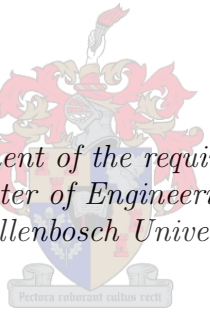
Fault Detection, Isolation and Reconfiguration for Autonomous Aircraft

by

Sandile Sithandwa Njabulo Memela



Thesis presented in fulfilment of the requirements for the degree of
Master of Engineering
at Stellenbosch University

Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Prof. Thomas Jones

December 2016

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2016

Abstract

This research is a continuation of a series of projects done at the Electronic Systems Laboratory (ESL) that deal with the design and development of Fault Tolerant Control Systems. The focus of this research is the design and development of a fault diagnostic system that allows an autonomous aircraft to automatically diagnose faults and reconfigure itself to maintain stable flight conditions. The motivation behind this research is to have the developed system be easily reconfigurable making it possible to use it in other UAVs at the ESL.

The Meraka Modular UAV, developed at the Council for Scientific and Industrial Research (CSIR), was chosen as a test-bed for the Fault Detection, Isolation and Reconfiguration (FDIR) System. A nonlinear mathematical model of the flight dynamics of the Meraka Modular UAV is derived, linearised and discretised for control and FDIR system purposes. A Failure Mode and Effects Analysis is performed in order to characterise and identify critical subsystem components as a means of guiding the fault modelling and identification process. The Hybrid Diagnosis Engine (HyDE), developed at NASA's Ames Research Center, and an Approximate Input Reconstruction Algorithm are used together to give the Meraka Modular UAV fault diagnostic capabilities. An expandable FDIR architecture is developed to facilitate the addition of more FDIR methods in an effort to allow the fault diagnostic system's functionality to be enhanced easily. The architecture was integrated based on the design of a simplified Integrated Vehicle Health Management (IVHM) System. This system consists of a Diagnostic Agent built into the ESL's custom Ground Station, the Meraka Modular UAV, and a Diagnostic System that runs on a Model B+ Raspberry Pi. A high fidelity software in the loop simulation environment was used to test the integrated fault diagnostic system to ensure that it will function as expected in the field when used by the Meraka Modular UAV and Ground Station Operator to perform diagnoses.

The fault diagnostic system is tested extensively by deliberately failing the UAV's actuators during simulated flight. The performance of the system was then verified by using flight test data collected by the Meraka Modular UAV during flight missions. When compared with the associated FDIR research, the FDI performance of the fault diagnostic system is found to be sensitive to the use of filters and relatively agnostic to actuator excitation. For the Meraka Modular UAV in particular, it is found that the fault diagnostic system performs as expected in the presence of disturbances and noise and improvements can be made by incorporating additional points of observation.

Uittreksel

Hierdie navorsing is deel van 'n reeks projekte wat by die Elektroniese Stelsels Labrotorium (ESL) onderneem is en wat te make het met die ontwerp en ontwikkeling van Fout Tolerante Beheersisteme, Die fokus van hierdie navorsing is die ontwerp en ontwikkeling van 'n fout diagnostiese sisteem wat dit moontlik maak vir 'n outomatiese vliegtuig om outomaties foute te diagnoseer en homself weer in te stel om stabiel te vlieg. Die motivering vir hierdie navorsing is om 'n sisteem te ontwikkel wat maklik heringestel kan word en in ander projekte onbemande vliegtuie (OV) by die Elektroniese Stelsel Labrotorium gebruik kan word.

Die Meraka Modulêre OV wat deur die Wetenskaplike en Nywerheidnavorsingsraad (WNNR) ontwikkel is, is gebruik vir die toets van die fout opspoor en herinstel sisteem. 'n Nie-liniêre wiskundige model van die vlug dinamiek van die Meraka Modulêre OV is afgelei en gelineariseer vir die beheer van Fout-deteksie, Isolasie en Herkonfigurasie (FDIH) sisteme. 'n Failure Mode and Effects Analise is gedoen om die kritiese subsisteem komponente se kenmerke te identifiseer om sodoende riglyne te bekom vir die fout modellering en identifikasie proses. Die Hibriede Diagnostiese Enjin (HyDE) wat by NASA se Ames Research Center, en 'n Approximate Input Reconstruction Algorithme word saam gebruik op die Meraka Modular UAV se fout diagnostiese vermoë te verskerp. Uitgebreide FDIH agitektuur is ontwikkel om die byvoeging van nog FDIH metodes te vergemaklik. Dit is gedoen om dit moontlik te maak om maklik die fout diagnostiese sisteem se werking te verbeter. Die argitektuur is geïntegreer baseer op die ontwikkeling van 'n vereenvoudigde Integrated Vehicle Health Management (IVHM) Sisteem. Hierdie sisteem bestaan uit n Diagnostiese Agent wat ingebou is in die ESL se custom grondstasie, die Meraka Modulêre OV, en 'n Diagnostiese Sisteem wat gebruik maak van 'n Model B+ Raspberry Pi. Hoëtrou sagteware in die simulasie omgewing is gebruik om die integreerde fout diagnostiese disteem te toets om sodoende seker te maak of dit soos verwag sal werk as die deur die Meraka Modulêre OV en Grondstasie Operateur gebruik word om ontledings te doen.

Die fout diagnostiese sisteem is deeglik toets deur om aspris die OV se aandrywers tydens die simuleerde vlug te laat faal. Hoe die sisteem presteer het is toe deur middel van die vlugdata wat deur die Meraka Modulêre OV gedurende die vlug versamel is, geverifeer. In vergeleke met die betrokke FDIH navorsing, is die FDI prestasie sensitief vir die gebruik van filters en beteklik agnosties teenoor die opwekking van die aandrywers. Veral in die geval van die Meraka Modular OV word daar gevind dat die fout diagnostiese sisteem soos verwag presteer as daar steurnisse en geraas is, en dat dit verbeter kan word as addisionele observasie punte inkorporeer word.

Acknowledgements

I would like to express my gratitude to the following people for their contribution. This project would not have been possible without their guidance and support.

- Prof. Thomas Jones for his enthusiasm for the project, continued advice, guidance and support.
- The lab engineers in the ESL for pointing me in the right direction and allowing me to disturb them while we anxiously did some problem solving.
- My friends for being there and enduring my ramblings about the worlds of fault diagnostics research.
- My family for their undying love, patience and support.

Contents

List of Algorithms

List of Figures

List of Tables

Nomenclature

Acronyms

AIRA	Approximate Input Reconstruction Algorithm
API	Application programming interface
ASCII	American Standard Code for Information Interchange
AVL	Athena Vortex Lattice
CAN	Controller Area Network
CG	Centre of Gravity
CPU	Central Processing Unit
CSIR	Council for Scientific and Industrial Research
DCM	Direction Cosine Matrix
DES	Discrete Event Systems
DOF	Degrees of Freedom
DTD	Document type definition
EOM	Equations of Motion
ESL	Electronic Systems Laboratory
FDD	Fault Detection and Diagnosis
FDI	Fault Detection and Isolation
FDIR	Fault Detection, Isolation and Reconfiguration
FMEA	Failure Mode and Effects Analysis
FTA	Fault Tree Analysis
FTC	Fault Tolerant Control
GME	Generic Modelling Environment
GUI	Graphical User Interface
HAM	Hazards Analysis Matrix
HIL	Hardware-in-the-loop
HyDE	Hybrid Diagnosis Engine

IMU	Inertial Measurement Unit
IVHM	Integrated Vehicle Health Management
MATLAB	Mathworks Simulation Program
MIMO	Multiple Input Multiple Output
MMAE	Multi Model Adaptive Estimator
NASA	National Aeronautics and Space Agency
NED	North East Down
NPRD	Nonelectronic Parts Reliability Data
OBC	Onboard Computer
QT	Cross platform application framework
RAM	Random Access Memory
RPN	Risk Priority Number
SHM	System Health Management
SIL	Software-in-the-loop
SISO	Single Input Single Output
TCP	Transmission Control Protocol
UAV	Unmanned Aerial Vehicle
VM	Virtual Machine
XML	Extensible Markup Language

Symbols

α	Angle of attack
β	Angle of side slip
$\delta\_$	Control surface deflection
δ	Actuator control input vector
ϕ	Roll angle
ψ	Yaw angle
τ	Engine lag time constant
θ	Pitch angle
Δ	Perturbation vector
Ω	Angular rate vector

Notation

$\overline{c}$	Mean aerodynamic chord

F	Force vector
f	System function
I	Mass moment of inertia tensor
u	System inputs
V	Velocity vector
x	System states
A	Aspect ratio
b	Wing span
C	Dimensionless aerodynamic constant
D	Down axis
E	East axis
e	Oswald efficiency factor
L	Roll moment
M	Pitch moment
m	Aircraft mass
N	Yaw moment or north axis
P	Roll rate
p	Perturbation roll rate
Q	Pitch rate
q	Perturbation pitch rate or dynamic pressure
R	Yaw rate
r	Perturbation yaw rate
S	Wing area
T	Thrust
U	Axial velocity
u	Axial perturbation velocity
V	Lateral velocity
v	Lateral perturbation velocity
W	Normal velocity
w	Normal perturbation velocity
X	Axial force
x	X axis

Y	Lateral force
y	Y axis
Z	Normal force
z	Z axis

Subscripts

A	Aileron
a	Aerodynamic force or moment
B	Body axes
D	Down axis
E	Elevator or east axis
F	Flaperon
g	Gravitational force or moment
I	Inertial axes
L	Roll moment
M	Pitch moment
N	North axis
N	Yaw moment
R	Rudder
T	Trim
t	Thrust force or moment
X	Axial force
Y	Lateral force
Z	Normal force

Chapter 1

Introduction

1.1 Autonomous Aircraft

The technological development over the past century has seen the increased use of automated and autonomous systems. UAVs (Unmanned aerial vehicles) are increasingly being used for economic benefit in civil and military operations [39]. Flight safety and reliability for these UAVs are vital and a IVHM (Integrated Vehicle Health Management) System is one of the measures available that can be used to ensure the optimal performance and safety of aerospace systems. This is achieved by observing the state or health of the vehicle and applying diagnostic and prognostic procedures to minimise the impact of faults when they occur and to help prevent critical system failure before it occurs through the use of FDIR (Fault Detection, Isolation and Reconfiguration) and related technology.

1.1.1 Problem Statement

The Electronic Systems Laboratory (ESL) in Stellenbosch University requires an FDIR System for the UAV technology currently being developed. The Meraka Modular UAV (Unmanned Aerial Vehicle) was chosen as a test-bed for the required FDIR system. The developed FDIR system must be capable of performing hybrid diagnosis using model based reasoning techniques. The motivation behind this work is to have the developed system serve as a platform for investigating hybrid diagnosis research within the ESL and for it to be easily reconfigurable making it possible to use it in other UAVs in the ESL.

The principal work required for this research is the practical implementation of a fault diagnosis and reconfiguration system that allows the Meraka Modular UAV to automatically detect significant changes in system behaviour and reconfigure the aircraft to maintain stable flight conditions in the presence of various faults detected in on-board sub-systems. The work includes the integration and expansion of associated FDI (Fault Detection and Isolation) and R (Re-allocation) research completed at the ESL. The objective here is to expand and improve upon the implemented systems using more current methods and technology. Once complete, system testing will be performed initially through a SIL (software in the loop) simulation followed by a HIL (hardware in the loop) simulation and a final flight test using the Meraka Modular UAV to demonstrate the developed system.

1.1.2 Research Scope

This research is a continuation of work done in the ESL towards the development of a fault tolerant and self-reconfiguring system. In these projects the aim was to develop controllers that would be robust to modelling errors and faults within the system model. The aim of this research is to develop a generalised fault diagnosis system that is housed in an expandable architecture. The majority of the focus in this research will be directed towards the development of this system and

the integration of developed FDIR technologies where possible. To this end, the Meraka Modular UAV will be able to autonomously diagnose faults in onboard subsystems and reconfigure itself to use fault-free subsystems given the level of the technology implemented.

1.1.3 Fault Tolerant Control Group

The Fault Tolerant Control Group within the ESL was created with the aim of developing FTC (Fault Tolerant Control) System technology that would allow UAVs to maintain stable flight in the presence of undesirable flight conditions and component failures.

This began with work done by Willem Basson [8] in 2011 focusing on the development and implementation of a Fault Tolerant Adaptive Control System for an Unmanned Aerial Vehicle on the Variable Stability UAV built at Stellenbosch University. He was able to show that adaptive control is effective as part of an integrated FTC System by demonstrating that the control system can re-stabilise a unstable aircraft due to damage-induced longitudinal shifts in the centre of gravity of the aircraft.

This was followed by work done by Lionel Basson [7] whose aim was the development of a control re-allocation (R) system to be used as part of the FTC architecture used in UAVs. The objective was to minimise the impact of faults that would necessitate the reconfiguration of the higher-level (control, guidance and navigation) systems on an aircraft. This was accomplished using a control allocation algorithm / system that solves a multi-optimisation problem. The developed system was implemented on the Meraka Modular UAV and was found to be applicable to a number of different conventional aircraft configurations.

In 2012, Hendrik Odendaal [38] developed an architecture capable of performing Fault Detection and Isolation (FDI) as part of the FTC System developed at the ESL. He implemented this system based on the Meraka Modular UAV and analysed the multiple model adaptive estimator (MMAE) and parity space passive fault detection and isolation methods. This resulted in the appropriate use cases for the different methods where the MMAE method was found to be more accurate in isolating faults and the parity space method was found to be more sensitive to the occurrence of faults.

1.1.4 Fault Detection, Isolation and Reconfiguration

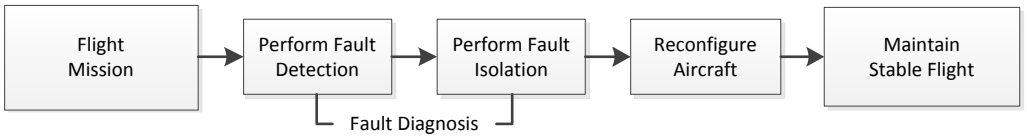


Figure 1.1: *FDIR System, showing the different FDIR stages.*

The following stages, as shown in Figure 1.1 above, describe the operation of the an FDIR System.

Stage 1 - Flight mission: At this stage the aircraft's guidance system maintains control of the aircraft and guides it along a specified flight trajectory. This represents normal aircraft behaviour at trim and allows for the simulation of the occurrence of faults by injecting faults into the system.

Stage 2 - Monitor aircraft behaviour: The aircraft's behaviour is continuously monitored at this stage in order to detect any simulated faults. This is done while the aircraft continues normal operation and any faults detected will need to be diagnosed.

Stage 3 - Perform fault isolation: After the detection of a fault, the FDIR system will attempt to determine what has caused the fault by isolating the root cause using the available sensor data and

system model. This information is then packaged in a form that can be used to reconfigure the aircraft.

Stage 4 - Reconfigure aircraft: The packaged information determined during the fault diagnosis stage is used to reconfigure the system. The reconfiguration of the aircraft is based on the fault and the available fault mitigation procedures needed in order to restore nominal flight operation using the existing fault free subsystems.

Stage 5 - Maintain stable flight: At this stage the FDIR system uses the new system configuration as the new model for the aircraft in order to maintain stable flight. The flight mission proceeds until such a time that there is a significant change in the aircraft's behaviour.

1.2 Fault Management

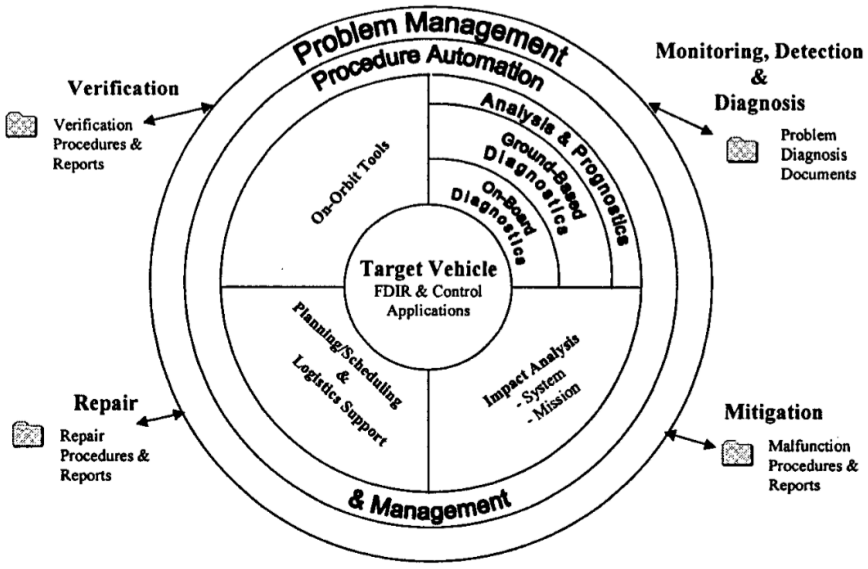


Figure 1.2: *IVHM Fault Life Cycle Model, showing the 4 different phases, sourced from [6].*

FDIR (Fault Detection, Isolation and Reconfiguration) Technology lies at the core of the IVHM Fault Life Cycle, which consists of Monitoring, Detection and Diagnosis, Mitigation, Repair and Verification as shown in Figure 1.2. Each quadrant in the Fault Management Model represents a different phase in the development of the IVHM System. Noteworthy elements here are the Target Vehicle, Onboard and Ground Based Diagnostics which inform decision processes used in the selection of the systems and FDIR technologies for this research project. It is from this model that the conceptual framework for IVHM Technology is developed.

1.2.1 Conceptual Framework

Figure 1.3 shows the conceptual framework for an IVHM System, here the various components interact to perform the function of Problem Management. The architecture can be grouped into three high level systems, on-board systems (left), operations support (top right) and maintenance and logistics support (bottom right).

The components in the on-board system serve different functions and of note is the diagnostic system which performs fault diagnosis using information retrieved from on-board subsystems. Within operations support, of note is the diagnostic agent and the fault history repository. These two components complement the function of the on-board diagnostic system at the ground station by

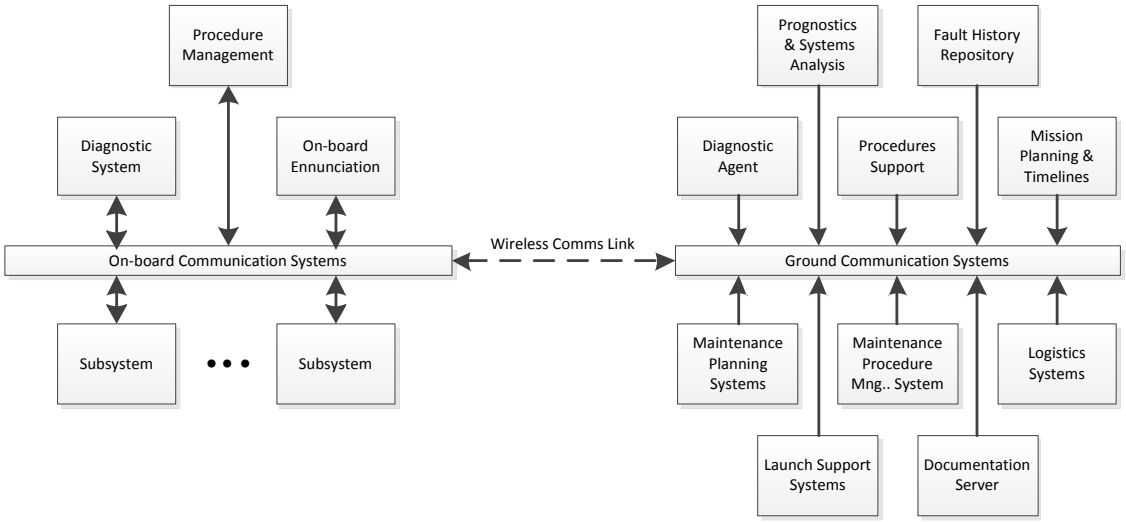


Figure 1.3: *Integrated Vehicle Health Management Blueprint, showing the onboard and ground based systems, adapted from [6].*

providing fault history and further diagnostic capability. This applies in cases where more processing is required or when dealing with systems and data that aren't time sensitive relative to the on-board operations. It is from this framework that further work for the project is based.

1.2.2 Fault Diagnostics Research

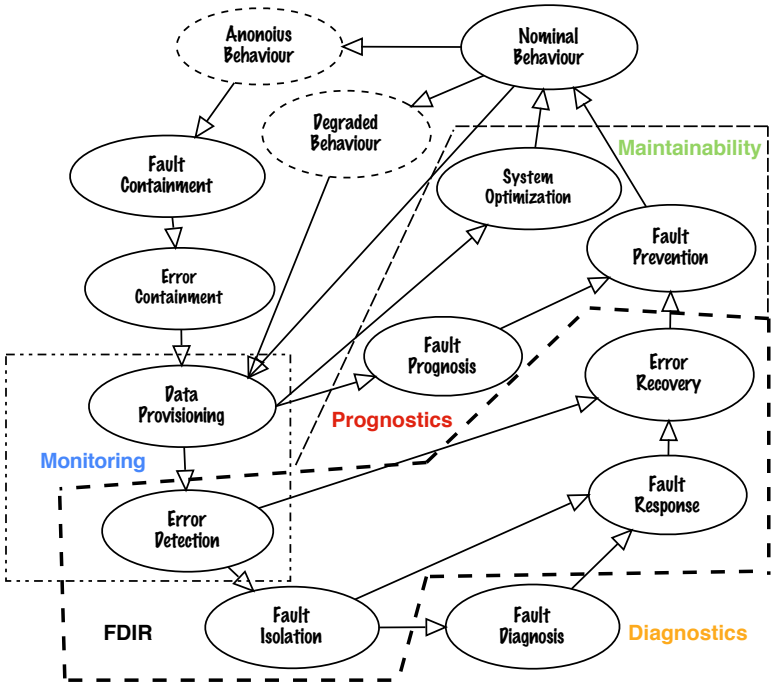


Figure 1.4: *IVHM Function Flow, showing the overlap between IVHM research, adapted from [28].*

The diagnostic system to be developed is described as shown in Figure 1.4 above as an overview of the overlap between the various Health Management Technologies and Fault Management Phases. This figure creates a clear distinction between the different fields of research which make it relatively easier to identify the pertinent fields of research according to the desired functionality. In this case Fault Diagnosis which is relatively isolated and FDIR which is comprised of Error Detection, Fault Isolation, Fault Response and Error Recovery.

The following selection criteria, based on the problem statement, is used to identify the applicable fault detection and diagnosis approach before selecting the appropriate FDIR technology:

- supports the integration of various FDIR technologies
- detects significant changes in system behaviour
- detects various faults in onboard subsystems
- uses models or is based on models
- does not require data initially

There are different approaches to fault detection and diagnosis and these include: model based reasoning; causal models; fault signatures, pattern recognition, and classifiers; neural networks, procedural/workflow approaches; event-oriented FDI; passive system monitoring; rule-based approaches and implementations and hybrid approaches [1].

Model based reasoning is characterised by the use of models of the observed system to perform fault detection [18]. Models are able to capture the behaviour evolution of a system and enable the diagnosis of multiple faults. The model-based reasoning method is able to predict system behaviour and compare it with observations to perform fault detection. Causal models are an extension or special case of the model-based reasoning fault detection method. They are used to capture the cause and effect relationship about a system in a qualitative manner versus the quantitative manner employed in model based reasoning [49]. This is done by using bond graphs where the model is trained using measurements or human input with expert knowledge.

FDD methods based on fault signatures, pattern recognition, and classifiers identify faults by comparing patterns or fault signatures, which represent the known symptoms for enumerated faults, with observed symptoms [37]. This is done by using a classifier which is an algorithm that gets trained with data that enables it to learn what the best match is for solving a classification problem. This method is preferable when knowledge about the underlying dynamics of the system is not available and may be implemented with the use of neural networks.

Neural networks aka. Artificial Neural Nets (ANNs) are nonlinear, multivariable models that are trained by using input / output data [5]. They are particularly important in cases where knowledge about the functional relationships between the inputs and outputs is known. A major drawback is the danger that the network may extrapolate or generalise as a consequence of insufficient training data. However when combined with other FDD methods, fault detection performance may be enhanced to encompass nonlinearities commonly encountered in practical systems.

In procedural/workflow approaches to fault detection and diagnosis, the decision process for making decisions using observed data is modelled using decision trees [16]. These methods require expert knowledge acquired from years of experience and a major drawback is that they require human input and might not handle unexpected situations well. This fault diagnosis method is particularly well suited for guiding human operators and may be used for situations that are particularly well understood.

In event oriented fault detection and diagnosis an event represents the change of state of a monitored object [48]. Fault detection is based on using for instance alarms as events instead of conventionally a fixed set of variables and drawing conclusions for multiple events. Event oriented systems find uses in the control rooms in process plants and are used mainly with the goal of supporting business processes and increasing safety by analysing logs to optimise thresholds.

Passive system monitoring is a fault detection and diagnosis method where events suggesting the presence of faults are reported to the diagnostic system by subsystems such as agents. In this case,

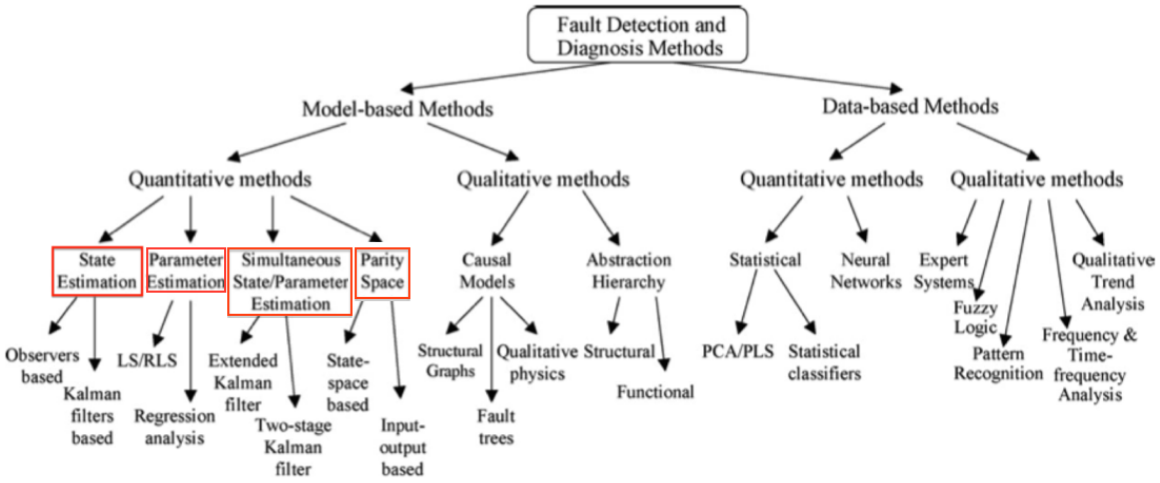
the agents routinely scan every variable of interest while the System Under Test (SUT) is observed during its normal operation [52]. This method finds use particularly in cases where the correctness of the system's behaviour must be ensured such as in network management systems.

Rule-based approaches and implementations function more as an interface or as a program control mechanism [12]. They are also known as expert systems and incorporate other fault detection methods rather than being a standalone fault detection and diagnosis technique. Rules are entered into the system and executed along with observed data using an inference engine which may request input from an operator. A difficulty inherent in using rule based systems is the encapsulation of changes over time which makes it difficult to see the bigger picture making maintenance and changes to the system difficult.

Hybrid approaches combine elements from distinct fault detection and diagnoses techniques to achieve the fault detection task [44]. One such example is the hybrid mode estimation system that uses a modelling formalism called concurrent probabilistic hybrid automata to combine a system's continuous dynamic models with hidden markov models to describe discrete changes in its behavioural modes [29].

Of the many different approaches mentioned, model based reasoning and hybrid diagnosis stand out as suitable methods for implementation. Based on the project statement, one can immediately see the applicability of the hybrid based approach since it is able to deal with systems with both discrete and continuous behaviour. In addition, sufficient data to train a data-centric FDD method is not readily available, and if used may lead to relatively poor diagnostic performance if the aircraft's dynamics change.

1.2.3 FDIR Technologies



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

Figure 1.5: *Fault Diagnostics Research, highlighting functional groupings between FDD technology, adapted from [27].*

Research into FDIR for aerospace applications has resulted in a vast amount of methods and technologies which can be used [53]. In cognisance of this, the search for applicable technologies has been constrained to readily available methods and technologies as shown in Figure 1.5. Implementing a practical system will require selecting the appropriate FDIR technology needed to accomplish the fault detection and diagnosis task. A diagnosis engine and quantitative method suited to modelling system behaviour and performing actuator FDI respectively will now be selected.

Diagnosis Engine

In his thesis, R Mohammadi [34] describes the use of a hybrid diagnoser, a framework to perform diagnosis in complex systems modelled using hybrid automata based on discrete event systems (DESs) that model banks of residual generators. In contrast to this, the work done in [43] also describes the development of a similar hybrid diagnosis system with the added capability of control reconfiguration. It is an object-based modelling framework which captures the time and event based dynamics of a system.

RODON is a commercial model based diagnosis engine used in the avionics and automotive industries. It was entered into the 20th International Workshop on Principles of Diagnosis (DX-09), a diagnostics competition defined by the NASA Ames Research Center, and tested on the ADAPT system [35]. LYDIA is another model based reasoning engine packaged as a software suite and designed for automated diagnosis [51]. It can be used in operator rooms in conjunction with a Supervisory Control and Data Acquisition (SCADA) System to visualise a computed Root-cause-of-failure (RCoF).

The Livingstone2 is a open-source Model-Based Diagnostic System [41] that was used onboard the X-37 spacecraft (an autonomous aircraft) to demonstrate the benefits of having an IVHM system onboard [10]. Livingstone2 is also a predecessor to HyDE (Hybrid Diagnosis Engine). HyDE is a diagnostic engine that combines control theory and stochastic diagnosis approaches to provide a general framework for diagnosing hybrid systems [11] and has been used in the FalconSAT-5 Sciencecraft [4]. HyDE was also used in the Drilling Automation for Mars Environment (DAME) Project [18] and formed part of the core diagnosis engine along with vibration classification and rule based diagnosis modules for enhanced fault diagnostics performance.

Diagnosis Engines			
Criteria	LYDIA	RODON	HyDE
Scalable Architecture	√	√	√
Hybrid Systems	√	√	√
Stochastic Reasoning	√	√	√
Multiple Faults	√	√	√
Licensing	√	x	√
FMEA	+	√	*
Embeddable	*	√	√
MATLAB Integration	+	√	+
Operating System	Windows, Linux	Windows	Windows, Linux, Solaris, VxWorks

Table 1.1: *A comparison between diagnosis engines [1, 51, 35, 11] Key: √ favourable, *less favourable, x not favourable, + applicable, - not applicable*

√ favourable, * less favourable, x not favourable, + applicable, - not applicable
Comparison between different fault detection methods \cite{stanley2012, feldman2006, bunus2009, hyde2000}

Following a more thorough investigation of the diagnosis engines [15], the notable diagnosis engines discussed were evaluated to determine their suitability for the project. Table 1.1 lists the selected diagnosis engines along with the relevant selection criteria derived from the project statement and scope. Licensing (freely available), Embeddable and MATLAB Integration are the mostly highly ranked (critical) selection criteria that informed the decision process to determine suitability.

Overall RODON ranks as the most functionally capable diagnosis engine according to the selection criteria and HyDE ranks as the most suitable diagnosis engine for this project according to the critical selection criteria. In addition to the system evaluation criteria in Appendix A.10.4, HyDE

(Hybrid Diagnosis Engine) was selected as the engine of choice to achieve the desired system functionality based on the results shown in Table 1.1.

Quantitative Method

Fault Tolerant Control Systems (FTCS) can be classified into active and passive fault tolerant control systems [27]. In passive fault detection the system's response is monitored and in active fault detection the system is excited by injecting external excitation signals [42]. In this research project, a passive fault tolerant control system is developed. State estimation, parameter estimation, parity space and a mixture of these are the four most commonly used FDD technologies or methods as highlighted in Figure 1.5 [27].

State estimation techniques are based on observers and Kalman Filters. A Kalman filter is an algorithm that makes it possible to estimate the states of the system as the system evolves in time and new measurements are taken [56]. The Kalman Filter is used in fields such as GPS, robotics, and even computer vision. Banks of Kalman Filters are used to perform fault detection by tuning each to identify a particular fault [55]. Parameter estimation fault detection and diagnosis methods are comprised of the least squares or recursive least squares and regression analysis methods. Least squares techniques are used to calculate estimates of system parameters through data fitting in order to create an input-output relationship based on a mathematical model [36]. This mathematical model is then associated with states to enable sensor and actuator detection.

The parity space method can be classified as being either input-output based or state space based. It is used to compute a residual vector that is nonzero in the presence of faults and can be highly sensitive to noise [30]. The residual used relies on analytical redundancies between the input and output and an unknown initial state, properties extracted from this signal are then used to perform fault detection.

Fault Detection and Diagnosis Methods

Criteria	Kalman Filter	Parity Space	Least Squares
Actuator Fault Detection	√	+	√
Computational Complexity	*	√	√
Multiple Faults Identifiable	√	*	√
Nonlinear Systems	√	√	+
Robustness	+	√	+

Table 1.2: *A comparison between fault detection methods [27]. Key: √ favourable, * less favourable, x not favourable, + applicable, - not applicable*

√ favourable, * less favourable, x not favourable, + applicable, - not applicable
Comparison betweendifferent fault detection methods \cite{jiang2008}

Table 1.2 contains a list of all the relevant FDD methods implemented by the FTC Group in the ESL toward this research in addition to the Least Squares FDD Method most recently identified. Selecting an appropriate FDD method is a trade-off between functionality and practical feasibility. Given that the aim of this project is to develop a generalised diagnosis engine housed in an expandable architecture (in an embedded system), the appropriate FDD method(s) can be selected and combined if and when needed for the particular diagnosis task. However, currently this project serves to suggest the practical feasibility of this system and computational complexity and multiple faults identifiable are the critical selection criteria (and constraints) used to select the most suited FDD method based on the project statement and scope.

Functionally the Kalman Filter is the most highly ranked FDD method according to Table 1.2 and [27] however it is relatively more computationally intensive and would be unsuitable for a generalised diagnosis engine that is yet to be fully developed and understood. Given that the

Kalman Filter and Parity Space FDD methods are relatively well understood within the FTC Group in the ESL and that the Least Squares Method satisfies the critical selection criteria for computational complexity and multiple identifiable faults, the Least Square Method was selected for use within this research project. A discussion with results is given in Section 5.4.2 to contrast the performance of the Parity Space Method with that of the Least Squares Method.

1.3 Proposed System

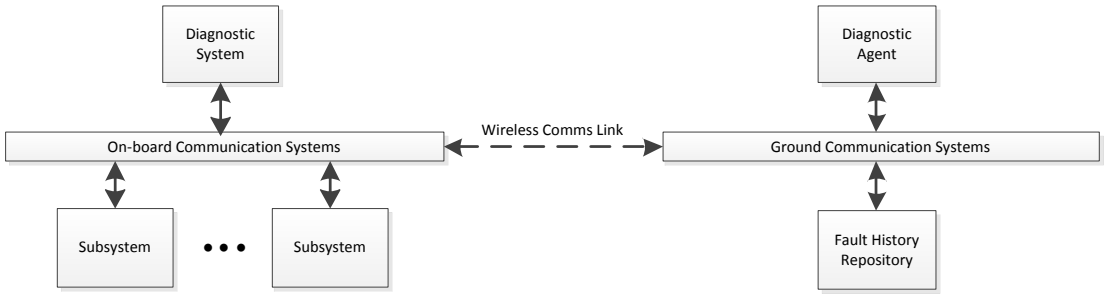


Figure 1.6: *A simplified Integrated Vehicle Health Management System, adapted from [6].*

The developed system is based on HyDE and the simplified IVHM System shown in Figure 1.6 where the Diagnostic System performs diagnoses onboard and is complemented by the Diagnostic Agent in the Ground Station. This research is a generalisation of the work done by Hendrik Odendaal and incorporates the fault decision from the FDI System component shown in Figure 1.7 into a system level reasoning engine. Focus was initially directed towards performing FDIR functionality based on the work done by Hendrik Odendaal and later shifted toward incorporating this work within HyDE to enable the diagnosis of on-board sub-systems. We use HyDE along with AIRA in order to perform actuator FDI through input reconstruction since the control surfaces don't have position sensors.

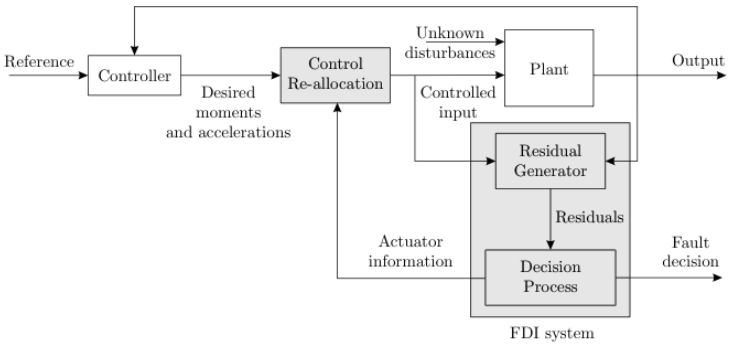


Figure 1.7: *FTC architecture with a FDI and control re-allocation module, sourced from [38].*

1.3.1 System Development

The System Architecture shown in Figure 1.8 depicts the UAV at the systems level. The Control System represents the FTC architecture shown in Figure 1.7 and is merely one of several of the UAV's sub-systems. The Diagnostics block represents the Diagnostic System that is the subject of this research. Its main function is to aggregate the fault diagnostics data that it collects from the various onboard sub-systems to inform decisions on how it reasons about the current state of the aircraft before reporting to the Guidance System.

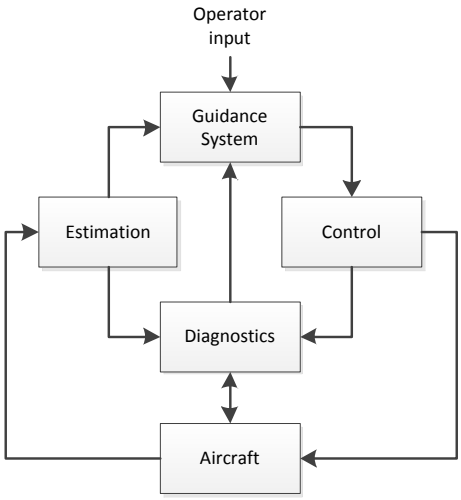


Figure 1.8: *System Architecture, showing the UAV's major subsystems along with the Diagnostic System.*

Fault Diagnostic System		
Function	Kalman, Bayes	Parity, CUSUM
Multiple Simultaneous Faults	-	-
Actuator Isolation (Left/Right)	√	√
Aileron Control Surfaces	√	√
Elevator Control Surfaces	√	√
Flaperon Control Surfaces	√	√
Rudder Control Surfaces	√	√
Actuator Nominal Mode	√	x
Actuator Stuck Faults	√	√

Table 1.3: *Implemented features of the FTC Group's relevant Fault Diagnostic Systems [38], key ✓ implemented, x not implemented, - not investigated.*

Implemented features of the FTC Group's relevant Fault Diagnostic Systems \cite{odendaal2012}, key √ implemented, x implemented, - not investigated.

The Fault Diagnostics Systems listed in Table 1.3 were implemented in the previous research project [38]. The MMAE method (Kalman) was combined with a Gaussian Bayes Classifier to create a Fault Diagnostic System. Similarly, the Parity Space method was combined with the CUSUM method to create another Fault Diagnostic System. In each case, the Gaussian Bayes Classifier and CUSUM methods were used to make fault decisions using observed data from the applicable FDD methods. Focus here was directed toward comparing the FDI performance of the two systems that detect single actuator stuck faults in order to gain a deeper understanding of the applicable use cases for each system.

The objective for this research project is to expand and improve upon these implemented fault diagnostic systems in order to allow the UAV to detect significant changes in system behaviour. The end goal is to enable the UAV to automatically diagnose faults in onboard subsystems and automatically reconfigure itself to maintain stable flight.

The Project Aim and Research Objectives Sections in Appendix A.1 and A.2 respectively encompass some of the major aspects of the vision for this line of research. Provisions are made for additional fault detection and reconfiguration technology during the design and development of the generalised fault diagnosis system however actual implementation of these technologies is beyond the scope of this research project.

This research project is the first step towards developing a generalised fault diagnostics system that will enable the integration and expansion of associated FDI [38], R [7] and active FTC [42] research already done by the FTC Group in the ESL at Stellenbosch University. This is achieved through the use of a diagnosis engine that is capable of performing system level reasoning by incorporating the fault decisions from the integrated FDIR and FTC technologies to create a Supervised FTC.

The iterative System Development Approach to be adopted in developing refined versions of this system is detailed in Appendix A and is outlined as follows:

- Model the aircraft and system faults - once the relevant FDIR methods have been selected, the aircraft and relevant subsystems along with their fault modes need to be modelled. This is done in Chapters 2 and 3.
- Implement FDIR methods selected - AIRA is investigated and implemented for each of the identified actuators in Chapter 3.
- Develop an expandable IVHM System architecture - the Fault Diagnostic System with HyDE at its core and that run's off the Model B+ Raspberry Pi is developed at this stage as outlined in Chapter 4.
- Integrate the FDIR technology - the complete IVHM System comprised of the Diagnostic Agent, Diagnostic System and Simulation Environment is integrated along with the applicable FDIR methods, namely AIRA, as detailed in Chapter 4.
- Get the existing aircraft system working - restore all associated hardware required for the experiments to working order, it is assumed that this is done as part of the System implementation step.
- System implementation - migrate the developed system from the development hardware into the test-bed system and integrate it with the relevant systems where necessary, this is done in Chapter 4.
- Simulate the integrated FDIR technology - the functionality of the developed systems is verified using a combination of SIL and HIL where applicable as outlined in Chapter 5.
- Perform a flight test - verify practical system performance by reusing collected flight data or performing a flight test as outlined in Chapter 5.

In developing this research project's diagnostic system which is based on HyDE, the following major developmental steps were taken:

- Perform a Reliability Analysis.
- Examine the existing systems and implement the required FDIR methods (AIRA).
- Develop models for the relevant systems.
- Model the relevant systems using the GME Tool with the HyDE Interpreter on Windows.
- Develop the Diagnostic Program on Windows using Microsoft Visual C++ 2010 Express.
- Simulate the applicable system using the developed models and a simulation environment (MATLAB).
- Migrate the Diagnostic Program to Linux and verify the Diagnostic Program's functionality.
- Migrate the Diagnostic Program to the Raspberry PI using the Netbeans IDE.
- Simulate and test the assembled embedded system in the target system.

1.3.2 Practical FDI

The interrelated concepts of states, behaviour, faults and failures within the context of System Health Management (SHM) are shown in Figure 1.9. According to [40] failure is the unacceptable performance of the intended function and a fault is defined as a physical or logical cause internal to the system, which explains a failure. A discrete fault is an abrupt and persistent change in a continuous system's behaviour. Actuators have several modes of operation, which include different kinds of faults, that give rise to several potential combinations of component configurations representing the different states of the system.

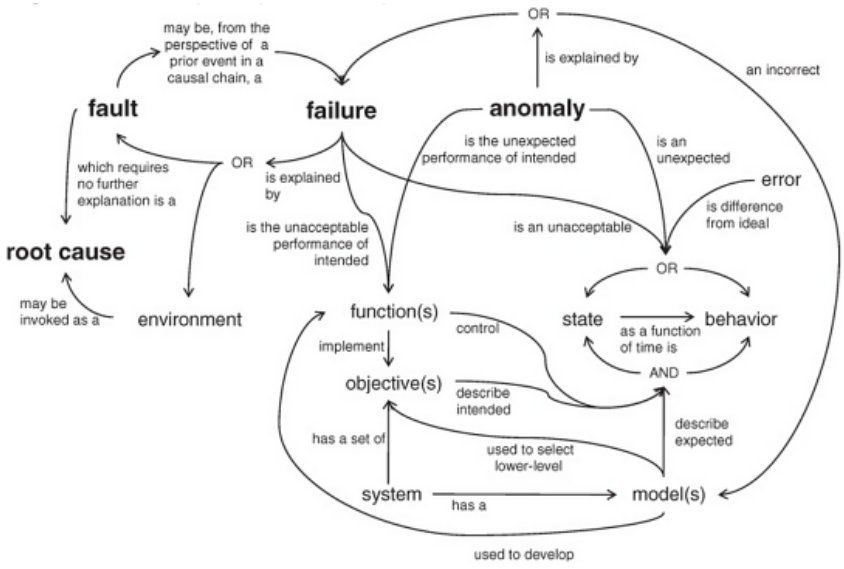


Figure 1.9: *Concept diagram for major SHM terms adapted from [40]*

In this research a passive FDI system is used to perform diagnosis and focus is limited to the actuators and critical sub-system components such as the avionics battery to reduce the computational complexity of tracking the system to be analysed. Practical FDI schemes are normally run online and are constrained to operating environments with limited computational capacity. An accurate mathematical model of the Meraka Modular UAV is available and is used to create a high fidelity simulation of the aircraft. It was also the deciding factor in selecting the model based reasoning approach adopted in this research to create a fault diagnostic system with improved fault detection capabilities.

1.3.3 Simulation Environment

Simulations were performed using MATLAB R2008b on an Intel Core i5 3.2 GHz computer with 4GB RAM by modelling the nonlinear dynamics of the UAV using Simulink. The Generic Modelling Environment (GME) Tool with HyDE interpreter was used to generate the system level representation (models) of the aircraft for use within the diagnostic system's program. MATLAB was interfaced with HyDE via an embedded function that communicates over a TCP connection with the diagnostic program in order to both simulate the dynamics of the aircraft and test the diagnostic engine during hypothesis testing. The control input history was recorded and reproduced during repeated hypothesis testing. The aircraft is put into trim when simulating straight and level flight. To simulate flight tests, actual flight test data is fed into the testbed system. A 700 MHz Model B+ Raspberry Pi with 512MB RAM is used during flight tests.

1.3.4 Available Sensors

The following sensors where used onboard the Meraka Modular UAV to collect data generated during practical flight testing:

- GPS - a differential GPS from Novatel
- accelerometers - a tri-axis inertial sensor from Analog Devices
- magnetometers - a three-axis high sensitivity magnetic sensor from Honeywell
- angle-of-attack and side slip sensor - a Mini Air data boom from Space Age Control
- static-pitot tube - a UAV airspeed and static pressure sensor from Freescale.

1.3.5 Practical Flight Testing

Practical flight tests are necessary in order to examine the performance of a fault diagnostics system under real world conditions. These flight tests were performed using the Meraka Modular UAV and faults were injected into the system to simulate actual actuator faults during normal flight. The data generated during these practical flight tests is used to simulate the actual flight dynamics of the UAV and to perform hypothesis testing equivalent to that done in [38].

1.4 Overview

There are six chapters in this thesis, following is an outline of each chapter.

Chapter 1 Introduction: This chapter gives an introduction to autonomous aircraft and puts into context the research topic of this thesis. It further highlights the fault management conceptual framework that forms the basis of the methodology followed in this research to develop and test the proposed system.

Chapter 2 Aircraft Model: The mathematical model used to simulate the dynamics of the Meraka Modular UAV is derived in this chapter. This begins with a description of the aircraft axis system used and the various frame transformations that can be done. This is followed by a derivation of the kinetic and kinematic equations used to describe the dynamics of the aircraft.

Chapter 3 System Modelling: In this chapter, the Hybrid Diagnosis Engine and the Approximate Input Reconstruction Algorithm is described. This is followed by a Fault Modelling and Effects Analysis (FMEA) and a description of the mathematical and declarative system models representing the Meraka Modular UAV and its sub-system components. These models along with the selected FDIR methods are used to model the different fault types that enable the FDIR System to perform fault diagnostics.

Chapter 4 System Integration: A description is given of the expandable FDIR architecture, based on HyDE, that is used in this research along with the implementation of the integrated FDIR System. This is followed by the methodology used to prepare and enable the integrated FDIR methods to perform diagnoses during aircraft simulation and flight tests. This chapter also outlines how the fault diagnostics system is utilised in the Meraka Modular UAV and by the ground station operator to perform autonomous fault diagnostics. A description is given to highlight how the Meraka Modular UAV maintains stable flight control and automated aircraft reconfiguration using fault-free subsystems.

Chapter 5 System Tests: The characteristics of the physical aircraft used and the system test conditions are given in this chapter. This is followed by the scenarios constructed in order to perform hypothesis testing during simulation and actual flight, and a comparison of the Fault Diagnostic System's performance as compared with associated FDIR research done in [38].

Chapter 6 Conclusion: A high-level summary of the work done in this research along with the results, conclusions drawn and recommendations for future work or possible improvements is given in this chapter.

Chapter 2

Aircraft Model

The theory behind the Meraka Modular UAV's aircraft model is based on the postgraduate module Advanced Automation 833 taught at Stellenbosch University. This aircraft model is based on an axis system that is used to adequately describe an aircraft and its behaviour. Modelling an aircraft's behaviour and subsequently its mechanics also requires kinematic and kinetic models. These models are then used to create a linearised model of the aircraft which is then discretised for Control and FDIR system purposes.

2.1 Axis System

Following is a definition of the aircraft's axis system in the form of inertial, body and wind reference frames. These reference frames or axes are the basis upon which the standard aircraft notation is defined in order to model the dynamics of the aircraft.

2.1.1 Inertial Axes

The standard north-east-down (NED) axis system adequately approximates an inertial reference frame. The inertial reference frame is required to apply Newton's equations of motion and it assumes a flat non-rotating earth. The origin of the inertial reference frame I is chosen to coincide with a point on the surface of the earth i.e. the lift off point on a runway. Looking at Figure 2.1, the x-axis points north, the y-axis points east and the z-axis points down to complete the orthogonal right handed axis system. The coordinates of the position vector are in the north, east, down inertial axis system.

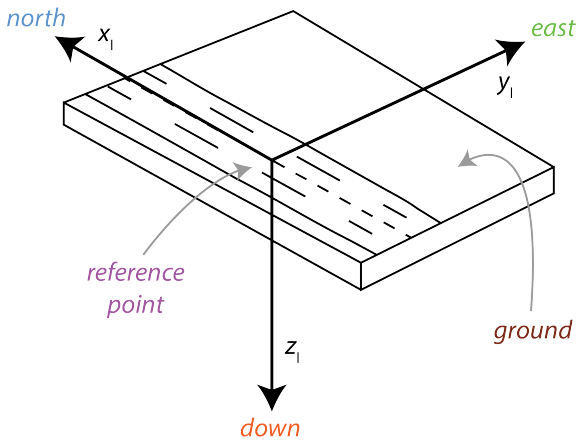


Figure 2.1: *Inertial Axis System, shown with a runway reference point.*

2.1.2 Body Axes

The body reference frame is fixed to the aircraft with its origin chosen to coincide with the aircraft's centre of mass as shown in Figure 2.2. The y-axis lies perpendicular to the plane of symmetry in the direction of the right wing. The x-axis lies in the plane of symmetry in the direction of the rotors. The z-axis points downward and completes the orthogonal right handed system.

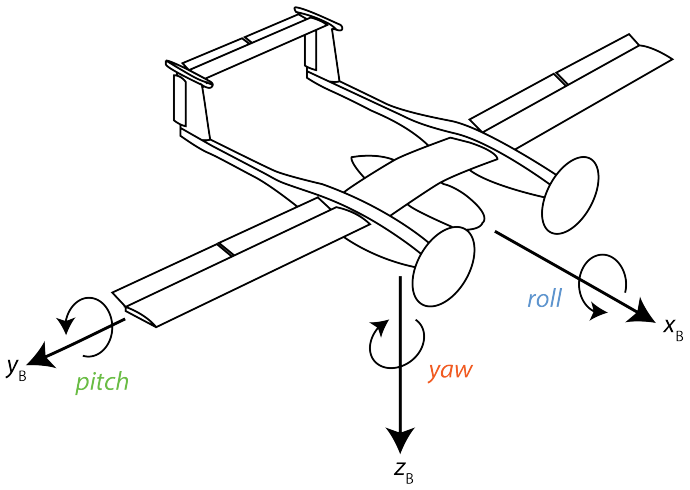


Figure 2.2: *Body Axis System, shown corresponding with the UAV's centre of mass.*

2.1.3 Wind Axes

Like the body axes, the origin of the wind axes is chosen to coincide with the centre of mass and moves with the aircraft. The x-axis points in the direction of the velocity vector V and in looking at β in Figure 2.3, the difference between the body axes and rotated wind axes becomes visible. The y-axis points in the direction of the starboard (right wing). The z-axis lies in the aircraft's plane of symmetry and points in the downward direction to complete the orthogonal right handed axis system.

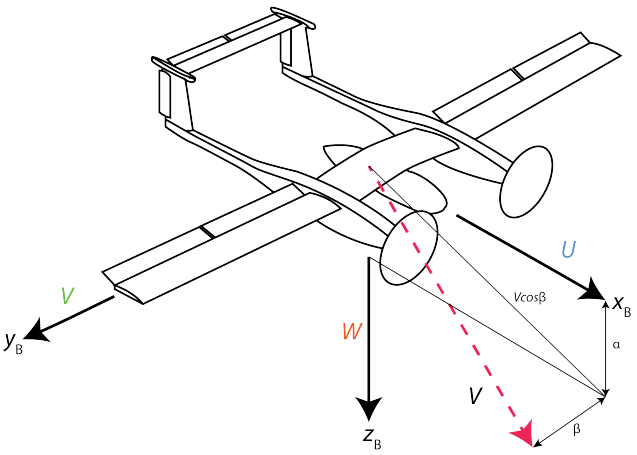


Figure 2.3: *Body Axis System, shown with the velocity vector as a reference for the Wind Axis System.*

2.1.4 Aircraft Notation

Figure 2.4 shows the aircraft's standard notation system, in body axes, used to describe the aircraft's kinematic and kinetic mechanics as follows:

- coordinates of force vector are in body axes: X axial, Y lateral, Z normal force
- coordinates of moment vector are in body axes: L roll, M pitch, N yaw moment
- coordinates of linear velocity vector are in body axes: U axial, V lateral, W normal velocity
- coordinates of angular velocity are in body axes: P roll rate, Q pitch rate, R yaw rate
- the left l and right r control surface deflections, keeping in line with the notation in [38], are two ailerons $\delta_{A_l}\delta_{A_r}$, two flaperons $\delta_{F_l}\delta_{F_r}$, two elevators $\delta_{E_l}\delta_{E_r}$, two rudders $\delta_{R_l}\delta_{R_r}$, and a positive deflection is defined as one that produces a negative moment.

