UNIVERSITÀ DEGLI STUDI DI NAPOLI “FEDERICO II”

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

Corso di Dottorato in Ingegneria Aerospaziale, Navale e della Qualità
XXVI Ciclo

Innovative Virtual Air Data Sensors: Algorithms and Flight Test Results

Anno 2015

Autore:	Luca Garbarino
Tutor Aziendale:	Dr. Federico Corraro
Tutor Universitario:	Prof. Giancarmine Fasano
Tutor Universitario:	Prof. Domenico Accardo
Coordinatore:	Prof. Luigi De Luca

To my family and friends

ACKNOWLEDGEMENTS

First of all I would like to express my gratitude to my academic tutors Prof. Domenico Accardo and Prof. Giancarmine Fasano and to my corporate tutor Dr. Federico Corraro for giving me the opportunity to grow professionally and humanely with this experience in a Phd course at the University of Naples supported by CIRA (Italian Aerospace Research Center) with the hope that the interesting activities started in the last four years will continue in the future.

All the scientific work carried out during these years has been possible only thanks to my colleagues involved in the On-Board and ATM Systems of CIRA, “Guidance, Navigation and Control” and “Flight Dynamics and Simulations” laboratories. More specifically I am grateful to the CIRA TECVOL project team because only with the inventive and high proficiency of their members was it possible to reach so many interesting research results validated through very challenging flight campaigns. In particular my specific acknowledgements are for the aforementioned Federico Corraro and for Gaetano Zazzaro, Ignazio Dimino, Nicola Genito, Edoardo Filippone, Ettore De Lellis and Antonio Vitale, for their passion in the research activities in which we worked together.

To my family, my mother Paola and my brother Davide, for having always been confidence in my abilities.

A very high thought to my father Luigi that I hope is proud of man that I have become even if isn’t near me.

Finally to him, the most important person in my life, my son Alessio, for the great strength and joy that knows infuse in every day of my life. To him I dedicate this work, as well as all my efforts.

TABLE OF CONTENTS

1 PREFACE

1.1 Abstract

This thesis deals with the design, prototype implementation and the assessment of virtual sensors for an Air Data System (ADS). The needs for the development of a virtual Air Data Sensors resides on two relevant aspects in aviation transport development: a) the opportunity to improve the safety of manned aviation, by implementing an affordable solution for ADS redundancy; b) the possibility to improve the reliability of unmanned air vehicles (UAVs), which can support their integration in non-segregated airspace.

Virtual sensors are normally considered as a relevant component of Fault Detection and Isolation approaches to the Fault Tolerant Control Theory, briefly introduced in the thesis.

Virtual sensors for both the Angle-of-Attack and the Indicated Air Speed have been developed and completely assessed in CIRA Laboratory test bed for Real-Time Simulation with Hardware and Pilot-in-the-Loop.

In some cases, the assessment has also been achieved in flight test, on the FLARE vehicle which CIRA operates specifically for the execution of flight test of prototype technologies.

The thesis first analyses and describe the basic methodologies for the virtual sensors design, identifying those better suitable for application to air data measures. In particular, both the model-based technique and the implementation of ADS Virtual Sensors through Artificial Neural Networks are described. Special attention is given to the data selection techniques aimed at network training and assessment, focusing the machine learning approach to this aspects.

The study case applied to the TECNAM P92 VLA vehicle is described in detail, posing attention to the description of both the laboratory set-up realization and to the specific upgrades which have been required to the TECNAM P92 vehicle for the in-flight tests of the developed virtual sensors.

The assessment of both the model-based Virtual Sensors and the ANN Virtual Sensors is finally documented.

1.2 List of Abbreviations

ADC	Air Data Computer
ADS	Air Data System
ANN	Artificial Neural Network
APB	Automatic Program Building
AOA	Angle Of Attack
AOS	Angle Of Sideslip
BR	Byzantine-Resilient
CRISP-DM	Cross-Industry Consortium Standard Process for Data Mining
DM	Data Mining
DOS	Dedicated Observer Scheme
EKF	Extended Kalman Filter
FCC	Flight Control Computer
FCRs	Fault Containment Regions
FBW	Fly-by-wire
FDD	Fault Detection and Diagnosis
FDIR	Fault Detection, Identification and Recovery
FIS	Fazzy Inference System
FLARE	Flying LAboratory Research
FSS	Full Scale Span
GLR	Generalized Likelihood Ratio
GOS	Generalized Observer Scheme
GPS	Global Positioning System
HIL	Hardware In the Loop
IAS	Indicated Air Speed
MM	Multiple Model (MM)
nSMR	n-Safe Modular Redundant
KDD	Knowledge Discovery in Database
KF	Kalman Filter
PCA	Principal Component Analysis
PEC	Position Error Correction
PWM	Pulse Width Modulation
RNNs	Recurrent Neural Networks
RSS	Residual Sum of Squares

RTCA	Radio Technical Commission for Aeronautics
SVD	Singular Value Decomposition
TAS	True Air Speed
TMR	Triple Modular Redundancy
UAV	Unmanned Aerial Vehicle
UKF	Unscented Kalman Filter

1.3 List of Symbols

e_j	Error signal at the output of neuron j
h	density altitude
J	Cost function
M	Mach number.
N_x, N_y, N_z	Body axis load factor
p, q, r	Body axis angular rate components
P_S	Static pressure
P_t	Total pressure
T_S	Static temperature
T_t	Total temperature
VS	Vertical Speed
V_x, V_y, V_z,	Velocity components
w_{km}	M-th synaptic weights of neuron k
α	Angle of attack
β	Sideslip angle
δ	Local gradient
δ_F	Flap surface extension
δ_S	Stabilator surface extension
η	Learning-rate parameter of the back-propagation algorithm
v	Induced local field
ρ	Air density
ρ_0	See level ISA density
σ	Standard deviation

φ	Activation function

2 INTRODUCTION

Today's air transportation system is an integral part of the global economies. It is the primary mechanism for connecting countries across the world through mobility of people, goods and services. Aviation accounts for more than $1.5 trillion annually of total US economic activity and is one of the few industries that generates a positive trade balance; $75.1 billion in 2013, alone. The aviation industry supports more than 11.8 million direct and indirect jobs, including more than one million high-quality manufacturing jobs.

The overarching impacts of aviation and the air transportation system can be felt right down to the individual; just about every product produced and purchased today has been touched by aviation in some way. Aircraft transport 17.7 billion tons of freight every year. You may not have flown today but something you needed did. Any such huge system, so relevant for the global worldwide economy, cannot avoid to invest large amount of resources in research activities addressed to innovate and improve itself. Research in the aeronautic field have already increased the capacity and improved the efficiency, safety, and environmental compatibility of the air transportation system. NASA continues to explore research and develop tools and technologies that can be integrated into even more advanced and efficient aircraft and airspace systems, including enabling game-changing concepts for the future.

As NASA do in USA, also in Europe several Aeronautical institutions (ACARE; EREA; European Commission) developed specific reports [B1],[B2],[B3], which try to design the future of the air transportation and the technologies which could support that future to become a reality.

A higher level of automation is a constant topic which appears in all those air transport perspectives.

In order to pursue general recognized goals, such as the improvement of the air transport efficiency, reduce its environmental impact, improve safety levels, in fact, higher automation level has been identified, together with engines and materials innovation.

Also more revolutionarily in the field of aeronautics, research in recent years has focused on UAVs, Unmanned Aerial Vehicles. This term is defined for the special class of vehicles that fly without the aid of a pilot on board. But this simple and intuitive definition cannot be exhaustive since the acronym UAV encompasses a wide variety of different systems, so a univocal and concise definition of UAVs is not applicable.

We can affirm that UAVs are powered reusable aircrafts without crew, which can be remotely piloted or can fly autonomously or semi-autonomously. They can board a large variety of payloads and execute a large variety of specific tasks, for a certain period of time, determined by their mission and within or beyond the Earth's atmosphere.

This inclusive definition helps to distinguish clearly UAVs from missiles that a superficial first analysis could associate. In fact, although also missiles do not have human operator on board and usually are remotely piloted, they are not a "means" to carry a payload but they are the load themselves. Moreover a missile cannot be reused.

The great importance of these vehicles and their growing use in various fields is mainly due to the absence of human operators on board, that is a very attractive feature in civil application and in military ones, since it allows to perform very risky missions without risking to lose human life and to execute lingering and boring routine tasks implying an unsustainable workload and stress to a human pilot.

Starting by the years 1980/1990, with the miniaturization and the development of applicable technologies, interest in the use of UAVs in the U.S. armed forces has grown significantly since they are considered the fighting machines cheaper and more capable compatibly with the least loss of life. By means of numerous flight experiments it began the systematic use of these vehicles by military forces in operations of such extreme delicacy: the war in the Balkans, the Gulf war, the war in Afghanistan and Pakistan, the war in Gaza, and more recently the war in Libya.

In order to increase safety, autonomy and reliability of UAVs many research projects have been launched around the world [B4]. In the U.S., for example, projects have been launched for the development of UAVs prototype more autonomy for carrying out flight missions that include more complex coordination between manned vehicles, UAVs and other military ground forces. Some of these prototypes are part of the class UCAV (Unmanned Combat Air Vehicle) as the Boeing X45, others are of the class of HALE (High Altitude, Long Endurance) and others of the class Mini and Micro UAVs (MAV) with sizes comparable with a mosquito.

Despite the global leaders in the UAVs business are currently the United States and Israel, whose vehicles are engaged in missions of reconnaissance and combat operations, even Europe is investing heavily in this market and there are many national and international projects under way. The Italian industry has a long tradition in using drones for reconnaissance and combat exercises such as the series of Mirach Meteor, the Meteor CAE (now Galileo Avionics founded in 1974) pioneered firstly target aircrafts and then aircrafts suitable to observe the battlefield. Besides also Alenia Aeronautica started the development of the last generation MALE (Medium Altitude, Long Endurance) UAVs with prototypes such as Sky X and Sky Y.

Large industrial and commercial implications of UAVs and thus their evolution require large investments in research with the aim to achieve in the close future the possibility of easily use them for civil applications. Currently, however, civilian applications are very less widespread than military ones, in fact, UAVs could be a valid alternative to human missions only in specific cases because they are not still so safe and reliable to allow them for a convenient use in civil airspace where the rules dictated by the aviation authorities, are applicable, at the moment, only to aircraft with crew. However, there are several ongoing research and experimentation in order to improve the reliability of UAVs and define the rules to allow UAVs to fly in civil airspace (for example the project INOUI - Innovative Operational UAS Integration).

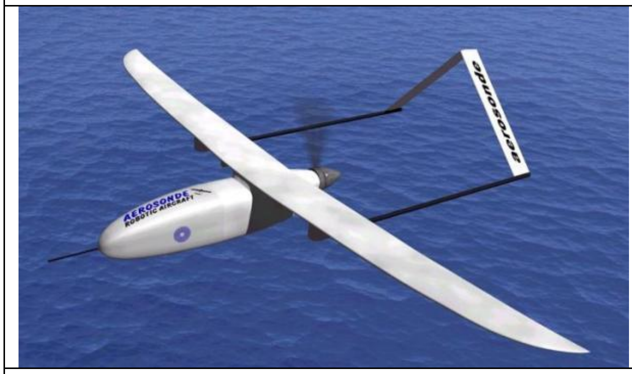
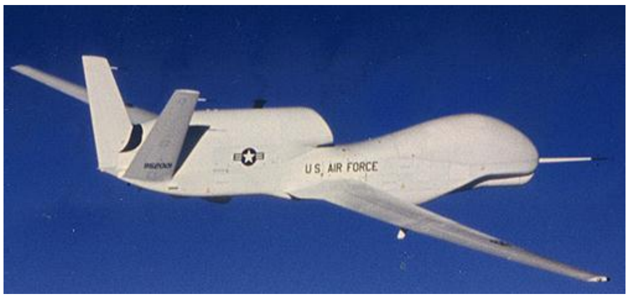
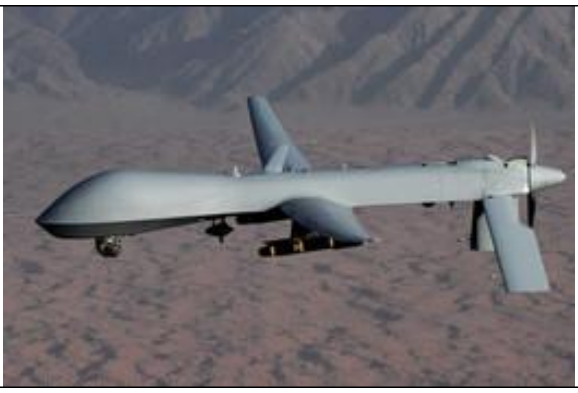
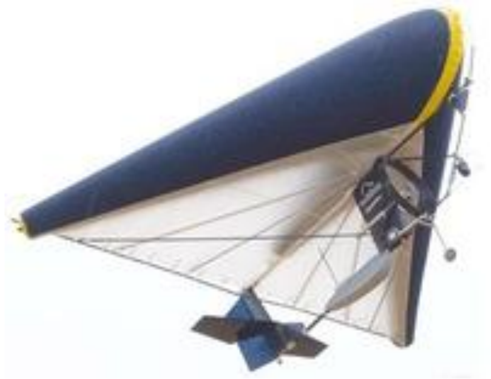
Aerosonde	Global Hawk
	
Predator	Nimbus UAV metaplane
	

Figure 2-1 – Some of current UAV models

Regarding the use of UAVs for not-military purposes some examples can be to use them as a tool for search and rescue, to find men missing in the desert, trapped in collapsed buildings, or lost in open sea. They can be used for surveillance of vast areas with low-cost systems, for example for the monitoring of numerous herds, for maintenance and security inspection of power lines, oil and gas pipelines or forest fire surveillance.

A growing use of UAVs has started also in civil application related to operation in areas too risky for manned aircraft: the NOAA (National Oceanic and Atmospheric Administration) began to use the unmanned system Aerosonde with the aim to hunt hurricanes. Aerosonde can fly into a hurricane and communicate near real-time data directly to the National hurricane Center in Florida, giving measurements very precise and detailed.

More recently, American UAV Global Hawk flew over the nuclear plant in Fukushima Dai-Chi in Japan, penetrating into the no-gone zone with the aim to monitor the reactors after the explosion caused in 2011 by the famous earthquake and taking also photos with infrared sensors. The high radioactivity would have been made it impossible for human operators.

A last important use of UAVs is that related to the border (on earth and on sea) control: the use of UAVs in the surveillance on Mediterranean Sea to control and support hundreds of persons immigrants and refugees from the northern coast of Africa towards Europe and specifically towards Italy.

One of the key challenges that shall be faced by the research community relates to UAVs flying autonomously in an “Unstructured Environment”. “Autonomy” here refers to the absence of human intervention, and “unstructured environment” is associated with uncertainty both in the outside world (meteorological conditions, air traffic, fixed and moving obstacles) and in the vehicle subsystems (failures). Considering that the flight envelope of an airplane indicates the regions that an airplane can safely fly without any undue risk and with an acceptable margin of safety a main objective of aerospace research is, therefore, to employ many innovative techniques, such as enhanced vision systems, to expand it.

Figure 2-2 – Aircraft landing in presence of strong crosswind

With reference to both the improvement of automation in manned vehicles and in increase of autonomy in unmanned vehicles, there are two main pathways along which research activities currently advance: the first is the development of advanced Flight Management Systems, embedding a growing number of functions which alleviate the intervention of the pilots in repetitive operations, moving his role from "operator" towards the role of supervisor of a flight. The second research topic is the improvement of the reliability of the on-board systems, that could be obtained also by improving the ability to preserve their capability of functioning also under degraded and failure conditions: Fault Tolerant Control Systems (FTCS) (see [B4] for a bibliographical review of such FTCS) is the specific name given to the health management of the Flight Control System/Flight Management Systems.

In order to assure the integrity of modern fly-by-wire flight control systems (FBW FCSs), replication of equipment and supporting systems is required. This is partly because the fault detection and isolation (FDI) is based on majority voting of like signals. Cost effective benefits are therefore expected form more integrated FDI algorithms [B5]. The FDI is a technique already used in the conventional approach to FDI, as implemented and certified on most modern aircraft [B5].

A relevant component of the FDI techniques is represented by redundancies of sensors. Normally, this redundancy is obtained with the hardware replications of sensors but, due to cost and weight reduction considerations and, in some cases, also for the impossibility to replicate sensors installations, the redundancy of the sensors is obtained through software implementation, and this software replication of equipment is identified as "virtual sensor".

This technology is particularly suitable for the replication of the Air Data Sensors. The ADS measures are actually relevant in safety of flight, both in commercial jetliners and also for General Aviation aircrafts.

Data on aviation accidents in the last decades confirms (Figure 2-3, Figure 2-4) that unusual attitudes, often due or linked to unavailable or incorrect air data measures, remain one of the most relevant cause for fatal accidents.

Furthermore, the analysis of some specific accident also highlighted that virtual sensor measures could represent an additional safety means in some critical conditions, also with respect to the common hardware redundancy.

In the accident of the Air France Flight 447 occurred on 2009, June 1st, which accounted for 228 fatalities, the primary cause has been identified in the contemporary obstruction of all the pitot tubes due to ice accretion on the sensors. The availability of a "virtual" measure of the relevant air data parameters could have provided some referenced measures which could have preserved the flyiability of the vehicle.

The more advanced and assessed “Virtual sensors” techniques are so-called “model-based” techniques, and some practical application of this kind of “virtual sensor” will be also presented in the thesis. Actually, the most challenging innovation presented in the thesis will be the realization of ADS virtual sensors though the use of the Artificial Neural Networks (ANN) approach.

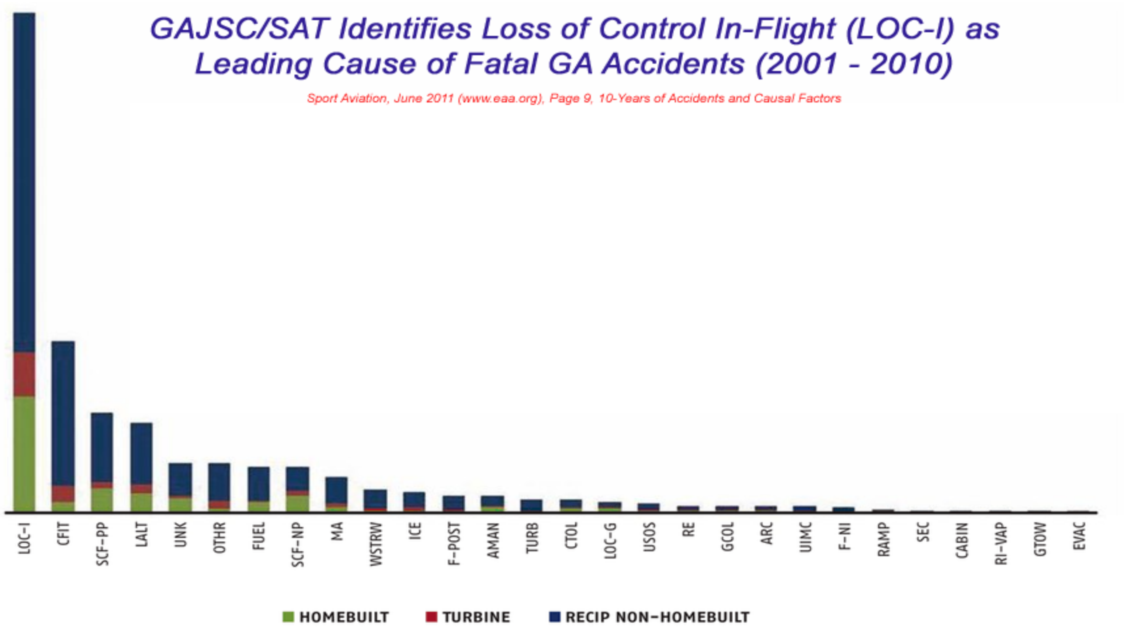


Figure 2-3 – First decade General Aviation fatal accidents causes chart.

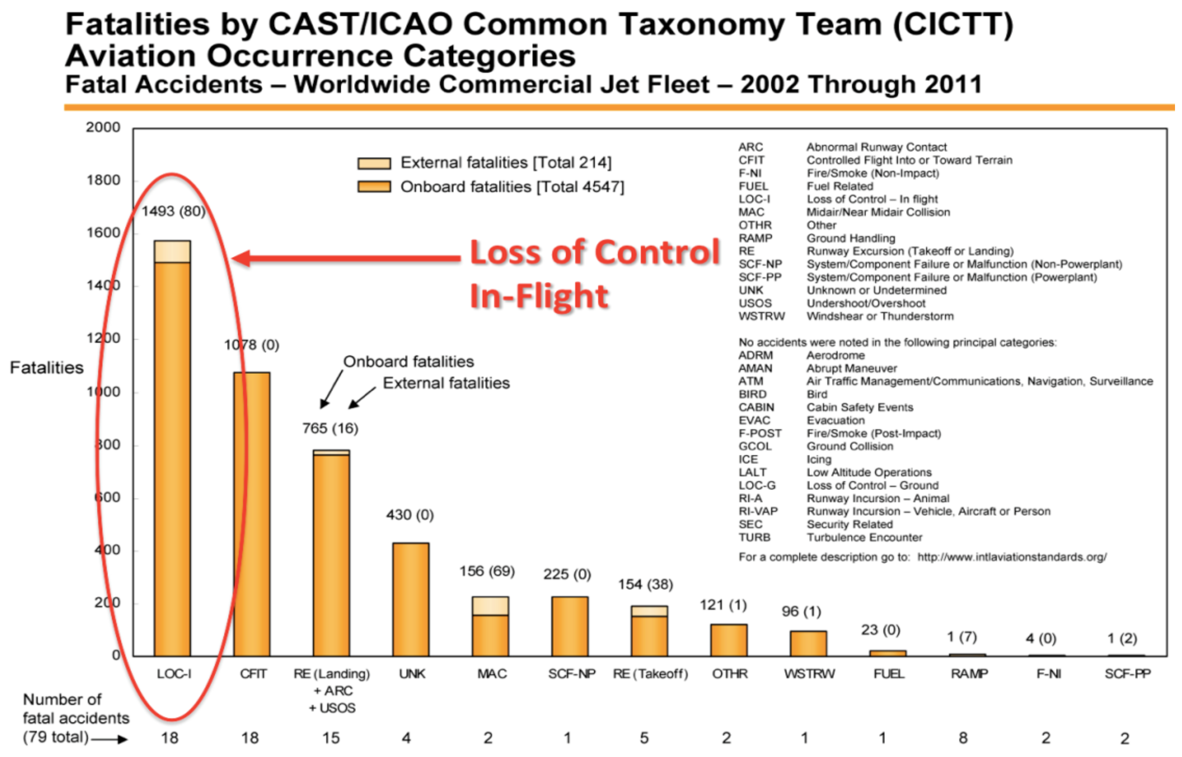


Figure 2-4– First decade Jet Airliner fatal accidents causes chart.

This thesis deals with the development of virtual sensors, as part of the FDI techniques, in two innovative aspects:

- It proposes some innovative approach to the development of a virtual sensor, in which the effective measure is obtained by using also dissimilar sensors, that is by using measures of other sensed parameters to re-build a relevant measure of a failed sensor;
- It applies to Unmanned Aerial Vehicles, in order to improve their autonomy, safety, and reliability levels, which can in turn support a wider use of such UAVs also in civil applications.

It has finally to be noticed that the activities here documented span from the design of such Virtual Sensors to their prototype realization and validation. A Rapid-Prototyping approach has been applied to the Virtual Sensors development. The Rapid Prototyping used allows to assure the successful application to the whole design process of the “V-Cycle” approach, including the real-time with hardware-in-the-loop validation of the system by means of the properly designed on ground-validation test rig, which includes in the real-time with hardware-in-the-loop simulation also the human-machine interfaces, in addition to the aircraft and external world simulator and Flight Control Computer (FCC). In this way, the laboratory test rig allowed to perform a complete simulation of the real in-flight mission, including the human factor too. The flight demonstrations, also performed for some of the developed techniques, completed the application of the V-Cycle approach and the validation of the Virtual Sensors prototype.

The thesis is organized as follows. Chapter 3 provides an overview of the current state-of-the-art on all the types of algorithms involved in estimating air data. Chapter 4 includes a detailed description of the proposed innovative virtual sensor algorithms for implementing an Air Data Computer. Chapter 5 describes the development and verification tools. Chapter 6 contains a summary of laboratory and flight results and an overview of the development process followed. Chapter 7 presents some concluding remarks.

3 SURVEY ON VIRTUAL SENSORS ALGORITHMS

3.1 Definition and fields of application

Virtual Sensors ([B6]) (also known as soft sensors) are software modules (Figure 3-1) which utilize measurable signals (Virtual Sensor inputs) in order to reconstruct a signal of interest (Virtual Sensor output) not directly available, due to failures occurred to specific measurement equipment or to the total lack of such kind of equipment. Although Virtual Sensors may often be developed based upon mathematical models obtained directly from the physics of the system and first principles, in many cases such mathematical models are unavailable, or the exact values of the parameter characterizing the model are unknown, or they are too complicated to be practically usable. For this reason the development of Virtual Sensors often has to be based upon system identification techniques. There exist in the literature some works that propose nonlinear Virtual Sensors in aerospace applications ([B6],[B7],[B8]); they have primarily been applied to implement FDI techniques to navigation sensors.

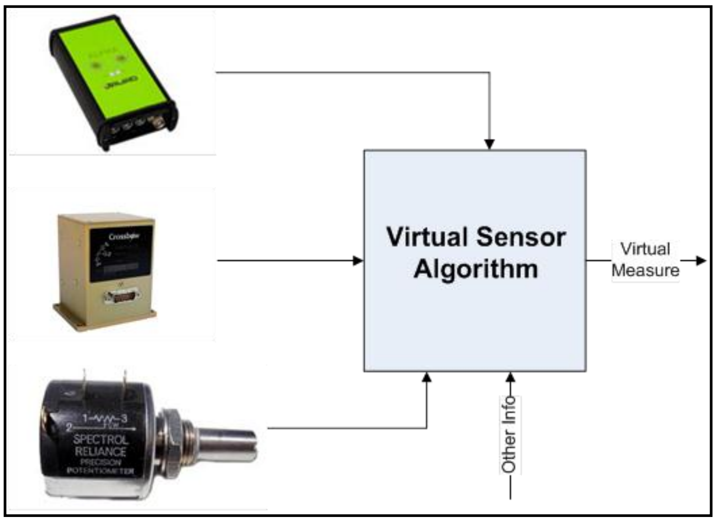
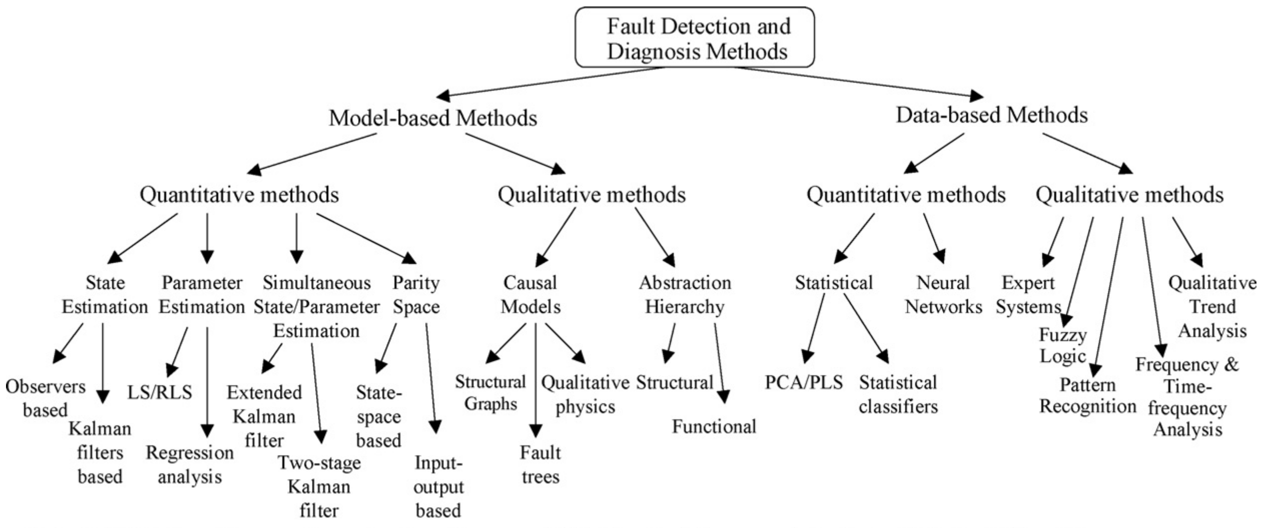


Figure 3-1 - Virtual Sensor Conceptual Architecture.

The availability of redundant sensors (physical and analytical) allows implementing Fault Detection, Identification and Recovery (FDIR) techniques in order to improve system’s reliability and safety, which have always been main issues in manned and Unmanned Aerial Vehicle UAVs [B9]. Indeed, these FDIR methods try to diagnose faults using the redundancy of some mathematical description of the system dynamics. Virtual sensors, as pointed out here above, may typically be used to realize measurements redundancy (analytical redundancy) without introducing multiple physical sensors, thus reducing costs and vehicle's weight. As such, Virtual Sensor methods are a significant part of the FDIR techniques.

In the literature, many applications of analytical redundancy for FDIR in flight control systems are reported. The use of virtual sensors in aerospace applications has not been widely investigated yet, although some applications to unmanned aircraft, either fixed wing UAVs ([B10]) or helicopter UAVs ([B11],[B12],[B13]) are reported in the literature.

A classification of the different techniques for FDIR, are shown in Figure 3-2 and a survey of these different approaches are described in [B4][1].



Note: LS/RLS: Least Squares/Recursive Least Squares; PCA: Principal Component Analysis; PLS: Partial Least Squares.

Figure 3-2: Fault Detection and Diagnosis Methods ([B4])

The existing FDD approaches can be generally classified into two categories:

- model-based,
- data-based (model-free) schemes;

these two schemes can further be classified as quantitative and qualitative approaches. Essentially, a quantitative model-based FDD scheme utilizes mathematical model to carry out FDD in real-time. Most frequently Model-Based Techniques applied are observer-based techniques [B14], [B15], [B16], parity-space methods [B17], and parameter estimation schemes [B18].

[1] Note that in references the FDIR techniques are often referred as Fault Detection and Diagnosis (FDD) techniques.

Data-based methods make instead normally use of large data recordings of input-output relationships of an unknown or un-modelled physical system, that is a black-box whose behavior is represented by means of non-linear complex approximated functions, fine-tuned by using the available input-output measures. The most input-output data are available, in terms of both quantity and extension of the covered envelope, the more precise will be the approximations. Data-based techniques are typically based on statistical methods, but more and more often "soft computational" methods are being used, and notably for the aim of this thesis, the Neural Network approach will be described in the following.

3.2 Model based Techniques

3.2.1 Observers and Kalman based methods

The solution of the problem of FDD in redundancy addressed using analytical methods of observers or Kalman filters can be divided into two logical steps:

- Residuals generation through appropriate filters.
- Residuals analysis for the generation of alarms (decision making).

The residuals have to be as much insensitive as possible to noise, disturbances and uncertainties and very sensitive to possible fault. The use of several residuals can allow, in some cases, even the isolation of a fault. The residual analysis for the detection and isolation is typically done through suitable technics or statistical thresholds. The observers output is used for the generation of residuals that are related to the error of estimate (difference between estimated output and measured output):

(Eq.1) $$r(t) = y(t) - \hat{y}(t)$$

Residuals should satisfy the property that the media is null when there is no fault (Invariance Relation) and is different from zero when there is a fault (*Fault Detectability*). For the detection and isolation of faults two possible techniques based on the use of filters bank are ([B19]):

- *Dedicated Observers Scheme (DOS)*: each filter is sensitive to a specific different fault.

- *Generalized Observers Scheme (GOS)*: each filter undergoes the influence of all faults except one.

In the case of DOS detection is dictated by the filter that is calibrated on that particular fault to which all others must be insensitive. It can detect more faults simultaneously but the selectivity of the filter is hard to get and you usually get at the expense of robustness. The DOS can also be compacted in a single observer if it is possible to find different residuals from the same observer, each of which is sensitive to a single fault. In the case of the GOS the detection is dictated by the filter which, not being sensitive to that particular fault, it does not exhibit variations of the residual. In this case it can be easier to link the filters to different faults but different faults are not detectable simoultaneously.

An alternative approach for the FDI is to use the observer (the KF in particular) for the estimation of a model parameter that may be indicative of a fault in stochastic environment. As an example, consider the following representation input-state-output of a linear system in which x,y,u,w,v,θ are state vectors, output, input, process noise, measurement noise,f and model' parameters, respectively:

$$\text{(Eq.2)}\qquad \begin{aligned} x(t+1) &= A(t)x(t)+B(t)u(t)+E(t)w^{'}(t)+F(t)\theta(t) \\ \theta(t+1) &= \theta(t)+w^{''}(t) \\ y(t) &= C(t)x(t)+G(t)v(t)+H(t)\theta(t) \end{aligned}$$

In this model, the parameter itself is subject to a linear dynamic driven by a part of process disturbance. If the value of the disturbance is zero, the parameter has a constant value; an impulse on the input channel $w^{''}$, instead determines a jump in the value of the parameter. In this case it is possible to design an asymptotic KF observer, assuming, as a state variable, the extended state (x,θ). In literature there are different techniques for synthesize an observer like that:

- Kalman filter for a linear system in continous time , also referred to as the Kalman-Bucy filter [B20].

- Kalman filter for a linear system in discrete time [B21].

- Extended Kalman Filter (EKF) for non linear dynamic systems in continuous time [B22].

- EKF for non linear dynamic systems in discrete time [B23].
- Unscented Kalman Filter (UKF) in discrete time [B24].
- H_∞ observers for linear and non-linear dynamic systems [B25].

3.2.2 Parity Space and Principal Component Analysis

In the field of analytical model-based FDI techniques, the analytical redundancy relations of the system are used to create residual signal. The approaches can be roughly classified into observer-based approaches and parameter estimation approaches as shown in Figure 3-2. another model based technique is Parity space. Parity space approaches have been proved to be structurally equivalent to the observer-based although the design procedures differ [B26]. The parity space methodology using the temporal redundancy has its advantages, especially in the discrete system. This method was firstly generalized by the Chow & Willsky, in 1984 [B27] and by others [B28] and [B29].

Let’s suppose to want detect and identify additive faults on input and output signals, for a system that is described by a state space model. The parity space approach, is a well-known method for this kind of problem, which is based on simple algebraic projections and geometry. The method computes a residual vector that is zero when no fault is present, and non-zero otherwise, to detect that a fault has occurred. The residual will also be different for different faults, to enable diagnosing which fault has occurred. The parity space approach often shows very good results in simulations, but it can be highly sensitive to measurement noise and process noise, since these are not taken into consideration in the design of the parity space. We will briefly review the results in [B29], where a state space model which includes both deterministic and stochastic unmeasurable disturbances is used and a statistical fault detection and isolation algorithm is derived. The probability for incorrect diagnosis can be computed explicitly for this method, given that only a single fault has occurred. A singular value decomposition (SVD) is used in computing the parity space, and this is instrumental in many approaches to fault detection. Moreover SVD is also the basic step in the principal components analysis (PCA) [B30], which is an alternative way to estimate a state space model from data if no model is available a priori. By a SVD of the covariance matrix for input output data, we can split the data into two parts, model and residual. The residual part can be used for fault detection similarly to the parity space residual. The covariance matrix can be estimated from normal operation data. To be able to isolate different faults we also need data from typical fault cases, to estimate how the faults affects the residuals.

3.3 Data-based techniques: Neural Network Approach

3.3.1 Introduction

In its most general form, a neural network is a machine that is designed to model the way in which the brain performs a particular task or function of interest; the network is usually implemented by using electronic components or is simulated in software on a digital computer. An important class of neural networks perform useful computations through a process of learning. To achieve good performance, neural networks employ a massive interconnection of simple computing cells referred to as “neurons” or “processing units”. A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network from its environment through a learning process.
- Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

The procedure used to perform the learning process is called a learning algorithm, the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective. The modification of synaptic weights provides the traditional method for the design of neural networks. Such an approach is the closest to linear adaptive filter theory, which is already well established and successfully applied in many diverse fields ([B31], [B32]).

It is apparent that a neural network derives its computing power through, first, its massively parallel distributed structure and, second, its ability to learn and therefore generalize. Generalization refers to the neural network’s production of reasonable outputs for inputs not encountered during training (learning).These two information processing capabilities make it possible for neural networks to find good approximate solutions to complex (large-scale) problems that are intractable. Neural networks offer the following useful properties and capabilities:

Nonlinearity: An artificial neuron can be linear or nonlinear. A neural network, made up of an interconnection of nonlinear neurons, is itself nonlinear. Moreover, the nonlinearity is of a special kind in the sense that it is distributed throughout the network.

Input–Output Mapping: A popular paradigm of learning, called supervised learning, involves modification of the synaptic weights of a neural network by applying a set of labelled training examples. Each example consists of a unique input signal and a corresponding desired (target) response. The network is presented with an example picked at random from the set, and the synaptic weights (free parameters) of the network are modified to minimize the difference between the desired response and the actual response of the network produced by the input signal in accordance with an appropriate statistical criterion. The training of the network is repeated for many examples in the set, until the network reaches a steady state where there are no further significant changes in the synaptic weights .The previously applied training examples may be reapplied during the training session, but in a different order. Thus the network learns from the examples by constructing an input–output mapping for the problem at hand. Such an approach brings to mind the study of nonparametric statistical inference, which is a branch of statistics dealing with model-free estimation, or, from a biological viewpoint, tabula rasa learning ([B33]); the term “nonparametric” is used here to signify the fact that no prior assumptions are made on a statistical model for the input data. In a nonparametric, the requirement is to “estimate” arbitrary decision boundaries in the input signal space using a set of examples, and to do so without invoking a probabilistic distribution model. A similar point of view is implicit in the supervised learning paradigm, which suggests a close analogy between the input–output mapping performed by a neural network and nonparametric statistical inference.

Adaptivity: Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment. In particular, a neural network trained to operate in a specific environment can be easily retrained to deal with minor changes in the operating environmental conditions. Moreover, when it is operating in a nonstationary environment (i.e., one where statistics change with time), a neural network may be designed to change its synaptic weights in real time. The natural architecture of a neural network for pattern classification, signal processing, and control applications, coupled with the adaptive capability of the network, makes it a useful tool in adaptive pattern classification, adaptive signal processing, and adaptive control. As a general rule, it may be said that the more adaptive we make a system, all the time ensuring that the system remains stable, the more robust its performance will likely be when the system is required to operate in a nonstationary environment. It should be emphasized, however, that adaptivity does not always lead to robustness; indeed, it may do the very opposite. For example, an adaptive system with short-time constants may change rapidly and therefore tend to respond to spurious disturbances, causing a drastic degradation in system performance. To realize the full benefits of adaptivity, the principal time constants of the system should be long enough for the system to ignore spurious disturbances, and yet short enough to respond to meaningful changes in the environment; the problem described here is referred to as the stability–plasticity dilemma ([B34]).

