is to accommodate the difference between the estimator value and the sensor value.

The least detection time is taken by the constant bias fault type with an average of $1s$ in sample time. For hard fault types, the average is $2s$ and $4.6s$ in sample time for large and small magnitude respectively. As expected, the higher the residual threshold, the longer the fault detection time.

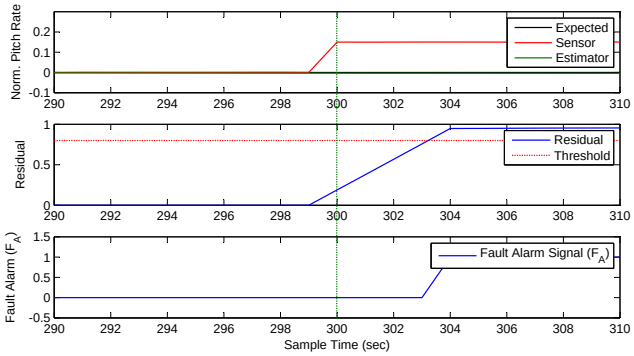


Fig. 7. Pitch sensor step fault simulation of small magnitude. This result is from scenario 1 where the fault occurs at 300s.

The soft fault types take the most amount of time to be detected. For large magnitude soft faults, the average detection time is $4s$ in sample time. The average detection time is even higher for small magnitude soft faults, with an average of $7s$ in sample time. These results are considerably higher compared to the detection time in the yaw rate sensor. The longer detection time is caused by the higher residual threshold.

In case of the step type faults, the average is at $1.2s$ in sample time for large magnitude. However for the small magnitude, the average is at $3.75s$ with a fault going undetected in scenario 1. The fault went undetected because the residual failed to trigger the threshold, as can be seen in Fig. 8. This could be solved by reducing the threshold τ, but risk false fault detection. Future work would consider additional inputs to the roll rate estimator to improve the estimate, and therefore improve the chances for detection.

TABLE VII
ROLL FDI RESULTS

Detection Time for Fault Types in Sample Time							
Scn.	Bias	Hard		Step		Soft	
-	-	L	S	L	S	L	S
1	1	2	5	2	–	4	7
2	1	2	4	1	4	4	7
3	1	2	4	1	3	4	7
4	1	2	5	1	4	4	7
5	1	2	5	1	4	4	7
Avg.	1	2	4.6	1.2	3.75	4	7

–: No Fault Detected, Threshold (τ): 0.8, Scn: Scenario, L: Large, S: Small

IX. RESULTS SUMMARY AND DISCUSSIONS

The FCC NN based SFDIA scheme is evaluated for failures in pitch, roll and yaw rate gyro sensors. Each sensor is manually injected with seven different faults at random locations on five different flight scenarios. The observations of the experiments can be summarized as follows:

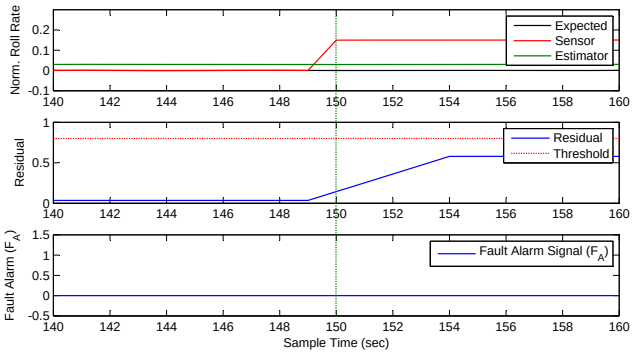


Fig. 8. Roll sensor step fault simulation of small magnitude. This result is from scenario 1 where the fault occurs at 150s.

- Sudden fault types like constant bias, hard additive, step additive are quicker to detect than faults that develop over time (e.g. soft additive faults).
- Faults with large magnitude are more easily detected than faults with small magnitude.
- Higher fault residual threshold to accommodate sensor estimator modeling errors and noise can increase the fault detection time.