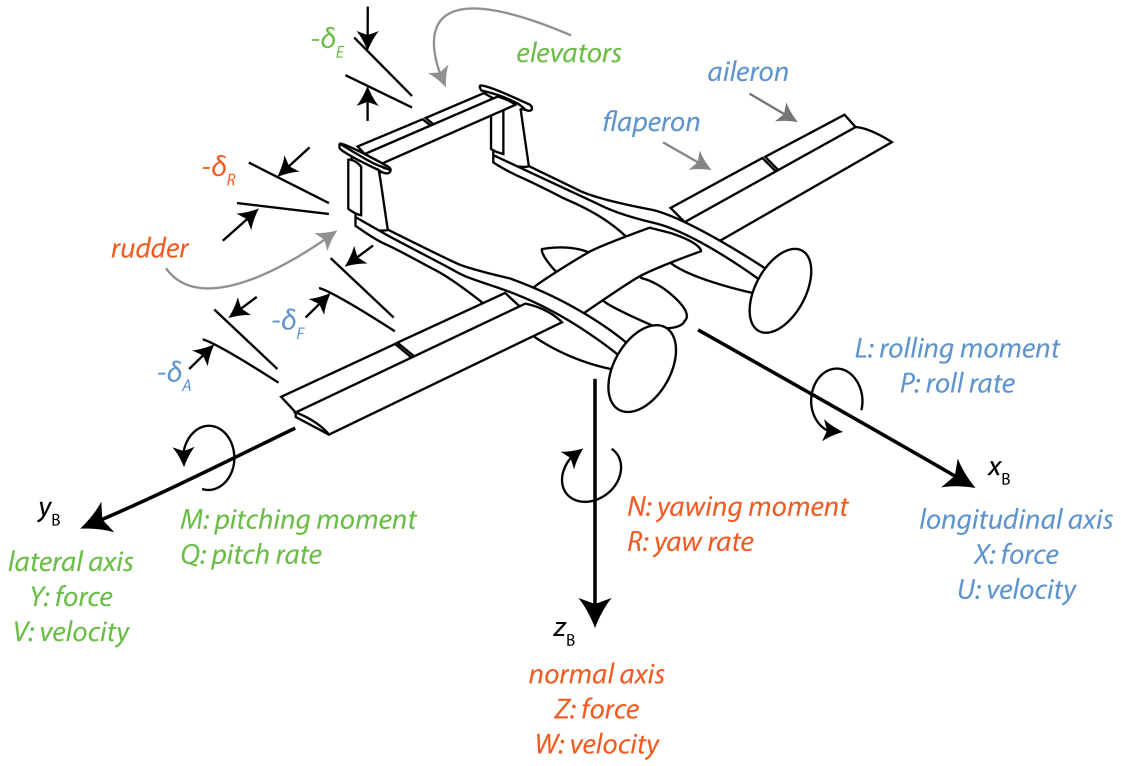
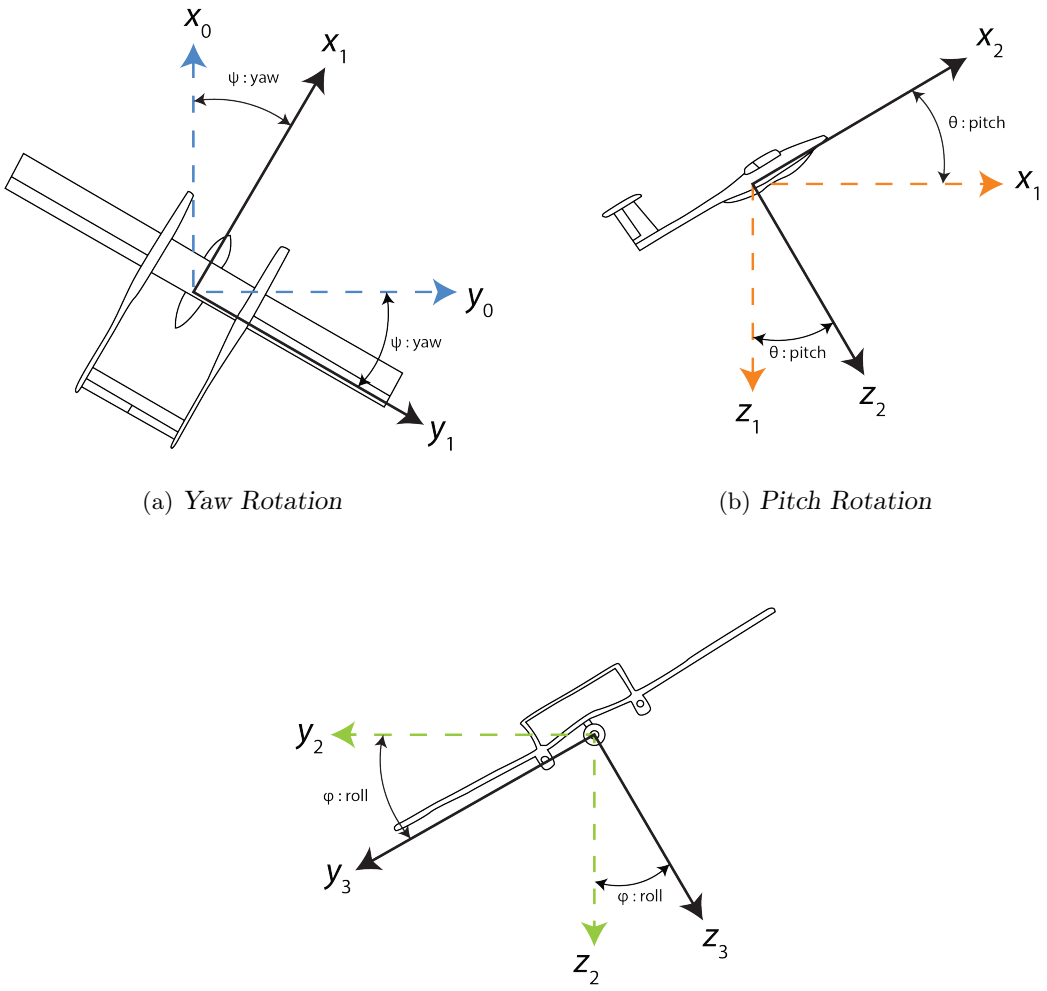


Figure 2.4: *Standard Aircraft Notation, for the Meraka Modular UAV's reference system.*

2.2 Aircraft Kinematics

Kinematics is the field of mechanics that relates various kinds of motion such as linear and angular velocity over time based on the aircraft's position and attitude. The aircraft's position is defined within the NED inertial reference frame and angular position, also known as attitude, is based on Euler angles which are defined in the body reference frame. Rotation matrices are used to translate vectors between the inertial and body reference frames.

2.2.1 Euler Angles



(a) *Yaw Rotation*

(b) *Pitch Rotation*

(c) *Roll Rotation*

Figure 2.5: *Attitude Parameters - showing the Euler 3-2-1 Sequence.*

Euler attitude parameters use three angles in a predefined order of rotation to describe the aircraft's attitude in the body axis system B with respect to the inertial axis system I. The axis system, as shown in Figure 2.2, begins with the inertial and body axis systems aligned. The body axis system then undergoes the following rotation sequence as shown in Figures 2.5a to 2.5c:

- positive yaw rotation through the heading angle ψ
- positive pitch rotation through the pitch angle θ
- positive roll rotation through the roll angle ϕ.

Euler angles are commonly used to parameterise the aircraft's attitude because of simplicity. They also always result in a singularity at a $+/-90°$ pitch angle for the Euler 3-2-1 sequence where changes in yaw and roll constitute the same motion. However in normal flight this singularity hardly comes into play as it is assumed aircraft stays within $|\theta| << \frac{\pi}{2}$. Other means of parameterisation include DCM parameters and quaternions which are more mathematically complex, for more info see [26].

2.2.2 Rotation Matrices

In the derivations that follow, the following applies:

$$c_x = \cos x \tag{2.1}$$
$$s_x = \sin x \tag{2.2}$$
$$t_x = \tan x \tag{2.3}$$

Given the coordinates of a vector $\mathbf{V}$ in the original axis system shown in Figure 2.4, where the inertial and body axes overlap:

$$\boldsymbol{V}_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{2.4}$$

The vector $\mathbf{V}$ undergoes the following rotations as shown in Figures 2.5a to 2.5c:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{2.5}$$
$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \tag{2.6}$$
$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \tag{2.7}$$

Multiplying equations 2.5 to 2.7 relates the vector $\mathbf{V}$ in the original axis system to the rotated axis system and gives:

$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{2.8}$$
$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\theta s_\phi \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\psi s_\theta s_\psi - s_\phi c_\psi & c_\theta c_\phi \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{2.9}$$

Following is the Direction Cosine Matrix (DCM) and its inverse which is its transpose since it can be shown that the DCM is orthogonal [26]. Both transformation matrices can be used to translate the coordinates of a vector in the body axis system B to the inertial axis system I and vice versa.

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\theta s_\phi \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\psi s_\theta s_\psi - s_\phi c_\psi & c_\theta c_\phi \end{bmatrix} \begin{bmatrix} x_I \\ y_I \\ z_I \end{bmatrix} \tag{2.10}$$
$$\begin{bmatrix} x_I \\ y_I \\ z_I \end{bmatrix} = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta c_\phi & c_0 c_\phi \end{bmatrix} \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} \tag{2.11}$$

2.2.3 Position and Attitude Dynamics

The position dynamics describe how the north, east, down states change over time as a function of the aircraft's velocity. The following equation describes the kinematic relation between the aircraft's position and velocity when both vectors are defined in inertial axes.

$$\begin{bmatrix} \dot{N} \\ \dot{E} \\ \dot{D} \end{bmatrix} = \begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix} \tag{2.12}$$

Where V_N, V_E, V_D are the north, east, down velocities. The position dynamics are defined as a function of U, V, W which are in body axes. A transformation matrix is required to relate the vector defined in body axes to the coordinates of the same vector in the inertial axis system, so substituting the DCM yields:

$$\begin{bmatrix} \dot{N} \\ \dot{E} \\ \dot{D} \end{bmatrix} = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_0 c_\phi \end{bmatrix} \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{2.13}$$

It can be shown that the aircraft's attitude parameters can be related to its kinematic states (i.e. P,Q,R) using Euler 3-2-1 dynamics. The following equation relates the roll, pitch and yaw rates with the rate of change in attitude where $|\theta| \neq \frac{\pi}{2}$ for $\theta = n\pi + \frac{\pi}{2}$, $n \in \mathcal{Z}$. During conventional flight, the pitch angle is far from the $+/-90°$ angle so the singularity doesn't become a problem.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \tag{2.14}$$

2.2.4 Velocity Transformations

Referring to Figure 2.3, the velocity vector can be expressed in cartesian and in polar form where $|\boldsymbol{V}|$ is the magnitude of the velocity vector, α is the angle of attack and β is the angle of sideslip:

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} \cos\beta\cos\alpha \\ \cos\beta\sin\alpha \\ \sin\beta \end{bmatrix} \left|\boldsymbol{V}\right| \tag{2.15}$$

$$\left.\begin{aligned} \alpha = & \quad \arctan\frac{W}{U} \\ \beta = & \arctan\frac{V}{U}\cos\alpha \end{aligned}\right\} U \neq 0 \tag{2.16}$$

2.2.5 Quaternions

Quaternions are also used to deal with the singularity that exists when using Euler angles since they don't exhibit this quality. It can be shown that Euler angles and quaternions can be used interchangeably when needed using rotation matrices. A quaternion transformation between two reference frames is described using four parameters instead of three which increases mathematical complexity. Quaternions are not used according to the design standards of systems developed for the Meraka Modular UAV in the ESL and associated FDIR research done thus far.

2.3 Aircraft Kinetics

Kinetics is the branch of mechanics that relates the forces and moments acting on a body to its kinematic state. In order to formulate these relationships, a few assumptions need to be made. Following this, the equations of the forces and moments can be defined using Newton's Laws of Motion and generalised for an aircraft whose variables are defined according to the axis system described.

2.3.1 Assumptions

The following simplifying assumptions were made with respect to the aircraft's dynamics:

- the earth is an inertial reference
- an aircraft is symmetrical about the XZ plane which implies that cross products of inertia I_{xy}, I_{xz} and I_{yz} are zero
- the aircraft is a rigid body i.e. the positions of each mass element remain fixed relative to the body's reference system
- the aircraft's mass remains constant, item the perturbations from equilibrium are small.

2.3.2 Equations of Forces

The translation of a rigid body can be described by momentum and Newton's second law relates the change in momentum to the external forces acting on the body and is written as [31]:

$$\sum \boldsymbol{F} = \left(\frac{d\boldsymbol{p}_B}{dt}\right)_I \tag{2.17}$$

$$= \frac{d}{dt}\left(m\boldsymbol{V}_B\right)_B + \boldsymbol{\Omega}_B \times \boldsymbol{p}_B \tag{2.18}$$

$$= \frac{d}{dt}(m\boldsymbol{V}_B)_B + \boldsymbol{\Omega}_B \times (m\boldsymbol{V}_B) \tag{2.19}$$

Where Σ represents all the forces $\boldsymbol{F}$ acting on aircraft in inertial axes I, $(\,)_I$ and$(\,)_B$ are terms in the inertial and body axes, $\boldsymbol{p}_B$ is the momentum, m is the aircraft's mass, $\boldsymbol{V}_B$ is the linear velocity and $\boldsymbol{\Omega}_B$ is angular velocity of an aircraft in body axes B.

2.3.3 Equations of Moments

The rotation of a rigid body can be described by angular velocity and Newton's second law relates the change in angular velocity to the sum of external moments acting on a body and is written as [31]:

$$\sum \boldsymbol{M} = \frac{d}{dt}\left(\boldsymbol{H}_B\right)_I \tag{2.20}$$

$$= \frac{d}{dt}\left(\boldsymbol{\Omega}\boldsymbol{I}_B\right)_I \tag{2.21}$$

$$= \frac{d}{dt}\left(\boldsymbol{H}_B\right)_B + \boldsymbol{\Omega}_B \times \left(\boldsymbol{H}_B\right) \tag{2.22}$$

$$= \frac{d}{dt}\left(\boldsymbol{\Omega}\boldsymbol{I}_B\right)_B + \boldsymbol{\Omega}_B \times \left(\boldsymbol{\Omega}_B\boldsymbol{I}_B\right) \tag{2.23}$$

Where $\boldsymbol{H}_B$ is angular momentum and the angular rate vector $\boldsymbol{\Omega}_B$ is:

$$\boldsymbol{\Omega}_B = \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \tag{2.24}$$

and $\boldsymbol{I}_B$ is mass moment of inertia tensor, written and simplified as according to the assumptions in Section 2.3.1:

$$\boldsymbol{I}_B = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xz} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \tag{2.25}$$

$$= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.26}$$

2.3.4 Equations of Motion

The kinematic state of an object i.e. position, velocity and acceleration is related to the forces and moments acting on that object through kinetics. The equations describing these forces and moments were derived previously and will be used in their classic form to model this relation.

$$X = m\left(\dot{U} + WQ - VR\right) \tag{2.27}$$

$$Y = m\left(\dot{V} + VR - WP\right) \tag{2.28}$$

$$Z = m\left(\dot{W} + VP - VQ\right) \tag{2.29}$$

$$L = \dot{P}I_{xx} + QR\left(I_{zz} - I_{yy}\right) \tag{2.30}$$

$$M = \dot{Q}I_{yy} + PR\left(I_{xx} - I_{zz}\right) \tag{2.31}$$

$$N = \dot{R}I_{zz} + PQ\left(I_{yy} - I_{xx}\right) \tag{2.32}$$

Where m is the aircraft's mass and I_{xx}, I_{yy} and I_{zz} are the principle moments of inertia about the respective body axes.

2.4 Forces and Moments

With the general 6 DOF EOM model formulated, we now define the forces and moments that act on an aircraft as a function of its current state according to the following models:

- aerodynamic
- engine
- gravitational.

The equations of motion formulated in equations 2.27 to 2.32 are composed of the aerodynamic, thrust and gravitational forces and moments such that:

$$X = X_a + X_t + X_g \tag{2.33}$$

$$Y = Y_a + Y_t + Y_g \tag{2.34}$$

$$Z = Z_a + Z_t + Z_g \tag{2.35}$$

$$L = L_a + L_t + L_g \tag{2.36}$$
$$M = M_a + M_t + M_g \tag{2.37}$$
$$N = N_a + N_t + N_g \tag{2.38}$$

Where subscripts a, t, g denote aerodynamic, thrust and gravitational force and moment components respectively.

2.4.1 Aerodynamic

According to the aerodynamic model derived in [38], the aerodynamic forces and moments can be described using Bernoulli's equation $Pstatic + \frac{1}{2}P|\boldsymbol{V}|^2 = c$ where $Pstatic$ is static pressure and $q = \frac{1}{2}P|\boldsymbol{V}|^2$ is the dynamic pressure where $|\boldsymbol{V}|$ is the translational velocity. The resultant aerodynamic force and moment is proportional to the product of dynamic pressure and airfoil area (S) and written as $F \propto \frac{1}{2}p|\boldsymbol{V}|^2 S$. Expanding the aerodynamic force and moment components gives:

$$X_a = C_X\left(\frac{1}{2}p|\boldsymbol{V}|^2 S\right) \tag{2.39}$$
$$Y_a = C_Y\left(\frac{1}{2}p|\boldsymbol{V}|^2 S\right) \tag{2.40}$$
$$Z_a = C_Z\left(\frac{1}{2}p|\boldsymbol{V}|^2 S\right) \tag{2.41}$$
$$L_a = C_L\left(\frac{1}{2}p|\boldsymbol{V}|^2 S\right) b \tag{2.42}$$
$$M_a = C_M\left(\frac{1}{2}p|\boldsymbol{V}|^2 S\right) \bar{c} \tag{2.43}$$
$$N_a = C_N\left(\frac{1}{2}p|\boldsymbol{V}|^2 S\right) b \tag{2.44}$$

where C are the dimensionless aerodynamic coefficients of proportionality , b wingspan and mean aerodynamic chord $\bar{c}$. The Athena Vortex Lattice (AVL) codes, developed at MIT and which are used in the ESL, produced results that are comparable to wind tunnel data obtained from the Council for Scientific and Industrial Research (CSIR) [38]. The dimensionless aerodynamic coefficients of proportionality are expanded as follows [38]:

$$C_d = C_{d_0} + \frac{C_l^2}{\pi A_e} \tag{2.45}$$
$$C_l = C_{l_0} + C_{l_\alpha}\alpha + \frac{\bar{c}}{2|\boldsymbol{V}|}C_{l_Q}Q + \boldsymbol{C_{l_\delta}\delta} \tag{2.46}$$
$$C_X = -C_d\cos\alpha + C_l\sin\alpha \tag{2.47}$$
$$C_Y = C_{Y_\beta}\beta + \frac{b}{2|\boldsymbol{V}|}C_{Y_P}P + \frac{b}{2|\boldsymbol{V}|}C_{Y_R}R + \boldsymbol{C_{Y_\delta}\delta} \tag{2.48}$$
$$C_Z = -C_d\sin\alpha - C_l\cos\alpha \tag{2.49}$$
$$C_L = C_{L_\beta}\beta + \frac{b}{2|\boldsymbol{V}|}C_{L_P}P + \frac{b}{2|\boldsymbol{V}|}C_{L_R}R + \boldsymbol{C_{L_\delta}\delta} \tag{2.50}$$
$$C_M = C_{M_0} + C_{M_\alpha}\alpha + \frac{2}{2|\boldsymbol{V}|}C_{M_Q}Q + \boldsymbol{C_{M_\delta}\delta} \tag{2.51}$$

$$C_N = C_{N_\beta}\beta + \frac{b}{2\,|\boldsymbol{V}|}C_{N_P}P + \frac{b}{2\,|\boldsymbol{V}|}C_{N_R}R + \boldsymbol{C}_{N_g}\boldsymbol{\delta} \tag{2.52}$$

Where the actuator commands $\boldsymbol{\delta}$ are:

$$\boldsymbol{\delta} = \begin{bmatrix} \delta_{A_r} & \delta_{A_l} & \delta_{E_r} & \delta_{E_l} & \delta_{F_r} & \delta_{F_l} & \delta_{R_r} & \delta_{R_l} \end{bmatrix}^T \tag{2.53}$$

and l and r are the left and right actuators, A and e are the aspect ratio and Oswald efficiency factor control derivatives that relate the actuator commands to the forces and moments $\boldsymbol{C}_{l_\delta}, \boldsymbol{C}_{Y_\delta}, \boldsymbol{C}_{L_\delta}, \boldsymbol{C}_{M_\delta}, \boldsymbol{C}_{N_\delta}$.

2.4.2 Engine

A first order lag model is used to capture the band limited nature of the propulsion sources on the Meraka Modular UAV and is defined in the Advanced Automation 833 Course Notes as:

$$\dot{T} = -\frac{1}{\tau}T + \frac{1}{\tau}T_c \tag{2.54}$$

Where T is the thrust magnitude, T_c is the thrust command and τ is the engine lag time constant. The Meraka Modular UAV has two engines positioned symmetrically to one another, the Onboard Computer (OBC) in conjunction with the Servo Board mixes the control inputs to generate equal levels of thrust in each motor. It is assumed that the thrust vector is close enough to the centre of gravity (CG) such that moments from the rotors are negligible. The following applies when the thrust vector is parallel with the x_B (body) axis:

$$X_t = T \tag{2.55}$$
$$Y_t = Z_t = 0 \tag{2.56}$$
$$L_t = M_t = N_t = 0 \tag{2.57}$$

2.4.3 Gravitational

In a flat earth NED axis system, the gravitational acceleration vector is modelled as providing the force that is equivalent to the aircraft's mass in the down direction, in inertial axes the gravitational force vector becomes:

$$\boldsymbol{F}_I = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{2.58}$$

Transforming this coordinate vector to body axes coordinates using the DCM transformation matrix gives:

$$\begin{bmatrix} X_g \\ Y_g \\ Z_g \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\theta s_\phi \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\psi s_\theta s_\psi - s_\phi c_\psi & c_\theta c_\phi \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{2.59}$$

$$= \begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} mg \tag{2.60}$$

Lastly, because the centre of gravity coincides with the centre of mass in a uniform gravitational field, the gravitation force doesn't produce any moments so:

$$L_g = M_g = M_g = 0 \tag{2.61}$$

2.5 Aircraft Model

An aircraft is well modelled as a six degree of freedom rigid body using the kinematic and kinetic equations defined. The aircraft model developed thus far is nonlinear and must be linearised and discretised for control and FDIR system design purposes. Following is a description of the three mathematical models of the Meraka Modular UAV in their various forms namely: nonlinear, linear and discretised state space representation.

2.5.1 Nonlinear Differential Equations

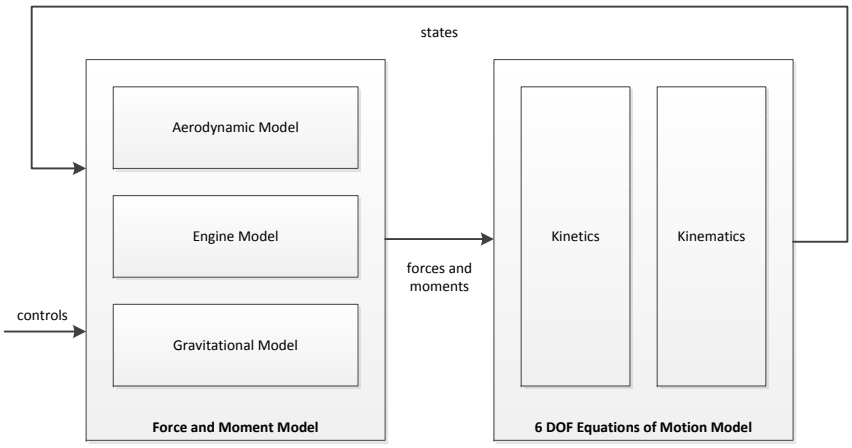


Figure 2.6: *6 DOF EOM Model, showing the relation between the different sub-models.*

Together the six degree of freedom equations of motion (6 DOF EOM) can be related in order to construct the aircraft model shown in Figure 2.6 based on the individually derived models. The kinematic equations relate the linear and angular velocity rates of change in attitude and position. The kinetic equations relate the forces and moments to the rate of change of the linear and angular velocities. Lastly, the forces and moments model the aircraft's specific aerodynamic, engine and gravitational dynamics. Following is the final set of the Meraka Modular UAV's nonlinear differential equations in their classic form:

$$\dot{U} = \frac{X}{m} + VR - WQ \tag{2.62}$$

$$\dot{V} = \frac{Y}{m} - UR + WP \tag{2.63}$$

$$\dot{W} = \frac{Z}{m} + UQ - VP \tag{2.64}$$

$$\dot{P} = \frac{1}{I_{xx}} \left(L - QR \left(I_{zz} - I_{yy} \right) \right) \tag{2.65}$$

$$\dot{Q} = \frac{1}{I_{yy}} \left(M - PR \left(I_{xx} - I_{zz} \right) \right) \tag{2.66}$$

$$\dot{R} = \frac{1}{I_{zz}} \left(N - PQ \left(I_{yy} - I_{xx} \right) \right) \tag{2.67}$$

$$\dot{\phi} = P + Q\left(s_\phi t_\theta\right) + R\left(s_\phi t_\theta\right) \tag{2.68}$$
$$\dot{\theta} = Q\left(c_\theta\right) - R\left(s_\phi\right) \tag{2.69}$$
$$\dot{\psi} = Q\left(s_\phi \sec\theta\right) + R\left(c_\phi sec\theta\right) \tag{2.70}$$
$$\dot{N} = U\left(c_\psi\, c_\theta\right) + V\left(c_\psi s_\theta s_\phi - s_\psi c_\phi\right) + W\left(c_\psi s_\theta c_\theta + s_\psi s_\phi\right) \tag{2.71}$$
$$\dot{E} = U\left(s_\psi\, c_\theta\right) + V\left(c_\psi s_\theta s_\phi + s_\psi c_\phi\right) + W\left(c_\psi s_\theta c_\theta - c_\psi s_\phi\right) \tag{2.72}$$
$$\dot{D} = -U\left(s_\theta\right) + V\left(c_\theta s_\phi\right) + W\left(c_\theta c_\phi\right) \tag{2.73}$$

2.5.2 Linearised Mathematical Model

The process of linearising (removing nonlinear terms from the equation) the aircraft's dynamics about trim results in the $\dot{\psi}, \dot{N}, \dot{E}, \dot{D}$ differential equations being omitted. The dynamics governing the ψ, N, E and D states don't affect the aircraft's fundamental dynamics which are subsequently given as:

$$\dot{U} = \frac{X}{m} + VR - WQ \tag{2.74}$$
$$\dot{V} = \frac{Y}{m} - UR + WP \tag{2.75}$$
$$\dot{W} = \frac{Z}{m} + UQ - VP \tag{2.76}$$
$$\dot{P} = \frac{1}{I_{xx}}\left(L - QR\left(I_{zz} - I_{yy}\right)\right) \tag{2.77}$$
$$\dot{Q} = \frac{1}{I_{yy}}\left(M - PR\left(I_{xx} - I_{zz}\right)\right) \tag{2.78}$$
$$\dot{R} = \frac{1}{I_{zz}}\left(N - PQ\left(I_{yy} - I_{xx}\right)\right) \tag{2.79}$$
$$\dot{\phi} = P + Q\left(\sin\phi\tan\theta\right) + R\left(\sin\phi\tan\theta\right) \tag{2.80}$$
$$\dot{\theta} = Q\left(\cos\theta\right) - R\left(\sin\phi\right) \tag{2.81}$$

These dynamics can be written concisely in the form $\dot{\boldsymbol{x}} = \mathbf{f}(\boldsymbol{x}, \boldsymbol{u})$ where:

$$\boldsymbol{x} = \begin{bmatrix} U & V & W & P & Q & R & \phi & \theta \end{bmatrix}^T \tag{2.82}$$
$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{\delta} & T \end{bmatrix}^T \tag{2.83}$$

where $\boldsymbol{x}$ and $\boldsymbol{u}$ are the aircraft's states and control inputs respectively. In describing the aircraft as a system with smooth non-linearities that is continuously differentiable $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$ and for linearisation the states and inputs are expanded as the sum of a trim value and a perturbation about trim which yields:

$$\boldsymbol{x} = \boldsymbol{x_T} + \boldsymbol{\Delta x} \tag{2.84}$$
$$\boldsymbol{u} = \boldsymbol{u_T} + \boldsymbol{\Delta u} \tag{2.85}$$

where perturbations from trim are written as lowercase letters and greek symbols:

$$\boldsymbol{\Delta x} = \begin{bmatrix} u & v & w & p & q & r & \phi & \theta \end{bmatrix}^T \tag{2.86}$$
$$\boldsymbol{\Delta u} = \begin{bmatrix} \boldsymbol{\Delta\delta} & \Delta T \end{bmatrix}^T \tag{2.87}$$

Expanding $\dot{\boldsymbol{x}} = \mathbf{f}(\boldsymbol{x}, \boldsymbol{u})$ about trim condition yields a Taylor Series of the form:

$$\dot{\boldsymbol{x}} + \boldsymbol{\Delta}\dot{\boldsymbol{x}} = \boldsymbol{f}\left(\boldsymbol{x_T} + \boldsymbol{\Delta x}, \boldsymbol{u_T} + \boldsymbol{\Delta u}\right) \tag{2.88}$$

$$= \boldsymbol{f}\left(\boldsymbol{x_T}, \boldsymbol{u_T}\right) + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_T \boldsymbol{\Delta x} + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_T \boldsymbol{\Delta u} + \textit{higher order terms} \tag{2.89}$$

where T means an evaluation about the trim working point where:

$$\boldsymbol{A}_T = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_T \tag{2.90}$$

$$\boldsymbol{B}_T = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_T \tag{2.91}$$

Assuming that the perturbations about trim are small, the dynamics can be approximated as $\dot{\boldsymbol{x}} = \boldsymbol{f}\left(\boldsymbol{x}, \boldsymbol{u}\right) \approx \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$ by ignoring higher order terms that are negligible. In noting that at trim condition $\dot{\boldsymbol{x}_T} = \boldsymbol{f}\left(\boldsymbol{x_T}, \boldsymbol{u_T}\right) = \boldsymbol{0}$ the dynamics of the system can be described by $\boldsymbol{\Delta}\dot{\boldsymbol{x}} = \boldsymbol{A}_T\boldsymbol{\Delta x} + \boldsymbol{B}_T\boldsymbol{\Delta u}$ which is a continuous state space approximation of a set of nonlinear equations about trim. The linearisation problem then becomes that of determining the vector partial derivatives that constitute $\boldsymbol{A}_T$ and $\boldsymbol{B}_T$.

The linearised mathematical model used in this research has been revised to represent the aircraft's fundamental dynamics so $\boldsymbol{\Delta x}$ becomes:

$$\boldsymbol{\Delta x} = \begin{bmatrix} u & v & w & p & q & r & \phi & \theta \end{bmatrix}^T \tag{2.92}$$

and since w and v are not used in aerodynamic equations the solution becomes:

$$\alpha = \arctan\left(\frac{w}{u}\right) = \arctan\left(\frac{w}{|\boldsymbol{V}|_T \cos(\beta) cos(\alpha)}\right) \approx \frac{w}{|\boldsymbol{V}|_T} \tag{2.93}$$

$$\beta = \arctan\left(\frac{v}{u}\cos(\alpha)\right) = \arctan\left(\frac{w}{|\boldsymbol{V}|_T \cos(\beta) cos(\alpha)}\cos(\alpha)\right) \approx \frac{v}{|\boldsymbol{V}|_T} \tag{2.94}$$

and since α, β are measurable using sensors, the state vector becomes $\boldsymbol{\Delta x} = \begin{bmatrix} \overline{v} & \alpha & q & \theta & \beta & p & r & \phi \end{bmatrix}^T$ and the $\boldsymbol{A}_T$ and $\boldsymbol{B}_T$ matrices are derived and expanded as shown in Appendix C based on the parameters and coefficients in Appendix B.

2.5.3 Discrete State Space Representation

In following the modelling methodology used in [38], discretising the continuous state space representation derived in the previous section (for state perturbations) gives:

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{G}\boldsymbol{u}_k \tag{2.95}$$

$$\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{u}_k \tag{2.96}$$

where $\boldsymbol{u}_k$ is the input vector, $\boldsymbol{y}_k$ is the measurement vector, $\boldsymbol{H}$ is the measurement matrix and k denotes the time step. Assuming a zero-order hold process, the discrete space matrices become [21]:

$$\boldsymbol{F} = e^{\boldsymbol{A}T} = \boldsymbol{I} + \boldsymbol{A}T + \frac{\boldsymbol{A}^2T^2}{2!} + \frac{\boldsymbol{A}^3T^3}{3!} + ... \approx \boldsymbol{I} + \boldsymbol{A}T \tag{2.97}$$

$$\boldsymbol{G} = \left(\int_0^T e^{\boldsymbol{A}\eta} d\eta\right)\boldsymbol{B} = \left(\boldsymbol{I} + \frac{\boldsymbol{A}T}{2!} + \frac{\boldsymbol{A}^2T^2}{3!} + ...\right)T\boldsymbol{B} \approx T\boldsymbol{B}$$

$$\boldsymbol{H} = \boldsymbol{C} \tag{2.98}$$

where $\boldsymbol{C}$ is the continuous state space measurement matrix and T is the time step length. In the approximation for the state space representation above, the higher order terms were ignored as they become increasingly smaller as the power of T increases.

2.5.4 Flight Envelope

In order to generate the linearised state space matrices representing the decoupled lateral and longitudinal dynamics of the system, the aircraft's behaviour must be evaluated at a working point.

The dimensionless aerodynamic coefficients were derived using AVL and a simulation of an aircraft flown at trim condition with a positive angle of attack [38]. In following the convention used in the research done thus far, an airspeed of 22m/s was selected and the lateral and longitudinal state space matrices were calculated as:

$$\begin{bmatrix} \alpha_T \\ \delta_{E_T} \end{bmatrix} = \begin{bmatrix} C_{L_\alpha} & C_{L_{\delta_E}} \\ C_{m_\alpha} & C_{m_{\delta_E}} \end{bmatrix}^{-1} \begin{bmatrix} \frac{mg}{q_T S} - C_{L_0} \\ -C_{m_0} \end{bmatrix} \tag{2.99}$$

$$T_T = q_T S C_{D_T} \cos\alpha_T - q_T S C_{L_T} \sin\alpha_T + mg \sin\alpha_T \tag{2.100}$$

where:

$$C_{D_T} = C_{D_0} + \frac{C_{L_T}^2}{\pi A e} \tag{2.101}$$

results in the following linearised state space matrices:

$$\boldsymbol{A}_{long} = \begin{bmatrix} -0.1312 & -2.3293 & 0 & -9.8086 \\ -0.0263 & -5.0906 & 0.9509 & 0.0062 \\ 0 & -21.3720 & -6.8251 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.102}$$

$$\boldsymbol{B}_{long} = \begin{bmatrix} 0 & 0 & 0.0385 & 0.0385 \\ -0.1612 & -0.1616 & 0 & 0 \\ -12.3041 & -12.3041 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.103}$$

$$\boldsymbol{A}_{lat} = \begin{bmatrix} -0.3567 & 0.0033 & -0.7765 & 0.3633 \\ -11.1230 & -7.1656 & 2.2419 & 0 \\ 19.2281 & -0.8859 & -1.1888 & 0 \\ 0 & 1 & -0.0169 & 0 \end{bmatrix} \tag{2.104}$$

$$\boldsymbol{B}_{lat} = \begin{bmatrix} -3.3197e-04 & -3.3197e-04 & 3.4598e-04 & -3.4598e-04 & \dots \\ -25.4541 & -25.4541 & -17.9489 & 17.9489 & \dots \\ 1.0778 & 1.0778 & -0.6575 & 0.6575 & \dots \\ 0 & 0 & 0 & 0 & \dots \end{bmatrix} \begin{bmatrix} \dots & 0.0037 & 0.0037 \\ \dots & -6.7148 & -6.7148 \\ \dots & -6.7148 & -6.7148 \\ \dots & 0 & 0 \end{bmatrix} \tag{2.105}$$

The longitudinal $\boldsymbol{x}_{long}$ and lateral $\boldsymbol{x}_{lat}$ states are, as stated in Appendix C:

$$\boldsymbol{x}_{long} = \begin{bmatrix} \overline{v} & \alpha & q & \theta \end{bmatrix}^T \tag{2.106}$$

$$\boldsymbol{x}_{lat} = \begin{bmatrix} \beta & p & r & \phi \end{bmatrix}^T \tag{2.107}$$

where $\overline{v}$ is the airspeed, α is the angle of attack, q is the pitch rate, θ is the pitch angle, β is the angle of side slip, p is the roll rate, r is the yaw rate and ϕ is the roll angle.

In the next chapter we use the $\boldsymbol{A}_{long}$, $\boldsymbol{B}_{long}$, $\boldsymbol{A}_{lat}$, $\boldsymbol{B}_{lat}$ state space matrices to construct transfer functions between the inputs and outputs of the decoupled aircraft model. This makes it possible to for instance, to determine the input-output relationship between the elevator control surface input and the measurable state output, the pitch rate q. Using the pitch rate, we will be able to for instance determine what the elevator control surface deflections are based on the behaviour of the aircraft during flight. Thereby making it possible to perform actuator FDI by comparing the commanded control surface inputs and the estimated actuator deflections. It is important to note that having decoupled the longitudinal and lateral dynamics of the aircraft, we are now able to perform calculations using a relatively smaller state space matrix which will add to the increased computational efficiency of the implemented algorithm.

2.6 Overview

In this Chapter the aircraft model for the Meraka Modular UAV is derived based on an axis system in the form of inertial, body and wind reference frames. These frames or axes are the basis upon which the standard notation is defined in order to model the dynamics of the aircraft.

The aircraft's kinematic relationships are derived in order to relate the aircraft's linear and angular velocity over time based on the aircraft's position and attitude. The aircraft's position is defined in terms of the inertial reference frame and angular position is defined in terms of Euler angles defined in the body reference frame. Rotation matrices used to translate vectors between the inertial and body reference frames are also derived.

The aircraft's kinematic relationships are used to relate the forces and moments acting on a body to its kinematic state. A few simplification assumptions are made in order to define the forces and moments acting on the aircraft using Newton's Laws of Motion. This is done in order to formulate the 6 DOF EOM model of the aircraft.

Models describing aerodynamic, engine and gravitational forces and moments specific to aircraft are also derived. The nonlinear aircraft model derived is linearised and discretised for control and FDIR purposes. The state space matrices are calculated for the aircraft's anticipated flight envelope at trim condition so that they can be used in the design and development of the fault diagnostic system's input observers in the next chapter.

Chapter 3

System Modelling

The Hybrid Diagnosis Engine (HyDE) and the Approximate Input Reconstruction Algorithm (AIRA) are used together to give the Meraka Modular UAV fault diagnostic capabilities. A Failure Mode and Effects Analysis (FMEA) is performed in order to characterise the subsystem components as a means of guiding the fault modelling and identification process that follows. We use HyDE along with AIRA in order to perform actuator FDI through input reconstruction since the control surfaces don't have position sensors.

3.1 Hybrid Diagnosis Engine

HyDE was developed by researchers at the University of California Santa Cruz in close collaboration with colleagues at the National Aeronautics and Space Administration's (NASA) Ames Research Center and several other organisations [11]. In this research, HyDE is used to give the Meraka Modular UAV the ability to diagnose faults and detect changes in its subsystem components. It does this through a system of declarative models and reasoning paradigms that it uses to simulate and predict system behaviour and reason about the state of the UAV. We begin investigating HyDE by examining the theory behind how HyDE works and by highlighting the processes encapsulated within HyDE using an example.

3.1.1 Overview

Model based diagnosis is based on general purpose models that describe the behaviour and or internal structure of a system. A hybrid system is a physical system whose behaviour is comprised of discrete and continuous changes. A stochastic hybrid system is a system with inherent elements of uncertainty due in part to a combination of: sensor noise from reported observations; incomplete system knowledge which results in approximate models and prior probabilities on the occurrence of faults; and uncertainty about the conditions that give rise to changes in discrete and continuous behaviour.

HyDE is a model based diagnosis or reasoning engine that uses candidate generation and consistency checking to diagnose discrete faults in stochastic hybrid systems [11]. It uses continuous and discrete (hybrid) models built by users and sensor data (observations) from the system to deduce the state of the system both forwards and backwards in time. The core functionality of HyDE lies in its ability to diagnose multiple discrete faults and handle hybrid system behaviour using qualitative and quantitative model based methods, and stochastic reasoning and it works on Windows, Solaris, Linux and VxWorks platforms. The key issues that make the diagnosis task difficult in addition to the complexity induced by the stochastic and hybrid nature of the system are: a) limited observability - the number of sensors that allow the system to be observed are limited and as a consequence faults are often not directly observable and must be isolated by reasoning about the system using information from available sensors; and b) time-delayed symptoms - the effects of

faults typically don't manifest immediately, consequently processes used to isolate faults have to reason both backwards and forwards in time to take into account the time delay. The time delay may be a result of integrating effects due to the presence of components with state and observable properties might only be affected by faults in certain discrete mode configurations and lastly the stochastic nature of hybrid systems make it impractical to use single points of comparison to make decisions. We now briefly examine the processes encapsulated within HyDE, for a detailed overview please see [11].

HyDE Models

HyDE models are the essential building blocks that enable the diagnosis engine to reason about the system. The most basic model is the component model which is used to generate the system model. The system model is then used to generate a candidate model which HyDE uses to reason about the system when diagnoses are requested. HyDE has support for multiple modelling paradigms and uses these to define how systems are described [11]. HyDE has support for the modelling paradigm used in the GME Tool which enables the model designer to graphically create models that can be exported to HyDE in XML format or simulated using the HyDE interpreter.

Component Model

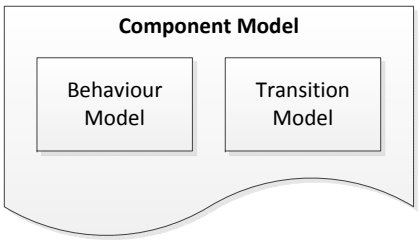


Figure 3.1: *HyDE Component Model*

The component model, shown in Figure 3.1, is comprised of a Behaviour Model and a Transition Model. Together, these inner component models describe the transition behaviour and behaviour evolution of the component.

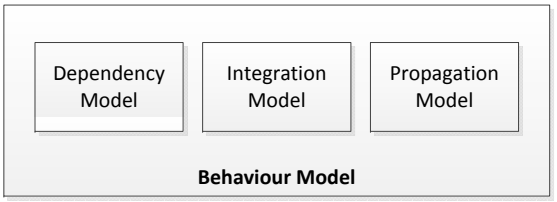


Figure 3.2: *HyDE Behaviour Model*

The Behaviour Model (M), shown in Figure 3.2, describes the behaviour evolution of the component and is specific to each modelling paradigm. It is comprised of a dependency model, integration model and a propagation model. Depending on the modelling paradigm, the same model could be used alone or separately to describe all three of these inner models.

The Dependency Model (DM) describes the dependencies between variables, models and components and is used to optimise candidate generation during the reasoning process. The dependencies described are between the variables V, the constraints (relations) R_t defined in the Integration Model and constraints R_g, R_{li} defined in the Propagation Model.

The Integration Model (IM) specifies how each variable's value is propagated across time steps during candidate testing. The Integration Model for a variable v_i whose first derivative has been defined would be described as $IM(v_i) = R_t[v_i(t_k), \delta v_i(t_k), v_i(t_{k-1}), \delta v_i(t_{k-1}), ...]$ where δv_i represents the first derivative of variable v_i and t_k represents the current time step.

The Propagation Model (PM) enables HyDE to estimate the values of unknown variables from known variables for the global model (system level) and local model (component level) within time steps during candidate testing. The Propagation Model would be defined as $PM_g = R_g(V)$ and as $PM(li) = R_{li}(V)$ for a global model g and local model l respectively where i is the ith location.

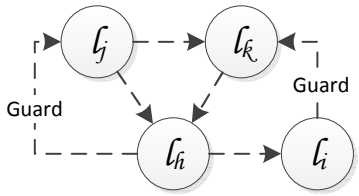


Figure 3.3: *HyDE Transition Model*

The Transition Model (T), shown in Figure 3.3, describes the transition behaviour for a system and is common to all modelling paradigms supported by HyDE. It is comprised of locations l_i which are the modes of operation (shown as circles) and transitions t_i which are transitions between modes of operation (shown as dotted lines) where i represents the ith location or transition and $t_i = l_j \rightarrow l_k$ represents the transition from location l_j to l_k. Transition guards (shown with the word *Guard* in Figure 3.3) describe the conditions in which the transition between two locations occurs and represents an observable change in a component's location i.e. from the *on* to off location of a lamp based on the state of a switch. Since the conditions in which a component transitions into a fault are not observed directly, HyDE must reason about the occurrence of faults from their symptom effects on the system. These are described as locations that have transitions without guards and constraints are used when the symptom effects are known i.e. low voltage and not used when the fault and its symptom effects are unknown or the behaviour is not expected.

System Model

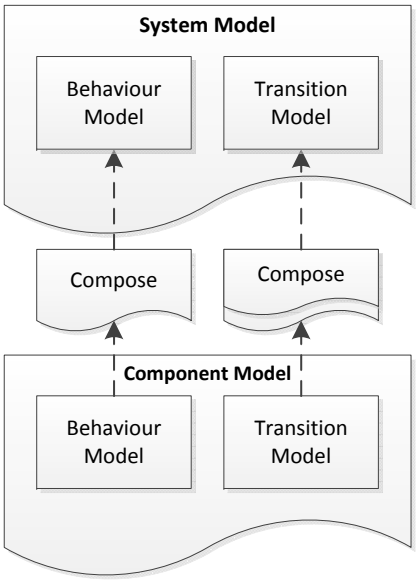


Figure 3.4: *HyDE System Model, adapted from [11].*

The System Model, shown in Figure 3.4, is composed from (in parallel for the transition model) the Component Model and describes the transition behaviour and behaviour evolution of the system. The System Model is used to create the candidate generated during the reasoning process. In the same way as the component model, the System Model is also specific to each modelling paradigm and can also be transformed between modelling paradigms when needed. It's properties include the variables V of the system, the domains D for the variables in the system and the guards G for transitions between locations.

The composition process for the Behaviour Model results in the composition of the system variables V_s defined as $V_s = V_g \cup V_1 \cup V_2 ... \cup V_n$ where V_g represents the global variables which don't belong to any component and $V_1, ...V_n$ represent the component variables where $1, 2, ..., n$ represents each component. The transition guards G don't change for each transition between the relevant component locations. The locations from each component in the system, defined as $l_s = \{l_{1i}\ l_{2j}\ ...\ l_{nk}\}$ where $i, j, ..., k$ are the locations in the respective components, are used to compose the system behaviour model $M_s = M_g \cup M_{connection} \cup M_{1i} \cup M_{2j}\ ... \cup M_{nk}$ where $\cup$ is the union operation specific to the modelling paradigm used for the component and system model. M_g is the behaviour model for the global model and $M_{connection}$ is the model that describes how subsystem level component models interact. The Transition Model resulting from the composition process consists of locations $SL = L_1 \times L_2 \times L_n$ to give a system location $sl_1 = (l_{1i},\ ...\ , l_{nk})$ that is a synchronous composition of each component $1, 2\ ...\ n$ in the system and $i, ..., k$ represents the respective component's location. Synchronous implies that the system location is specific to the state of the system at that point in time. In addition, the transitions in the system are composed by defining transitions between the locations in each component in $SL = L_1 \times L_2 \times L_n$ to give a synchronous system transition $Trans_{\rightarrow t}$ for system location $sl_1 = (l_{1i},\ ...\ , l_{nk})$ where t is an instance in time.

Candidate Model

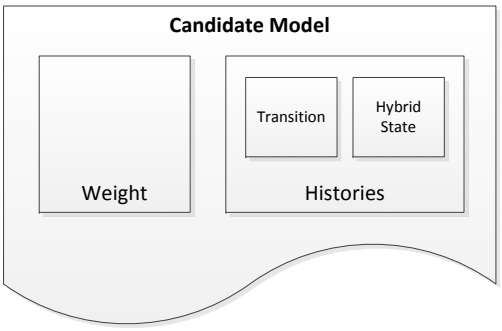


Figure 3.5: *HyDE Candidate Model*

The Candidate Model, shown in Figure 3.5, represents the trajectory of the system's behaviour evolution and consists of a weight w, hybrid state HS and transition $Trans_{\rightarrow t}$ histories. A candidate model is inferred from the system models, the initial locations and the values of each variable and the reported observations. The candidate's weight w is used to rank the candidate based on the degree of fit between the predicted value and reported observation, and the prior probabilities of the system transitions assumed to be taken.

The hybrid state history is a snapshot of the system's state and is defined as $HS = (SL, VV)$ where SL are the locations in the system and VV are the variables and their corresponding values. The hybrid state history defines the current locations of each component $SL = \{(comp_i \rightarrow l_i) \mid 1 \leq i \leq m\}$ and the current values for each of the variables $VV = \{(value_i \rightarrow v_i) \mid 1 \leq i \leq n\}$. The hybrid and transition histories for a candidate after HyDE has been run for a couple time steps $\{1 \leq i \leq o\}$ would be in the form $\{(Trans_{\rightarrow ti}, hs_{ti-}, hs_{ti+}) \mid 1 \leq i \leq o\}$ where ti is the ith time step, and $-/+$ represent the beginning and end of the current time step respectively.

HyDE Reasoning

HyDE Reasoning, shown in Figure 3.6, is the maintenance of a *set* C of weighted candidates (w_i, c_i) generated using a candidate model. Upon initialisation of HyDE, the candidate set is initialised with a candidate with the system's known initial hybrid state or a randomly sampled candidate. The initial candidate is of the form $\{(Trans_{\rightarrow t}, hs_{t0-}, hs_{t?})\}$ where ? represents an unknown hybrid state. The reasoning process for each requested diagnosis consists of three main steps, which will be discussed further, and begins with 1) Candidate Set Management which is followed by 2) Candidate Testing and lastly ends with 3) Candidate Generation. Candidate Set Management maintains the candidate set by adding and removing candidates. Candidate Testing operates on the highest ranked candidate and reports inconsistencies resulting from estimating the candidate's hybrid state and updating the candidate's weight. Candidate Generation creates candidate generators from reported inconsistencies and supplies the next untested potential candidate to the Candidate Set Management.

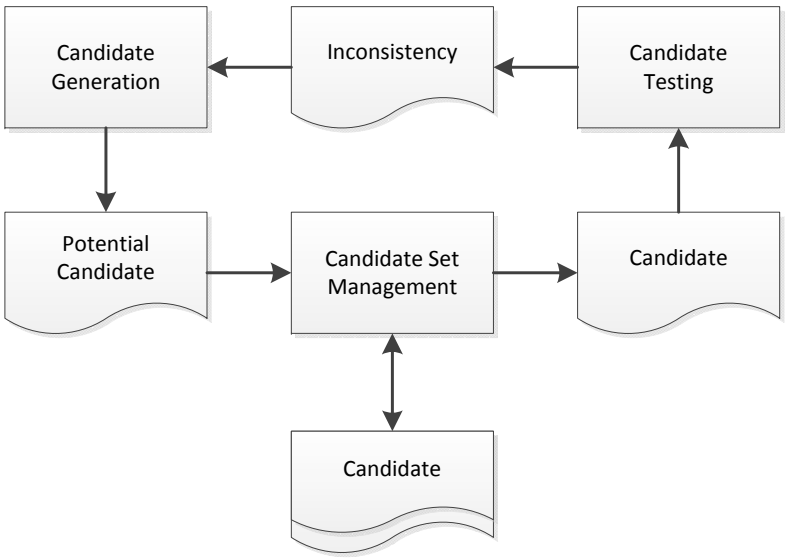


Figure 3.6: *HyDE Reasoning Process, adapted from [11].*

Candidate Set Management

Candidate Set Management, shown in Figure 3.7, consists of the following steps:

1. *Update candidate weights* by Candidate Testing each candidate in the candidate set describing the alternate trajectories of the system's behaviour evolution.
2. *Prune candidates* after updating candidate weights by removing all candidates with weights that are below a certain threshold.
3. *Add candidates* by requesting a new potential candidate from the candidates generated during Candidate Generation after Candidate Testing. If the number of candidates in the set is above a specified minimum resample the weights or proceed with refilling the candidate set.
4. *Re-sample weights* by normalising the candidate weights and resampling candidates based on the distribution of weights and adding the sampled candidates to the normalised candidate set.

A diagnosis is made available after the steps within Candidate Set Management are complete and this in turn results in an updated or new candidate set based on the reported inputs, predicted values and reported observations.

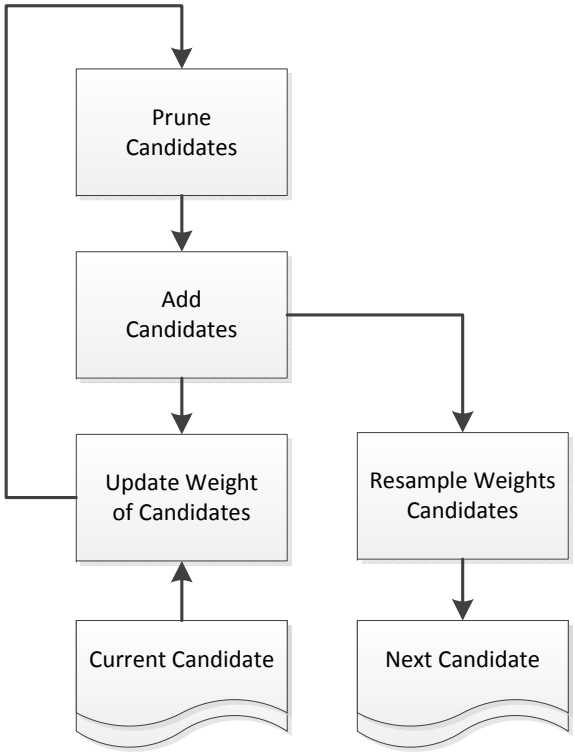


Figure 3.7: *HyDE Candidate Set Management, adapted from [11].*

Candidate Testing

The validity of the candidate that most accurately describes the current state and trajectory of the system must be tested each time a diagnosis is requested. This is done by simulating the system and comparing predicted values with reported observations. The Candidate Testing process, shown in Figure 3.8, consists of the following steps:

1. *Find enabled transitions* by collecting transitions from the system location associated with hs_{ti-1} from the given candidate and estimate the beginning hybrid state's system location $SL(hs_{ti-})$ using $Trans_{\rightarrow ti}$ created for each of the enabled locations from the previous time step ti.