Evidential Response: In the context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to select, but also about the confidence in the decision made. This latter information may be used to reject ambiguous patterns, should they arise, and thereby improve the classification performance of the network.

Contextual Information: Knowledge is represented by the very structure and activation state of a neural network. Every neuron in the network is potentially affected by the global activity of all other neurons in the network. Consequently, contextual information is dealt with naturally by a neural network.

3.3.2 Models of neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network .The block diagram of Figure 3-3 shows the model of a neuron, which forms the basis for designing a large family of neural networks studied in later chapters. Here, we identify three basic elements of the neural model:

- A set of synapses, or connecting links, each of which is characterized by a weight or strength of its own. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. It is important to make a note of the manner in which the subscripts of the synaptic weight w_{kj} are written. The first subscript in w_{kj} refers to the neuron in question, and the second subscript refers to the input end of the synapse to which the weight refers. Unlike the weight of a synapse in the brain, the synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.

- An adder for summing the input signals, weighted by the respective synaptic strengths of the neuron; the operations described here constitute a linear combiner.
- An activation function for limiting the amplitude of the output of a neuron. The activation function is also referred to as a squashing function, in that it squashes (limits) the permissible amplitude range of the output signal to some finite value.

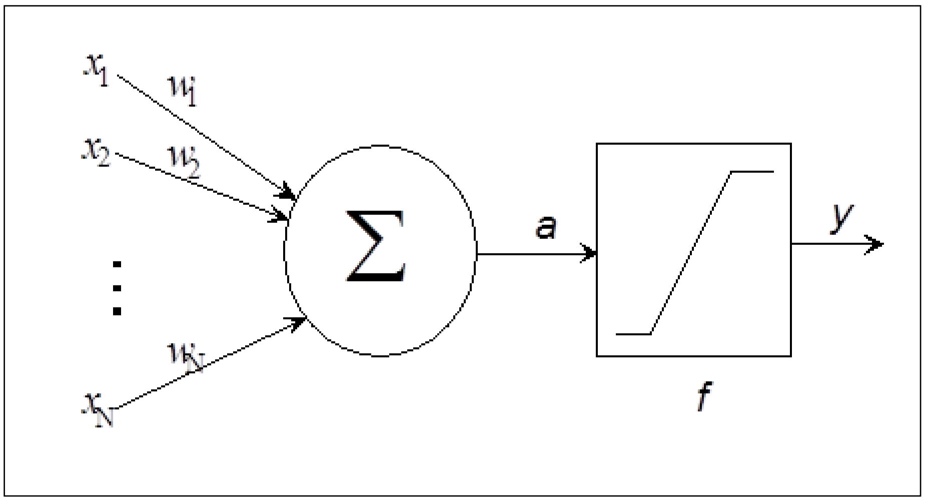


Figure 3-3: Nonlinear model of a neuron, labelled *k*.

Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval [0,1], or, alternatively, [-1,1]. The neural model of Figure 3-3 also includes an externally applied bias, denoted by b_k. The bias b_k has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative, respectively. In mathematical terms, we may describe the neuron *k* depicted in Figure 3-3 by writing the pair of equations:

(Eq.3) $$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

and

(Eq.4) $$y_k = \varphi(u_k + b_k)$$

where $x_1, x_2, \ldots, x_m$ are the input signals; $w_{k1}, w_{k2}, \ldots, w_{km}$ are the respective synaptic weights of neuron k; u_k (not shown in Figure 3-3) is the linear combiner output due to the input signals; b_k is the bias; $\varphi(\cdot)$ is the activation function; and y_k is the output signal of the neuron. The use of bias b_k has the effect of applying an affine transformation to the output u_k of the linear combiner in the model of Figure 3-3, as shown by

(Eq.5) $$v_k = u_k + b_k$$

The activation function, denoted by $\varphi(v)$, defines the output of a neuron in terms of the induced local field v. In what follows, we identify two basic types of activation functions:

- Threshold Function
- Sigmoid Function

The threshold activation function, described in Figure 3-4, is

(Eq.6) $$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

In engineering, this form of threshold function is commonly referred to as a *Heaviside function.* Correspondingly, the output of neuron k employing such a threshold function is expressed as

(Eq.7) $$y_k = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases}$$

where v_k is the induced local field of the neuron; that is,

(Eq.8) $$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k$$

In neural computation, such a neuron is referred to as the McCulloch–Pitts model ([B35]);. In this model, the output of a neuron takes on the value of 1 if the induced local field of that neuron is nonnegative, and 0 otherwise. This statement describes the all-or-none property of the McCulloch–Pitts model.

The sigmoid activation function, whose graph is “S”-shaped, is by far the most common form of activation function used in the construction of neural networks. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. An example of the sigmoid function is the logistic function, defined by

(Eq.9) $$\varphi(v) = \frac{1}{1+e^{-av}}$$

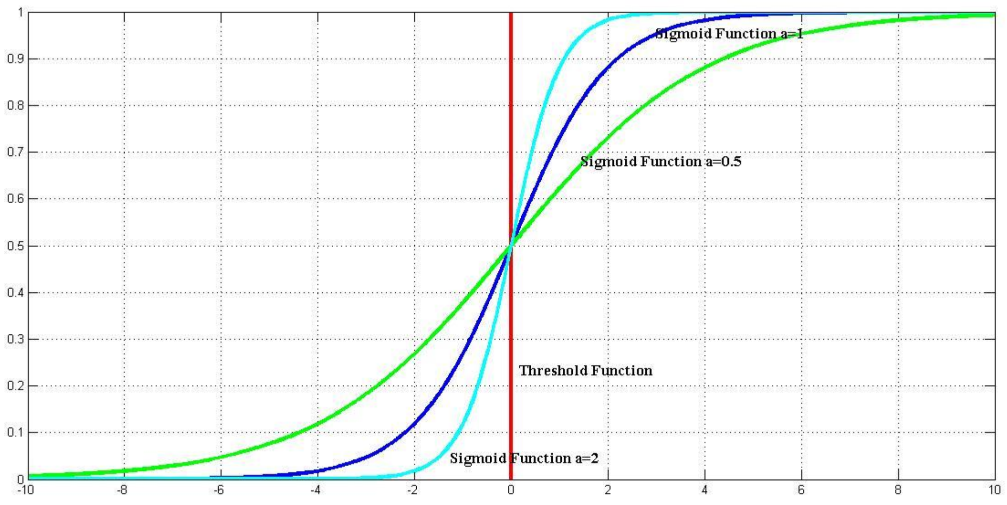


Figure 3-4 – Examples of Sigmoid and Threshold Functions

where a is the slope parameter of the sigmoid function. By varying the parameter a, we obtain sigmoid functions of different slopes, as illustrated in Figure 3-4. In fact, the slope at the origin equals $a=1$. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function. Where as a threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1. Note also that the sigmoid function is differentiable, whereas the threshold function is not. In fact differentiability is an important feature in the neural network theory.

The activation functions defined in (Eq.6) and (Eq.9) range from 0 to 1. It is sometimes desirable to have the activation function range from -1 to 1, in which case, the activation function is an odd function of the induced local field. Specifically, the threshold function of (Eq.6) is now defined as

(Eq.10) $$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ 0 & \text{if } v < 0 \end{cases}$$

which is commonly referred to as the sign function. For the corresponding form of a sigmoid function, we may use the hyperbolic tangent function, defined by

(Eq.11) $\varphi(v) = \tanh(v)$

Allowing an activation function of the sigmoid type to assume negative values as prescribed by (Eq.11) may yield practical benefits over the logistic function of (Eq.9).

3.3.3 Types of Neural Networks Architecture

Different types of neural network architectures can be defined; the most important are:

- Feed-forward single and multi-layer networks: The feed-forward neural network was the first and arguably most simple type of artificial neural network devised. In this network the information moves in only one direction. From the input nodes data goes through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

- Recurrent networks: Contrary to feed-forward networks, RNNs are models with bi-directional data flow. While a feed-forward network propagates data linearly from input to output, RNNs also propagate data from later processing stages to earlier stages.

- Modular networks: is a realization of a neural network architecture in which several small networks cooperate or compete to solve problems.

- Dynamic neural networks: these kind of networks not only deal with nonlinear multivariate behavior, but also include (learning of) time-dependent behavior such as various transient phenomena and delay effects.

- Cascading neural networks: this kind of network uses an initial simplified structure, then automatically trains and adds new hidden units one by one, creating a multi-layer structure. Once a new hidden unit has been added to the network, its input-side weights are frozen. This unit then becomes a permanent feature-detector in the network, available for producing outputs or for creating other, more complex feature detectors. The Cascade-Correlation architecture has several advantages over existing algorithms: it learns very quickly, the network determines its own size and topology, it retains the structures it has built even if the training set changes, and it requires no back-propagation of error signals through the connections of the network.

- Neuro-fuzzy networks: these kinds of network has a fuzzy inference system in the body of an artificial neural network.

The architecture selected in the present thesis for the realization of the virtual sensor is a Feed-forward Multilayer Network. This selection has been done in order to keep low the computational workload and to use the supervised learning training algorithm, which is described in the next sub-section.

The following three points highlight the basic features of multilayer perceptrons:

- The model of each neuron in the network includes a nonlinear activation function that is differentiable.
- The network contains one or more layers that are hidden from both the input and output nodes.
- The network exhibits a high degree of connectivity, the extent of which is determined by synaptic weights of the network.

3.3.4 Neural Network Learning Algorithms

Learning is one of the main features of artificial neural networks. A neural network learns from its environment through an interactive process of adjustments to its weights and bias values. Theoretically, the network becomes more knowledgeable after each iteration of the training process. During the learning process, the following events occur in sequence: first, the neural network is stimulated by the environment inputs; next, the neural network changes its parameters as a result of its environmental stimulation; finally, the neural network response to the environment stimulus changes because of the changes that have occurred in its internal structure. A learning algorithm is a prescribed set of well-defined rules for the solution of a learning problem. Learning algorithms differ from each other in the way in which the adjustment to a synaptic weight of a neuron is formulated. Another factor to be considered is the manner in which a neural network made up of a set of inter-connected neurons, relates to its environment.

3.3.4.1 Supervised Learning

In Figure 3-5 is described the block diagram that illustrates supervised learning. The knowledge was being represented by a set of input-output examples. The teacher is able to provide the neural network with a desired response for that training vector. The desired response represents the optimum action to be performed by the neural network. The network parameters are adjusted under the combined influence of the training vector and the error signal. This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network emulate the teacher. The form of supervised learning is the error-correction learning. It is a closed-loop feedback system, but the unknown environment is not in the loop. As a performance measure for the system we may think in terms of the mean-square error or the sum of squared errors over the training sample, defined as a function of the free parameters of the system. This function may be visualized as a multidimensional error-performance surface. The true error surface is averaged over all possible input-output examples. Any given operation of the system under the teacher's supervision is represented as a point on the error surface. For the system to improve performance over time and therefore learn from the teacher, the operating point has to move down successively toward a minimum point of the error surface. Nevertheless, given an algorithm designed to minimize the cost function, an adequate set of input-output examples, and enough time permitted to do the training, a supervised learning system is usually able to perform such tasks as pattern classification and function approximation.

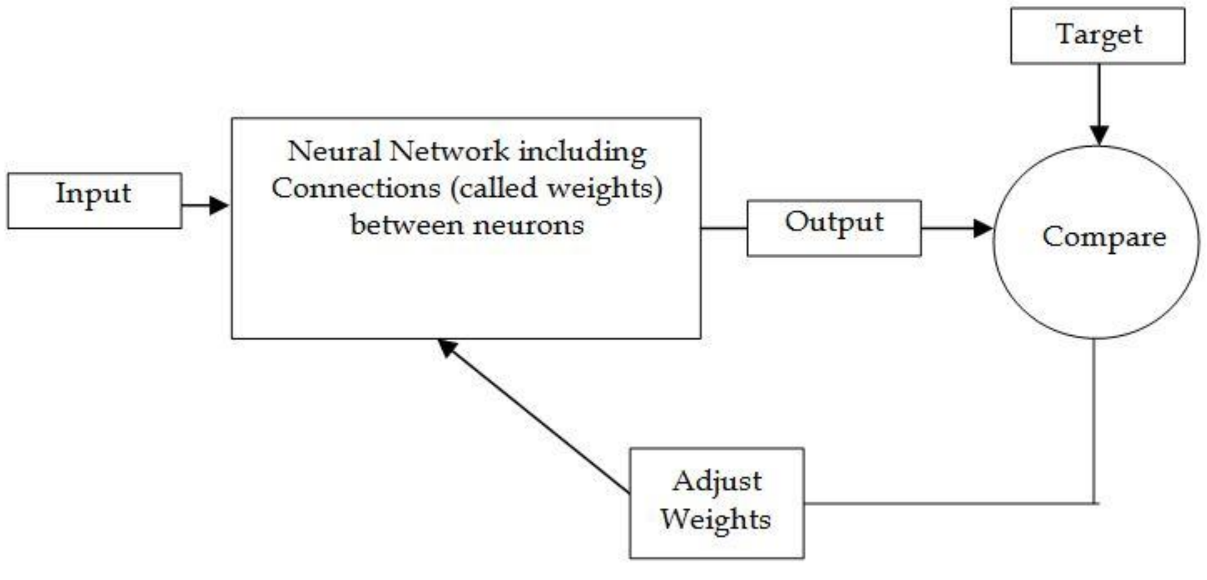

Target
Input
Neural Network including Connections (called weights) between neurons
Output
Compare
Adjust Weights

Figure 3-5: Supervised learning

Depending on how the supervised learning of the multilayer perceptron is actually performed, we may identify two different methods namely, batch learning and online learning.

In the batch method of supervised learning, adjustments to the synaptic weights of the multilayer perceptron are performed after the presentation of all the N examples in the training sample t that constitute one *epoch* of training. In other words, the cost function for batch learning is defined by the average error energy e_{av}. Adjustments to the synaptic weights of the multilayer perceptron are made on an *epoch-by-epoch basis*. Correspondingly, one realization of the learning curve is obtained by plotting e_{av} versus the number of epochs, where, for each epoch of training, the examples in the training sample t are *randomly shuffled*. The learning curve is then computed by *ensemble averaging* a large enough number of such realizations, where each realization is performed for a *different set of initial conditions* chosen at random. The method of gradient descent ([B36]) could be used to perform the training, with the following advantages:

- *accurate estimation* of the gradient vector (i.e., the derivative of the cost function e_{av} with respect to the weight vector w), thereby guaranteeing, under simple conditions, convergence of the method of steepest descent to a local minimum ([B36]);
- *parallelization of the learning process*. However, from a practical perspective, batch learning is rather demanding in terms of storage requirements. In a statistical context, batch learning may be viewed as a form of statistical inference. It is therefore well suited for solving nonlinear regression problems

In the on-line method of supervised learning, adjustments to the synaptic weights of the multilayer perceptron are performed on an *example-by-example basis.* The cost function to be minimized is therefore the total instantaneous error energy $\xi(n)$. For a given set of initial conditions, a single realization of the learning curve is obtained by plotting the final value $\xi(N)$ versus the number of epochs used in the training session, where, as before, the training examples are randomly shuffled after each epoch. Given that the training examples are presented to the network in a random manner, the use of on-line learning makes the search in the multidimensional weight space stochastic in nature; it is for this reason that the method of on-line learning is sometimes referred to as a *stochastic method.* This stochastic characteristic has the desirable effect of making it less likely for the learning process to be trapped in a local minimum, which is a definite advantage of on-line learning over batch learning. Another advantage of on-line learning is the fact that it requires much less storage than batch learning. Moreover, when the training data are *redundant* (i.e., the training sample t contains several copies of the same example), we find that, unlike batch learning, on-line learning is able to take advantage of this redundancy because the examples are presented one at a time.

Another useful property of on-line learning is its ability to *track small changes* in the training data, particularly when the environment responsible for generating the data is non-stationary. To summarize, the on-line learning is highly popular for solving *pattern-classification problems* for two important practical reasons:

- On-line learning is simple to implement.
- It provides effective solutions to large-scale and difficult pattern-classification problems.

3.3.4.2 Unsupervised Learning

There is no teacher to oversee the learning process when it comes to unsupervised learning. There are no labeled examples of the function to be learned by the network, in fact Figure 3-6 shows the block diagram of one form of a reinforcement learning system built around a critic that converts a primary reinforcement signal received from the environment into a higher quality reinforcement signal called the heuristic reinforcement signal, both of which are scalar input. Delayed-reinforcement learning is very appealing and provides the basis for the system to interact with its environment, thereby developing the ability to learn to perform a prescribed task solely on the basis of the outcomes of its experience that results form the interaction. In unsupervised learning there is no external teacher to oversee the learning process. Rather, provision is made for a task-independent measure of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically.

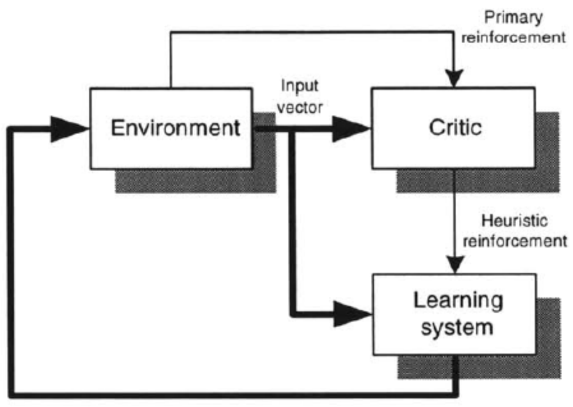


Figure 3-6: Unsupervised learning

3.3.5 Backpropagation Algorithm

The training method of multilayer perceptrons is the back-propagation algorithm [B37]. The development of the back-propagation algorithm in the mid-1980s represented a landmark in neural networks in that it provided a computationally efficient method for the training of multilayer perceptrons. The training proceeds in two phases:

1. In the *forward phase*, the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output. Thus, in this phase, changes are confined to the activation potentials and outputs of the neurons in the network.

2. In the *backward phase*, an error signal is produced by comparing the output of the network with a desired response. The resulting error signal is propagated through the network, again layer by layer, but this time the propagation is performed in the backward direction. In this second phase, successive adjustments are made to the synaptic weights of the network. Calculation of the adjustments for the output layer is straightforward, but it is much more challenging for the hidden layers.

Figure 3-7 shows the architectural graph of a multiplayer perceptron with two hidden layers and an output layer. To set the stage for a description of the multilayer perceptron in its general form, the network shown here is fully connected. This means that a neuron in any layer of the network is connected to all the neurons (nodes) in the previous layer. Signal flow through the network progresses in a forward direction, from left to right and on a layer-by-layer basis.

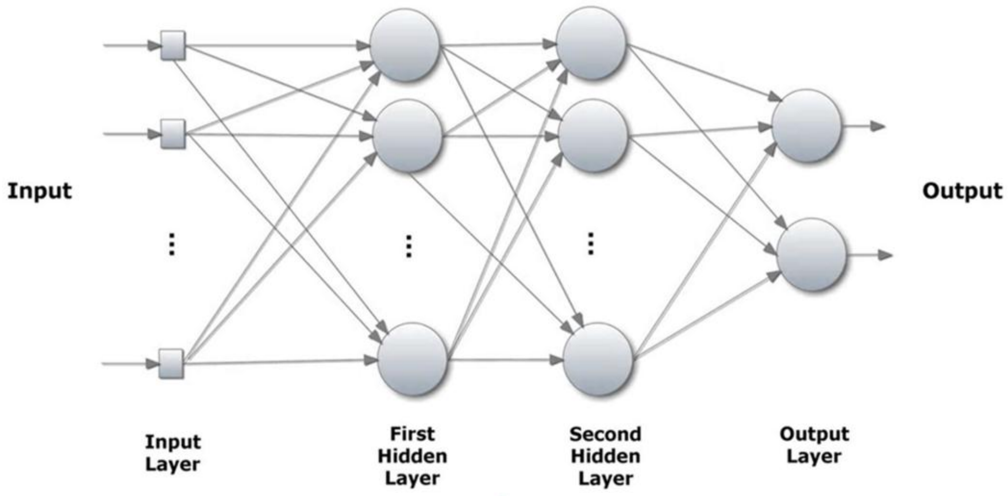


Figure 3-7: Architectural graph of a multilayer perceptron with two hidden layers

Two kinds of signals are identified in this network as shown Figure 3-8:

1. *Function Signals.* A function signal is an input signal (stimulus) that comes in at the input end of the network, propagates forward (neuron by neuron) through the network, and emerges at the output end of the network as an output signal. We refer to such a signal as a "function signal" for two reasons. First, it is presumed to perform a useful function at the output of the network. Second, at each neuron of the network through which a function signal passes, the signal is calculated as a function of the inputs and associated weights applied to that neuron. The function signal is also referred to as the input signal.

2. *Error Signals*. An error signal originates at an output neuron of the network and propagates backward (layer by layer) through the network. We refer to it as an "error signal" because its computation by every neuron of the network involves an error-dependent function in one form or another. The output neurons constitute the output layer of the network. The remaining neurons constitute hidden layers of the network. Thus, the hidden units are not part of the output or input of the network hence their designation as "hidden." The first hidden layer is fed from the input layer made up of sensory units (source nodes); the resulting outputs of the first hidden layer are in turn applied to the next hidden layer; and so on for the rest of the network.

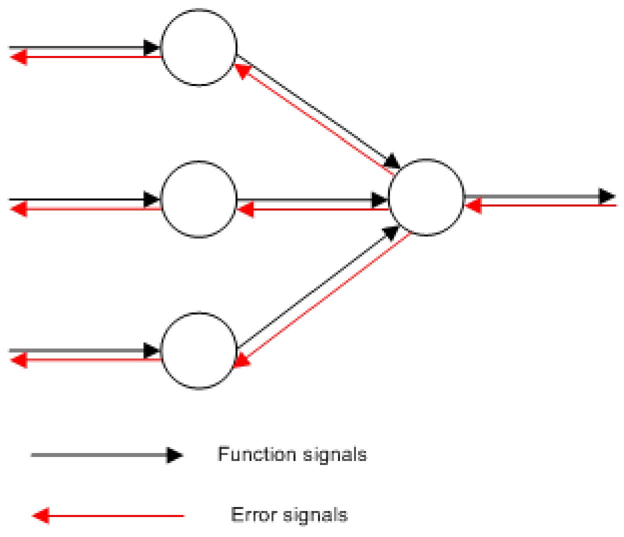


Figure 3-8: Illustration of the directions of two basic signal flows in a multilayer perceptron

Each hidden or output neuron of a multilayer perceptron is designed to perform two computations:

1. The computation of the function signal appearing at the output of each neuron, which is expressed as a continuous nonlinear function of the input signal and synaptic weights associated with that neuron.

2. The computation of an estimate of the gradient vector (i.e., the gradients of the error surface with respect to the weights connected to the inputs of a neuron), which is needed for the backward pass through the network.

The hidden neurons act as *feature detectors*; as such, they play a critical role in the operation of a multilayer perceptron. As the learning process progresses across the multilayer perceptron, the hidden neurons begin to gradually “discover” the salient features that characterize the training data. They do so by performing a nonlinear transformation on the input data into a new space called the *feature space*. Indeed, it is the formation of this feature space through supervised learning that distinguishes the multilayer perceptron from the single layer perceptron.

The purpose of back-propagation is to adjust the network weights so the network produces the desired output in response to every input pattern in a predetermined set of training patterns. It is a supervised algorithm, for every input pattern, there is an externally corresponding specified correct output, which acts as a target for the network to imitate. The difference between the output value and the desired target values is called as an error. It is necessary to minimize the errors. The learning with a teacher model must decide which patterns to include in the training set and specify the correct output for each. It is an off-line algorithm in the sense that training and normal operation occur at different times. In the usual case, training could be considered part of the producing process wherein the network is trained once for a particular function, then frozen and put into operation. No further learning occurs after the initial phase.

In order to train a back-propagation neural network, it is necessary to have a set of input patterns and corresponding desired output, and an error function that measures the cost of differences between network output and the desired values. This is the basic step to implement a back-propagation neural network.

- Present a training pattern and propagate it through the network to obtain the desired outputs.
- Compare the network outputs with the desired target values and then calculate the error.
- Calculate the derivatives $\frac{\partial E}{\partial w_{kj}}$ of the error with respect to the weights.
- Adjust the weights to minimize the error.
- Repeat the above procedure until the error is acceptably small or the limit of iteration is reached.

The error at the output of neuron j at the presentation of the nth training example is defined by:

(Eq.12) $$e_j(n) = d_j(n) - y_j(n)$$

Define the instantaneous value of the error energy for neuron j as $\frac{1}{2}e_j^2(n)$. The instantaneous value $\xi(n)$ of the total error energy is obtained by summing $\frac{1}{2}e_j^2(n)$ over all neurons in the output layer; these are neurons for which error signals can be calculated directly. The total error energy can be written as

(Eq.13) $$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

Where the set C includes all the neurons in the output layer of the network. Let N denote the total number of the training set. The average squared error energy can be written as

(Eq.14) $$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(n)$$

The instantaneous error energy $\xi(n)$, and the average error energy ξ_{av}, is a function of all the free parameters of the network. For a given training set, ξ_{av} represents the cost function as a measure of learning performance. The objective of the learning process is to adjust the free parameters of the network to minimize ξ_{av}. Consider a method of training in which the weights are updated on a pattern-by-pattern basis until one complete presentation of the entire training set as been dealt with. The adjustments to the weights are made in accordance with the respective errors computed for each pattern presented to the network.

The average of these individual weight changes over the training set is an estimate of the true change that would result from modifying the weights based on minimizing the cost function ξ_{av} over the entire training set.

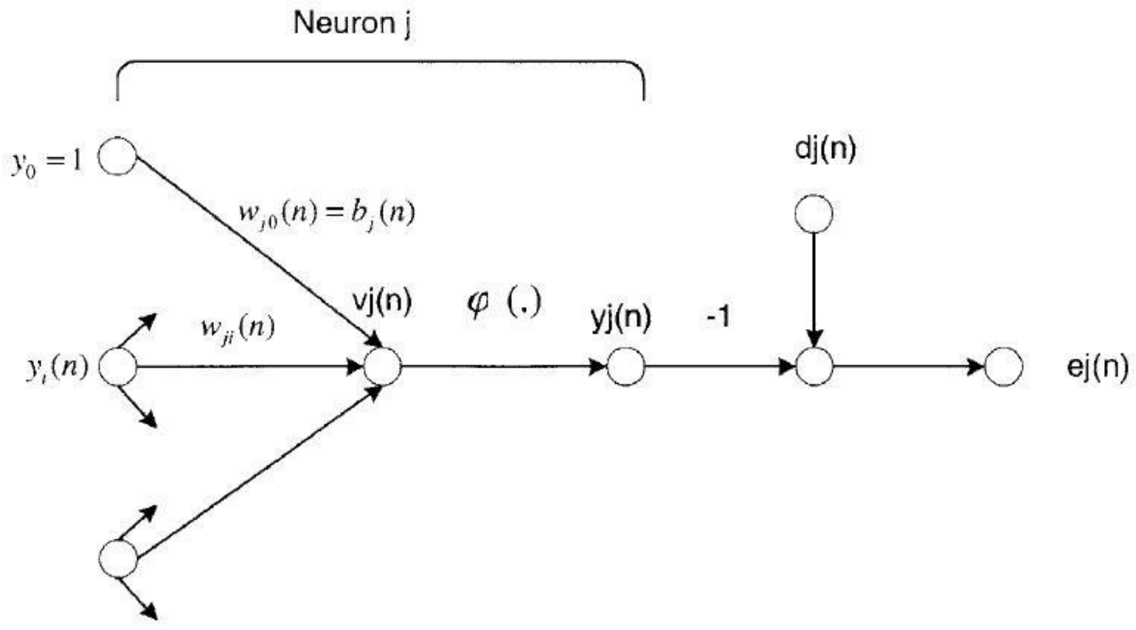


Figure 3-9: Signal flow graph highlighting the details of output of neuron j

From Figure 3-9, neuron j is fed by a set of function signals produced by a layer of neurons to its left. The induced local field $v_j(n)$ produced at the input of the activation function associated with neuron j is therefore

(Eq.15) $$v_j = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

where m is the total number of inputs fed to neuron j. The synaptic weight w_{j0} equals the bias b_j applied to neuron j. The function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

(Eq.16) $$y_j(n) = \varphi_j(v_j(n))$$

The back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $\Delta w_{ji}(n)$, which is proportional to the partial derivation $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$. This gradient can be expressed as:

(Eq.17) $$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

The partial derivative $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$ determines the direction of search in weight space for the synaptic weight w_{ji}. Differentiating both sides of (Eq.13) with respect to $e_j(n)$

(Eq.18) $$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n)$$

Differentiating both sides of (Eq.12) with respect to $y_j(n)$,

(Eq.19) $$\frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

Differentiating (Eq.16) with respect to $v_j(n)$

(Eq.20) $$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n))$$

Finally, differentiating (Eq.15) with respect to $w_{ji}(n)$ yields

(Eq.21) $$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_j(n)$$

Substitute (Eq.18) to (Eq.21) in (Eq.17) yields

(Eq.22) $$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j(v_j(n))y_j(n)$$

The correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the delta rule:

(Eq.23) $$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

Where η is the learning-rate parameter of the back-propagation algorithm. The use of the minus sign in (Eq.23) accounts for gradient descent in weight space $\xi(n)$. Substitute (Eq.22) in (Eq.23) yields

(Eq.24) $$\Delta w_{ji}(n) = \eta\delta_j(n)y_j(n)$$

Where the local gradient $\delta_j(n)$ is defined by

(Eq.25) $$\delta_j(n) = -\frac{\partial \xi(n)}{\partial v_j(n)} = -\frac{\partial \xi(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi_j(v_j(n))$$

The local gradient points to required changes in synaptic weights. According to (Eq.25), the local gradient $\delta(n)$ for output neuron j is equal to the product of the corresponding error signal $e_j(n)$ for that neuron and the derivative $\varphi_j(v_j(n))$ of the associated activation function.

The key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j. There are two cases, depending on where in the network neuron j is located.

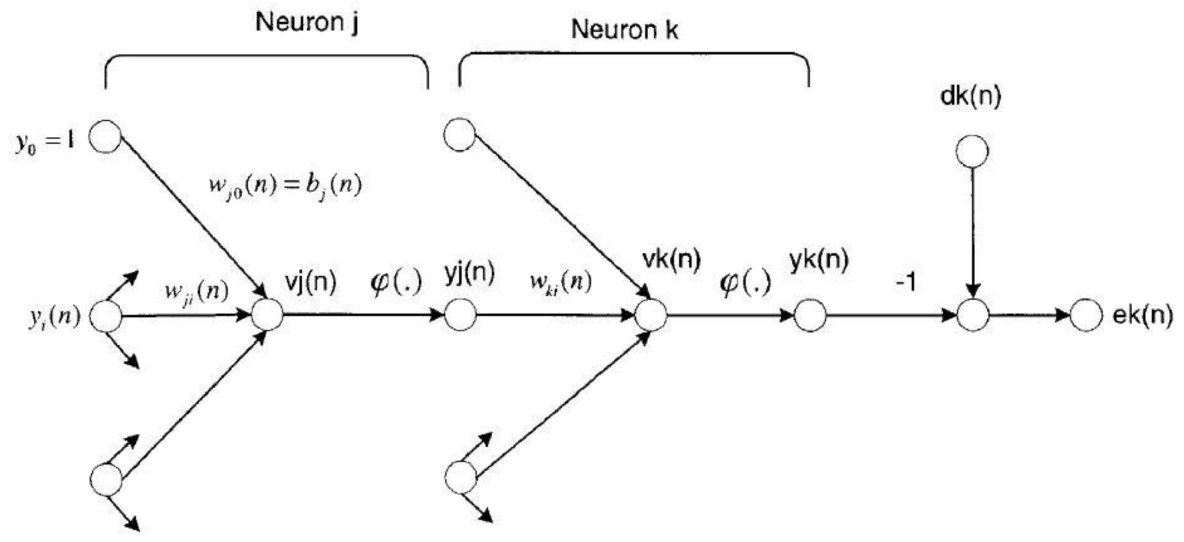


Figure 3-10: Signal flow graph highlighting the details of output of neuron k connected to hidden neuron j

<u>Neuron j is an Output Node</u>

When neuron j is located in the output layer of the network, which means $j = 0$, it is supplied with a desired response of its own. The error signal $e_j(n)$ associated with this neuron is $e_j(n) = d_j(n) - y_j(n)$. Having determined $e_j(n)$, it is a straightforward matter to compute the local gradient $\delta_j(n)$using (Eq.25). The expression of the local gradient for neuron x in layer 0 is $\delta_x^0 = e_x^0 \cdot f^{0'}(n_x^0)$

<u>Neuron j Is a Hidden Node</u>

When neuron j is located in a hidden layer of the network, there is no specified desired response for that neuron. The error signal for a hidden neuron would have to be determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected; this is where the development of the back-propagation algorithm gets complicated. Consider the situation presented in Figure 3-10, which depicts neuron j as a hidden node of the network. According to (Eq.25), it may redefine the local gradient $\delta_j(n)$for hidden neuron j as

(Eq.26) $$\delta_j(n) = -\frac{\partial\xi(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial\xi(n)}{\partial y_j(n)}\varphi_j^{'}(v_j(n))$$

Use (Eq.20) to calculate the partial derivative $\frac{\partial \xi(n)}{\partial y_j(n)}$,

(Eq.27) $$\xi(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n)$$

(Eq.13) with index k used in place of index j. Differentiating (Eq.27) with respect to the function signal $y_j(n)$

(Eq.28) $$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}$$

Use the chain rule for the partial derivative $\frac{\partial e_k(n)}{\partial y_j(n)}$, and rewrite (Eq.28) in the equivalent form

(Eq.29) $$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial e_k(n)}{\partial v_k(n)}$$

From Figure 3-10, it can be found that

(Eq.30) $$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n))$$

Therefore

(Eq.31) $$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k(v_k(n))$$

From Figure 3-10, for neuron k the induced local field is

(Eq.32) $$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n)$$

Where m is the total number of inputs applied to neuron k. The synaptic weight $w_{k0}(n)$ is equal to the bias $b_k(n)$ applied to neuron k, and the corresponding input is fixed at the value $+1$. Differentiating (Eq.31) with respect to $y_j(n)$ yields

(Eq.33) $$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

Using equations (Eq.31) and (Eq.32) in (Eq.29) we get the desired partial derivative:

(Eq.34) $$\frac{\partial \xi(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi_k(v_k(n))w_{kj}(n) = -\sum_k \delta_k(n)w_{kj}(n)$$

The definition of the local gradient $\delta_k(n)$ given in (Eq.26) with the index k substituted for j. Substituting Equations (Eq.34) in (Eq.26) we get the desired partial derivative:

(Eq.35) $$\delta_j(n) = \varphi_j(v_j(n))\sum_k \delta_k(n)w_{kj}(n)$$

The factor $\varphi_j(v_j(n))$ involved in the computation of the local gradient $\delta_j(n)$ in (Eq.35) depends solely on the activation function associated with hidden neuron j. The remaining factor involved in this computation, namely the summation over k, depends on two sets of terms. The first set of terms, the $\delta_k(n)$, requires knowledge of the error signals $e_k(n)$, for all neurons that lie in the layer to the immediate right of hidden neuron j, and that are directly connected to neuron k: see Figure 3-10. The second set of terms, the $w_{kj}(n)$, consists of the synaptic weights associated with these connections.

This is the relation for back-propagation algorithm. Firstly, the correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule:

(Eq.36) $$(\Delta w_{ji}(n)) = (\eta)(\delta_j(n))(y_i(n))$$

Which weight correction = learning-rate parameter times local gradient times input signal of neuron j. Secondly, the local gradient $\delta_j(n)$ depends on whether neuron j is an output node or a hidden node:

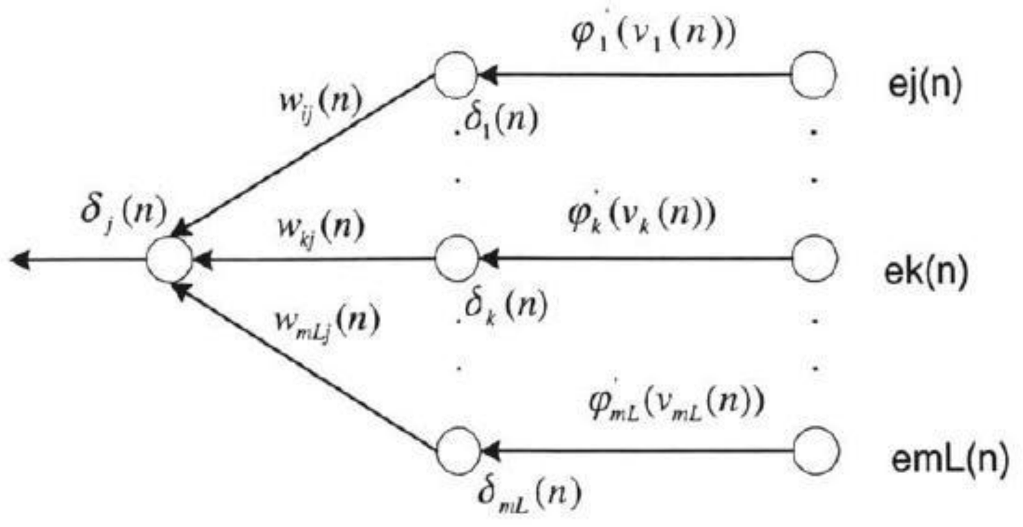


Figure 3-11: Signal flow graph of a part of the adjoint system pertaining to back propagation of error signals.

Neuron j is an Output Node

$\delta_j(n)$ equals the product of the derivative $\varphi_j'(v_j(n))$ and the error signal $e_j(n)$, both of which associated with neuron j

Neuron j is a Hidden Node

$\delta_j(n)$ equals the product of the derivative $\varphi_j'(v_j(n))$ and the weighted sum of δs computed for the neurons in the next hidden or output layer that are connected to neuron j.