These observations are consistent across the three gyro sensors. All faults were detected by the presented SFDIA scheme, except for one in roll rate sensor. This undetected fault is a step fault with small magnitude. In this case, the fault went undetected because the residual failed to trigger the threshold. However, out of the 105 failure cases evaluated, only one went undetected.

The FDI results presented here can be compared to the SFDIA scheme presented in [11]. The SFDIA scheme presented in [11] is based on the extended minimum resource allocating radial basis function (EMRAN-RBF) NN. With their SFDIA scheme, pitch rate faults were detected in $1.24s$, $1s$ and $1.86s$ for hard, step and soft faults respectively. In comparison, the SFDIA scheme presented here, detected the large magnitude ($A = 3\ deg/s$) faults in an average time of $1.4s$, $0.8s$ and $3s$ for hard, step and soft faults respectively.

The results are fairly comparable, except for the case of soft failure, where the presented SFDIA scheme took $1.14s$ longer. This difference in performance can be accounted for by the fact that the SFDIA scheme presented in [11] uses a sampling time of $20ms$, compared to the $1s$ sampling time used in the scheme presented here. The higher the sampling frequency, the quicker the faults are detected. Besides the sampling frequency, the SFDIA scheme presented here just uses 3 inputs compared to 4 inputs in [11].

One of the drawbacks of the presented SFDIA scheme is the fixed threshold based detection mechanism. Selecting a fixed fault threshold is a challenging task especially, in a dynamic system which is susceptible to noise and modeling inaccuracy. If the threshold is too high, the fault might take longer to be detected or worse, go undetected. Having a low threshold on the other hand might increase the rate of false alarms. The sliding averaging window mechanism does help reduce the effect of noise and modeling inaccuracy. However, if the dynamics of the system changes in the future, the thresholds would have to be evaluated and fixed again.

An alternative to the fixed threshold based detection mechanism is an adaptive threshold. In this mechanism, the fault threshold adapts to the changes in the system dynamics with time. Such a mechanism, as presented in the [7] and [51], would increase the robustness of the SFDIA scheme.

X. Conclusion

Sensors are an important part of any control system. A failure in a sensor could degrade the system's performance and can destabilize the system's operation. Therefore it is important for a system to have the ability to detect and accommodate sensor failures to maintain its reliability; especially in safety critical applications.

An aircraft can be considered as a safety critical system, where any failure can result in loss of life and significant damage to environment or property. This research investigates the development of a fault tolerant sensor system, with the aircraft as the example application.

In this paper, a neural network (NN) based sensor failure detection, identification and accommodation (SFDIA) scheme is presented. This scheme uses the fully connected cascade (FCC) NN architecture that was trained using the neuron by neuron (NBN) learning algorithm. As evident from Table I, this architecture is more efficient than the popular multi-layer perceptron (MLP) NN architecture. It requires less number of neurons to solve a problem compared to the MLP architecture. Therefore savings can be made in terms of computational expense, by using the FCC architecture instead of the MLP, for any NN based application.

The SFDIA scheme presented here addresses failures in the pitch, roll and yaw rate sensors. In total, seven sensor failure types are considered for each sensor. The FCC NN based sensor estimators can replicate a sensor's measurements with as little as two neurons; and out of the 105 failure experiments, only one fault went undetected.

Acknowledgements

This research is part-funded by the University of Central Lancashire (UCLan) and Military Air and Information (MAI), BAE Systems, UK. The authors thank Adam Bedford of UCLan and Mohiuddin Rahman of University of Glasgow, UK, for their advice with this research.

References

[1] E. Balaban, A. Saxena, P. Bansal, K. F. Goebel, and S. Curran, "Modeling, Detection, and Disambiguation of Sensor Faults for Aerospace Applications," *IEEE Sensors Journal*, vol. 9, no. 12, pp. 1907–1917, Dec. 2009.