2. *Propagate across time steps* by estimating the new variable values $VV(hs_{ti-})$ using sensed inputs and the Integration Model IM_{ti} that is applied to $VV(hs_{ti-1})$.

3. *Propagate within time step* by loading the Propagation Model (PM) for $SL(hs_{ti-})$ which needs to be composed should it not have been cached.

4. *Estimate the end hybrid state* hs_{ti+} by loading the Propagation Model (PM) and simulating the system or by using a propagation algorithm, and prepare to report the inconsistency I_{ti} to Candidate Generation System if the execution fails.

5. *Compare* the predicted values with the observed values using a comparison function CF which is used to create residuals that indicate the degree of fit between the predicted and observed values when taking into account the defined noise model for each variable.

6. *Update weight of candidate* by multiplying the candidate's weight with $r_{overall}$, the overall degree of fit determined during the previous step.

7. *Report inconsistency* I_{ti} to the Candidate Generation System if $r_{overall}$ is less than the specified minimum threshold.

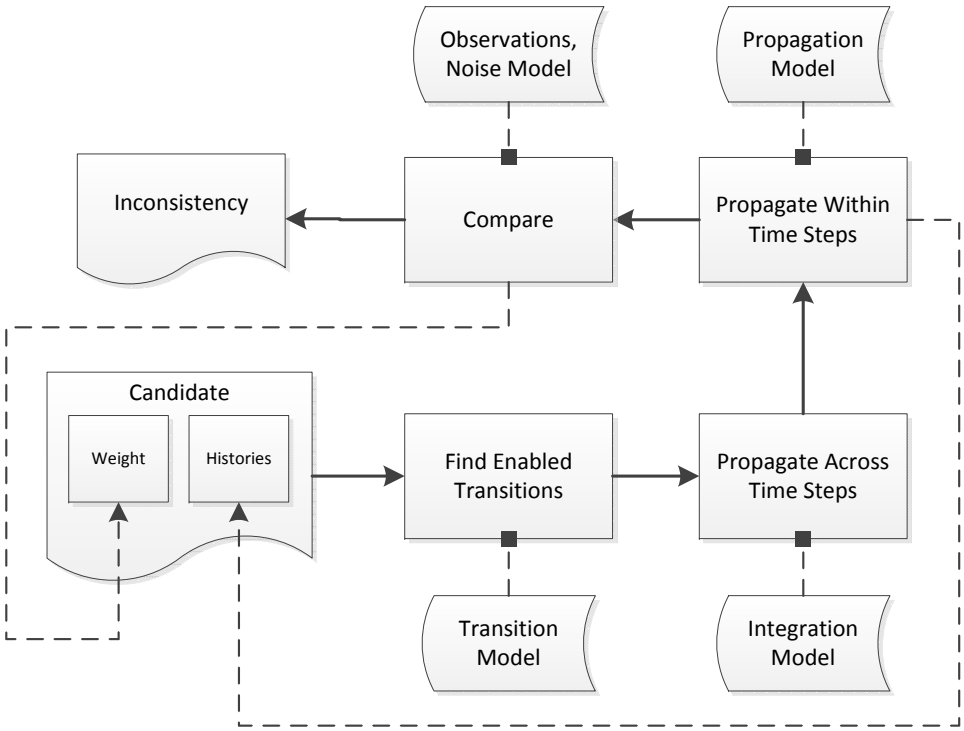


Figure 3.8: *HyDE Candidate Testing, adapted from [11].*

Candidate Generation

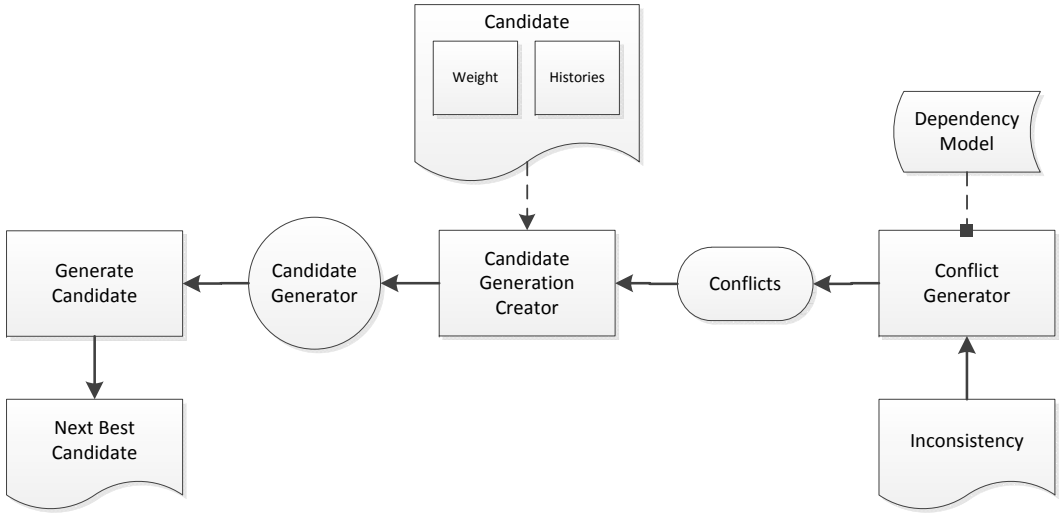


Figure 3.9: *HyDE Candidate Generation adapted from [11]*

Candidate Generation, shown in Figure 3.9, includes a set of Candidate Generators created from candidates or reported inconsistencies. When a candidate is requested, each generator reports its best candidate. The next best candidate is selected amongst the best of the best candidates reported before reporting it to the Candidate Set Management.

Figure 3.10: *HyDE Conflict Generator*

A conflict is a list of timed transitions that give rise to an inconsistency when taken by the system. The Conflict Generator, shown in Figure 3.10, generates a set of unguarded transitions $Trans_{unguarded}$ that resolve conflicts using sibling transitions for the entire set of conflicts. The conflicts are generated directly using HyDE's proprietary conflict directed search algorithm. A conflict is created for each inconsistent variable by retrieving the Dependency Model DM_{ti} corresponding to $SL(hs_{ti})$. For each component, the matching transitions from $Trans_{\rightarrow ti}$ with time stamp ti are added to the conflict set. To begin the backward transversal, the same process is repeated for the previous n user specified steps for each of the Dependency Models DM_{ti-n} for all the variables that depend on the inconsistency I_{ti}.

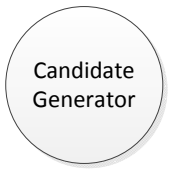


Figure 3.11: *HyDE Candidate Generator*

The Candidate Generator, shown Figure 3.11, is based on either a candidate or a conflict. The generator based on a candidate always reports the same candidate and is created as a consequence of Candidate Testing. The conflict based generator is created with initial hybrid state hs_{ti-n}, initial weight equivalent to the weight of the candidate that generated the inconsistency, and the conflict set generated from the inconsistency. The resulting candidate from either generator type is created with an initial hybrid state and preloaded with transitions that go back in time.

One of the most important consequences of knowing how HyDE works at a fundamental level is being able to 'think' in terms of how HyDE models and reasons about systems. Knowledge about the fundamentals behind how AIRA works made it possible to optimise the real time version of the algorithm. In the same way, optimising the model development and reasoning process in HyDE requires knowledge about how HyDE works.

The most critical aspect of the modelling process is knowing what to model and deciding what level of abstraction is needed in order to create declarative HyDE models that will result in meaningful diagnoses that are performed within a reasonable amount of time. The Reliability Analysis described in Section 3.3 fulfilled the purpose of guiding the fault modelling and identification process. It also made it possible to define the transition probabilities for guarded and unguarded transitions (faults) that influence the ranking of locations and subsequently the weights of the candidates in the candidate set. By adjusting the parameters contained in the Harness file, a considerable amount of influence can be exerted on the reasoning process to such an extent that the time HyDE takes to perform diagnoses can be reduced significantly - if we know for instance that HyDE is spending a lot of CPU time pruning candidates from the candidate set because the candidate weights are below an unreasonable threshold.

HyDE has its limitations and to overcome these, the model designer needs to be able to augment HyDE's functionality by incorporating fault decisions or additional processing logic (i.e. complex matrix calculations or algorithms) performed outside HyDE using representative components internally, as is the case in this research project with the Smart Adapters and in [4]. Doing this properly will require an understanding of how to get data into and out of HyDE and how often by configuring state variables and their associated noise models using the Harness parameters or by adjusting the rate at which observations are reported. This was the case in the thermostat example where the integration model HyDE was using was erroneously causing guarded transitions to occur because of an unreasonably large sample time for the reported observations. This functionality (Harness) is used extensively in this research project to configure the 10% '*Percentage*' and 0.007 *rad Constant* sensor noise threshold harness parameters used to account for varying

observations in addition to the backtracking functionality configured using the 'n user specified steps' that determine the amount of telemetry HyDE needs to store in order to perform actuator FDI. Another important aspect that needs to be understood is inter-component communication within HyDE using local and global variables modified by constraints within the relevant locations and creating state machines using transition guards that allow components to exhibit autonomous behaviour. As you will see in Section 3.5, these are some of the issues that had to be addressed in order to accurately model the functional relationships described by the Characterised Subsystem Model in Section 3.3.5. Understanding how HyDE works is required in order to develop HyDE models that maximise the advantages of having an expandable FDIR architecture. In the following subsections we investigate HyDE using a concrete example of a simplified heating system.

3.1.2 Model

HyDE models describe the expected system behaviour during nominal and fault conditions. They are similar in nature to the simulation models created in MATLAB which makes it easy to translate them into HyDE by modifying them to represent hybrid automata (finite state automata with equations in each state) [29]. They are constructed in a hierarchical and modular fashion by building component and subsystem models. Within the system model the components are connected together using shared variables.

The component models represent behaviour using locations or modes of operation (which include faulty modes) and transitions which represent the conditions under which the system changes locations. Faults are modelled as transitions that occur under unknown conditions and therefore must be inferred through reasoning about the system. The behaviour of the components is described using a set of variables or parameters and relations governing the interactions between these variables.

The hybrid nature of the system is captured using a combination of the behavioural and transitional models. The stochastic nature of the system is described using probabilities associated with transitions as well as noise on sensed variables or output observations. This mathematical and declarative model of the system is the key input to HyDE and determines the quality of the diagnostic analyses that are performed.

HyDE accepts models in three file formats: the non-XML ASCII file with the *. ham file extension and the nested or flat XML model file with the *. hxm file extension. The advantage of the XML file format is that it comes with automatic validation. The Generic Modelling Environment (GME) is a GUI based tool, built at Vanderbilt University, that allows application developers or users to graphically create models and check these for various validity conditions before they are exported as XML or ASCII files. These files can then be accepted as input when accessing the HyDE Application Programming Interface (API) programmatically using C++.

Model Example

As an example, we now investigative how a furnace connected to a thermostat is modelled in HyDE, please see the HyDE Reference Manual for more information. The furnace has two nominal locations off and on and a fault location $unknown$. The transitions between the locations depend on the temperature and the thermostat's setting. The indoor temperature partly depends on the furnace's activity. The furnace may turn on arbitrarily once the thermostat is set and not in direct response to the commanded setting. The unguarded transitions that respond to commands are declared as:

```
1 <Transition from="off" to="unknown" probability="0.001"/>
2 <Transition from="on" to="unknown" probability="0.001"/>
```

and the guarded transitions that respond to the indoor temperature are:

```
<Transition from="off" to="on" guard="indoorTemperature = setting - 5"/>
<Transition from="on" to="off" guard="indoorTemperature = setting + 5"/>
```

The first transition causes the furnace to switch from off to on when the indoor temperature is 5° below the thermostat's setting. The second transition causes the furnace to switch from on to off when the indoor temperature is 5° above the thermostat's setting. From the guarded transitions above we can see that the indoor temperature is a state variable and is being used to characterise the state of the system. HyDE automatically updates the value of a state variable if its derivative can be described using an ordinary differential equation. For all real state variables, HyDE implicitly creates a variable for the first derivative of the state variable with respect to time. The derivative's variable name has the same name as the state variable with D appended e.g. $indoorTemperature$ and $indoorTemperatureD$. If the furnace is off, the indoor temperature will tend to reach equilibrium with the outdoor temperature. We therefore need to define a variable $outdoorTemperature$ as follows:

```
<Variable name="outdoorTemperature" type="Real"/>
```

Practical experience suggests that when the furnace is off, the rate of change of the indoor temperature is proportional to the difference between the indoor temperature and the outdoor temperature and this relationship can be characterised using a constant of proportionality:

```
<Constant name="diffusionRate" type="Real" value="0.1"/>
```

We can then use this constant to define the differential equation that describes the change in indoor temperature when the furnace is off:

```
<Constraint expression= "indoorTemperatureD = (outdoorTemperature - indoorTemperature)
    * diffusionRate"/>
```

An indoor temperature that exceeds the outdoor temperature results in a negative derivative because the diffusion rate is positive which causes the the indoor temperature to decrease and visa versa. When the indoor temperature and the outdoor temperature are equal no change results. The outdoor temperature is assumed to be independent of the system so it's not a state variable and can be represented as a constant or as an input variable. The heating rate is used to quantify the effect that the furnace has on the indoor temperature when it is on and is defined as:

```
<Constant name="heatingRate" type="Real" value="0.1"/>
```

which makes the differential equation describing the change in indoor temperature:

```
<Constraint expression="indoorTemperatureD = heatingRate"/>
```

The complete model for this basic heating system can therefore be defined using the following XML code in flat format:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE System SYSTEM "hxm_flat.dtd">
<System name="C:\HyDE\Project3\Project3\aircon.hxm" type="Flat">
<Constant name="diffusionRate" type="Real" value="0.1"/>
<Constant name="furnace.heatingRate" type="Real" value="0.1"/>
<Variable name="indoorTemperature" type="Real" state="true" domain=""/>
<Variable name="outdoorTemperature" type="Real" state="false" domain=""/>
<Variable name="thermostat.setting" type="Real" state="false" domain=""/>
<Component name="furnace"/>
<Component name="thermostat"/>
<Location name="furnace.off" component="furnace"/>
<Location name="furnace.on" component="furnace"/>
<Location name="furnace.unknown" component="furnace"/>
<Transition from="furnace.on" to="furnace.off" probability="1"
    guard="(indoorTemperature)=((thermostat.setting)+(5))" reset=""/>
<Transition from="furnace.on" to="furnace.unknown" probability="0.001" guard=""
    reset=""/>
<Transition from="furnace.off" to="furnace.unknown" probability="0.001" guard=""
    reset=""/>
<Transition from="furnace.off" to="furnace.on" probability="1"
    guard="(indoorTemperature)=((thermostat.setting)-(5))" reset=""/>
<Constraint
    expression="indoorTemperatureD=((outdoorTemperature-indoorTemperature)*diffusionRate)"
    location="furnace.off" isassignmentconstraint="false"/>
<Constraint expression="(indoorTemperatureD)=(furnace.heatingRate)"
    location="furnace.on" isassignmentconstraint="false"/>
</System>
```

An equivalent model of the complete model of the basic heating system derived above can also be generated graphically using the GME Tool. This accelerates the model development process of more complex HyDE models making exporting a model to XML format a relatively trivial process. Figure 3.12 shows a graphical version of the simplified heating system model using the GME Tool.

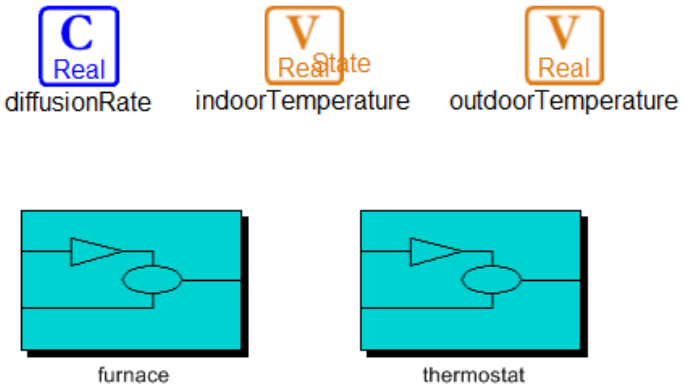


Figure 3.12: *System model of the simplified heating system.*

The GME Tool allows a model designer to drag and drop components representing variables, constants, components, locations and transitions. It is evident from Figure 3.12 that the heating system has: an input variable $indoorTemperature$ and an output variable $outdoorTemperature$; a constant $diffusionRate$ with the two subsystem components being the furnace and thermostat. Figure 3.13 shows the component model of the furnace with the various locations on, off and $unknown$ and the transitions between them. The constraints for each location are represented using constraint objects that allow the equations to be entered.

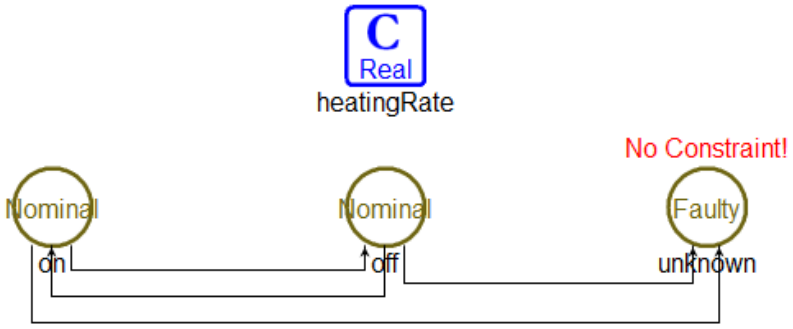


Figure 3.13: *Component model of the furnace.*

3.1.3 Harness

The harness is the second input that is used in HyDE and defines two classes of information: firstly the initial conditions of the hybrid system before execution begins and secondly the configuration parameters that determine how HyDE performs diagnoses. The initial conditions consist of:

- the values of the state variables
- the locations or modes of operation of the components
- the input and output configuration for variables reported as observations.

It is assumed that input variables are observable and the harness is stored as a non-XML ASCII file with the *. ham file extension or as an XML file with the *. hxm file extension who's format is specified using the Document Type Definition (DTD) file.

Harness Example

Having discussed the harness we now go back to the heating example. HyDE uses the differential equations to update the value of the state variable during each time step using the update algorithm $V(t_{i+1}) = V(t_i) + (t_{i+1} - t_i) * V^{'}(t_i)$ meaning that the state variable V is incremented by $\Delta V = V^{'}(t) * \Delta t$ when HyDE progresses from time t_i to time t_{i+1}. The smaller the ΔT the better the approximation becomes. When looking at the following harness file written in flat XML format, one can easily see the two classes of information being the initial conditions of the hybrid system as well as the configuration parameters. Another evident property is the sensor noise setting for observations of the indoorTemperature and the initial 20° indoor temperature.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Harness SYSTEM "hxh.dtd">
<Harness>
<Component name="furnace" initiallocation="furnace.on"/>
<Variable name="thermostat.setting" input="true"/>
<Variable name="indoorTemperature" output="true" sensornoise="0.020000" state="true"
    initialvalue="20"/>
<Variable name="outdoorTemperature" input="true"/>
<Parameter name="CommandsToBackTrackAcross" value="UNBOUNDED"/>
<Parameter name="FilterType" value="constraints"/>
<Parameter name="GenerateUniqueCandidates" value="false"/>
<Parameter name="HistoryTime" value="5"/>
<Parameter name="Integrator" value="Euler"/>
<Parameter name="SystemLocationChangesToBackTrackAcross" value="UNBOUNDED"/>
<Parameter name="MaximumCandidateCount" value="1"/>
```

```
<Parameter name="MaximumCandidateSize" value="UNBOUNDED"/>
<Parameter name="MaximumCandidatesToTry" value="UNBOUNDED"/>
<Parameter name="MinimumCandidateProbability" value="0"/>
<Parameter name="NoiseModel" value="Percentage"/>
<Parameter name="NumberOfConsistencyChecks" value="1"/>
<Parameter name="PreferNewerCandidates" value="false"/>
<Parameter name="maximumcandidategenerationtime" value="UNBOUNDED"/>
<Parameter name="UseDependencyGraphs" value="true"/>
</Harness>
```

For the example heating system, we defined a two percent sensor noise margin for observations of the indoor temperature and the Euler method as the integration method of choice. These configuration parameters can be adjusted in GME before exporting the harness file and testing the diagnostic engine using the observations reported in the scenario file.

3.1.4 Scenario

The last input to HyDE is the set of observations that are derived from the hybrid system observed. These observations can be stored as non-XML ASCII files with the *. has file extension and used during model development in GME or as function calls for real time execution using the HyDE API. The scenario is a time series of values for the commands, input and output variables that are defined in the harness file. HyDE reasons about all the time steps that have requests when observations are supplied using function calls. When using scenario files, HyDE reasons about all the time steps with observations and the *step* keyword can also be used to force HyDE to reason about the time steps that have no observations.

Scenario Example

When HyDE is run with the model that has been defined, it begins by establishing the outdoor temperature and the indoor temperature, set the thermostat's temperature and start the furnace in its initial location as shown in the following scenario file.

```
1  OBSERVE thermostat.setting 25.00
1  OBSERVE outdoorTemperature 15.00

50    OBSERVE thermostat.setting 25.00
50    OBSERVE outdoorTemperature 15.00

100   OBSERVE thermostat.setting 25.00
100   OBSERVE outdoorTemperature 15.00

150   OBSERVE thermostat.setting 25.00
150   OBSERVE outdoorTemperature 15.00

155   OBSERVE thermostat.setting 25.00
155   OBSERVE outdoorTemperature 15.00

160   OBSERVE thermostat.setting 25.00
160   OBSERVE outdoorTemperature 15.00

165   OBSERVE thermostat.setting 25.00
165   OBSERVE outdoorTemperature 15.00

170   OBSERVE thermostat.setting 25.00
170   OBSERVE outdoorTemperature 15.00

```

```
250   OBSERVE thermostat.setting 25.00
250   OBSERVE outdoorTemperature 15.00
250   OBSERVE indoorTemperature 26.10
```

In looking at the scenario file, one can see that the observations for the indoor temperature are not always reported to the diagnostic engine. In this case, it will continue simulating the system's behaviour until it receives information that suggests that the simulation of the system no longer represents the actual state of the system.

At this point, the simulated behaviour will then be used to revise the set of candidates that represent the state of the system. Figure 3.14 shows an example of a scenario file that has been loaded into the GME Tool, at this point one can then simulate the system and probe HyDE for diagnoses to verify that the model functions as intended.

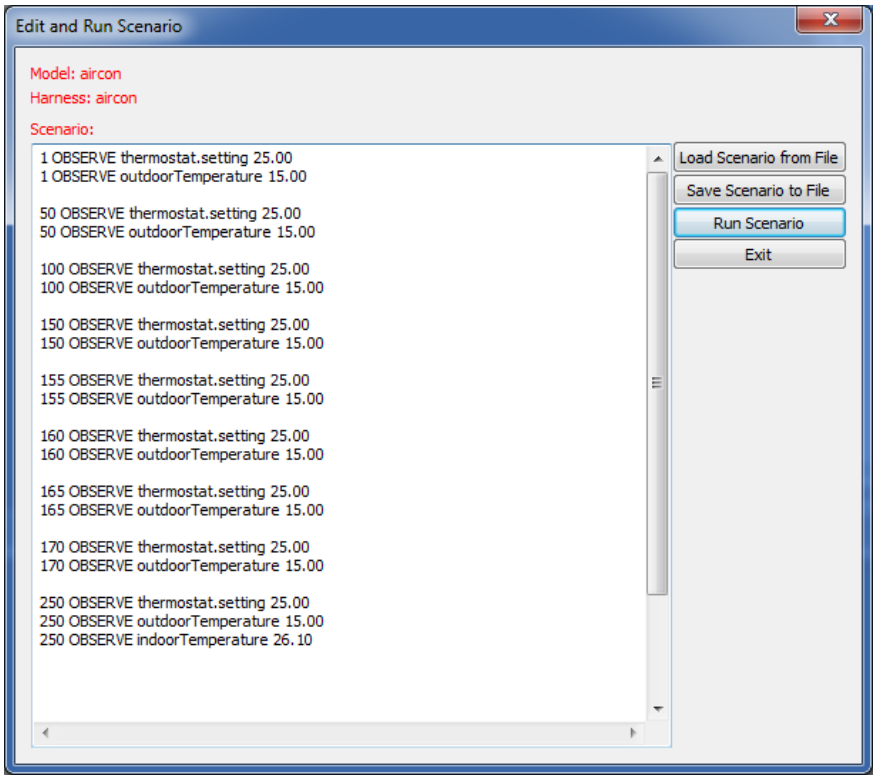


Figure 3.14: *Scenario file loaded into the GME Tool.*

3.1.5 Reasoning

HyDE uses the model for both simulation and candidate generation. The diagnosis or reasoning process revolves around the maintenance of a set of candidates, that might include multiple hypothesised faults, that are consistent with observations reported. The candidate lists the transitions (between fault modes) that are assumed to have been taken by the system at each time step. Reporting observations prompts HyDE to begin the reasoning process to determine if the candidates in the candidate set are still consistent with the reported observations. Consistent candidates remain in the candidate set and the information about inconsistent candidates is used to generate successor candidates while inconsistent candidates are discarded.

Figure 3.15 shows the main steps in the reasoning process and these include: 1) simulation, 2) comparison, and 3) candidate generation. The model corresponding to the candidate being tested

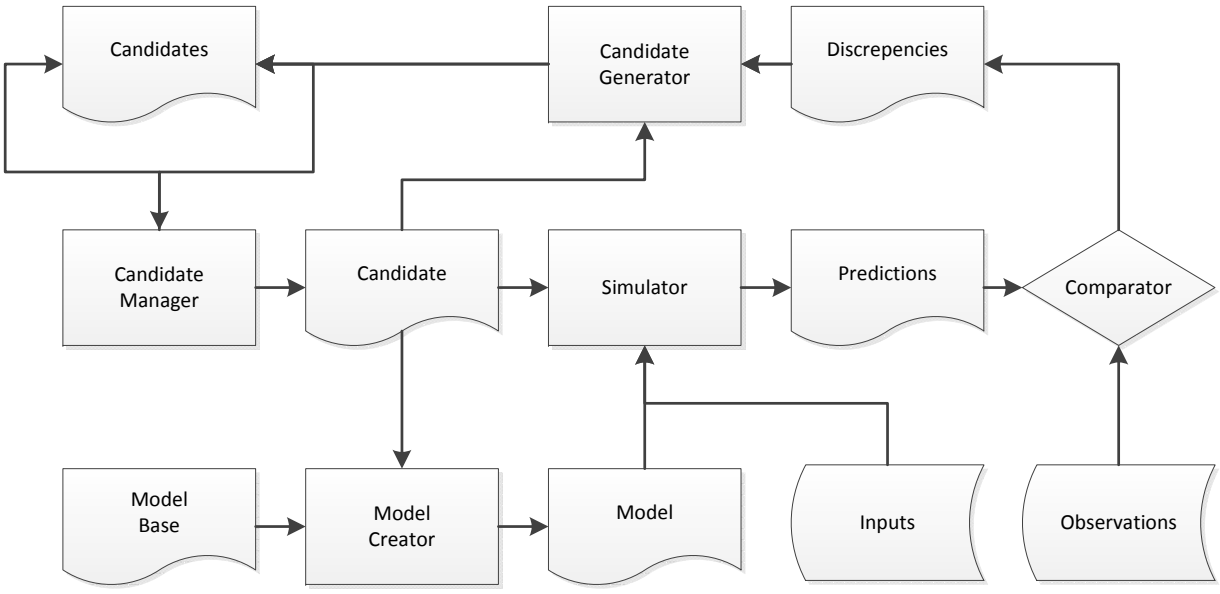


Figure 3.15: *HyDE Reasoning Process, showing major functional steps, adapted from [11].*

is used to simulate the behaviour of the system. The goal of the simulation is to predict the expected values of the variables using the component models that correspond to the sensed observations.

The comparison step uses predictions from the simulation step and compares them with actual sensor readings to identify inconsistencies and if a discrepancy is discovered the model is then used to determine the cause. The uncertainties present (i.e. noise and sensor values) along with the hybrid nature of the system is taken into account during the reasoning process. The main component of the HyDE reasoning process revolves around the generation and maintenance of new candidates when the existing candidates become inconsistent. A conflict directed search process, driven by inconsistencies generated during simulation, is used to identify the cause(s) of discrepancies.

Each time the application calls the $HyDE :: step()$ function, HyDE will update the value of the state variable $indoorTemperature$ when the transition guard is activated during an autonomous transition out of the initial location. The transition will fire and the furnace will autonomously change location thereby changing the active constraints and the differential equation governing the change in indoor temperature. When we report indoor temperature observations, we prompt HyDE to perform a diagnosis which results in a consistency check that ascertains the state of the system.

Reasoning Example

Following is an example of a C++ application that uses the HyDE API to simulate the behaviour of the simplified heating system and report diagnoses that represent the state of the system. The declarative model of the system is input into the application using the XML files generated by the GME Tool. This application can then be executed in real time in an embedded system that automates the behaviour of the heating system and one that is connected to an interface that can report the state of the system to a person.

In this example an instance of HyDE is created and the model files *aircon.hxm* and *aircon.hxh* representing the model and harness file respectively are loaded. This is followed by a series of observations that are reported to HyDE and simulation steps that are requested of HyDE. Here the time between the steps varies, for systems modelled in general, this would correlate with the sample time of the system being simulated and observed.

```
// include HyDE API header file
#include <API\HyDE.hpp>
// include io stream library
#include <iostream>

// open the HyDE API
using HyDEAPI::HyDE;
// open the std API
using namespace std;

// create the main method @author Sandile Memela
int main(int argc, char **argv) {

   try { // attempt to execute codes

      HyDE hyde;                        // instantiate HyDE
      hyde.loadModel("aircon.hxm");     // load a simple HyDE model
      hyde.loadHarness("aircon.hxh"); // load a simple configuration file

      // report observations instruct HyDE to start reasoning
      hyde.reportObservation(1, "thermostat.setting", "25.00");
      hyde.reportObservation(1, "outdoorTemperature", "15.00");
      hyde.step(1);
      hyde.reportObservation(50, "thermostat.setting", "25.00");
      hyde.reportObservation(50, "outdoorTemperature", "15.00");
      hyde.step(50);
      hyde.reportObservation(100, "thermostat.setting", "25.00");
      hyde.reportObservation(100, "outdoorTemperature", "15.00");
      hyde.step(100);
      hyde.reportObservation(150, "thermostat.setting", "25.00");
      hyde.reportObservation(150, "outdoorTemperature", "15.00");
      hyde.step(150);
      hyde.reportObservation(155, "thermostat.setting", "25.00");
      hyde.reportObservation(155, "outdoorTemperature", "15.00");
      hyde.step(155);
      hyde.reportObservation(160, "thermostat.setting", "25.00");
      hyde.reportObservation(160, "outdoorTemperature", "15.00");
      hyde.step(160);
      hyde.reportObservation(165, "thermostat.setting", "25.00");
      hyde.reportObservation(165, "outdoorTemperature", "15.00");
      hyde.step(165);
      hyde.reportObservation(170, "thermostat.setting", "25.00");
      hyde.reportObservation(170, "outdoorTemperature", "15.00");
      hyde.reportObservation(170, "indoorTemperature", "18.10");
      hyde.step(170);
      hyde.reportObservation(250, "thermostat.setting", "25.00");
      hyde.reportObservation(250, "outdoorTemperature", "15.00");
      hyde.reportObservation(250, "indoorTemperature", "20.10");
      hyde.step(250);

   } catch (const exception& e) { // catch any errors that occur
      cerr << e.what() << endl;
   }

   system("pause"); // wait for system exit

}
```

Figure 3.16 shows the diagnostic results requested during each time step in the application. Here each candidate contains a list of the components in the system and the state that each component is in at that time. At time 170, one can see that the reported observations are still valid because a 2% sensor noise tolerance was defined for the observations of the indoor temperature.

When the reported observation is inconsistent with the predicted observation, HyDE will generate a list of candidates and order them based on the transition probabilities between likely locations that contain constraints that explain the reported observations. HyDE reasons backwards and forwards in time when attempting to generate candidates for models with stored observations or telemetry.

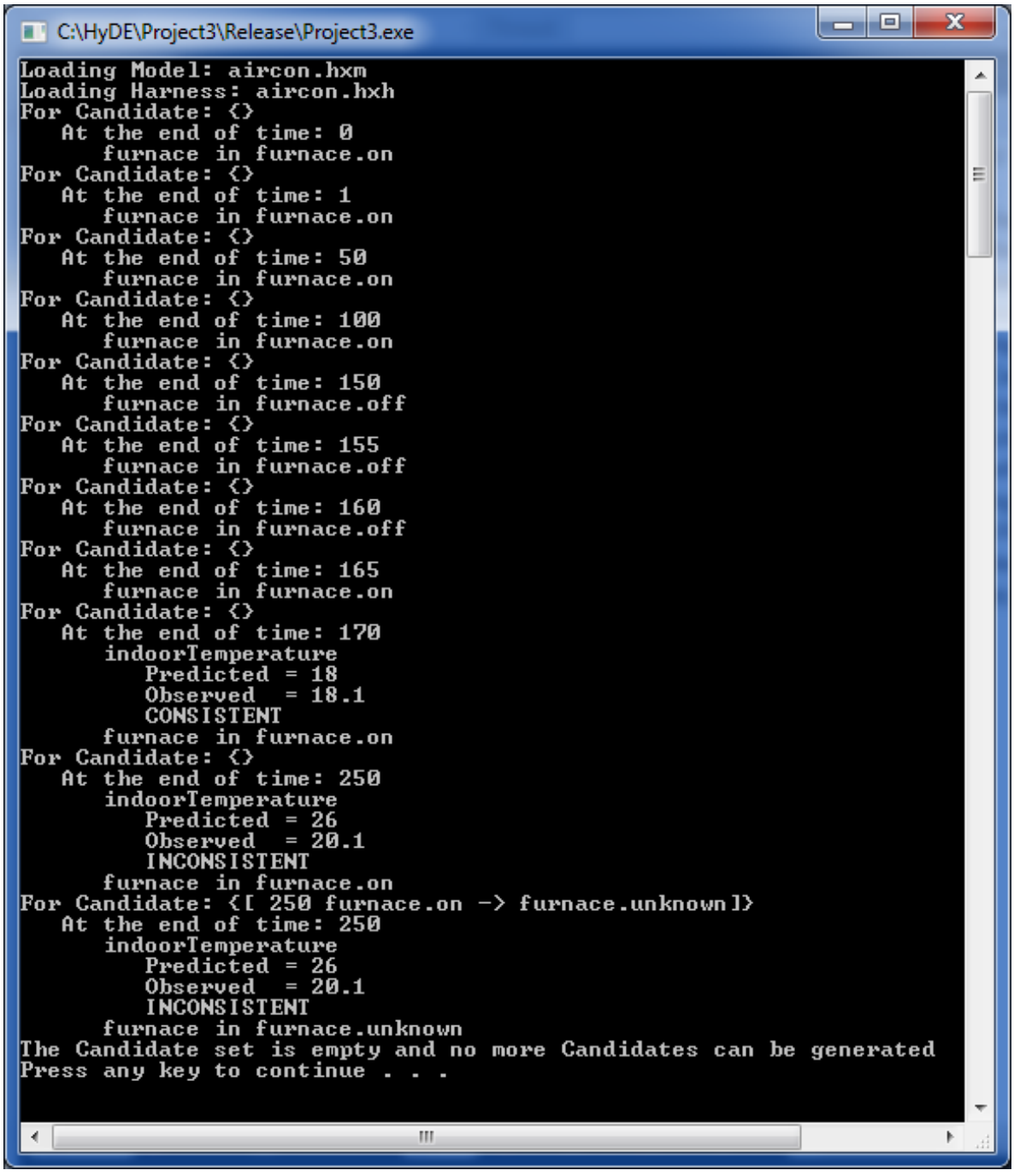


Figure 3.16: *Output of the c++ application accessing the HyDE API.*

3.2 Approximate Input Reconstruction

One of the key issues identified in the diagnosis task is limited observability. The number of sensors that allow the system to be observed are limited on the Meraka Modular UAV. The Approximate Input Reconstruction Algorithm or Method (AIRA), a Least Squares Algorithm (shown in Figure 1.5) based on the Moore-Penrose Generalised Inverse, is used to reconstruct unknown inputs which can be used for fault detection. In this research, AIRA is used to estimate inputs that are compared with commanded values. We use HyDE along with AIRA in order to perform actuator FDI through input reconstruction since the control surfaces don't have position sensors.

3.2.1 Overview

Discrepancies between known and unknown inputs to the system are used to perform fault detection. Unknown inputs to the system can be reconstructed using input observers based on System Inversion Techniques as a part of the fault detection process. Once available these unknown inputs, reconstructed using sensor measurements, can be compared with a model of the system along with the known inputs in order to determine discrepancies that suggest the presence of faults.

System Inversion

AIRA is a Least Squares Algorithm with a similar foundation to the parity space approach, for a detailed overview please see [25]. The applicability of AIRA to practical systems is heavily dependent on a sound understanding of a critical issue inherent to System Inversion Techniques, namely the presence of zeros. To understand AIRA we begin by examining this foundation from the perspective of System Inversion in the case of the linear discrete time system $G(z)$ shown in Figure 3.17 with inputs U and outputs Y.

Figure 3.17: *System Transfer Function, with input U and output Y.*

Inverse systems derived through System Inversion Techniques can be defined in two broad categories: left inversion and right inversion. Left inversion techniques are used in input observers and right inversion techniques are used in feed forward control. In the left inverse system $G_L(z)$, shown in Figure 3.18, which satisfies the condition $G_L(z)G(z) = I$ where I is an Identity Matrix, U is output when Y is input. Similarly in the right inverse system $G_R(z)$, shown in Figure 3.19, which satisfies the condition $G(z)G_R(z) = I$, U is output given the desired output Y.

Figure 3.18: *Left Inverse System, with input Y and output U.*

Figure 3.19: *Right Inverse System, with input Y and output U.*

Moving forward, focus will be directed toward the left inversion technique by highlighting the inversion process for SISO (Single Input Single Output) and MIMO (Multiple Input Multiple Output) Systems. A system is invertible when there is a one-to-one relationship between its input and output [33]. Consider the SISO system $G(z)$ defined by the transfer function:

$$G(z) = \frac{Y(Z)}{U(Z)} \tag{3.1}$$

If the system is invertible then the system inversion process is relatively straight forward, $G_L(z)$ is determined by the transfer function:

$$G_L(z) = \frac{U(Z)}{Y(Z)} = \frac{1}{G(Z)} \tag{3.2}$$

The locations of the system's zeros affect the phase characteristics of the inverted system. The locations of the zeros of a linear time invariant system are related to the characteristics of its phase spectral function. A minimum phase system is a linear time invariant system that is, with its inverse, casual and stable for small times k [33]. It will have all its zeros inside the unit circle and on the left hand side of the complex plane. A maximum phase system is a linear time invariant system that is stable and causal with an inverse that is causal and unstable for large times k [33]. Namely, it will have zeros on the right hand side of the complex plane and zeros outside the unit circle. A mixed phase system has zeros on either side of the unit circle. Consider the MIMO system defined by the transfer function:

$$\begin{bmatrix} Y_1(z) \\ \vdots \\ Y_n(z) \end{bmatrix} = \begin{bmatrix} G_{11}(z) & G_{12}(z) & \dots & G_{1m}(z) \\ \vdots & \vdots & \vdots & \vdots \\ G_{n1}(z) & G_{n2}(z) & \dots & G_{nm}(z) \end{bmatrix} \begin{bmatrix} U_1(z) \\ \vdots \\ U_m(z) \end{bmatrix} \tag{3.3}$$

In this case, it is evident that the inversion process becomes non trivial which is why a more sophisticated approach must be adopted to match the complexity of the inversion process. This is where AIRA is employed to perform the system inversion. A generalised inverse is a matrix that has some of the properties of an inverse matrix [22]. The Moore-Penrose Generalised Inverse G_I of G satisfies all of the conditions below [22]:

- $GG_IG = G$
- $G_IGG_I = G_I$
- $(GG_I)^T = G_IG$
- $(G_IG)^T = GG_I$

The Moore-Penrose Generalised Inverse exists for any matrix G and when G has full rank, G_I can be expressed using a simple equation [2]. When the linear system G has linearly independent columns, the left inverse G_L in $G_LG = I$ can be expressed as [2]:

$$G_L = (G^TG)^{-1}G^T \tag{3.4}$$

When the linear system G has linearly independent rows, the right inverse G_R in $GG_R = I$ can be expressed as [2]:

$$G_R = G^T(GG^T)^{-1} \tag{3.5}$$

The linear least squares algorithm is used when fitting a mathematical model to data when the idealised value (output) is expressed linearly in terms of the unknown parameters (inputs) of the model. This applies to our case where we would like to determine the unknown inputs using the generated outputs. We use the Moore-Penrose Generalised Inverse to determine the solution to the equation of the form $Y = GU$ where U solves the least squares problem [3].

Input Reconstruction

Consider the linearised state space representation defined in Appendix C representing the decoupled lateral and longitudinal dynamics of the Meraka Modular UAV:

$$\begin{bmatrix} \boldsymbol{\Delta \dot{x}}_{long} \\ \boldsymbol{\Delta \dot{x}}_{lat} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{long} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_{lat} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta x}_{long} \\ \boldsymbol{\Delta x}_{lat} \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}_{long} \\ \boldsymbol{B}_{lat} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_{long} \\ \boldsymbol{u}_{lat} \end{bmatrix} \tag{3.6}$$

The state space representation is discretised to give the following linear discrete time system:

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{G}\boldsymbol{u}_k \tag{3.7}$$
$$\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{J}\boldsymbol{u}_k \tag{3.8}$$

where $\boldsymbol{x}_k \in \mathbb{R}^n, \boldsymbol{u}_k \in \mathbb{R}^m, \boldsymbol{y}_k \in \mathbb{R}^p, \boldsymbol{F} \in \mathbb{R}^{n\times n}, \boldsymbol{G} \in \mathbb{R}^{n\times m}, \boldsymbol{H} \in \mathbb{R}^{p\times n}$ and $\boldsymbol{J} \in \mathbb{R}^{p\times m}$. We would like to reconstruct the commanded input history $\boldsymbol{U}_r$ of length r:

$$\boldsymbol{U}_r = \begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_r \end{bmatrix} \tag{3.9}$$

where r is a non negative integer, using the measured output sequence $\boldsymbol{Y}_r$:

$$\boldsymbol{Y}_r = \begin{bmatrix} \boldsymbol{y}_0 \\ \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_r \end{bmatrix} \tag{3.10}$$

To determine the output sequence we find expressions for the output $\boldsymbol{Y}_r$ and expanding $\boldsymbol{Y}_r$ in Equation 3.10 we get:

$$\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{J}\boldsymbol{u}_k \tag{3.11}$$
$$\boldsymbol{y}_{k+1} = \boldsymbol{H}\boldsymbol{x}_{k+1} + \boldsymbol{J}\boldsymbol{u}_{k+1} \tag{3.12}$$
$$= \boldsymbol{H}\boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{H}\boldsymbol{G}\boldsymbol{u}_k + \boldsymbol{J}\boldsymbol{u}_k \tag{3.13}$$

and continuing we get:

$$\boldsymbol{Y}_r = \begin{bmatrix} \boldsymbol{y}_k \\ \boldsymbol{y}_{k+1} \\ \vdots \\ \boldsymbol{y}_{k+r} \end{bmatrix} = \begin{bmatrix} \boldsymbol{H} \\ \boldsymbol{H}\boldsymbol{F} \\ \vdots \\ \boldsymbol{H}\boldsymbol{F}^r \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} \boldsymbol{J} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{H}\boldsymbol{G} & \boldsymbol{J} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{H}\boldsymbol{F}^{r-1}\boldsymbol{G} & \boldsymbol{H}\boldsymbol{F}^{r-2}\boldsymbol{G} & \dots & \boldsymbol{H}\boldsymbol{G} & \boldsymbol{J} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_k \\ \boldsymbol{u}_{k+1} \\ \vdots \\ \boldsymbol{u}_{k+r} \end{bmatrix} \tag{3.14}$$

where $\boldsymbol{\Gamma}_r \in \mathbb{R}^{(r+1)p\times n}$ is the Observability Matrix given by:

$$\boldsymbol{\Gamma}_r = \begin{bmatrix} \boldsymbol{H} \\ \boldsymbol{HF} \\ \boldsymbol{HF}^2 \\ \vdots \\ \boldsymbol{HF}^r \end{bmatrix} \tag{3.15}$$

and where the Markov Parameters $\mathbb{M}_a$ [9]:

$$\mathbb{M}_a = \begin{cases} \boldsymbol{J} & a = 0 \\ \boldsymbol{HF}^{a-1}\boldsymbol{G}, & a \geq 1 \end{cases} \tag{3.16}$$

are used to construct the Hankel Matrix $\boldsymbol{M}_r \in \mathbb{R}^{(r+1)p \times (r+1)m}$ given by:

$$\boldsymbol{M}_r = \begin{bmatrix} \boldsymbol{J} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{HG} & \boldsymbol{J} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{HF}^{r-2}\boldsymbol{G} & \boldsymbol{HF}^{r-3}\boldsymbol{G} & \dots & \boldsymbol{J} & \boldsymbol{0} \\ \boldsymbol{HF}^{r-1}\boldsymbol{G} & \boldsymbol{HF}^{r-2}\boldsymbol{G} & \dots & \boldsymbol{HG} & \boldsymbol{J} \end{bmatrix} \tag{3.17}$$

To determine the unknown input history $\boldsymbol{U}_r$ and unknown initial state $\boldsymbol{x}_0$ we define the output sequence $\boldsymbol{Y}_r$ adapted from Equation 3.14 as:

$$\boldsymbol{Y}_r = \boldsymbol{\Gamma}_r \boldsymbol{x}_0 + \boldsymbol{M}_r \boldsymbol{U}_r = \Psi_r \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{U}_r \end{bmatrix} \tag{3.18}$$

where $\Psi_r \in \mathbb{R}^{(r+1)p \times [n+m(r+1)]}$ is given by:

$$\Psi_r = \begin{bmatrix} \boldsymbol{\Gamma}_r & \boldsymbol{M}_r \end{bmatrix} \tag{3.19}$$

Notice how only the initial state and sensor measurements are used in 3.18, this adds to the increased accuracy of AIRA. Define $l = rank(\boldsymbol{F}) > 0$, the linear discrete time system 3.7, 3.8 is l delay input and state observable when there exists an input history of length $r \geq l$ such that $\mathbb{U}_r = \{\boldsymbol{0}\}$ for all $r_0 \geq 1$ where [25]:

$$r_0 = \left\lceil \frac{n}{l-m} \right\rceil - 1 \tag{3.20}$$

$$\mathbb{U}_r = \left\{ \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{U}_r \end{bmatrix} \in \mathbb{R}^{n+(r+1)m} : \boldsymbol{Y}_r = 0 \right\} \tag{3.21}$$

and $\lceil a \rceil$, $\mathcal{N}$, $\mathbb{U}_r = \mathcal{N}(\Psi_r)$ denotes the smallest integer greater than or equal to a, the null space, and the input and state unobservable subspace respectively. Definition 3.5 in [25] states that a system is input and state observable when $\mathbb{U}_r = \{\boldsymbol{0}\}$ for all $r \geq r_0$. According to Theorem 3.6 in [25], Ψ_r has full column rank for all $r \geq r_0$ when:

- the discrete time system is input and state observable
- there exists $r \geq r_0$ where $rank(\Psi_r) = n + (r+1)m$
- $rank(\Psi_r) = n + (r+1)m$ for all $r \geq r_0$
- $\boldsymbol{Y}_r = 0$ if and only if $\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{U}_0 \end{bmatrix} = 0$ for all $r \geq r_0$
- and $rank(\Psi_{n-1}) = n(m+1)$

This enables us to determine the unique solution of Equation 3.18 which is:

$$\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{U}_r \end{bmatrix} = \Psi_r^{\dagger} Y_r = \left(\Psi_r^T \Psi_r\right)^{-1} \Psi_r^T Y_r \tag{3.22}$$

where † represents the Moore-Penrose Generalised Inverse. The presence of invariant zeros in the linear discrete time system 3.7, 3.8 implies that the input and state unobservable subspace $\mathbb{U}_r \neq \{\boldsymbol{0}\}$ and according to Definition 3.5 in [25], the system is no longer input and state observable. Consequently exact reconstruction is no longer possible and the Moore-Penrose Generalised Inverse is no longer given by Equation 3.22. By accounting for the locations of the invariant zeros relative to the unit disc in the discrete plane, approximate reconstruction can be achieved. According to Definition 4.1 in [25], the reconstructed state and input is given by:

$$\boldsymbol{x}_0 = \boldsymbol{x}_{0,rec} + \boldsymbol{x}_{0,unrec}, \tag{3.23}$$

$$\boldsymbol{U}_r = \boldsymbol{U}_{r,rec} + \boldsymbol{U}_{r,unrec} \tag{3.24}$$

where rec and $unrec$ are the observable and unobservable state and input respectively. This suggests that the output sequence or response for the linear time system has contributions from unobservable system inputs and states $\boldsymbol{x}_{0,unrec}, \boldsymbol{U}_{r,unrec}$ which in turn corrupt and affect the accuracy of the reconstructed input. We now investigate the use of the AIRA using a few examples [9] to show that the reconstructed input $\boldsymbol{U}_{r,unrec}$ is small for small and large values of k, respectively in a system that has only minimum phase and non minimum phase zeros. In both cases the reconstruction process involves a batch-processing algorithm. Note that in the mixed-phase case, the input estimates have good accuracy for neither small or large values of k making the input reconstruction non causal.

3.2.2 Minimum Phase Zeros

In applying AIRA, the application designer needs to identify and account for the locations of the system's zeros relative to the unit disc. Consider now a minimum phase system with initial state x_0, simulated using the MATLAB script in Appendix D.1:

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} 0.6 & -0.22 & 0.096 \\ 0.5 & 0 & 0 \\ 0 & 0.125 & 0 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix} \boldsymbol{u}_k \tag{3.25}$$

$$\boldsymbol{y}_k = \begin{bmatrix} 0 & 0.5 & -2 \end{bmatrix} \boldsymbol{x}_k \tag{3.26}$$

$$\boldsymbol{x}_0 = \begin{bmatrix} 2 \\ 0.1 \\ -1 \end{bmatrix} \tag{3.27}$$

The region inside the disc corresponds to the left half of the s-plane. Figure 3.20 shows the pole zero plots of the minimum phase system with transmission zeros inside the unit disc in the z plane.

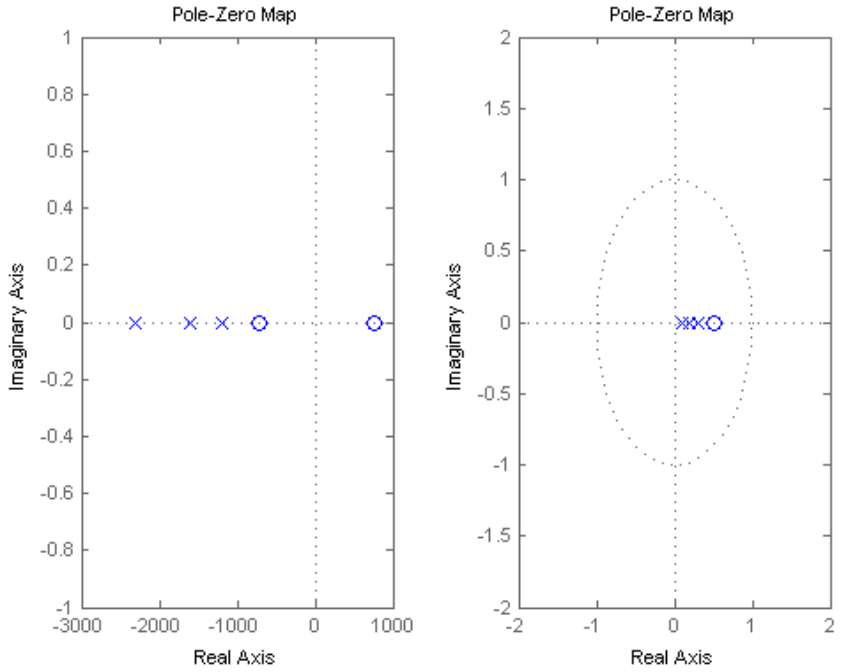


Figure 3.20: *A minimum phase system's pole zero plots in the s and z domains*

If all of the transmission zeros are contained in the open unit disc, then approximate causal reconstruction is possible for sufficiently large data sets and large times k. This is a consequence of the unobservable components dying out with the increase in times k as shown in Figure 3.21.

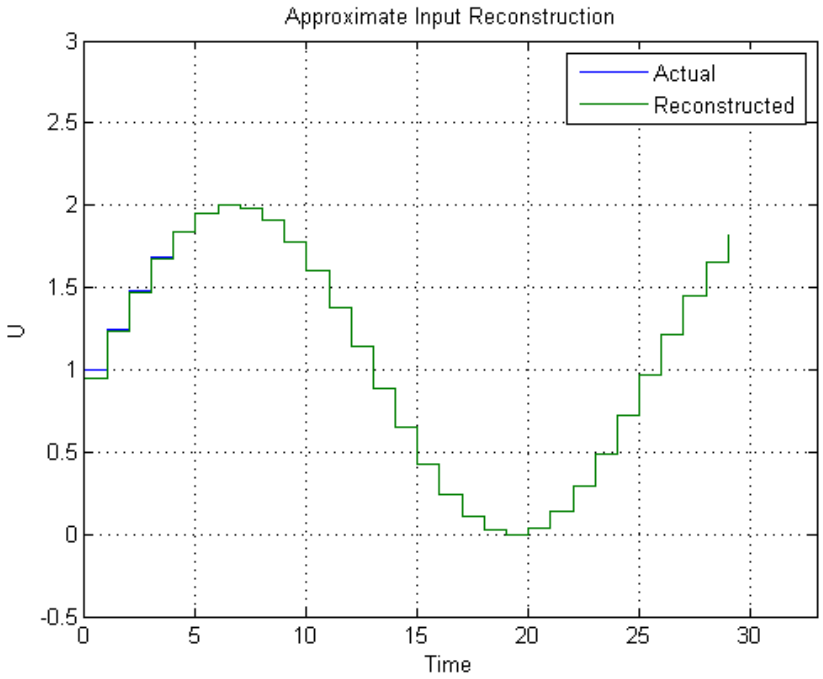


Figure 3.21: *Reconstructed input for a minimum phase system, used for illustrative purposes - the units are not significant.*

3.2.3 Non Minimum Phase Zeros

If on the other hand, all the transmission zeros are contained in the complement of the closed unit disc, then approximate non causal reconstruction is possible for sufficiently large data sets and small times k. Consider now the minimum phase system given by equations 3.25 - 3.27 with $\boldsymbol{y}_k = \begin{bmatrix} 0 & 0.5 & -6 \end{bmatrix} \boldsymbol{x}_k$, simulated using the MATLAB script in Appendix D.2. Figure 3.22 shows the pole zero plots in the s and z domains respectively and the non minimum phase zero outside the unit circle. Figure 3.23 shows the reconstructed input where the unobservable input increases for large times k.