Forward Pass Phase

In the forward pass, the weights remain unaltered throughout the network. The function signals appearing at the output of neuron j is:

(Eq.37) $$y_j(n)=\varphi(v_j(n))$$

Where $v_j(n)$ is the induced local field of neuron j

(Eq.38) $$v_j(n)=\sum_i^m w_{ji}(n)y_i(n)$$

where m is the total number of inputs applied to neuron j, and $w_{ji}(n)$ is the synaptic weight connecting neuron i to neuron j, and $y_i(n)$ is the input signal of neuron j. If neuron j is in the first hidden layer

(Eq.39) $\quad y_i(n) = x_i(n)$

where $x_i(n)$ is the ith element of the input vector. If neuron j is the output layer of the network

(Eq.40) $\quad y_i(n) = o_j(n)$

where $o_j(n)$ is the jth element of the output vector. This output is compared with the desired response $d_j(n)$, obtaining the error signal $e_j(n)$ for the jth output neuron. Thus the forward pass begins at the first hidden layer by presenting it with the input vector, and terminates at the output layer by computing the error signal for each neuron of this layer.

Backward Pass Phase

Backward pass, starts at the output layer by passing the error signals through the network, and recursively computing the local gradient for each neuron. This process permits the synaptic weights of the network to undergo changes in accordance with the delta rule of (Eq.36). First, it uses (Eq.36) to compute the changes to the weights of all the connections feeding into the output layer. Second, (Eq.35) is applied to compute δ for all neurons in the penultimate layer. This recursive computation is continued, layer by layer, by propagating the changes to all synaptic weights in the network.

Learning Rate Phase

The back-propagation algorithm provides an approximation in weight space computed by the method of steepest descent. From one iteration, the smaller the learning-rate η, the smaller the changes to the synaptic weights will be, and the smoother will be in weight space. However it will have a slower rate of learning. If the learning-rate η is too large, the large changes in the synaptic weights may become unstable. It was assumed that the learning-rate parameter is a constant in fact, this parameter should be connection-dependent. In the application of the back-propagation algorithm, the synaptic weight may be adjustable, or weights can be fixed during the adaptation.

Activation Function

The computation of the δ for each neuron of the multilayer perceptron requires knowledge of the derivative of the activation function $\varphi(\cdot)$ associated with that neuron. For this derivative to exist, we require the function $\varphi(\cdot)$ to be continuous. In basic terms, differentiability is the only requirement that an activation function has to satisfy. An example of a continuously differentiable nonlinear activation function commonly used in multilayer perceptrons is sigmoidal nonlinearity, two forms of which are described here:

- *Logistic Function*. This form of sigmoidal nonlinearity, in its general form, is defined by

(Eq.41) $$\varphi_j\left(v_j(n)\right)=\frac{1}{1+e^{-av_j(n)}} \quad a>0$$

where $v_j(n)$ is the induced local field of neuron j and a is an adjustable positive parameter. According to this nonlinearity, the amplitude of the output lies inside the range $0 \le y_j \le 1$.

- *Hyperbolic tangent function*. Another commonly used form of sigmoidal nonlinearity is the hyperbolic tangent function, which, in its most general form, is defined by

(Eq.42) $$\varphi_j\left(v_j(n)\right)=a\tanh\left(bv_j(n)\right)$$

where a and b are positive constants. In reality, the hyperbolic tangent function is just the logistic function rescaled and biased.

Stopping Criteria

In general, the back-propagation algorithm cannot be shown to converge, and there are no well-defined criteria for stopping its operation. Rather, there are some reasonable criteria, each with its own practical merit, that may be used to terminate the weight adjustments. To formulate such a criterion, it is logical to think in terms of the unique properties of a *local* or *global minimum* of the error surface. Let the weight vector w^* denote a minimum, be it local or global. A necessary condition for w^* to be a minimum is that the gradient vector $g(w)$ (i.e., first-order partial derivative) of the error surface with respect to the weight vector w must be zero at $w = w^*$. Accordingly, we may formulate a sensible convergence criterion for back-propagation learning as follows [B38]:

The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.

The drawback of this convergence criterion is that, for successful trials, learning times may be long. Also, it requires the computation of the gradient vector $g(w)$. Another unique property of a minimum that we can use is the fact that the cost function $\xi_{av}(w)$ is stationary at the point $w = w^*$. Another different criterion of convergence:

The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small.

The rate of change in the average squared error is typically considered to be small enough if it lies in the range of 0.1 to 1 percent per epoch. Sometimes a value as small as 0.01 percent per epoch is used. Unfortunately, this criterion may result in a premature termination of the learning process.

There is another useful, and theoretically supported, criterion for convergence: After each learning iteration, the network is tested for its generalization performance. The learning process is stopped when the generalization performance is adequate or when it is apparent that the generalization performance has peaked.

Generalization

In back-propagation learning, we typically start with a training sample and use the back-propagation algorithm to compute the synaptic weights of a multilayer perceptron by loading (encoding) as many of the training examples as possible into the network. The hope is that the neural network so designed will *generalize* well. A network is said to generalize well when the input–output mapping computed by the network is correct (or nearly so) for test data never used in creating or training the network; the term “generalization” is borrowed from psychology. Here, it is assumed that the test data are drawn from the same population used to generate the training data.

The learning process (i.e., training of a neural network) may be viewed as a “curve-fitting” problem. The network itself may be considered simply as a nonlinear input–output mapping. Such a viewpoint then permits us to look at generalization not as a mystical property of neural networks, but rather simply as the effect of a good nonlinear interpolation of the input data. The network performs useful interpolation primarily because multilayer perceptrons with continuous activation functions lead to output functions that are also continuous.

A neural network that is designed to generalize well will produce a correct input–output mapping even when the input is slightly different from the examples used to train the network. When, however, a neural network learns too many input–output examples, the network may end up memorizing the training data. It may do so by finding a feature (due to noise, for example) that is present in the training data, but not true of the underlying function that is to be modeled. Such a phenomenon is referred to as *overfitting* or *overtraining*. When the network is over-trained, it loses the ability to generalize between similar input–output patterns.

Generalization is influenced by three factors:

- the size of the training sample and how representative the training sample is of the environment of interest
- the architecture of the neural network
- the physical complexity of the problem at hand.

Clearly, we have no control over the lattermost factor. In the context of the other two factors, we may view the issue of generalization from two different perspectives:

- The architecture of the network is fixed (hopefully in accordance with the physical complexity of the underlying problem), and the issue to be resolved is that of determining the size of the training sample needed for a good generalization to occur.
- The size of the training sample is fixed, and the issue of interest is that of determining the best architecture of network for achieving good generalization.

Approximations of Functions

A multilayer perceptron trained with the back-propagation algorithm may be viewed as a practical vehicle for performing a *nonlinear input–output* mapping of a general nature. To be specific, let m_0 denote the number of input (source) nodes of a multiplayer perceptron, and let M= m_L denote the number of neurons in the output layer of the network. The input–output relationship of the network defines a mapping from an m_0-dimensional Euclidean input space to an M-dimensional Euclidean output space, which is infinitely continuously differentiable when the activation function is likewise.

Universal Approximation Theorem

In the mathematical theory of artificial neural networks, the universal approximation theorem states [B39] that a feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function. The theorem thus states that simple neural networks can represent a wide variety of interesting functions when given appropriate parameters; it does not touch upon the algorithmic learnability of those parameters.

One of the first versions of the theorem was proved by George Cybenko in 1989 for sigmoid activation functions [B40].

Kurt Hornik showed in 1991 [B41] that it is not the specific choice of the activation function, but rather the multilayer feed-forward architecture itself which gives neural networks the potential of being universal approximators. The output units are always assumed to be linear. The theorem is the following [B42]:

Let $\varphi(\cdot)$be a non-constant, bounded, and monotone-increasing continuous function. Let I_{m0} denote the m_0-dimensional unit hypercube $[0,1]^{m0}$. The space of continuous functions on I_{m0} is denoted by C(I_{m0}). Then, given any function $f \in C\,(I_{m0})$ and ε >0, there exist an integer m_1 and sets of real constants α_i, b_i, and w_{ij}, where i=1....,1, m_1 and j =1...., m_0 such that we may define

$$F(x_1,...x_{m0}) = \sum_{i=1}^{m_1} \alpha_i \varphi\left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i\right)$$

as an approximate realization of the function $f(\cdot)$; that is,

$| F(x_1,...x_{m0}) - f(x_1,...x_{m0}) | < \varepsilon$

for all $x_1, x_2,...x_{m0}$ *that lie in the input space.*

The universal approximation theorem is directly applicable to multilayer perceptrons. We first note, for example, that the hyperbolic tangent function used as the nonlinearity in a neural model for the construction of a multilayer perceptron is indeed a non-constant, bounded, and monotone-increasing function; it therefore satisfies the conditions imposed on the function $\varphi(\cdot)$ Next, we note that equation of the theorem represents the output of a multilayer perceptron described as follows:

1. The network has m0 input nodes and a single hidden layer consisting of m1 neurons; the inputs are denoted by $x_1,...x_{m0}$
2. Hidden neuron i has synaptic weights $w_{i1},...w_{m0}$, and bias b_i
3. The network output is a linear combination of the outputs of the hidden neurons, with $\alpha_1,...\alpha_{m1}$ defining the synaptic weights of the output layer

The universal approximation theorem is an existence theorem in the sense that it provides the mathematical justification for the approximation of an arbitrary continuous function as opposed to exact representation. The theorem equation, which is the backbone of the entire theorem, merely generalizes approximations by finite Fourier series. In effect, the theorem states that *a single hidden layer is sufficient for a multilayer perceptron to compute a uniform approximation to a given training set represented by the set of inputs* $x_1,...x_{m0}$ *and a desired (target) output f (* $x_1,...x_{m0}$ *)*

<u>PROS AND CONS OF BACK-PROPAGATION LEARNING</u>

First and foremost, it should be understood that the back-propagation algorithm is not an algorithm intended for the optimum design of a multilayer perceptron. Rather, the correct way to describe it is to say:

The back-propagation algorithm is a computationally efficient technique for computing the gradients (i.e., first-order derivatives) of the cost function $\xi(w)$*, expressed as a function of the* *adjustable parameters (synaptic weights and bias terms) that characterize the multilayer perceptron.*

The computational power of the algorithm is derived from two distinct properties:

1. The back-propagation algorithm is simple to compute locally.
2. It performs stochastic gradient descent in weight space, when the algorithm is implemented in its on-line (sequential) mode of learning

The back-propagation algorithm is an example of a connectionist paradigm that relies on local computations to discover the information-processing capabilities of neural networks.

This form of computational restriction is referred to as the locality constraint, in the sense that the computation performed by each neuron in the network is influenced solely by those other neurons that are in physical contact with it. The use of local computations in the design of (artificial) neural networks is usually advocated for three principal reasons:

1. Neural networks that perform local computations are often held up as *metaphors* for biological neural networks.
2. The use of local computations permits a graceful degradation in performance caused by hardware errors and therefore provides the basis for a *fault-tolerant* network design.
3. Local computations favour the use of *parallel architectures* as an efficient method for the implementation of neural networks.

A multilayer perceptron trained with the back-propagation algorithm manifests itself as a nested *sigmoidal structure*, written for the case of a single output in the compact form

(Eq.43) $$F(x,w)=\varphi\left(\sum_k w_{ok}\varphi\left(\sum_j w_{kj}\varphi\left(\ldots\varphi\left(\sum_i w_{li}x_i\right)\right)\right)\right)$$

where $\varphi(\cdot)$ is a sigmoid activation function; w_{ok} is the synaptic weight from neuron k in the last hidden layer to the single output neuron o, and so on for the other synaptic weights; and x_i is the ith element of the input vector x. The weight vector w denotes the entire set of synaptic weights ordered by layer, then neurons in a layer, and then synapses in a neuron. The scheme of nested nonlinear functions described in (Eq.43) is unusual in classical approximation theory. It is a universal approximator.

Sensitivity Analysis

Another computational benefit gained from the use of back-propagation learning is the efficient manner in which we can carry out a sensitivity analysis of the input–output mapping realized by the algorithm. The sensitivity of an input–output mapping function F with respect to a parameter of the function, denoted by ω, is defined by [B43]

(Eq.44) $$S_{\omega}^{F} = \frac{\partial F / F}{\partial \omega / \omega}$$

Consider then a multilayer perceptron trained with the back-propagation algorithm. Let the function $F(w)$ be the input–output mapping realized by this network; w denotes the vector of all synaptic weights (including biases) contained in the network. In the back propagation algorithm, we showed that the partial derivatives of the function $F(w)$ with respect to all the elements of the weight vector w can be computed efficiently. In particular, we see that the complexity involved in computing each of these partial derivatives is linear in W, the total number of weights contained in the network. This linearity holds regardless of where the synaptic weight in question appears in the chain of computations.

Robustness

Hassibi and Kailath ([B44]) have shown that the back-propagation algorithm is a locally H^{∞}-optimal filter. The term “local” means that the initial value of the weight vector used in the back-propagation algorithm is sufficiently close to the optimum value w* of the weight vector to ensure that the algorithm does not get trapped in a poor local minimum. In conceptual terms, it is satisfying to see that back-propagation algorithms belong to the same class of H^{∞}-optimal filters.

Convergence

The back-propagation algorithm uses an “instantaneous estimate” for the gradient of the error surface in weight space. The algorithm is therefore stochastic in nature; that is, it has a tendency to zigzag its way about the true direction to a minimum on the error surface. Indeed, back-propagation learning is an application of a statistical method known as stochastic approximation that was originally proposed by Robbins and Monro ([B45]). Consequently, it tends to converge slowly. We may identify two fundamental causes for this property ([B46]):

1. The error surface is fairly flat along a weight dimension, which means that the derivative of the error surface with respect to that weight is small in magnitude. In such a situation, the adjustment applied to the weight is small, and consequently many iterations of the algorithm may be required to produce a significant reduction in the error performance of the network. Alternatively, the error surface is highly curved along a weight dimension, in which case the derivative of the error surface with respect to that weight is large in magnitude. In this second situation, the adjustment applied to the weight is large, which may cause the algorithm to overshoot the minimum of the error surface.

2. The direction of the negative gradient vector (i.e., the negative derivative of the cost function with respect to the vector of weights) may point away from the minimum of the error surface: hence, the adjustments applied to the weights may induce the algorithm to move in the wrong direction.

Local Minima

Another peculiarity of the error surface that affects the performance of the back-propagation algorithm is the presence of *local minima* (i.e., isolated valleys) in addition to global minima; in general, it is difficult to determine the numbers of local and global minima. Since back-propagation learning is basically a hill-climbing technique, it runs the risk of being trapped in a local minimum where every small change in synaptic weights increases the cost function. But somewhere else in the weight space, there exists another set of synaptic weights for which the cost function is smaller than the local minimum in which the network is stuck. It is clearly undesirable to have the learning process terminate at a local minimum, especially if it is located far above a global minimum.

4 PROPOSED VIRTUAL SENSORS FOR AIR DATA COMPUTER

In this section the specific work carried out during the PhD at CIRA is detailed. In particular, the design, development, integration and testing of a low cost air data system, developed to provide accurate and reliable data to the flight control system during the critical flight phases of take-off and landing is presented. Commercial air data systems are generally designed to provide cockpit information and not to be enslaved to a flight control system, therefore they are not able to fit the requirements (precision, reliability, etc.) of a FCS specifically designed to autonomously manage critical phases of the flight. This reason motivates the need for the development of a custom air data system (ADS). In particular, the thesis presents the algorithms implemented to compute air data and to perform model based self-diagnostic, the architecture of the air data system and its integration into the flying platform. The testing process is carried out both in laboratory and in-flight in order to validate the algorithms, evaluate the performance and accomplish the calibration of the system. The laboratory test rig is mainly constituted by a Real-Time Hardware in-the-Loop simulation platform. The in-flight test have been performed by using a Very Light Aircraft, a suitably modified Tecnam P92, identified as FLARE (Flying Laboratory for Aeronautical REsearch), which is used as an UAV to perform an autonomous mission, during which the ADS performances are assessed.

While in Section 3 some theoretical foundation have been introduced, in the present and following Sections the specific choices on which the results here documented have been achieved, are described.

4.1 Air Data Computer Architecture and Algorithms

The design process of the custom Air Data System (ADS) started from the identification of system requirements. They define the raw flight data to be measured and the sensors performances, in terms of data accuracy, output rate and latency, which are essentially due to the constraints of the flight control system. Requirements also concern the system size and weight, constrained by the vehicle characteristics. System requirements leaded the following design phases which consisted in the selection of the on board sensors and the definition of the ADS hardware architecture.

Moreover advanced ADSs deliver flight critical information to the aircraft and pilot throughout the flight regime. Key products include angle of attack and stall protection systems, pitot and pitot static probes, outside and total air temperature sensors as shown in Figure 4-1. Designed to meet advancements in fly-by-wire and digital flight controls, an ADS provides more accurate digital air data output and improved system performance. Usually a sensors suite composed only by and pitot static probes, outside and total air temperature sensors, provides the inputs to an Air Data Computer (ADC) which computes all the flight data, as shown in Figure 4-1

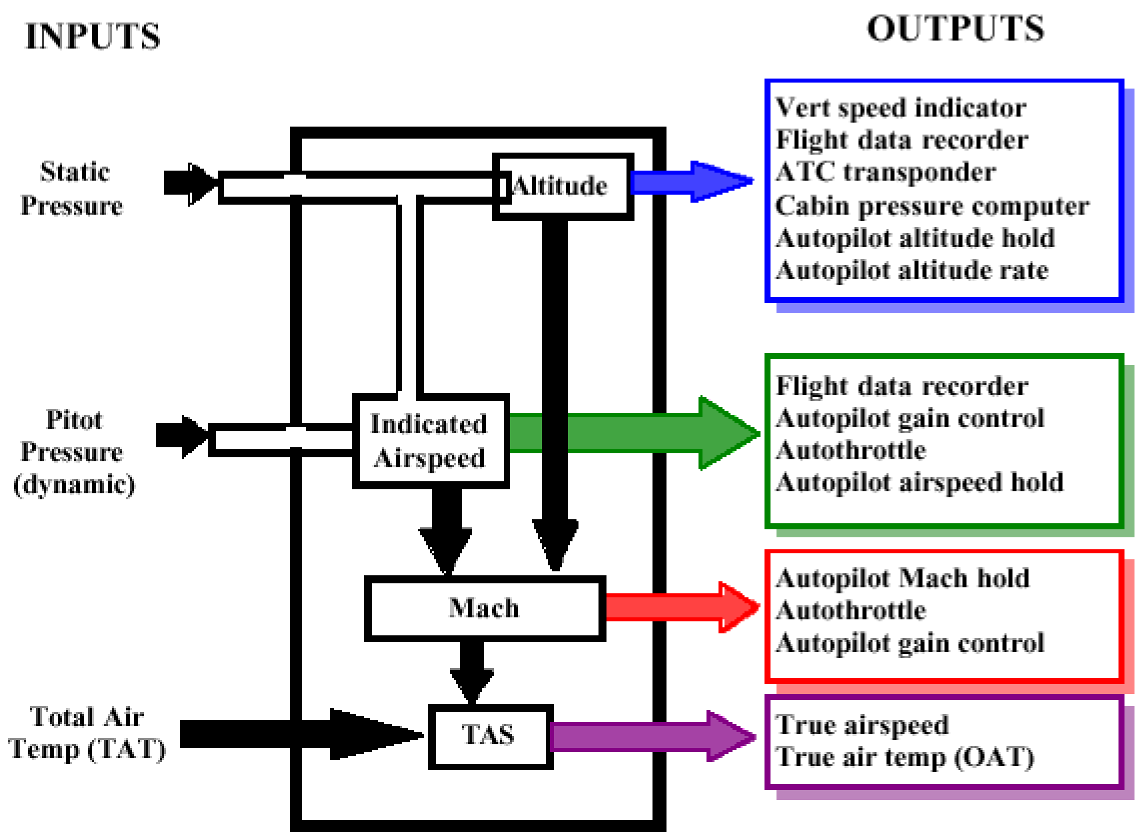


Figure 4-1: Air Data Computer Architecture

The air data measurements are computed through the classical air data equations **[**B47], reported here for the sake of completeness. They allow computing IAS [m/s], Mach number (*M*), TAS [m/s], Pressure Altitude (*PALT*) [m], Vertical Speed (*VS*) [m/s], density altitude (*h*) [m], and Total temperature (T_t) [K] starting from the raw measurements of total pressure (P_t) [mbar], static pressure (P_S) [mbar] and static temperature (T_S) [K], as follows:

(Eq.45) $$IAS = 760.92\sqrt{\left(\frac{Q_c}{1013.25}+1\right)^{(\gamma-1)/\gamma}-1}$$

(Eq.46) $$M = \sqrt{5\left[\left(\frac{P_t}{P_s}\right)^{(\gamma-1)/\gamma}-1\right]}$$

(Eq.47) $$TAS = M\sqrt{\gamma R T_S}$$

(Eq.48) $$PALT = -\frac{T_0}{\lambda}\left(1-\frac{P_s}{Baroset}\right)^{\frac{R\lambda}{9.807}}$$

(Eq.49) $$VS = \frac{d}{dt}PALT$$

(Eq.50) $$h = 44345.9\left[1-\left(\frac{\rho}{\rho_0}\right)^{0.234907}\right]$$

(Eq.51) $$T_t = T_s\cdot\left(1+\frac{\gamma-1}{2}M^2\right)$$

where Qc [mbar] is the impact pressure, evaluated as difference between total and static pressure, γ is the heat capacity ratio, R [J/(kg*K)] is the ideal gas constant, T_0 [K] and λ [K/m] are the see level temperature and temperature vertical gradient provided by the International Standard Atmosphere (ISA), respectively. Next, Baroset [mbar] is the static pressure value on the runway; it is a function of local weather conditions on the airport and is set by the pilot. Finally, ρ_0 [kg/m^3] represents the see level ISA density, whilst the air density ρ [kg/m^3] is computed through the equation of the ideal gases $\rho = P_s/RT_s$. The air data set is completed by the measurements of the aerodynamic angles, that is, angle of attack and angle of sideslip.

4.2 Model Based Virtual Sensors for Air Data Measurements

4.2.1 Virtual Angle of Attack

A simplified Model-Based virtual sensor for the Angle of Attack has been developed for evaluation in level longitudinal flight. A more complex AoA virtual Sensor, based on ANN, for more general flight conditions has also been designed and used.

For pure longitudinal level flight, simple flight mechanics relations between attitude angles apply. In particular, the relation:

(Eq.52) $\alpha = \vartheta - \gamma$

has been used as model-based AOA.

4.2.2 Virtual Indicated Airspeed

In level flight trim conditions, there exist equilibrium equations which relate the IAS to angle of attack and stabilator deflection [B48]. Although these relations are quadratic, due to the limited flight envelope of the FLARE vehicle they have been approximated through linear ones and experimental data confirmed this approximation is correct. Therefore, differently from other works presented in the literature [B5], the equations implemented in the proposed virtual sensors are very simple:

(Eq.53) $$IAS_{VIR.1} = \left(A_{01} + A_{11} \frac{\delta_F}{\delta_{F\,\max}} \right) \delta_S + \left(B_{01} + B_{11} \frac{\delta_F}{\delta_{F\,\max}} \right)$$

(Eq.54) $$IAS_{VIR.2} = \left(A_{02} + A_{12} \frac{\delta_F}{\delta_{F\,\max}} \right) \alpha + \left(B_{02} + B_{12} \frac{\delta_F}{\delta_{F\,\max}} \right)$$

where $\delta_{F\,max}$ and δ_F are maximum and current flap surface extension, respectively, whereas A_{01}, A_{02}, A_{11}, A_{12}, B_{01}, B_{02}, B_{11}, B_{12} are coefficients identified from the analysis of data gathered in dedicated flight tests, δ_S is stabilator surface deflection and α is the angle of attack.

4.3 Artificial Neural Network Virtual Sensors for Air Data Measurements

The basic principles and approaches to use ANN as Virtual Sensors has been presented in Section 3.3. Different ANNs can normally be used to generate synthetic measurements of the relevant air data variables on the basis of information provided by other sensors (GPS, AHRS, surfaces displacement sensors). In the considered case, Knowledge Discovery in Data Base Process is proposed for the training, validation and test phases of ANNs. In particular, the Cross-Industry Standard Process for Data Mining (CRISP-DM) is applied.

4.3.1 Data Mining Approach for Virtual Sensors based on Artificial Neural Network

Data Mining (DM) refers to extraction or "mining" knowledge from large amounts of data. In particular, DM – also called "Knowledge Discovery in Database" – is the process of automatically discovering useful knowledge in large data repositories. DM draws upon ideas, such as sampling, estimation, and hypothesis testing from statistics and search algorithms, modeling techniques, and learning theories from artificial intelligence, pattern recognition, and machine learning. Mainly DM tasks are:

- predictive tasks: the goal is to predict the value of a particular attribute (commonly known as the target or dependent variable) based on the values of other attributes (known as independent variables). This task is accomplished through the application of classification algorithms (i.e. Neural Networks, Inductive Decision Trees, Bayesian Nets, Support Vector Machines, …).

- descriptive tasks: the goal is to derive patterns (correlations, trends, clusters, anomalies, …) that summarize the underlying relationships in data, i.e. by applying statistical methods or cluster analysis approach.

Knowledge Discovery in Database (KDD) process will be carried out through the realization of the standard model process conceived from the Cross-Industry Consortium Standard Process for Data Mining (CRISP-DM). In the next Figure 4-2, a typical KDD process is described where DM is the algorithmic step [B49].

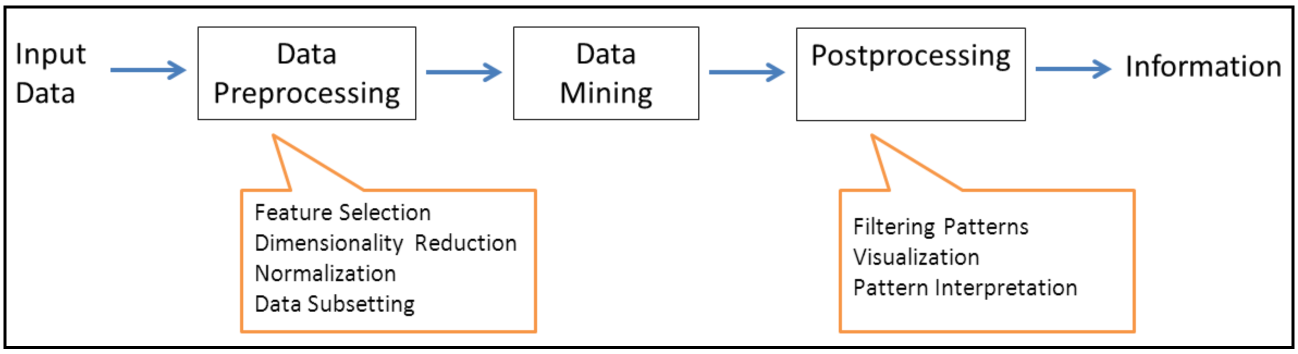


Figure 4-2 – Knowledge Discovery Process

DM approach can represent useful analysis methods for a lot of complex phenomena because it has the ability to work with many data described by a huge number of variables.

The classification techniques are addressed to build forecasting models for discrete variables based on other independent variables values as sketched in Figure 4-3.

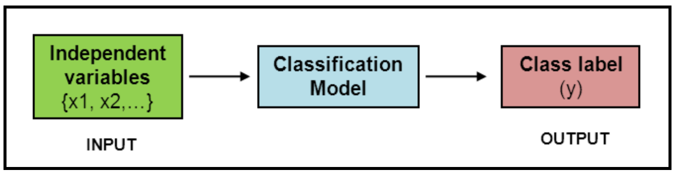


Figure 4-3 – Classification as the result of transforming independent variables in class label

A classification problem typically involves a training set of data already classified from which the classifier learns how to assign to a specific set of values of the input variables the class label of the target variable and also a test set of data to be used in order to evaluate classifier performance.

Classification model evaluation is made by a confusion matrix and an accuracy (performance metric) given by the ratio between the number of right previsions and the total number of previsions or, equivalently, the error rate, given by the ratio between the number of wrong previsions and the total number of previsions.

The classification model's main goal is to minimize the gap between target variable forecasted value and real value, or, equivalently, maximize the accuracy.

Many classification models are available and could be developed, i.e. Artificial Neural Networks (ANN), Inductive Decision Trees and Bayesian Networks. In particular, ANN are methodologies capable of establishing relationships between the independent variables (predictors) and the dependent variable (class target), through the experimentation of a multitude of situations and a net of interacting nodes. The interactions between the nodes are weighted and the weights are assigned by a training algorithm. Every ANN has one or more hidden layers. A decision tree is a graphical representation of all the possible alternatives by a tree like model and it can also be used as a prediction tool. Compared to Artificial Neural Networks, Decision Trees have the advantage of being easily readable and interpretable in new and useful knowledge. Finally, Bayesian Networks are directed acyclic graphs whose nodes represent random variables in the

Bayesian sense. Therefore, the nodes represent variables, while the connections represent their conditional dependencies.

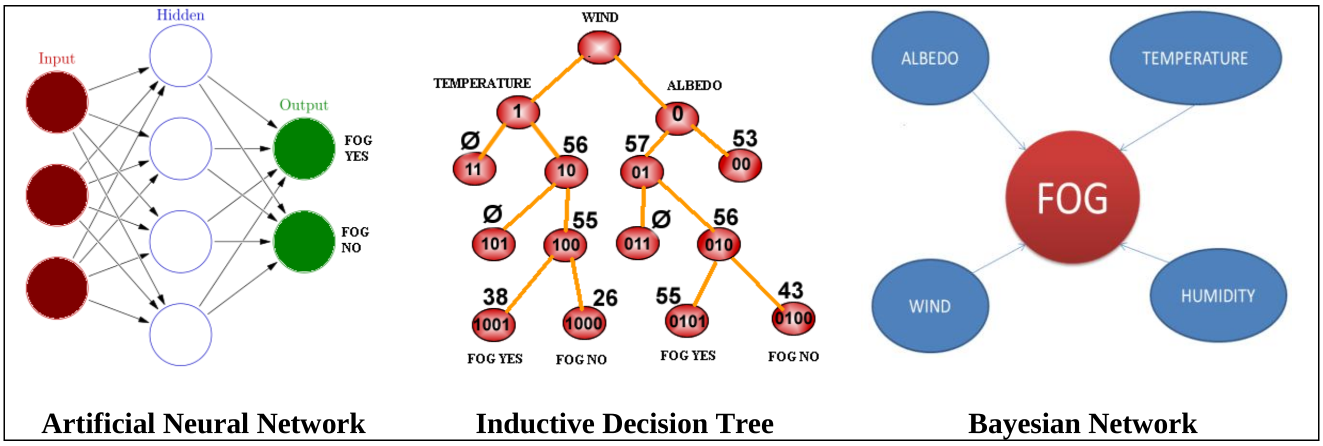


Figure 4-4: Data Mining Techniques

4.3.2 CRISP-DM

DM can be carried out through the realization of the KDD process according to the standard process conceived from the Cross-Industry Consortium Standard Process for Data Mining (CRISP-DM) [B50].

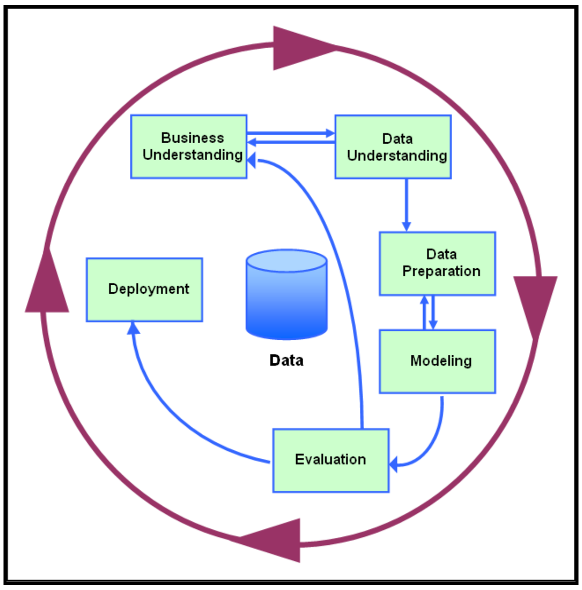


Figure 4-5 : Phases of the CRISP-DM reference model

This model consists of six phases intended as a cyclical process (see Figure 4-5):

Business Understanding. Business understanding includes determining business objectives, assessing the current situation, establishing DM goals, and developing a project plan.

Data Understanding. Once business objectives and the project plan are established, data understanding considers data requirements. This step can include initial data collection, data description, data exploration, and the verification of data quality. Data exploration such as viewing summary statistics (which includes the visual display of categorical variables) can occur at the end of this phase. Models such as cluster analysis can also be applied during this phase, with the intent of identifying patterns in the data. This phase is detailed presented in section 6.2.2.2.

Data Preparation. Once the data resources available are identified, they need to be selected, cleaned, built into the desired form, and formatted. Data cleaning and data transformation in preparation of data modeling needs to occur in this phase. Data exploration at a greater depth can be applied during this phase, and additional models utilized. This phase is detailed presented in section 4.3.2.1.

Modeling. DM software tools such as visualization (plotting data and establishing relationships) and cluster analysis (to identify which variables go well together) are useful for initial analysis. Tools such as generalized rule induction can develop initial association rules. Once greater data understanding is gained (often through pattern recognition triggered by viewing model output), more detailed models appropriate to the data type can be applied. The division of data into training and test sets is also needed for modeling.

Evaluation. Model results should be evaluated in the context of the business objectives established in the first phase. This will lead to the identification of other needs, frequently reverting to prior phases of CRISP-DM. Gaining business understanding is an iterative procedure in DM, where the results of various visualization, statistical, and artificial intelligence tools show the user new relationships that provide a deeper understanding of organizational operations.

Deployment. DM can be used to both verify previously held hypotheses, or for knowledge discovery (identification of unexpected and useful relationships). Through the knowledge discovered in the earlier phases of the CRISP-DM process, sound models can be obtained that may then be applied to business operations for many purposes, including prediction or identification of key situations. These models need to be monitored for changes in operating conditions, because what might be true today may not be true a year from now. If significant changes do occur, the model should be redone. It’s also wise to record the results of DM projects so documented evidence is available for future studies.

This six-phase process is not a rigid, by-the-numbers procedure. There’s usually a great deal of backtracking. Additionally, experienced analysts may not need to apply each phase for every study. But CRISP-DM provides a useful framework for DM and KDD.

The data understanding phase covers all activities to understand the statistical characteristic of all dataset, and usually a priori connection between the input data and the target variable is done, for finding a priori relation between all the variables. Usually the most easy technique is to use the correlation coefficient between the input and the output, but it is possible also to use techniques such as Independent Component Analysis (ICA) [B51] or information Gain method [B52].

4.3.2.1 Data Preparation

This phase covers all activities needed to collect the final dataset to be used in the subsequent Modeling Phase. Data preparation tasks include data cleaning, data integration and transformation, data and attribute selection and reduction. First of all, a clustering algorithm is applied in order to explore the available dataset and to select data dissimilar from one another. Clustering or Cluster Analysis divides data into groups (clusters).

Moreover clustering is a form of unsupervised learning whereby a set of observations (i.e., data points) is partitioned into natural groupings or clusters of patterns in such a way that the measure of similarity between any pair of observations assigned to each cluster minimizes a specified cost function. There is a plethora of clustering techniques to choose from. We have chosen to focus on the so-called K-means algorithm (Figure 4-6), because it is simple to implement, yet effective in performance, two features that have made it highly popular.

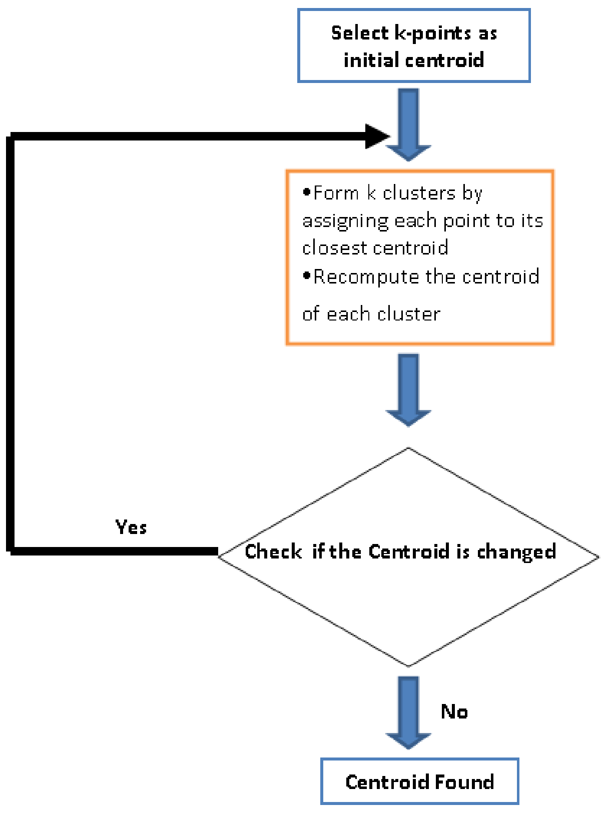

Select k-points as initial centroid
•Form k clusters by assigning each point to its closest centroid
•Recompute the centroid of each cluster
Check if the Centroid is changed
Yes
No
Centroid Found

Figure 4-6: K-Means Algorithm

Let $\{X_i\}_{i=1}^{N}$ denote a set of multidimensional observations that is to be partitioned into a proposed set of K clusters, where K is smaller than the number of observations, N. Let the relationship.

(Eq.55) $j = C(i) \quad i = 1, 2, \ldots, N$

denote a many-to-one mapper, called the *encoder*, which assigns the ith observation x_i to the jth cluster according to a rule yet to be defined. To do this encoding, we need a measure of similarity between every pair of vectors x_i and $x_{i'}$, which is denoted by $d(x_i, x_{i'})$.When the measure $d(x_i, x_{i'})$ is small enough, both x_i and $x_{i'}$ are assigned to the same cluster; otherwise, they are assigned to different clusters. To optimize the clustering process, we introduce the following cost function [B53]:

(Eq.56) $J(C) = \frac{1}{2}\sum_{j=1}^{K}\sum_{C(i)=j}\sum_{C(i')=j} d(x_i, x_{i'})$

For a prescribed K, the requirement is to find the encoder $C(i)=j$ for which the cost function $J(C)$ is minimized. At this point in the discussion, we note that the encoder C is unknown hence the functional dependence of the cost function J on C.

In K-means clustering, the squared Euclidean norm is used to define the measure of similarity between the observations x_i and $x_{i'}$, as shown by

(Eq.57) $$d\left(x_i, x_{i'}\right) = \left\| x_i - x_{i'} \right\|^2$$

Hence, substituting (Eq.57) into (Eq.56), we get

(Eq.58) $$J(C) = \frac{1}{2}\sum_{j=1}^{K} \sum_{C(i)=j} \sum_{C(i')=j} \left\| x_i - x_{i'} \right\|^2$$

We now make two points:

- The squared Euclidean distance between the observations x_i and $x_{i'}$ is *symmetric*; that is $\left\| x_{i'} - x_i \right\|^2 = \left\| x_i - x_{i'} \right\|^2$
- The inner summation in (Eq.58) reads as follows: For a given *i*, the encoder C assigns to cluster j all the observations that are closest to x_i. Except for a scaling factor, the sum of the observations so assigned is an *estimate of the mean vector* pertaining to cluster j; the scaling factor in question is $\frac{1}{N_j}$, where N_j is the number of data points within cluster j.