[2] J. C. D. Silva, A. Saxena, E. Balaban, and K. Goebel, "A knowledge-based system approach for sensor fault modeling, detection and mitigation," *Expert Systems with Applications*, vol. 39, no. 12, pp. 10 977–10 989, Sep. 2012.

[3] I. Samy, I. Postlethwaite, and D. Gu, "Neural network based sensor validation scheme demonstrated on an unmanned air vehicle (UAV) model," in *2008 47th IEEE Conf. Decision and Control*. IEEE, 2008, pp. 1237–1242.

[4] G. Campa, M. Fravolini, M. Napolitano, and B. Seanor, "Neural networks-based sensor validation for the flight control system of a B777 research model," in *Proc. Amer. Control Conf.*, vol. 1. American Automatic Control Council, 2002, pp. 412–417.

[5] Q. Wang, F. Yang, Q. Ge, and Q. Yang, "A sensor network modeling and fault detection method for large wind farms by using neural networks," in *Proc. 11th IEEE Int. Conf. Control & Autom.* IEEE, 2014, pp. 308–313.

[6] Z. F. Wang, J.-L. Zarader, and S. Argentieri, "Aircraft fault diagnosis and decision system based on improved artificial neural networks," *2012 IEEE/ASME Int. Conf. Adv. Intelligent Mechatronics (AIM)*, pp. 1123–1128, Jul. 2012.

[7] M. Mrugalski, "An unscented Kalman filter in designing dynamic GMDH neural networks for robust fault detection," *Int. J. Applied Mathematics and Computer Sci.*, vol. 23, no. 1, pp. 157–169, Jan. 2013.

[8] S. Rajendran, U. Govindarajan, S. Senthilvadivelu, and S. Uandai, "Intelligent sensor fault-tolerant control for variable speed wind electrical systems," *IET Power Elec.*, vol. 6, no. 7, pp. 1308–1319, Aug. 2013.

[9] D. Gastaldello, A. Souza, C. Ramos, P. da Costa Junior, and M. Zago, "Fault location in underground systems using artificial neural networks and PSCAD/EMTDC," in *2012 IEEE 16th Int. Conf. Intelligent Engineering Systems (INES)*. IEEE, Jun. 2012, pp. 423–427.

[10] M. R. Napolitano, Y. An, and B. A. Seanor, "A fault tolerant flight control system for sensor and actuator failures using neural networks," *Aircraft Design*, vol. 3, no. 2, pp. 103–128, Jun. 2000.

[11] I. Samy, I. Postlethwaite, and D.-W. Gu, "Survey and application of sensor fault detection and isolation schemes," *Control Engineering Practice*, vol. 19, no. 7, pp. 658–674, Jul. 2011.

[12] M. Meireles, P. Almeida, and M. Simoes, "A comprehensive review for industrial applicability of artificial neural networks," *IEEE Trans. Ind. Electron.*, vol. 50, no. 3, pp. 585–601, Jun. 2003.

[13] B. M. Wilamowski, "How to not get frustrated with neural networks," *2011 IEEE Int. Conf. Industrial Technology*, pp. 5–11, Mar. 2011.

[14] B. Wilamowski, N. Cotton, O. Kaynak, and G. Dundar, "Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks," *IEEE Trans. Ind. Electron.*, vol. 55, no. 10, pp. 3784–3790, Oct. 2008.

[15] D. Hunter, H. Yu, and M. Pukish, "Selection of Proper Neural Network Sizes and Architectures: A Comparative Study," *IEEE Trans. Ind. Informatics*, vol. 8, no. 2, pp. 228–240, May 2012.

[16] B. M. Wilamowski, "Challenges in applications of computational intelligence in industrial electronics," in *2010 IEEE Int. Symposium on Ind. Electron. (ISIE)*. IEEE, Jul. 2010, pp. 15–22.