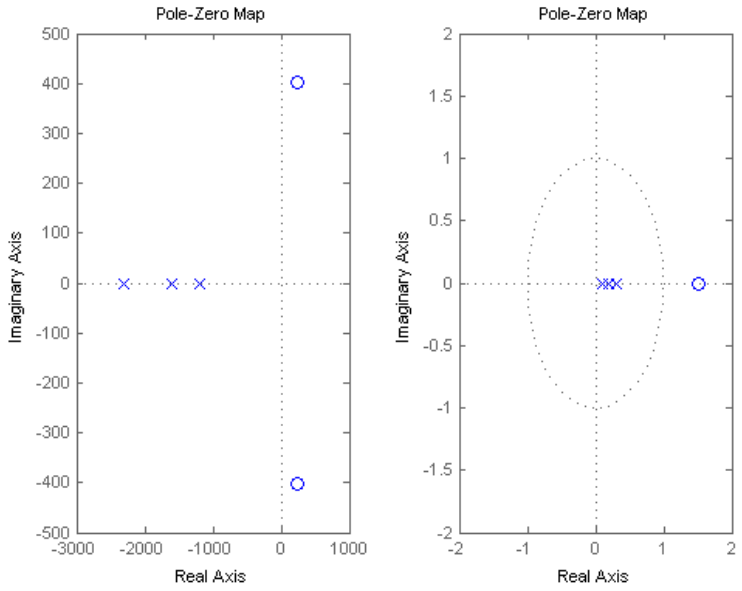


Figure 3.22: *A nonminimum phase system's pole zero plots in the s and z domains.*

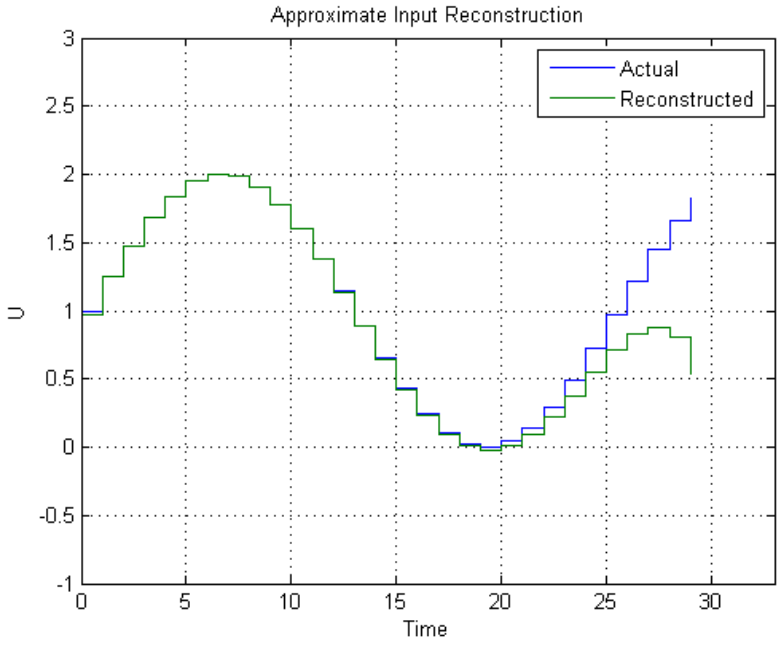


Figure 3.23: *Reconstructed input for a nonminimum phase system, used for illustrative purposes - the units are not significant.*

Consequently if all of the system transmission zeros are contained in either the open unit disc or the complement of the closed unit disc, the approximate non causal reconstruction is possible for sufficiently large data sets and small times k.

3.2.4 Zeros on Unit Circle

If at least one transmission zero lies on the unit circle, then approximate input reconstruction is not possible since a persistent reconstruction error will corrupt the reconstructed inputs. Consider now the system given by equations 3.25 - 3.27 with $\boldsymbol{y}_k = \begin{bmatrix} 0 & 0.5 & -4 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 0 \end{bmatrix} \boldsymbol{u}k$, and $\boldsymbol{y}_k = \begin{bmatrix} 0 & 0.5 & 4 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 0 \end{bmatrix} \boldsymbol{u}k$ simulated using the MATLAB script in Appendix D.3. Figures 3.24 and 3.25 show the pole zero plots in the s and z domains respectively and the zeros on either side of the unit circle. Figure 3.26 and 3.27 show the reconstructed input where the unobservable input remains constant for large times k.

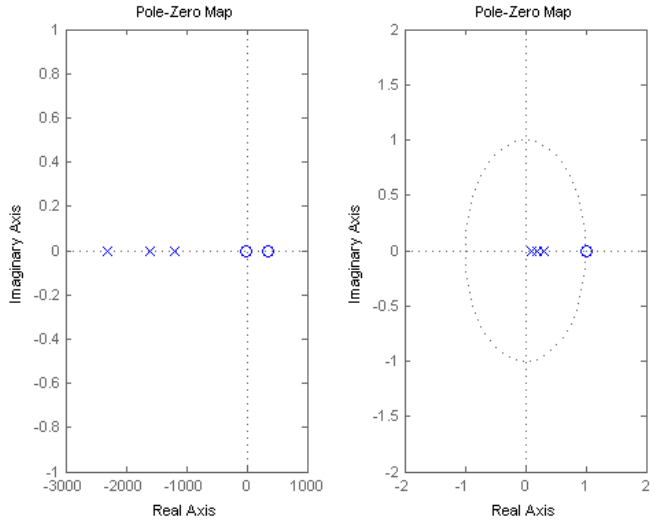


Figure 3.24: *A system's pole zero plots in the s and z domains with a zero on the unit circle.*

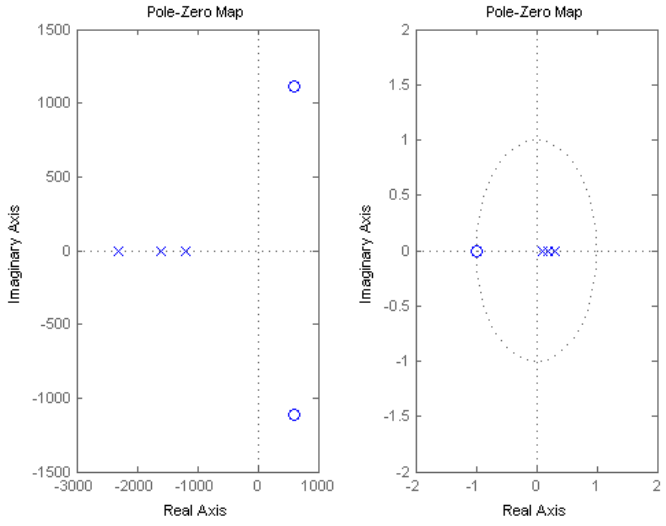


Figure 3.25: *A system's pole zero plots in the s and z domains with a zero on the unit circle.*

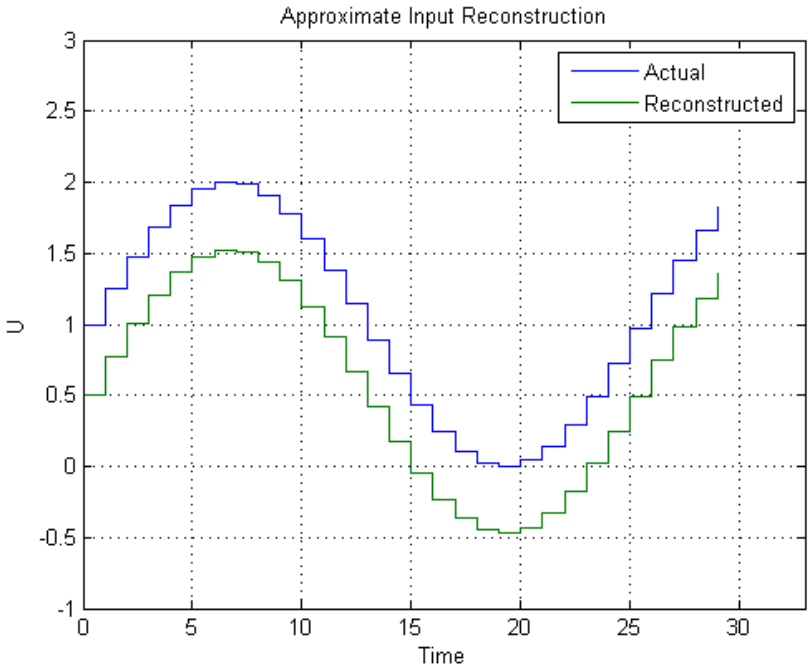


Figure 3.26: *Reconstructed input for system with zeros on unit circle, used for illustrative purposes - the units are not significant.*

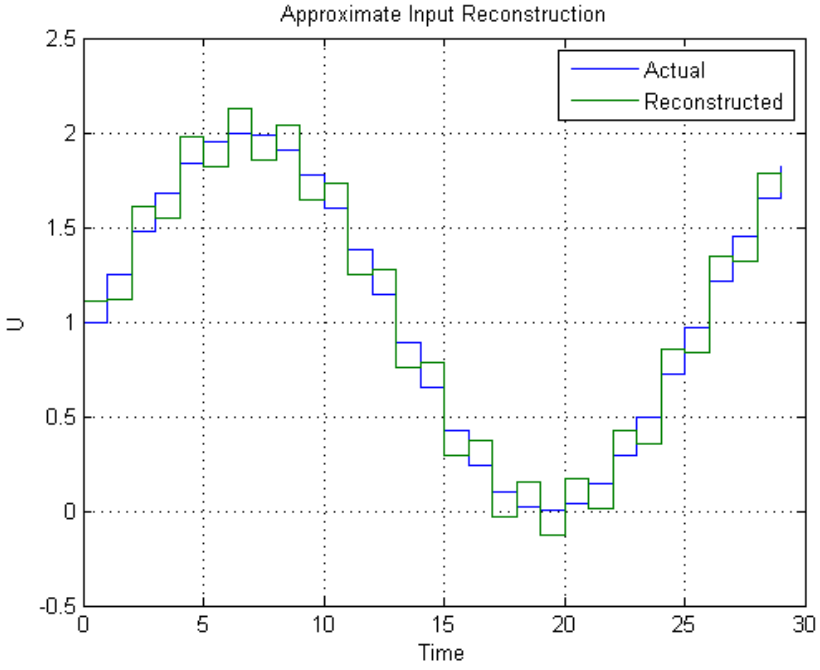


Figure 3.27: *Reconstructed input for system with zeros on unit circle, used for illustrative purposes - the units are not significant.*

3.2.5 Online Computation

The previous examples were investigated using the following implementation of AIRA in MATLAB. This script can be ported to an embedded system for online or realtime estimation of inputs. The script is included to highlight the simplicity of implementing AIRA given that such sophisticated functionality can be achieved through a couple lines of MATLAB code.

```
%% Approximate Input Reconstruction Algorithm @author Sandile Memela

% Initialise variables
IRA.r1 = SYS.r;
IRA.r2 = SYS.r;
IRA.p = size(SYS.H,1);
IRA.n = size(SYS.H,2);
IRA.m = size(SYS.G,2);
IRA.l = rank(SYS.F);
IRA.Mrr = zeros(IRA.p*(IRA.r1+1), IRA.m*(IRA.r2+1));
IRA.Gr = zeros(IRA.p*(IRA.r1+1), IRA.n);

% Generate the hankel matrix
for i = 0:IRA.r2
    a = 0;
    for j = 0:IRA.r1
        if j >= i
            if a == 0
                IRA.Mrr(IRA.p*(j)+1:IRA.p*(j)+IRA.p, IRA.m*(i)+1:IRA.m*(i)+IRA.m) = SYS.J;
            else
                IRA.Mrr(IRA.p*(j)+1:IRA.p*(j)+IRA.p, IRA.m*(i)+1:IRA.m*(i)+IRA.m) =
                    SYS.H*(SYS.F^(a-1))*SYS.G;
            end
            a = a+1;
        end
    end
end

% Generate the observability matrix
for j = 0:IRA.r1
    IRA.Gr(IRA.p*(j)+1:IRA.p*(j)+IRA.p,:) = SYS.H*(SYS.F^(j));
end

% Calculate and shift the reconstructed input
IRA.U = pinv([IRA.Gr IRA.Mrr].'*[IRA.Gr IRA.Mrr])*[IRA.Gr IRA.Mrr].'*SYS.y';
IRA.U = IRA.U(IRA.l+1:size(IRA.U,1),:);

%% Plot the reconstructed input
h1 = stairs(SYS.k(1:SYS.r-IRA.l),[SYS.u(:,1:SYS.r-IRA.l)' IRA.U(1:SYS.r-IRA.l,:)],'-');
legend(h1,'Actual','Reconstructed');
title('Approximate Input Reconstruction');
axis([0 SYS.r -1 3.0]);
xlabel('Time');
ylabel('U');
```

An advantage of this algorithm is its simplicity in implementation. A major disadvantage is the large computational requirement for systems with a large number of inputs and outputs. This results as a consequence of having to calculate the inverse of a square matrix that grows in size in response to the desired number of inputs to be estimated. In light of this, the decision is made in this research project to perform input estimation using this algorithm and SISO systems. This is done in an effort to reduce the effect of the unobservable components present in the reconstructed input. To simplify the computational complexity and to ensure the target fault diagnostic system's feasibility, inputs and outputs that seem to have observable and correlated relationships will be selected when abstracting a SISO version of the linearised and discretised MIMO model of the aircraft. In the case of the mathematical model of the aircraft that has been defined, the input and output state space matrices can be inspected to see which states are more completely represented.

3.3 Subsystem Characterisation

The fault modelling and identification process that follows requires a clearly defined description of the functional relationships that exist between subsystem components. To this end, a Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) is performed in order to characterise the subsystem components. This enables us to identify which components need to be modelled first, how, why and how they are functionally related which isn't evident from looking at the bare System or Subsystem Component Overview. This is critical to selecting the relevant subsystems and constraining the complexity of the system model used to perform diagnoses in HyDE.

3.3.1 Meraka Modular UAV Subsystems

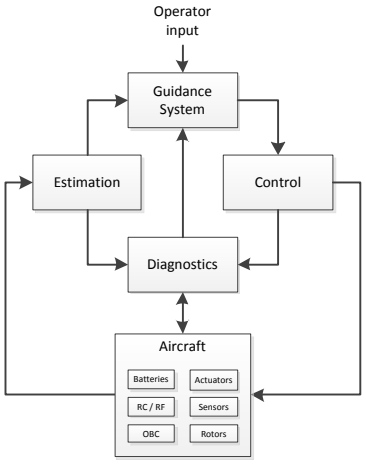


Figure 3.28: *System Component Overview, showing detail for aircraft subsystems.*

The UAV's subsystem components, estimation and guidance systems were examined which resulted in the component overview shown in Figure 3.28. A detailed subsystem overview, shown in Figure 3.29, was generated in order to identify the UAV's components. The major systems depicted are the autonomous aircraft and the ground station. The subsystem level component schematic represents key subsystem components. The schematic is used to model the structural and functional relationships that determine the behaviour of the onboard subsystems.

Table 3.1 is a list of all of subsystem components and their functional component groupings. Focus is placed on electronic components at the subsystem level. Effects that are not directly related to the aircraft are excluded, an example of this is the ambient temperature. The subsystem component manifest is used as one of the inputs to the Reliability Analysis adapted from comparable work in [46] and performed in this project.

3.3.2 Failure Mode and Effects Analysis

The Failure Mode and Effects Analysis is a bottom up reliability analysis that is used for qualitatively analysing the ways in which a system fails [40]. It contains tabulated data for each of the subsystem components in the UAV beginning at the component level and assigns different failure types or modes, causes and effects to each component. Each part and its various failure modes that can potentially result in an end effect failure are considered as well as determining the severity of each mode and the compensation procedures. The entire system is broken down into a list of individual components which is used as one of the process inputs when performing the FMEA, shown in Figure 3.30, as outlined below:

1. for each of the component inputs, determine the ways that the input can go wrong

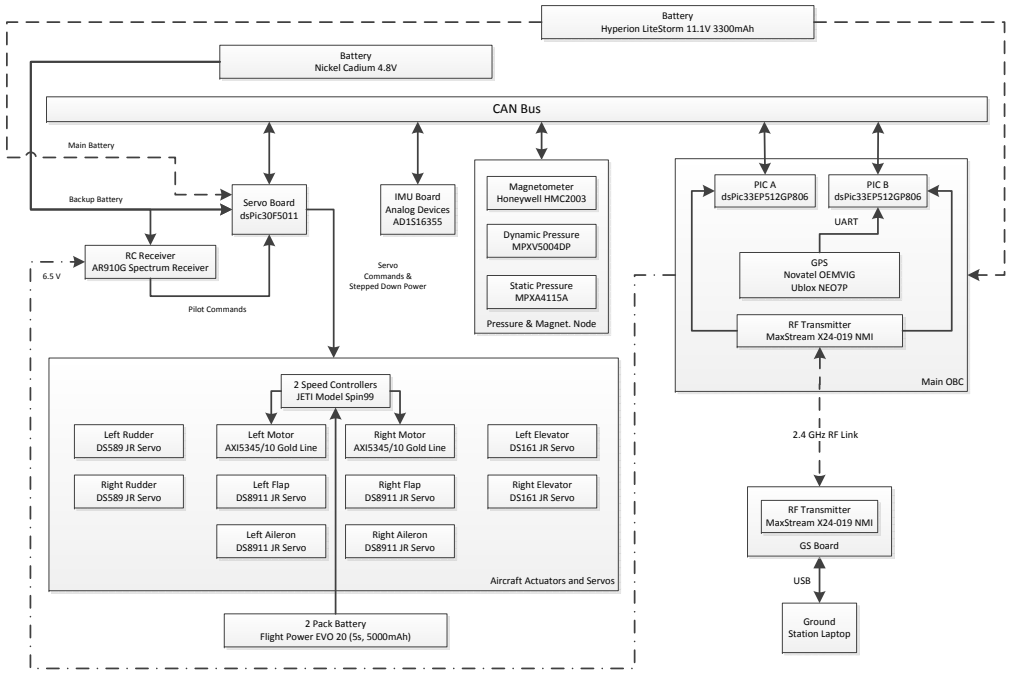


Figure 3.29: *Detailed Aircraft Subsystem Component Overview*

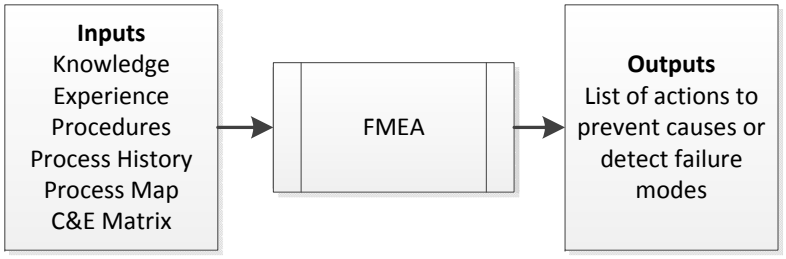


Figure 3.30: *Failure Mode and Effects Analysis, detailing process inputs and outputs.*

2. for each failure mode identified, determine the effects and severity level
3. identify the potential causes for each failure mode and select the associated level of occurrence
4. list the current controls or compensation methods for each cause and select the level of detection for each cause
5. calculate the risk priority number (RPN) who's range is between 1 (not severe) to 10 (very severe):
   (a) predicted severity: gauges the importance of the effect on the aircraft
   (b) occurrence: frequency given to the cause that brings about failure mode
   (c) detection levels: the ability of the control scheme to detect the cause of the failure
   (d) $RPN = severity \times occurrence \times detection$
6. develop the recommended compensation actions prioritised by high RPNs.

A simplified FMEA is done and can be found in Appendix E.1. As a consequence, only the severity of each failure mode and component failure rate is considered.The components are assigned a severity to quantify how their failure impacts the entire system. The severity categories are used to categorise the severity of the failure effects. The failure effects that are of concern for this research

Subsystem Components		
Aircraft		
1.1	Avionics Battery	Powers Onboard Computer (OBC) and Servo Board
1.2	Backup Battery	Powers Servo Board and RC Receiver
1.3	Motor Batteries	Powers motors
Actuation		
2.1	RC Receiver	Allows manual control of the actuation system
2.2	Servo Board	Applies actuation commands to servos and motors
2.3	Speed Controller	Controls the speed of the motors
2.4	Left Rudder	Servo deflects left rudder control surface
2.5	Right Rudder	Servo deflects right rudder control surface
2.6	Left Elevator	Servo deflects left elevator control surface
2.7	Right Elevator	Servo deflects right elevator control surface
2.8	Left Aileron	Servo deflects left aileron control surface
2.9	Right Aileron	Servo deflects right aileron control surface
2.10	Left Flaperon	Servo deflects left flaperon control surface
2.11	Right Flaperon	Servo deflects right flaperon control surface
2.12	Left Motor	Generates thrust on the left of the aircraft
2.13	Right Motor	Generates thrust on the right of the aircraft
Control		
3.1	PICA	Main aircraft controller
3.2	PICB	Controls the GPS module
3.3	RF Transmitter	Transmits data to Ground Station receiver
3.4	CAN Bus	Enables communication between subsystems
Sensors		
4.1	GPS	Tracks aircraft's position, velocity
4.2	IMU	Measures translation accelerations and angular rates
4.3	Magnetometer	Determines aircraft's attitude
4.4	Dynamic Pressure	Determines airspeed of aircraft
4.5	Static Pressure	Determines aircraft's altitude

Table 3.1: *Subsystem Component Manifest, outlining component groupings, indices and functions.*

#	Severity Category	End Effect
A	Uncontrolled emergency landing	Risk of catastrophic damage high
B	Controlled emergency landing	Risk of catastrophic damage low
C	Mission critical failure	Spoiling or loosing flight data

Table 3.2: *Severity Categories, with associated undesirable end effects.*

have been separated into three ranked categories, in order of severity as shown in Table 3.2, and include:

- failure effects that affect the aircraft's ability to fly manually i.e. loss of radio control or a stuck control surface that results in the loss of trim, in this case an uncontrolled landing is expected with a high possibility for catastrophic damage
- failure effects that affect the aircraft's ability to respond commands i.e. loss of thrust but the aircraft remains trimmed, in this case a controlled landing is expected with a low possibility for damage
- failure effects that affect the success of the flight mission and that are considered mission critical i.e. the faulty sensor measurements that result in the loss of crucial flight data.

A list of the component failure modes for one of the UAV's actuators is shown in Table 3.3. A component's failure modes are the ways in which it malfunctions based on an identifiable cause. The cause of the failure initiates a process which leads to the failure mode and serves as the source of the malfunction. A detailed list of the UAV's Component Failure Modes can be found in Appendix E.1.1.

Failure Mode Number	Failure Mode	Failure Cause
2.6.1	Stuck failure	Servo halts or gets jammed
2.6.2	Hardover failure	Servo rotates in one direction
2.6.3	Floating failure	Loose mechanical connection
2.6.4	Partial loss failure	Slipping mechanical connection
2.6.5	Unknown	Unexpected actuator behaviour

Table 3.3: *Component Failure Modes, shown with component indices, excerpt from Appendix E.1.1.*

Once the component failure modes have been identified, the effects of the failure on the entire system are determined as shown in Table 3.4. The effects are considered chronologically starting with local subsystem effects before ultimately ending with the final system level effects. A detailed list of the UAV's Failure Effects Analysis can be found in Appendix E.1.2.

Failure Mode Number	Failure Effects (Local)	Failure Effects (Next Level)	Failure Effects (End Effect)	Severity Category
1.1.1	Non ideal voltage supply	On board components behave erratically	Inaccurate flight data	C
1.1.2	Low voltage supply	OBC and Servo Board looses avionics battery power	Controlled emergency landing	B
1.1.3	Unknown battery fault	OBC and Servo Board malfunctions	Controlled emergency landing	B

Table 3.4: *Component Failure Effects, identified at different system levels, excerpt from Appendix E.1.2.*

3.3.3 Failure Detection Methods

The failure detection methods, as shown in Table 3.5, are the methods that can be applied to recognise the failure mode. The compensating features consider either reconfiguration actions that can be performed autonomously or the future redesign possibilities for increasing the reliability of the aircraft. The failure detection is directly related to the failure cause and therefore the compensation action that can be performed or taken. A detailed list of the UAV's Failure Detection Methods can be found in Appendix E.1.3.

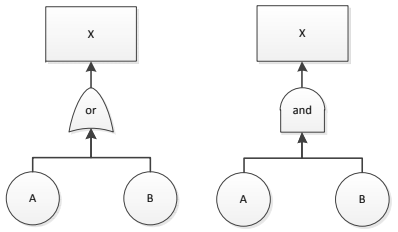
3.3.4 Fault Tree Analysis

The Fault Tree Analysis is a top down reliability analysis method that is used for quantitatively analysing the ways in which a system can fail [40]. Knowledge about the failure relationships defined during the FMEA is one of the means available that are used to organise components. The failure rates for system level end effect events are determined using fault trees constructed from AND and predominantly OR gates. The functional relationships between the components including local, subsystem and system level effects are used to identify the non redundant subsystems that must

Failure Mode Number	Failure Detection Method	Compensation Action
1.1.1	Battery precision voltage sensor	Monitor battery level, notify ground station operator
1.1.2	Battery precision voltage sensor	Prevent start up of on board components then activate backup battery power for select components
1.1.3	Battery precision voltage sensor	Activate backup battery for select components

Table 3.5: *Failure Detection Methods, outlining compensation actions for the particular component.*

fail in order for the end effect event to occur using an iterative top-down process for major and individual system components. The FTA end effects correspond to the severity categories in the FMEA in the case of the Reliability Analysis for the Meraka Modular UAV. Software packages such as RAM Commander [45] or the Logan Fault Tree and Event Analysis Software [32] can be used to construct both the fault trees and perform the FMEA.



(a) A OR gate (b) A AND gate

Figure 3.31: Component Fault Trees, two types shown.

The boolean OR and AND gates are shown in Figure 3.31 and these constitute the basic building blocks for calculating the component failure rates at subsystem levels in the fault trees. The top event X, representing a non redundant system as shown in Figure 3.31a, occurs if component A or B fails. The top event X, representing a redundant system, shown in Figure 3.31b, occurs if both component A and B fails. The probability of failure p_X for a system failure characterised by an OR gate is given by:

$$p_X = p_A + p_B - (p_A p_B) \tag{3.28}$$

The term $p_A p_B$ arises from the probability that both A and B, shown in Figure 3.31a, fail simultaneously i.e. the servo board receives commands erratically from the RC receiver because of insufficient battery voltage or corrupted commands received from the RC transmitter. If we assume that no two faults within the same non redundant component fault tree occur simultaneously, then we can ignore the term $p_A p_B$. The probability for the system failure p_X for a system characterised by an AND gate is given by:

$$p_X = p_A p_B \tag{3.29}$$

where both A and B as shown in Figure 3.31b must fail for X to occur. The Nonelectronic Parts Reliability Data (NPRD) 1991 Handbook [54] is used to determine estimates for probabilities of failure using component failure rates and the RAC's Failure Mode Distributions 1991 Handbook [13] can be used to determine the fault mode distributions for each component. This is useful when it is necessary to determine the probability that the component is in either failure mode especially

for components modelled in HyDE that have multiple locations and transitions with transition probabilities. The upper bounds for the component failure rates are chosen as pessimistic estimates of the probabilities of failure quoted in the NPRD Handbook. To determine the equivalence between the component failure rates and probabilities of failure we use the NASA Fault Tree Handbook with Aerospace Applications, that defines the component probability p is by [17]:

$$p = 1 - e^{-\lambda t} \tag{3.30}$$

where λ is the component failure rate with the units probabilities per unit time and t is the time interval. Failure rates defined in the NPRD Handbook are quoted as one failure per million hours of usage. For the purposes of this research, the rates are defined with respect to $t = 1$ hour corresponding to a 1 hour UAV flight mission. With the maximum NPRD failure rate being failures per million hours, for each subsystem component $\lambda < 0.1$ which implies that 3.30 can be simplified to [17]:

$$p = \lambda t \tag{3.31}$$

which means that the quoted component failure rates can be divided by 1000000 which yields the approximate probability of failure for a component during one UAV flight mission. Failure rates from the NPRD Handbook provide adequate estimates and no conversion factors will be applied in the case of failure rates for the Meraka Modular UAV seeing that the relative probabilities for the component failures are what is important to rank component failures and not the actual failure probabilities themselves. The details behind the UAV's Fault Tree Analysis can be found in Appendix E.2.

3.3.5 Hazards Analysis Matrix

The severity categories, listed in Table 3.2, and component failure rates determined during the FMEA and FTA are used to generate the Hazards Analysis Matrix (HAM), shown in Figure 3.32. The severity categories divide the vertical axis and the failure rates per hour divide the horizontal axis. The components with the most critical and the least critical components are located on the upper right and lower left quadrants respectively. The HAM was used as a guide when characterising the subsystem components, as shown in Figure 3.33, in the UAV so that priority is given to the most critical components when developing the declarative model of the subsystem components for use in the fault diagnostic system.

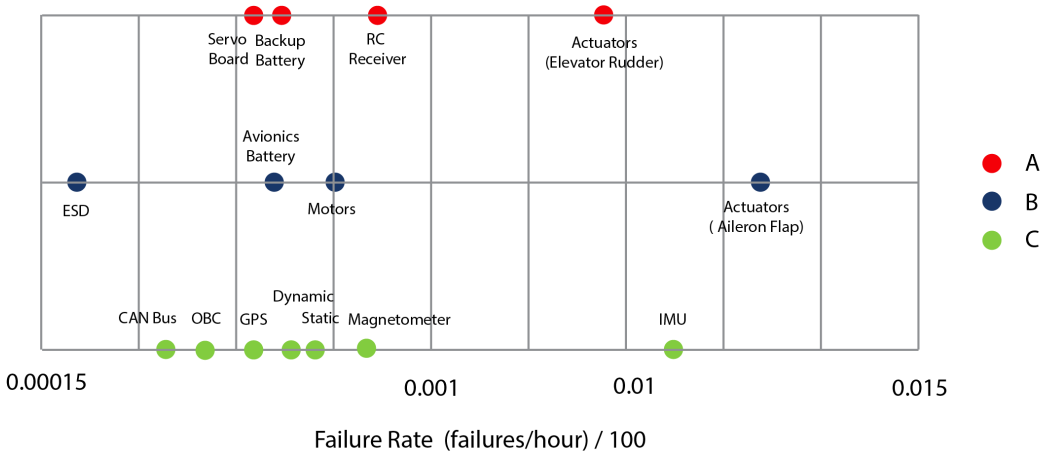


Figure 3.32: *Hazards Analysis Matrix, used to identify critical subsystem components by severity category.*

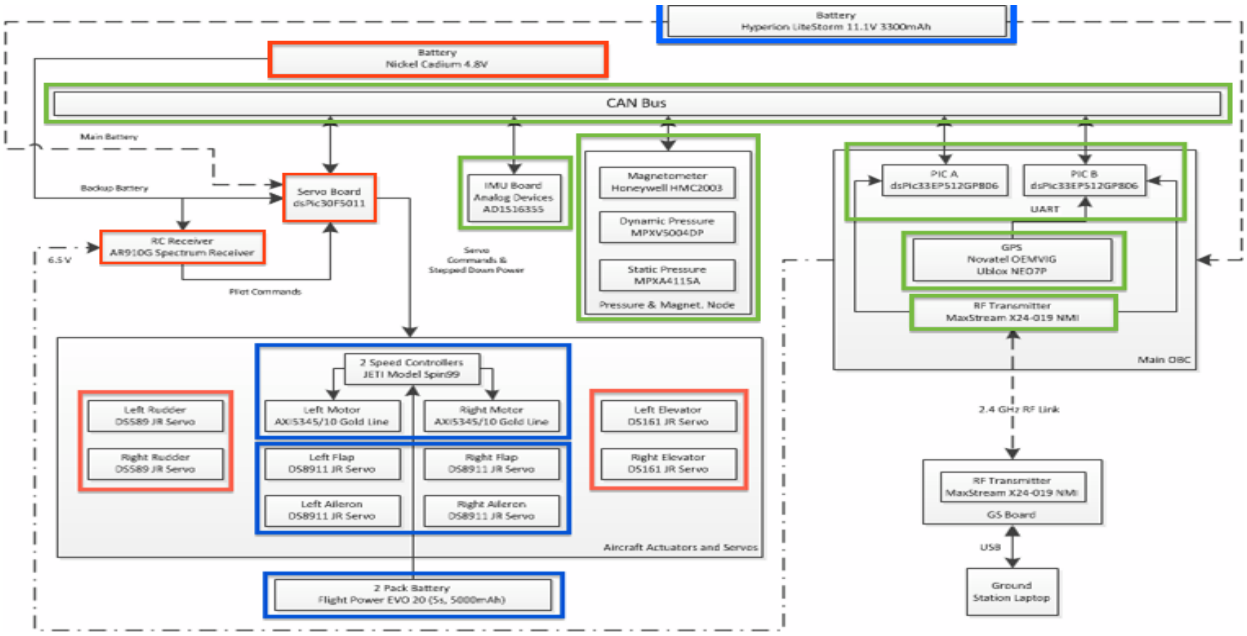


Figure 3.33: *Characterised Subsystem Model, highlighting critical subsystem components.*

The results of the Reliability Analysis appear to correlate well with what one would expect from a process that characterises the onboard subsystems on the UAV. The rudders and elevators are the most critical components as they directly affect the aircraft's ability to fly safely and have the greatest effect on manoeuvrability. The subsystem components that enable the UAV to make use of these components have also been identified as critical and in doing so, it becomes evident that just modelling the actuators as part of the FDI process is not sufficient and in addition, the avionics battery, servo board and RC receiver also need to be modelled for the fault diagnostics to be effective in preventing the occurrence of an uncontrolled landing that could potentially result in catastrophic damage to the Meraka Modular UAV.

3.4 Fault Modelling and Identification

The process of fault modelling and identification can begin now that the various subsystem components of the UAV have been characterised. This starts off with the selection of system critical components and is followed by an overview of the pertinent modes of operation. The mechanisms behind the identification of the modes of operation are then defined before presenting a blueprint for constructing the declarative model used in the fault diagnostic engine.

3.4.1 System Critical Components

System critical components are selected in order to limit the complexity of developing the declarative model to be used for diagnostics. Critical components are characterised as having a Severity Category A as shown in Figure 3.33 with the red outlines and include:

- Rudders and Elevators
- RC Receiver
- Backup Battery
- Servo Board

Given that actuator FDIR is an important aspect of this research, more detailed models are constructed for actuators where as for the other components the declarative models are abstracted.

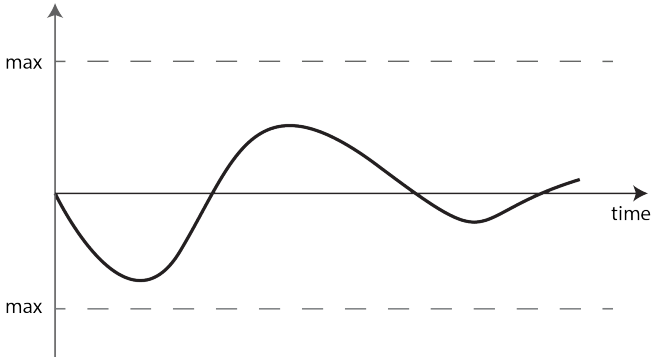
This results in the following revised list of the components of interest, which includes the remaining less critical components characterised as having a Severity Category B, shown with blue outlines in Figure 3.33:

- abstracted models:
  - Backup and Avionics Batteries
  - Servo Board and RC receiver
- detailed models:
  - Rudders and Elevators
  - Ailerons and Flaperons

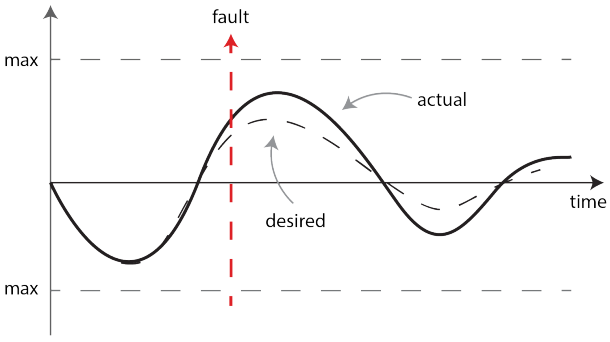
It's important to realise that this revised list has components that are functionally related, this is an important aspect to model especially for system level reasoning since an actuator may get stuck because it is not receiving power because the servo board is malfunctioning due to a battery fault. Reasoning in this way wouldn't be possible had these functional relationships been omitted.

3.4.2 Modes of Operation

We now investigate the modes of operation of the detailed actuator models. The modes of operation of a component define the configuration that determines how it behaves in that instance in time which is also a characteristic of a hybrid system [14]. In the case of an aircraft control surface, it is expected that it will either be in a nominal, stuck or floating mode of operation [50].



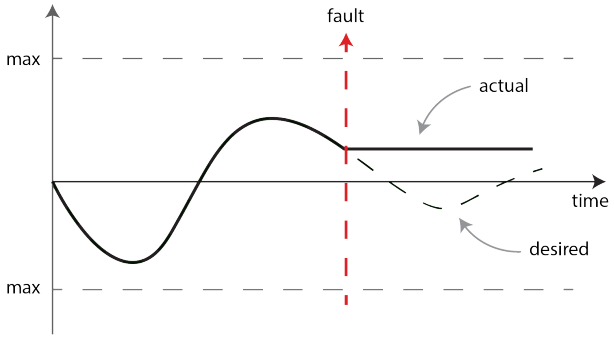
(a) *Nominal mode, notice how the commanded and actual deflections overlap.*



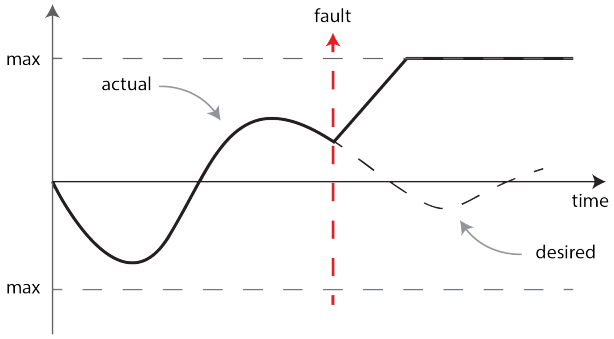
(b) *Partial loss of effectiveness failure, notice how the actual deflection deviates after the fault occurs.*

Figure 3.34: *Actuator nominal modes of operation.*

Figures 3.34a and 3.34b show a control surface's desired versus actual deflections in the ideal case where the two deflections match and in the unideal case where the actuator partially looses effectiveness while still trying to track the commanded control input. Deflection tracking and the average deflection positions will make it easy to identify these modes.



(a) *Stuck failure, notice how the actual deflection remains constant after the fault occurs.*



(b) *Hardover failure, notice how the actual deflection goes to max and remains constant after the fault occurs.*

Figure 3.35: *Actuator stuck modes of operation.*

Figures 3.35a and 3.35b show a control surface's desired versus actual deflections in the unideal case where the actuator is stuck at a random position and at the maximum deflection point respectively. Using the first derivative of the actual deflection will make it easy to identify this mode.

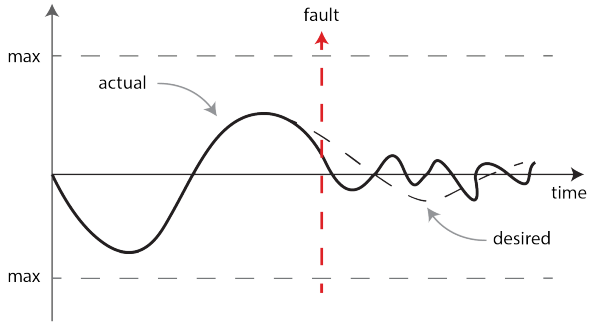


Figure 3.36: *Actuator floating mode of operation, notice how the actual deflection becomes random after the fault occurs.*

Figure 3.36 shows a control surface's desired versus actual deflections in the unideal case where the actuator is floating and no longer responds to commanded inputs. Using the cross correlation between the actual and desired deflections will make it easy to identify this mode.

3.4.3 Identification of Modes

We now define the mechanisms behind the identification of modes starting with the input reconstruction and ending off with stochastic signal processing. In the case of the aircraft's actuators, the most direct way is to reconstruct and compare the actual and desired deflections to determine the mode of operation that the actuator is currently in.

Hybrid Diagnosis

One of the key issues identified in the diagnosis task is limited observability. The number of sensors that allow the system to be observed are limited on the Meraka Modular UAV. AIRA in the same way as the Kalman Filter, is used to estimate certain properties of the system which might not be directly observable. We use AIRA to estimate the deflection angles of the control surfaces onboard the Meraka Modular UAV since they don't have position sensors. We do this by 'reconstructing the input' that would have given rise to the control surface's actual deflection angle. In the ideal case, the actual and desired deflection angles for the control surfaces are similar when everything is working. When something breaks however, the control surfaces don't necessarily respond to the control inputs anymore and it's possible to see this based on how the UAV behaves during flight.

Practical systems exhibit phenomena which can be observed using sensors i.e. acceleration in a car. In order to become aware of the effects of these phenomena, devices or tools suitable for observing these phenomena in a quantifiable manner must be used or developed. Fault detection methods must be selected and used in conjunction with sensors to detect the occurrence of specific desirable or undesirable phenomena which is a step beyond the data collection sensors perform. The Kalman Filter can be 'tuned' to become sensitive to the occurrence of specific phenomena i.e. a stuck fault [38]. However in order to say definitively that there is a stuck fault a decision must be actively made to either dismiss or accept an identified occurrence as a definitive indicator of a fault. This is what a Gaussian Bayes Classifier can be used for in conjunction with the Kalman Filter by deciding how similar an identified occurrence is to the actual known stuck fault. We use HyDE along with AIRA and a few signal processing methods in order to perform actuator FDI. HyDE in this case is the intelligence that reasons about the state of the entire system or component in order to identify faults based on observations reported by any number and combination of sensors and fault detection methods sensitive to the occurrence of specific phenomena.

Design Process

Figure 3.37: *AIRA operational model, highlighting a system's cause-effect input-output relationship.*

AIRA gives the control engineer considerable flexibility in its application. The only hard limit is that the state space representation of the system must have no invariant zeros for exact reconstruction or no zeros on the unit circle for approximate reconstruction. Little if any literature exists on how to maximise the effectiveness of AIRA and for the purposes of this research, the application of the algorithm is based on insights gained during its investigation. In the case of embedded systems with limited computational resources, AIRA performs best when applied to SISO systems. As a rule of thumb, represented by the applied design constraint in Figure 3.37, the best state space representation of the system being observed is one where there are no unobservable components and or one where there is a direct causal mathematical relationship between the input, states and output. The rationale being, if a controller can be designed to produce a desired response then a direct observable causal relationship between the input and output response exists barring the

presence of significant noise. In using the control loops, the accuracy of the algorithm can be increased significantly given that a well defined (quantifiable relationship between the input and output has been identified) state space representation is being used to produce the desired response. It is also the case that conventional aircraft utilise control systems to ensure safe flight. In the case of the UAV's aircraft model, the state space matrices were inspected in order to identify inputs and outputs that most readily satisfy the above design constraint and the systems were sampled at 50 Hz. When integrating a feedback control system into the UAV, AIRA can be reapplied to determine the most suitable state space representation for input reconstruction.

The results of the investigation done in [38] concerning the use of passive fault detection methods onboard the UAV for actuator FDI actively suggest that discerning between the left and right actuators is a difficult task for the FDD method to perform given the limited observability of the control surfaces and that a limited subset of the complete longitudinal and lateral flight dynamics of the aircraft are modelled in [38] and in this research project. In light of this, an active decision is made in this research to focus on tracking the combined behaviour of both left and right actuators simultaneously and using only the left control surface inputs in order to limit the scope of the research and the complexity of the developed system. It will still be possible to identify faults despite not knowing whether it is the left or right control surface. Given that additional FDIR techniques will be integrated with the diagnostic system in future and this is a prototype system, provision is made through an expandable FDIR architecture for either an active fault tolerant control method that excites the individual control surfaces or the installation of additional sensors to enable the fault diagnostic system to discern between the left and right control surface deflections by changing the Fault Diagnostic System.

To determine the control surface deflections for the ailerons, rudders, elevators and flaperons using AIRA we perform the following steps iteratively:

1. Generate a state space representation of the system - done in Section 2.5.4.
2. Identify the outputs that correlate well to changes in inputs.
3. Generate a state space representation describing the output response to these inputs.
4. Account for the locations of the zeros in this this state space representation (inside, outside or on the unit circle).
5. Decide whether to continue with the identified state space representation or modify it for use by filtering the zero.
6. Generate an output response and use the chosen state space representation to verify input reconstruction.

Elevator Deflection

$$\Delta \dot{\boldsymbol{x}}_{\delta_E} = \begin{bmatrix} -0.1312 & -2.3293 & 0 & -9.8086 \\ -0.0263 & -5.0906 & 0.9509 & 0.0062 \\ 0 & -21.3720 & -6.8251 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \overline{v} \\ \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ -0.1614 \\ -12.3041 \\ 0 \end{bmatrix} \delta_E \tag{3.32}$$

$$q_{\delta_E} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \overline{v} \\ \alpha \\ q \\ \theta \end{bmatrix} \tag{3.33}$$

Selecting the pitch rate q_{δ_E} to observe the effect of the elevator input was inspired by the Pitch Rate Damper Longitudinal Controller. When inspecting the state space matrices (3.32, 3.33), it is clearly evident that a mathematical relationship exists between the input, the state q and the measured output q_{δ_E}. Applying MATLAB's $ss2tf$ function gives the following transfer function:

$$\frac{q_{\delta_E}(s)}{\delta_E(s)} = \frac{s(s+0.1181)(s+4.8233)}{(s+0.0520 \pm j0.3121)(s+5.9715 \pm j4.4235)} \tag{3.34}$$

Using MATLAB's *pzmap* function generated the following pole zero plots in the continuous and discrete domains shown in Figures 3.38 and 3.39 respectively.

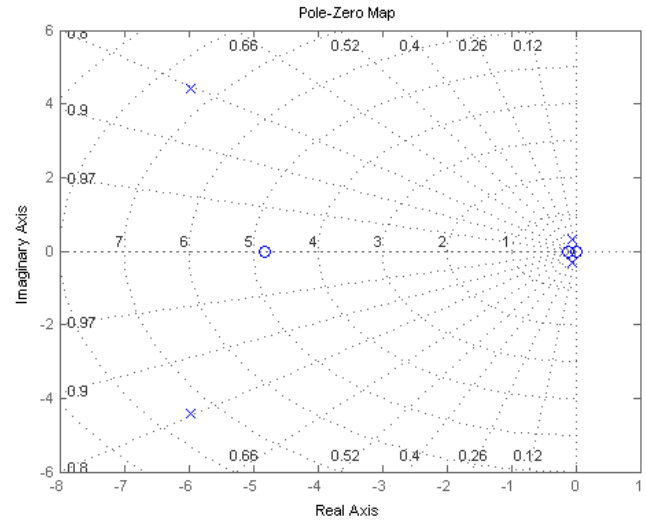


Figure 3.38: *Elevator Deflection:* $q_{\delta_E}(s)$ *response function, s-plane pole (+) zero (o) plot with a zero on the origin.*

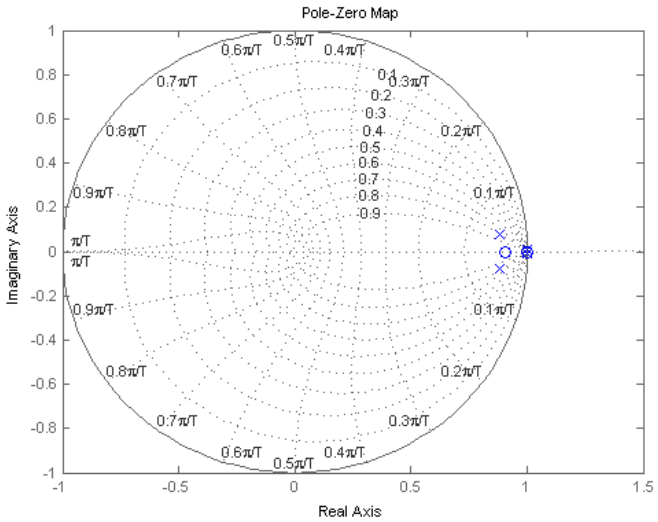


Figure 3.39: *Elevator Deflection:* $q_{\delta_E}(s)$ *response function, z-plane pole (+) zero (o) plot with a zero on the unit circle.*

Upon inspecting the transfer function, it is evident that there is a zero on the unit circle which can be seen in the pole zero plot shown in Figure 3.38 and that the state space representation is a minimum phase system as can be seen in Figure 3.39 because all the zeros are inside the unit circle. This implies that causal reconstruction is possible and that the zero on the unit circle will need to be filtered as there will likely be a persistent reconstruction error in the form of an offset as shown in Figure 3.40.

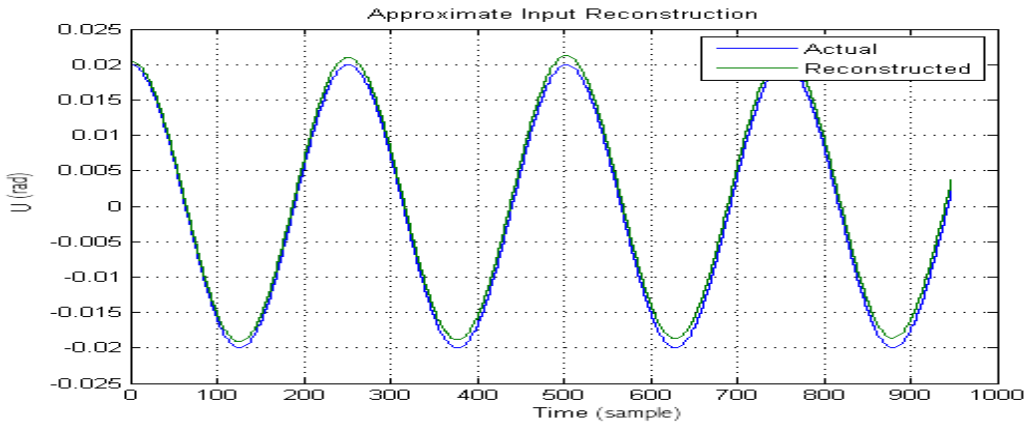


Figure 3.40: *Unfiltered Elevator Input Reconstruction, shown with a persistent reconstruction error.*

Filtering the zero from equation 3.34 yields the following equation:

$$\frac{\overline{q}_{\delta_E}(s)}{\delta_E(s)} = \frac{(s+0.1181)(s+4.8233)}{(s+0.0520 \pm j0.3121)(s+5.9715 \pm j4.4235)} \tag{3.35}$$

Using MATLAB's *pzmap* function generated the following pole zero plots in the continuous and discrete domains shown in Figures 3.41 and 3.42 respectively. Upon inspecting the transfer function, it is evident that there is no zero on the unit circle which can be seen in the pole zero plots shown in Figures 3.41 and 3.42. This implies that more accurate causal reconstruction is possible as shown in Figure 3.43.