On account of these two points, we may therefore reduce (Eq.58) to the simplified form

(Eq.59) $$J(C) = \frac{1}{2}\sum_{j=1}^{K} \sum_{C(i)=j} \left\| x_i - \hat{\mu}_j \right\|^2$$

where $\hat{\mu}_j$ denotes the "estimated" mean vector associated with cluster j. In effect, the mean $\hat{\mu}_j$ may be viewed as the center of cluster j. In light of (Eq.59), we may now restate the clustering problem as follows:

Given a set of N observations, find the encoder C that assigns these observations to the K clusters in such a way that, within each cluster, the average measure of dissimilarity of the assigned observations from the cluster mean is minimized.

Indeed, it is because of the essence of this statement that the clustering technique described herein is commonly known as the K-means algorithm. For an interpretation of the cost function $J(C)$ defined in (Eq.59), we may say that, except for a scaling factor $\frac{1}{N_j}$, the inner summation in this equation is an estimate of the variance of the observations associated with cluster j for a given encoder C, as shown by

$$\text{(Eq.60)} \qquad \sigma_j^2 = \sum_{C(i)=j} \left\| x_i - \hat{\mu}_j \right\|^2$$

Accordingly, we may view the cost function $J(C)$ as a measure of the total cluster variance resulting from the assignments of all the N observations to the K clusters that are made by encoder C.

With encoder C being unknown, how do we minimize the cost function $J(C)$? To address this key question, we use an iterative descent algorithm, each iteration of which involves a two-step optimization. The first step uses the nearest neighbour rule to minimize the cost function $J(C)$ of (Eq.60) with respect to the mean vector $\hat{\mu}_j$ for a given encoder C. The second step minimizes the inner summation of (Eq.60) with respect to the encoder C for a given mean vector $\hat{\mu}_j$.This two-step iterative procedure is continued until convergence is attained.

Thus, in mathematical terms, the K-means algorithm proceeds in two steps:

- **Step 1**. For a given encoder C, the total cluster variance is minimized with respect to the assigned set of cluster means $\{\hat{\mu}_j\}_{i=1}^{K}$; that is, we perform, the following minimization:

$$\text{(Eq.61)} \qquad \min_{\{\hat{\mu}_j\}_{i=1}^{K}} \sum_{j=1}^{K} \sum_{C(i)=j} \left\| x_i - \hat{\mu}_j \right\|^2 \quad \text{for a given } C$$

- **Step 2.** Having computed the optimized cluster means in step 1, we next optimize the encoder as follows:

(Eq.62) $$C(i) = \arg\min_{1 \le j \le K} \left\| x(i) - \hat{\mu}_j \right\|^2$$

Starting from some initial choice of the encoder C, the algorithm goes back and forth between these two steps until there is no further change in the cluster assignments.

Each of these two steps is designed to reduce the cost function $J(C)$ in its own way; hence, convergence of the algorithm is assured. However, because the algorithm lacks a global optimality criterion, the result may converge to a local minimum, resulting in a suboptimal solution to the clustering assignment. Nevertheless, the algorithm has practical advantages:

1. The K-means algorithm is computationally efficient, in that its complexity is linear in the number of clusters.

2. When the clusters are compactly distributed in data space, they are faithfully recovered by the algorithm.

One last comment is in order: To initialize the K-means algorithm, the recommended procedure is to start the algorithm with many different random choices for the means for the proposed size K and then choose the particular set for which the double summation in (Eq.59) assumes the smallest value [B53].

In 1965 Cover [B54] proposed a statement in computational learning theory and is one of the primary theoretical motivations for the use of non-linear kernel methods in machine learning applications. The theorem states that given a set of training data that is not linearly separable, one can with high probability transform it into a training set that is linearly separable by projecting it into a higher-dimensional space via some non-linear transformation.

The K-means algorithm applies a nonlinear transformation to the input signal x. We say so because the measure of dissimilarity namely, the squared Euclidean distance $\left\| x - x_j \right\|^2$, on which it is based is a nonlinear function of the input signal x for a given cluster center x_j. Furthermore, with each cluster discovered by the K-means algorithm defining a particular computational unit in the hidden layer, it follows that if the number of clusters, K, is large enough, the K-means algorithm will satisfy the other requirement of Cover's theorem that is, that the dimensionality of the hidden layer is high enough. We therefore conclude that the K-means algorithm is indeed computationally powerful enough to transform a set of nonlinearly separable patterns into separable ones in accordance with this theorem.

After the data selection through the K-means algorithm, it is necessary to select the data set to use in the Modeling and Evaluation phase of the CRISP-DM process.

Cross-validation, sometimes called rotation estimation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, the model is trained with a certain dataset (training dataset) for which the expected results are known. After this training phase the model is validated by comparing the results related to a new dataset of “first seen” data (testing dataset) and the known expected results.

The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like over fitting and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

4.3.2.1.1 k-*Fold cross-validation*

In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general k remains an unfixed parameter.

When k=n (the number of observations), the k-fold cross-validation is exactly the leave-one-out cross-validation.

In stratified k-fold cross-validation, the folds are selected so that the mean response value is approximately equal in all the folds. In the case of a dichotomous classification, this means that each fold contains roughly the same proportions of the two types of class labels.

4.3.2.1.2 2-fold cross-validation

This is the simplest variation of k-fold cross-validation. Also, called holdout method [B55]. For each fold, we randomly assign data points to two sets d0 and d1, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two). We then train on d0 and test on d1, followed by training on d1 and testing on d0.

This has the advantage that our training and test sets are both large, and each data point is used for both training and validation on each fold

Cross validation is a model evaluation method that is better than residuals. The problem with residual evaluations is that they do not give an indication of how well the learner will do when it is asked to make new predictions for data it has not already seen. One way to overcome this problem is to not use the entire data set when training a learner. Some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on "new" data. This is the basic idea for a whole class of model evaluation methods called cross validation.

The holdout method is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set (it has never seen these output values before). The errors it makes are accumulated as before to give the mean absolute test set error, which is used to evaluate the model. The advantage of this method is that it is usually preferable to the residual method and takes no longer to compute. However, its evaluation can have a high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different depending on how the division is made.

4.4 Fault Tolerant Architecture

4.4.1 Introduction

System reliability requirements derive from the function criticality level and maximum exposure time. A flight-critical function is one whose loss might result in the loss of the aircraft itself, and possibly the persons on-board as well. In the latter case, the system is termed safety-critical. Here, a distinction can be made between a civil transport, where flight-critical implies safety-critical, and a combat aircraft. The latter admits to the possibility of the crew ejecting from an unflyable aircraft, so its system reliability requirements may be lower. A mission-critical function is one whose loss would result in the compromising or aborting of an associated mission. For avionics systems, a higher cost usually associates with the loss of an aircraft than with the abort of a mission (an antimissile mission to repel a nuclear weapon could be an exception). Thus, a full-time flight-critical system would normally pose much more demanding reliability requirements than a flight-phase mission-critical system. Next, the system reliability requirements coupled with the marginal reliabilities of system components determine the level of redundancy to ensure fault survivability, i.e., the minimum number of faults that must survive. To ensure meeting reliability requirements then, the evolution of a fault-tolerant design must be based on an interplay between design configuration commitments and substantiating analyses. For civil transport aircraft, the level of redundancy for a flight-phase critical function like automatic all-weather landing is typically single fail-operational, meaning that the system should remain operable after any potential single elemental failure. Alternatively, a full-time critical function like fly-by-wire primary flight controls is typically double fail-operational. It should be noted that a function that is not flight-critical itself can have failure modes that threaten safety of flight. In the case of active controls to alleviate structural loads due to gusts or manoeuvring, the function would not be critical where the purpose is merely to reduce structural fatigue effects. If the associated flight control surfaces have the authority during a hardover or runaway fault case to cause structural damage, then such a failure mode is safety-critical. In such cases, the failure modes must be designed to be fail-passive, which precludes any active mode failure effect like a hardover control surface. A failure mode that exhibits active behaviour can still be failsafe, however, if the rate and severity of the active effects are well within the flight crews capability to manage safely.

4.4.2 Design Approach

It is virtually impossible to design a complex avionics system that will tolerate all possible faults. Faults can include both permanent and transient faults, hardware and software faults, and they can occur singularly or concurrently. Timing faults directly trace to the requirement of real-time response within a few milliseconds, and may be attributable to both hardware and software contributions to data latency, as well as incorrect data.

The implementation of fault tolerance entails increased system overhead, complexity, and validation challenges. The overhead, which is essentially increased system resources and associated management activities, lies in added hardware, communications, and computational demands. The expanded complexity derives from more intricate connectivity and dependencies among both hardware and software elements; the greater number of system states that may be assumed; and appreciably more involved logic to manage the system. Validation challenges are posed by the need to identify, assess, and confirm the capability to sustain system functionality under a broad range of potential fault cases. Hence, the design approach taken for fault-tolerant avionics must attain a balance between the costs incurred in implementing fault tolerance and the degree of dependability realized.

Design approach encompasses both system concepts, development methodology, and fault tolerance elements. The system concepts derive largely from the basic fault-tolerance options introduced in the beginning of the section, with emphasis on judicious combinations of features that are adapted to given application attributes. The development methodology reduces to mutually supportive assurance-driven methods that propagate consistency, enforce accountability, and exact high levels of certitude as to system dependability. Fault tolerance design elements tend to unify the design concepts and methods in the sense of providing an orderly pattern of system organization and evolution. These fault tolerance elements, which in general should appear in some form in any fault-tolerant system, are:

- ***Error Detection*** — recognition of the incidence of a fault
- ***Damage Assessment*** — diagnosis of the locus of a fault
- ***Fault Containment*** — restriction of the scope of effects of a fault

- ***Error Recovery*** — restoration of a restartable error-free state
- ***Service Continuation*** — sustained delivery of system services
- ***Fault Treatment*** — repair of fault

A fundamental design parameter that spans these elements and constrains their mechanization is that of the granularity of fault handling. Basically, the detection, isolation, and recovery from a fault should occur at the same level of modularity to achieve a balanced and coherent design. In general, it is not beneficial or justified to discriminate or contain a fault at a level lower than that of the associated fault-handling boundary. There may be exceptions, however, especially in the case of fault detection, where a finer degree of granularity may be employed to take advantage of built-in test features or to reduce fault latency.

Depending on the basis for its instigation, fault containment may involve the inhibition of damage propagation of a physical fault and/or the suppression of an erroneous computation. Physical fault containment has to be designed into the hardware, and software error-state containment has to be designed into the applications software. In most cases, an erroneous software state must be corrected because of applications program discrepancies introduced during the delay in detecting a fault. This error recovery may entail resetting certain data object values and backtracking a control flow path in an operable processor. At this point, the readiness of the underlying architecture, including the coordination of operable components, must be ensured by the infrastructure. Typically, this activity relies heavily on system management software for fault tolerance. Service continuation, then, begins with the establishment of a suitable applications state for program restart. In an avionics system, this sequence of fault tolerance activities must take place rather quickly because of real-time functional demands. Accordingly, an absolute time budget must be defined, with tolerances for worst-case performance, for responsive service continuation.

4.4.3 Redundancy Options

Fault tolerance is usually based on some form of redundancy to extend system reliability through the invocation of alternative resources. The redundancy may be in hardware, software, time, or combinations thereof. There are three basic types of redundancy in hardware and software: static, dynamic, and hybrid. Static redundancy masks faults by taking a majority of the results from replicated tasks. Dynamic redundancy takes a two-step procedure for detection of, and recovery from faults. Hybrid redundancy is a combination of static and dynamic redundancy [B56].

A broad range of redundancy implementation options exist to mechanize desired levels and types of fault tolerance. Figure 4-7 presents a taxonomy of redundancy options that may be invoked in appropriate combinations to yield an encompassing fault tolerance architecture. First of all a lot of these techniques are applied to hardware components, but with the developing of virtual sensors , these architecture can be applied with analytical redundancy.

As indicated in Figure 4-7 ,the three categories of fault-tolerant architectures are masking, reconfiguration, and hybrid.

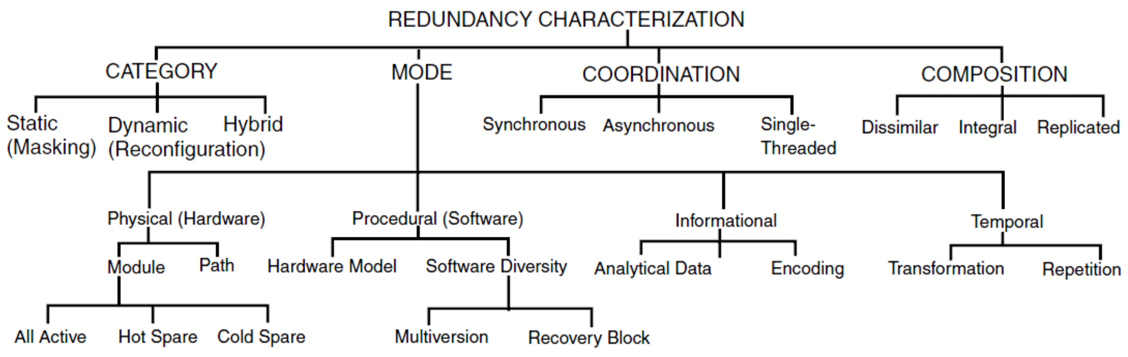


Figure 4-7: Redundancy classification [B57]

4.4.3.1 Fault Masking

The masking approach is classical per von Neumann's triple modular redundancy (TMR) concept, which has been generalized for arbitrary levels of redundancy. The TMR concept centers on a voter that, within a spares exhaustion constraint, precludes a faulted signal from proceeding along a signal path. The approach is passive in that no reconfiguration is required to prevent the propagation of an erroneous state or to isolate a fault.

Modular avionics systems consisting of multiple identical modules and a voter require a trade-off of reliability and safety. A "module" is not constrained to be a hardware module; a module represents an entity capable of producing an output. When safety is considered along with reliability, the module design affects both safety and reliability. It is usually expected that reliability and safety should improve with added redundancy. If a module has built-in error detection capability, it is possible to increase both reliability and safety with the addition of one module providing input to a voter. If no error detection capability exists at the module level, at least two additional modules are required to improve both reliability and safety. An error control arbitration strategy is the function implemented by the voter to decide what is the correct output, and when the errors in the module outputs are excessive so that the correct output cannot be determined, the voter may put out an unsafe signal. Reliability and safety of an n-module safe modular redundant (nSMR) depend on the individual module reliability and on the particular arbitration strategy used. No single arbitration strategy is optimal for improving both reliability and safety. Reliability is defined as the probability the voter's data output is correct and the voter does not assert the unsafe signal. Safety Reliability plus the probability the voter asserts the unsafe signal. As system reliability and safety are interrelated, increasing system reliability may result in a decrease in system safety, and vice versa [B58]. Voters that use bit-for-bit comparison have been employed when faults consist of arbitrary behaviour on the part of failed components, even to the extreme of displaying seemingly intelligent malicious behaviour [B59]. Such faults have been called Byzantine faults. Requirements levied on an architecture tolerant of Byzantine faults (referred to as Byzantine-resilient [BR]) comprise a lower bound on the number of fault containment regions, their connectivity, their synchrony, and the utilization of certain simple information exchange protocols. No a priori assumptions about component behaviour are required when using bit-for-bit comparison. The dominant contributor to failure of correctly designed BR system architecture is the common-mode failure.

Fault effects must be masked until recovery measures can be taken. A redundant system must be managed to continue correct operation in the presence of a fault. One approach is to partition the redundant elements into individual fault containment regions (FCRs). An FCR is a collection of components that operates correctly regardless of any arbitrary logical or electrical fault outside the region.

Masking faults and errors provides correct operation of the system with the need for immediate damage assessment, fault isolation, and system reconfiguration [B60].

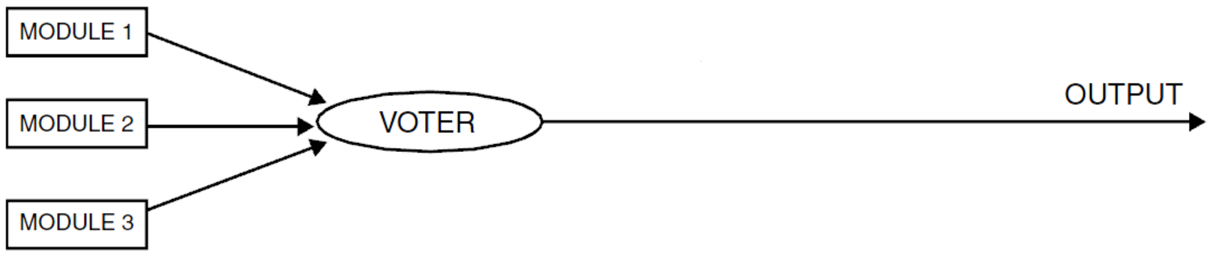


Figure 4-8: MASKING (Triple Modular Redundancy)

4.4.3.2 Reconfiguration

Hardware interlocks provide the first level of defence prior to reconfiguration or the use of the remaining non-faulty channels. In a triplex or higher redundancy system, the majority of channels can disable the output of a failed channel. Prior to taking this action, the system will determine whether the failure is permanent or transient.

Once the system determines a fault is permanent or persistent, the next step is to ascertain what functions are required for the remainder of the mission and whether the system needs to invoke damage assessment, fault isolation, and reconfiguration of the remaining system assets. The designer of a system required for long-duration missions may undertake to implement a design having reconfiguration capability

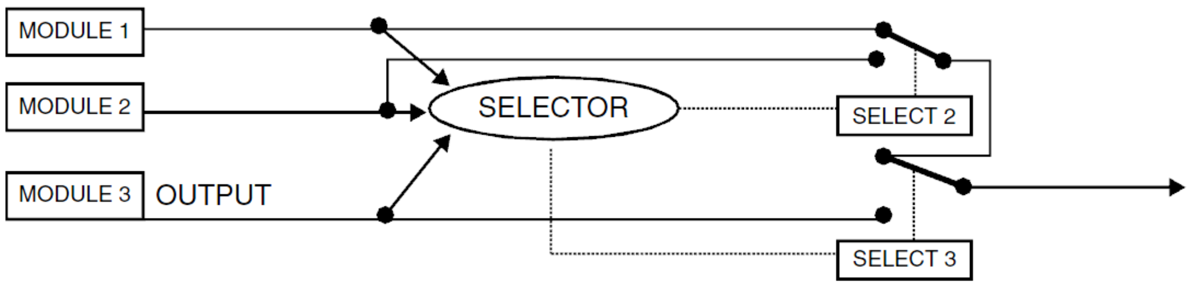


Figure 4-9: RECONFIGURATION (Triplex Redundancy)

4.4.3.3 Hybrid Fault Tolerance

Hybrid fault tolerance uses hybrid redundancy, which is a combination of static and dynamic redundancy, i.e., masking, detection, and recovery that may involve reconfiguration. A system using hybrid redundancy will have N-active redundant modules, as well as spare (S) modules. A disagreement detector detects if the output of any of the active modules is different from the voter output. If a module output disagrees with the voter, the switching circuit replaces the failed module with a spare. A hybrid (N,S) system cannot have more than $\frac{N-1}{2}$ failed modules at a time in the core, or the system will incorrectly switch out the good module when two out of three have failed.

Hybrid fault tolerance employs a combination of masking and reconfiguration. The intent is to draw on strengths of both approaches to achieve superior fault tolerance. Masking precludes an erroneous state from affecting system operation and thus obviating the need for error recovery. Reconfiguration removes faulted inputs to the voter so that multiple faults cannot defeat the voter. Masking and reconfiguration actions are typically implemented in a voter-comparator mechanism.

Figure 4-10 depicts a hybrid TMR arrangement with a standby spare channel to yield double fail operational capability. Upon the first active channel failure, it is switched out of the voter-input configuration, and the standby channel is switched in. Upon a second channel failure, the discrepant input to the voter is switched out. Only two inputs remain then, so a succeeding (third) channel failure can be detected but not properly be identified by the voter per se. With a voter that selects the lower of two remaining signals, and hence precludes a hardover output, a persistent signals disagreement results in a fail-passive loss of system function.

An alternative double-fail operational configuration would forego the standby channel switching and simply employ a quadruplex voter. This architecture is actually rather prevalent in dedicated flight-critical systems like fly-by-wire (FBW) flight control systems. This architecture still employs reconfiguration to remove faulty inputs to the voter.

The fault tolerance design elements described in the beginning are reflected in the fault-tolerant architecture in Figure 4-8 by way of annotations. For example, error detection is provided by the comparators; damage assessment is then accomplished by the reconfiguration logic using the various comparator states. Fault containment and service continuation are both realized through the voter, which also obviates the need for error recovery. Last, fault treatment is accomplished by the faulty path switching prompted be the reconfiguration logic. Thus, this simple example illustrates at a high level how the various aspects of fault tolerance can be incorporated into an integrated design.

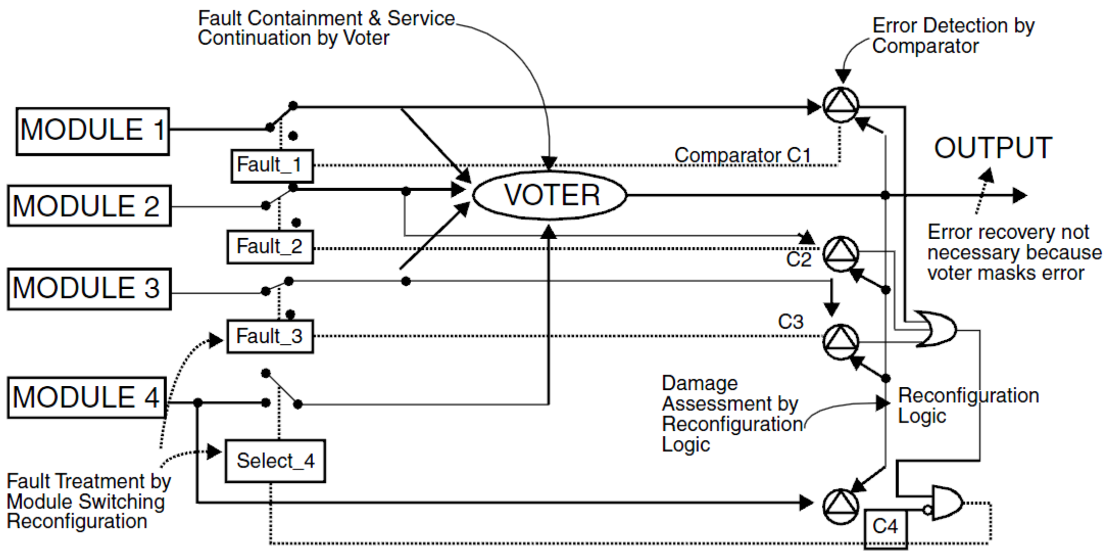

Fault Containment & Service Continuation by Voter
Error Detection by Comparator
MODULE 1
MODULE 2
MODULE 3
MODULE 4
Fault_1
Fault_2
Fault_3
Select_4
VOTER
Comparator C1
C2
C3
C4
OUTPUT
Error recovery not necessary because voter masks error
Damage Assessment by Reconfiguration Logic
Reconfiguration Logic
Fault Treatment by Module Switching Reconfiguration

Figure 4-10: Hybrid TMR arrangement

4.4.3.4 Voter Comparators

Voter comparators are very widely used in fault-tolerant avionics systems, and they are generally vital to the integrity and safety of the associated systems. Because of the crucial role of voter comparators, special care must be exercised in their development. These dynamic system elements, which can be implemented in software as well as hardware, are not as simple as they might seem. In particular, device integrity and threshold parameter settings can be problematic.

Certain basic elements and concerns apply over the range of voter-comparator variants. A conceptual view of a triplex voter-comparator is depicted in Figure 4-10. The voter here is taken to be a middle signal selector, which means that the intermediate level of three inputs is selected as the output. The voter section precedes the comparators because the output of the voter is an input to each comparator. Basically, the voter output is considered the standard of correctness, and any input signal that persists in varying too much from the standard is adjudged to be erroneous.

In Figure 4-10, the respective inputs to each of the signal paths is an amplitude-modulated pulse train, as is normal in digital processing. Each iteration of the voter is a separate selection, so each voter output is apt to derive from any input path. This is seen in Figure 4-11, where the output pulse train components are numbered per the input path selected at each point in time. At each increment of time, the voter output is applied to each of the comparators, and the difference with each input signal is fed to a corresponding amplitude threshold detector. The amplitude threshold is set so that accumulated tolerances are not apt to trip the detector. As shown here, the amplitude detector issues a set output when an excessive difference is first observed. When the difference falls back within the threshold, a reset output is issued.

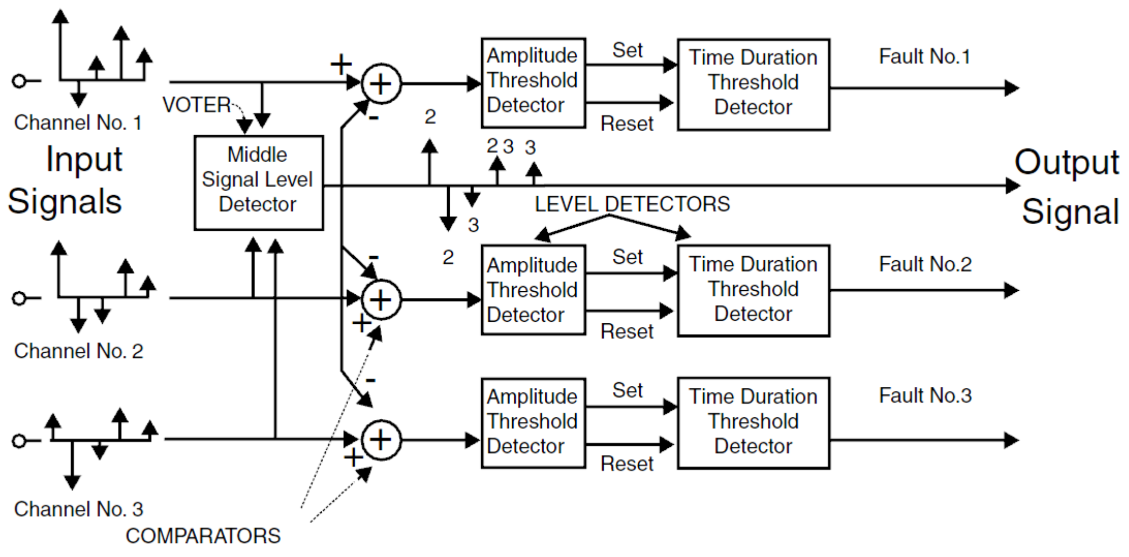


Figure 4-11: Triplex voter-comparator

Because transient effects may produce short-term amplitude detector trips, a timing threshold is applied to the output of each amplitude detector. Typically, a given number of consecutive out-of tolerance amplitude threshold trips are necessary to declare a faulty signal. Hence, a time duration threshold detector begins a count whenever a set signal is received, and in the absence of further inputs, increments the count for each sample interval thereafter. If a given cycle count is exceeded, then an erroneous state is declared and a fault logic signal is set for the affected channels. Otherwise, the count is returned to zero when a reset signal is received.

The setting of both the timing and amplitude thresholds is of crucial importance because of the trade-off between nuisance fault logic trips and slow response to actual faults. Nuisance trips erode user confidence in a system, their unwarranted trips can potentially cause resource depletion. On the other hand, a belated fault response may permit an unsafe condition or catastrophic event to occur. The allowable time to recover from a given type of fault, which is application-dependent, is the key to setting the thresholds properly. The degree of channel synchronization and data skewing also affect the threshold settings, because they must accommodate any looseness. The trade-off can become quite problematic where fast fault recovery is required.

Because the integrity and functionality of the system is at stake, the detailed design of a voter comparator must be subject to careful assessment at all stages of development. In the case of a hardware implemented device, its fault detection aspects must be thoroughly examined. Passive failures in circuitry that is not normally used are the main concern. Built-in test, self-monitoring, or fail-hard symptoms are customary approaches to device integrity. In the case of software-implemented voter comparators, their dependability can be reinforced through formal proof methods and in-service verified code.

4.4.3.5 Analytical Redundancy

A reversal check takes the outputs from a system and calculates what the inputs should have been to produce that output. The calculated inputs can then be compared with the actual inputs to check whether there is an error. Systems providing mathematical functions often lend themselves to reversal checks [B60].

Analytic redundancy using either of two general error detection methods, multiple model (MM) or generalized likelihood ratio (GLR), is a form of reversal check. Both methods make use of a model of the identified systems or through observers models.

Underlying both the GLR and MM methods is the issue of using system redundancy to generate comparison signals that can be used for the detection of failures. The fundamental idea involved in finding comparison signals is to use system redundancy, i.e., known relationships among measured variables to generate signals that are small under normal operation and which display predictable patterns when particular anomalies develop. All failure detection is based on analytical relationships between sensed variables, including voting methods, which assume that sensors measure precisely the same variable. The trade-off using analytical relationships is that we can reduce hardware redundancy and maintain the same level of fail-operability. In addition, analytical redundancy allows extracting more information from the data, permitting detection of subtle changes in system component characteristics. On the other hand, the use of this information can cause problems if there are large uncertainties in the parameters specifying the analytical relationships [B61].

The second part of a failure detection algorithm is the decision rule that uses the available comparison signals to make decisions on the interruption of normal operation by the declaration of failures. One advantage of these methods is that the decision rule — maximize and compare to a threshold — are simple, while the main disadvantage is that the rule does not explicitly reflect the desired trade-offs. The Bayesian Sequential Decision approach, in which an algorithm for the calculation of the approximate Bayes decision, has exactly the opposite properties, i.e., it allows for a direct incorporation of performance trade-offs, but is extremely complex. The Bayes Sequential Decision requires to choose a stopping rule and terminal decision rule to minimize the total expected cost, and the expected cost that is accrued before stopping [B61].

4.5 ADS System Requirements and Design Constraints

The technologies developed within the PhD period have been deployed by means of a flying test bed consisting of a piloted aircraft, named FLARE, equipped with On-Board Avionics System and On-Ground Control Station, designed and integrated by CIRA. FLARE is able to allow experimentation about autonomous take-off and landing, mission automation and detect, sense & avoid. The design of a custom ADS was needed to have an autothrottle able to properly control the vehicle TAS during the autolanding execution. The performances requirements at the touch down for the autolanding maneuver are synthesized in **Table 4-1**. The derived autothrottle requirements are reported in Table 4-2.

Moreover, integrity of the air data estimation has been considered within the design of the custom ADS. In other terms the availability of the air data estimation with the desired accuracy must be continuously verified and disclosed, so that the autolanding decision making logic can activate the proper phases. Finally the algorithms for air data estimation must be executable in the onboard FCC and the available space is strictly limited to the required ADS primary sensors (listed in Table 4-3), without any redundancy. The subsequent requirements for a high performance ADS estimating True Air Speed, Pressure Altitude, Total Air Temperature, Angle of Attack and Sideslip Angle, are reported in Table 4-4.

Variable	**Allowed Range**	**Desired Successful Rate**
Longitudinal position along X runway axes (nominal touch down point respect to runway lower limit)	100 m	6σ
Lateral position along Y runway axes (nominal touch down point is the runway center line)	10 m	6σ
Maximum Load Factor	2 g	3σ
Maximum Ground Speed	28 m/s	3σ
Minimum True Air Speed	20.8 m/s (full flap)	3σ
Vertical Speed (nominal value 0.3 m/s)	0.1÷2 m/s	3σ
Maximum Lateral Speed	2 m/s	3σ
Maximum Roll Angle	±5 deg	3σ
Pitch Angle (nominal value 6 deg)	3÷10 deg	3σ
Heading Angle (nominal value equal to the runway direction)	±10 deg	3σ

Table 4-1- Autolanding requirements for performances at touch down

Variable	**Desired Value**	**Desired Successful Rate**
Maximum static error in response to a unit step	1%	3σ
Maximum RMS error deviations	5m/sec	3σ
Maximum overshoot in response to a unit step	20%	3σ
Minimum 3dB Band	1 rad/sec	N/A

Table 4-2 - Autothrottle derived requirements

Primary sensors
Differential pressure sensor
Static pressure sensor
Total Air Temperature sensor
Angle of Attack sensor
Sideslip Angle sensor

Table 4-3 – ADS Custom Primary Sensors

Variable	**Desired Value**
Maximum Latency Time to measure the Static Pressure, Differential Pressure and Total Air Temperature	60 ms
Static Pressure Range	[800 1100] mbar
Differential Pressure Range	[0 20] mbar
Temperature Range	[-50 50] °C
Angle of Attack Range	[-20 20] deg
Angle of Sideslip Range	[-20 20] deg
Static Pressure Accuracy	30 Pa
Differential Pressure Accuracy	20 Pa
Temperature Accuracy	1 °C
Angle of Attack Accuracy	0.5 deg
Sideslip Angle Accuracy	0.5 deg
Maximum Weight	1 kg

Table 4-4 – ADS Custom Requirements

4.5.1 Component choice and hardware architecture

Based on the above defined requirements, and also considering constraints on the equipment's cost, the following COTS sensors have been chosen:

- Static pressure model 144SC0811 BARO (PCB) **[**B62]
- Differential pressure model HCXM020D6V by SensorTechnics [B63]
- Temperature transmitter model WT109PT by SENSCA [B64]
- Temperature sensor PT100 class B [B65]
- Mini air data boom model 100400 by Space Age Control **[**B66]

The hardware architecture of the custom Air Data System is shown in Figure 4-12. The two pressure sensors are connected to a conditioning and filtering system, which filters data at 10 Hz and returns the output voltage from the transducers in the range ± 10 V, compatible with the analogic inputs of the FCC. The Air Data Computer (ADC) receives the measurements of temperature, static and differential pressure, aerodynamic angles. It hosts the ADS algorithms, which compute the air data measurements and perform sensors fault detection and isolation (FDI) by using analytical redundancy.

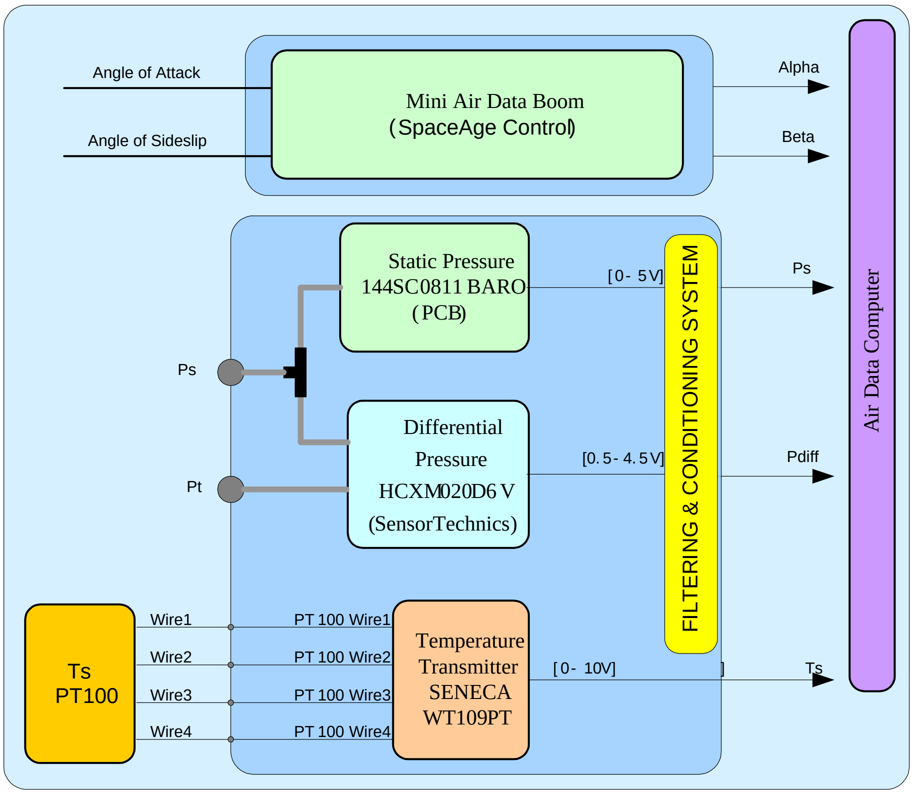


Figure 4-12: Air Data System Architecture

5 DEVELOPMENT OF VERIFICATION TOOLS

All the algorithms related to the CIRA GNC System and, in particular, that described in this dissertation has been developed following a well-defined development process cycle deepened in the following capitol (§6.1). The methodology adopted is the model-based design and the development of models to support the verification activities during the different phases of the process is crucial in the same way as the proper algorithms development. The several environments used for the verification stages are briefly described in the following paragraph and was developed by CIRA with the support of the author during the last fourteen years. The neural network approach detailed in specific paragraphs (e.g. §3.3 and §4.3) were developed in the framework of the author Phd and have been included in the publication [AR1], [AR2], [AR3] and [AR4].

5.1 Numeric Simulation Environment

The numeric simulation environment refers to a complete detailed model of the aeronautical flying demonstrator FLARE. This simulation model has been implemented in Matlab^{TM}/Simulink^{TM}, and can be considered representative of an UAV.