[17] B. Wilamowski, "Neural network architectures and learning algorithms," *IEEE Ind. Electron. Magazine*, vol. 3, no. 4, pp. 56–63, Dec. 2009.

[18] S. Hussain, M. Mokhtar, and J. M. Howe, "Aircraft sensor estimation for fault tolerant flight control system using fully connected cascade neural network," in *Proc. Int. Joint Conf. Neural Netw.* IEEE, 2013, pp. 1–8.

[19] P. Goupil, "AIRBUS state of the art and practices on FDI and FTC in flight control system," *Control Engineering Practice*, vol. 19, no. 6, pp. 524–539, Jun. 2011.

[20] Y. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *Ann. Rev. in Control*, vol. 32, no. 2, pp. 229–252, Dec. 2008.

[21] R. Isermann and P. Ballé, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control Engineering Practice*, vol. 5, no. 5, pp. 709–719, May 1997.

[22] G. Campa, M. L. Fravolini, B. Seanor, M. R. Napolitano, D. D. Gobbo, G. Yu, and S. Gururajan, "On-line learning neural networks for sensor validation for the flight control system of a B777 research scale model," *Int. J. Robust and Nonlinear Control*, vol. 12, no. 11, pp. 987–1007, Sep. 2002.

[23] Y. An, "A design of fault tolerant flight control systems for sensor and actuator failures using on-line learning neural network," PhD Thesis, West Virginia University, US, 1998.

[24] I. Samy, I. Postlethwaite, and D.-W. Gu, "Detection and accommodation of sensor faults in UAVs- a comparison of NN and EKF based approaches," in *49th IEEE Conf. Decision and Control (CDC)*. IEEE, Dec. 2010, pp. 4365–4372.

[25] S. Julier and J. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," in *Proc. SPIE 3068, Signal Processing, Sensor Fusion, and Target Recognition VI*. SPIE, 1997.

[26] G. Heredia and A. Ollero, "Sensor fault detection in small autonomous helicopters using observer/Kalman filter identification," in *2009 IEEE Int. Conf. Mechatronics*, vol. 00, no. April. IEEE, 2009, pp. 1–6.

[27] C. Hajiyev and H. E. Soken, "Robust Estimation of UAV Dynamics in the Presence of Measurement Faults," *J. Aerospace Engineering*, vol. 25, no. 1, pp. 80–89, Jan. 2012.

[28] S. Kim, J. Choi, and Y. Kim, "Fault detection and diagnosis of aircraft actuators using fuzzy-tuning IMM filter," *IEEE Trans. Aerospace and Electronic Systems*, vol. 44, no. 3, pp. 940–952, Jul. 2008.

[29] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A Survey of Fault Detection, Isolation, and Reconfiguration Methods," *IEEE Trans. Control Systems Technology*, vol. 18, no. 3, pp. 636–653, May 2010.

[30] G. H. B. Foo, Z. Xinan, and D. M. Vilathgamuwa, "A Sensor Fault Detection and Isolation Method in Interior Permanent-Magnet Synchronous Motor Drives Based on an Extended Kalman Filter," *IEEE Trans. Ind. Electron.*, vol. 60, no. 8, pp. 3485–3495, Aug. 2013.

[31] H. Guo-jian, L. Gui-xiong, C. Geng-xin, and C. Tie-qun, "Self-recovery method based on auto-associative neural network for intelligent sensors," in *Proc. 8th World World Congr. Intell. Control Autom.*, no. 2007. IEEE, Jul. 2010, pp. 6918–6922.

[32] S. Gururajan, M. L. Fravolini, H. Chao, M. Rhudy, and M. R. Napolitano, "Performance evaluation of neural network based approaches for airspeed Sensor Failure Accommodation on a small UAV," in *21st Mediterranean Conference on Control and Automation*. IEEE, Jun. 2013, pp. 603–608.

[33] T.-H. Guo and J. Musgrave, "Neural network based sensor validation for reusable rocket engines," in *Proc. Amer. Control Conf.*, vol. 2. American Autom Control Council, 1995, pp. 1367–1372.