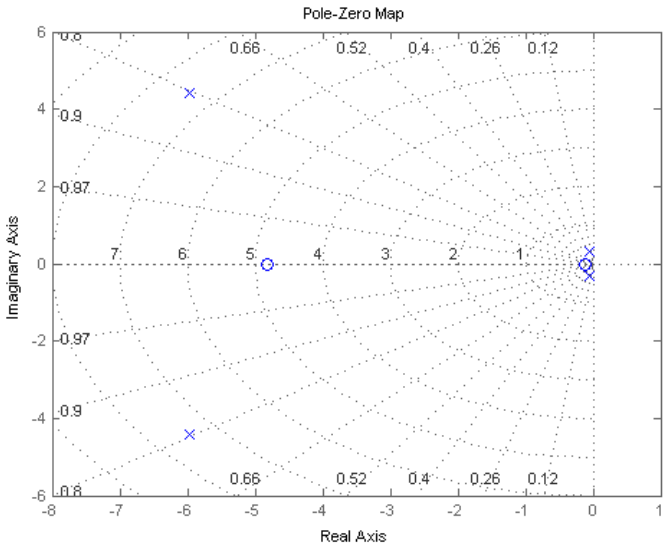


Figure 3.41: *Filtered Elevator Deflection: $\overline{q}_{\delta_E}(s)$ response function, s-plane pole (+) zero (o) plot, showing minimum phase zeros on the left hand plane.*

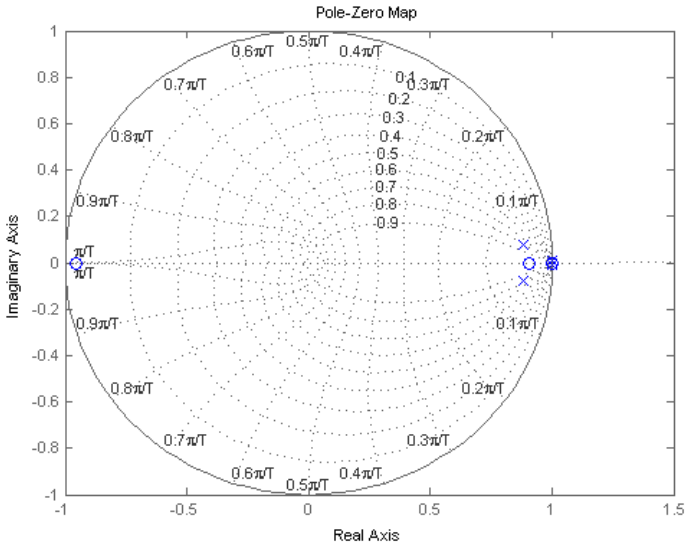


Figure 3.42: *Filtered Elevator Deflection:* $\overline{q}_{\delta_E}(s)$ *response function, z-plane pole (+) zero (o) plot, showing minimum phase zeros inside the unit circle.*

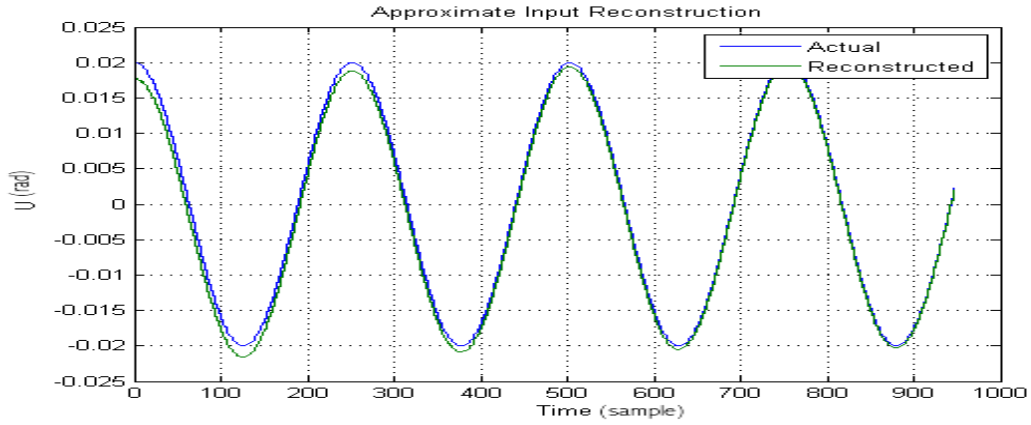


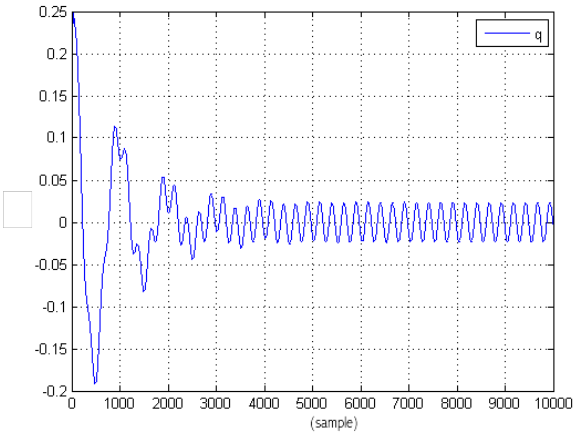
Figure 3.43: *Filtered Elevator Input Reconstruction (causal), shown without a persistent error.*

A Precautionary Note: The effect of filtering the zero from the elevator's transfer function merely makes the response less damped which amounts to increased reconstruction accuracy however after some time the filtered and unfiltered responses converge as can be seen in Figures 3.44a, 3.44b.

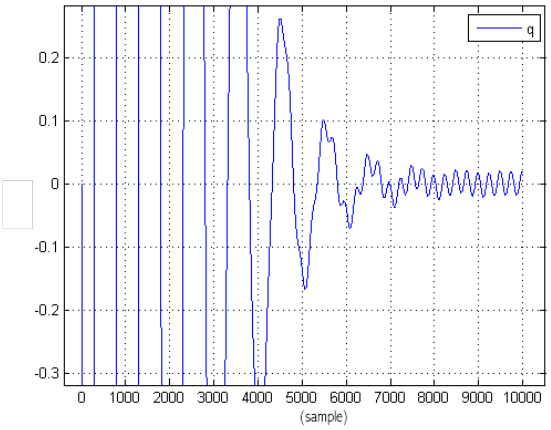
Rudder Deflection

$$\boldsymbol{\Delta\dot{x}}_{\delta_R} = \begin{bmatrix} -0.3567 & 0.0033 & -0.7765 & 0.3633 \\ -11.1230 & -7.1656 & 2.2419 & 0 \\ 19.2281 & -0.8859 & -1.1888 & 0 \\ 0 & 1 & -0.0169 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} + \begin{bmatrix} 0.0037 \\ 0.4545 \\ -6.7148 \\ 0 \end{bmatrix} \delta_R \tag{3.36}$$

$$\beta p \phi_{\delta_R} = \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} \tag{3.37}$$



(a) Unfiltered pitch rate response $q_{\delta_E}(s)$, more damped.



(b) Zero filtered pitch rate response $\overline{q}_{\delta_E}(s)$, less damped.

Figure 3.44: Pitch rate responses to elevator input.

Selecting the angle of side slip, roll angle and roll rate $\beta p \phi_{\delta_R}$ to observe the effect of the rudder input was inspired by the Dutch Roll Damper Lateral Controller. The outputs were combined to reduce the time taken for the unobservable components to die out during reconstruction. This also makes sense as it is easier to discern the effect of the rudder when considering the angle of side slip, roll angle and roll rates as the effect of the ailerons also comes into play. When inspecting the state space matrices (3.36, 3.37), it is clearly evident that a mathematical relationship exists between the input, the states β p, ϕ and the measured output $\beta p \phi_{\delta_R}$. Applying MATLAB's $ss2tf$ function gives the following transfer function:

$$\frac{\beta p \phi_{\delta_R}(s)}{\delta_R(s)} = \frac{(s + 1.7105 \pm j1.7671)(s - 22.1426)}{(s + 0.8186 \pm j4.0953)(s + 7.1537)(s - 0.0797)} \tag{3.38}$$

Using MATLAB's *pzmap* function generated the following pole zero plots in the continuous and discrete domains shown in Figures 3.45 and 3.46 respectively. Upon inspecting the transfer function, it is evident that there is no zero on the unit circle which can be seen in the pole zero plot shown in Figure 3.45 and that the state space representation is a minimum phase system as can be seen in Figure 3.46 because all the zeros are inside the unit circle. This implies that causal reconstruction is possible as shown in Figure 3.47.

Aileron Deflection

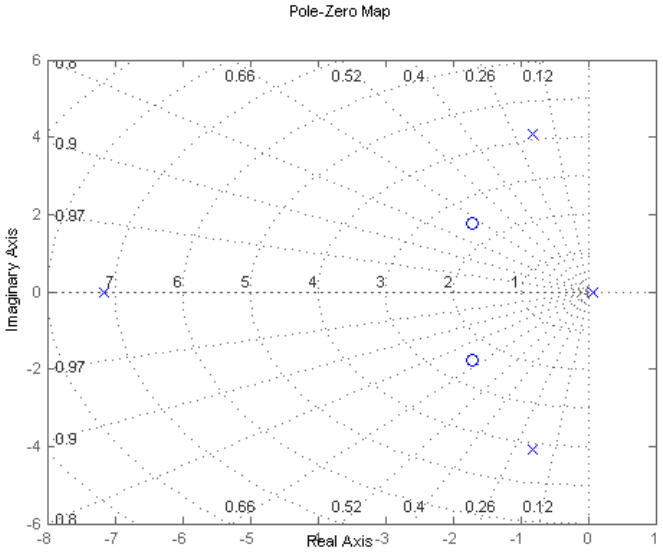


Figure 3.45: *Rudder Deflection:* $\beta p \phi_{\delta_R}(s)$ *response function, s-plane pole (+) zero (o) plot, showing minimum phase zeros on the left half plane.*

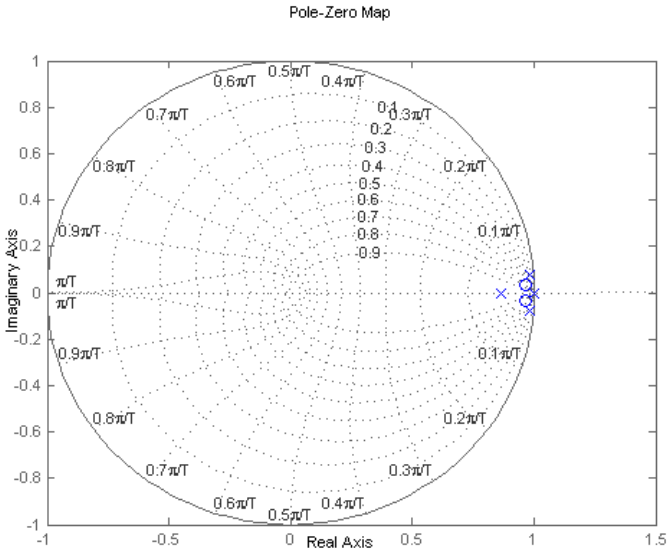


Figure 3.46: *Rudder Deflection:* $\beta p \phi_{\delta_R}(s)$ *response function, z-plane pole (+) zero (o) plot, showing minimum phase zeros inside the unit circle.*

$$\boldsymbol{\Delta \dot{x}}_{\delta_A} = \begin{bmatrix} -0.3567 & 0.0033 & -0.7765 & 0.3633 \\ -11.1230 & -7.1656 & 2.2419 & 0 \\ 19.2281 & -0.8859 & -1.1888 & 0 \\ 0 & 1 & -0.0169 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} + \begin{bmatrix} -3.3197e-04 \\ -25.4541 \\ 1.0778 \\ 0 \end{bmatrix} \delta_A \tag{3.39}$$

$$r_{\delta_A} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} \tag{3.40}$$

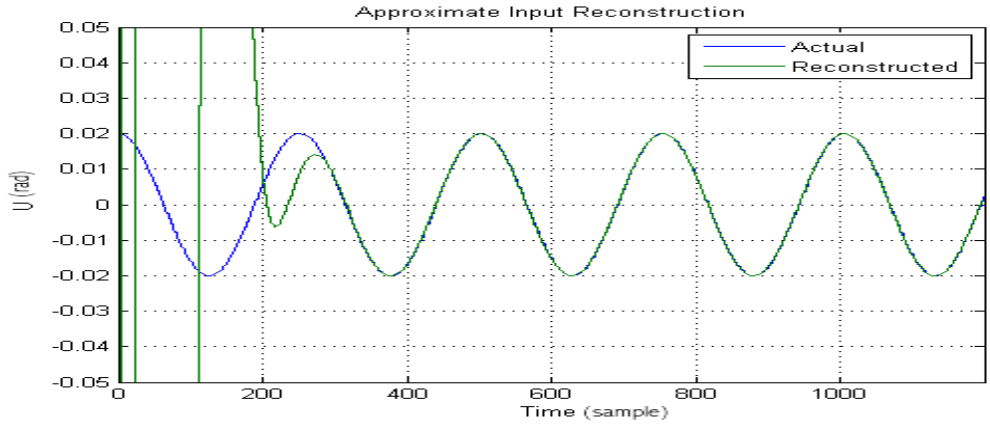


Figure 3.47: *Rudder Input Reconstruction (causal).*

Selecting the yaw rate r_{δ_A} to observe the effect of the aileron input was inspired by the Roll Angle Lateral Controller, the output selection is based on the time taken for the unobservable components to die out and the quality of the reconstruction. When inspecting the state space matrices, it is clearly evident that a mathematical relationship exists between the input, the state r and the measured output r_{δ_A}. Applying MATLAB's $ss2tf$ function gives the following transfer function:

$$\frac{r_{\delta_A}(s)}{\delta_A(s)} = \frac{(s+2.6691)(s-2.1588)(s+27.9282)}{(s-0.0797)(s+7.1537)(s+0.8186 \pm j4.0952)} \tag{3.41}$$

Using MATLAB's *pzmap* function generated the following pole zero plots in the continuous and discrete domains shown in Figures 3.48 and 3.49 respectively.

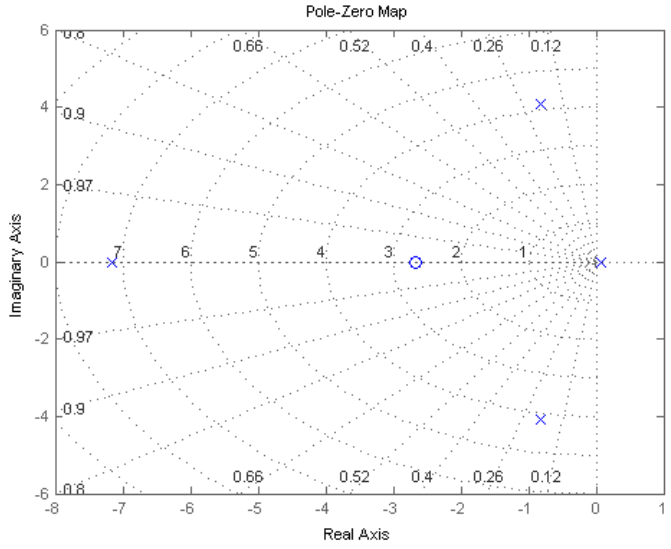


Figure 3.48: *Aileron Deflection: $r_{\delta_A}(s)$ response function, s-plane pole (+) zero (o) plot, showing mixed-phase zeros on both halves of the complex plane.*

Upon inspecting the transfer function, it is evident that there is no zero on the unit circle which can be seen in the pole zero plot shown in Figure 3.48 and that the state space representation is a system with both minimum and non minimum phase zeros as can be seen in Figure 3.49 because there are zeros inside and outside the unit circle. This implies that causal reconstruction is possible to a certain extent (the required input history r will need to be adjusted during implementation) and the latest accurately reconstructed inputs will be delayed somewhat as shown in Figure 3.50.

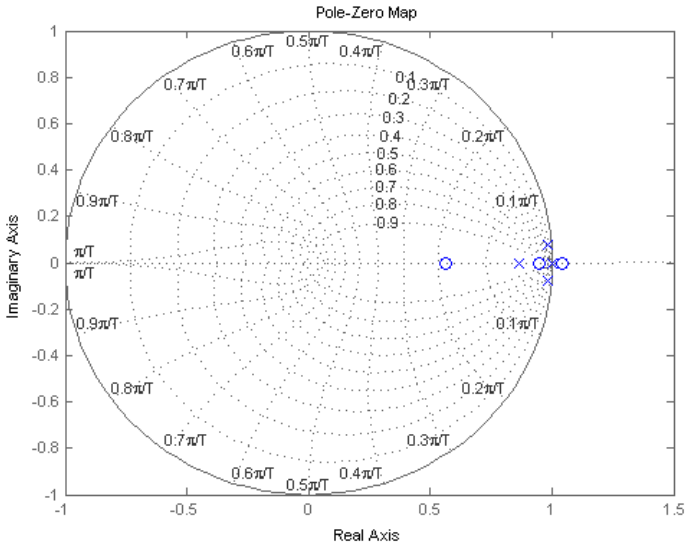


Figure 3.49: *Aileron Deflection:* $r_{\delta_A}(s)$ *response function, z-plane pole (+) zero (o) plot, showing mixed phase zeros inside and outside the unit circle.*

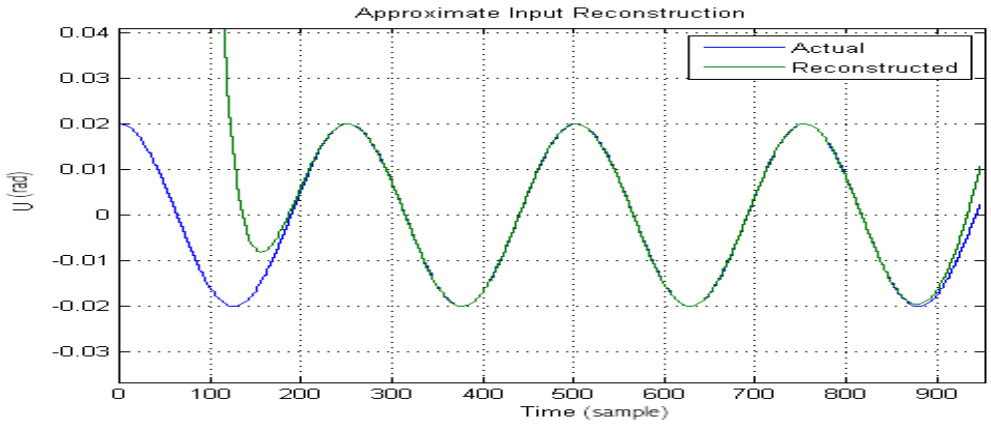


Figure 3.50: *Aileron Input Reconstruction (mixed), representing a mixed phase system with growing unobservable input components shown by the deviation between inputs.*

Flaperon Deflection

$$\boldsymbol{\Delta\dot{x}}_{\delta_F} = \begin{bmatrix} -0.3567 & 0.0033 & -0.7765 & 0.3633 \\ -11.1230 & -7.1656 & 2.2419 & 0 \\ 19.2281 & -0.8859 & -1.1888 & 0 \\ 0 & 1 & -0.0169 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} + \begin{bmatrix} 3.4598e-04 \\ -17.9489 \\ -0.6575 \\ 0 \end{bmatrix} \delta_F \tag{3.42}$$

$$r_{\delta_F} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} \tag{3.43}$$

Selecting the yaw rate r_{δ_F} to observe the effect of the flaperon input was inspired by the Roll Angle Lateral Controller, the output was selected based on the time taken for the unobservable components to die out and the quality of the reconstruction. When inspecting the state space matrices (3.42, 3.43), it is clearly evident that a mathematical relationship exists between the input, the state r and the measured output r_{δ_F}. Applying MATLAB's $ss2tf$ function gives the following transfer function:

$$\frac{r_{\delta_R}(s)}{\delta_R(s)} = \frac{(s+3.2386)(s-3.7116)(s-16.2006)}{(s-0.0797)(s+7.1537)(s+0.8186 \pm j4.0952)} \tag{3.44}$$

Using MATLAB's *pzmap* function generated the following pole zero plots in the continuous and discrete domains shown in Figures 3.51 and 3.52 respectively.

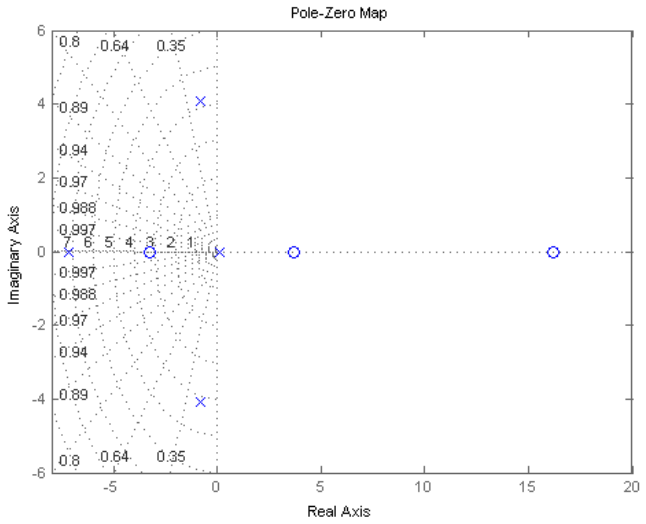


Figure 3.51: *Flaperon Deflection: $r_{\delta_R}(s)$ response function, s-plane pole (+) zero (o) plot, showing mixed-phase zeros on both halves complex plane.*

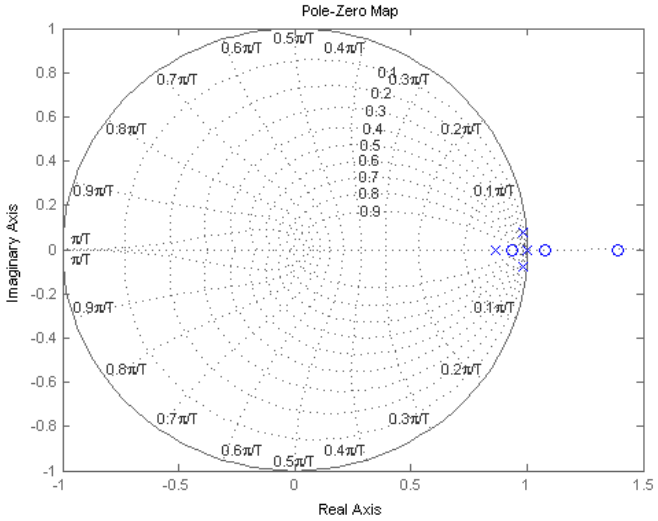


Figure 3.52: *Flaperon Deflection: $r_{\delta_R}(s)$ response function, z-plane pole (+) zero (o) plot, showing mixed phase zeros inside and outside the unit circle.*

Upon inspecting the transfer function, it is evident that there is no zero on the unit circle which can be seen in the pole zero plot shown in Figure 3.51 and that the state space representation is a system with both minimum and non minimum phase zeros as can be seen in Figure 3.52 because there zeros are inside and outside the unit circle. This implies that causal reconstruction is possible to a certain extent and the latest accurately reconstructed inputs will be delayed somewhat as shown in Figure 3.53. In the case of the flaperons, the input state matrices are different for the left and right input and the same process is applied in determining the state space representation needed for input reconstruction.

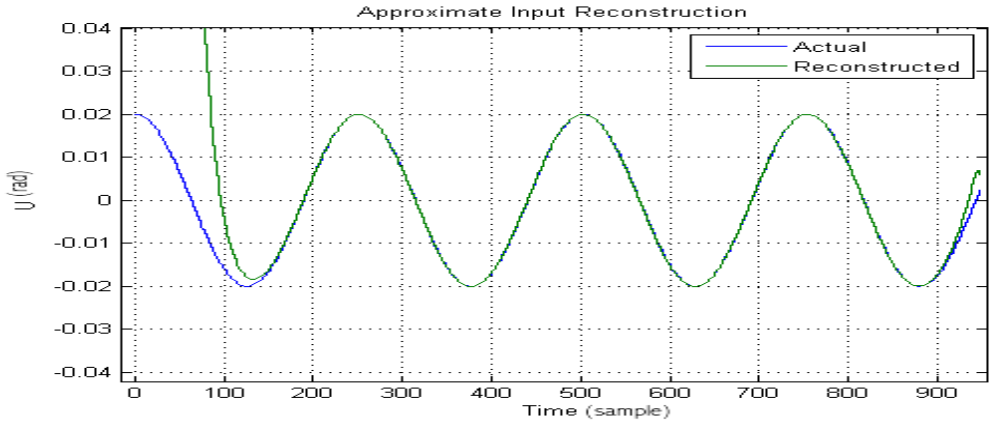


Figure 3.53: *Flaperon Input Reconstruction (causal), representing a mixed phase system with growing unobservable input components shown by the deviation between inputs.*

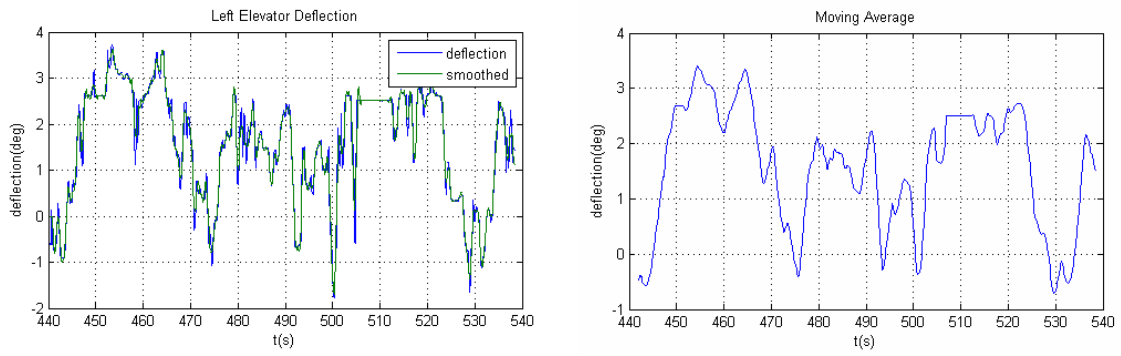
Signal Processing

We can now extract properties from the estimated control surface deflections having designed estimators based on AIRA. The pertinent modes of operation for the aircraft's actuators are analysed in Section 3.4.2. A few signal properties were identified that make it possible to mathematically distinguish between the different modes of operation given the actual and desired control surface deflections:

1. deflection average - to determine what the average deflection values are
2. first derivative - to determine when the deflection is constant
3. cross correlation - to quantify the difference between the deflections

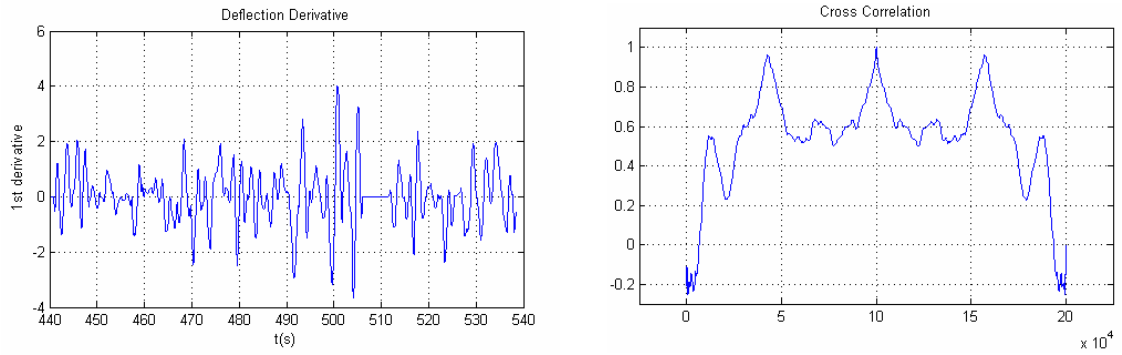
Once the actual and desired control surface deflections are available, signal processing functions in MATLAB can be used to determine the required properties from the deflection signals. Figures 3.54 and 3.55 show the result of signal processing that has been applied to an elevator's reconstructed control input deflections in order to determine these properties using the *sgolay*, *xcorr*, *tsmovavg* MATLAB functions.

The process of extracting these properties begins by first smoothing the reconstructed control surface deflection given that noise is inherent in practical systems. Once this is done, the moving average is determined based on the desired window length. This can be an additional step that is used to further filter the signal. After this is done, the first derivate is calculated followed by the cross correlation between the actual and desired control surface deflections. The cross correlation is normalised to the $[1, -1]$ range, to allow for the standardisation of the decision process, and only the maximum value is used during the decision process. Similar signals will have values closer to 1 and dissimilar signals will have values closer to -1.



(a) *Smoothed input* (b) *Moving average*

Figure 3.54: *Smoothing applied to elevator input.*



(a) *First derivative* (b) *Cross correlation*

Figure 3.55: *Statistical properties of elevator input.*

3.4.4 Fault Detection in FDIR System

The following table outlines properties that can be used by the Diagnostic System to uniquely identify the mode of operation of each actuator.

Mode	Observations	Constraints	Rate
Nominal	1 correlation 2 derivative 3 average	correlation = 0.9 average = deflection	0.9998
Partial	1 correlation 2 derivative 3 average	correlation = 0.7	2.655e-5
Stuck	1 correlation 2 derivative 3 average	correlation < 0.4 derivative = 0 average != deflection	1.810e-5
Hardover	1 correlation 2 derivative 3 average	correlation < 0.4 derivative = 0 abs(average) = max	3.258e-5
Floating	1 correlation 2 derivative 3 average	correlation < 0.6 derivative > 0	4.345e-5

Table 3.6: *Signal properties and failure rates used to identify actuator modes of operation in HyDE.*

Actuator modes of operation

Table 3.6 lists the different actuator modes of operation along with the relevant symptom effects listed as constraints. The Rate column is the probability that each mode of operation will occur based on the Reliability Analysis already outlined in Section 3.3. It is calculated using the component failure rate for an actuator and the failure mode distributions for each actuator failure mode. In HyDE modelling, this column represents the transition probabilities that would be used to model transitions between modes of operation. The observations column represents all the numbered signals or observations (single values) that are used in the decision process at any instant in time.

The nominal mode of operation is characterised by the actual and desired deflections being equal. The cross correlation value of 0.9 was chosen as a design decision to allow for slight variation between the two deflection signals. In addition, it is expected that the average deflection values will be relatively similar to contrast this mode with the partial loss of effectiveness mode. HyDE is able to add sensor noise tolerances to observations reported making it easy to take this variation due to noise into account.

While in the partial loss of effectiveness mode, the control surface deflection varies and there is no clear limit to how it will vary so comparing the average deflections would not be suitable. However, since the control surface still attempts to track the signal and since the cross correlation doesn't vary significantly depending on the signal's magnitude, the cross correlation can be used. A slightly lower value 0.7 is chosen to allow for a greater degree of dissimilarity between the actual and desired control surface deflections.

The stuck modes of operation are characterised by having a derivative of zero $derivative = 0$ to reflect that the deflection is constant. The control surface deflection doesn't change in response to the control input anymore. The cross correlation of 0.4 between the actual and desired deflections is also used to address the case where both the input and the output deflections are relatively constant. It is desirable that the actual and desired control surface deflections aren't similar in order to reduce the occurrence of false alarms. In addition, for the stuck mode in particular, the average deflection value is used to reflect that the desired and actual deflection values must not be similar $average\ != deflection$. For the hardover fault, it is desirable that the average value be at maximum deflection.

The floating mode of operation is characterised by a control surface deflection that changes randomly. To reflect this we use a cross correlation of 0.6 to reflect the fact that it may vary but in a similar way to the desired deflection. A higher cross correlation is also used in order to differentiate between the stuck mode of operation and the floating mode operation. A deflection that varies is characterised by a derivative that is not equal to zero. We enforce this as a constraint in order for the floating mode of operation to not be confused with either the partial loss of effectiveness or stuck modes of operation. This could result in a case where HyDE reports that the control surface is in the partial mode and then momentarily moves into the floating mode and visa versa.

A high fidelity software in the loop simulation environment is used to test the integrated fault diagnostic system. This ensures that it will function as expected in the field when used by the Meraka Modular UAV and Ground Station Operator to maintain stable flight or request diagnoses. Following is an outline for developing the Meraka's declarative HyDE models and ways the fault diagnostic system is used to achieve FDIR functionality.

3.5 Modelling Onboard Subsystems

Following is a description of the process of incrementally translating the Meraka's subsystem components into a diagnosable declarative model.

3.5.1 Using the GME Tool

In Section 3.4.1 the subsystem components were characterised and system critical components were selected in order to limit the complexity of constructing the declarative model used for fault diagnostics. Figure 3.56, shown with Category A and B components highlighted, is the result of this process and with the FDIR technology integrated, it's possible to begin the process of generating the declarative models using the GME Tool.

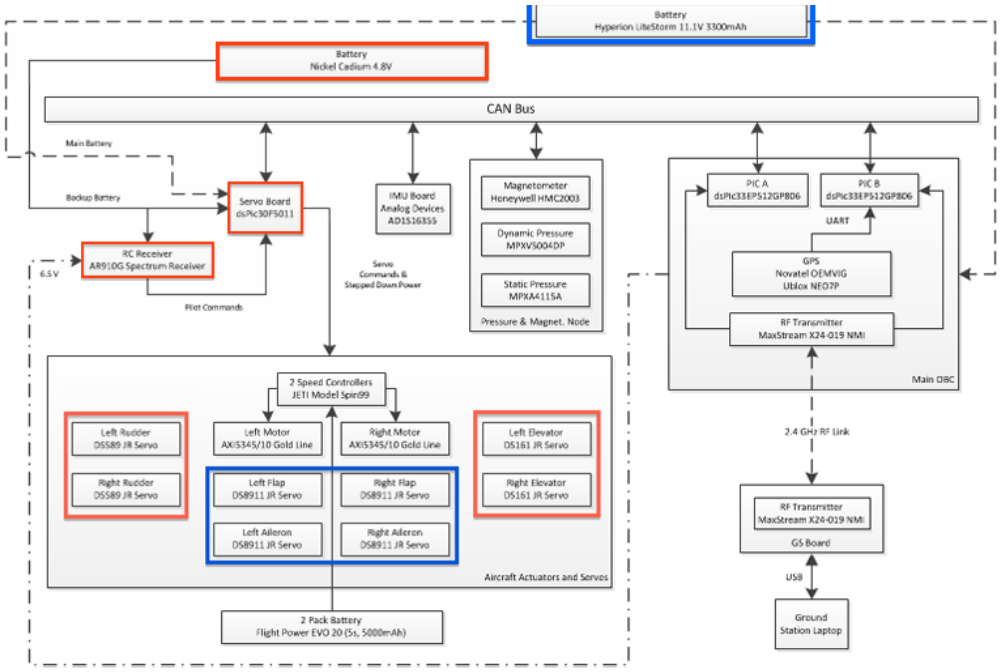


Figure 3.56: *Characterised Subsystem Model, shown with the Category A and B components highlighted in red and blue respectively.*

The first step is to create a list of all the subsystem components to be modelled along with their component numbers, desired HyDE prefixes and modes of operation as shown in Table 3.7. The Simulink Model in Section 4.2.2 and the characterised subsystem model shown in Figure 3.56 are used to define the component groupings at the system and subsystem levels. The second step requires the creation of system level components in HyDE, according to the system level component names in Table 3.7, along with their port variables and connections as shown in Figure 3.57. The modelling process for HyDE models created using the GME tool is iterative in a top-down fashion. Interconnections are initially modelled at the system level with subsystem components encapsulating the necessary component level detail according to the desired level of abstraction.

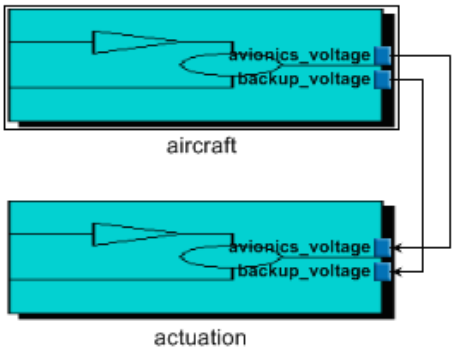


Figure 3.57: *System level declarative component model of the UAV in GME.*

Subsystem Components			
#	Component	HyDE Prefix (s)	Mode
Aircraft			
1.1 – 1.2	Batteries	aircraft.avionics_battery, aircraft.backup_battery	nominal
			degraded
			discharged
			unknown
Actuation			
2.1	RC Receiver	actuation.rc_receiver	on
			off
			out_range
			system_failure
2.2	Servo Board	actuation.servo_board	on
			off
			miscalibrated
			unknown
2.4 – 2.11	Actuators	actuation.left_aileron, actuation.right_aileron, actuation.left_flaperon, actuation.right_flaperon, actuation.left_elevator, actuation.right_elevator, actuation.left_rudder, actuation.right_rudder	off
			nominal
			partial
			stuck
			hardover
			floating
			unknown

Table 3.7: *Characterised subsystem components to be modelled.*

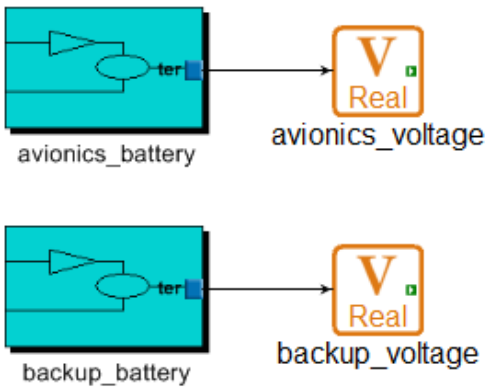


Figure 3.58: *Aircraft system level component model in GME.*

Figures 3.58 and 3.59 show the aircraft and actuation system level declarative component models in GME. Both the avionics and backup batteries skeleton component models have been modelled as well as the RC Receiver, Servo Board and actuator skeleton component models.

The last iterative step in the modelling process consists of adding the component level detail to each component based on its properties as outlined in the Reliability Analysis in Appendix E. Before actual modelling in GME can begin, the mode or location, constraints, transitions and transition probabilities must be defined and extracted from the Reliability Analysis and tabulated. Once this is done, the process of adding the component level detail to each component becomes trivial.

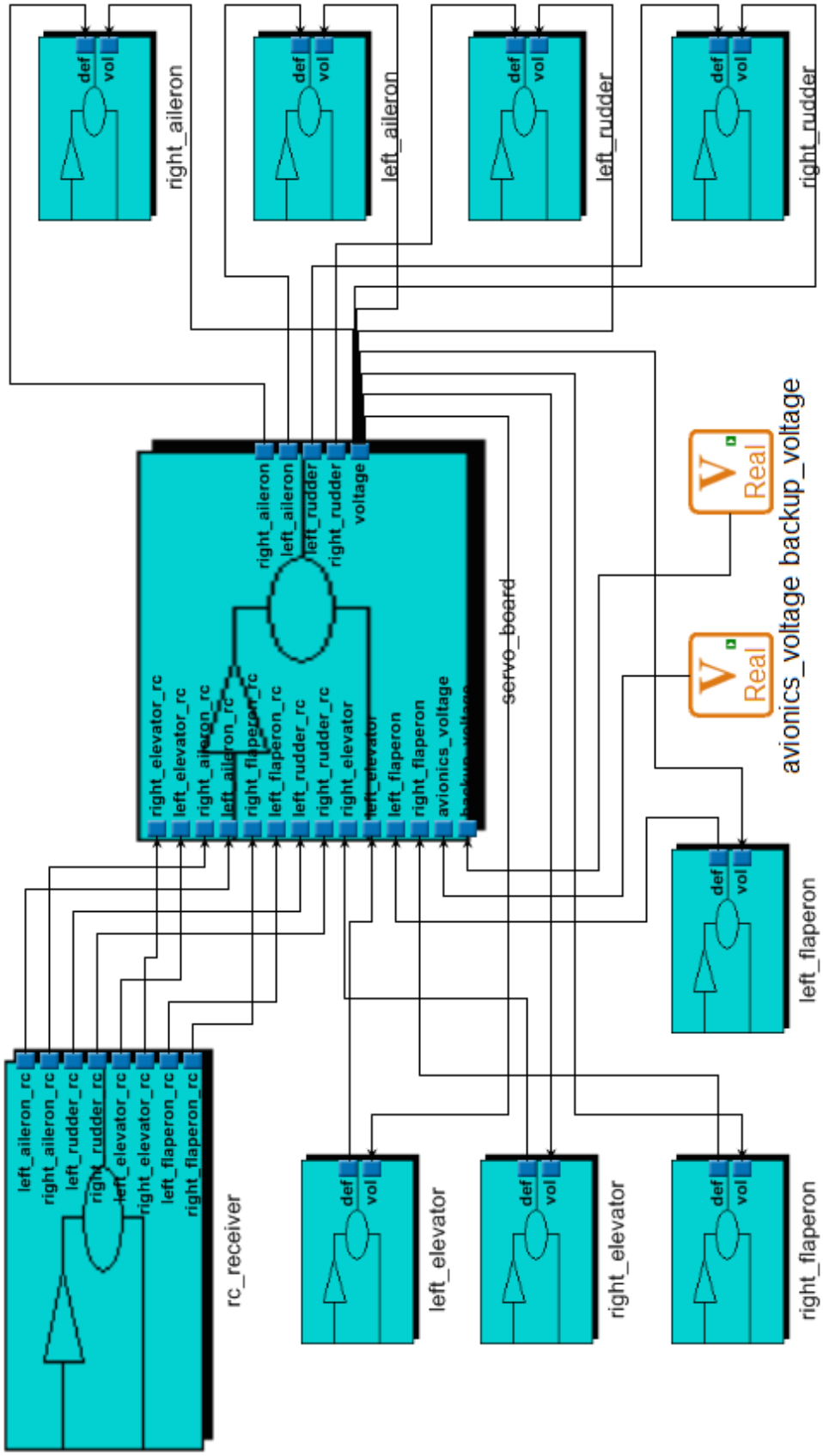


Figure 3.59: *Actuation system level component model in GME, shown with the Actuators, RC Receiver and Servo Board.*

3.5.2 Modelling the Batteries

The aircraft subsystem components are the first components to be modelled followed by the actuation subsystem components. Table 3.8 lists the component level properties for the aircraft's batteries. In it, one can find the mode, constraints, transitions and transition rates for both the avionics battery and the backup battery. The transition rates make it possible for HyDE to rank the modes in descending order of prior probability when there are multiple consistent candidates. The transition guards are the constraints in italics and enable HyDE models to actively or autonomously switch between locations or modes of operation, modes with guards are highlighted.

Avionics Battery				
#	Mode	Constraints	Transitions	Rate
N	nominal	battery_voltage = terminal_voltage	E, I, U	0.99998
E	degraded	(battery_voltage-terminal_voltage)/battery_voltage <= 0.7	N, E, U	1.245e-5
I	discharged	terminal_voltage = 0	E, U	3.511e-6
U	unknown			
Backup Battery				
#	Mode	Constraints	Transitions	Rate
N	nominal	battery_voltage = terminal_voltage	E, I, U	0.99998
E	degraded	(battery_voltage-terminal_voltage)/battery_voltage <= 0.7	N, E, U	8.330e-6
I	discharged	terminal_voltage = 0	E, U	3.240e-6
U	unknown			

Table 3.8: *Component level properties for the aircraft's batteries, with highlighted constant, input and port variables.*

Figure 3.60 is the result of translating the listed component properties in Table 3.8 into a detailed model that can be simulated. In this model, the battery voltage is defined as a constant and the terminal voltage is the observed voltage across the battery's terminals.

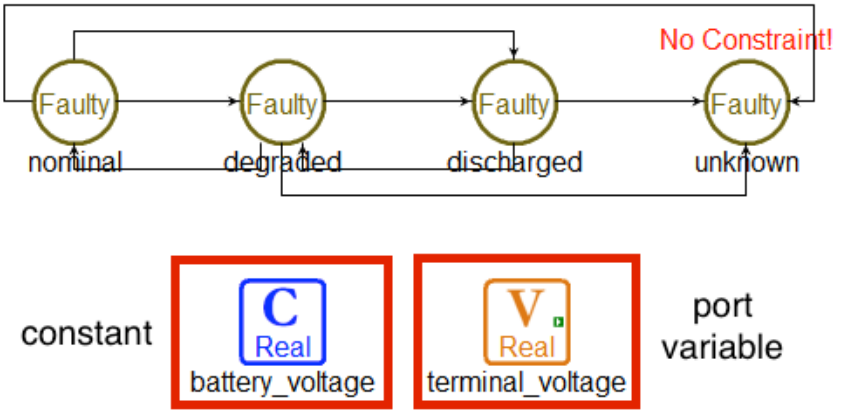


Figure 3.60: *Subsystem level component model of a battery, with highlighted constant and port variable.*

3.5.3 Modelling the RC Receiver

#	Mode	Constraints	Transitions	Rate
rc_receiver				
O	on	*voltage = voltage_rating* Tx = Rx frame_loss <= 20	F, R, U	0.9993
F	off	*voltage = 0* Tx = 0	O	0.9993
R	out_range	voltage = voltage_rating frame_loss > 20 && frame_loss <= 100	O, F, U	3.733e-4
U	system_failure			3.1801e-4
rc_receiver.power				
P	primary_power	*obc_voltage >= backup_voltage*	S	1
S	secondary_power	*backup_voltage > obc_voltage*	P	1

Rc receiver modes of operation

Table 3.9: *Component level properties for the RC receiver.*

Table 3.9 lists the component level properties for the RC receiver. The constraints contain the variables Tx and Rx which are used to represent actuator commands transmitted to the Servo Board and the mixed actuator commands received from the Servo Board. The RC receiver has the ability to receive power either directly from the OBC or from the backup battery. This functionality is modelled using an internal *power* component within the RC receiver as shown in Table 3.9 and allows the RC receiver to switch its power source autonomously.

Figure 3.61 is the result of translating the listed component properties in Table 3.9 into a detailed model that can be simulated. In this model, the $frame\_loss$ is a measure of the quality of the transmission between the controller and the receiver and is reported by the RC receiver onboard the UAV. Figure 3.62 is the subsystem level component model of the RC receiver showing the power component with its internal $primary\_power$ and $secondary\_power$ locations shown in Figure 3.63.

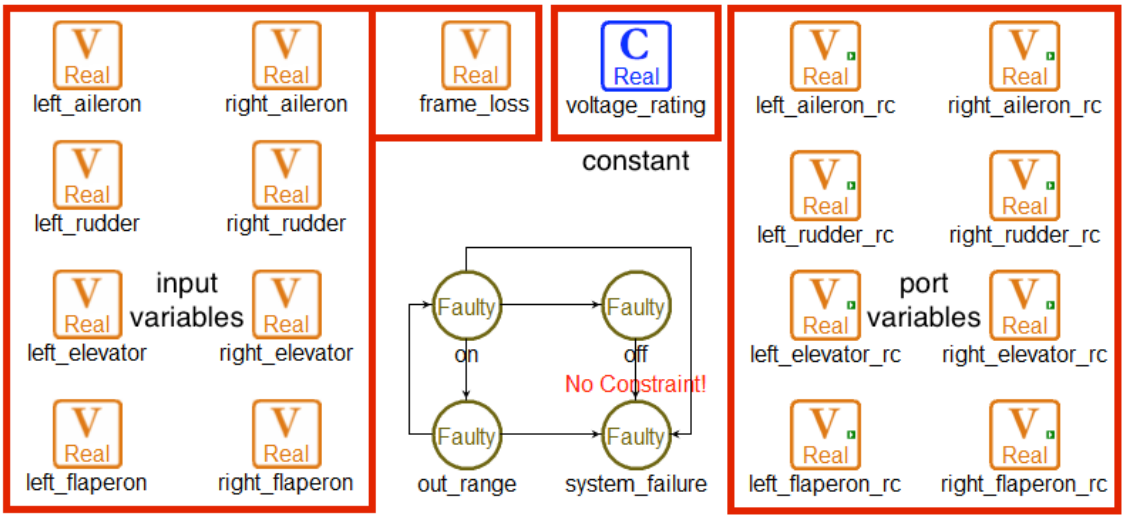


Figure 3.61: *Subsystem level component model of the RC receiver's locations with highlighted input and output port variables.*

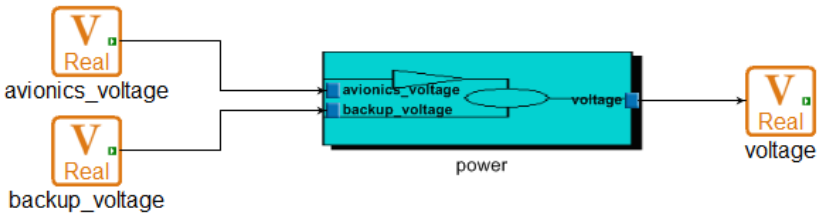


Figure 3.62: *Subsystem level component model of the RC receiver showing the power component.*

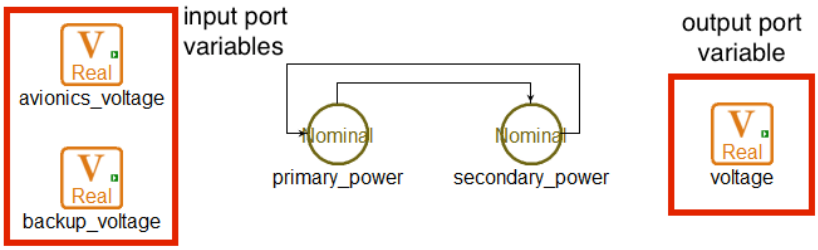


Figure 3.63: *The RC receiver's component model of the power subcomponent with highlighted input and output port variables.*

3.5.4 Modelling the Servo Board

The Servo Board's functionality resembles that of the RC receiver in many ways as can be seen in Table 3.10 and Figure 3.64. Both subsystem components have the *power* subcomponent and make use of the Tx and Rx variables.

#	Mode	Constraints	Transitions	Rate
servo_board				
O	on	*voltage = voltage_rating* Tx = Rx	F, M, U	0.99998
F	off	*voltage = 0* Tx = 0	O	2.76e-6
M	miscalibrated	voltage = voltage_rating (Tx-Rx)/Tx <= 0.9	O, F, U	9.24e-6
U	unknown			
servo_board.commands				
P	primary_commands	*ap_arm*	S	1
S	secondary_commands	*ap_arm*	P	1
servo_board.power				
P	primary_power	*avionics_voltage >= backup_voltage*	S	1
S	secondary_power	*backup_voltage > avionics_voltage*	P	1

Table 3.10: *Component level properties for the servo board.*

Servo board modes of operation

The Servo Board also has the ability to receive actuator commands from either the OBC or the RC Receiver in the same way that it can switch between power sources autonomously and this behaviour is modelled using the subcomponents shown in Figures 3.65 and 3.67. The servo board's commands subcomponent performs the task of switching between receiving commands from the RC receiver and the OBC based on the transition guard ap_arm shown in Figure 3.66 which signals whether the auto pilot has been enabled.

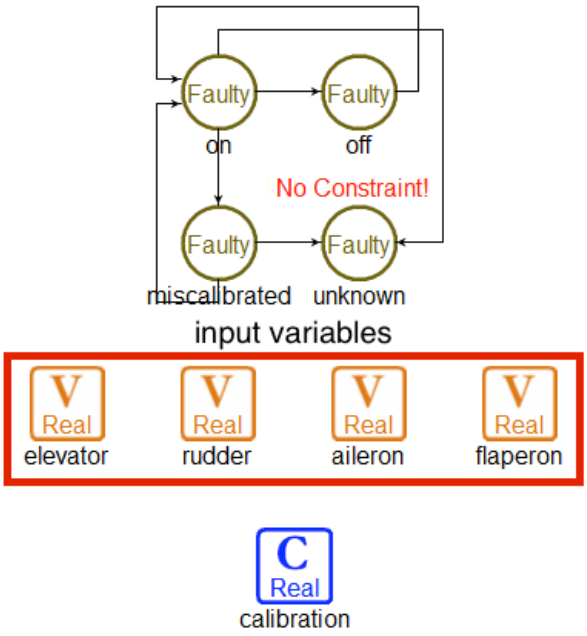


Figure 3.64: *Subsystem level component model of the servo board's locations, with highlighted input variables.*

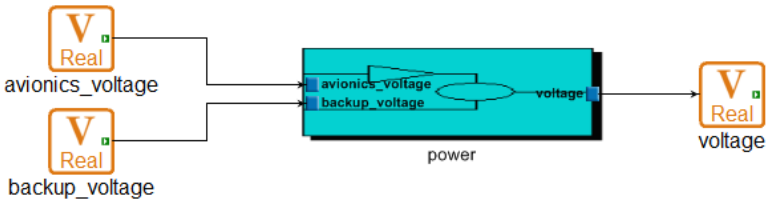


Figure 3.65: *Subsystem level component model of the servo board showing the power component.*

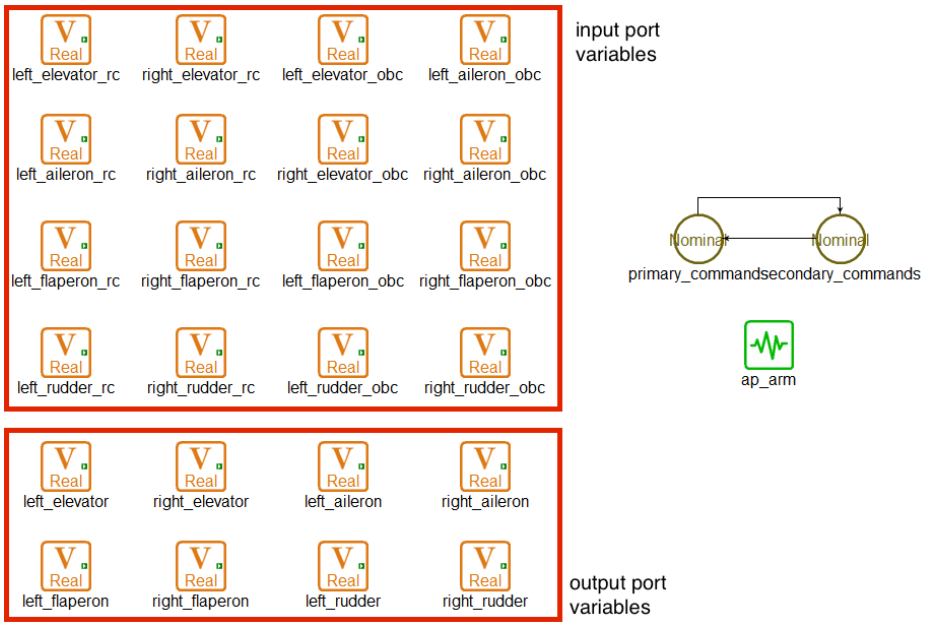


Figure 3.66: *The servo board's component model of the commands subcomponent, with highlighted input and output port variables.*

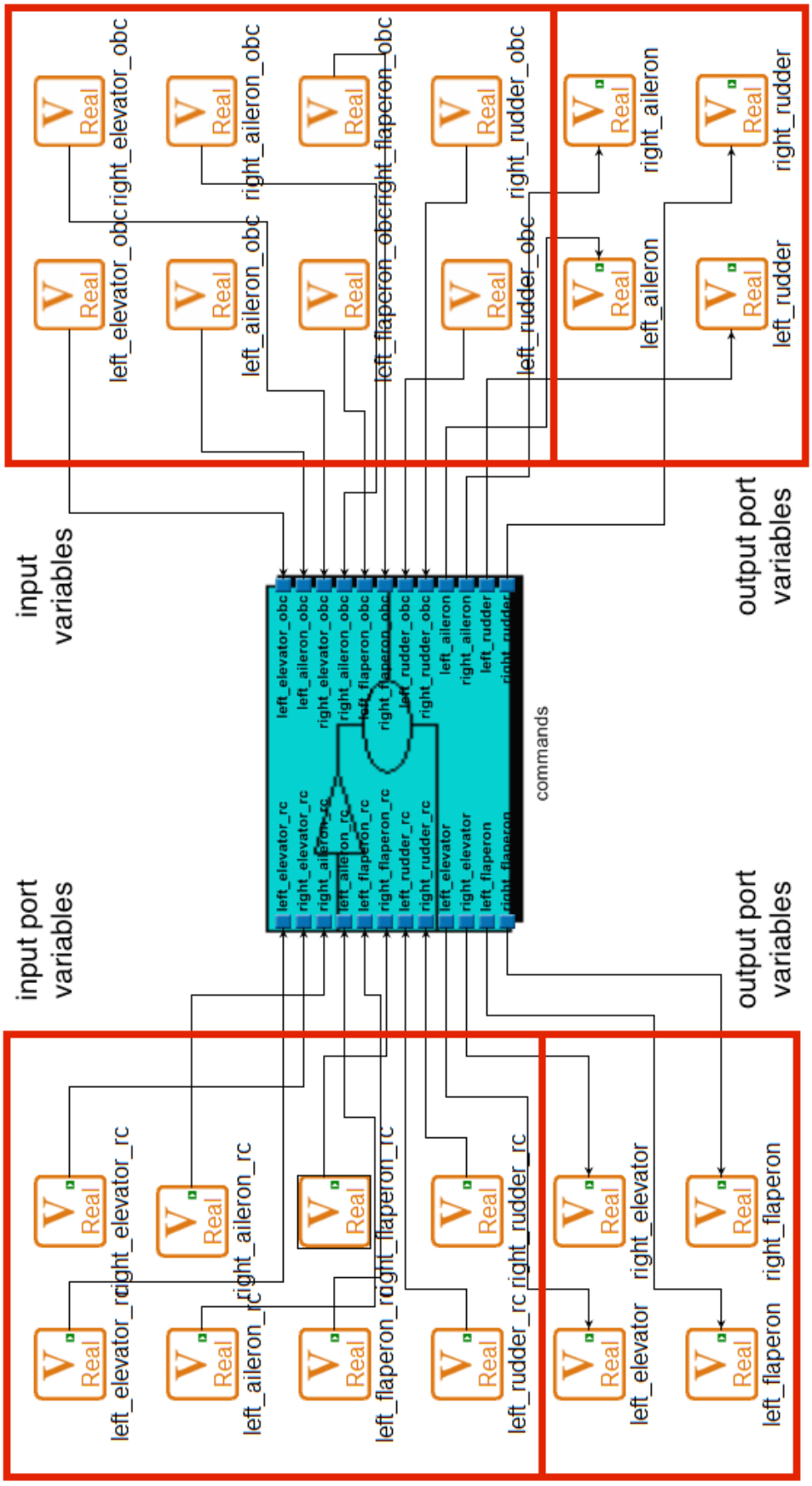


Figure 3.67: *Subsystem level component model of the servo board showing the commands component, with highlighted input and output port variables.*

3.5.5 Modelling the Actuators

#	Mode	Constraints	Transitions	Rate
N	nominal	*voltage = voltage_rating* correlation = 0.9 average = deflection	O, P, S, H, F, U	0.9998
P	partial	voltage = voltage_rating correlation = 0.7	N, S, H, F, U	2.655e-5
S	stuck	voltage = voltage_rating correlation < 0.4 derivative = 0 average != deflection	N, P, H, F, U	1.810e-5
H	hardover	voltage = voltage_rating correlation < 0.4 derivative = 0 average = max	N, P, S, F, U	3.258e-5
F	floating	voltage = voltage_rating correlation < 0.6 derivative > 0	N, P, S, H, U	4.345e-5
O	off	*voltage = 0*	N	
U	unknown			

Table 3.11: Component level properties for the actuators.

actuator modes of operation

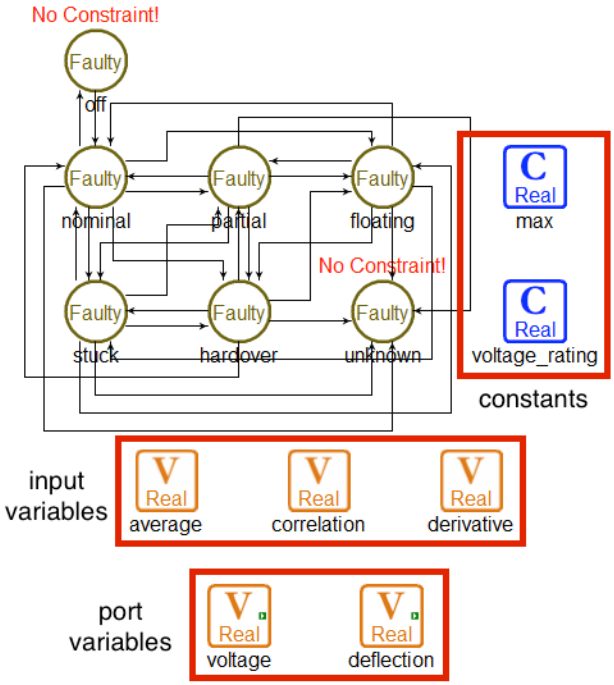


Figure 3.68: *Subsystem level component model of the actuator, with highlighted constants, input and output port variables.*

The component level properties, listed in Table 3.11, for the actuators are based on the work done in Section 3.4 since their diagnoses depend on observations reported using signals generated by smart adapters. The result of modelling these component level properties for the actuators is shown in Figure 3.68. The *voltage_rating* and *max* constants represent the rated DC voltage and max deflections for the actuator. The last step in the modelling process consists of testing the model using scenarios and exporting it into an XML file that can be read by the diagnostic program. We now move onto describing how HyDE enables autonomous fault diagnostics.