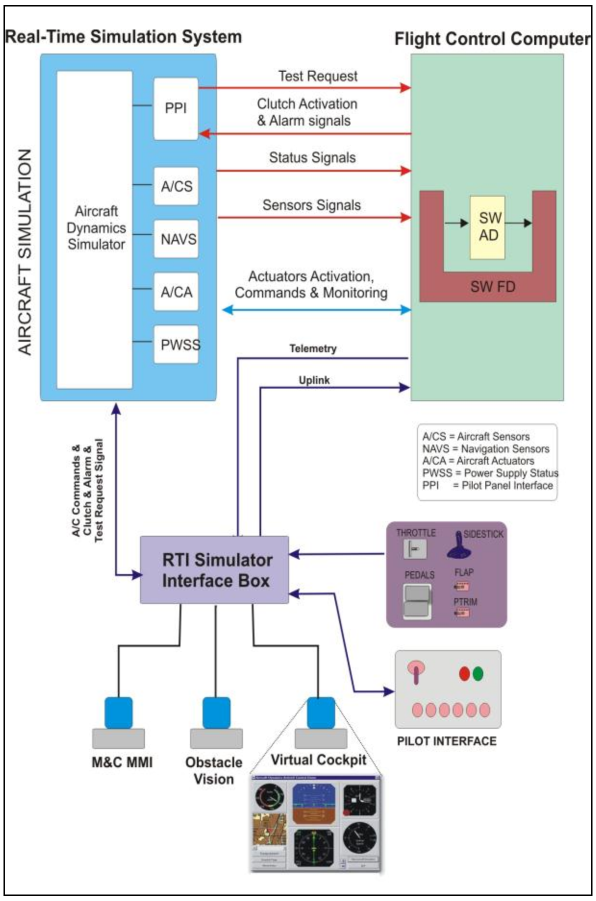

Real-Time Simulation System
Flight Control Computer
AIRCRAFT SIMULATION
Aircraft Dynamics Simulator
PPI
A/CS
NAVS
A/CA
PWSS
Test Request
Clutch Activation & Alarm signals
Status Signals
Sensors Signals
Actuators Activation, Commands & Monitoring
Telemetry
Uplink
SW AD
SW FD
A/CS = Aircraft Sensors
NAVS = Navigation Sensors
A/CA = Aircraft Actuators
PWSS = Power Supply Status
PPI = Pilot Panel Interface
A/C Commands & Clutch & Alarm & Test Request Signal
RTI Simulator Interface Box
THROTTLE
SIDESTICK
PEDALS
FLAP
PTRIM
PILOT INTERFACE
M&C MMI
Obstacle Vision
Virtual Cockpit

Figure 5-1 – Laboratory Set-up

The numeric simulation environment completely reproduces the laboratory test rig set-up, as depicted in Figure 5-1. It integrates the model of all the GNC algorithms and in general all the application SW downloaded in the Flight Control Computer used in the flight demonstrator. Moreover the numeric simulation environment also takes into account some peculiarities of the laboratory test rig such as HW signals filtering and serialization. Below a short description of main simulation modules is reported:

Aircraft-Dynamic-Simulator - This module includes the simulation model of FLARE. It includes: the 6DoF model of the aircraft [AR2], engine model, servo-actuation models, landing gear detailed model [B67], and external environment (atmosphere) including a Von Karman or Dryden model of turbulence. This module is configurable to set the turbulence level, type of servos (position or velocity controlled), injecting fixed wind disturbances, etc. Aerodynamics has been tuned using parameter identification from flight data, while mass and inertia data has been derived from internal avionic configuration and constructor data.

NAVS - This module contains the off-line models of on-board sensors used for both rotation and navigation and the sensors used by auxiliary aircraft systems. The first set entails: GPS, AHRS, ADS raw measurements and Laser Altimeter. This models includes dynamics, latency (where applicable) and measurement error models that can be configured before a simulation session. The second set entails: aero surface position sensors, engine sensors (rpm, temperature, etc.) landing gear sensors.

In addition, it has to be specified that in the NAVS block is specifically included also the *ADS_B sensor model*, which reproduces the functionalities of Obstacle Detection and Identification. This block is used for simulating the algorithms related to the function of Autonomous Collision Avoidance implemented in the CIRA GNC but not described in this thesis.

5.1.1 Identification Methods for Model Based Virtual Sensors

The design of a ADS by means of Model-Based Virtual Sensor methodology, requires high level of fidelity of simulation model. In fact, to properly design a flight control system and the related sensors model, an accurate aircraft model is necessary.

Fidelity of vehicle simulation models depends to a large extent on the accuracy of the aero-dynamic database representing the aircraft. In the majority of the cases an aerodynamic database derived from analytical predictions, wind tunnel measurements on a scaled model or extrapolation of existing data from similar configurations is incorporated. Such a priori aerodynamic databases, although valid over the entire flight envelope, are often associated with certain limitations of validity arising from, for example, model scaling, Reynolds number, dynamic derivative, and cross coupling effects. System identification methodology, evolved over the past decades, provides a powerful and sophisticated tool to identify from flight data aerodynamic characteristics valid over the entire operational flight envelope.
For conventionally stable flight vehicles, the most commonly and widely applied parameter estimation methods are regression analysis and output error method [AR2]. Regression analysis is based on linear models for the aerodynamic phenomena and requires error-free and compatible measurements. The output error method is applicable to general nonlinear systems and accounts for measurement noise. Parameter estimation needs to be followed by a step called model validation to asses model fidelity. The application of the output error method to FLARE vehicle,

followed by a validation of the identified model has been applied for the scope of the work of this thesis.

Data Mining Approach for Virtual Sensors based on Artificial Neural Network

In order to accomplish the Virtual Sensor software implementation, some specific suitable software development environment have been selected and used.

In particular, the Data Preparation phase, as required by the CRSP_DM approach, needs an algorithm to select the input attributes for the creation of neural models.

The MATLAB^{TM}/SIMULINK^{TM} software development environment and its toolboxes have been widely utilized both to obtain the Neural Network software (namely, the NN Toolbox has been used) and to implement the whole FDD architecture.

5.2 Laboratory Test Rig

In the framework of the projects connected with the UAV CIRA program, with the aim of performing a real-time ground validation of the on board segment SW, a test rig that simulates the on board system and the on ground system has been set up.

The laboratory experimental testing is needed to check correctness of software implementation, to solve software to hardware integration issues, to verify real time execution of algorithms, and to perform a final functional assessment of the overall system before going in flight.

The ADS testing process included both laboratory and in-flight tests. Laboratory tests were carried out using an Air Data Test Set in order to check, before flight, the correct implementation of the algorithms, the performances of the sensors setup and the correct integration into the FCC.

The figure below depicts a typical architecture for a test rig performing such laboratory testing [AR5].

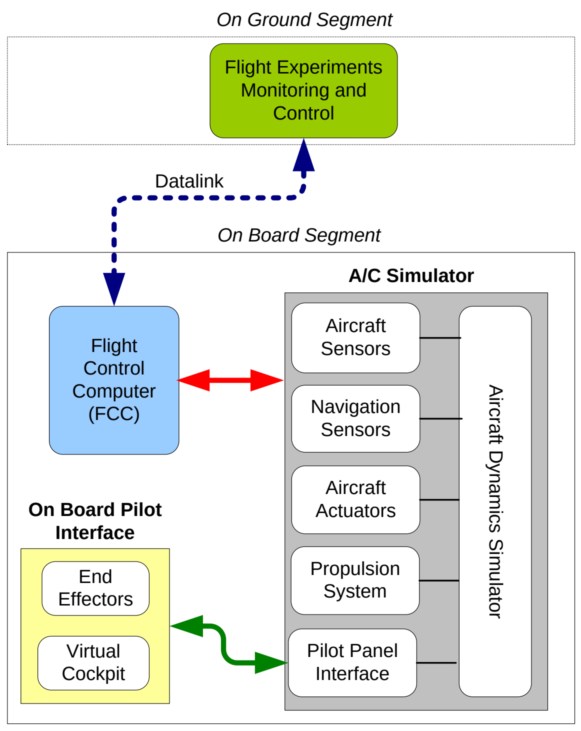


Figure 5-2: On Ground Validation Test Rig

The FCC of the test rig is the same Flight Control Computer used for the on board segment of the CIRA experimental vehicle. The rig contains also modules for the simulation of other relevant subsystems (such as data link), a ground control station (that might be a functional emulator or the actual ground control station) and a simulator workbench that is used for monitoring and configuring the real time simulator and the laboratory test execution. In particular the Pilot Panel Interfaces allows executing pilot-in-the-loop simulation whereas the pilot has the task of on-board safety pilot as in the real in-flight tests with FLARE.

5.3 Flying Demonstrator

Flight tests were executed for in-flight calibration of the ADS and final assessment. The CIRA aeronautical experimental platform FLARE (Flying Laboratory for Aerospace Researches) was used. It is a piloted flying test-bed, whose on board avionic system has been designed and integrated by CIRA. This flying platform is an experimental ultra-light aircraft with designed empty weight of 281 Kg, max take-off weight of 450 Kg, max speed s/l about 218 km/h, cruising speed about 190 km/h, wing area of 13.2 m2, wing span of 9.6 m and maximum engine power of 100 hp. The vehicle’s aerodynamic effectors are stabilator, ailerons and rudder, for longitudinal, lateral and directional control, respectively. Moreover a plain flap is used to increase lift during take-off and landing.

The on-board pilot has the task to put the aircraft in the condition foreseen for the flight test and to monitor the flight test overriding (with suitable available switches) in case of safety problems.

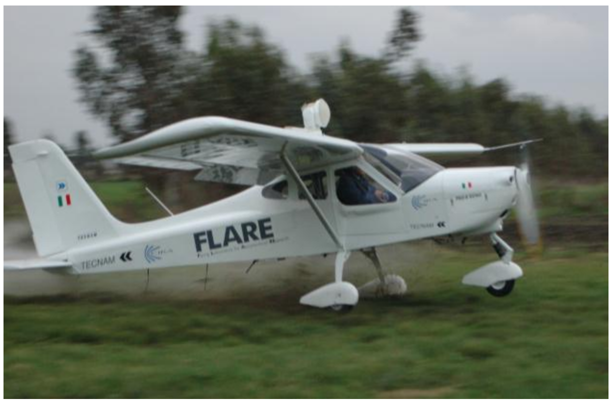


Figure 5-3 – CIRA's Aeronautical Flying Platform FLARE

This platform includes a Ground Control Station (GCS) installed in a big shelter fixed on the ground near the runway (see Figure 5-4). GCS is designed to manage telemetry data, present them to the flight test engineers through dedicated human-machine interfaces and for remote reconfiguration of the on board avionics system. GCS entails: a bidirectional datalink with an operating range of about 6 km, a meteorological station, a GPS base station, used to send differential correction to the on board GPS sensor, a virtual Cockpit HMI for presenting information in a pilot-like cockpit, an engineering HMI for presenting information in an engineering-like way, autonomous mid-air flight, automatic landing and automatic collision avoidance HMI for controlling flight experiments of such different algorithms.

Figure 5-4 – CIRA Aeronautical Ground Control Station for FLARE

The on board Avionics System of FLARE includes all devices needed to perform the in-flight experimental validation of advanced guidance, navigation and control functionalities. A brief description of such devices, integrated using Commercial-Off-The-Shelf (COTS) components, is below reported.

- A Flight Control Computer (FCC) based on a PowerPC processor integrated in modular HW architecture. The SW development environment for the FCC allows for automatic real-time coding directly from Simulink diagrams.
- A navigation sensor suite including a two DGPS-RTK L1/L2 system capable to provide position measurements with an accuracy of few centimetres, a solid state Attitude Heading Reference System (AHRS) with MEMS sensor technology and two dedicated sensors for distance to ground measurements respectively using radar and laser technologies. The altimeter sensors can be alternatively mounted because of weight limitations.
- A radar device installed on the top of the plane for obstacle detection.
- Digital electromechanical servos to command both aerodynamic surfaces and throttle, driven by the FCC via PWM signals.
- A digital bidirectional data link system able to exchange data between on board FCC and the Ground Control Station with a maximum bit-rate of 9.600 bit/sec in uplink and 115.200 bit/sec in downlink.

6 NUMERICAL AND EXPERIMENTAL ASSESSMENTS

This chapter presents numerical verification, ground validation phases results and relevant flight demonstrations, performed to assess effectiveness of the proposed system. All the tests were performed by CIRA in the framework of the aeronautical project TECVOL.

6.1 Development and Implementation Process

The design and test process adopted for the algorithms described in this thesis are the same ones adopted in the TECVOL project. The development phases have been performed by using a top-down process, usually well known as V-cycle (Figure 6-1). Actually the whole design process can be divided into three main phases, which have been identified according to the RTCA standard DO-178C [B68] & DO-331 [B69]: *requirement definition & system design*, *implementation & on-ground testing* and *integration & flight validation*. Each phase will be briefly described in the following paragraphs. This approach has remarkable advantages, especially on the quality of the final product, including:

- close correlation between control system specifications, SW implementation and related documentation;
- reduction of the code generation time;
- “strong” control over implementation and/or specification errors.

6.1.1 Requirement definition and System Design

During this phase, system and user requirements are defined, system architecture is delineated and the design of the control system is carried out through the use of simulation models (*Model Based Design*). The simulation models are developed in Matlab/Simulink environment. All requirements (from GN&C system requirements to equipment specifications) produced during the design activities are defined with unique identification tags, traced in both directions of development stages (downward and upward) and maintained during all the project life. All test reports are traced with respect to one or more requirements. Justification and compliance matrixes are provided at each stage of development giving evidence about analysis and tests performed to define each requirement or to verify design compliance. Each subsystem is validated through robustness and performance off-line analysis.

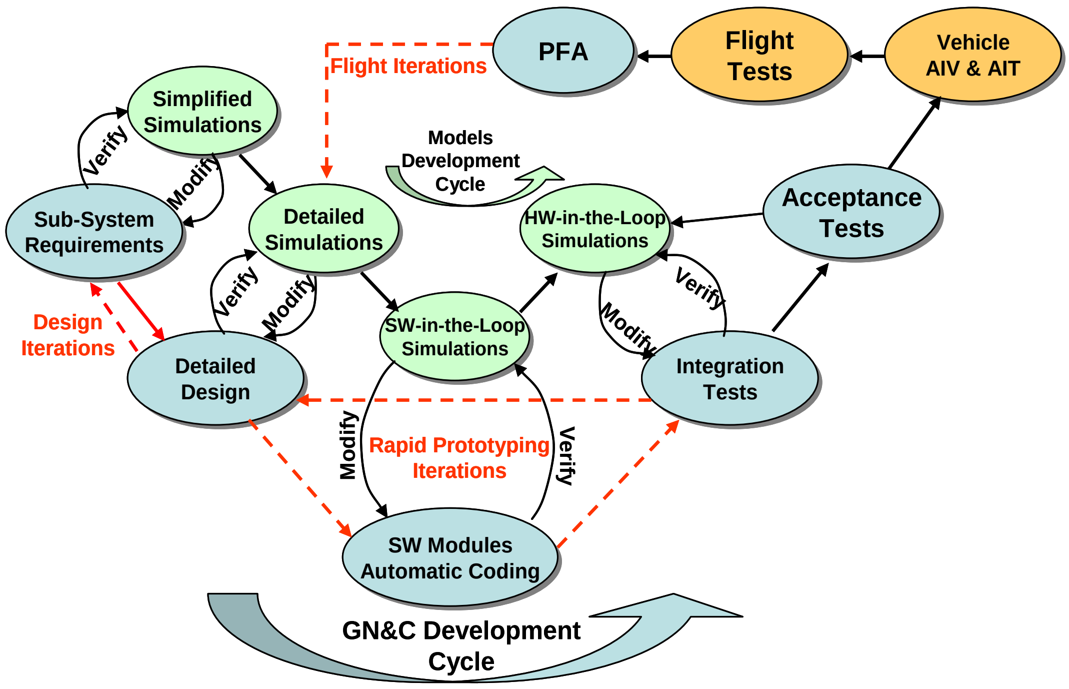


Figure 6-1 – GN&C Algorithm Development Cycle at CIRA

6.1.2 *Implementation & On-Ground Testing*

The GNC modules are designed directly by means of a Matlab/Simulink environment endowed with a specific tool (Embedded Coder) that allows the Automatic Program Building (APB) for a specific target machine. This approach has two main advantages:

- automation logics and control algorithms can be developed using a high level programming language;
- debugging can be easily done during both preliminary simulations, while defining the control strategy, and validation of the control system using HIL simulations.

During this phase the high level code is integrated with C/C++ code and downloaded to the target machine, which manages the resulting application according to its micro-kernel's primitives. Always during this phase, HIL simulations are performed, allowing validation and testing of the control system, interacting with both the simulated environment and the real instrumentation (feedback sensors, real Human Machine Interfaces, Airborne Virtual Cockpits, and Ground Control Stations). An interesting feature of this technique is the capability to monitor and/or to modify (using SW tools) the system parameters and the control strategy. In this way, the validation process and the control system fine tuning become easy and quick. This will allow executing rapid iterations among SW low level specification (Detailed Design), SW unit tests and integrated real time tests phases (*Rapid Prototyping Iterations*).

6.1.3 Integration and Flight Validation

This is the final development phase [AR5]. The GNC HW equipment is going to be integrated and GNC SW is going to be targeted and deployed in the host flight control computer. The GNC HW/SW equipment, finally, is going to be integrated in the aircraft and the system accepted after successful acceptance test sessions. During the flight tests analysis of data and events, comparison with expected results and parameter model identification used. The test allows performance validation,refinement of simulation models, increasing their prediction accuracy and/or reducing their level of uncertainty, and identification of eventual GN&C algorithm enhancements or needed modifications (due to, for example, a not satisfactory behaviour during flight). This further design iteration is executed in such projects where multiple missions are planned with increasing level of mission difficulty or risk. Finally, this iteration is used to gather the maximum information content from from the execution of a flight test in order to perform next missions with lower risks or with an higher level of difficulty and to finally increase know-how and experience for future projects.

With reference to the above development cycle, the activities described in this dissertation have been mainly focused on the design of fault tolerant algorithms related to ADS diagnostic through virtual sensors. Anyway, CIRA, supported by the author, has performed all other activities of above cycle to finally execute some flight tests with the above described flight demonstrator FLARE. The following paragraphs report the key results of these activities that demonstrate effectiveness of the proposed algorithms.

6.2 Numerical and Laboratory Assessments

As explained in Section 4, both Model-based and Neural network based virtual sensors have been designed and tested. Different techniques have been used with reference to different phases of flight or conditions, depending on which one is suitable for the specific application. In the following paragraphs the virtual sensor numerical assessment for the various virtual sensors are reported.

6.2.1 Model Based Numerical Assessments

In this paragraph, the assessment of the simple IAS virtual sensors implementation, which formulation is described in §4.2.2 is reported.

In the following figure you can find the results of two Lab tests where the Model Based Virtual Sensors outputs described by (Eq.53) and (Eq.54) and the simulated true IAS were compared. The difference has been considered satisfactory.

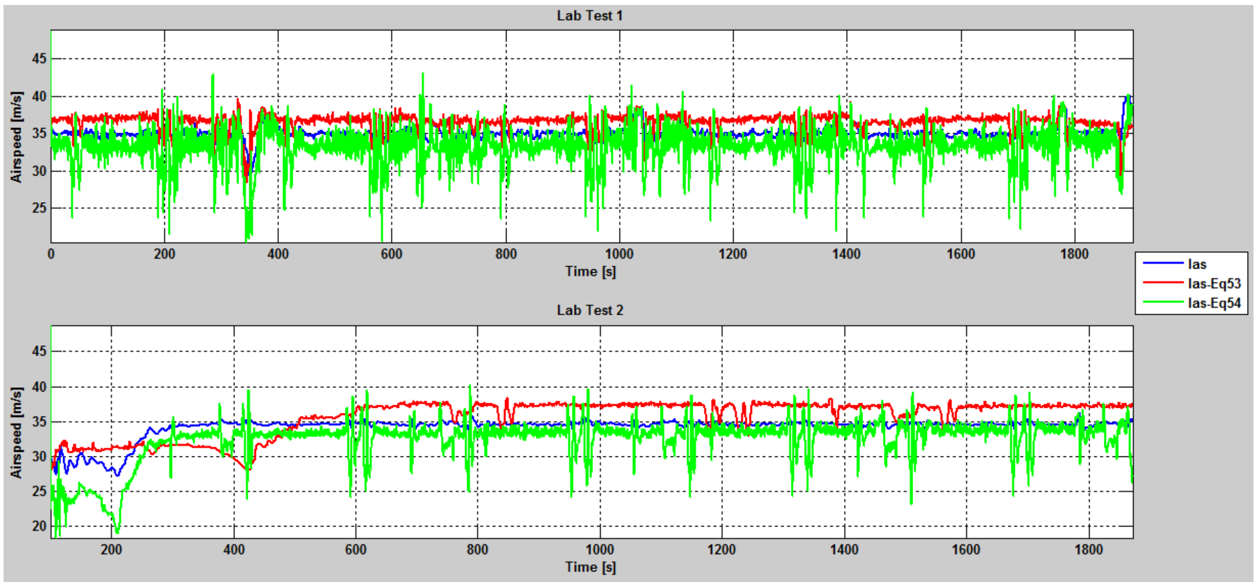


Figure 6-2 - Model Based Virtual Sensors Lab Assessment

6.2.2 Neural Network Numerical Assessments

Numerical assessment of the proposed ANN virtual sensors was carried out by means of the CRISP-DM methodology described in §4.3.2 and also tested with numeric simulator described in §5.1.

6.2.2.1 Business Understanding

The target is to deploy a fault tolerant architecture for the ADS installed on FLARE where there isn't any hardware redundancy. The architecture has been developed for the Indicated Airspeed and for the Angle of Attack. Obviously the same procedure can be adopted also for the True

Airspeed and for the Angle of Sideslip. FLARE has been flown for the last 8 years, but we consider only the last 29 flights, because the aircraft configuration has been changed from the beginning of the program and also the aerodynamic configuration, and it isn't usefully to use data from a different configuration. Using analytical redundancy it is necessary to define some statistical parameters for evaluating the performance, of a virtual measure referred as p and the real measure referred as a, which are:

- *Root Mean Squared Error* (RMSE): It measures the error rate of a regression model. It can only be compared between models whose errors are measured in the same units.

(Eq.63) $$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}}$$

- *Mean Absolute Error* (MAE): It has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units. It is usually similar in magnitude to RMSE, but slightly smaller.

(Eq.64) $$MAE = \frac{\sum_{i=1}^{n}|p_i - a_i|}{n}$$

- *Pearson Correlation Coefficent* (R) : It measures the statistical correlation between the a and p. It ranges from 1 for perfectly correlated results, through 0 where there is no correlation, to -1 when the results are perfectly correlated negatively.

(Eq.65)
$$R = \frac{S_{PA}}{\sqrt{S_P S_A}}$$
$$S_{PA} = \frac{\sum_{i=1}^{n}\left(p_i - \overline{p}\right)\left(a_i - \overline{a}\right)}{n-1}$$
$$S_P = \frac{\sum_{i=1}^{n}\left(p_i - \overline{p}\right)^2}{n-1} \quad S_A = \frac{\sum_{i=1}^{n}\left(p_i - \overline{p}\right)^2}{n-1}$$
$$\overline{p} = \frac{1}{n}\sum_{i=1}^{n} p_i \quad \overline{a} = \frac{1}{n}\sum_{i=1}^{n} a_i$$

The performance requirements are reported in Table 6-1.

Virtual Measure	RMSE	MAE	R
IAS	<1.5 m/s	<1 m/s	>0.9
AOA	<1.3 deg	<1 deg	>0.93

Table 6-1: Virtual measures requirements

6.2.2.2 Data Understanding.

This phase involves collecting data in order to become familiar with them, identify data quality problems and identify initial insights or interesting subsets. Data provided for analysis are constituted by 536750 records, representing the time histories of flight parameters related to 29 flights. The variables selection is based on physical considerations done on the aircraft dynamics equation [B48]; in fact all these variables influence the aircraft dynamics, and are shown in Table 6-2. The manner in which those variables influence the aircraft dynamics can be evaluated only if the aircraft model is known. The aircraft model that usually is not a priori known can be estimated through different techniques [AR2], [B70] and some uncertainties usually remain.

	VARIABLES	DESCRIPTION	UNIT
1	Stabilator	Mobile surface	[deg]
2	Aileron	Mobile surface	[deg]
3	Rudder	Mobile surface	[deg]
4	Flap	Mobile surface	[deg]
5	Throttle	Thrust Command	N/A
6	AOA	Angle of attack	[deg]
7	AOS	Angle of sideslip	[deg]
8	Nx	x load factor	[g]
9	Ny	y load factor	[g]
10	Nz	z load factor	[g]
11	p	Roll rate angle	[deg/s]
12	q	Pitch rate angle	[deg/s]
13	r	Yaw rate angle	[deg/s]
14	Phi	Roll attitude angle	[deg]
15	Theta	Pitch attitude angle	[deg]
16	Psi	Heading Veichle	[deg]
17	Vx	GPS x velocity	[m/s]
18	Vy	GPS y velocity	[m/s]
19	Vz	GPS z velocity	[m/s]
20	IAS	Indicated Airspeed	[m/s]
21	TAS	True Airspeed	[m/s]

Table 6-2. Variables and descriptions

In the following figures are reported the variables distribution; they confirm the used dataset is representative of the whole FLARE flight envelope.

Stabilator [deg]	
Mean	-5.362
Std Dev	1.590
Std Err Mean	0.002
Upper 95% Mean	-5.358
Lower 95% Mean	-5.367
N	536750

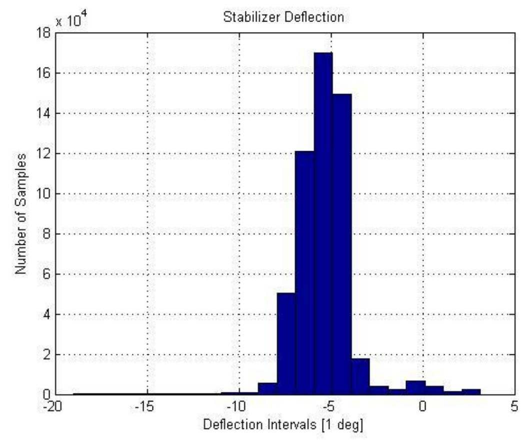


Figure 6-3 – Stabilator Deflection Statistical Data

AoA [deg]	
Mean	8.126
Std Dev	3.985
Std Err Mean	0.005
Upper 95% Mean	8.136
Lower 95% Mean	8.115
N	536750

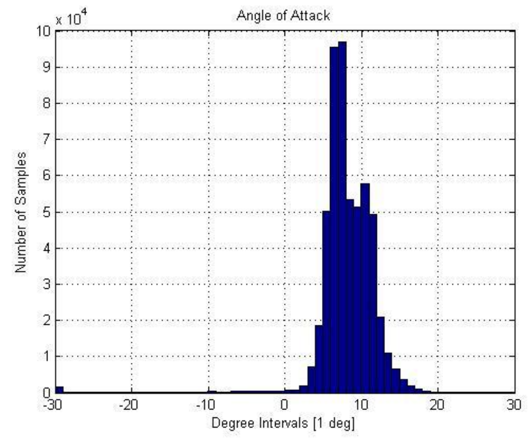


Figure 6-4 – Angle of Attack Statistical Data

Flap [deg]	
Mean	2.627
Std Dev	8.461
Std Err Mean	0.011
Upper 95% Mean	2.650
Lower 95% Mean	2.60
N	536750

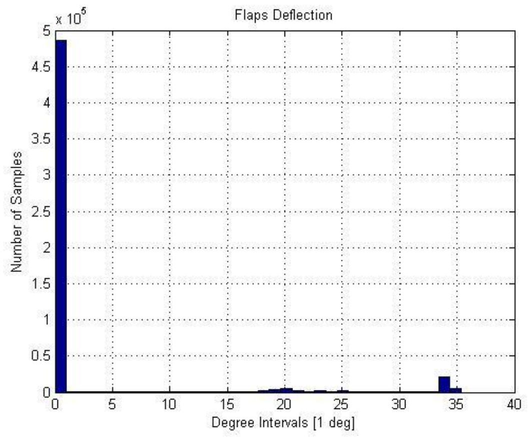


Figure 6-5 – Flap Deflection Statistical Data

Nx [m/s^2]

Mean	-0.081
Std Dev	0.0540
Std Err Mean	0.0000737
Upper 95% Mean	-0.0812
Lower 95% Mean	-0.0815
N	536750

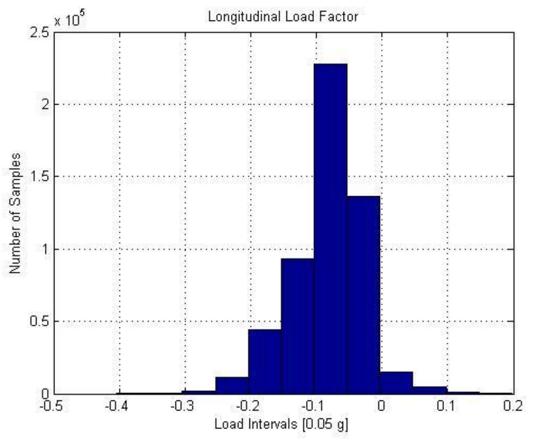


Figure 6-6 – X-Body Axis Load Factor Statistical Data

Ny [m/s^2]

Mean	-0.00113
Std Dev	0.0244
Std Err Mean	0.0000334
Upper 95% Mean	-0.00107
Lower 95% Mean	-0.001201
N	536750

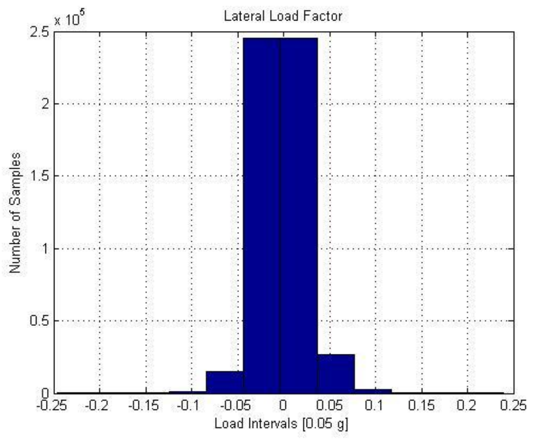


Figure 6-7 – Y-Body Axis Load Factor Statistical Data

Nz [m/s^2]

Mean	1.012
Std Dev	0.067
Std Err Mean	9.188e-5
Upper 95% Mean	1.012
Lower 95% Mean	1.012
N	536750

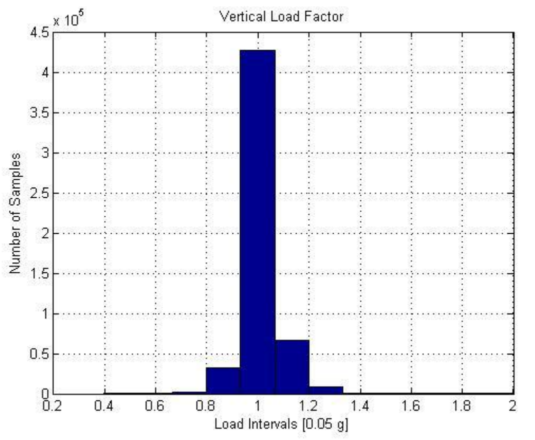


Figure 6-8 – Z-Body Axis Load Factor Statistical Data

Vx [m/s]	
Mean	-0.041
Std Dev	16.749
Std Err Mean	0.0222
Upper 95% Mean	0.0036
Lower 95% Mean	-0.086
N	536750

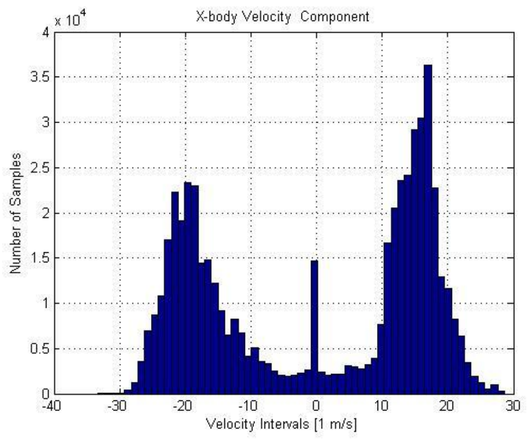


Figure 6-9 – Longitudinal Velocity Statistical Data

Vy [m/s]	
Mean	0.454
Std Dev	25.37
Std Err Mean	0.034
Upper 95% Mean	0.522
Lower 95% Mean	0.387
N	536750

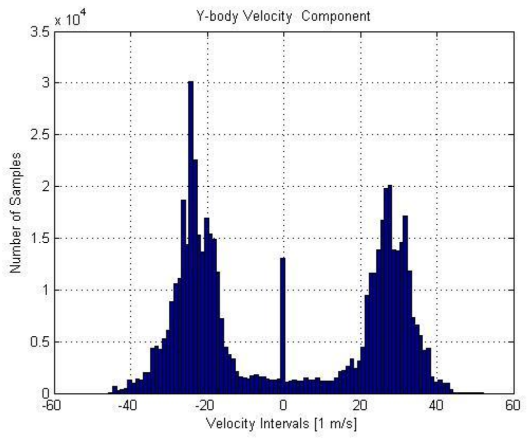


Figure 6-10 – Lateral Velocity Statistical Data

Vz [m/s]	
Mean	0.077
Std Dev	13.912
Std Err Mean	0.018
Upper 95% Mean	0.114
Lower 95% Mean	0.040
N	536750

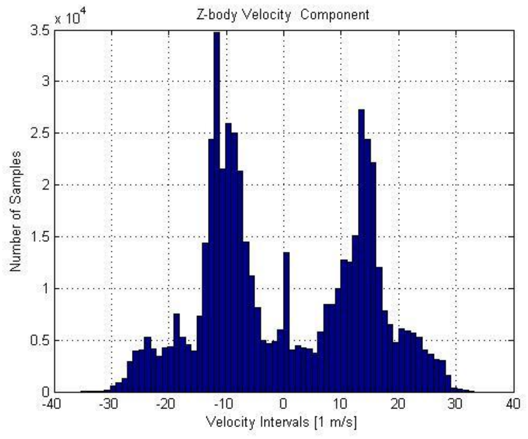


Figure 6-11 – Vertical Velocity Statistical Data

AoS [deg]	
Mean	0.587
Std Dev	3.040
Std Err Mean	0.004
Upper 95% Mean	0.595
Lower 95% Mean	0.579
N	536750

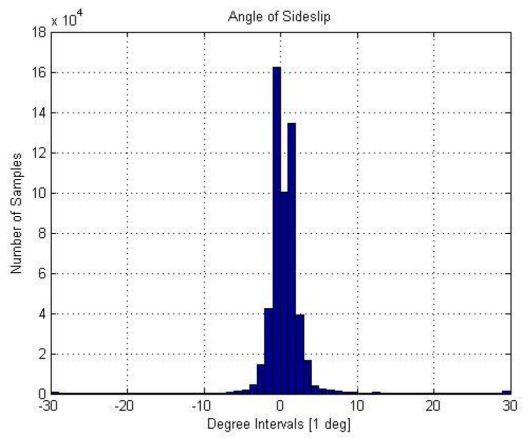


Figure 6-12 – Angle of Sideslip Statistical Data

Phi [deg]	
Mean	-0.316
Std Dev	9.282
Std Err Mean	0.0126
Upper 95% Mean	-0.291
Lower 95% Mean	-0.341
N	536750

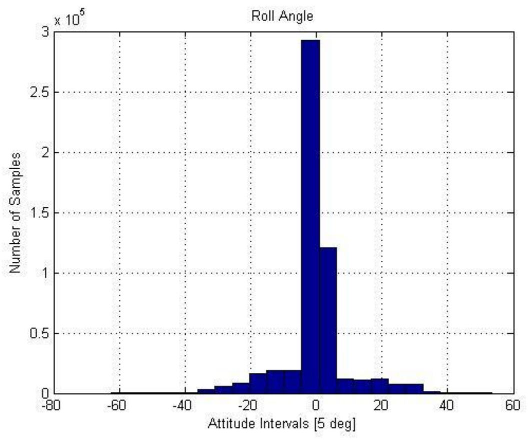


Figure 6-13 – Roll Angle Statistical Data

Theta [deg]	
Mean	4.399
Std Dev	2.99
Std Err Mean	0.004
Upper 95% Mean	4.407
Lower 95% Mean	4.391
N	536750

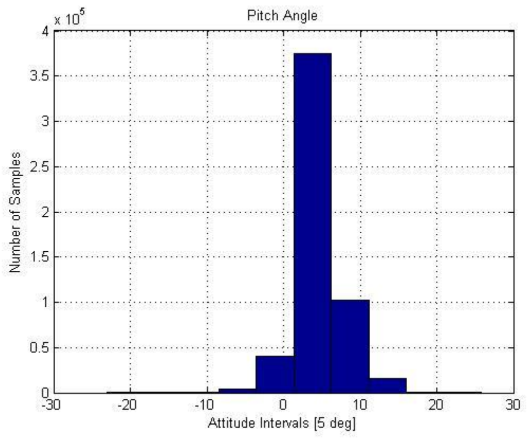


Figure 6-14 – Pitch Angle Statistical Data

Psi [deg]

Mean	-25.292
Std Dev	96.225
Std Err Mean	0.131
Upper 95% Mean	-25.035
Lower 95% Mean	-25.550
N	536750

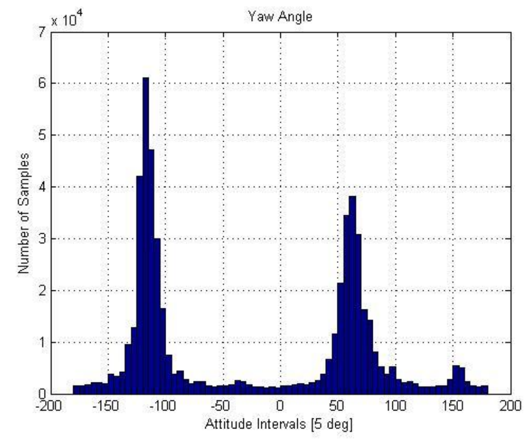


Figure 6-15 – Yaw Angle Statistical Data

Ias [m/s]

Mean	31.886
Std Dev	5.850
Std Err Mean	0.007
Upper 95% Mean	31.909
Lower 95% Mean	31.870
N	536750

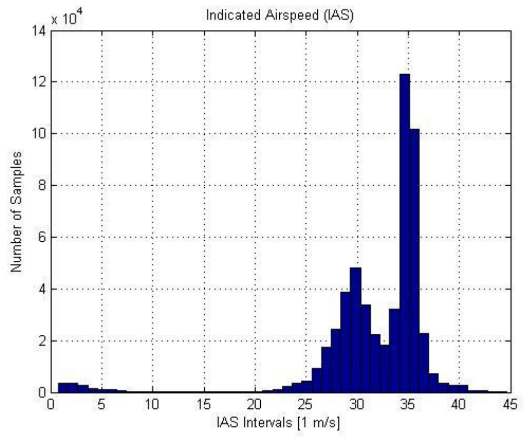


Figure 6-16 – Indicated Airspeed Statistical Data

Aileron [deg]

Mean	-1.6065
Std Dev	1.3413
Std Err Mean	0.001831
Upper 95% Mean	-1.61007
Lower 95% Mean	-1.60290
N	536750