[34] S. Toma, L. Capocchi, and G.-A. Capolino, "Wound-Rotor Induction Generator Inter-Turn Short-Circuits Diagnosis Using a New Digital Neural Network," *IEEE Trans. Ind. Electron.*, vol. 60, no. 9, pp. 4043–4052, Sep. 2013.

[35] H. Yu and W. Auburn, "Fast and efficient and training of neural networks," in *3rd Int. Conf. Human System Interaction*. IEEE, May 2010, pp. 175–181.

[36] B. M. Wilamowski, N. Cotton, J. Hewlett, and O. Kaynak, "Neural Network Trainer with Second Order Learning Algorithms," in *2007 Int. Conf. Intelligent Engineering Systems*. IEEE, Jun. 2007, pp. 127–132.

[37] B. M. Wilamowski, "C++ implementation of neural networks trainer," in *2009 Int. Conf. Intelligent Engineering Systems*. IEEE, Apr. 2009, pp. 257–262.

[38] ——, "Advanced learning algorithms," in *2009 Int. Conf. Intelligent Engineering Systems*. IEEE, Apr. 2009, pp. 9–17.

[39] B. Wilamowski, "Understanding Neural Networks," in *Intelligent Systems*, ser. Electrical Engineering Handbook. CRC Press, Feb. 2011, pp. 1–11.

[40] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks: The state of the art," *Int. J. Forecasting*, vol. 14, no. 1, pp. 35–62, Mar. 1998.

[41] Laminar Research, "X-Plane 10 Global — The World's Most Advanced Flight Simulator — X-Plane.com." [Online]. Available: http://www.x-plane.com

[42] ——, "FAA-Certified X-Plane." [Online]. Available: http://www.x-plane.com/pro/certified/

[43] ——, "X-Plane 10 Manual," 2012. [Online]. Available: http://www.x-plane.com/files/manuals/X-Plane_10_Desktop_manual.pdf

[44] R. Collinson, "Navigation Systems," in *Introduction to Avionics Systems*. Dordrecht: Springer Netherlands, 2011, pp. 303–376.

[45] R. P. G. Collinson, "Inertial Sensors and Attitude Derivation," in *Introduction to Avionics Systems*. Springer Netherlands, 2011, pp. 255–302.

[46] D.-M. Ma, J.-K. Shiau, I.-C. Wang, and Y.-H. Lin, "Attitude determination using a MEMS-based flight information measurement unit." *Sensors*, vol. 12, no. 1, pp. 1–23, Jan. 2012.

[47] R. Collinson, "Aerodynamics and Aircraft Control," in *Introduction to Avionics Systems*. Dordrecht: Springer Netherlands, 2011, pp. 101–117.

[48] A. Kozionov, M. Kalinkin, A. Natekin, and A. Loginov, "Wavelet-based sensor validation: Differentiating abrupt sensor faults from system dynamics," in *2011 IEEE 7th Int. Symposium on Intelligent Signal Processing*. IEEE, Sep. 2011, pp. 1–5.

[49] I. Samy, "Development and Evaluation of Neural Network Models For Cost Reduction in Unmanned Air Vehicles," PhD Thesis, University of Leicester, UK, May 2009.

[50] G. Heredia, a. Ollero, M. Bejar, and R. Mahtani, "Sensor and actuator fault detection in small autonomous helicopters," *Mechatronics*, vol. 18, no. 2, pp. 90–99, Mar. 2008.

[51] M. Perhinschi, M. Napolitano, G. Campa, B. Seanor, J. Burken, and R. Larson, "An adaptive threshold approach for the design of an actuator failure detection and identification scheme," *IEEE Trans. Control Systems Technology*, vol. 14, no. 3, pp. 519–525, May 2006.

"You have no idea how strong you are."
– Dr. Banu Abdallah

"The best way out is always through."

– Robert Frost