3.6 Overview

An introduction and overview to HyDE is given in this chapter. The Hybrid Diagnosis Engine (HyDE) is used to give the Meraka Modular UAV the ability to diagnose faults and detect changes in its subsystem components. HyDE is investigated using a concrete example of a simplified heating system within the context of a HyDE model, harness and scenario. The reasoning process HyDE applies to diagnosis is also discussed.

Input reconstruction using the Approximate Input Reconstruction Algorithm (AIRA), based on the parity space approach, is investigated using examples outlining its behaviour under certain conditions. In this research, AIRA is used to estimate inputs that are compared with commanded values. The investigation of AIRA results in a novel way of implementing the algorithm to create SISO based smart adapters, mentioned in Chapter 4, designed based on the aircraft's inner control loops. It is discovered that accurate input reconstruction is possible using the UAV's linearised aircraft model with outputs generated with modulated sinusoidal inputs of any size and poor reconstruction results for constant input signals.

Subsystem characterisation is done through a simplified Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA) and culminates in the creation of a Hazards Analysis Matrix (HAM) that is divided into three end event categories. This is done as a means of guiding the fault modelling and identification process that follows.

The fault modelling and identification process requires a clearly defined description of the functional relationships that exist between subsystem components. This is followed by the selection of system critical components and is followed by an overview of the pertinent modes of operation for each selected component. The mechanisms behind the identification of the modes of operation are then defined before presenting a blueprint for constructing the declarative model in the fault diagnostic engine using the Generic Modelling Environment Tool (GME).

The process of incrementally translating the UAV's subsystem components into a diagnosable declarative model is given. This process is based on the use of the characterised subsystem model and the Generic Modelling Environment (GME) Tool. Selected models that have an observable impact on Category A and B end event effects are modelled and these include the batteries, RC receiver, servo board and actuators (control surfaces).

Chapter 4

System Integration

An expandable FDIR architecture is developed to facilitate the addition of more FDIR methods in an effort to allow the fault diagnostic system's functionality to be enhanced easily. The architecture is integrated based on the design of a simplified Integrated Vehicle Health Management (IVHM) System. This system consists of a Diagnostic Agent built into the ESL's custom Ground Station, the Meraka Modular UAV, and a Diagnostic System that runs off a Model B+ Raspberry Pi.

4.1 Expandable FDIR Architecture

Building a model based diagnostic system requires that the control engineer have a good idea of how to design and build models suited to the diagnosis task. In many cases, the diagnosis problem is not clearly defined and having the flexibility to experiment with different kinds of models and modelling strategies is paramount. This is where having an expandable FDIR architecture becomes an advantage. Following is an overview of the IVHM architecture and corresponding system configuration as well as the SIL configuration used to develop the integrated system.

4.1.1 IVHM Architecture

In the simplified IVHM architecture Figure 4.1, described in Section 1.3, the Diagnostic System runs off a Model B+ Raspberry Pi which can be connected to the Meraka Modular UAV to communicate with the OBC either directly using its PiCAN Interface Board or via ethernet. The Ground Station software is being used as the foundation for the Diagnostic Agent along with a few modifications.

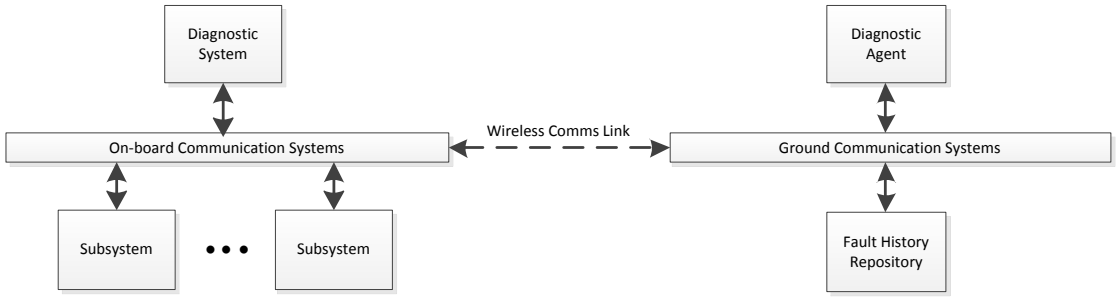


Figure 4.1: *Simplified IVHM Architecture, showing the complementary Diagnostic Agent and System.*

With the Diagnostic System installed onboard the UAV, it can not only perform the diagnosis task in real time, it can also communicate with the other systems onboard as part of a fault mitigation strategy. The Diagnostic System utilises the onboard communications system to communicate with the OBC and the existing wireless comms link to communicate with and receive commands from the Diagnostic Agent which complements the functionality of the Diagnostic Agent on ground.

4.1.2 Integrated System

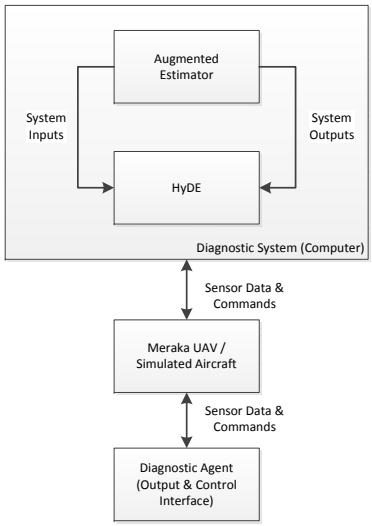


Figure 4.2: *IVHM System Configuration, showing the flow of information between systems.*

For HyDE to perform diagnosis, the system shown in Figure 4.2 is developed to realise the IVHM System Architecture shown in Figure 4.1.The estimation system shown in Figure 4.2 is augmented in order to derive additional observations representing for example actuator deflections based on the available estimated system states and sensor measurements. The approach used to derive these additional measurements is based on the Approximate Input Reconstruction Algorithm as described in Chapter 3.

4.1.3 SIL Configuration

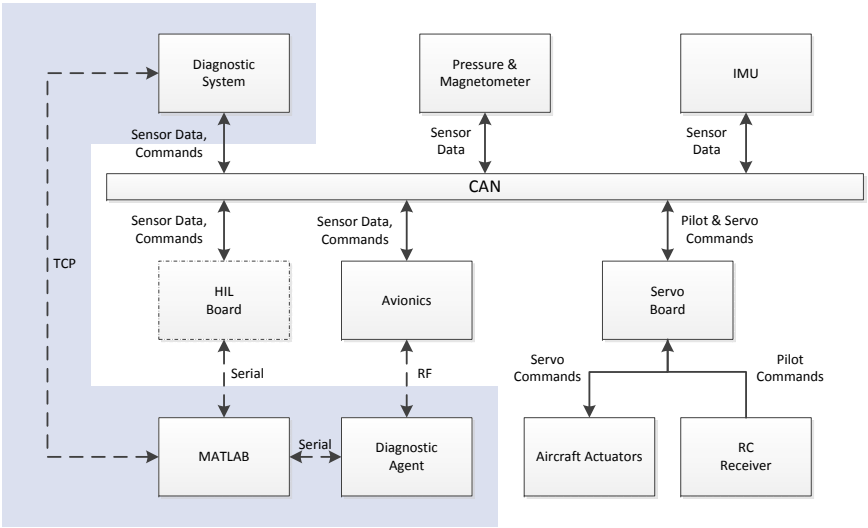


Figure 4.3: *HIL Configuration, showing the complete test-bed system highlighting SIL components.*

The HIL Configuration shown in Figure 4.3 above is the test-bed system used to develop the Diagnostic System and Diagnostic Agent using the Meraka UAV's onboard subsystems and HIL Board shown with a dashed outline. With the use of this configuration, it becomes possible to integrate and test the target system with the Raspberry PI connected. Before this process can begin, SIL based development and testing needs to be done by decoupling the components in the shaded region from the rest of the target system.

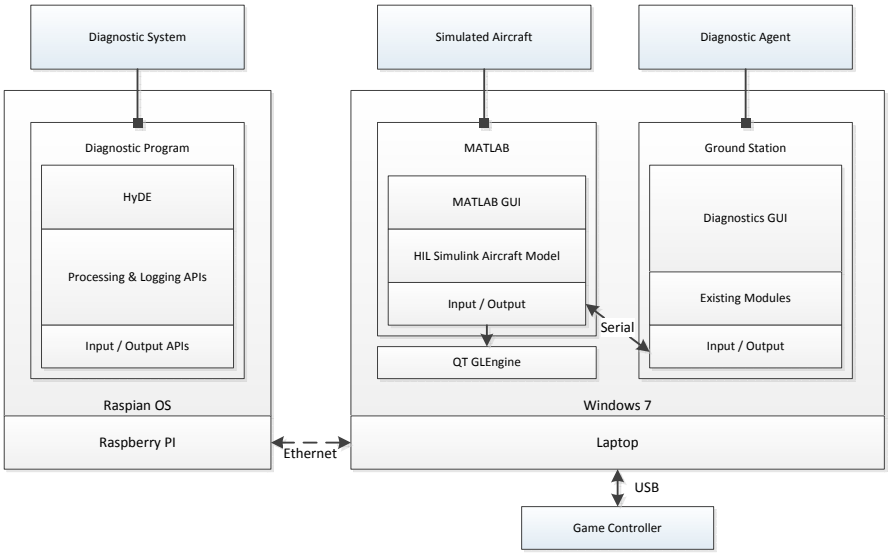


Figure 4.4: *SIL Configuration, highlighting the assembled technology for each system.*

SIL based development and testing requires assembling the SIL configuration shown in Figure 4.4 with the different systems, subsystem components and communications interfaces between them. Following is an overview of how these components are integrated within the context of the simplified IVHM System being assembled.

4.2 Simulation Environment

We now give an overview of the functionality that is enabled by the simulation environment within the context of the simplified IVHM System being assembled.

4.2.1 System Integration

MATLAB serves as the mediator between the Diagnostic Agent and Diagnostic System to facilitate the exchange of information into and out of the simulation environment. It does this through a *mex* s-function which is executed at every time step during simulation. It allows a serial connection to be established with the Ground Station and an ethernet connection to be established with the diagnostic program during SIL based development and testing.

4.2.2 Simulink Model

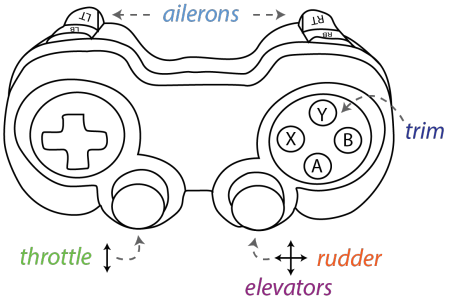


Figure 4.5: *Button allocation on the game controller.*

As part of the SIL setup, manual control of the Meraka UAV within the simulation environment is simulated using the game controller. The game controller's joysticks and controls are assigned

functions which correspond with the inputs needed by the simulated aircraft as shown in Figure 4.5. The control outputs from the game controller for the left and right actuator inputs are combined to ensure that the UAV remains balanced during flight.

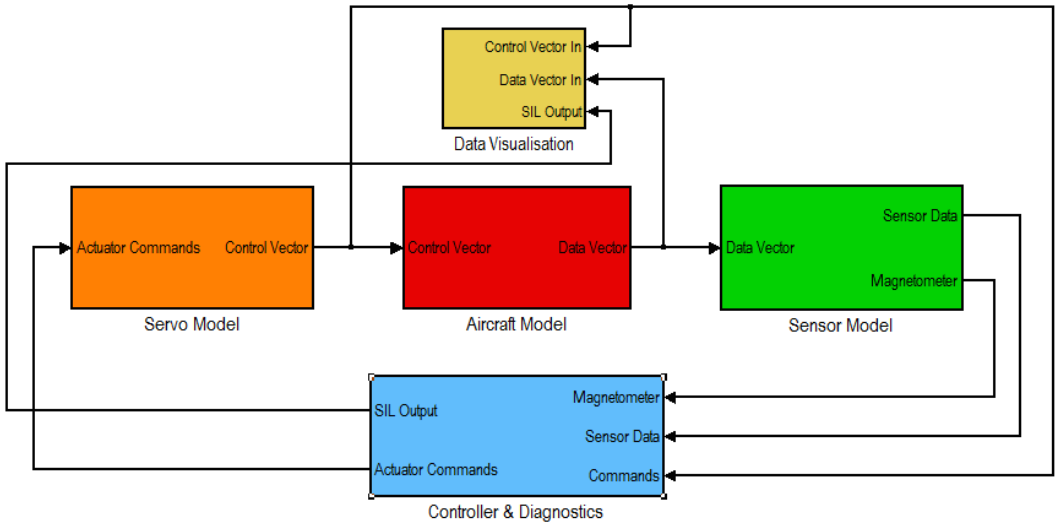


Figure 4.6: *Simulink Model of the Meraka Modular UAV.*

The Simulink model, shown in Figure 4.6, simulates the nonlinear aircraft dynamics and communicates with the diagnostic program and ground station using the *mex* s-function. The main components in the Simulink model are:

- *aircraft model* - a non-linear model of the Meraka Modular UAV
- *sensor model* - a sensor model which packages simulation values as sensor data
- *controller & diagnostics* - the control and diagnostics block which contains a modified HIL block and interface to the game controller
- *servo model* - models of the aircraft's actuators that simulate the responses of the servos and motors used on board
- *data visualisation* - the sim visualisation which is used for data logging and interfacing the Simulink model with the QTGLEngine

4.2.3 MATLAB GUI

A closer look at the SIL setup is shown in Figure 4.7 with a MATLAB GUI on the left and the QTGLEngine on the right. The MATLAB GUI is used to control the simulation, configure MATLAB and display sensor measurements generated using the Simulink model of the aircraft. These measurements are then used to verify the performance of the Diagnostic System. The simulated aircraft component grouping groups functions that configure the simulated aircraft. Callback functions are used to setup the Simulink model's parameters by loading constants and aircraft parameters and open the QTGL Engine to visualise the Meraka Modular UAV using the outputs from the non linear aircraft block. The simulation component grouping groups functions that adjust the aircraft simulation parameters. The data source for simulation can be toggled between flight data or the data from the non linear aircraft model. It is also possible to send commands to the Simulink model to start and stop the simulation. The measurements component grouping groups functions that make it possible to inspect the actuator deflections either for each actuator or all the actuators. Callback functions generate figures from data that has been logged during the operation of the simulation for the rudders, elevators, ailerons and flaps. Additional actuator specific measurements that are shown are the first derivative, correlation and average.

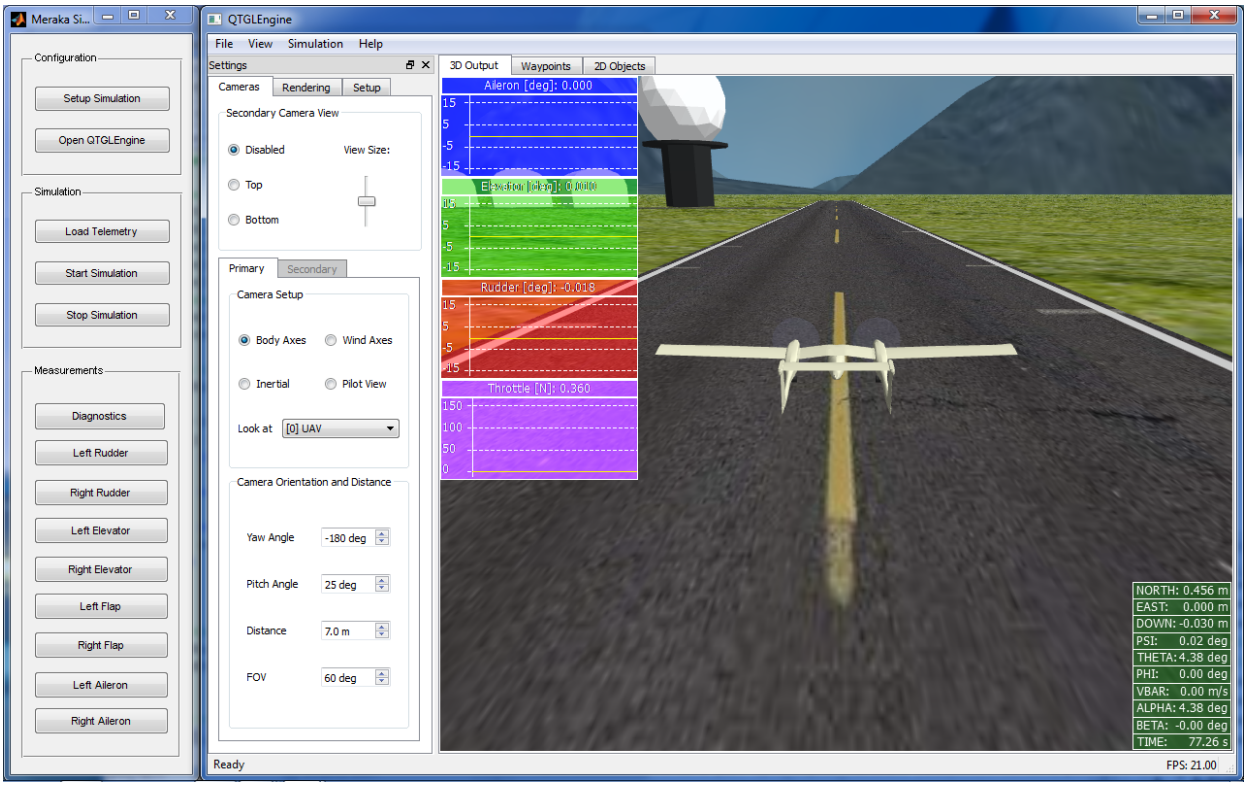


Figure 4.7: *Meraka Modular UAV simulation using MATLAB and QT.*

4.3 Diagnostic Agent

We now give an overview of the functionality that is enabled by the Diagnostic Agent within the context of the simplified IVHM System being assembled.

4.3.1 System Integration

The Diagnostic Agent or Ground Station has several options for connecting with other systems. These include serial and ethernet connections that are used to communicate with the UAV via an RF connection and a TCP server for data logging. One of the available serial com ports have been assigned for use with MATLAB and allow the Ground Station to connect to MATLAB through the *mex* s-function via an emulated serial com port. Any data that needs to be sent to the Diagnostic System via the UAV goes through MATLAB.

4.3.2 Existing Modules

Existing modules in the Ground Station's stack were adapted in order to incorporate the functionality needed by the Diagnostics GUI. Any callback functions that are triggered through the UI are translated into commands that can be sent to the Diagnostic System through the UAV. The functionality to also translate the data received from the Diagnostic Agent into information that can be visualised on the interface is also contained within these adapted modules.

4.3.3 Diagnostics GUI

The Diagnostic Panel, shown in Figure 4.8 with a red border, in the Ground Station was modified to reflect the added functionality delivered by HyDE. The main component groupings include the scenario, diagnostics, time, stage, system schematic and HyDE panels. The scenario panel is used to setup the sample time, sleep time, total time, faults, and sensor noise diagnostics parameters. These parameters are used by the diagnostic program when creating and managing the HyDE

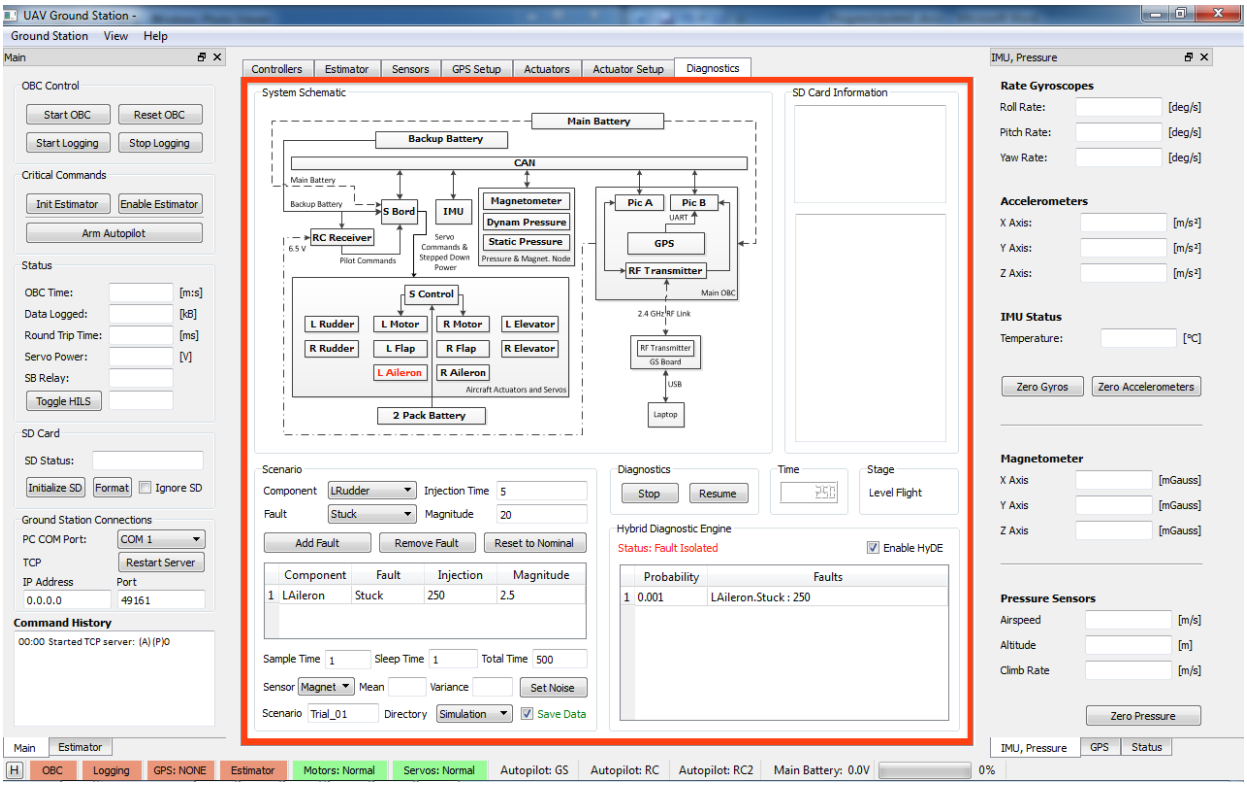


Figure 4.8: *Diagnostics Panel on the Ground Station.*

instance that runs on the Raspberry PI. The fault injection scheduler in the diagnostic program receives commands from the Diagnostic Agent and commands the OBC; data monitors or smart adapters in the augmented estimator to simulate the fault until commanded to remove it.

The diagnostic panel is used to start or stop, and pause or resume the execution of a scenario based on the scenario's execution parameters entered in the scenario panel. The time and stage panels are indicators received from the diagnostic program representing the execution time in seconds of the simulation and the status or stage of execution of the diagnostic program.

The HyDE panel allows the Ground Station operator to activate and deactivate the execution of the HyDE instance in the diagnostic program. The diagnoses are also listed in the form of a candidate set that specifies the state of each component which includes its name, location, prior probability at the requested time step. The system schematic is updated each time a diagnosis in the form of a candidate set is sent from the diagnostic system based on the mode or location of each component.

4.4 Diagnostic System

We now give an overview of the functionality that is enabled by the Diagnostic System within the context of the simplified IVHM System being assembled.

4.4.1 System Integration

The Model B+ Raspberry PI, shown in Figure 4.9, with Raspian OS is used as the computational platform that the diagnostic program runs off. It communicates with the Diagnostic Agent and the rest of the subsystems onboard the UAV using ethernet and has the option of also using the PiCAN expansion board to connect to the CAN (Controller Area Network). Sensor data and commands are fetched directly from the OBC instead of the individual components. Each component onboard the UAV reports its own status directly to the OBC. When the OBC receives data from each

component, it first transforms this data into a usable form before it sends it to the Ground Station. The Diagnostic System leverages access to the OBC to get the sensor measurements instead of repeating the same task that is performed by the OBC. Collecting observations in this way allows the Diagnostic System to focus on performing diagnoses instead of transforming data collected from onboard subsystems into usable form.

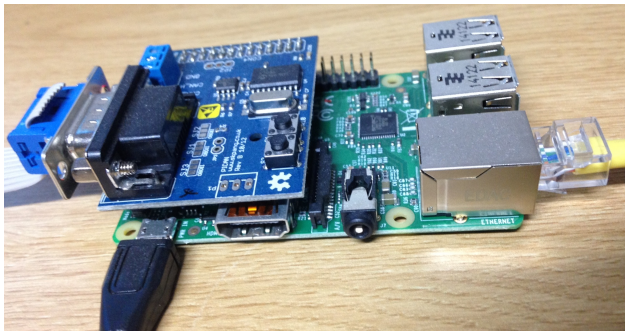


Figure 4.9: *Model B+ Raspberry PI with PiCAN Expansion Board.*

4.4.2 Processing and Logging

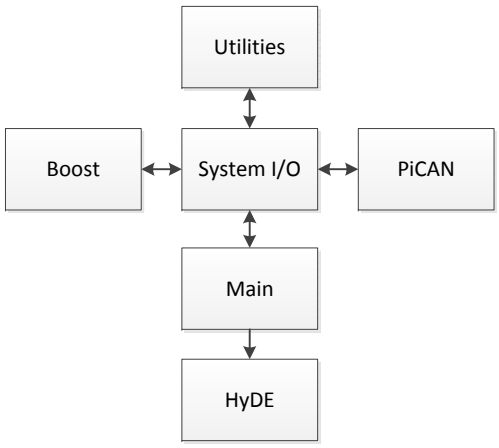


Figure 4.10: *Functional overview of the diagnostic program.*

Figure 4.10 shows a functional overview of the components in the diagnostic program. The *main* function in the diagnostic program responds to program requests from the Diagnostic Agent or UAV and activates the diagnostic program's subroutines accordingly. In addition to maintaining the execution time, the diagnostic program's processing and logging layer packages the reported diagnoses so that they can be transmitted to the Ground Station.

Additional components of interest within the Diagnostic Program are $System\ I/O$, $Boost$, $PiCAN$, $Utilities$ and $HyDE$ blocks. The $System\ I/O$ block serves the function of abstracting away the implementation level detail necessary for the Diagnostic Program to interact with the rest of the IVHM System on behalf of the *main* function. The $Boost$ and $PiCAN$ blocks enable the Diagnostic Program to communicate using either ethernet or the CAN bus. The $Utilities$ block contains additional logic that enables the Diagnostic Program to leverage access to the Boost Library in order to perform functions such as creating and synchronising threads. The $HyDE$ block deals directly with the HyDE instance that is instantiated and houses the logic necessary for the Augmented Estimator to perform its task as mediator.

The Augmented Estimator and the Fault Injection Scheduler, shown in Figure 4.11, are arguably the most important elements of the Diagnostic System's Diagnostic Program. For HyDE to function, it needs to be able to receive data in the form of observations that are meaningful for the diagnosis task at hand. This is the function that the Augmented Estimator provides by being the synchronised mediator between HyDE and the rest of the IVHM System. The Fault Injection Scheduler's function is to take over control from the OBC when needed in order to simulate the desired faults for the particular scenario being simulated. Once the simulated fault has reached the end of its lifetime, the Fault Injection Scheduler will stop simulating the fault and relinquish control over the relevant component by commanding the OBC to resume standard operational control over the component.

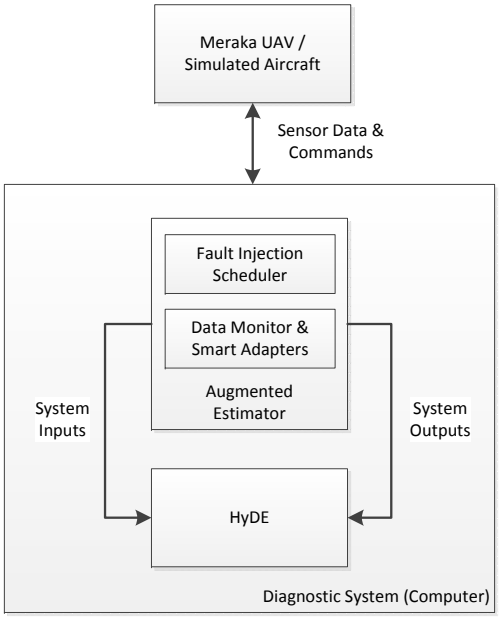


Figure 4.11: *Meraka Modular UAV HyDE Interface.*

Data monitors and smart adapters, shown in Figure 4.12, are used to prepare observations to be reported in HyDE and run in different threads of execution. They simulate sensor noise when commanded by the Fault Injection Scheduler and convert sensor data from the UAV into a usable form and use it to estimate or calculate additional data that can be reported as observations. These observations are reported to the HyDE instance in the next level of the diagnostic program's stack. Smart adapters can also be additional FDIR methods that produce diagnosis results that can be reported to HyDE in order to reason about the system. Smart adapters are used for signal processing using the incoming sensor data in order to simulate sensor noise and calculate the first derivative, correlation and actuator deflection averages after smoothing the reconstructed input.

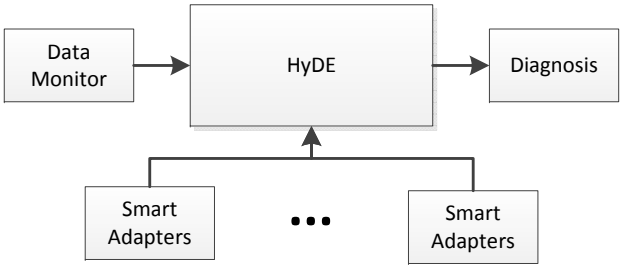


Figure 4.12: *HyDE Interface, shown with data monitor and smart adapters.*

Integration with the Diagnostic Program is done by using code generated with MATLAB's code generator to obviate the need for the integration of additional processing and logging APIs. The actuator deflections are determined by calculating the reconstructed inputs using an implementation of AIRA, shown in Algorithm 1, repurposed for real time execution coupled with a digital FIR low pass filter designed using MATLAB's *pwelch* and *dsp.LowpassFilter* functions.

Algorithm 1 *Real time approximate input reconstruction*

procedure REALTIMEAIRA
initialise:
    initialise *system* variables
    initialise *aira* variables
    calculate *hankel* matrix
    calculate *moorepenroseinverse* matrix
    publish *state* 'initialised'
while **if** *outputbuffer* **is not** full:
    outputbuffer ← *sample*
    samplecount ← *samplecount* + 1
    if *outputbuffer* **is** full **publish** *state* 'ready'
while state **is** 'ready' and *samplecount* = length (*ouputbuffer*):
    outputbuffer ← shift(*outputbuffer*, 1)
    outputbuffer(lastsample) ← sample
    window ← filter(*outputbuffer*)
    window ← shift(reconstructinput(*window*), *delay*)
    inputbuffer ← shift(*inputbuffer*, *delay*)
    inputbuffer ← latestsample(*window*)
    smoothedinput ← smoothing(*inputbuffer*)
    inputsignal ← latestsample(*smoothedinput*)
    correlationsignal ← max(xcorrelation(*smoothedinput*))
    averagesignal ← latestsample(average(*smoothedinput*))
    derivativesignal ← latestsample(derivative(*smoothedinput*))

The smart adapters that incorporate a realtime version of AIRA need to be initialised before they can be used in real time. We know from the theory behind AIRA that given the size of the Hankel matrix, calculating the Moore-Penrose Inverse will take a long time which isn't sufficient for near real time calculation. Once the smart adapters are initialised, output data from the sensors is buffered before the reconstructed input and signals can become available.

4.4.3 HyDE Interface

The HyDE layer in the Diagnostic Program's stack creates and controls the HyDE instance and loads the harness and model files generated using the GME Tool. The fault injection scheduler is used to inject simulated faults into the system and coordinates this using the execution time maintained in the processing and logging layer. Depending on the type of fault, the data from the processing and logging layer is either modified or commands are sent to the UAV to fail specific components.

Observations directly from the OBC or smart adapters through the processing and logging layer are reported to HyDE at 50Hz or at a sample rate specified using the Diagnostic Agent. The main function of the HyDE interface layer is to correctly report the sensor data to HyDE and ensure that it corresponds to observations that are relevant to the system being simulated. Diagnoses

are requested and sent at the sampling frequency set by the Ground Station Operator. Before the requested diagnoses can be sent, they are first converted into candidate set objects from which vital information about the subsystem can be extracted such as locations and the values of the variables for each component.

4.5 Automated Fault Diagnosis

Following is a description of how HyDE is used to enable both autonomous fault diagnosis and automated aircraft reconfiguration.

4.5.1 Monitoring Aircraft Behaviour

Automating the process of monitoring the behaviour of the UAV depends partly on tracking the state of the system while it's operating which is enabled through the use of HyDE. The other part of the process involves processing and visualising this information in an actionable way. A straight forward way to do this is by using indicators that encapsulate desired states of components and conventionally this is done using printouts of the values of variables or coloured labels representing a component's state being either *nominal*, *faulty* or *warning*.

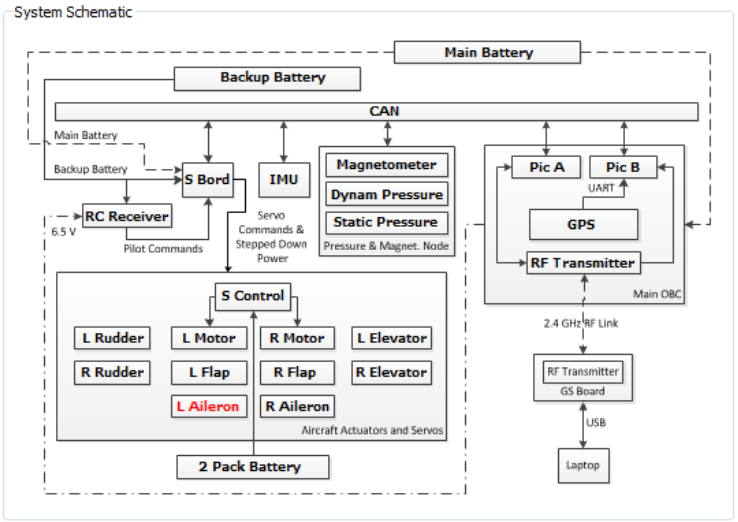


Figure 4.13: *Diagnostic Agent's system schematic representing the UAV's subsystems.*

Figure 4.13 shows a screenshot from the Ground Station's Diagnostic's interface and in it one can see a system schematic which represents the subsystem components of the Meraka Modular UAV. The interface to the Diagnostic System is accessible graphically using the Ground Station Software which represents the Diagnostic Agent. Each time the Diagnostic Agent receives diagnoses from the Diagnostic System, it changes the colour of the relevant component on the Ground Station to either *red*, *yellow*, or *green* which represents either a *known* or *unknown* fault or a *nominal* location for each component respectively.

4.5.2 Performing Fault Isolation

Automated Fault Diagnosis is achieved through the use of both the Diagnostic System and Diagnostic Agent. The Diagnostic System is configured using the scenario panel and the diagnosis panel as shown in Figure 4.14. The interface allows the vehicle operator to send commands to the Diagnostic System and receive status updates. Sensors can be configured, diagnoses can be saved and

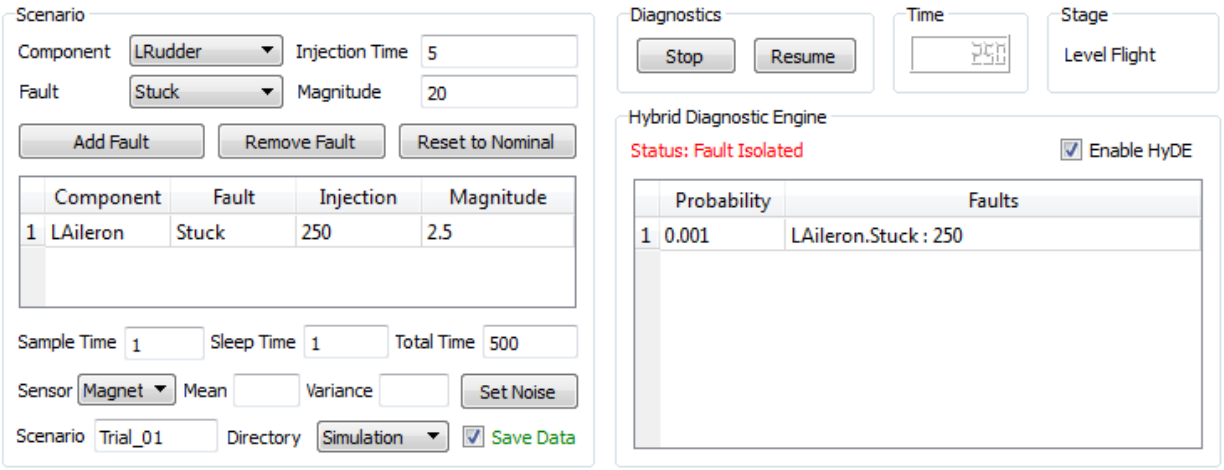


Figure 4.14: *Diagnostic Agent's Scenario and Diagnostics Interface to HyDE.*

faults can also be injected into the various components. Faults detected at each time step are listed in the HyDE panel along with the fault probability. The reported diagnoses are transformed to represent the state of the system using colour-coded labels for the respective on-board subsystems.

4.6 Utilising Fault-Free Subsystems

An effective method for detecting and utilising fault-free subsystems is needed to ensure the safety and reliability of the Meraka Modular UAV during flight missions. This process begins with fault diagnosis and is followed by aircraft reconfiguration. We now briefly discuss the mechanisms that will enable the UAV to perform these functions autonomously.

4.6.1 Reconfiguring the Aircraft

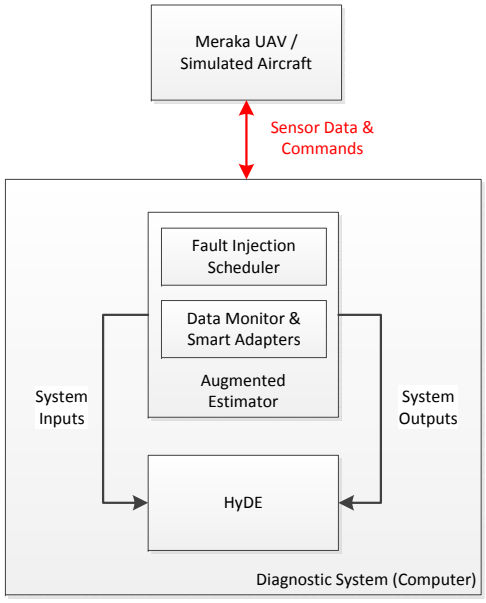


Figure 4.15: *Meraka Modular UAV HyDE Interface.*

The UAV is connected directly, as shown in red in Figure 4.15, to the Diagnostic System. This enables it to request diagnoses on demand or in real time and to also get an overall impression of the state of the system and the subsystem components using HyDE during the fault diagnosis

stage shown in red in Figure 4.16. This information, once extracted from the diagnoses reported by HyDE, can be used by the aircraft's guidance system or system supervisor operating in the OBC to make adjustments to the way the subsystem components are used during the reconfiguration stage shown in yellow in Figure 4.16. This could for instance include switching the power source to the backup batteries or even notifying the control system about actuators that are malfunctioning which could result in the activation of the control allocation optimisation algorithm to compensate for actuator stuck faults.

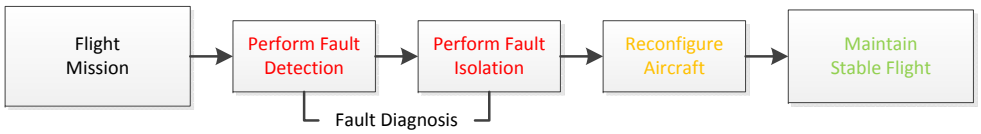


Figure 4.16: *Operational stages in FDIR.*

The implementation of the stages shown in yellow and green in Figure 4.16 are beyond scope of this research project and will be enabled in future iterations of the developed fault diagnostic system and modified target system based on the design of an IVHM System.

4.6.2 Maintaining Stable Flight

The development of the FDIR system began with an outline, listed in Chapter 1, of the stages that describe the operation of an FDIR System as shown Figure 4.16. We now briefly put into context how the FDIR system, based on HyDE, enables the UAV to maintain stable flight shown in green in Figure 4.16.

At this stage the UAV's system supervisor, shown in Figure 4.17, will have been instructed to use the new system configuration as the new model for the aircraft in order to maintain stable flight.

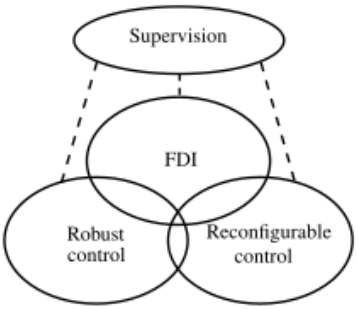


Figure 4.17: *Supervised Fault Tolerant Control, adapted from [50].*

With the FDI component at the center of the supervised fault tolerant control system, the reliability and adaptability of the system is increased. The end result of using HyDE within this context is that the flight mission is able proceed until such a time that there is a significant change in the aircraft's behaviour.

An important issue to take note of during the last stage, shown in green in Figure 4.16, is that the relevant systems onboard the UAV must be aware of the new system configuration and have the flight mission adapted accordingly. This adaptation needs to be prioritised based on the end effect mitigation procedures defined during the Reliability Analysis done before the fault modelling and identification process. This needs to be done in order to make system reconfiguration effective in reducing potentially adverse end effect events most notably an uncontrolled landing.

4.7 Overview

In this chapter an expandable FDIR architecture is developed to facilitate the addition of more FDIR methods in an effort to allow the fault diagnostic system's functionality to be enhanced easily. The architecture is integrated based on the design of a simplified Integrated Vehicle Health Management (IVHM) System. An overview of the IVHM architecture, corresponding system configuration as well as the SIL configuration used to develop the integrated system is given.

The simulation environment based on MATLAB serves as the mediator between the Diagnostic Agent and Diagnostic System. It also allows the control engineer to test the fault diagnostic system being developed. A Simulink Model is used to simulate the non linear dynamics of the aircraft using a repurposed game controller, MATLAB GUI and QTGLEngine.

The Diagnostic Agent or Ground Station is used as the command central that sends instructions to the UAV from the ground station operator's computer. It has several options for connecting with other systems and connects with the MATLAB based simulation environment through a virtual serial port. A brand new Diagnostics GUI was built into the Ground Station to monitor the UAV and detect faults and changes in subsystems in real time.

The Diagnostic System built on the Model B+ Raspberry Pi is used to perform diagnoses onboard the Meraka Modular UAV. It communicates with the Diagnostic Agent and the rest of the subsystems onboard the UAV using ethernet and has the option of also using the PiCAN expansion board to connect to the CAN (Controller Area Network). Data monitors and smart adapters that implement a real time version of AIRA are used to perform input reconstruction and report observations to the HyDE instance instantiated in the diagnostic program. In Chapter 5, the declarative subsystem component models for the UAV are developed for use in the fault diagnostic system.

A description of how HyDE is used to enable both autonomous fault diagnosis and automated aircraft reconfiguration is given. This includes a description of how the process of monitoring the behaviour of the UAV and performing fault isolation is automated. The Diagnostic Agent's brand new diagnostic interface is described and includes a scenario, diagnostics and a system schematic panel that updates in real time representing the states of the UAV's onboard subsystems.

The process of utilising fault-free subsystems begins with fault diagnosis and is followed by aircraft reconfiguration. The mechanisms that enable the UAV to perform these functions autonomously are described. These include the reconfiguration of the aircraft and maintaining stable flight in the presence of a modified system configuration. In Chapter 6 system tests are performed using the integrated fault diagnostic engine and declarative models of the Meraka Modular UAV's critical subsystems.

Chapter 5

System Tests

As part of the system testing process, we begin by defining the test conditions and then describe how the developed fault diagnostic system is tested extensively by deliberately failing the UAV's actuators during simulated flight. The performance of the system is then verified by using flight test data collected by the Meraka Modular UAV during flight missions.

5.1 Test Conditions

Defining the test conditions begins by describing the link between the theoretical model of the UAV and the physical aircraft. Focus can then be shifted toward specifying the flight trajectory followed by the aircraft and the various hypotheses that need to be tested and the performance measures used.

5.1.1 Physical Aircraft

Figure 5.1: *An image of the Meraka Modular UAV.*

The mathematical model developed in Chapter 2 that describes the dynamics of the Meraka Modular UAV, shown in Figure 5.1, is based on the use of AVL software. The declarative models for HyDE and the state space representations used in AIRA are also based on the mathematical model of the aircraft. It is important to analyse the UAV's structure, shown in Figure 5.2 in order to gain insight into how the actuators behave. The aircraft's reference system is defined according to the control surface deflection δ and location in the wings and tail, and anticipated flight based on the geometric structure in Figure 5.2. The size of all control surfaces on the UAV are small relative to the size of each structure that they are housed in. Hypotheses are evaluated by reporting observations derived from the behaviour of the actuators and other components during tests.

It is evident that the moments caused by the deflection of the elevators relative to the y_B axis are large by observing where the elevators are relative to the rest of the UAV. It is expected that the

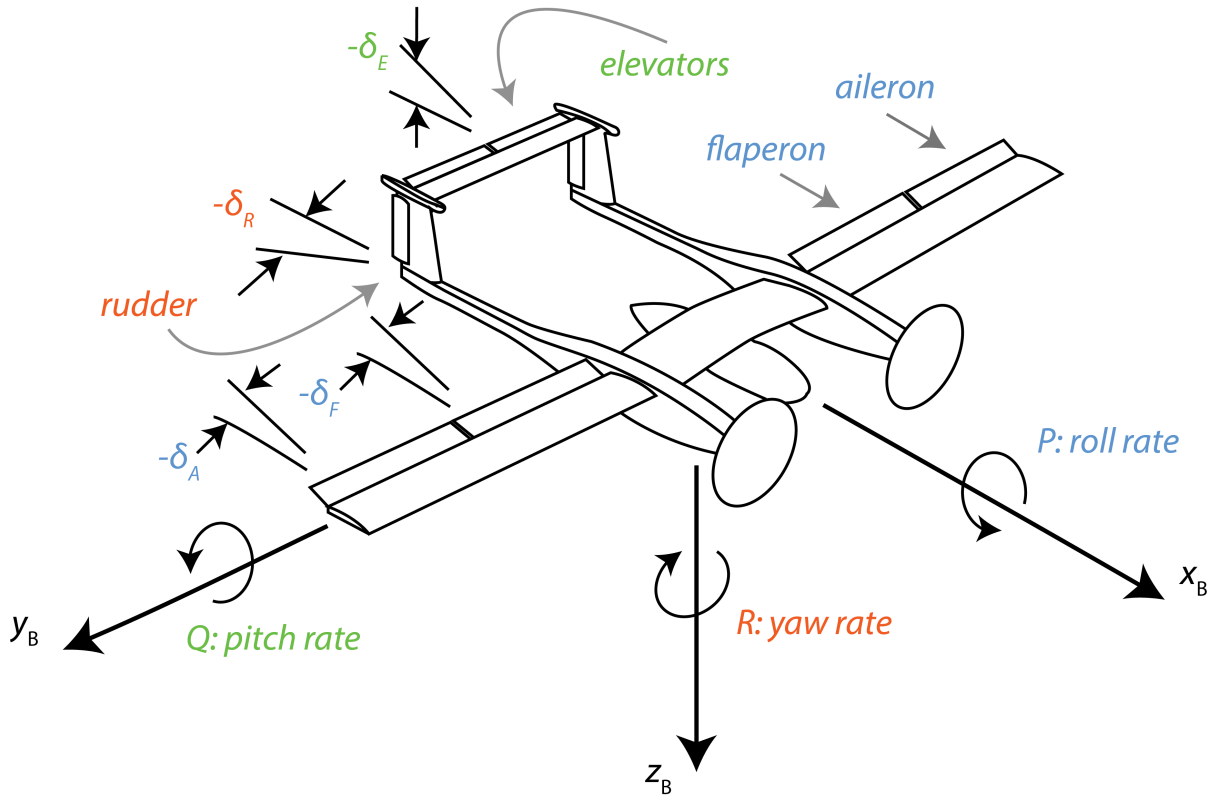


Figure 5.2: *A geometric model of the Meraka Modular UAV showing components of interest and the associated reference system.*

best FDI performance will result from using the pitch rate p and that it will be difficult for the FDIR system to discern between the left and right elevators because of the aircraft's symmetry. Similarly, the rudders also produce small moments around the x_B axis but produce larger moments about the z_B axis. It is expected that the best FDI performance will result from using the yaw rate r combined with another observable property affected by the rudder's deflection.

The ailerons and flaperons produce large moments about the x_B axis and their combined effect affects both the aircraft's roll and yaw rates given that the rudder has a smaller effect on the yaw rate r. However in light of this, it is expected that the best FDI performance will result form using a combination of the roll rate p and the yaw rates r. It is also expected that it will be difficult for the FDIR system to discern between the left and right control surfaces as the combined effect of the control surfaces result in the observed behaviour the aircraft exhibits.

An accurate mathematical model of the UAV should exhibit the same characteristics observed from the geometric model and these can be readily seen by inspecting the state space matrices, written in Section 2.5.4, calculated at trim.

5.1.2 Flight Trajectory

Flight tests are a crucial part of validating the effectiveness of the design and development process. The following flight tests were conducted in [38] to test the performance of the FDIR system:

1. Simulate actuator stuck faults at 0°
2. Simulate actuator stuck faults at 2.5°.

The control reallocation system developed in [7] was tested at 2.5° intervals to solve the control optimisation problem for faulty actuators within a 0° to 15° deflection range. As a consequence

of this, in the previous research project [38], the decision was made to simulate and test FDI performance for 0° and 2.5° stuck faults. Given that the same flight test data collected in [38] is reused, the same stuck in place faults are simulated in this research project as well. The flight tests were conducted at the Helderberg Radio Flyers Club in the Western Cape South Africa [38]. The actuators on the aircraft were failed deliberately for the durations specified and flight test data was collected using onboard sensors. During the flight tests, it is reported that the wind velocity was more than the anticipated 2 m/s velocity and the minimum and maximum wind gusts were 2.3 and 7.7 m/s respectively. The FDI system needs to take these into account to ensure effective FDI performance.