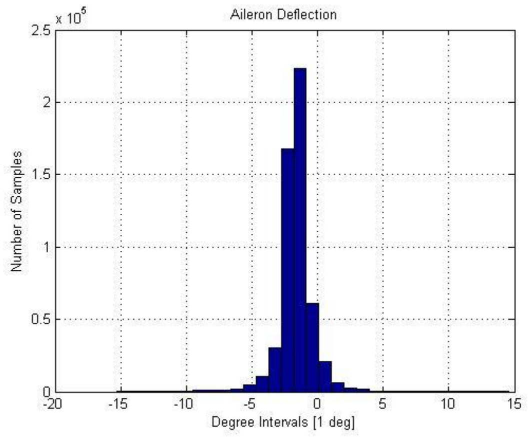


Figure 6-17 – Aileron Deflections Statistical data

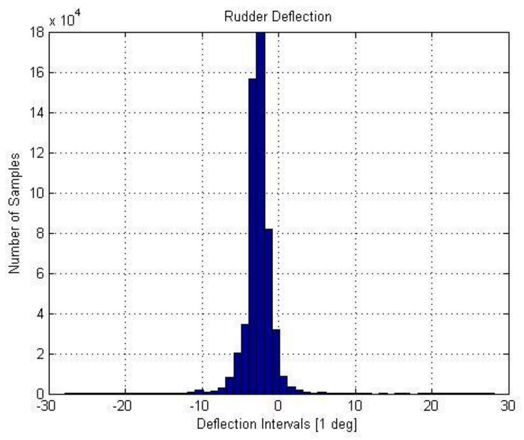
Rudder [deg]		
Mean	-2.6712	
Std Dev	1.6721	
Std Err Mean	0.002282	
Upper 95% Mean	-2.67571	
Lower 95% Mean	-2.66676	
N	536750	

Figure 6-18 – Rudder Deflections Statistical Data

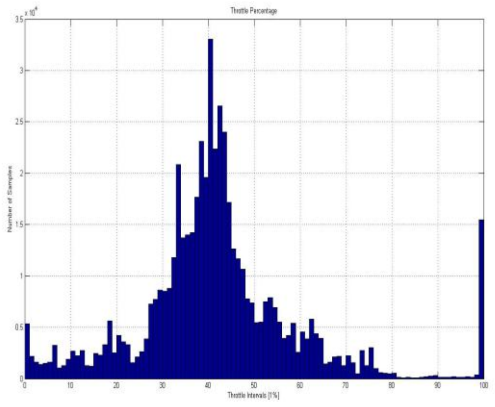
Throttle [%]		
Mean	0.4201	
Std Dev	0.1770	
Std Err Mean	0.000242	
Upper 95% Mean	0.41962	
Lower 95% Mean	0.42056	
N	536750	

Figure 6-19 – Throttle Displacements Statistical data

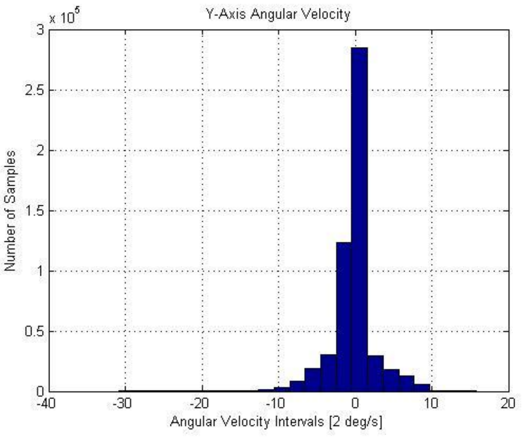
Pitch velocity [deg/s]		
Mean	-0.1227	
Std Dev	2.6520	
Std Err Mean	0.00362	
Upper 95% Mean	-0.12984	
Lower 95% Mean	-0.11565	
N	536750	

Figure 6-20 – Pitch Statistical Data

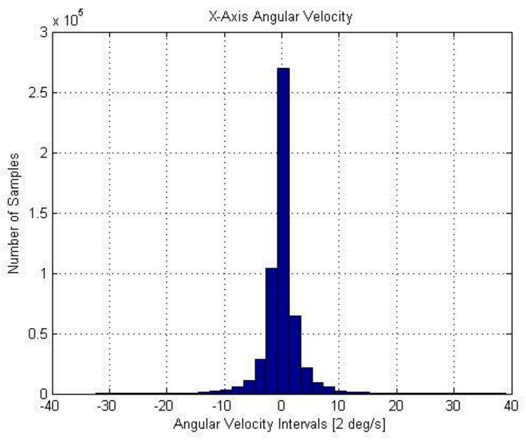
Roll velocity [deg/s]		
Mean	0.0207	
Std Dev	3.4705	
Std Err Mean	0.004737	
Upper 95% Mean	0.01145	
Lower 95% Mean	0.03002	
N	536750	

Figure 6-21 – Roll Statistical Data

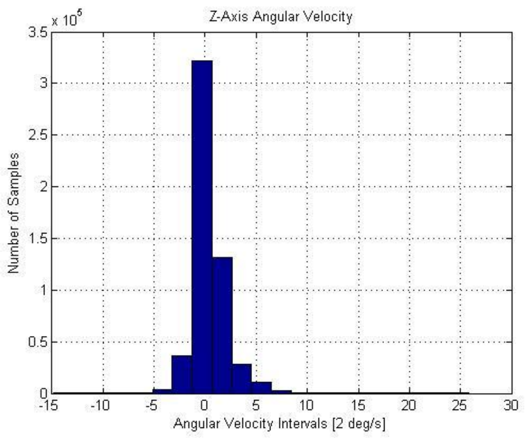
Yaw velocity [deg/s]		
Mean	0.42261	
Std Dev	1.6449	
Std Err Mean	0.002245	
Upper 95% Mean	0.41821	
Lower 95% Mean	0.42701	
N	536750	

Figure 6-22 – Yaw Statistical data

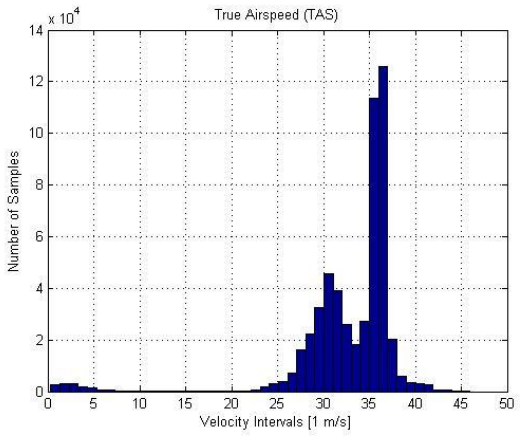
TAS [m/s]		
Mean	32.844	
Std Dev	6.008	
Std Err Mean	0.0082	
Upper 95% Mean	32.82849	
Lower 95% Mean	32.86063	
N	536750	

Figure 6-23 – True Airspeed Statistical Data

Moreover it is also possible do define the correlation coefficient between the input variables versus the target variable, in this case the target are IAS and AOA.

In the table the following notation were used:

- R is the correlation coefficient
- P is the value for testing the hypothesis of no correlation. This value is the probability of getting a correlation as large as the observed value by random chance, when the true correlation is zero. If P(i,j) is small, say less than 0.05, then the correlation R is significant.
- RLO and RUP are the lower and upper bounds for a 95%confidence interval for each coefficient.

AOA				
Input	**R**	**P**	**RLO**	**RUP**
Aileron	-0.1173	0	-0.1199	-0.1147
AOS	-0.2652	0	-0.2677	-0.2627
Flap	-0.1374	0	-0.1400	-0.1348
Ias	0.1655	0	0.1629	0.1681
Throttle	0.2230	0	0.2205	0.2256
Nx	-0.521	0	-0.5235	-0.5196
Ny	0.1486	0	0.1460	0.1513
Nz	0.1044	0	0.1017	0.1070
p	-0.1193	0	-0.1220	-0.1167
Phi	-0.0233	0	-0.0260	-0.0207
PitchTrim	-0.3566	0	-0.3589	-0.3543
Psi	0.0522	0	0.0495	0.05487
q	0.0721	0	0.0694	0.0747
r	-0.0561	0	-0.0588	-0.0535
Stabilator	-0.4749	0	-0.4770	-0.4728
Tas	0.1776	0	0.1750	0.1802
Theta	0.4870	0	0.4850	0.4891
Rudder	0.14	0	0.1373	0.1426
Vx	0.0036	0.0077	0.0001	0.0063
Vy	0.0353	0	0.0326	0.0380
Vz	0	0.4959	-0.0036	0.0017

Table 6-3. Input Correlation Coefficient for AOA target attribute

IAS				
Input	**R**	**P**	**RLO**	**RUP**
Aileron	-0.1361	0	-0.1388	-0.1335
AOA	0.1655	0	0.1630	0.1681
AOS	-0.2362	0	-0.2388	-0.2337
Flap	-0.5113	0	-0.5133	-0.5093
Throttle	0.1368	0	0.1342	0.1394
Nx	0.3257	0	0.3233	0.3280
Ny	-0.1253	0	-0.1279	-0.122
Nz	0.1102	0	0.1075	0.1128
p	-0.01165	0	-0.0143	-0.008
Phi	0.0337	0	0.0310	0.0364
PitchTrim	0.08516	0	0.0825	0.0878
Psi	0.1295	0	0.1269	0.1322
q	0.0850	0	0.0824	0.0877
r	0.0609	0	0.0582	0.0635
Stabilator	-0.2214	0	-0.223	-0.218
Tas	0.9979	0	0.9978	0.9978
Theta	-0.2590	0	-0.2615	-0.256
Rudder	0.2268	0	0.2242	0.2293
Vx	-0.0533	0	-0.0559	-0.050
Vy	0.0237	0	0.0210	0.0263
Vz	0.0470	0	0.0443	0.0496

Table 6-4. Input Correlation Coefficient for IAS target attribute

6.2.2.3 Data Preparation.

This phase covers all activities needed to collect the final dataset to be used in the subsequent Modeling Phase. Data preparation tasks include data cleaning, data integration and transformation, data and attribute selection and reduction. First of all, a clustering algorithm (k-means) is applied in order to explore the available dataset and to select five flights for the testing phase. These flights are statistically representative of the whole sample of flights and also dissimilar from one another. The algorithm , as described in §4.3.2.1, is based on the concept of distance (for example, Euclidean distance), in the sense that the closer are the object in the dataset, the more similar to each other they are.

The remaining flights are used for training and validation of all the ANNs. Moreover, in the Data Preparation phase, an algorithm is used to select the input attributes for the creation of neural models. In order to sample the whole dataset and to obtain a smaller subset to apply a clustering algorithm, WEKA tool has got a Resample filter that produces a random subsample of the dataset [B71]. The percentage of the original dataset has been fixed to 10%. Using WEKA tool, the clustering algorithm selected is SimpleKMeans. The Euclidean distance is fixed for the distance function parameter and number of clusters (i.e., of flights) is k=5. The k-means algorithm divides the space into k=5 clusters of records. Each record is a vector in 22 dimensions (21 flight attributes and the name of the each flight) which represents an instant of flight, or an instant recording of the 21 measures. Each cluster has a highest number of records belonging to a flight. This flight is chosen as representative of the cluster.

The five flights identified are chosen as test flights and are shown in Table 6-5 , while the cluster centroids are reported in Table 6-6

Number	Flight Name	Number of records
1	ff28	19019
2	fg07	18853
3	fg08	19295
4	unina11	22873
5	unina17	18723
		Tot=98736

Table 6-5. Flights for testing models

Attribute	Full Data	1	2	3	4	5
	536750	203420	221030	47050	34000	31250
Stabilator [deg]	-5.374	-5.623	-5.433	-3.937	-5.501	-5.354
Alpha [deg]	8.12	8.55	8.13	5.91	8.76	7.95
Flap [deg]	2.59	0.039	0.092	28.90	0.06	0.1
Nx [g]	-0.08	-0.082	-0.084	-0.062	-0.088	-0.073
Ny [g]	-0.001	-0.002	0.002	-0.006	-0.002	-0.007
Nz [g]	1.01	1.007	1.007	1.003	1.04	1.05
Vx [m/s]	-0.09	-18.54	15.42	7.74	11.05	-13.65
Vy [m/s]	0.46	27.83	-23.09	-11.52	23.84	-18.38
Vz [m/s]	0.12	13.15	-10.46	-5.97	-18.84	20.08
Beta [deg]	0.59	0.39	0.74	1.19	0.30	0.13
Phi [deg]	-0.3	-0.35	0.11	-0.07	-2.08	-1.31
Theta [deg]	4.39	4.54	4.65	3.03	4.37	3.66
Psi [deg]	-25.35	62.9	-116.94	-94.45	142.81	-30.97
Ias [m/s]	31.89	32.94	32.61	21.35	33.24	34.31

Table 6-6. K-Means centroids results

The algorithm used for this partitioning is the Holdout Method applied in Figure 6-24 and described in §4.3.2.1.1 and §4.3.2.1.2, where the largest dataset is randomly divided into three subsets:

- *Training set* is a subset of the dataset used to build predictive ore regressive models.
- *Validation set* is a subset – typically 25% or 33% of the full dataset – used to assess the performance of model built in the training phase. It provides a test platform for fine tuning model's parameters and selecting the best-performing model.
- *Test set* or unseen examples is a subset of the dataset to assess the likely future performance of a model.

The Cluster Analysis has produced 5 test sets relevant to 5 distinct full flights. In order to split the entire collection into Training and Validation sets, the remaining 437987 records (536750 records in the original set – 98763 records of the 5 flights obtained by the cluster analysis) are sampled getting 109496 records (25% of the remaining records).

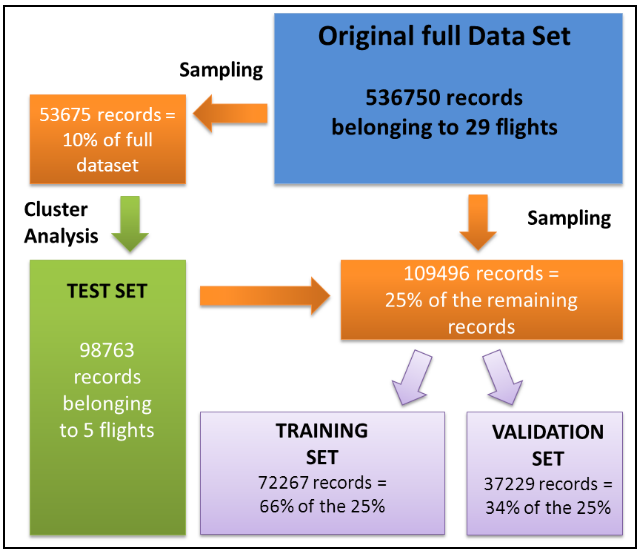

Original full Data Set
536750 records belonging to 29 flights
Sampling
53675 records = 10% of full dataset
Cluster Analysis
Sampling
TEST SET
98763 records belonging to 5 flights
109496 records = 25% of the remaining records
TRAINING SET
72267 records = 66% of the 25%
VALIDATION SET
37229 records = 34% of the 25%

Figure 6-24 : Holdout method applied to FLARE flight data set

6.2.2.4 Modeling and Evaluation

In this phases, various modelling techniques can be selected and applied, and their parameters are calibrated to optimal values. Moreover models built by the modeling phase are evaluated and the steps involved in building them reviewed to ensure the business objectives have been achieved. In order to build a regression model for IAS and AOA, the Feedforward Multilayer Network algorithm, described in §3.3.3, is selected to realize a set of ANNs, the parameters are settled and a lot of models are compared.

In the modelling phase, various regression neural models are selected and compared using some metrics, and their parameters are calibrated. Using the Neural Toolbox provided by Matlab, the algorithm's parameters listed in Table 6-7 have to be calibrated.

Name	Description	Abbreviation
Learning Rate	The amount the weights are updated	L
Momentum	Momentum applied to the weights during updating	M
Training Time	The number of epochs to train through	N
Validation Threshold	Used to terminate validation testing. The value here dictates how many times in a row the validation set error can get worse before training is terminated	E
Hidden Layers	This defines the hidden layers of the neural network. This is a list of positive whole numbers	H

Table 6-7. Feedforward parameters

For example for the IAS target during the modeling phase six models, with all the related parameters reported in Table 6-8 were developed.

	L	M	N	E	H
1	0.4	0.15	500	20	7
2	0.3	0.2	200	20	15
3	0.3	0.2	10	12	15
4	0.35	0.15	10	20	25
5	0.28	0.2	20	15	45
6	0.4	0	500	20	7

Table 6-8. IAS ANN network parameters values for six different models

After building a number of different regression models, performance obtained on the dataset belonging to the flights of Table 6-5 are reported in Table 6-9.

	Model	Validation	ff28	fg07	fg08	unina11	unina17	Average on five flights
Instances			19019	18853	19295	22873	18723	
R (Eq.65) MAE (Eq.64) RMSE (Eq.63)	1	0.95 0.92 1.74	0.65 0.88 1.06	0.95 1.06 2.08	0.94 1.05 2.18	0.97 0.40 0.76	0.99 0.63 0.80	0.90 0.80 1.38
R MAE RMSE	2	0.97 0.75 1.37	0.62 0.65 0.93	0.98 0.74 1.29	0.98 0.69 1.28	0.98 0.37 0.60	0.99 0.58 0.71	0.91 0.61 0.96
R MAE RMSE	3	0.96 0.86 1.57	0.66 0.88 1.04	0.95 0.92 1.97	0.95 0.94 2.07	0.97 0.39 0.72	0.98 0.66 0.86	0.90 0.76 1.33
R MAE RMSE	4	0.96 0.84 1.56	0.85 1.18 1.23	0.96 1.39 1.81	0.94 1.28 2.18	0.97 0.77 0.96	0.98 0.87 0.90	0.94 1.10 1.41
R MAE RMSE	5	0.97 0.80 1.47	0.55 0.77 1.03	0.97 0.97 1.65	0.96 0.98 1.76	0.98 0.40 0.62	0.99 0.61 0.77	0.89 0.74 1.17
R MAE RMSE	6	0.97 0.77 1.44	0.69 0.89 1.06	0.96 0.94 1.81	0.96 0.86 1.82	0.98 0.37 0.60	0.98 0.67 0.86	0.92 0.75 1.23

Table 6-9. IAS performance

They all satisfy requirements except for the model number 4 which has the highest MAE. As resulting from the Table 6-9 analysis, the red-highlighted model results as the best in average. In order to display the behaviour of the chosen red model, several graphs are shown in the following figures, where:

Blue line graphs the 'true IAS'

Red line graphs the 'predicted IAS'

All graphs are for flight "unina11" which shows very high performance. The 'Predicted IAS' blue curve calculated by the red model is very close to the red 'IAS' curve.

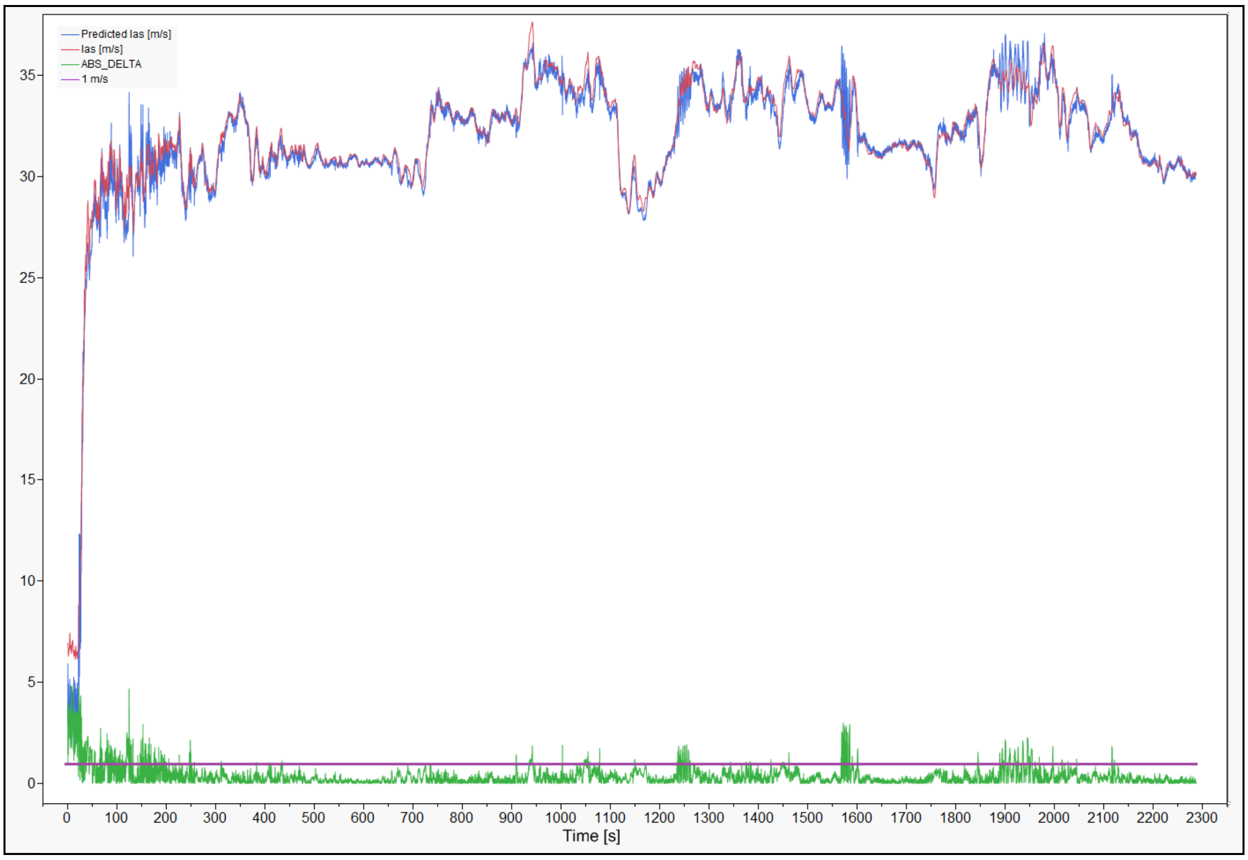


Figure 6-25: Predicted IAS vs IAS' (m/s) as function of Time

In order to enhance the discrepancy between the two curves, the absolute error is calculated point by point:

(Eq.66) $$ABS_{DELTA} = |real\,measure - virtual\,measure| = |a_i - p_i|$$

Figure 6-26 shows the Bubble Plot of 'Predicted IAS' vs 'IAS' seized by ABS_DELTA of "unina11" flight; in particular, the green line is the bisector: the closer each point is to the bisector, the smaller the distance between the two curves and the smaller the bubble. ; the violet line is the desired error upper bound level (1 m/s).

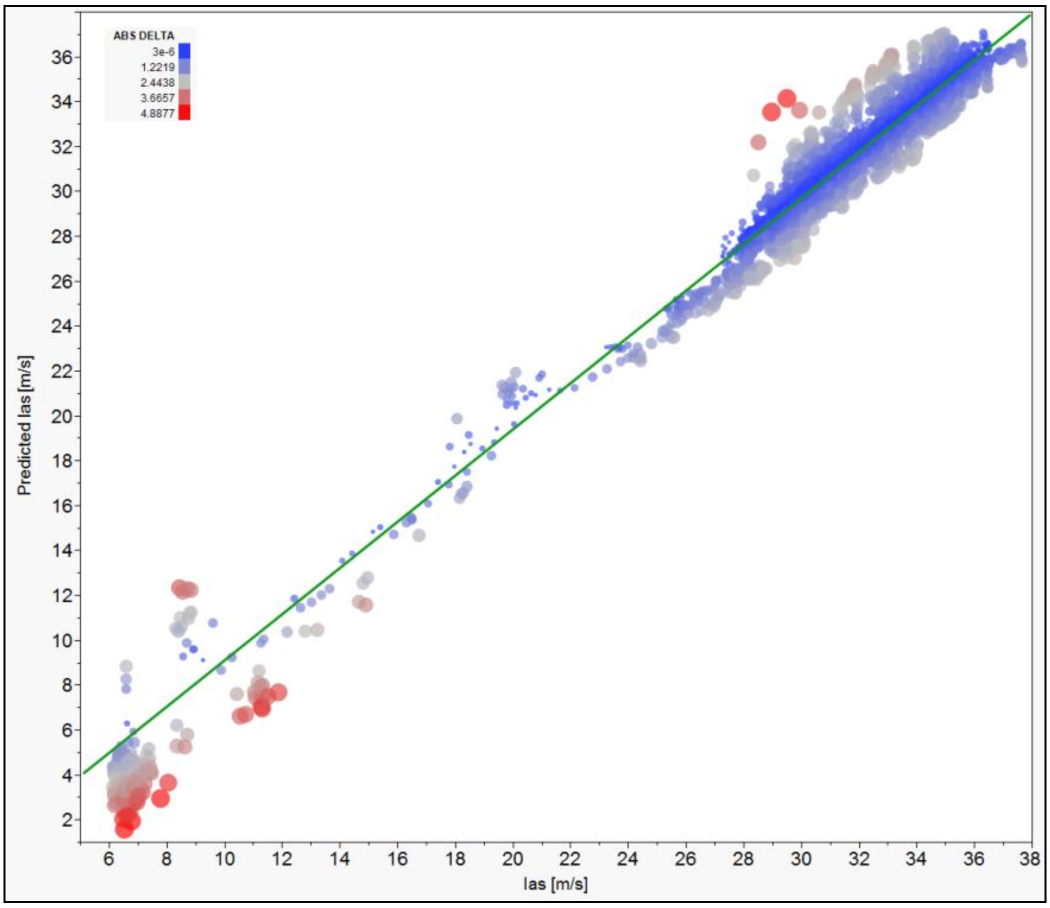


Figure 6-26: Bubble Plot of 'Predicted IAS' (m/s) by 'IAS' (m/s) seized by ABS_DELTA

In the case of the performance for the AOA, the following results are achieved, when specifically, as for the IAS measure, an ANN with the following characteristic has been designed (Table 6-10)

	ANN input	ff28	fg07	fg08	unina11	unina17	Average
Istances		19019	18853	19295	22873	18723	
R MAE RMSE	See Table 6-3. AOS and IAS Excluded	0.9511 0.3650 0.4428	0.9381 0.4319 0.9048	0.9460 0.5501 0.8160	0.9718 0.2827 0.4272	0.9813 0.2589 0.3763	0.9576 0.3777 0.5934

Table 6-10: AoA ANN performance

At this point of the process, to realize a fault tolerant architecture, it is necessary to check if we use a different number of inputs, which are the ANN performance. This argument is also valid for the IAS measure.

In Table 6-3 a priori correlation between the inputs and the outputs is shown. From the back-propagation algorithm, it is possible after the training to do a posteriori analysis (Eq.44) about the input sensitivity. Concerning FDI, if we suppose to have a GPS fault, then it is possible to design an ANN for the AoA estimation whose performances satisfy the requirement of Table 6-1. The input selected are shown in Table 6-11.

AOA
Nx
Ny
Nz
p
q
r
Phi
Theta
Psi
Ias

Table 6-11: AoA ANN reduced input

The performance results are reported in the following table.

	ANN input	ff28	fg07	fg08	unina11	unina17	Average
Istances		19019	18853	19295	22873	18723	
R MAE RMSE	See Table 6-11	0.9175 0.3661 0.5078	0.9584 0.4551 0.7433	0.9481 0.4582 0.7228	0.9647 0.3861 0.5082	0.9797 0.2778 0.4065	0.9537 0.3887 0.5777

Table 6-12: AoA reduced ANN performance

In Figure 6-27 a comparison between the AoA true value and the result of the two different ANNs described in Table 6-10 and Table 6-11 is presented.

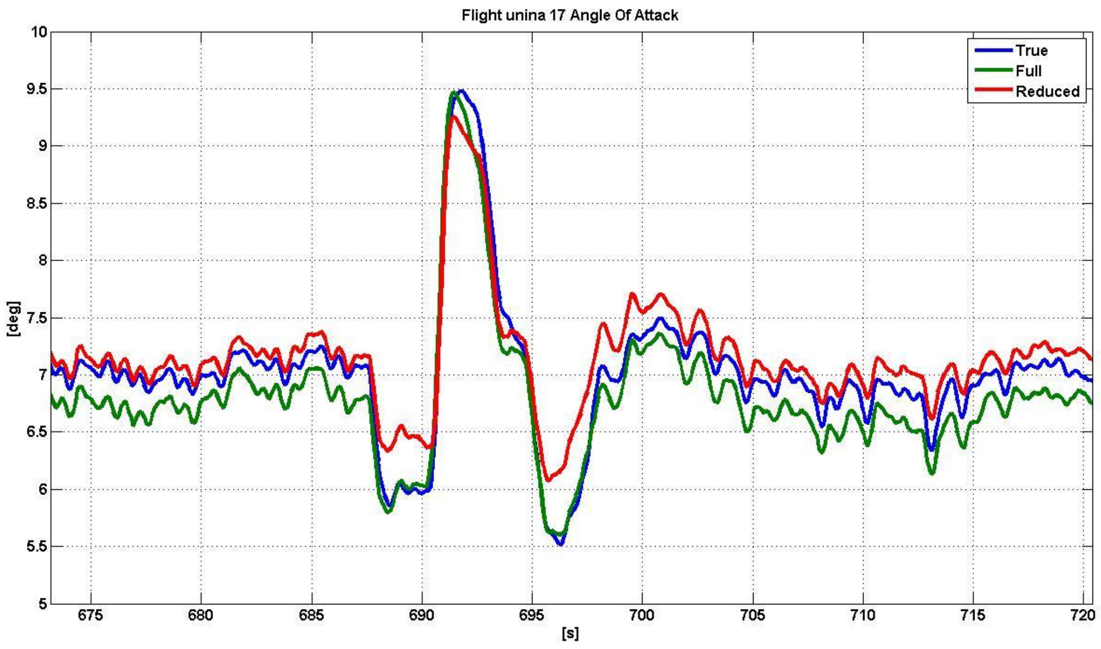


Figure 6-27:Comparison between true AoA with the virtual measures provided by two different ANNs

In Figure 6-28 a comparison between the virtual measures provided by the ANN reported in Table 6-8 and the one provided by the model based of (Eq.54) is shown.

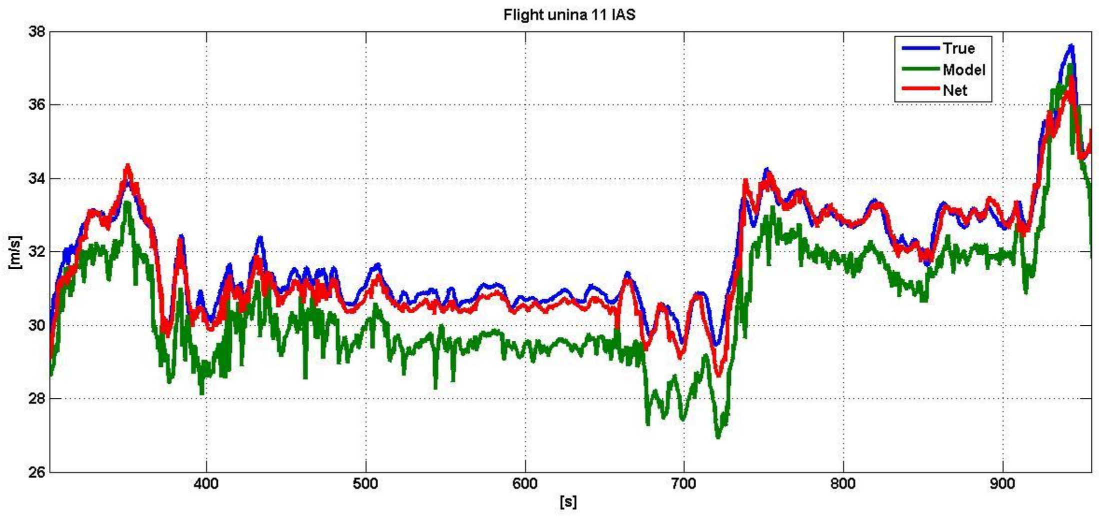


Figure 6-28: Comparison between true AoA with the virtual measures provided by an ANN and Model based

In this case the performances of the model based virtual sensor are worse than the one by the ANN.

In this phase the ANN has been implemented in Matlab/Simulink and it has been tested to verify the coding with a test vector extracted from the Test Data Set.

6.2.2.5 Deployment

The FDI system, depicted in Figure 6-29, is implemented in the FLARE FCC to detect a fault of the Indicated Airspeed measure. The system is based on the use of a family of ANNs, each one using a different set of inputs. The inputs to ANN1 consist in GPS Velocity, AHRS data, Surface Position, Angle of Attack and Sideslip. In case of GPS data absence, the ANN2 is used that receives inputs in terms of AHRS data, Surface Position, Angle of Attack and Sideslip.

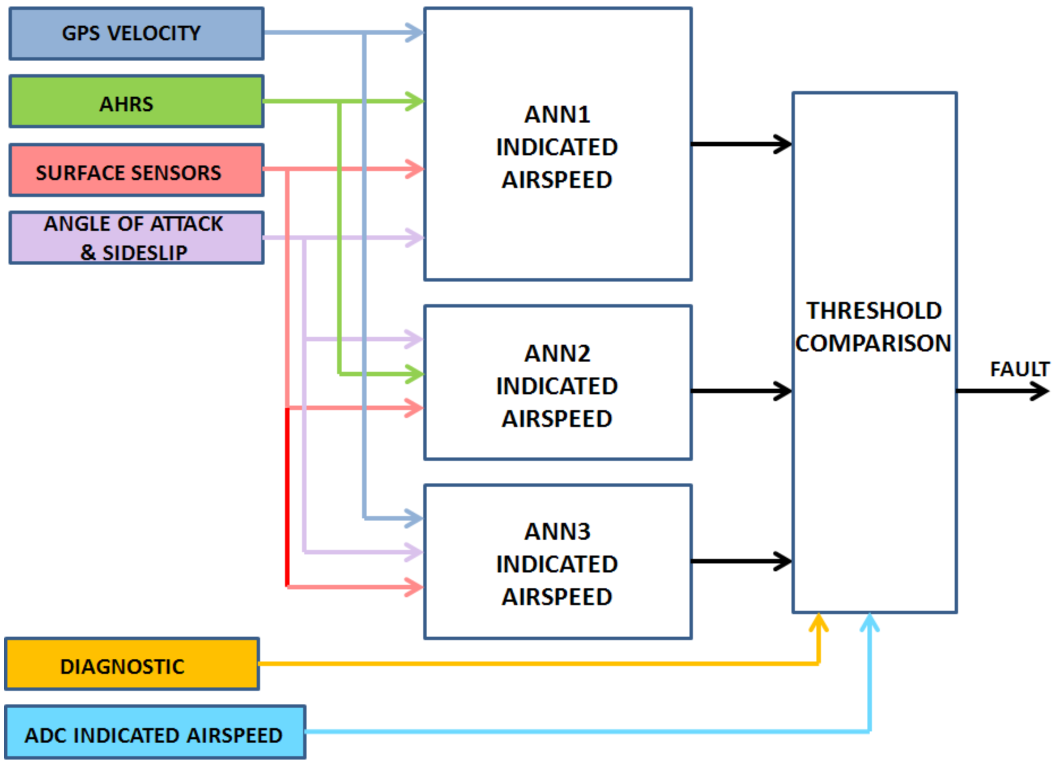


Figure 6-29; Indicated Airspeed FDI Architecture

The ANN3 is used when the AHRS data are not available, so it receives inputs in terms of GPS Velocity, Surface Position, Angle of Attack and Sideslip. The diagnostic module, finally, receives data from the built-in diagnostic systems available on some of the sensors installed on FLARE, i.e. GPS and AHRS, which allow determining sensor output availability.

Each time a failure occurs, the best performing ANN will be selected, based on the residual inputs availability.

The output of the selected ANN is compared with the actual measure in order to identify the fault

occurrence, which is detected if difference overcomes a predefined threshold. Furthermore, the threshold is dynamically scheduled on the base of the ANN performance. Obviously to avoid false alarms the fault has to be hold for a fixed continuous time, also in this case the value of this observation time is dynamically scheduled on the basis of the ANN selected. In this way also in presence of different fault of the on board sensor a virtual sensor could be used to identify fault of the interested measure using this architecture. The same architecture could be used to detect fault of the true airspeed, angle of attack and sideslip.

The architecture described in Figure 6-29 when there isn't any fault of the measurements provided in input to each ANN, can be configured as one of the classical fault-tolerant architecture shown in Figure 4-8, Figure 4-9, Figure 4-10 and Figure 4-11.

At the same way an FDI architecture for IAS can be defined using the model based sensor described in 4.2.2. In order to increase the reliability and robustness the algorithm is composed of two layers:

- limit checking layer, which checks that the physical measurements provided in input to the algorithms are not affected by a fault;
- model based layer, which implements the virtual sensors.

In the limit checking layer, the absence of fault on the measurements of aerodynamic effectors is verified by comparing the commanded deflection, set by the pilot or by the FCC, with the actual one, measured by a dedicated sensor for each aerodynamic surface. The validation of IAS measurement is carried out by ascertaining that they belong to pre-fixed ranges and that their time variations are compatible with the vehicle performance. The model based scheme for the FDI of the IAS is shown in Figure 6-30. It is composed by two virtual sensors, the first one uses in input the physical measurement of flap and stabilator deflections, whereas the second one receives in input the physical measurements of angle of attack and flap deflection.

In the diagnostic block, a voting-logic is applied to detect the fault on the IAS measurement. If the limit checking layer identifies a fault on one of the input measurements, the corresponding virtual sensor is disabled and the voter is performed on a limited number of signals. Finally, it is worthy to note that equations (Eq.53) and (Eq.54) only hold in a limited flight envelope. When the vehicle is outside this envelope, the FDI algorithm is disabled.

This FDI architecture is the classical described in 4.4.3.1 implemented using analytical redundancy.

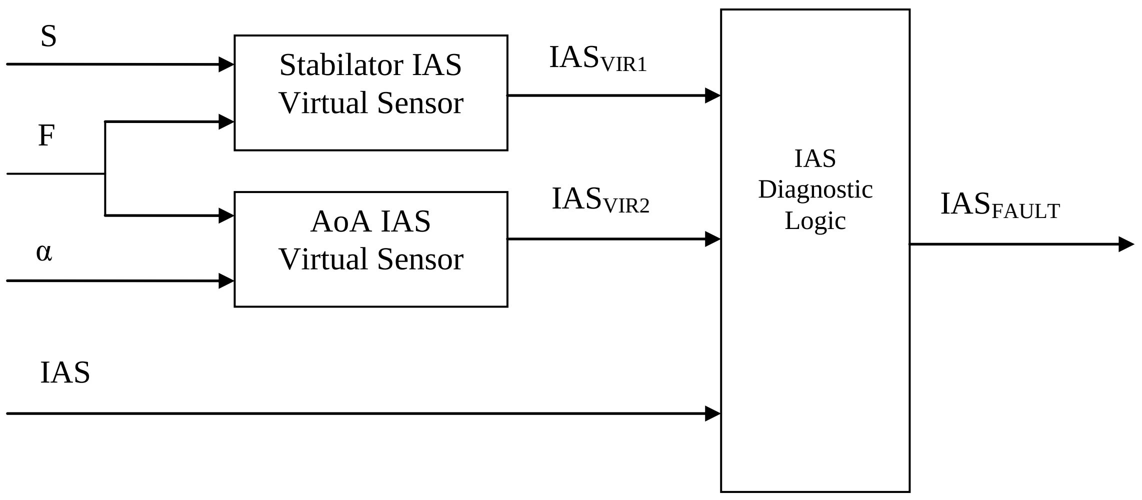


Figure 6-30: IAS diagnostic scheme

6.2.3 Numerical Result

Figure 6-31 presents a comparison between real and virtual measurements of the IAS (top plot) and detection of a fault on the physical sensor (bottom plot); the test was performed at constant altitude (middle plot).