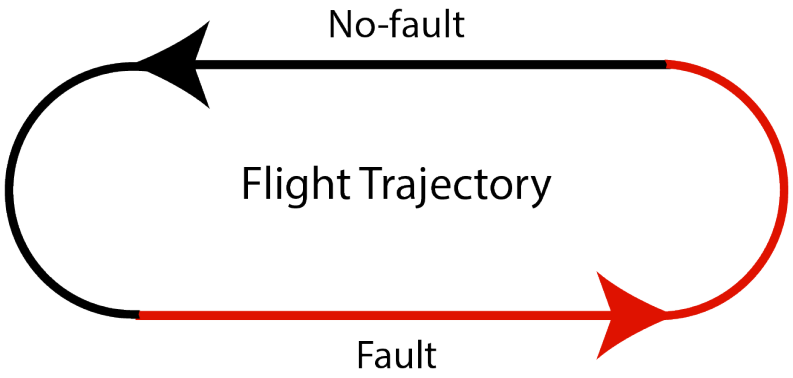


Figure 5.3: *Flight trajectory, adapted from [38].*

The ESL's UAV pilot took on the role of the UAV's control system as shown in Figure 1.8. The UAV was flown in the flight trajectory shown in Figure 5.3. No faults were injected into the UAV for half the time it was on the track (30 seconds). For the other half of the time it was on the track (30 seconds), faults were injected into the UAV for specific durations. This flight sequence was repeated until all tests were completed.

5.1.3 Scenarios

Structuring the approach used to test the FDIR system is crucial for gaining reproducible and specific insights in an effective manner. These particular scenarios were chosen in [38] to investigate the FDI performance at different levels of actuator excitation as well as flight trajectory. The performance of the FDI methods compared in [38] depends on the use of residuals in order to inform the fault decisions that are taken concerning the existence of a fault. It was found that an increase in actuator excitation (due to process noise or flight manoeuvres) makes the existence of faults more prominent [38]. This improves FDI performance by making it easier to identify active fault scenarios due to an increase in the variance of the residuals generated by each FDD method. The following scenarios were generated in order to test certain hypotheses:

1. Straight and level flight with minimal process noise.
    - It is hypothesised that poor FDI performance will result as a consequence of minimal process noise which decreases the variance of the residuals used.
2. Circular flight path that closely resembles flight test trajectory.
    - It is hypothesised that poor FDI performance will result when the aircraft performs manoeuvres that deviate significantly from trim working point designed for.
3. Arbitrary flight mission that induces actuator excitation.
    - It is hypothesised that FDI performance improves as a consequence of additional actuator excitation due to the additional actuator excitation from the flight manoeuvres.

4. Straight and level flight with substantial process noise.

    - It is hypothesised that FDI performance improves as a consequence of substantial process noise which increases actuator excitation and the variance of the residuals used.

Actuator deflections need to be shown and it is desirable that fault decisions made by the FDIR system follow the real scenarios as closely as possible. It is expected that high levels of process noise and unmodelled dynamics may affect the outcome. According to [38], the following flight test event summaries shown in Table 5.1 were performed where simulations of Flight Test 1 are done according to Scenario 4 and Flight Test 2 according to Scenario 1 unless stated otherwise.

Flight Test One			
Start Time (s)	End Time (s)	Event Description	
35.6	65.6	Fail left aileron at 0°	stuck
90.32	123.52	Fail right aileron at 0°	stuck
146.84	180.92	Fail left elevator at 0° from trim	stuck
204.44	236.92	Fail right elevator at 0° from trim	stuck
261.76	292.12	Fail left rudder at 0°	stuck
316.32	347.36	Fail right rudder at 0°	stuck
Flight Test Two			
Start Time (s)	End Time (s)	Event Description	
28.96	63.4	Fail left aileron at +2.5°	stuck
87.72	120.70	Fail right aileron at +2.5°	stuck
145	182.2	Fail left elevator at -2.5° from trim	stuck
204.6	238.7	Fail right elevator at -2.5° from trim	stuck
263.2	289.08	Fail left rudder at +2.5°	stuck
311.44	339.6	Fail right rudder at +2.5°	stuck

Flight test event summary

Table 5.1: *First and second flight test event descriptions.*

5.1.4 Performance Measures

It is necessary to identify specific standardised performance measures that can be used when evaluating the performance and quality of the developed fault diagnostic system. The quality of an FDIR system can be determined in the following ways according to [20]:

- false alarm rate - spurious faults per unit time announced
- missed detection rate - rates per unit time that faults are missed
- fault sensitivity - fault magnitude required in non-nominal location
- detection time - time taken to report or isolate non-nominal locations
- robustness - stability of the diagnostic performance in the presence of noise.

Please see Sections 1.3.3 and 1.3.5 for a description of the test conditions in addition to those reflected by the reconstructed actuator inputs shown for each flight test. Modulated control input signals (sinusoidal) are used as actuator inputs during model development for component testing only and not during system tests. Actuator excitation is varied from zero to three degrees [38] in order to determine fault sensitivity and to determine robustness, the aerodynamic stability and process noise is varied in the MATLAB Simulation.

5.2 Simulation Tests

Simulated flight based hypothesis testing begins with an evaluation of the effectiveness of the input reconstruction algorithm. This is followed by an analysis of the simulation results and the FDI performance of the fault diagnostic system.

5.2.1 Reconstructed Inputs

In conventional flight, the aircraft's flight envelope is limited to trim condition. From the scenarios listed in Section 5.1.3, straight and level flight most closely resembles conventional aircraft flight and it is with this in mind that the effectiveness of the input reconstruction algorithm is evaluated. Consider the reconstructed inputs representing estimated and actual actuator deflections, shown in Figure 5.4, that are produced through a simulation resembling the first flight test event description in Table 5.1. It is evident that the actuator deflections are accurately reconstructed with a consistent reconstruction delay even during faults. Note that the left and right actuator deflections are similar by design and provisions were made, see Section 3.4.3, for the future integration of AFTC methods or position sensors to enable the Diagnostic System to discern between the control surfaces.

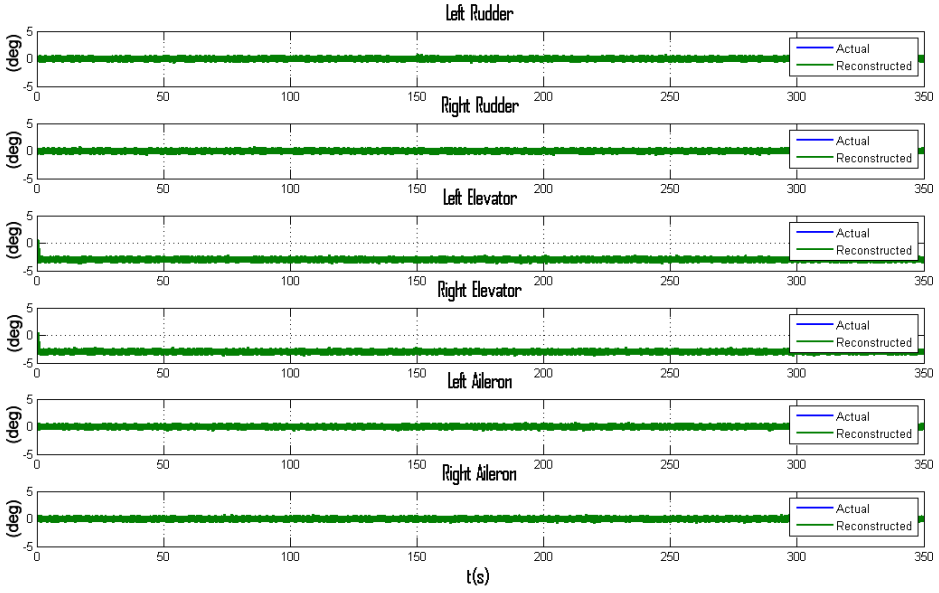


Figure 5.4: *Reconstructed actuator deflections produced during simulations of straight and level flight with process noise (2 m/s wind velocity).*

Consider the reconstructed inputs representing actuator deflections, as shown in Figure 5.5, produced during a simulation resembling the second flight test event description in Table 5.1. It is important to notice that AIRA's ability to estimate the control surface deflection is not affected by the occurrence of faults since the states observed are related to the aircraft's flight dynamics.

5.2.2 Simulation Results

Having considered the effectiveness of the input reconstruction algorithm, we now move onto investigating the simulation results generated from the scenarios outlined in Section 5.1.3. The first three of the scenarios are simulated according to the flight test one event description listed in Table 5.1. This is followed by a more in depth investigation through a simulation of Scenario 4 according to the flight test two event description listed in Table 5.1. Table 5.2 contains a summary of the results produced during simulations of Scenario 1 - straight and level flight with minimal process noise. It is expected that poor FDI performance will result as a consequence of minimal process noise. Due to the increased sensitivity of AIRA and filtering incorporated in the smart adapters,

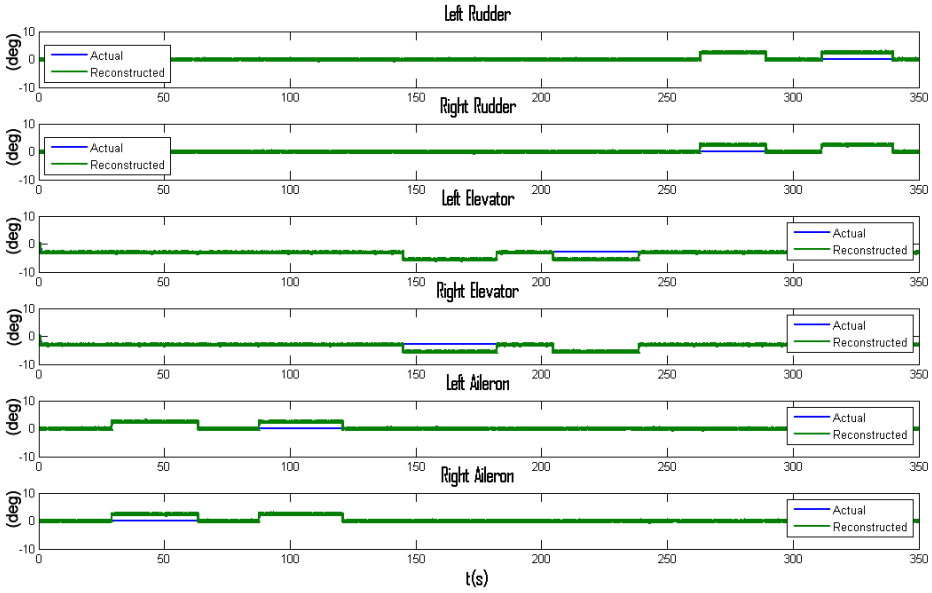


Figure 5.5: *Reconstructed actuator deflections produced during simulations of straight and level flight without process noise.*

the fault diagnostic system is still able to detect and isolate faults with the elevator having the fastest isolation time followed by the ailerons and lastly the rudders. The varying detection times throughout the system tests are attributed largely to the natural motion of the aircraft characterised by the phugoid, dutch roll and short period modes. These modes tend to naturally move the aircraft around and in different directions. The actuators that either have higher frequency natural disturbances or larger actuator excitation are generally detected faster than those that don't. Since passive fault detection is performed, the Fault Diagnostic System needs to wait for the actuators to receive enough excitation to breach the 0.4° actuator excitation threshold used in HyDE. In addition, HyDE only finds out about faults about 1 second after they've occurred and must backtrack to discover what fault occurred and when based on reported inconsistencies.

Detailed Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.8.1, 2.9.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck	Left & right aileron stuck at 0°	1.0	1.8	Yes
2.6.1, 2.7.1	actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right elevator stuck at 0° from trim	1.0	1.6	Yes
2.4.1, 2.5.1	actuation.left_rudder.stuck, actuation.right_rudder.stuck	Left & right rudder stuck at 0°	1.0	2.0	Yes

Table 5.2: *Results of diagnoses performed during simulations of straight and level flight with process noise, where t_d is the detection time and t_i is the total fault detection and isolation time.*

Table 5.3 lists the results produced during simulations of Scenario 2 - a circular flight path that closely resembles flight test trajectory. When compared to the results in Table 5.2, the isolation times are noticeably larger, with the rudder having the largest, although the detection times are similar. This is a consequence of the aircraft producing manoeuvres that move it out of the trim condition flight envelope which is the working point the reconstruction algorithm is designed around. The circular trajectory also produces biases in the yaw and roll rates which inadvertently affect the reconstructed aileron and rudder deflections.

Detailed Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.8.1, 2.9.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck	Left & right aileron stuck at 0°	1.0	2.0	Yes
2.6.1, 2.7.1	actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right elevator stuck at 0° from trim	1.0	1.5	Yes
2.4.1, 2.5.1	actuation.left_rudder.stuck, actuation.right_rudder.stuck	Left & right rudder stuck at 0°	1.0	2.5	Yes

Table 5.3: *Results of diagnoses performed during simulations of a circular flight path, where t_d is the detection time and t_i is the total fault detection and isolation time.*

Detailed Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.8.1, 2.9.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck	Left & right aileron stuck at 0°	1.0	1.3	Yes
2.6.1, 2.7.1	actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right elevator stuck at 0° from trim	1.0	1.1	Yes
2.4.1, 2.5.1	actuation.left_rudder.stuck, actuation.right_rudder.stuck	Left & right rudder stuck at 0°	1.0	1.2	Yes

Table 5.4: *Results of diagnoses performed during simulations of arbitrary flight missions, where t_d is the detection time and t_i is the total fault detection and isolation time.*

Simulations of Scenario 3 - an arbitrary flight mission that induces actuator excitation, appear to have comparably produced the best results as seen in Table 5.4. The isolation times are considerably smaller, the FDI performance improves as a consequence of additional actuator excitation. This is inline with the underlying principle of the fault diagnostic system's smart adapters. Actively producing a noticeable difference between desired and actual actuator deflections directly affects the correlation between the two quantities which makes it easier for the FDI system to reason about the state of the component.

Simulations of Scenario 4 - straight and level flight with substantial process noise, have produced results, shown in Table 5.5, that are comparable in nature to those produced during simulations of Scenario 3. The additional process noise results in increased actuator excitation which leads to improved performance during straight and level flight. In this more thorough analysis, the results of injecting faults into abstracted system components are included as well. It is important to note that all the detailed model diagnoses listed contain multiple discrete faults instead of just one fault and this is a consequence of HyDE's ability to detect and reason about multiple discrete faults.

To reduce the isolation times for the other actuator faults, some tuning of the thresholds needed to be done. It is also worthwhile noticing that the relative fault isolation times between the detailed and abstracted models are similar. However the detection times for the abstracted components are considerably faster due to not having a delay from using smart adapters to report observations. Overall during the simulation of the Scenarios in Section 5.1.3, HyDE generated the most likely candidates and detected the faults consistently with varying isolation times for each component. Having investigated HyDE's diagnostic capabilities in simulation, we can now move to analysing the FDI performance using standardised performance measures.

Detailed Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.8.1, 2.9.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck	Left & right aileron stuck at 2.5°	1.0	1.5	Yes
2.6.1, 2.7.1	actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right elevator stuck at 2.5° from trim	1.0	1.1	Yes
2.4.1, 2.5.1	actuation.left_rudder.stuck, actuation.right_rudder.stuck	Left & right rudder stuck at 2.5°	1.0	1.2	Yes
2.8.1, 2.9.1, 2.6.1, 2.7.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck, actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right aileron & elevator stuck at 2.5°	1.0	1.5	Yes
2.10.4, 2.11.4	actuation.left_flaperon.partial, actuation.right_flaperon.partial,	Left and right flaperon loose effectiveness	1.0	1.3	Yes
2.4.2, 2.5.2	actuation.left_rudder.hardover, actuation.right_rudder.hardover,	Left and right rudder hardover	1.0	1.2	Yes
2.6.3, 2.7.3	actuation.left_elevator.floating, actuation.right_elevator.floating,	Left and right elevator floating	1.0	1.3	Yes
Abstracted Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.2.1	actuation.servo_board.miscalibrated	Servo board miscalibrated	0.2	0.3	Yes
2.1.1	actuation.rc_receiver.out_range	RC receiver out of range	0.2	0.5	Yes
1.1.1	aircraft.avionics_battery.degraded	Avionics battery degraded	0.1	0.3	Yes
1.2.2	aircraft.backup_battery.discharged	Backup battery discharged	0.1	0.2	Yes
1.2.3	aircraft.backup_battery.unknown	Backup battery fault unknown	0.1	0.5	Yes

Table 5.5: *Results of diagnoses performed during simulations of straight and level flight, where t_d is the detection time and t_i is the total fault detection and isolation time.*

5.2.3 Diagnostic Performance

We now analyse the FDI performance that is expected during a typical 1 hour fight mission by generating standardised performance indicators using simulation results from 120 experiments, lasting 30 seconds each, performed on the linux virtual machine emulating the Raspberry Pi. Generating these performance indicators helps one understand the FDI performance under practical conditions where opportunities for testing are limited. In following the methodology adopted in the previous research project [38], the following performance indicators will be used to determine the FDI system's fault sensitivity and detection time:

- process noise - the process noise resulting from wind disturbances is varied from low wind to a substantial amount of wind
- parameter variation - the system parameters are varied by changing the aerodynamic stability and control derivatives
- actuator excitation - the actuator deflections are varied between the allowable limits

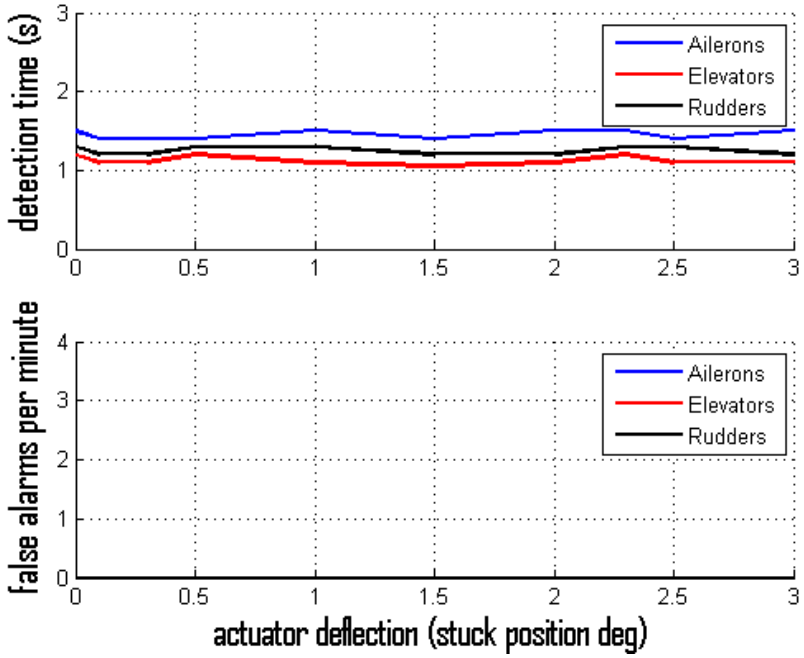


Figure 5.6: *Simulated detection time and false alarms per minute due to varying stuck positions.*

Figure 5.6 shows the detection time and false alarms per minute trends due to varying actuator deflection stuck positions. These indicators are constructed by using the averages of the results from the experiments performed during straight and level flight with substantial process noise. Missed detections are not considered given that HyDE in general takes less than 5 seconds to detect and isolate faults. A readily noticeable characteristic of this figure is that the there were no false alarms and that the total fault detection and isolation times vary between stuck positions. This is to be expected given that the results aren't reproduced exactly during each experiment due to the inherent randomness of the simulation environment with ambient wind disturbances. The number of experiments performed over the stuck position interval is also relatively low, however it is expected that the trends for the detection times will average to relatively constant values with the increase in experiments performed. Looking at the detection times, it is evident that the elevator detection time is the fastest followed by the ailerons and rudders. The false alarm rate remains consistently low and the detection times average to about 1.5 seconds at 2 degrees. This is the consequence of having tuned the thresholds used in the constraints for each actuator mode of operation as a trade off between fault sensitivity and false alarms due to disturbances. For larger deflections, it is expected that the FDI performance degrades as the aircraft flies outside trim condition. However, while still in the trim flight envelope, AIRA will perform as expected and reconstruct the actuator input regardless of the magnitude of the deflection.

Figure 5.7 shows how the performance indicators change due to varying all the dimensionless aerodynamic constants from their original values (100%). The detection times correspond to the experiments used to investigate stuck faults at trim deflection. The stuck position needed to remain constant while the dimensionless aerodynamic constants were varied in order to derive the performance indicator through the experiments. The lowest detection time is around 100% of the original values and degrades considerably around a 10% deviation. The increase in detection time is attributed to the increase in false alarms which affects HyDE's reasoning process causing HyDE to search for additional consistent candidates. This is the inherent cost of the increased sensitivity of AIRA which requires more accurate process parameters and flight around trim. FDI performance is poor under ideal conditions it then improves with increased process noise before degrading as can be seen by the increasing false alarms per minute rate in Figure 5.8. To derive this performance indicator, the stuck positions remained at trim deflection (for each control surface investigated) before averaging the results. An important observation to make here is that when

there aren't any wind disturbances (wind velocity is $0m/s$) the aircraft's natural modes of motion tend to influence the passive diagnosis process by inducing smaller levels of excitation at lower frequencies. Once the ambient wind velocity increases, the process noise increases and with it the level of actuator excitation along with a decrease in cross correlation between the actual and desired actuator deflections. This explains why the total fault detection and isolation time decreases drastically, at around $0.2m/s$, with increasing wind velocity before increasing again. For larger wind velocities this effect is most pronounced at around $3m/s$ as is evident from the increasing false alarms per minute rate. This implies that noise makes it difficult to distinguish between the actuator modes and that about 30 false alarms can be expected for wind velocities in the range of 4 m/s over a 1 hour flight mission.

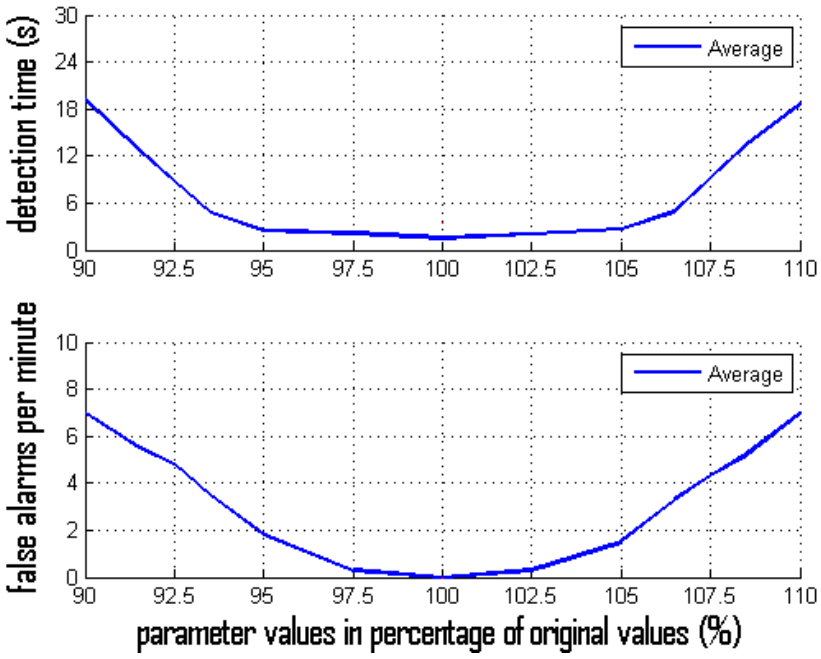


Figure 5.7: *HyDE detection time and false alarms per minute due to varying parameters.*

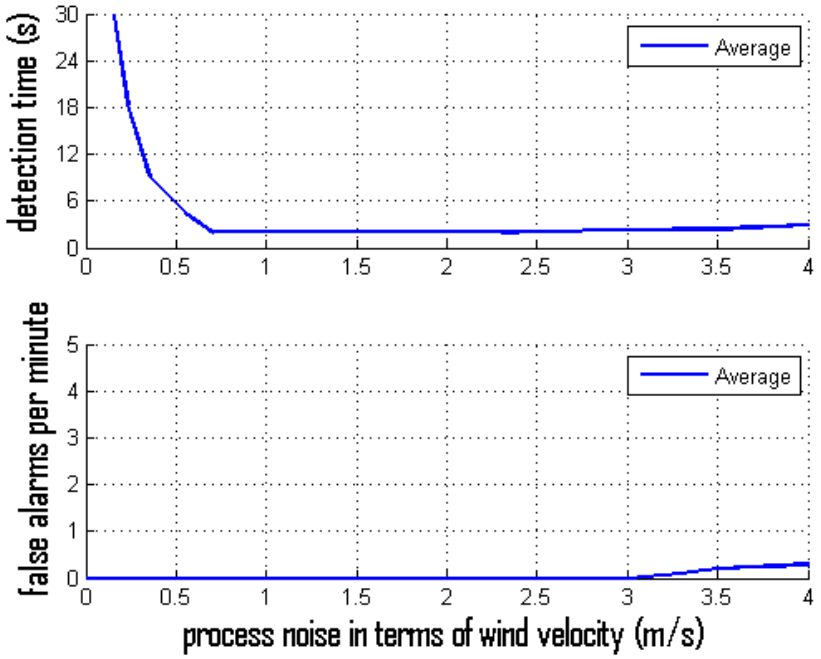


Figure 5.8: *HyDE detection time and false alarms per minute due to process noise.*

5.3 Flight Tests

Actual flight based hypothesis testing begins with an evaluation of the effectiveness of the input reconstruction algorithm. This is followed by an analysis of the flight test results and the FDI performance of the fault diagnostic system.

5.3.1 Reconstructed Inputs

Having evaluated the input reconstruction algorithm through simulation tests in Section 5.2.1, we now move to the results produced during practical flight testing. Flight testing is done according to Scenarios 1 and 4 in Section 5.1.3 based on the flight test event descriptions in Table 5.1. Consider the reconstructed inputs representing estimated and actual actuator deflections, shown in Figure 5.9, that are produced during the first flight test with the event description shown in Table 5.1. It is evident that the actuator deflections are accurately reconstructed within reasonable margins and with a consistent reconstruction delay even during faults and filtering is effective in reducing the effects of noise. It is important to observe that the reconstruction algorithm follows the faulty actuator input for both left and right actuators, this reconstructed input along with the commanded input is what is fed into HyDE to perform actuator fault diagnosis.

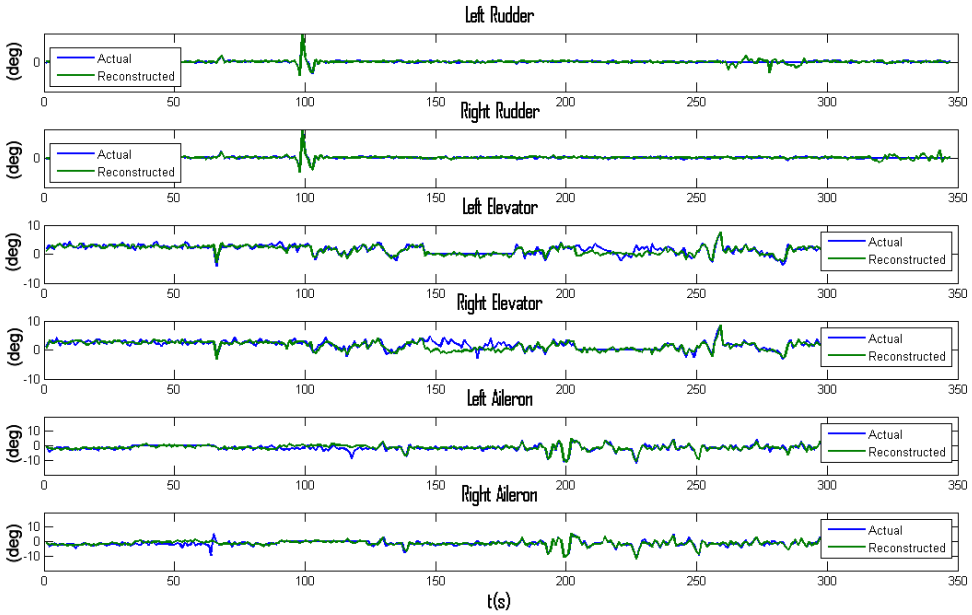


Figure 5.9: *Reconstructed actuator deflections produced with data from flight test one.*

Consider now the reconstructed inputs representing actuator deflections, as shown in Figure 5.10, produced during the second flight test with the event description shown in Table 5.1. Unfortunately the experiments with the rudder deflections weren't successful for both flight tests. Note that the left and right reconstructed actuator deflections are similar and have been adjusted to account for the control input mixing done by the servo board however only the least correlated input is used. Just as in the case of the simulated input reconstruction in Section 5.2.1, actuator excitation that results from the added process noise adds to the effectiveness of the input reconstruction algorithm.

5.3.2 Flight Test Results

Having considered the effectiveness of the input reconstruction algorithm, we now move onto investigating the flight test results generated from the scenarios outlined in Section 5.1.3 according to the flight test event descriptions in Table 5.1. For detail regarding the actual tests, please see [38]. The fault diagnostic system was tested using the flight test data collected onboard the Meraka Modular UAV. Flight test data was simulated and fed into the diagnostic system along with the

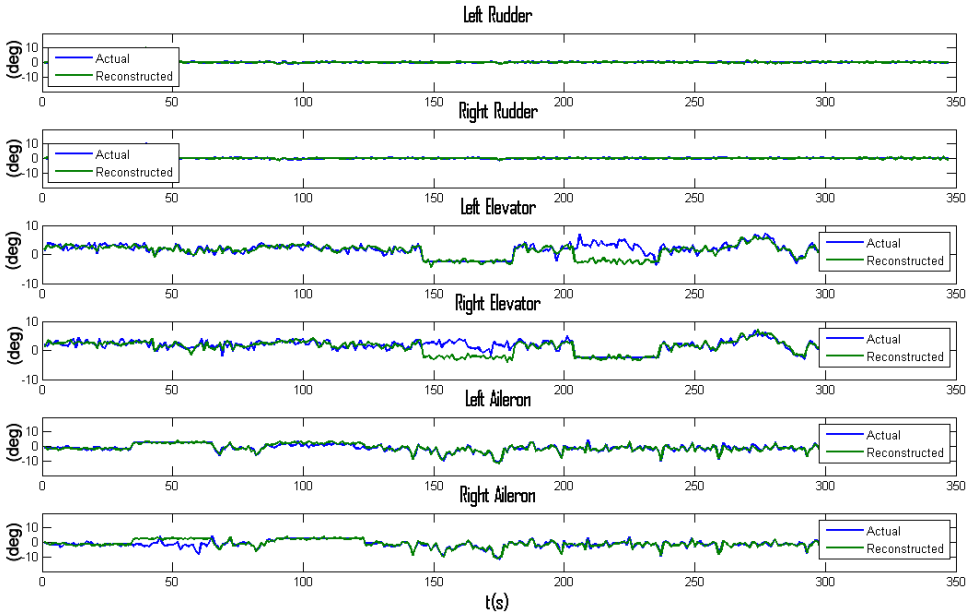


Figure 5.10: *Reconstructed actuator deflections produced with data from flight test two.*

Detailed Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.8.1, 2.9.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck	Left & right aileron stuck at 0°	1.0	5.0	Yes
2.6.1, 2.7.1	actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right elevator stuck at 0° from trim	1.0	2.5	Yes
2.4.1, 2.5.1	actuation.left_rudder.stuck, actuation.right_rudder.stuck	Left & right rudder stuck at 0°	1.0	3.0	Yes

Table 5.6: *Results of diagnoses performed using data from flight test one, where t_d is the detection time and t_i is the total fault detection and isolation time.*

then injected locked-in-place or stuck faults. The input reconstruction algorithm was also modified to match the flight conditions the flight test data was collected under. Tables 5.6 and 5.7 list the diagnoses produced using data from flight test one and two. The performance of the fault diagnostic system is as expected according to insights gained during simulation tests with the added difference of the isolation times taking considerably longer. This is due to the inherent noise and increased wind disturbances that affect the accuracy of the diagnoses - HyDE spends more time searching for and prioritising candidates that are consistent. HyDE also takes considerably longer to detect 0° stuck faults relative to the 2.5° stuck faults. Performing back transversals to isolate faults becomes easier when the actuator stuck positions i.e. at 2.5° deviate from the 0° trim deflections.

5.3.3 Diagnostic Performance

Standardised performance indicators were generated through simulation tests done in Section 5.2.2. These performance indicators make it possible to determine what FDI performance can be expected and by what margin they will need to be adjusted to reflect physical reality. The fault diagnostic program was tested while running on the Raspberry Pi instead of the linux virtual machine.

Only scenarios 1 and 4 were generated during flight tests one and two so effectively only a few data points from Tables 5.6 and 5.7 are available for each performance measure. These are overlaid

Detailed Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.8.1, 2.9.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck	Left & right aileron stuck at 2.5°	1.0	3.5	Yes
2.6.1, 2.7.1	actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right elevator stuck at 2.5° from trim	1.0	1.5	Yes
2.4.1, 2.5.1	actuation.left_rudder.stuck, actuation.right_rudder.stuck	Left & right rudder stuck at 2.5°	1.0	2.0	Yes

Table 5.7: *Results of diagnoses performed using data from flight test two, where t_d is the detection time and t_i is the total fault detection and isolation time.*

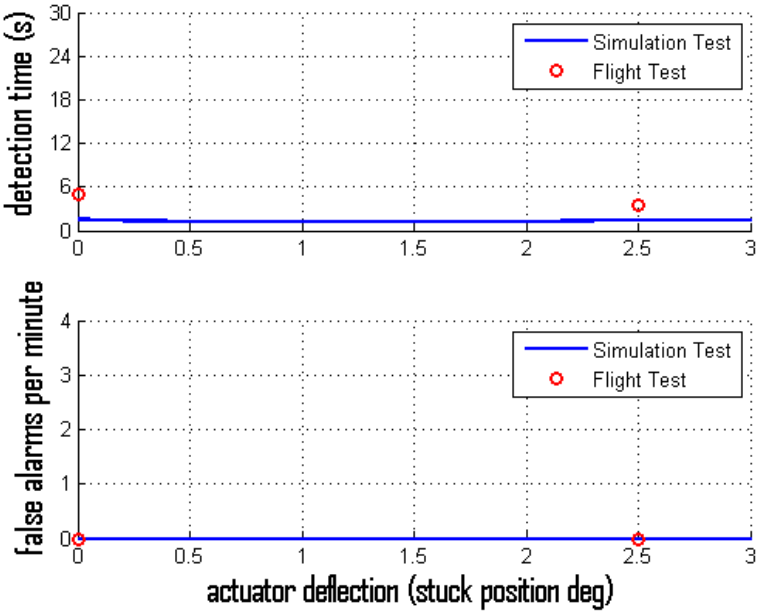


Figure 5.11: *HyDE detection time and false alarms per minute due to varying stuck positions.*

onto the relevant performance measures as shown in Figures 5.11, 5.12 and 5.13 to see how well the trends match and by how much they need to be adjusted. These results suggest that there are unmodelled dynamics in the simulations partly due to the slightly inaccurate parameters shown by the shifted flight test results in Figure 5.12. They also show that practically, fault detection takes considerably longer although the flight test and simulation results are comparable. The flight test data also has inherent noise due to the flight test conditions which explains the more parabolic nature of the flight test results due to varying parameters. The slightly longer detection times in this case is a good indication and suggests that the Raspberry Pi's more constrained processing environment (less RAM = 512 MB and processing speed = 700 MHz) is able to still effectively perform fault diagnosis which is a crucial observation. No examples have been found in literature that mention the Raspberry Pi being used to perform diagnoses using HyDE.

5.4 Evaluation of Results

Evaluating the simulation and flight test results is a crucial part of determining the effectiveness of the design and development process. Furthermore, comparing the results produced in this research project with the research project that directly precedes it also serves as a means of gauging the progress that has been made. Lastly, briefly investigating results produced independently is an effective way of getting an indication of the FDI performance of the fault diagnostic system.

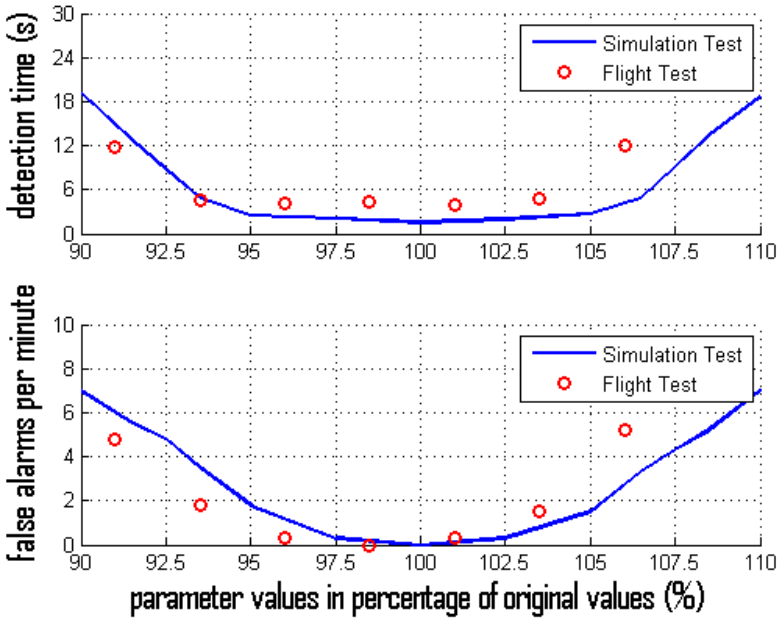


Figure 5.12: *HyDE detection time and false alarms per minute due to varying parameters.*

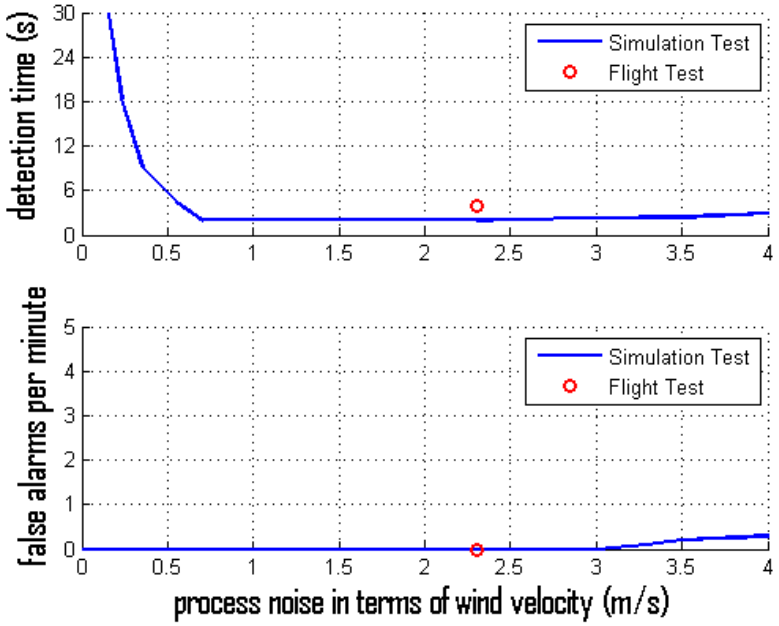


Figure 5.13: *HyDE detection time and false alarms per minute due to process noise.*

5.4.1 Tests Evaluation

Table 5.8 shows a summary of the diagnosis results generated through using simulation and flight test data for both detailed and abstracted models. All relevant actuator fault modes are listed along with their varying isolation times. The detection times of the more detailed models are larger which also suggests that they are more computationally intensive. In contrast to this, the abstracted models have considerably lower isolation times and were relatively easier to implement. Noticeable differences between the actuator faults suggest that fault detection that involves ailerons or flaperons takes longer to isolate followed by rudder faults and lastly elevator faults. HyDE's flexible modelling paradigm enables the control engineer to easily model systems that exhibit behaviour that is discrete, continuous or both whilst allowing different levels of abstraction. Systems modelled using higher levels of abstraction through the use of thresholds are generally faster. More detailed system models in general are better able to detect and isolate faults using less sensors

however they generally don't incorporate fault detection during transients. The main disadvantage of using more detailed models is the increase in computational requirements and the implementation time required which is generally unbounded. Having evaluated the results we now move to comparing the FDI performance indicators.

Detailed Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.8.1, 2.9.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck	Left & right aileron stuck at 2.5°	1.0	3.5	Yes
2.6.1, 2.7.1	actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right elevator stuck at 2.5° from trim	1.0	1.5	Yes
2.4.1, 2.5.1	actuation.left_rudder.stuck, actuation.right_rudder.stuck	Left & right rudder stuck at 2.5°	1.0	2.0	Yes
2.8.1, 2.9.1, 2.6.1, 2.7.1	actuation.left_aileron.stuck, actuation.right_aileron.stuck, actuation.left_elevator.stuck, actuation.right_elevator.stuck	Left & right aileron & elevator stuck at 2.5°	1.0	3.5	Yes
2.10.4, 2.11.4	actuation.left_flaperon.partial, actuation.right_flaperon.partial,	Left and right flaperon loose effectiveness	1.0	3.0	Yes
2.4.2, 2.5.2	actuation.left_rudder.hardover, actuation.right_rudder.hardover,	Left and right rudder hardover	1.0	1.5	Yes
2.6.3, 2.7.3	actuation.left_elevator.floating, actuation.right_elevator.floating,	Left and right elevator floating	1.0	1.5	Yes
Abstracted Models					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.2.1	actuation.servo_board.miscalibrated	Servo board miscalibrated	0.2	0.5	Yes
2.1.1	actuation.rc_receiver.out_range	RC receiver out of range	0.2	1.0	Yes
1.1.1	aircraft.avionics_battery.degraded	Avionics battery degraded	0.2	0.5	Yes
1.2.2	aircraft.backup_battery.discharged	Backup battery discharged	0.2	0.5	Yes
1.2.3	aircraft.backup_battery.unknown	Backup battery fault unknown	0.2	1.0	Yes

Table 5.8: *Summary of the diagnosis results generated through simulation and flight tests, where t_d is the detection time and t_i is the total fault detection and isolation time.*

5.4.2 Results Comparison

We now compare the FDI performance indicators of FDI systems based on the optimised parity space method used in the previous research project and the approximate input reconstruction algorithm (AIRA) used in this research project. This research is a continuation of a series of projects done at the Electronic System Laboratory (ESL) that deal with the design and development of Fault Tolerant Control Systems. The focus of the research project that immediately precedes this one is the analysis and comparison of two methods for UAV actuator FDI. This resulted in the appropriate use cases for the different methods where the MMAE method (bank of Kalman filters) was found to be more accurate in isolating faults and the parity space method was found

to be more sensitive to the occurrence of faults as stated in Section 1.1.3. The MMAE and parity space methods were evaluated in order to establish their suitability for integration with HyDE in Section 1.2.3. The MMAE method requires a bank of Kalman filters to be enumerated for each possible failure mode so practically this would imply, for the case of actuator locked-in-place faults, having to enumerate a filter for each actuator deflection interval. For this particular use case this would be impractical so the MMAE method was ruled out despite its superior filtering capability. The parity space method was also evaluated and it was found that the method is preferable since it has superior fault sensitivity and in this case fault detection is based on the difference between desired and actual actuator deflections. However the only disadvantage is its ability to provide a quantifiable measure of this difference which would assist with the decision process in HyDE without the use of residuals that encapsulate this information. So with this in mind the approximate input reconstruction algorithm was selected for integration with HyDE as it encompasses the superior sensitivity derived from it being based on the same foundation as the parity space method and it also provides an accessible means to measure the difference between desired and actual actuator deflections. This obviates the need to enumerate each possible interval magnitude for each actuator fault mode and also creates the opportunity to diagnose different kinds of faults. The main disadvantage of the approximate input reconstruction algorithm at this stage is its lack of filtering capabilities which is why filters are incorporated into the smart adapters interfaced with HyDE to leverage the advantages of having an alternate to the Kalman Filter.

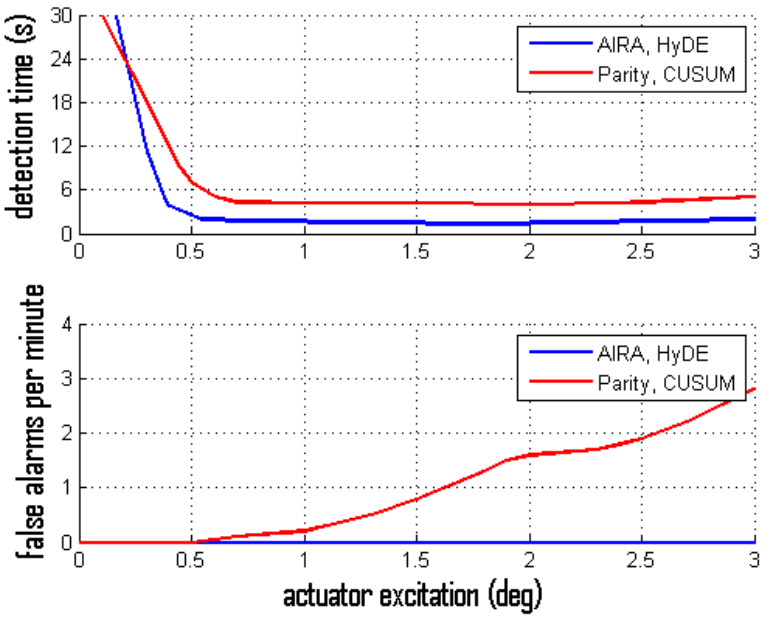


Figure 5.14: *Compared detection time and false alarms per minute due to actuator excitation, [38].*

When comparing the FDI performance due to actuator excitation for straight and level flight without substantial process noise, as shown in Figure 5.14, it is evident that the average detection time of HyDE is comparably lower than the optimised parity space method. The FDI indicator for the AIRA, HyDE system is calculated using the detection time due to actuator stuck faults at trim deflection. The Parity, CUSUM system has comparably superior fault detection performance for lower levels of actuator excitation below the $0.4°, 0.007\ rad$ threshold for the AIRA, HyDE system. The drastic increase in the AIRA, HyDE system detection time at lower levels of excitation is due to the slower aircraft response time and increasing cross correlation between the actual and desired control surface deflections, the average deflections become similar. We can expect the false alarms to increase at larger excitation values as the aircraft flies further away from trim since AIRA begins inducing increasing amounts of noise into the reconstructed signal. Given that AIRA is insensitive to the magnitude of actuator excitation the AIRA, HyDE system's false alarms per minute rate is consistent, within acceptable limits of conventional flight, along with a marginal

variation in detection time when compared with the variation in actuator excitation. This implies that the AIRA, HyDE system's FDI performance is relatively agnostic to actuator excitation when compared with the optimised parity space approach which has the worst performance when there is larger actuator excitation given that it is based on a method that is optimised for trim flight and the use of residuals which are sensitive to the magnitude of actuator excitation.

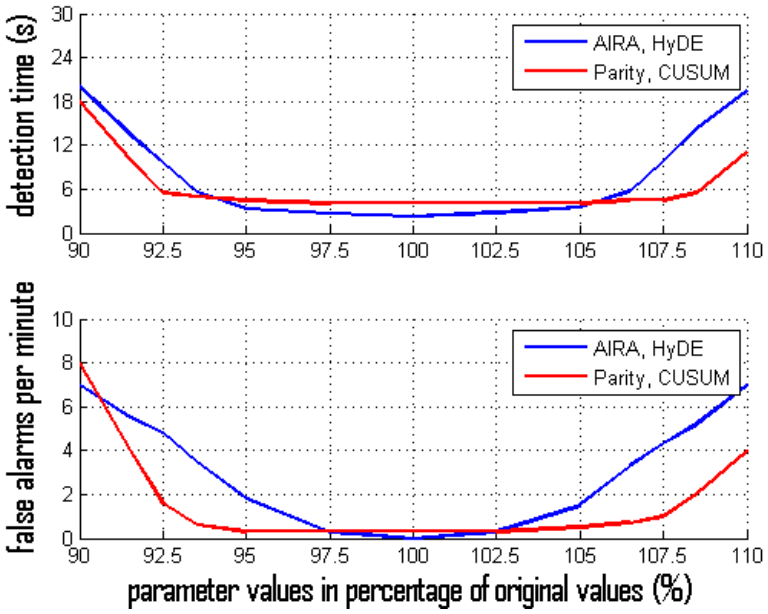


Figure 5.15: *Compared detection time and false alarms per minute due to varying parameters, [38].*

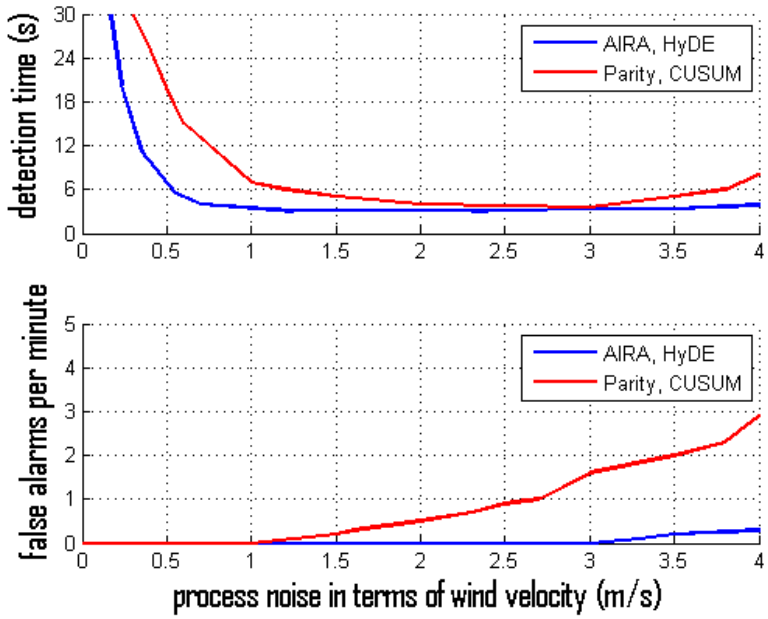


Figure 5.16: *Compared detection time and false alarms per minute due to process noise, [38].*

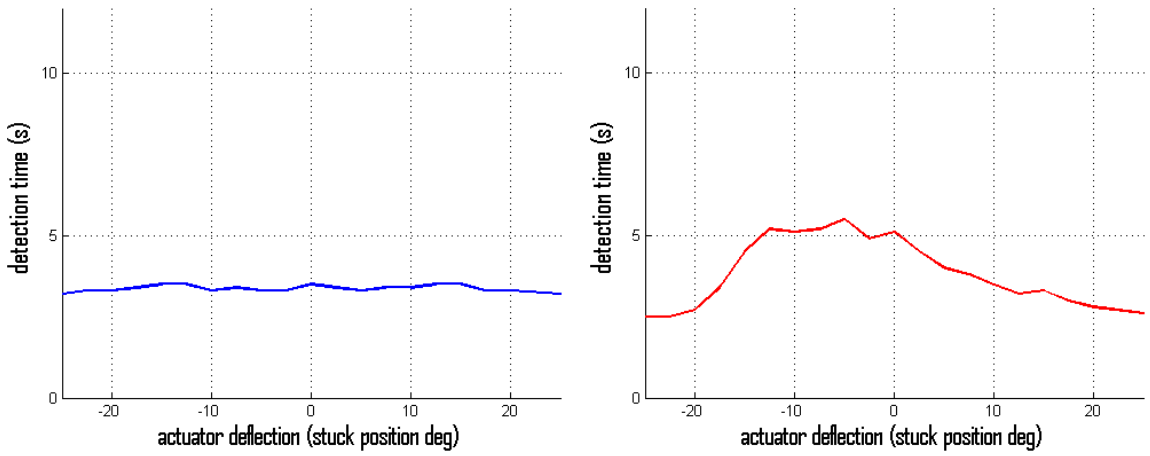
When compared with the optimised parity space method, it is evident that AIRA is more stringent in its requirements for accurate process parameters compared with the more flexible optimised parity space method. This is based on the more parabolic detection times and false alarms per minute rate as can be seen in Figure 5.15. The superior robustness to the variation in system parameters afforded by the parity space method is crucial for instances where there is uncertainty

concerning the system parameters to use. This added robustness is essential in cases where there are unmodelled dynamics and the models used are an approximation. The false alarm rates for both FDD methods also increase in a relatively similar manner. These results suggest that to optimise for the decrease in false alarms, focus should be directed toward tuning the system parameters as well as those of the diagnostic system itself.

Figure 5.16 shows how the presence of substantial process noise affects FDI performance. Once sufficient process noise exists, the actuators receive enough excitation to enable passive fault detection as reflected by the drastic decrease in fault detection time. It is clear that the FDI performance based on the parity space method degrades significantly with the increase in process noise. The AIRA, HyDE system however is relatively unaffected by changes in process noise at larger magnitudes. This can be attributed to use of low pass filters which filter out the additional noise before the reconstruction process compared to the filterless Parity, CUSUM fault diagnostic system. In addition the AIRA based FDI method's FDI performance improves at lower levels of process noise due to an increase in excitation and decrease in the cross correlation between the actual and desired deflections. The AIRA, HyDE system does not make direct use of residuals which are also the largest contributing factor to the varying performance of the Parity, CUSUM system. The false alarms per minute rate is expected to increase with increased process noise due to the lack of inherent filtering capabilities in AIRA, however with the added filters more acceptable FDI performance can be expected for the conventional actuator deflection range. These results suggest that the use of filters in the fault detection and diagnosis task is of paramount importance and may result in improved performance.

5.4.3 Associated Research

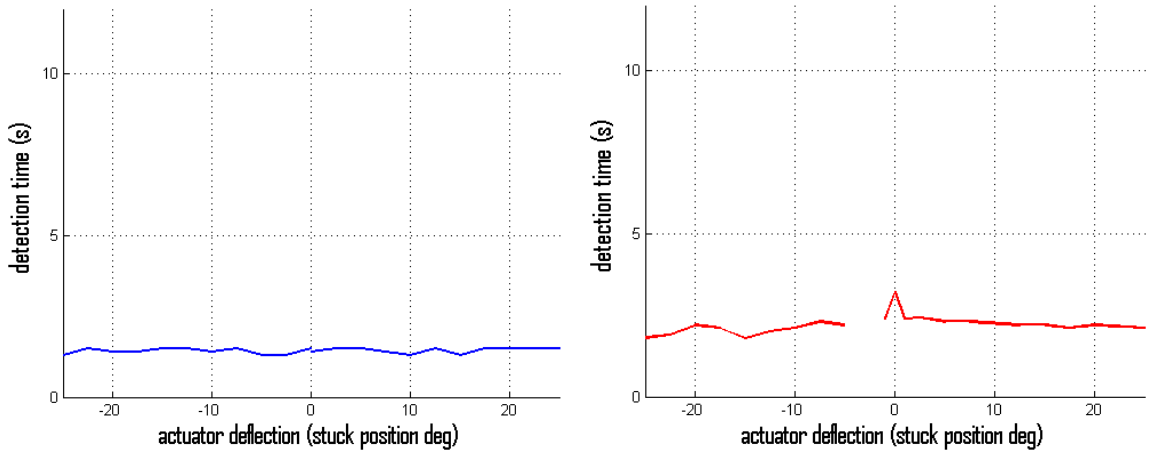
Having compared this research project's results with the results of the previous research project, we can move onto comparing the FDI performance with results produced in a similar research project [24]. Briefly investigating results produced independently is an effective way of getting an indication of the FDI performance of the fault diagnostic system. The aim is to get a qualitative indicator of the performance of the developed system given that the the comparison is not done using standardised FDI performance indicators and the testing conditions for the two research projects aren't identical. The research done by Dr. Redouane Hallouzi on Multiple-Model Based Diagnosis for Adaptive Fault-Tolerant Control contains comparable FDI performance indicators [24]. Focus is directed toward making a comparison between the fault detection times for the aileron, elevator and rudder faults with those of the associated research project with comparable testing conditions.