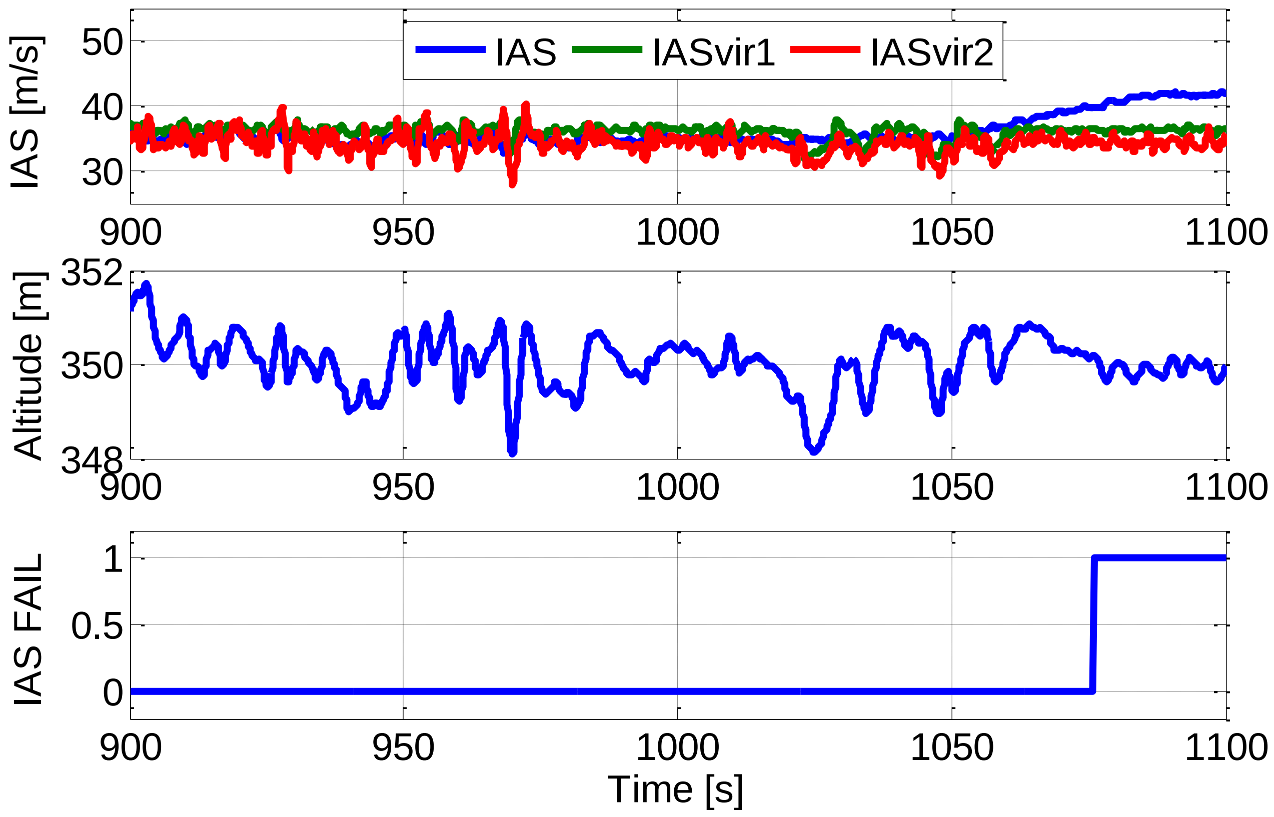


Figure 6-31: Model based FDI for IAS measurement

The validation of the FDI algorithm was performed through the flight data already used to validate the ANN virtual sensor. In order to validate also the diagnostic logic and algorithms, different types of fault were injected offline to the recorded primaries. ADC measurements. During the test shown in Figure 6-32 a fault of the differential pressure sensor was injected, which also caused a consequent measurement error on the IAS.

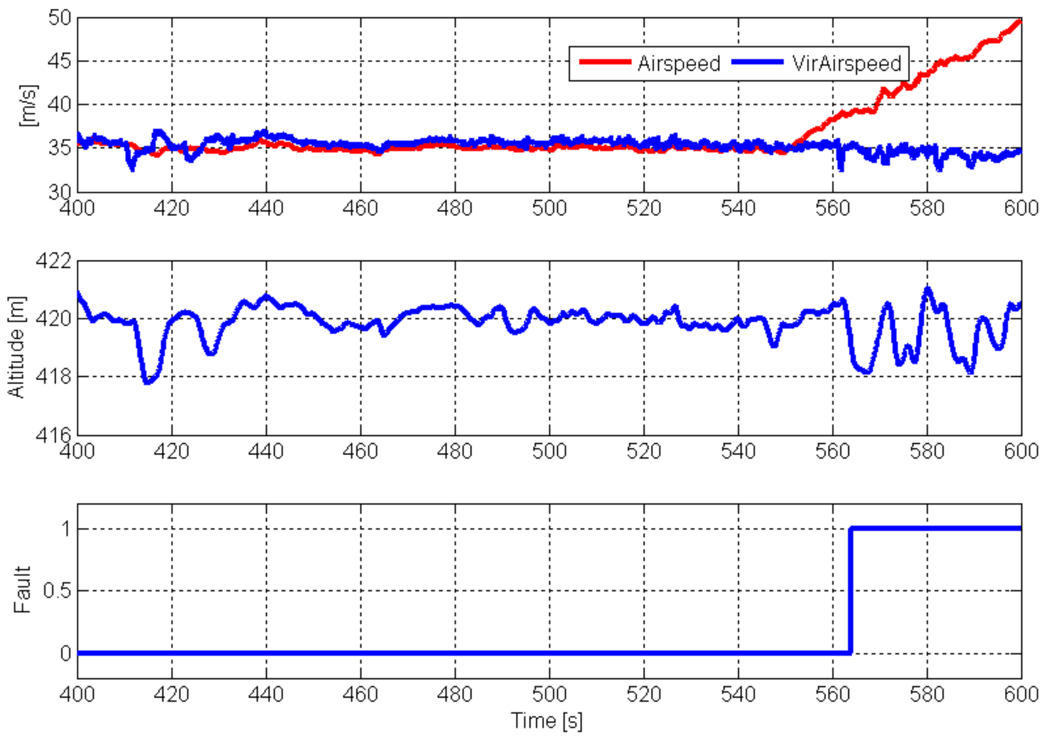


Figure 6-32: Indicated Airspeed Fault Detection

The fault produced a drift of the measurement until it reached the measurement saturation value. In the first part of the plot, when the Air Data Computer is working in nominal conditions, the ANN virtual sensor provided good estimation of the Indicated Airspeed, indeed the virtual measurement was very close to the real measurement (top plot in the figure). At 550 seconds flight time, the fault was injected, and the Indicated Airspeed started a slow drift. Since the ANN virtual sensors didn't use the differential pressure measurements, it continued to estimate correctly the Indicated Airspeed, whereas the real measurement diverged. The diagnostic algorithms detected the fault as soon as its effect was relevant, about 13 seconds after the injection (bottom plot). The test was performed at constant altitude, as shown in the middle plot.

Figure 6-33 shows the same test executed injecting a GPS fault, that provided measurements to the ANN virtual sensor, also under this circumstance, the FDI architecture was able to detect the fault.

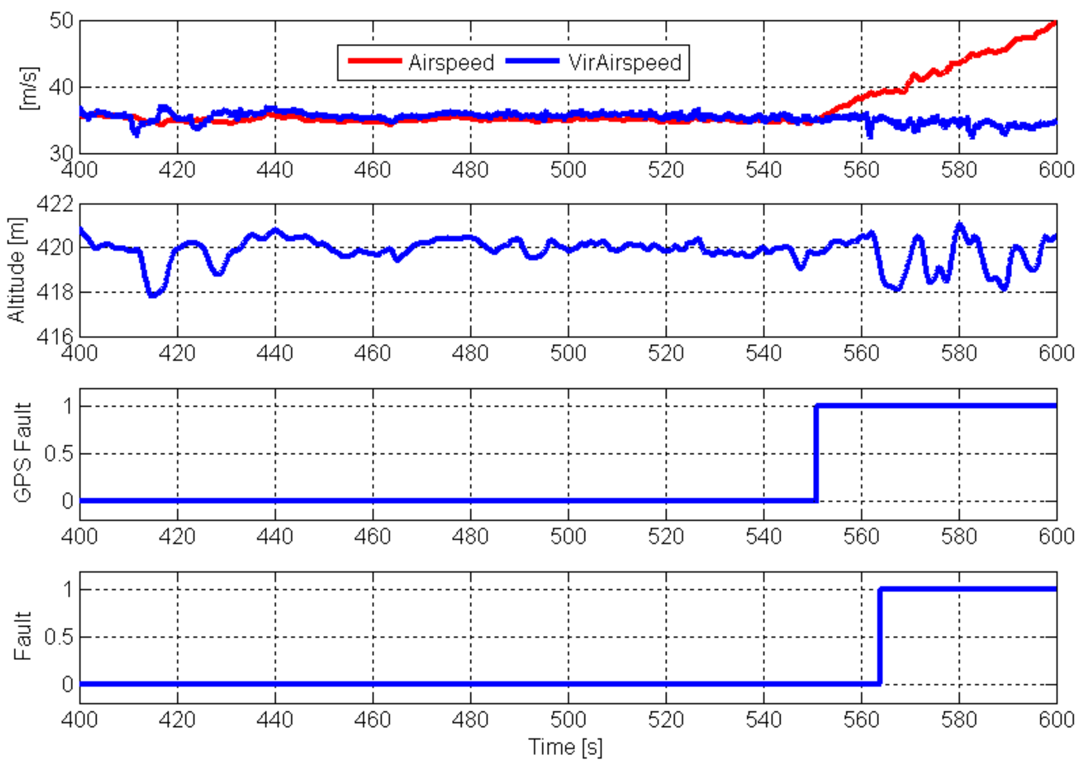


Figure 6-33: Indicated Airspeed Fault Detection with GPS Fault

The GPS fault was injected at 551 seconds flight time, the FDI system automatically switched to an ANN virtual Indicated Airspeed which didn't use as input GPS measurements. Also in this case since the ANN virtual sensors didn't use the GPS, it continued to estimate correctly the Indicated Airspeed, whereas the measurement provided by the baseline ADS diverged. The diagnostic algorithms detected the fault as soon as its effect was relevant, in this case about 15 seconds after the injection (third plot). The test, like the previous one, was performed at constant altitude, as shown in the second plot.

Similar tests have been performed for the AOA measure with the same results Figure 6-34 and Figure 6-35, present the results obtained in take-off and landing flight phases.

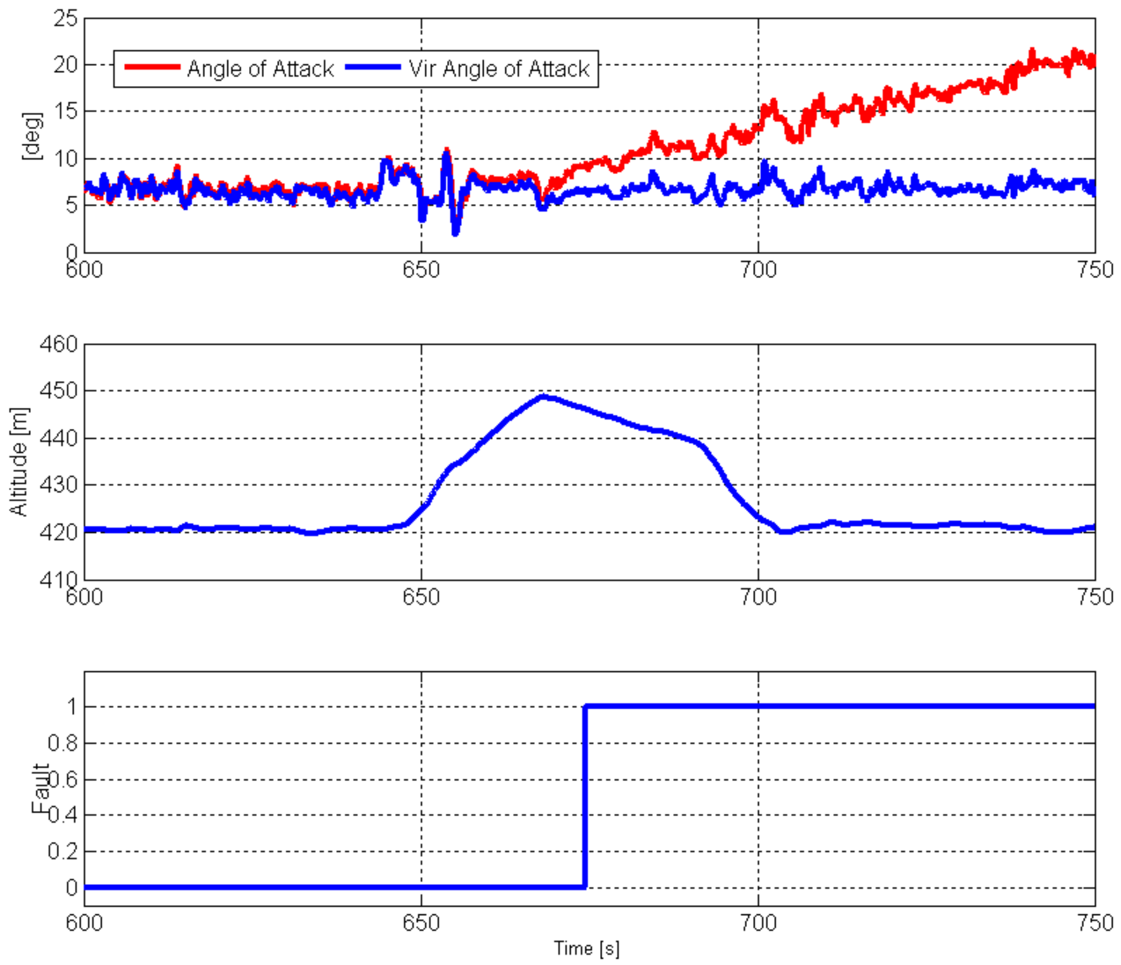


Figure 6-34: Angle of attack Fault Detection

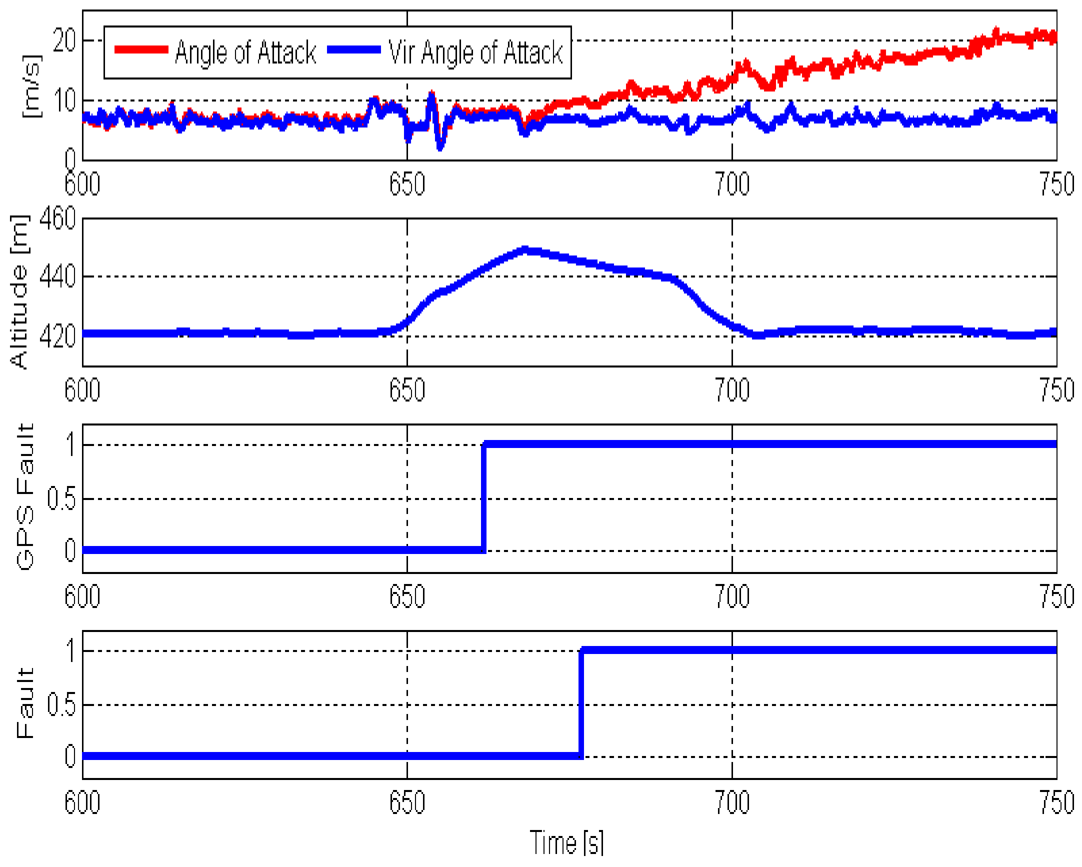


Figure 6-35: Angle of attack Fault Detection with GPS Fault

6.2.4 Laboratory Assessments

ADS laboratory tests were carried out to check:

- The raw measurements acquisition process, the conversion to engineering units and the performances of primary sensors.
- The correct implementation of the algorithms for the computation of the derived measurements.
- The validation of the diagnostic logic and algorithms.

Concerning the primary sensors and the acquisition process of the custom ADS, the following experimental tests were performed:

- Acceptance Tests, to verify the compliance of the set-up to the accuracy requirements, foreseen within the TECVOL project. Final acceptance of the ADS was also subject to the subsequent execution of a series of flight tests with the aim of verifying the correct operation of the setup in the real operating environment.

- Calibration Tests, whose objective was the definition of the sensors calibration laws.
- Validation Tests, which aimed at validating the calibration curves and the developed relationships.

6.2.4.1 Acceptance Test

The acceptance tests for each sensor are described in the following. The test setup is shown in Figure 6-36.

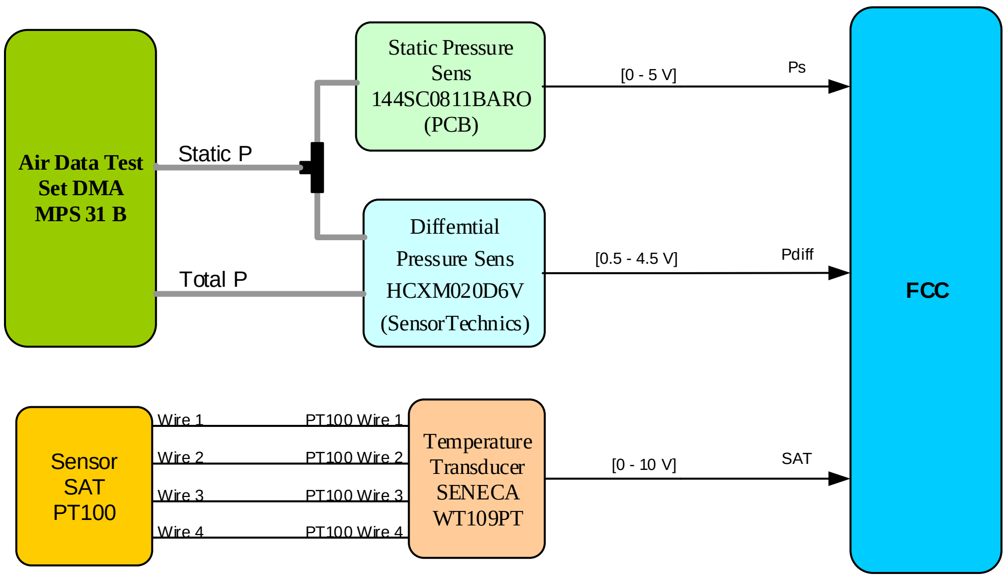

Air Data Test Set DMA MPS 31 B
Static P
Total P
Static Pressure Sens 144SC0811BARO (PCB)
Differntial Pressure Sens HCXM020D6V (SensorTechnics)
[0 - 5 V]
Ps
[0.5 - 4.5 V]
Pdiff
FCC
Sensor SAT PT100
Wire 1
Wire 2
Wire 3
Wire 4
PT100 Wire 1
PT100 Wire 2
PT100 Wire 3
PT100 Wire 4
Temperature Transducer SENECA WT109PT
[0 - 10 V]
SAT

Figure 6-36: Test Setup Architecture

6.2.4.1.1 Static Pressure Sensor

The test's objective is to verify that the transducer's accuracy is compliance with the one reported in the related datasheet [B62].

The FCC I/O was configured to allow the correct acquisition and conditioning of the input signal from the sensor.

The following instrumentation, information and datasheets were used:

- Air Data Test Set DMA MPS 31 B s/n 5990.
- 12V Power supply.
- Sensor calibration and electrical charts.

- datasheet, 144S…-PCB Series Signal conditioned precision pressure transducers , SENSORTECHNICS November 2002

The following test procedure was executed.

a. Verify each device of the Test Setup is correctly connected.
b. Vary the static pressure by means of the air data test set covering the full pressure scale. Each pressure variation has to be hold until pressure regime is achieved.
c. Take note of the voltage and the pressure (the pressure can be calculated from the voltage by using the calibration chart of the datasheet) for each pressure point.
d. Evaluate the transducer error considering both environmental conditions and power supply voltage by means of the related datasheet charts.
e. Evaluate the average pressures and voltage for each pressure point by means of the FCC data recorded.
f. Evaluate the span errors (FSS) between each average value and each nominal value generated with the air data test set. These errors have to be compared with the ones at point d) of this procedure.

The test has been performed at 25°C powering up the transducer with 12VDC. The global error has been evaluated with the following formula:

$$error = \sqrt{0.1^2 + (0.05 \cdot 4)^2 + (0.03 \cdot 4)^2} = 0.25\% FSS$$

In Table 6-13 the averaged measures are reported.

Test Pressure (Pa)	Average Pressure (V)	Average Pressure (Pa)	Error in %FSS
80000	0.003	80019	0.07
90000	1.673	90040	0.13
100000	3.339	100034	0.11
101325	3.558	101349	0.08
102271	3.714	102284	0.04
108866	4.809	108853	0.04

Table 6-13 - Static Pressure Sensor Acceptance Test Summary

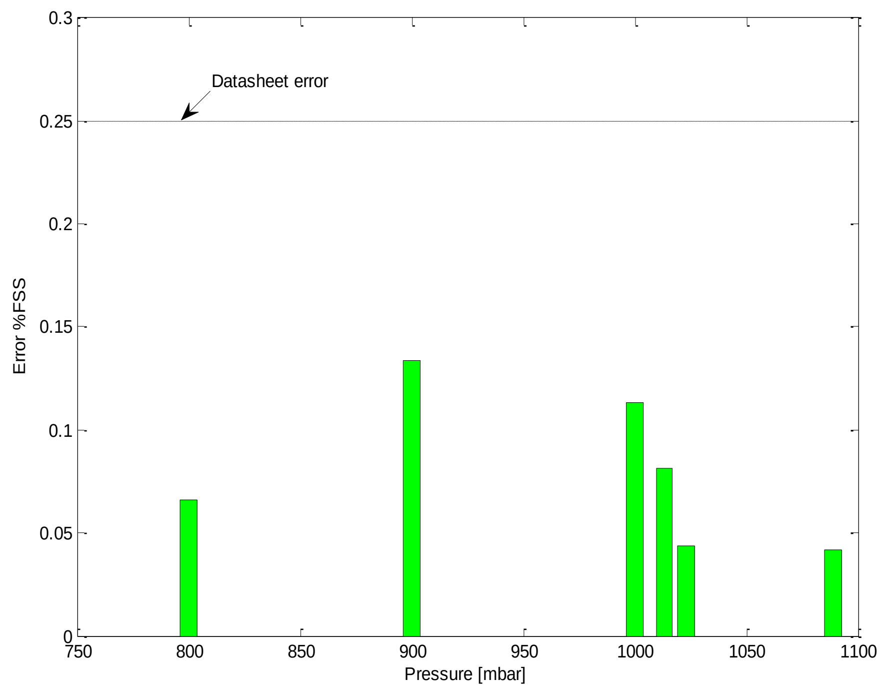


Figure 6-37 – Comparison between evaluated errors and datasheet errors

Concluding the sensors' behavior is compliant with the datasheet. The maximum pressure of the allowed datasheet (see ref. TBD) range wasn't tested for a limited air data set test output.

6.2.4.1.2 Differential Pressure Sensor

The test's objective is to verify that the transducer's accuracy is compliance with the one reported in the related datasheet [B63].

The FCC I/O was configured to allow the correct acquisition and conditioning of the input signal from the sensor.

The following instrumentation, information and datasheets were used:

- Air Data Test Set DMA MPS 31 B s/n 5990.
- 12V Power supply.
- Sensor calibration and electrical charts.
- datasheet, HCX Series, Fully Signal conditioned pressure transducers , SENSORTECHNICS March 2004

The following test procedure was executed.

a. Verify each device of the Test Setup is correctly connected.
b. Vary the dynamic pressure by means of the air data test set covering the full pressure scale. Each pressure variation has to be hold until pressure regime is achieved.
c. By means of a notebook connected to the FCC, take note of the voltage and the pressure (the pressure can be calculated from the voltage by using the calibration chart of the datasheet) for each pressure point.
d. Evaluate the transducer error considering both environmental conditions and power supply voltage by means of the related datasheet charts.
e. Evaluate the average pressures and voltage for each pressure point by means of the FCC data recorded.
f. Evaluate the span errors (FSS) between each average value and each nominal value generated with the air data test set. These errors have to be compared with the ones at point d) of this procedure.

The test has been performed at 25°C powering up the transducer with 12VDC. The global error has been evaluated with the following formula:

$$error = \sqrt{0.5^2 + (0.05 \cdot 7)^2 + (0.03 \cdot 7)^2} = 0.65\%FSO$$

In Table 6-14 the averaged measures are reported.

Test Pressure (Pa)	Average Pressure (V)	Average Pressure (Pa)	Error in %FSS
383	1.271	385	0.12
473	1.452	475	0.14
573	1.650	574	0.10
682	1.868	684	0.11
801	2.109	804	0.18
930	2.363	931	0.08
1000	2.505	1002	0.12
1068	2.641	1070	0.12
1500	3.507	1503	0.18

Table 6-14 - Static Pressure Sensor Acceptance Test Summary

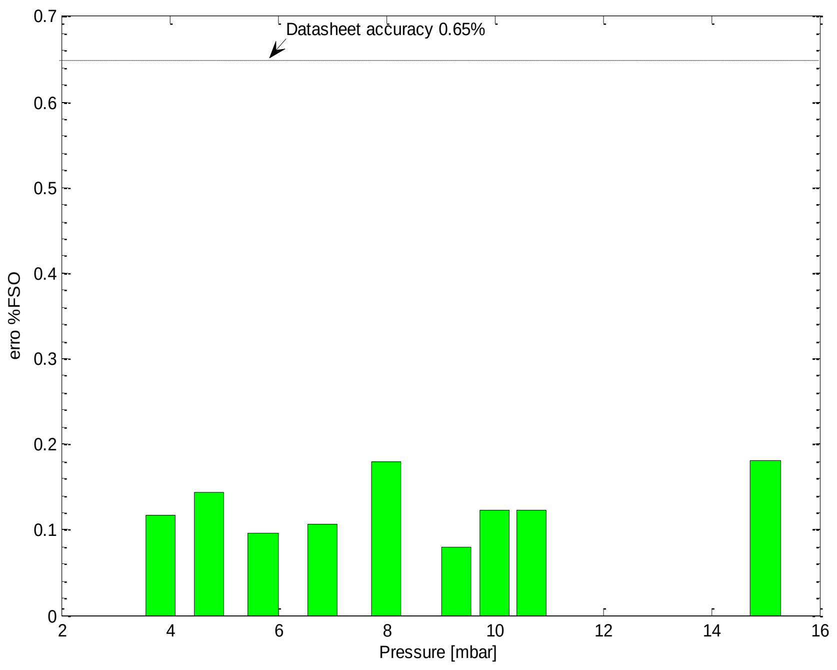


Figure 6-38 – Comparison between evaluated errors and datasheet errors

Concluding the sensors' behavior is compliant with the datasheet. The maximum pressure of the allowed datasheet (see ref. TBD) range wasn't tested for a limited air data set test output.

6.2.4.1.3 Temperature Sensor

The test's objective is to verify the accuracy of the temperature sensor.

The FCC I/O was configured to allow the correct acquisition and conditioning of the input signal from the sensor.

The following instrumentation, information and datasheets [B64] and [B65] were used:

- Temperature sensor: PT100 class B
- Temperature transmitter: SENECA WT109PT
- Infrared thermometer Fulke
- 12V Power supply.
- datasheet *PT100 WK109PT*

The following test procedure was executed.

a. Verify each device of the Test Setup is correctly connected.

b. As described into the manual, set the following parameters
    - PT100 Type: 4 wires
    - Output signal: 0-10V
    - Scale bottom: 0°C
    - Full scale: 100°C
c. Measure PT100 probe temperature in different times of the day by means of the thermometer
d. Store temperature and voltage for each temperature point by means of the FCC data recording notebook.
e. Evaluate the nominal error of the sensors considering the actual environmental conditions and the sensor datasheet
f. Calculate the average temperature and voltage for each temperature point
g. Evaluate the displacement between measurements and nominal values

The test has been performed at 22°C powering up the converter with 24VDC.

In Table 6-14 the averaged measures are reported.

Test Temperature (°C)	Average Temperature (V)	Average Temperature (°C)	Error in °C
21.9	2.23	22.3	0.4
23.2	2.35	23.5	0.3

Table 6-15 - Temperature Sensor Acceptance Test Summary

The sensor was not tested in temperatures different from the environmental one because a temperature chamber was not available.

6.2.4.2 Calibration procedures

The calibration parameters obtained through the calibration tests were included in the ADS software and allowed to further reduce the measurements error of the sensors with respect to the calibration provided by the datasheet, as shown in Figure 6-37 for the static pressure. Similar results hold for the other sensors.

6.2.4.2.1 Static Pressure Sensor

The calibration of the static pressure acquisition chain is described in the following. Specifically the objective of this calibration procedure is to find, in static condition, the transfer function between the ADC board voltage and the static pressure applied to the sensor. The outputs of the calibration procedure are the parameters to be used for conditioning the final ADC acquisition chain.

The following instrumentation, information and datasheets were used:

- Air Data Test Set DMA MPS 31 B s/n 5990.
- 12V Power supply.

The following calibration procedure has to be executed.

a. Verify each device of the Setup is correctly connected.
b. Vary the static pressure by means of the air data test set covering the full pressure scale. Each pressure variation has to be hold until pressure regime is achieved.
c. Take note of the voltage and the pressure (the pressure can be calculated from the voltage by using the calibration chart of the datasheet) for each pressure point.
d. Built the calibration chart between the output voltage of the sensor and the applied pressure.

The achieved calibration curve is shown in Figure 6-40. The inverted calibration curve to be used in the conditioning SW is shown in Figure 6-41. The calibration curve can be compared with the nominal one in Figure 6-39.

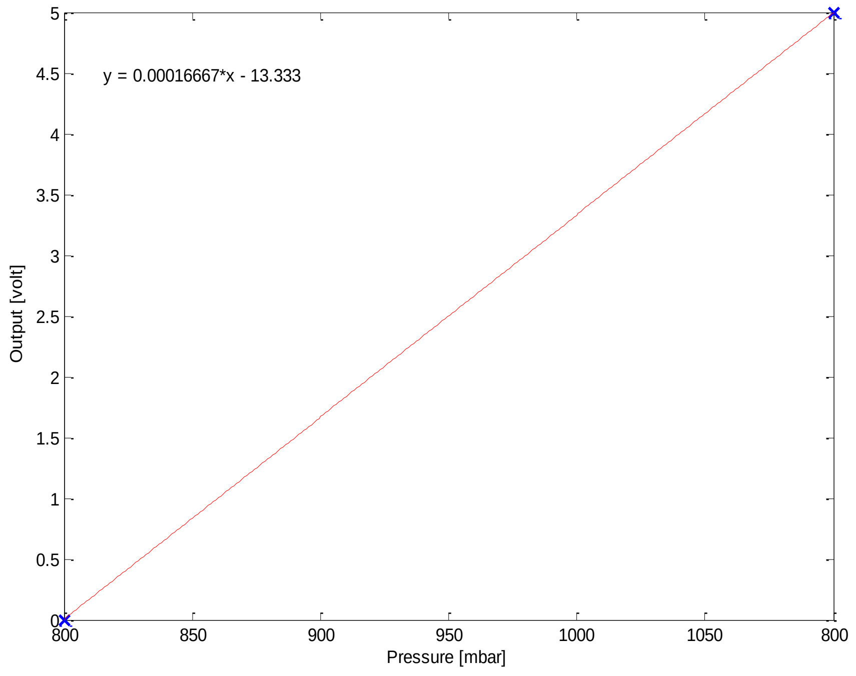


Figure 6-39– nominal conversion curve (datasheet)

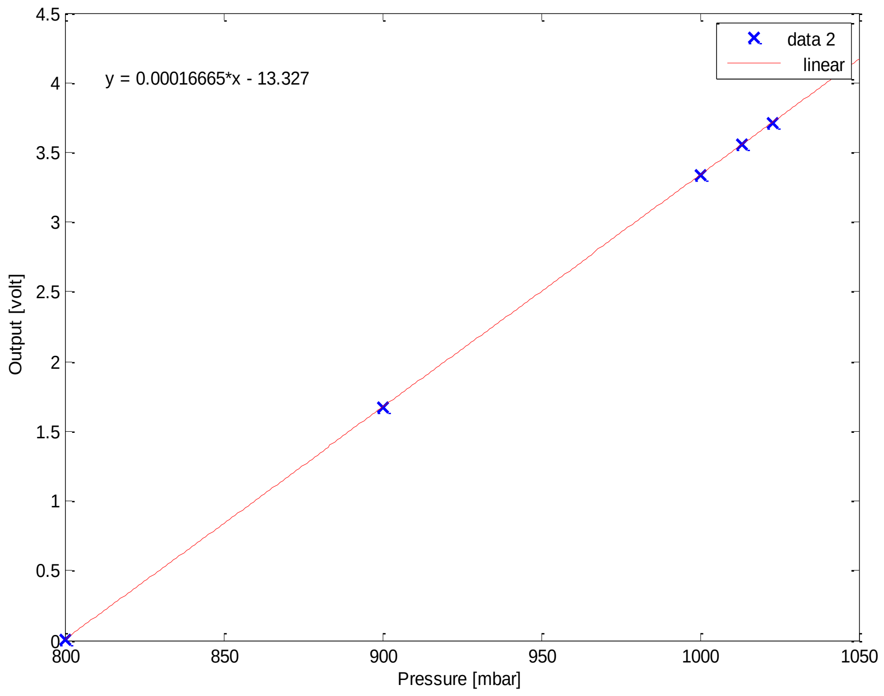


Figure 6-40– Calibration conversion curve

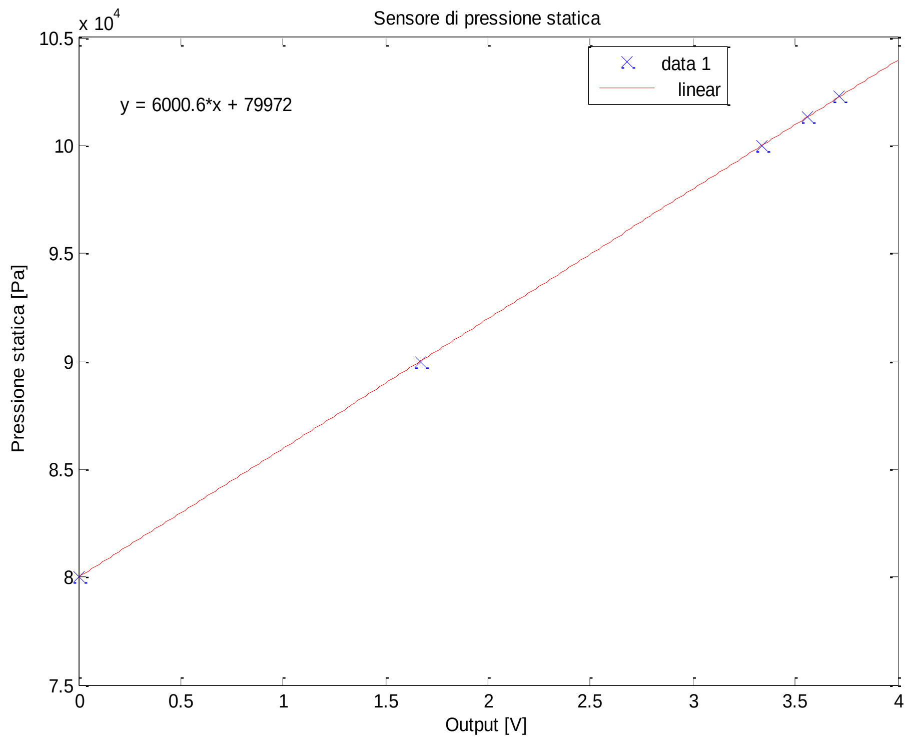


Figure 6-41– Inverted calibration conversion curve

6.2.4.2.2 Differntial Pressure Sensor

The calibration of the differential pressure acquisition chain is described in the following. Specifically the objective of this calibration procedure is to find, in static condition, the transfer function between the ADC board voltage and the dynamic pressure applied to the sensor. The outputs of the calibration procedure are the parameters to be used for conditioning the final ADC acquisition chain.

The following instrumentation, information and datasheets were used:

- Air Data Test Set DMA MPS 31 B s/n 5990.
- 12V Power supply.

The following calibration procedure has to be executed.

a. Verify each device of the Setup is correctly connected.
b. Vary the dynamic pressure by means of the air data test set covering the full pressure scale. Each pressure variation has to be hold until pressure regime is achieved.

c. Take note of the voltage and the pressure for each pressure point. The final voltage will be the average of the samples recorded at each pressure point.

d. Built the calibration chart between the output voltage of the sensor and the applied pressure.

The achieved calibration curve is shown in Figure 6-43. The inverted calibration curve to be used in the conditioning SW is shown in Figure 6-44. The calibration curve can be compared with the nominal one in Figure 6-42.

-

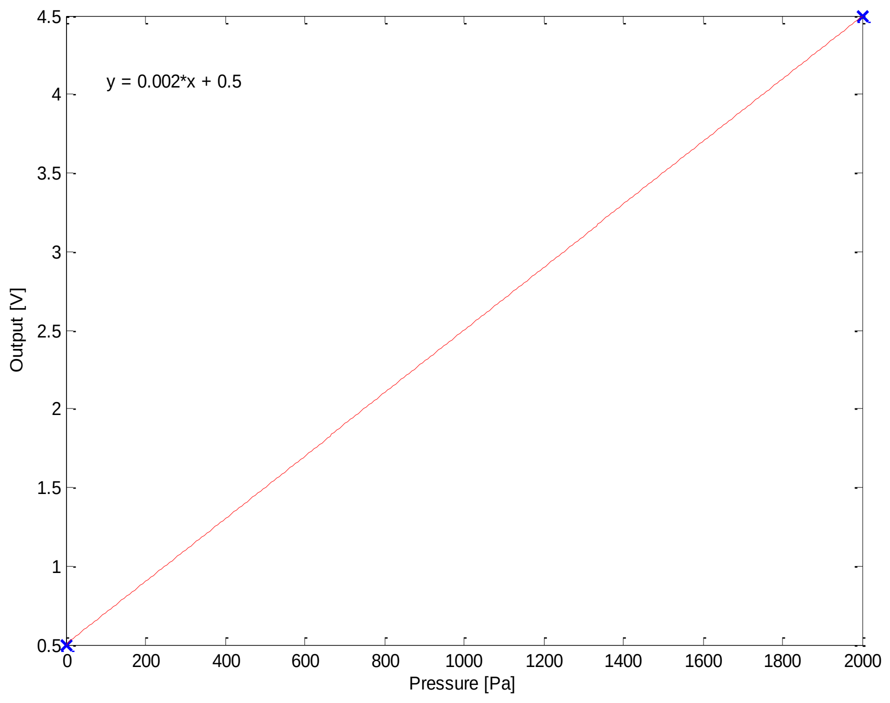


Figure 6-42– nominal conversion curve (datasheet)

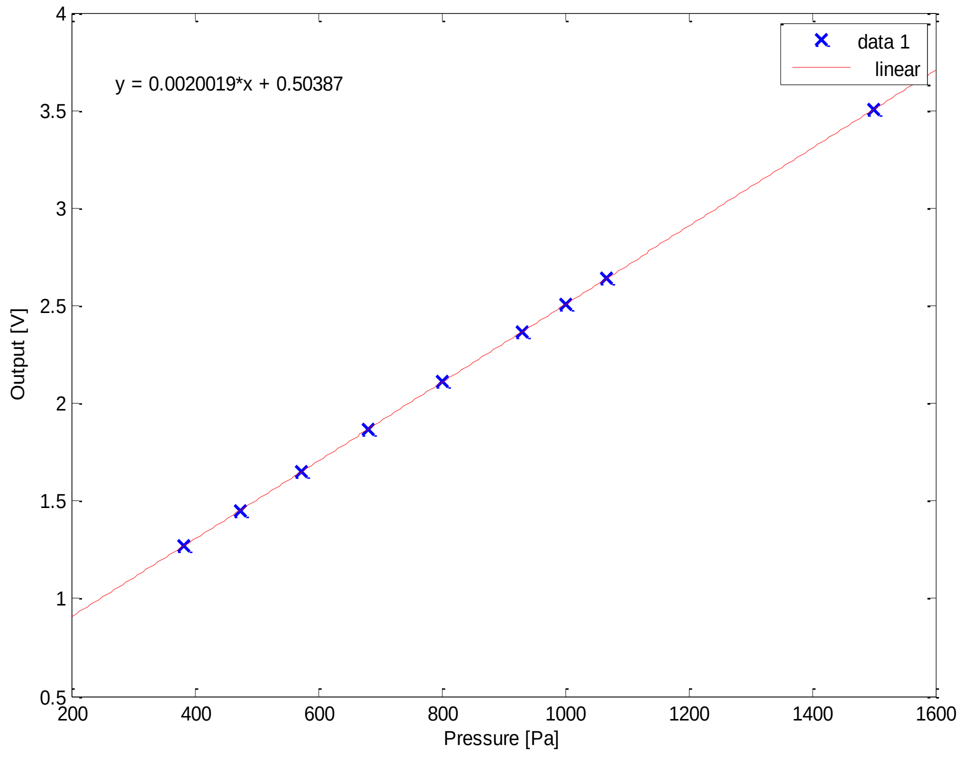


Figure 6-43– Calibration conversion curve

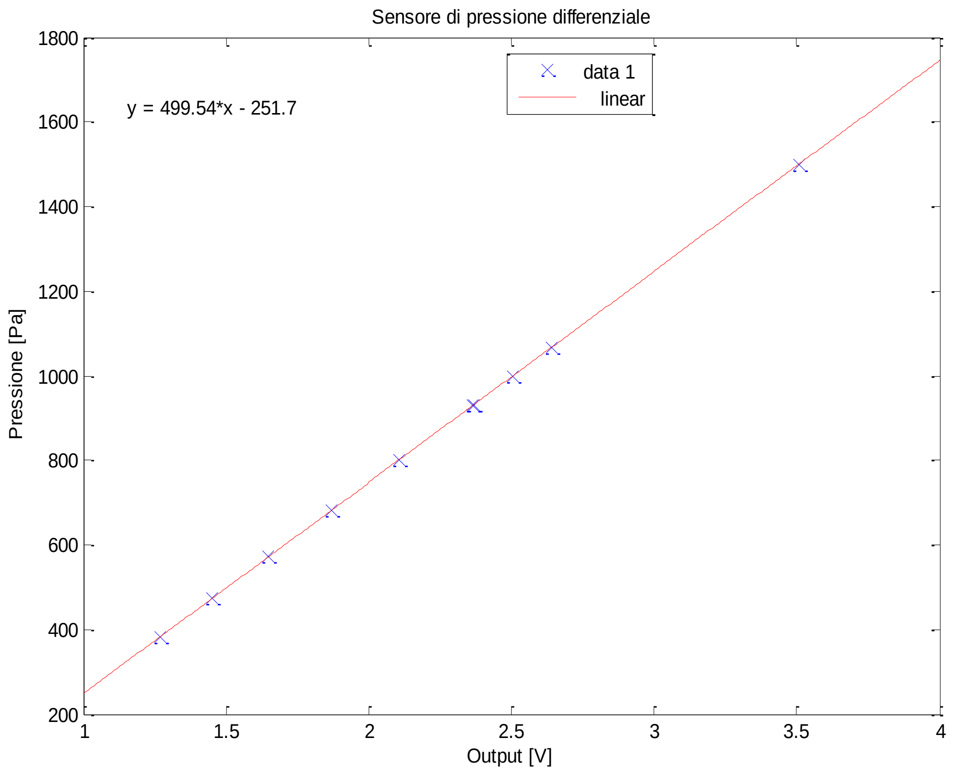


Figure 6-44– Inverted calibration conversion curve

6.3 Flight Demonstrations

The model based virtual sensors presented in this thesis have been finally validated in flight, using the FLARE vehicle. The next subsections present the procedure used to calibrate a commercial ADS, used as reference measurement system, the performance of such commercial ADS and the comparison with the developed virtual sensors. During the flight test failures were not injected, for safety reason. Consequently, the FDI algorithms and architecture were not subject to this validation test.

6.3.1 In Flight Calibration

The probes, used to measure pressure, angle of attack, angle of sideslip and temperature, are influenced by the aerodynamic field near the aircraft, which introduces an error in the measurements depending of their position. In-flight calibration allowed to calculate and correct this error.

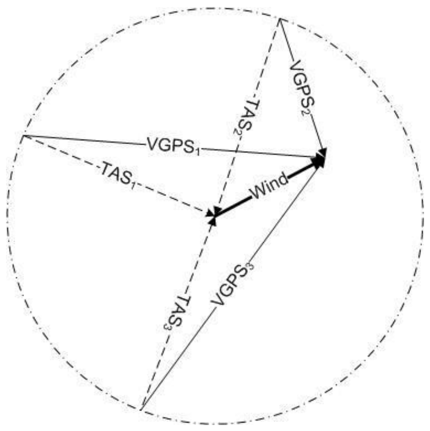
The position error correction (PEC) on the TAS was computed following the technique described in **[**B72]. This methodology uses the GPS to evaluate the true airspeed assuming constant wind. The aircraft has to fly for at least three straight legs (in different directions) maintaining constant airspeed and altitude and measuring ground speed and track with a GPS. Then three equations in three unknowns can be solved giving wind speed, wind direction and TAS. Figure 6-45 helps to understand the method, showing ground speed as the vector sum of wind and true airspeed.

The angle of attack ([B66]) position error has been corrected measuring the true value as difference of pitch angle and slope angle in steady state rectilinear flight.

The calibration was performed in 6 test points:

- TAS 135 km/h Flap 0°
- TAS 120 km/h Flap 0°
- TAS 110 km/h Flap 0°
- TAS 110 km/h Flap 36°
- TAS 100 km/h Flap 36°
- TAS 90 km/h Flap 36°

In particular for each test point for legs were flown. In this way four different combinations of three legs were available. The calibration was executed on each of these combinations and the results were averaged. Figure 6-46 shows the rectangular circuits performed during one test point.



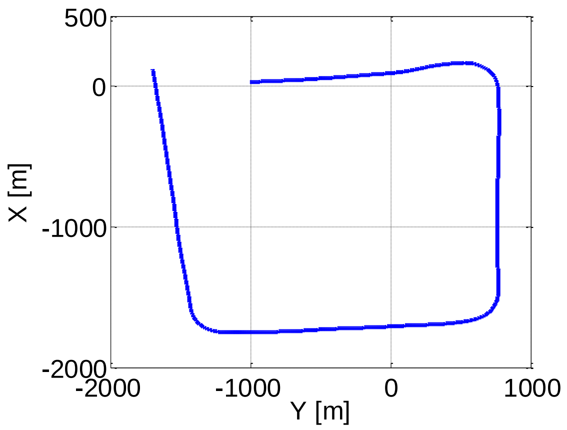


Figure 6-45: Ground speed vector computation **Figure 6-46: Aircraft trajectory during a test point**

The main data acquired during one test point are presented in Figure 6-47 and Figure 6-48. Figure 6-47 shows the ground track during the test, where a variation of about 90 degree for each legs was performed. In Figure 6-48 the wind effect on the ground speed is evident, in the first and third legs. In fact, as estimated during the tests, there was a wind of about 4.5 m/s with 360 deg of direction.

The results of the calibration and the errors due to the positioning of the pitot are reported in [B72], where Vtrue is the true airspeed computed following the described procedure, whereas TAS is the measurement provided by the ADS before position error correction. As can be seen the error is very low, maximum 0.4 m/s.

	Vtrue	Wind	Wind Direction	TAS	ΔTAS	Flap
	(m/s)	m/s	Deg	(m/s)	(m/s)	(deg)
1	24,362	5,3	356	24	0,4	36,0
2	27,885	5,4	359	27,8	0,1	36,0
3	29,895	4,8	357	29,8	0,1	36,0
4	30,446	4,5	1	30,25	0,2	0,0
5	32,339	4,5	1	32	0,3	0,0
6	36,302	4,0	358	36	0,3	0,0

Table 6-16 – TAS correction.

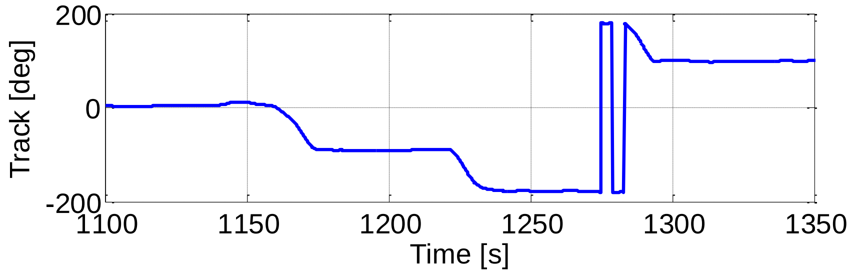


Figure 6-47: Ground track during a test point

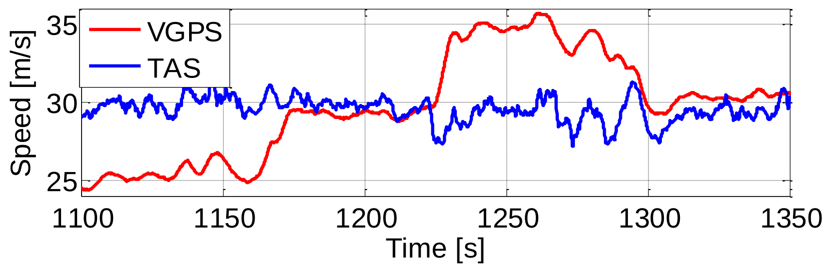


Figure 6-48: Comparison between true air speed and GPS velocity during a test point

Finally, the data required to calibrate the angle of attack, following the above described procedure, are illustrated in Figure 6-49.

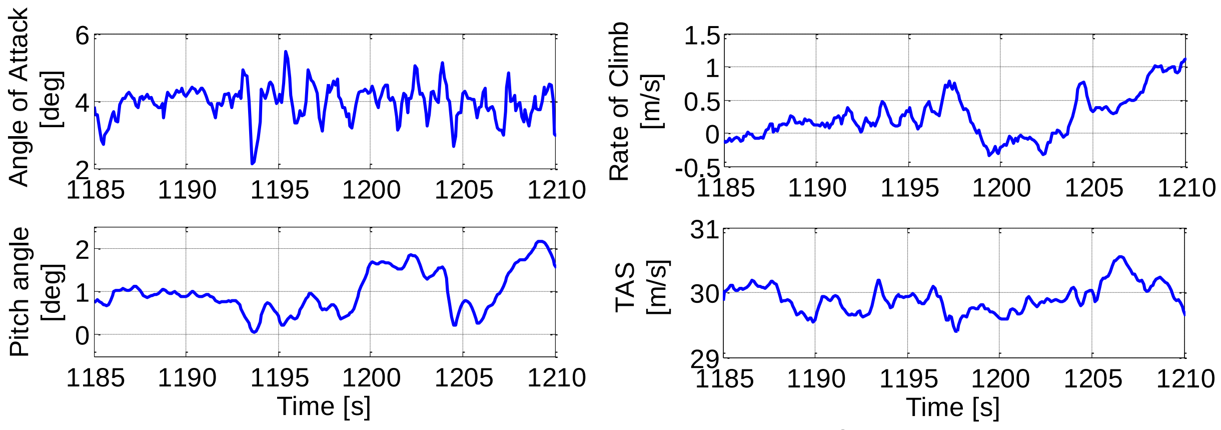


Figure 6-49: Time histories used to calibrate the angle of attack sensors.

6.3.2 Air Data System Experimental Result

In order to demonstrate the effectiveness of the proposed system, the performance of the custom ADS were tested in-flight during the TECVOL experimentation [AR6].

Figure 6-50 presents the comparison during a flight test between the altitude above ground level derived from the GPS measurement and the pressure altitude computed by the ADS using the measurement of the static pressure and the Baroset evaluated before starting the test. The matching between the measurements is very good with small errors in the final part of the test, which are due to changed atmospheric conditions. However the ADS was able to catch the dynamic behaviour of the measurement along the whole flight.

Figure 6-51 shows the TAS (provided by the ADS and its related control reference trajectory) and the inertial velocity of the vehicle during the Flare and Pre-Touch Down phases. During the Flare phase the reference TAS changes linearly with the longitudinal position along the Runway axis (XRW) between the nominal TAS set-point for the Ramp phase and the desired TAS value at the touch down. The TAS at the touch down point (indicated as ground contact in the figure) was compliant with the related performance requirement.

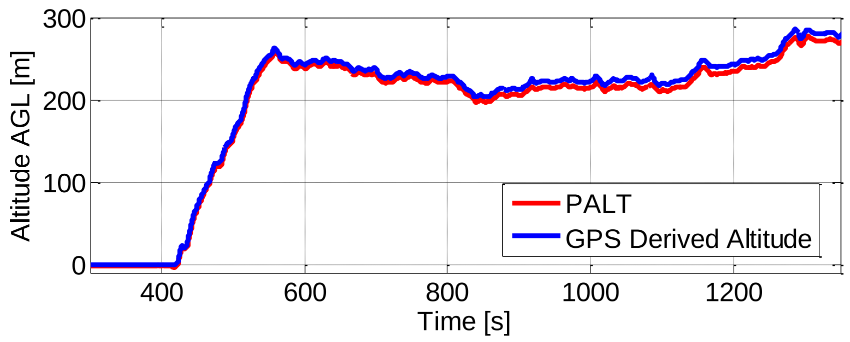


Figure 6-50: Comparison between the pressure altitude and the GPS altitude.

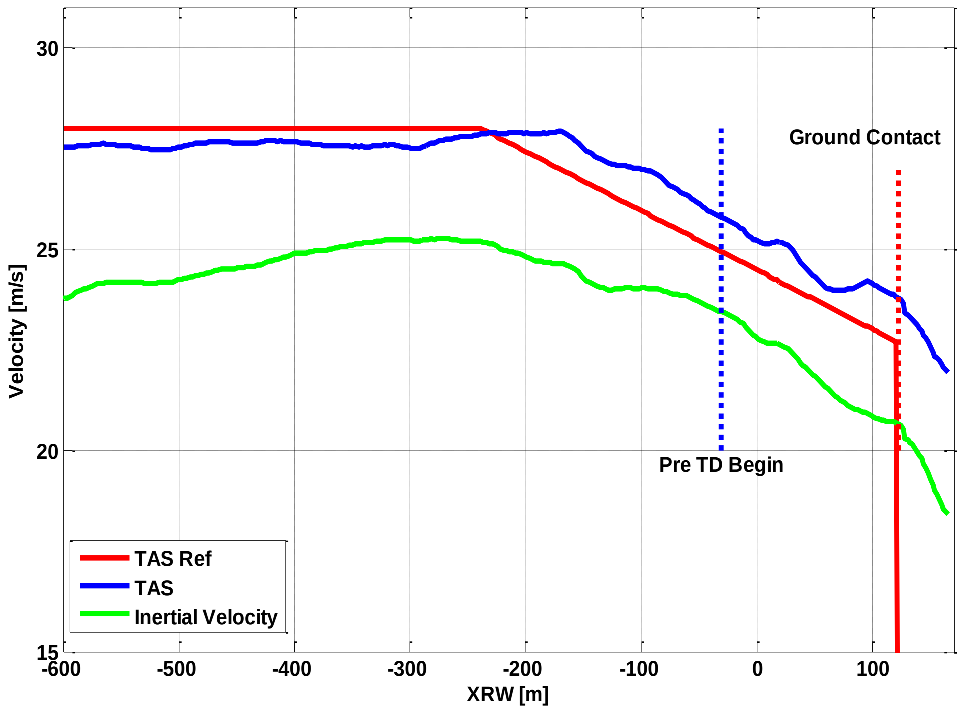


Figure 6-51: TAS and inertial speed during Flare and Pre-Touch Down phases

6.3.3 Model Based Virtual Sensors Experimental Result

In the following figure you can find the results of three Flight tests where the Model Based Virtual Sensors outputs described by (Eq.53) and (Eq.54) and the IAS achieved by the commercial ADC installed on the FLARE were compared. The difference has been considered satisfactory.

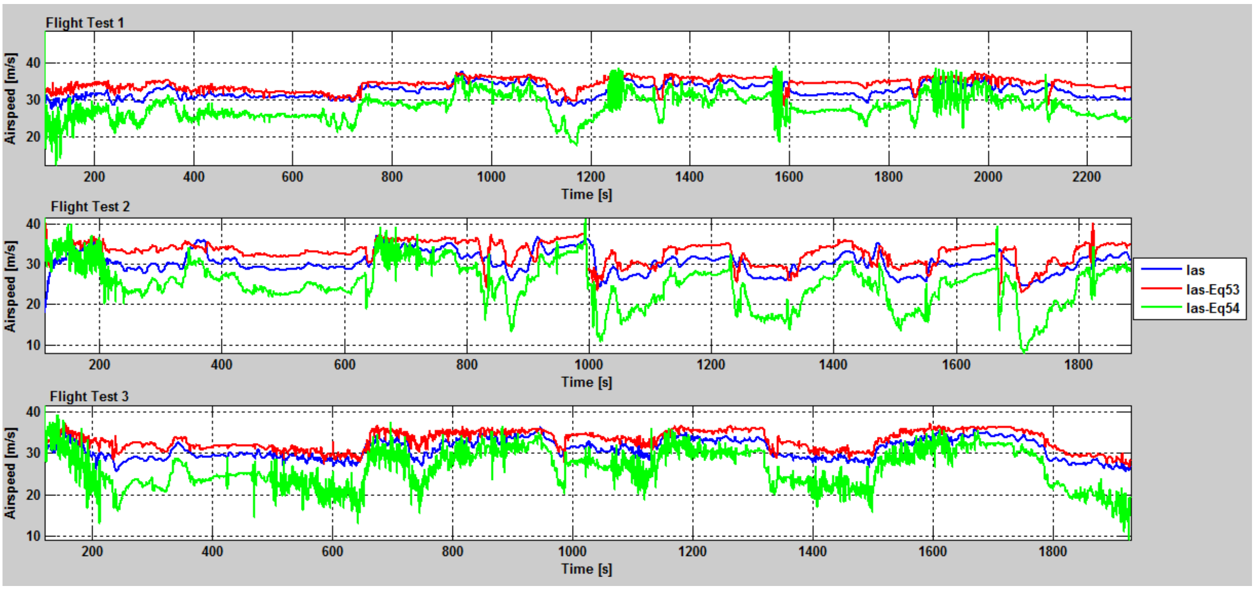


Figure 6-52 - Model Based Virtual Sensors Experimental Assessment

7 CONCLUSIONS

Several research activities have been developed in recent years in order to expand the flight envelope and to improve safety levels of modern aircrafts, both manned and unmanned. In particular, several efforts were dedicated in achieving a satisfactory functional redundancy without the necessity and the drawbacks of physical HW redundancy.

Crucial states of the A/C are the one related to air data and, therefore, the ADS (Air Data System) is a very critical device or feature (in the case the functionality is implemented in a more general FCC). The main aim of an ADS is to provide accurate and reliable air data to the Flight Control System above all during critical flight phases like take-off and landing.

Among other air data the angle of attack control performed by a pilot is decisive for avoiding one of the most important causes of Loss of Control accidents. Whether during an emergency situation or during a routine flight operation, knowledge of an angle of attack of an aircraft can help pilots of manned aircrafts or remote pilots of UAVs in performing safer and more stable manoeuvers.

Several tests have confirmed that a custom ADS, even if realized only using commercial of the shelf components, could be able to fit the stringent requirements imposed by the flight control system.

In the framework of this doctorate two different approaches have been explored by the author for developing a custom ADS: a model based estimation algorithm and a related FDIR and neural network based algorithm.

Effectiveness of each developed algorithm has been demonstrated using numerical simulations, HW-in-the-loop real time simulations and flight testing in relevant scenarios. All the algorithms were developed and validated at CIRA (Italian Research Aerospace Center) in the framework of TECVOL project. In the framework of this project the developed systems were tested on a very light aircraft named FLARE, therefore the choice of the sensors and the hardware implementation is applicable to this class of vehicle, whereas the software architecture can be reused for air data systems designed for other classes of aircraft.

The design process of the model based ADS started from the identification of system requirements. They define the raw flight data to be measured and the sensors performances, in terms of data accuracy, output rate and latency, which are essentially due to the constraints of the flight control system. This models used for estimating the air data were identified by analysis of FLARE flight tests. Different estimation algorithms were designed and exploited in conjunction to the classical air data equations for developing a reliable ADS and related FDIR features. The experimental results were satisfactory but they were reliable only if applied to a specific flying platform. To overtake this drawback another approach relating to neural networks has been addressed.

A neural network architecture for fault detection and isolation in Air Data Systems has been developed. All the design phases have been successfully completed, starting from the design of the ANN using the CRISP-DM methodology to the testing with the flight data.

The training dataset for the learning phase of the design has been acquired by both numerical simulations performed in laboratory and flight data recording. Different architectures have been designed for different air data measurements (e.g. indicated airspeed, angle of attack, etc.) and for different possible sets of measures to be used as inputs. The measures’ sources taken into account are inertial sensors, GPS, pressures sensors. A promising angle of attack estimator has been developed considering the above mentioned sources.

Though this approach is general and can be used on each possible aerodynamic configuration, the preliminary flight tests necessary to achieve a sufficient dataset for training and testing the neural network still represents a limit. In particular, in case the aircraft changes, the virtual sensors shall be retrained. However these limitations can be overcome by using datasets generated by simulation models, if they are reliable, or updating the training of the neural network online, if some changes are made to the aircraft. Therefore a further development of the proposed algorithms will regard a method to perform continuous assessment of the neural network performance and, in case of performance downgrade, to update the virtual sensors by adjusting its parameters online.

8 AUTHOR REFERENCES

[AR1] Garbarino, Genito, Baraniello, De Lellis, Vitale***, "Low Cost Air Data Computer for UAV in-flight Experimentation" –*** 23th Annual European Chapter Symposium at the Society of Flight Test Engineers, Amsterdam (Netherlands), June 2012

[AR2] Fedele, Genito, Vitale and Garbarino, "***Experimental aircraft system identification from flight data: Procedures and Results***" – CEAS 2013, 4th European Air and Space Conference, Linköping (Sweden), September 2013

[AR3] Garbarino, Zazzaro, Genito, Fasano and Accardo***, "Neural Network Based Architecture for Fault Detection and Isolation in Air Data System" –*** 32nd Digital Avionics Systems Conference, Syracuse (NY), October 2013

[AR4] Orefice, Di Vito, Garbarino, Corraro, Accardo, Fasano, ***"Real-Time Validation of an ADS-B Based Aircraft Conflict Detection System"***, in Proceedings of the American Institute of Aeronautics and Astronautics (AIAA) SciTech 2015 Conference, , pp. 15, Kissimmee, Florida, 5-9 January, 2015

[AR5] De Lellis, Corraro, Garbarino, Di Vito Vittorio, "***Design process and real-time validation of an innovative autonomous mid-air flight and landing system***" – WASET 2011, July 2011, Paris, ISSN 2010-376X, pp. 97-107

[AR6] N. Genito, C. Marrone, E. De Lellis, L. Garbarino, *Autonomous Take Off System: Development and Experimental Validation*, XXI Congresso Nazionale AIDAA – III CEAS Air&Space Conference, Venezia, Ottobre 2011

9 BIBLIOGRAPHY

[B1] ACARE Flight Plan 2050. Europe’s Vision for Aviation. 2011

[B2] ACARE Strategic Research and Innovation Agenda, 2012.

[B3] EREA Vision for the Future. Towards the future generation of the Air Transport System

[B4] Youmin Zhang, Jin Jiang, “Bibliographical review on reconfigurable fault-tolerant control systems”, Annual Reviews in Control 32 (2008) 229–252.

[B5] M. Oosterom, R. Babuska, “Virtual Sensor for the Angle-of-Attack Signal in Small Commercial Aircraft”, 2006 IEEE International Conference on Fuzzy Systems, 16-21 July, 2006.

[B6] Samara P., Fouskitakis G., Sakellariou J., Fassois S. Aircraft angle-of-attack virtual sensor design *via* a functional pooling NARX methodology. Proceedings of the European Control Conference; University of Cambridge, UK. September, 2003; p. 309.

[B7] Oosterom M., Babuska R. Virtual Sensor for Fault Detection and Isolation in Flight Control Systems—Fuzzy Modeling Approach. Proceedings of the 39th IEEE Conference on Decision and Control; Sydney, Australia. December, 2000; pp. 2645–2650.

[B8] Tomczyk A. Simple Virtual Attitude Sensors for General Aviation Aircraft. Aircr. Eng. Aerosp. Technol. 2006;78:310–314.

[B9] Ollero A., Merino L. Control and Perception Techniques for Aerial Robotics. Annu. Rev. Control. 2004;28:167–178.

[B10] Napolitano M., Windon D., Casanova J., Innocenti M., Silvestri G. Kalman Filters and Neural-Network Schemes for Sensor Validation in Flight Control Systems. IEEE Trans. Control Syst. Technol. 1998;6:596–611.

[B11] Drozeski G., Saha B., Vachtsevanos G. A Fault Detection and Reconfigurable Control Architecture for Unmanned Aerial Vehicles. Proceedings of the IEEE Aerospace Conference; Big-Sky, MT, USA. March, 2005.

[B12] Heredia G., Ollero A., Bejar M., Mahtani R. Sensor and Actuator Fault Detection in Small Autonomous Helicopters. Mechatronics. 2008;18:90–99.

[B13] Heredia G., Ollero A. Sensor Fault Detection in Small Autonomous Helicopters using Observer/Kalman Filter Identification. Proceedings of the 5th IEEE International Conference on Mechatronics; Málaga, Spain. April, 2009.

[B14] Nebula, R. Palumbo, G. Morani, and F. Corraro, “Virtual Air Data System Architecture for Space Reentry Applications”, Journal of spacecraft and rockets, Vol. 46, No. 4, July–August 2009

[B15] Huo Y., Ioannou A. and Mirmirani M., “Fault tolerant control and reconfiguration for high performance aircraft: review”, CATT technical report 01 – 11 – 01, 2001.

[B16] Patton R. J., Frank P. M. and Clark R. N., “Issues of fault diagnosis for dynamic systems”, Springer-Verlag, 2000.

[B17] Chen J., Patton R. J. and Chen Z., “Active fault-tolerant flight control systems design using the linear matrix inequality method”, Trans. of the Institute of Measurement and Control, 21, 1999.

[B18] Pachter M. and Huang Y. S., “Fault tolerant flight control”, Journal of Guidance, Control and Dynamics, 26 (1) January-February 2002.

[B19] Clark R.N., “Instrument fault detection”, IEEE Trans. Aerosp. Electron. Syst., vol. 3, pp. 456–465, 1978.

[B20] Bucy, R.S. and Joseph, P.D.,”Filtering for Stochastic Processes with Applications to Guidance”, John Wiley & Sons, 1968; 2nd Edition, AMS Chelsea Publ., 2005. ISBN 0-8218-3782-6.

[B21] J. Y. Keller and D. Sauter, "Kalman filter for discrete-time stochastic linear systems subject to intermittent unknown inputs," IEEE transactions on Automatic Control, vol. 58, pp. 1882-1887, 2013.

[B22] H.W. Sorenson (ed), Kalman Filtering: Theory and Applications, IEEE Press, 1985

[B23] A. Gelb, Applied Optimal Estimation, MIT Press, 1974, Chapter 6

[B24] M.S. Grewal and A.P. Andrews, Kalman Filtering, Prentice Hall, 1993, Chapter 5

[B25] Dan Simon, “Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches”, ISBN: 978-0-471-70858-2, July 2006, Wiley

[B26] Gertler , J. Li, B. “Analytical redundancy methods in fault detection and isolation”, Proceedings of the IFACIIMACS Symposium SAFEPROCESS 9, pp. 9-21, 1991, Baden-Baden

[B27] Chow, E. Y.; Willsky, A. S., “Analytic redundancy and the design of robust fault detection systems”, IEEE Transaction on Automatic Control, Vol. 29, Jul. 1984, 603-614

[B28] Ding, X., L. Guo and T. Jeinsch, “A characterization of parity space and its application to robust fault detection”, IEEE Transactions on Automatic Control 44 (2), 337–343, (1999)

[B29] Gustafsson, Fredrik, “Stochastic fault diagnosability in parity spaces”, In:Proceedings of the 15th IFAC World Congress, Barcelona, Spain, 2002

[B30] Ali Zolghadri, David Henry, Jérôme Cieslak, Denis Efimov, Philippe Goupil, “Fault Diagnosis and Fault-Tolerant Control and Guidance for Aerospace Vehicles: From Theory to Application”, Springer Science & Business Media, 2013.

[B31] Widrow,B., and S.D. Stearns, 1985. “Adaptive Signal Processing, Englewood” Cliffs, NJ: Prentice-Hall

[B32] Haykin, S., 2002. Adaptive Filter Theory, 4th ed., Englewood Cliffs, NJ: Prentice Hall

[B33] Geman, S., E. Bienenstock, and R. Doursat, 1992. “Neural networks and the bias/variance dilemma,” Neural Computation, vol. 4, pp. 1–58.

[B34] Grossberg, S., 1988. Neural Networks and Natural Intelligence, Cambridge, MA: MIT Press.

[B35] McCulloch,W.S., and W. Pitts, 1943.“A logical calculus of the ideas immanent in nervous activity, ”Bulletin of Mathematical Biophysics, vol. 5, pp. 115–133.

[B36] Mokhtar S. Bazaraa, Hanif D. Sherali, C. M. Shetty, “Nonlinear Programming: Theory and Algorithms”, Wiley, 2006, ISBN-13: 978-0471486008

[B37] Rumelhart,D.E., and J.L. McClelland, eds., 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, Cambridge, MA: MIT Press.

[B38] Kramer, A.H., and A. Sangiovanni-Vincentelli, 1989. “Efficient parallel learning algorithms for neural networks,” Advances in neural Information Processing Systems, vol. 1, pp. 40–48, San Mateo, CA: Morgan Kaufmann

[B39] Balázs Csanád Csáji. “Approximation with Artificial Neural Networks”, Faculty of Sciences; Eötvös Loránd University, Hungary

[B40] Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

[B41] Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4(2), 251–257.

[B42] Hassoun, M. (1995) Fundamentals of Artificial Neural Networks MIT Press, p. 48

[B43] Haykin, Simon (2009). Neural Networks and Learning Machines Third Edition. Upper Saddle River, New Jersey: Pearson Education Inc. ISBN 978-0-13-147139-9

[B44] Hassibi,B., and T. Kailath, 1995.“Hq optimal training algorithms and their relation to back propagation,”Advances in Neural Information Proccessing Systems, vol. 7, pp. 191–198.

[B45] Robbins, H., and S. Monro, 1951. “A stochastic approximation method,” Annals of Mathematical Statistics, vol. 22, pp. 400–407

[B46] Jacobs, R.A., 1988. “Increased rates of convergence through learning rate adaptation,” Neural Networks, vol. 1, pp. 295–307

[B47] D. T. Ward, and T. W. Strganac, “Introduction to Flight Test Engineering”, Second Edition, Kendall/Hunt Publishing Company 2001

[B48] Roskam J., Airplane Flight Dynamics and Automatic Flight Controls, 1st ed., DARcorporation, Lawrence KS, 2001, Chap. 1

[B49] Tan P., Steinbach M., Kumar V., *Introduction to Data Mining*, Pearson Addison Wesley, 2005

[B50] Chapman P., Clinton J., Kerber R., Khabaza T., Reinartz T., Shearer C., Wirth R., CRISP DM 1.0. Step by step data mining guide, SPSS, 2000.

[B51] Aapo Hyvärinen, Juha Karhunen, Erkki Oja,” Independent Component Analysis”,Wiley, ISBN: 978-0-471-40540-5, 2001.

[B52] Khalid, S.; Khalil, T.; Nasreen, S. "A survey of feature selection and feature extraction techniques in machine learning", *Science and Information Conference (SAI), 2014,* On page(s): 372 – 378.

[B53] Hastie,T., R.Tibshirani, and J. Friedman, 2001. The Elements of Statistical Learning: Data Mining,Inference, and Prediction, New York: Springer.

[B54] Cover, T.M. (1965). "Geometrical and Statistical properties of systems of linear inequalities with applications in pattern recognition". IEEE Transactions on Electronic Computers. EC-14: 326–334. doi:10.1109/pgec.1965.264137.

[B55] MacQueen, J. B., ”Some Methods for classification and Analysis of Multivariate Observations”, Proceedings of 5th Berkley Symposium on Mathematical Statistics and Probability. University of California Press., pp. 281, 1967

[B56] Shin, K. G. and Hagbae, K., “A Time Redundancy Approach to TMR Failures Using Fault-State Likelihoods”, IEEE Trans. Comput., 43(10): 1151–1162, 1994

[B57] Cary R. Spitzer, “The Avionic Handbook”, by Cary R. Spitzer, 2000

[B58] Vaidya, N. H. and Pradhan, D. K., 1993. Fault-Tolerant Design Strategies for High Reliability and Safety, IEEE Trans. Comput., 42(10): 1195–1206

[B59] Lala, J. H. and Harper, R. E., 1994. Architectural Principles for Safety-Critical Real-Time Applications, Proc. IEEE, 82(1): 25–40.

[B60] Anderson, T. and Lee, P. A., 1981. Fault Tolerance, Principles and Practices, Prentice-Hall, London.

[B61] Willsky, A. S., 1980. Failure Detection in Dynamic Systems, Fault Tolerance Design and Redundancy Management Techniques, AGARD-LS-109

[B62] Sensor Technics 144S PCB Series, November 2002, available at http://www.sensortechnics.com/cms/upload/datasheets/ds_standard-144s-pcb-005.pdf

[B63] Sensor Technics HCX Series, March 2004, available at http://www.picmicros.org.uk/Interfacing/DATA/HCX%20Pressure%20sensors%20(Sensor%20Technics).pdf

[B64] SENECA K109PT PT100 Converter, available at http://www.farnell.com/datasheets/63957.pdf

[B65] LABFACILITY, DRG 010632A - PT100, Temperature Sensor, available at http://www.farnell.com/datasheets/50450.pdf

[B66] SpaceAge Control Mini Air Data Boom 100400, 1999.

[B67] N. Genito, C. Marrone, E. De Lellis, A Real-Time Landing Gear Simulation Model for a Very Light Aircraft: Development and Experimental Validation, AIDAA, Milano, Luglio 2009

[B68] RTCA DO-178C, Software Consideration in Airborne System and Equipment Verification, 2011.

[B69] RTCA DO-331, Model-Based Development and Verification Supplement to DO-178C and DO-278A, 2011.

[B70] Jategaonkar R.V., Flight Vehicle System Identification: A Time Domain Methodology, 1st ed., AIAA Progress in Aeronautics and Astronautics, Vol. 216, AIAA, Reston VA, 2006, Chap. 4

[B71] Witten H.I., Frank E., Data Mining. Pratical Machine Learning Tools and Techniques, Second Edition, Elsevier, Morgan Kaufmann Series, 2005

[B72] D. Gray “Using GPS to Accurately Establish True Airspeed”, Technical report, National Test Pilot School, available at http://contrails.free.fr/temp/TAS_FNL4.pdf, , June 1998.