(a) *HyDE* (b) *MMAE adapted from [24]*

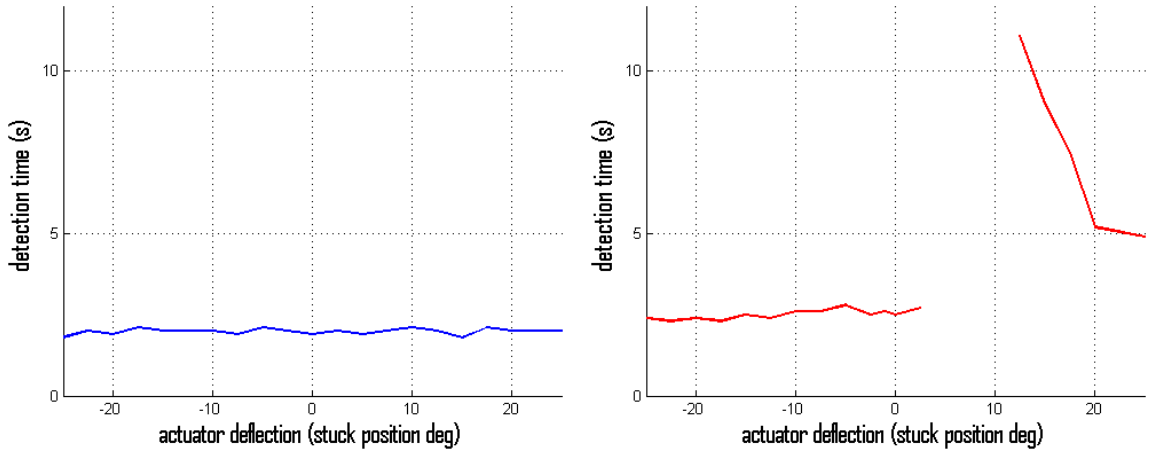
Figure 5.17: *Fault detection times for aileron input due to varying stuck positions.*

The detection times of the aileron stuck faults for HyDE and the MMAE based diagnosis appear to be comparable with the exception that the detection times for HyDE are lower as shown in Figure 5.17. Interestingly when it comes to the fault detection times for the elevators, the results are comparably lower for both diagnosis methods when compared with the aileron stuck faults as shown in Figures 5.17 and 5.18. Quite a few incorrect isolations occur for elevator stuck faults closer to the trim deflection angle, this is attributed to the lack of actuator excitation as is the case in this research project. The detection times for the rudder stuck faults are also quite similar as shown in Figure 5.19 and follow the trend where elevators have the fastest detection times followed by rudders and lastly ailerons. For the MMAE rudder detection times however, there appears to be a region where there are no faults detected as a consequence of insufficient additional excitation to enable effective fault detection.



(a) *HyDE* (b) *MMAE adapted from [24]*

Figure 5.18: *Fault detection times for elevator input due to varying stuck positions.*



(a) *HyDE* (b) *MMAE adapted from [24]*

Figure 5.19: *Fault detection times for rudder input due to varying stuck positions.*

Tables 5.9 and 5.10 contain a summary of the FDI performance indicators for the HyDE based fault diagnosis system and the MMAE based fault diagnosis method. It is apparent that the fault detection rates of the two methods for the stuck in place faults are comparable with HyDE's fault detection rates being higher and fault detection times being faster. It's worthwhile mentioning that HyDE has higher false alarm rates compared to the MMAE method and this can be attributed to the effects of wind in the aircraft's ambient environment contributing to added process noise.

Property	Elevator Stuck	Aileron Stuck	Rudder Stuck
Valid Isolation	97.2%	95.7%	94.3%
Missed Alarm	0%	0%	0%
False Alarm	0.3%	0.2%	0%
Time delay (s)	1.5	3.5	2

Table 5.9: *Summary of HyDE isolation results.*

Rate	Elevator Stuck	Aileron Stuck	Rudder Stuck
Valid Isolation	91.4%	93.4%	75.2%
Missed Alarm	0%	0%	21.8%
False Alarm	0%	0%	0%
Time delay (s)	2	5	3.5

Table 5.10: *Summary of MMAE isolation results sourced from [24].*

5.4.4 Out of Scope Investigation

An important aspect of the development of a fault diagnostic system to consider is the improvement of the system's ability to track system behaviour accurately and provide valid diagnoses without undue development effort.

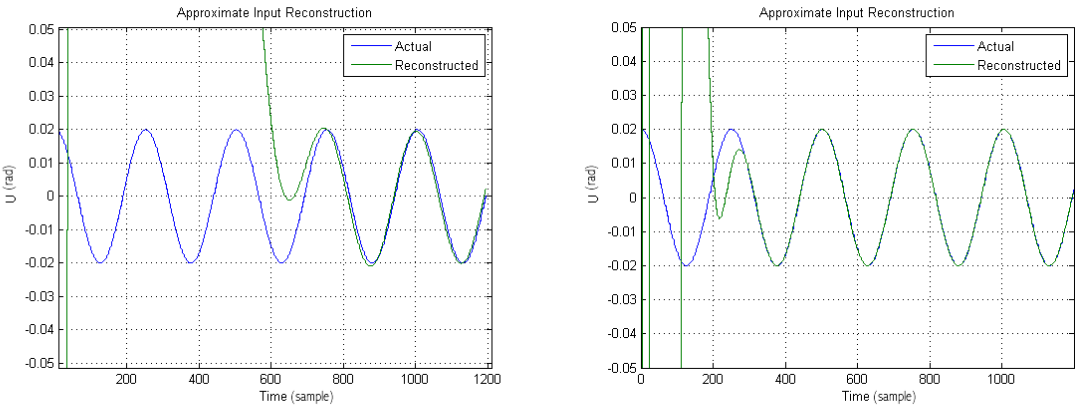
Control	Sensitive Variables					
Aileron	β	p	r	ϕ	φ	E
Flaperon	β	p	r	ϕ	φ	E
Rudder	β	p	r	ϕ	φ	E
Elevator	α	q	$\bar{v}$	θ	D	N
Engine 1	p	q	r	α	β	θ
Engine 2	p	q	r	α	β	θ

Table 5.11: *Sensitive state variables for control inputs adapted from [19].*

In developing the declarative model for the UAV's actuators, the first element that had to be identified is the causal relationship between the inputs and the observable outputs. In this research, the observable outputs are the states of the decoupled aircraft dynamics. Table 5.11 shows the state variables that are sensitive to changes of the control inputs to the aircraft. The highlighted cells show the state variable used in the implementation of the smart adapters. For the rudder control input, three of the states are highlighted which implies that for a smart adapter with acceptable FDI performance, additional points of observation were required. As an example of this principle, Figure 5.20 shows the reconstructed rudder inputs before and after an additional variable was incorporated. In general this principle applies to other subsystems and future improvements to the fault diagnostic system will need to take this principle into account during the design process.

5.5 System Level Reasoning

The developed Fault Diagnostic System's ability to reason about the system is an important aspect to investigate given that HyDE is a hybrid diagnosis engine. The aim of this research project is to develop a generalised fault diagnosis system that is housed in an expandable architecture. Up until this point, we have performed benchmark tests to evaluate the system's performance at the component level. We now move on to investigating some of the emergent properties exhibited by a system that simulates components that can be influenced by one another other.



(a) *Two state variables: p, r*

(b) *Three state variables: β, p, ϕ*

Figure 5.20: *Effect of incorporating additional points of observation in rudder input reconstruction.*

5.5.1 Mode Evaluation

	Fault Diagnostic System		
Function	Kalman, Bayes	Parity, CUSUM	AIRA, HyDE
Multiple Simultaneous Faults	-	-	√
Actuator Isolation (Left/Right)	√	√	x
Aileron Control Surfaces	√	√	√
Elevator Control Surfaces	√	√	√
Flaperon Control Surfaces	√	√	√
Rudder Control Surfaces	√	√	√
Actuator Nominal Mode	√	x	√
Actuator Stuck Faults	√	√	√
Actuator Hardover Faults			√
Actuator Partial Faults			√
Actuator Floating Faults			√
Actuator Unknown Faults			√
Actuator Off			√
Avionics Battery			√
Backup Battery			√
Battery Isolation			√
Battery Nominal Mode			√
Battery Degraded Fault			√
Battery Discharged Fault			√
Battery Unknown Fault			√
RC Receiver On Mode			√
RC Receiver Off Mode			√
RC Receiver Out of Range Fault			√
RC Receiver System Failure Fault			√
Servo Board On Mode			√
Servo Board Off Mode			√
Servo Board Miscalibrated Fault			√
Servo Board System Failure Fault			√

Table 5.12: *Implemented features of the FTC Group's relevant Fault Diagnostic Systems [38], key ✓ implemented, x not implemented, - not investigated.*

Implemented features of the FTC Group's relevant Fault Diagnostic Systems \cite{odendaal2012}, key √ implemented, x not implemented, - not investigated.

Table 5.12 contains a listing of all the relevant fault detection and diagnosis methods investigated by the Fault Tolerant Control Group in the ESL at Stellenbosch University thus far. The table lists the functionality implemented in each self contained system and also serves to suggest the progress of the research toward developing a generalised Fault Diagnostic System for the Meraka Modular UAV. The Parity Space Method and AIRA are compared in this research project in order to identify the relative strengths and weaknesses of each of the fault detection methods. Table 5.12 aims to highlight the functionality enabled by each of the developed Fault Diagnostic Systems. The initial systems focussed predominantly on actuator fault detection where as with this research, focus is directed toward system development and the migration of existing fault detection methods into a generalised system. Evaluating this generalised system requires an awareness and understanding of each of the modes of operation listed in Table 5.12. One of the ways available to do this is a Cause and Effects Matrix.

5.5.2 Cause and Effects Matrix

Cause and Effects Matrix									Rank
I/O	a_bat	b_bat	rc	board	aile	elev	flap	rudd	-
a_bat	√		√	√					3
b_bat		√	√	√					3
rc			√	√					2
board				√	√	√	√	√	5
aile					√				1
elev						√			1
flap							√		1
rudd								√	1

Table 5.13: *Cause and Effects Matrix, outlining functional relationships between components.*

Reasoning about a hybrid system with several subsystem components is a challenging task. This is due to the complexities inherent in tracking the behaviour evolution of each component and determining alternative trajectories in a system's modes and states. Evaluating a FDD method in order to determine it's capability to reason is not a straight forward task due to the multitude of permutations of modes of operation a interconnected system gives rise to.

A simplified cause and effects matrix, shown in Table 5.13, is used to give a functional overview of the 'reasoning space' for the system that has been modelled in HyDE based on comparable work done in [23]. Each of the major components of interest are represented in the table where *a_bat* is the Avionics Battery, *b_bat* is the Backup Battery, *rc* is the RC Receiver, *board* is the Servo Board, *aile* represents the aileron control surfaces, *elev* represents the elevators control surfaces, *flap* represents the flaperon control surfaces and *rudd* represents the rudder control surfaces. The cause and effects matrix has a list of all the inputs on the left hand side column and a list of all the corresponding outputs on the top row. The highlighted cells correspond to functional input output relationships that have been identified between components based on the Characterised Subsystem Component Model generated in Section 3.3. The green diagonal entries in the matrix correspond to the component level diagnoses. These are equivalent to a FDD method that can only reason about a component individually without taking into consideration other components that may influence the component's behaviour. The cells that are in blue in the matrix correspond to the output components influenced directly by the input components, this functionality wouldn't be possible without HyDE being able to model the interconnection of components at the system and subsystem level. When changes in the input components occur the effects cascade into the output components as well which then influence their output components until the effect has propagated to all the relevant components in the system. The right most column lists the ranking of the

respective component based on the number of components the input component affects directly. The from these results, it is clear that the servo board followed by the batteries are the most important components onboard the UAV. We use this understanding to prioritise the generation of scenarios that we can use to determine how well the UAV's subsystems are modelled and how accurate HyDE is at reasoning about the state of the system.

5.5.3 Hybrid Diagnosis

The results generated during the benchmark tests are reused to shed some light into how HyDE has been reasoning about the system. Table 5.14 contains a list of the diagnoses requested from HyDE in response to the fault scenarios injected into the system during benchmark testing that has been done thus far. In addition to the mode of interest a list has been given of all other components and their locations except the other actuators since they are similar.

Hybrid Diagnosis					
Mode	Diagnosis	Description	t_d	t_i	Valid
2.2.1	actuation.servo_board.miscalibrated aircraft.avionics_battery.nominal aircraft.backup_battery.nominal actuation.rc_receiver.on actuation.left_elevator.nominal	Servo board miscalibrated	0.2	0.5	Yes
1.1.1	aircraft.avionics_battery.degraded aircraft.backup_battery.nominal actuation.rc_receiver.on actuation.servo_board.on actuation.left_elevator.nominal	Avionics battery degraded	0.2	0.5	Yes
1.2.2	aircraft.backup_battery.discharged aircraft.avionics_battery.nominal actuation.rc_receiver.off actuation.servo_board.off actuation.left_elevator.off	Backup battery discharged	0.2	0.5	Yes
2.1.1	actuation.rc_receiver.out_range aircraft.avionics_battery.nominal aircraft.backup_battery.nominal actuation.servo_board.unknown actuation.left_elevator.unknown	RC receiver out of range	0.2	1.0	Yes

Table 5.14: *Hybrid Diagnosis outlining system level reasoning to changes in modes of operation, where t_d is the detection time and t_i is the total fault detection and isolation time.*

What we would like to see is the consistency in the modes of operation of the subsystem components. The entire system is reset to nominal state before, and after faults are injected into the system. When the Servo Board is miscalibrated we expect that the rest of the system is relatively unaffected. This is confirmed by the observation that each of the components are still either nominal or on. For the second test, we expect that the avionics battery degrading shouldn't affect the rest of the system and this is confirmed by observing that all the components are either on or nominal at this point in time. If however the backup battery becomes discharged then we expected nothing to happen to the disconnected avionics battery and the rest of the system to switch off. This occurs as anticipated given that each of the components in the off mode of operation are constrained to only operating when there is a specified voltage. When the RC receiver is out of range of the transmitter, we expect that the batteries will remain unaffected and for the servo board and actuators to specify that unexpected behaviour has been encountered. This occurs as expected given that no provisions have been made to model what the components should do if for some reason the RC receiver stops working thereby preventing it from sending control inputs to the servo board and actuators.

5.6 Overview

In this chapter the test conditions are defined followed by a description of how the developed fault diagnostic system is tested extensively by deliberately failing the UAV's actuators during simulated flight. This begins by defining the test conditions and by describing the link between the theoretical model of the UAV and the physical aircraft. Focus is placed on specifying the flight trajectory followed by the aircraft and the various hypotheses that need to be tested and the performance measures that can be used.

The effectiveness of the input reconstruction algorithm is evaluated using simulation and flight based hypothesis testing. This is followed by an analysis of the simulation and flight test results and the FDI performance of the fault diagnostic system. The performance of the system is then verified by using flight test data collected by the Meraka Modular UAV during flight missions.

An evaluation of the simulation and flight test results is done as a crucial part of determining the effectiveness of the design and development process. Furthermore, comparing the results produced in this research project with the research project that directly precedes it also serves as a means of gauging the progress that has been made. Lastly, investigating independently produced results is an effective way to get an indication of the FDI performance of the fault diagnostic system. It is discovered that the incorporation of additional points of observation enhances the FDI performance of the fault diagnostic system.

The developed fault diagnostic system's ability to reason at the system level is investigated. This investigation begins with an evaluation of the implemented features of the FTC Group's relevant fault diagnostic systems and diagnosable modes of operation. This is followed by the use of a cause and effects matrix which is used to give a functional overview of the reasoning space of the system modelled in HyDE. Thereafter an evaluation of the fault diagnostic system's ability to perform hybrid diagnosis is done to confirm that it performs as intended.

Chapter 6

Conclusion

6.1 Conclusion

The aim of this research is to develop a generalised fault diagnosis system that is housed in an expandable architecture as stated in Section 1.1.2. The research scope is more clearly outlined in the Diagnostic System's Design and Development Specification in Appendix A. All of the research's requirements and most basic objectives are met as outlined in Sections A.3 and A.2 respectively. These include basic objectives concerning the development and integration of an expandable FDIR architecture and optional extras such as diagnosing onboard power and embedded hardware; and the implementation of new FDIR methods.

One of the core capabilities of HyDE is its ability to allow the control engineer to model systems using abstracted concepts such as components, locations, transitions, commands, variables and constraints with added support for logical and first order differential equations. Performing a Failure Mode and Effects Analysis goes a long way in formalising the process of inferring the various faults and defining the diagnostic scope of the declarative models. Using the FMEA and FTA the diagnostic scope was limited to Category A and B end event effects resulting in uncontrolled and controlled emergency landings.

The main difference with HyDE when compared to other diagnostic systems is based on its diagnostic model being predictive and on its model based reasoning approach. Many diagnostic systems start with sensor data and classify it where as with HyDE, both the nominal and faulty system behaviour is simulated. In addition with the use of the approximate input reconstruction algorithm, HyDE's flexible modelling language, extensible architecture and conflict directed search algorithm contribute towards the Diagnostic System's enhanced reasoning capabilities.

This research has resulted in a larger number and type of subsystem components on the Meraka Modular UAV that can be diagnosed and include the left and right elevators, rudders and ailerons; the RC receiver and servo board, and lastly the avionics and backup batteries. The isolation time for each component varied and it was found that the average detection time of 4 seconds for ailerons, 2.5 for rudders and 2 for elevators can be decreased by increasing HyDE's minimum candidate probability parameter for scenarios which results in less CPU time spent searching for candidates.

The fault diagnostic system was tested extensively by deliberately failing the UAV's actuators during simulated flight. The performance of the system was then verified by using flight test data collected by the Meraka Modular UAV during flight missions. When compared with the associated FDIR research, the FDI performance of the fault diagnostic system is found to be relatively agnostic to actuator excitation. For the Meraka Modular UAV in particular, it is found that the fault diagnostic system performs as expected in the presence of disturbances and noise and improvements can be made by incorporating additional points of observation.

6.2 Contributions

More states were incorporated into the state space matrices that describe the Meraka Modular UAV using decoupled aircraft dynamics. An innovative system inversion technique was investigated and implemented in order to reconstruct inputs that correspond to actuator deflections. A new design method for applying the approximate input reconstruction algorithm (AIRA) to the linearised aircraft model was also described. Through investigation, it was found that modulating the inputs with sinusoidal signals improves input reconstruction for nonlinear aircraft models using SISO state space representations.

HyDE, an innovative generic framework for stochastic model based reasoning developed at NASA along with a real time version of AIRA was integrated in an expandable FDIR architecture based on the Model B+ Raspberry Pi which is the first for HyDE. The ESL's Ground Station was modified to include a diagnostics panel that allows the ground station operator to visually monitor the changes in the subsystem components and faults that occur onboard the Meraka Modular UAV in real time.

A Reliability Analysis framework based on the Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) was introduced into the design and development process to allow subsystems to be categorised. Future research can now be done towards ensuring that negative effects and the possible occurrence of end effect events such as uncontrolled emergency landings, controlled emergency landings and failures in flight missions are minimised.

As a consequence of this research, additional failure modes have been incorporated into the FDIR architecture being developed. It is now possible to not only detect stuck faults but also hardcover, partial loss of effectiveness, floating and the nominal mode of operation for actuators. In addition, subsystem components such as the batteries, RC receiver and servo board can now be diagnosed.

6.3 Recommendations

The implementation of additional optional project objectives that weren't met in this research project is recommended. This would give the UAV the ability to detect and diagnose plant faults, sensor, rotor (engine) problems and OBC problems in addition to tracking the behaviour of the actuators, RC receiver, servo board and onboard batteries.

Setting up a maintained and standardised HIL based testbed environment, with a comprehensive fault diagnostics test suite, for UAV's that is similar to the Advanced Diagnostics and Prognostics Testbed (ADAPT) at NASA Ames Research Center is recommended. This testbed would be fully equipped with sensors, OBC, servo board, RF receiver and HIL Board and would streamline the development and testing of future diagnostics platforms. The purchase of updated versions of the NPRD and FMD Handbooks, and commercially available HyDE software for MATLAB will provide a more feature rich toolset that will further enhance the design and development process for future systems.

The design and development of an independent, standardised and rigorously tested fault diagnostics platform that incorporates the design elements of a supervised active control system, based on this research for the vehicles in the ESL is recommended. The standardisation of the fault diagnostic system's housing and the interface between fault diagnostic system and the vehicles in the ESL is also recommended. This would make using the same diagnostic platform in different vehicles less cumbersome when accompanied by a specification document outlining the procedures to be followed when integrating the fault diagnostics platform in other vehicles. Designing and implementing a closed loop control system for the Meraka Modular UAV and the reapplication of the input reconstruction algorithm will result in more accurate input reconstruction. This results as a consequence of using a direct causal relationship between the control inputs and measured

output variables. Investigating and implementing approaches to enable the nonlinear operation of the parity space method can also add to resolving AIRA's inability to operate effectively outside trim. This would enable the fault diagnostic system to reliably operate in a larger range of aircraft manoeuvres and in parts of the aircraft's flight mission that don't include straight and level flight considering that these other parts contain critical stages such as Autonomous Take-off and Landing (ATOL).

The incorporation of active FDI for critical FDI tasks such as left and right aileron detection is recommended. The incorporation of more points of observations in future research is of paramount importance especially for cases where these observations can be derived from the subsystems electronically without the need for additional hardware. This will further add to the diagnostic system's ability to reason about the state of the system and track the behaviour evolution of the subsystem components through time.

Appendix A

System Development

A.1 Project Aim

The aim of this project is to allow an autonomous Unmanned Aircraft to be able to automatically diagnose faults or significant changes in the behaviour of on-board sub-systems and if possible it should be able to automatically reconfigure the aircraft to maintain stable flight control using fault-free sub-systems.

A.2 Research Objectives

The desired research output for this project includes the development of the following:

1. A Fault Diagnosis and Reconfiguration System - which consists of:
    - creating a FDIR engine structure that is expandable
    - combining the chain of fault detection and isolation with reallocation technologies
    - demonstrating this process using a system that can be embedded on board the Meraka Modular UAV.
2. Optionally, a Generalised FDIR System - which consists of:
    - extending the detected fault types, in addition to actuator (control surface) faults, to include:
        - plant faults
        - sensor problems
        - rotor (engine) problems
        - on-board power (battery) problems
        - embedded hardware (OBC - on-board computer) problems
        - all sorts of other things in more of a general path such as the embedded software.
    - the implementation of new FDIR methods including for example:
        - the combination of active and passive fault detection
        - new FDIR methods that might be applicable to the various fault types
        - the reconfiguration of on-board sub-systems e.g. power, OBC hardware, sensors etc.

A.3 Research Requirements

The following project requirements have been identified as per the problem statement and project proposal:

- draw from, expand and improve upon associated research performed in the ESL:
  - development of a working model for the autonomous aircraft system and FDIR methods
  - identification and modelling of the fault types to be diagnosed in the developed FDIR system
- combine fault detection and isolation research with the reconfiguration technologies already demonstrated:
  - development of an expandable architecture for the integration of FDIR technology
  - implementation of integrated FDIR technology within a simulation environment
- design, test and demonstrate a system capable of running on-board the CSIR DPSS Modular UAV (Autonomous Unmanned Aircraft) that allows it to:
  - automatically diagnose faults or significant changes in the behaviour of on-board sub-systems
  - automatically reconfigure the aircraft to maintain stable flight control using fault-free sub-systems.

A.4 Development Approach

• Model the aircraft and system faults - The aircraft system must be modelled at this stage along with the fault modes for the existing sub-systems. Having done this, it will be possible to create a simulation of the system in order to verify the correct operation of the FDIR System to be developed. The existing aircraft system also must be analysed to ensure that the aircraft control system and estimator are in a working condition in order to avoid creating a faulty system model.

• Implement FDIR methods selected - Selected FDIR methods will need to be implemented separately and tested before they are integrated within the larger FDIR system. In this way the efficacy of each method will be verified and in doing so, the pathways available for the integration of the methods will become apparent which will then allow for a clearer overview of each method's functionality.

• Develop an expandable IVHM System architecture - The developed Diagnosis System will be comprised of varying FDIR components which then work together in order to produce the intended functionality. It is therefore essential that an expandable architecture be developed which gives the system the flexibility for components to be added as needed.

• Integrate the FDIR technology - The FDIR technology then needs to be integrated within the developed IVHM System using the expandable system architecture. Once the technology has been integrated, it can then be simulated before being implemented practically on-board the aircraft.

• Get the existing aircraft system working - The existing aircraft system must be restored to working condition before further work can be done. This will ensure that any faults or problems with the existing system can be resolved before attempting to make modifications.

• System Implementation - The developed IVHM System along with the integrated FDIR technology will then be practically implemented on-board the Meraka Modular UAV and associated aircraft system components. This will include the design and development of the actual hardware to be used or the augmentation of existing components which will allow system implementation.

• Simulate the integrated FDIR technology - The developed system will need to be simulated initially via a SIL (Software in the Loop) Simulation in order to verify that the developed system

performs as expected and to remove any problems early in the development of the system. This can be followed by a HIL (Hardware in the Loop) Simulation in order to verify that the developed system will perform as expected when implemented on the aircraft system's hardware.

- Perform a flight test - The flight test will be used to practically demonstrate the functionality of the developed system. Here the various fault conditions will be triggered artificially under trim conditions and the data generated will be used to verify the performance of the developed system.

A.5 Criteria for Success

The success of the project is rests on the level of completion of the following outcomes:

- Successfully modelling the fault modes of the Meraka Modular UAV using applicable FDIR methods that allow the aircraft to diagnose faults.
- The successful combination of FDIR research and the development of an expandable architecture that can be used as part of an accurate simulation of the aircraft.
- The implementation of a system capable of running on board the Meraka Modular UAV and a successful flight-test that verifies the autonomous fault diagnostic functionality of the aircraft.

These loosely defined criteria for successful implementation, based on the research requirements, can be used to define the preliminary design specifications for the project.

A.6 Research Phases

The following phases for the execution of the project have been identified given the problem statement and the proposed solution.

Project Phases:

1. Initiation Phase
2. Definition Phase
3. Design and Development (Prototype) Phase
4. Documentation Phase
5. Presentation and Delivery Phase
6. Termination Phase

The project phases were then used to guide development approach that would be taken in order to complete the project successfully as shown in Figure A.1. This begins with the project proposal followed by literature study, which includes course work. The development of the prototype requires modelling the system and the respective fault modes, followed by the research and integration of FDIR technology that is then used to develop the desired diagnostic system. The performance of this system is then tested practically before writing the project documentation.

A.7 Preliminary Design Specifications

These preliminary design specifications were derived from the project requirements and define the communication, control, autonomy, estimation, environment and safety of the system to be developed.

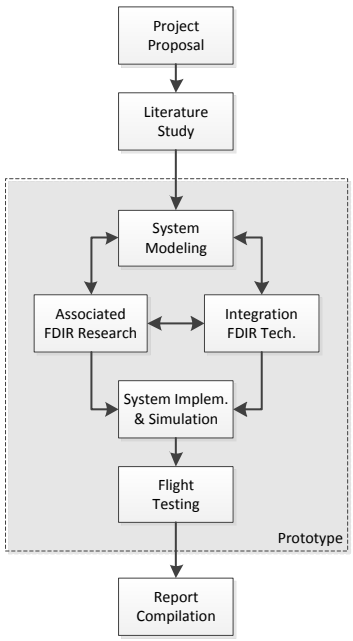


Figure A.1: *Research Project Decomposition*

A.7.1 Autonomy Requirements

The following autonomy requirements where identified to ensure that the aircraft performs autonomously:

- The system must be able to autonomously perform fault detection, isolation and if possible reconfiguration.
- It is intended for the system to be autonomous in nature therefore it is required that it be able to maintain stable flight and track given paths and follow instructions.
- We assume that there are no obstacles in the aircraft's flight path so it wont have to perform collision avoidance.

A.7.2 Communication Requirements

The following communication requirements were identified to ensure operability between the aircraft and the ground station:

- The ground station should send and receive data relating to the aircraft's state, position and fault conditions.
- The aircraft should be able to send and receive instructions and data from the ground station and communicate with the on-board components as necessary.

A.7.3 Control Requirements

In order to ensure accurate control of the aircraft, the following requirements where identified:

- It is important that the aircraft be able to reduce the error between the output and its reference as accurately as possible.
- The aircraft's bandwidth must be chosen so that it allows for realistic performance while maintaining robustness to modelling error.
- The aircraft's control system must be able to maintain effective control of the aircraft whilst performing all related activity on board.

A.7.4 Environment Considerations

The following needs to be considered for the aircraft's testing and operating environment:

- It is assumed that the aircraft will be high enough above the ground and clear from any obstacles that may collide with it.
- Wind disturbances and related weather conditions are likely to have a significant impact on the aircraft's performance. It can be assumed that these effects can be modelled as process noise and disturbances.

A.7.5 Estimation Requirements

The following estimation requirements where identified for the project:

- The estimator dynamics can be assumed to be much faster than the controller dynamics.
- It is assumed that estimation of the aircraft's states will be available during all control updates and fairly accurate to avoid undesirable control effects.

A.7.6 Safety Requirements

The safety of the system is of paramount importance therefore the following requirements where identified:

- The aircraft will not attempt to simulate injected faults beyond a specified time in order to prevent the likelihood of damage to the system resulting from the faults simulated.
- Should the aircraft's autopilot malfunction, the land pilot will always have the ability to resume full control of the aircraft and safely land it.
- For safety requirements, it is assumed that the aircraft will not allow the on-board power systems or critical aircraft sub-systems to be disengaged at any stage, even as part of reconfiguration procedures.

A.8 System Architecture

It is preferable for the physical implementation of the system to not be state dependent therefore; the Figure A.2 describes the state independent system. The diagnostic system will be the main focus of the work to be done given that the existing systems should be in working order. The operator input is the only source of input to the autonomous system that governs its operation.

A.9 Software Architecture

The following software architecture governs the operation of the developed system.

A.9.1 System Interface

The following operator input can be given to the system as instructions to guide its state transitions:

1. Guidance: Causes the aircraft to disengage control of the aircraft and await further instruction to either receive commands or execute a flight mission.
2. Mission: This instruction is used to cause the aircraft to execute its flight mission by tracking waypoints at trim conditions.

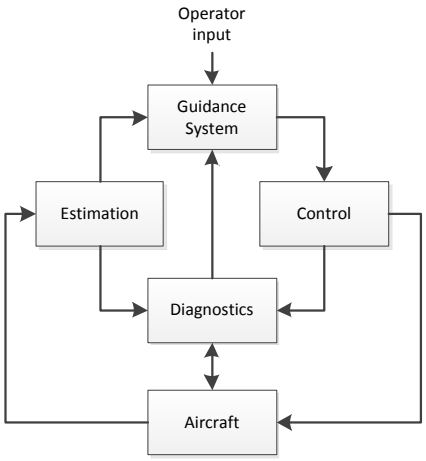


Figure A.2: *Autonomous Aircraft System Architecture*

3. Probe: Instructs the Guidance System to inject faults into the aircraft, which can be used to simulate faults.

4. Suspend: This instruction commands the aircraft to suspend or resume any activity that it is currently performing which would lead to it remaining in its current state.

5. Restore: Restores the aircraft to nominal operating conditions by returning the aircraft to nominal free fault conditions.

6. Command: Issues the instruction that causes the aircraft to follow and maintain given position or translation requests.

A.9.2 System States

The overall states of the aircraft can be described as follows:

0. Disengage: At this state, the system is waits to receive instruction to either follow given commands or execute the flight mission stored on-board the flight computer.

1. Flight mission: At this stage the aircraft's guidance system maintains control of the aircraft and guides it along a specified flight trajectory. This will then simulate normal aircraft behaviour at trim that then allows for the simulation of the occurrence of faults by injecting faults into the system.

2. Monitor aircraft behaviour: The aircraft's behaviour is continuously monitored at this stage in order to detect any simulated faults. This will be done while the aircraft continues normal operation and any faults detected will then need to be isolated.

3. Perform fault isolation: After the detection of a fault, the FDIR system will attempt to determine what has caused the fault by isolating the root cause using the available sensor data and system model.

4. Reconfigure aircraft: After the fault isolation, the system will attempt to reconfigure aircraft based on the fault and the available fault mitigation procedures needed in order to restore nominal flight operation using the existing fault free subsystems.

5. Maintain stable flight: At this state the FDIR system will use the new system configuration as the new model for the aircraft in order to maintain stable flight. The flight mission will then proceed until such a time that there is a significant change in the aircraft's behaviour or until instructions are received to disengage the system or follow commands.

6. Command: At this state, the system receives commands that can be used to verify the correct operation of the on-board hardware and software.

A.9.3 State Transitions

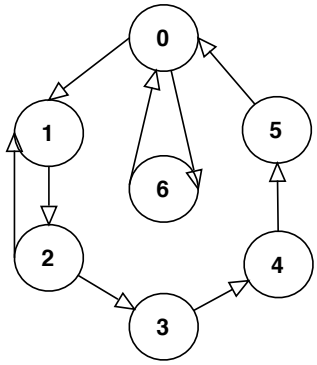


Figure A.3: *System state transitions*

The states described above are linked by the autonomous structure shown in Figure A.3. These lines suggest the flow of control of the aircraft's systems where transitions are governed by commands given or the states themselves.

A.10 Issues, Constraints and Limitations

There are potential issues and limitations that must be noted as these may affect the overall system, hardware and software required for the successful completion of this project.

A.10.1 System Overview

Since this is a continuation of the work done by others in the ESL, any prior work that is fundamental to the operation of the aircraft and associated software must be verified to ensure that the system will perform as expected.

A.10.2 Developed Hardware

Since this project requires a flight test or using flight test data, the existing hardware must also be restored to a working condition. It is also important to note that there is only one aircraft and with any damage to the aircraft, time penalties will be incurred to fix it, which will ultimately affect the schedule.

A.10.3 Developed Software

It is assumed that the ground station software along with the aircraft embedded software is in working order. This will need to be verified before further work on the aircraft can proceed along with the guidance system. The SIL simulation software along with the development software for the existing hardware will need to be reintegrated and tested before the aircraft and its components can be used and tested.

A.10.4 Engine Selection

In addition to those already identified, the following issues were taken into consideration when determining the suitability of the diagnosis engine: it supports multiple modelling paradigms, has near realtime processing, handles time delays in components, handles stochastic reasoning, models

component behaviour, incorporates models for noise, can be accessed via an API, incorporates Failure Mode Effects Analysis (FMEA), can be executed on an embedded system, is open source or freely available, runs on Windows / Linux / RTOS, enables varied levels of abstraction, diagnoses discrete and continuous behaviour, diagnoses multiple faults, uses model based reasoning.

Appendix B

Meraka Modular UAV

B.1 Aircraft Parameters

The aircraft parameters and the moment of inertia tensor were determined using a three dimensional geometric model of the Meraka Modular UAV in Autodesk Inventor.

	Parameter	Value
m	Mass	26.0kg
$\bar{c}$	Mean aerodynamic chord	0.36m
e	Oswald efficiency factor	0.85
A	Wing aspect ratio	11.11
S	Wing reference area	$1.44m^2$
b	Wing span	4.0m

Table B.1: *Aircraft Parameters*

$$\boldsymbol{I}_b = \begin{bmatrix} 12.18 & 0 & 0 \\ 0 & 5.86 & 0 \\ 0 & 0 & 17.29 \end{bmatrix} kg.m^2 \tag{B.1}$$

B.2 Aerodynamic Coefficients

These aerodynamic coefficients were calculated with AVL Software using a simulation of the aircraft at a trim velocity of 22 m/s and a positive angle of attack. The control derivatives are used in the actuator vector $\boldsymbol{\delta}$ during calculations of the non-dimensional aerodynamic force and moment coefficients. The symbols on the left specifically denote the subscripts of the coefficients above each column in Tables B.2 and B.3.

	C_D	C_L	C_Y	C_l	C_m	C_n
0	0.06	0.5			-0.05	
α		5.557928			-1.069455	
β			-0.389444	-0.071508		0.102214
p			0.049295	-0.621899		-0.063578
q		8.046991			-18.442581	
r			0.244026	-0.085316		-0.085316

Table B.2: *Stability Derivatives*

Actuator	$\boldsymbol{C_{L_\delta}}$	$\boldsymbol{C_{Y_\delta}}$	$\boldsymbol{C_{l_\delta}}$	$\boldsymbol{C_{m_\delta}}$	$\boldsymbol{C_{n_\delta}}$
A_l	-0.47515	-0.009786	-0.16364	0.062452	0.005796
A_r	0.47515	-0.009786	-0.16364	-0.062452	0.005796
E_l	0.17624	-0.028361	0.0072193	-0.6157	0.0092819
E_r	0.17624	0.028361	-0.0072193	-0.6157	-0.0092819
F_l	0.59232	-0.010199	0.11539	-0.065031	0.003495
F_r	0.59232	0.010199	-0.11539	-0.065031	-0.003495
R_l	-0.3856	0.10766	0.0029221	0.13189	-0.035695
R_r	0.3865	0.10766	0.0029221	-0.13189	-0.035695

Table B.3: *Control Derivatives*

Appendix C

Linearised Dynamics

C.1 State Matrices

The following state space matrices encompass the linearised dynamics of the aircraft at trim. Modifications that have been made to the state space matrices have been based on the simplification assumptions listed in Chapter 2.

$$\boldsymbol{\Delta\dot{x}} = \boldsymbol{A}_T\boldsymbol{\Delta x} + \boldsymbol{B}_T\boldsymbol{\Delta u} \tag{C.1}$$

$$\begin{bmatrix} \boldsymbol{\Delta\dot{x}}_{long} \\ \boldsymbol{\Delta\dot{x}}_{lat} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{long} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_{lat} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta x}_{long} \\ \boldsymbol{\Delta x}_{lat} \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}_{long} \\ \boldsymbol{B}_{lat} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_{long} \\ \boldsymbol{u}_{lat} \end{bmatrix} \tag{C.2}$$

The modifications made allow the aircraft's dynamics to be decoupled into longitudinal and lateral dynamics.

$$\begin{aligned} \boldsymbol{\Delta\dot{x}}_{long} = &\begin{bmatrix} \frac{\partial\dot{U}}{\partial U} & V_T\frac{\partial\dot{U}}{\partial W} & \frac{\partial\dot{V}}{\partial Q} & \frac{\partial\dot{U}}{\partial\theta} \\ \frac{1}{V_T}\frac{\partial\dot{W}}{\partial U} & \frac{\partial\dot{W}}{\partial W} & \frac{1}{V_T}\frac{\partial\dot{W}}{\partial Q} & \frac{1}{V_T}\frac{\partial\dot{W}}{\partial\theta} \\ \frac{\partial\dot{Q}}{\partial U} & V_T\frac{\partial\dot{Q}}{\partial W} & \frac{\partial\dot{Q}}{\partial Q} & \frac{\partial\dot{Q}}{\partial\theta} \\ \frac{\partial\dot{\theta}}{\partial U} & V_T\frac{\partial\dot{\theta}}{\partial W} & \frac{\partial\dot{\theta}}{\partial Q} & \frac{\partial\dot{\theta}}{\partial\theta} \end{bmatrix} \begin{bmatrix} \overline{v} \\ \alpha \\ q \\ \theta \end{bmatrix} \\ &+ \begin{bmatrix} \frac{\partial\dot{U}}{\partial\delta_{Er}} & \frac{\partial\dot{U}}{\partial\delta_{El}} & \frac{\partial\dot{U}}{\partial\Delta Tr} & \frac{\partial\dot{U}}{\partial\Delta_{Tl}} \\ \frac{1}{V_T}\frac{\partial\dot{W}}{\partial\delta_{Er}} & \frac{1}{V_T}\frac{\partial\dot{W}}{\partial\delta_{El}} & \frac{1}{V_T}\frac{\partial\dot{W}}{\partial\Delta Tr} & \frac{1}{V_T}\frac{\partial\dot{W}}{\partial\Delta_{Tl}} \\ \frac{\partial\dot{Q}}{\partial\delta_{Er}} & \frac{\partial\dot{Q}}{\partial\delta_{El}} & \frac{\partial\dot{Q}}{\partial\Delta Tr} & \frac{\partial\dot{Q}}{\partial\Delta_{Tl}} \\ \frac{\partial\dot{\theta}}{\partial\delta_{Er}} & \frac{\partial\dot{\theta}}{\partial\delta_{El}} & \frac{\partial\dot{\theta}}{\partial\Delta Tr} & \frac{\partial\dot{\theta}}{\partial\Delta_{Tl}} \end{bmatrix} \begin{bmatrix} \delta_{Er} \\ \delta_{El} \\ \Delta_{Tr} \\ \Delta_{Tl} \end{bmatrix} \end{aligned} \tag{C.3}$$

$$\boldsymbol{\Delta\dot{x}}_{lat} = \begin{bmatrix} \frac{\partial\dot{V}}{\partial V} & \frac{1}{V_T}\frac{\partial\dot{V}}{\partial P} & \frac{1}{V_T}\frac{\partial\dot{V}}{\partial R} & \frac{1}{V_T}\frac{\partial\dot{V}}{\partial \phi} \\ \frac{1}{V_T}\frac{\partial\dot{P}}{\partial V} & \frac{\partial\dot{P}}{\partial P} & \frac{\partial\dot{P}}{\partial R} & \frac{\partial\dot{P}}{\partial \phi} \\ \frac{1}{V_T}\frac{\partial\dot{R}}{\partial V} & \frac{\partial\dot{R}}{\partial P} & \frac{\partial\dot{R}}{\partial R} & \frac{\partial\dot{R}}{\partial \phi} \\ \frac{1}{V_T}\frac{\partial\dot{\phi}}{\partial V} & \frac{\partial\dot{\phi}}{\partial P} & \frac{\partial\dot{\phi}}{\partial R} & \frac{\partial\dot{\phi}}{\partial \phi} \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix}$$
$$+ \begin{bmatrix} \frac{1}{V_T}\frac{\partial\dot{V}}{\partial \delta_{Ar}} & \frac{1}{V_T}\frac{\partial\dot{V}}{\partial \delta_{Al}} & \frac{1}{V_T}\frac{\partial\dot{V}}{\partial \delta_{Fr}} & \frac{1}{V_T}\frac{\partial\dot{V}}{\partial \delta_{Fl}} & \frac{1}{V_T}\frac{\partial\dot{V}}{\partial \delta_{Rr}} & \frac{1}{V_T}\frac{\partial\dot{V}}{\partial \delta_{Rl}} \\ \frac{\partial\dot{P}}{\partial \delta_{Ar}} & \frac{\partial\dot{P}}{\partial \delta_{Al}} & \frac{\partial\dot{P}}{\partial \delta_{Fr}} & \frac{\partial\dot{P}}{\partial \delta_{Fl}} & \frac{\partial\dot{P}}{\partial \delta_{Rr}} & \frac{\partial\dot{P}}{\partial \delta_{Rl}} \\ \frac{\partial\dot{R}}{\partial \delta_{Ar}} & \frac{\partial\dot{R}}{\partial \delta_{Al}} & \frac{\partial\dot{R}}{\partial \delta_{Fr}} & \frac{\partial\dot{R}}{\partial \delta_{Fl}} & \frac{\partial\dot{R}}{\partial \delta_{Rr}} & \frac{\partial\dot{R}}{\partial \delta_{Rl}} \\ \frac{\partial\dot{\phi}}{\partial \delta_{Ar}} & \frac{\partial\dot{\phi}}{\partial \delta_{Al}} & \frac{\partial\dot{\phi}}{\partial \delta_{Fr}} & \frac{\partial\dot{\phi}}{\partial \delta_{Fl}} & \frac{\partial\dot{\phi}}{\partial \delta_{Rr}} & \frac{\partial\dot{\phi}}{\partial \delta_{Rl}} \end{bmatrix} \begin{bmatrix} \delta_{Ar} \\ \delta_{Al} \\ \delta_{Fr} \\ \delta_{Fl} \\ \delta_{Rr} \\ \delta_{Rl} \end{bmatrix} \tag{C.4}$$

C.2 Partial Derivatives

The partial derivatives for each of the nonzero state space matrices $\boldsymbol{A}_{long}$, $\boldsymbol{A}_{lat}$, $\boldsymbol{B}_{long}$, $\boldsymbol{B}_{lat}$ are evaluated at trim.

$$\left.\frac{\dot{U}}{U}\right|_T = \frac{\partial}{\partial U}\left(\frac{X}{m} + VR - WQ\right) \tag{C.5}$$

$$= \frac{\rho V_T S}{m} C_{X_T} \tag{C.6}$$

$$V_T \left.\frac{\partial \dot{U}}{\partial W}\right|_T = V_T \frac{\partial}{\partial \alpha}\left(\frac{X}{m} + VR - WQ\right) \tag{C.7}$$

$$= \frac{V_T}{m}\frac{\partial}{\partial \alpha}\left[C_{X_T}\left(\frac{1}{2}P|V|^2 S\right)\right] \tag{C.8}$$

$$= \frac{V_T}{m}\frac{\partial}{\partial \alpha}\left\{(-C_d \cos\alpha + C_l \sin\alpha)\left(\frac{1}{2}P|V|^2 S\right)\right\} \tag{C.9}$$

$$= \frac{q_T S}{m}\left(C_{L_\alpha}\alpha_T + C_{L_T} - \frac{2C_{L_T}C_{L_\alpha}}{\pi A e}\right) \tag{C.10}$$

$$\left.\frac{\partial \dot{U}}{\partial Q}\right|_T = \frac{\partial}{\partial Q}\left(\frac{X}{m} + VR - WQ\right) \tag{C.11}$$

$$= 0 \tag{C.12}$$

$$V_T \left.\frac{\partial \dot{U}}{\partial \theta}\right|_T = \frac{\partial}{\partial \theta}\left(\frac{X}{m} + VR - WQ\right) \tag{C.13}$$

$$= \frac{\partial}{\partial \theta}\left(\frac{-mgsin\theta}{m}\right) \tag{C.14}$$

$$= -gcos\theta_T \tag{C.15}$$

$$\frac{1}{V_T}\left.\frac{\partial \dot{W}}{\partial U}\right|_T = \frac{1}{V_T}\frac{\partial}{\partial U}\left(\frac{Z}{m} + UQ - VP\right) \tag{C.16}$$

$$= \frac{-pS}{m} C_{L_T} \tag{C.17}$$

$$\left.\frac{\partial \dot{W}}{\partial W}\right|_T = \frac{\partial}{\partial W}\left(\frac{Z}{m} + UQ - VP\right) \tag{C.18}$$

$$= \frac{1}{m}\frac{\partial}{\partial \alpha}\left[C_Z\left(\frac{1}{2}\rho|V|^2 S\right) + mg\cos\theta\cos\phi\right] \tag{C.19}$$

$$= \frac{-q_T S}{m V_T} C_{L_\alpha} \tag{C.20}$$

$$\frac{1}{V_T}\frac{\partial \dot{W}}{\partial Q}\bigg|_T = \frac{1}{V_T}\frac{\partial}{\partial Q}\left(\frac{Z}{m} + UQ - VP\right) \tag{C.21}$$

$$= 1 - \frac{q_T S\overline{c}}{mV_T 2V_T} C_{L_Q} \tag{C.22}$$

$$\frac{1}{V_T}\frac{\partial \dot{W}}{\partial \theta}\bigg|_T = \frac{1}{V_T}\frac{\partial}{\partial \theta}\left(\frac{Z}{m} + UQ - VP\right) \tag{C.23}$$

$$= \frac{1}{V_T m}\frac{\partial}{\partial \theta}\left[C_Z\left(\frac{1}{2}\rho|V|^2 S\right) + mg\cos\theta\cos\phi\right] \tag{C.24}$$

$$= -\frac{1}{V_T} g\sin\theta_T \tag{C.25}$$

$$\frac{\partial \dot{Q}}{\partial U}\bigg|_T = \frac{\partial}{\partial \overline{V}}\left[\frac{M}{I_{yy}} - \frac{PR}{I_{yy}}\left(I_{xx} - I_{zz}\right)\right] \tag{C.26}$$

$$= 0 \tag{C.27}$$

$$V_T\frac{\partial \dot{Q}}{\partial W}\bigg|_T = V_T\frac{\partial}{\partial \alpha}\left[\frac{M}{I_{yy}} - \frac{PR}{I_{yy}}\left(I_{xx} - I_{zz}\right)\right] \tag{C.28}$$

$$= \frac{q_T S\overline{c}}{I_{yy}} C_{m_\alpha} \tag{C.29}$$

$$\frac{\partial \dot{Q}}{\partial Q}\bigg|_T = \frac{\partial}{\partial Q}\left[\frac{M}{I_{yy}} - \frac{PR}{I_{yy}}\left(I_{xx} - I_{zz}\right)\right] \tag{C.30}$$

$$= \frac{q_T S\overline{c}}{I_{yy}2V_T} C_{m_Q} \tag{C.31}$$

$$\frac{\partial \dot{Q}}{\partial \theta}\bigg|_T = \frac{\partial}{\partial \theta}\left[\frac{M}{I_{yy}} - \frac{PR}{I_{yy}}\left(I_{xx} - I_{zz}\right)\right] \tag{C.32}$$

$$= 0 \tag{C.33}$$

$$\frac{\partial \dot{\theta}}{\partial U}\bigg|_T = \frac{\partial}{\partial \overline{V}}\left(Q\cos\theta - R\sin\phi\right) \tag{C.34}$$

$$= 0 \tag{C.35}$$

$$\frac{\partial \dot{\theta}}{\partial W}\bigg|_T = \frac{\partial}{\partial \alpha}\left(Q\cos\theta - R\sin\phi\right) \tag{C.36}$$

$$= 0 \tag{C.37}$$

$$\left.\frac{\partial\dot{\theta}}{\partial Q}\right|_T = \frac{\partial}{\partial Q}\left(Q\cos\theta - R\sin\phi\right) \tag{C.38}$$

$$= 1 \tag{C.39}$$

$$\left.\frac{\partial\dot{\theta}}{\partial\theta}\right|_T = \frac{\partial}{\partial\theta}\left(Q\cos\theta - R\sin\phi\right) \tag{C.40}$$

$$= 0 \tag{C.41}$$

$$\left.\frac{\partial\dot{V}}{\partial V}\right|_T = \frac{\partial}{\partial\beta}\left(\frac{Y}{m} - UR + WP\right) \tag{C.42}$$

$$= \frac{\partial}{\partial\beta}\left[\frac{1}{m}\left\{C_Y\left(\frac{1}{2}\rho|V|^2S\right)\right\}\right] \tag{C.43}$$

$$= \frac{q_T S}{mV_T}C_{y_\beta} \tag{C.44}$$

$$\left.\frac{1}{V_T}\frac{\partial\dot{V}}{\partial P}\right|_T = \frac{1}{V_T}\frac{\partial}{\partial P}\left(\frac{Y}{m} - UR + WP\right) \tag{C.45}$$

$$= \frac{q_T Sb}{mV_T 2V_T}C_{y_p} \tag{C.46}$$

$$\left.\frac{1}{V_T}\frac{\partial\dot{V}}{\partial P}\right|_T = \frac{1}{V_T}\frac{\partial}{\partial P}\left(\frac{Y}{m} - UR + WP\right) \tag{C.47}$$

$$= \frac{q_T S}{mV_T}C_{y_r} - 1 \tag{C.48}$$

$$\left.\frac{1}{V_T}\frac{\partial\dot{V}}{\partial\phi}\right|_T = \frac{\partial}{\partial\phi}\left(\frac{Y}{m} - UR + WP\right) \tag{C.49}$$

$$= \frac{g}{V_T}\cos\theta_T \tag{C.50}$$

$$\left.V_T\frac{\partial\dot{P}}{\partial V}\right|_T = V_T\frac{\partial}{\partial\beta}\left[\frac{L}{I_{xx}} - \frac{QR}{I_{xx}}\left(I_{zz} - I_{yy}\right)\right] \tag{C.51}$$

$$= \frac{q_T Sb}{I_{xx}}C_{l_\beta} \tag{C.52}$$

$$\left.\frac{\partial\dot{P}}{\partial P}\right|_T = \frac{\partial}{\partial P}\left[\frac{L}{I_{xx}} - \frac{QR}{I_{xx}}\left(I_{zz} - I_{yy}\right)\right] \tag{C.53}$$

$$= \frac{q_T Sb^2}{I_{xx}2V_T}C_{l_p} \tag{C.54}$$

$$\left.\frac{\partial \dot{P}}{\partial R}\right|_T = \frac{\partial}{\partial R}\left[\frac{L}{I_{xx}} - \frac{QR}{I_{xx}}\left(I_{zz} - I_{yy}\right)\right] \tag{C.55}$$

$$= \frac{q_T S b^2}{2 I_{xx} V_T} C_{l_r} \tag{C.56}$$

$$\left.\frac{\partial \dot{P}}{\partial \phi}\right|_T = \frac{\partial}{\partial \phi}\left[\frac{L}{I_{xx}} - \frac{QR}{I_{xx}}\left(I_{zz} - I_{yy}\right)\right] \tag{C.57}$$

$$= 0 \tag{C.58}$$

$$V_T \left.\frac{\partial \dot{R}}{\partial V}\right|_T = V_T \frac{\partial}{\partial \beta}\left[\frac{N}{I_{zz}} - \frac{PQ}{I_{zz}}\left(I_{yy} - I_{xx}\right)\right] \tag{C.59}$$

$$= \frac{q_T S b}{I_{zz}} C_{n_\beta} \tag{C.60}$$

$$\left.\frac{\partial \dot{R}}{\partial P}\right|_T = \frac{\partial}{\partial P}\left[\frac{N}{I_{zz}} - \frac{PQ}{I_{zz}}\left(I_{yy} - I_{xx}\right)\right] \tag{C.61}$$

$$= \frac{q_T S b^2}{I_{zz} 2 V_T} C_{n_p} \tag{C.62}$$

$$\left.\frac{\partial \dot{R}}{\partial P}\right|_T = \frac{\partial}{\partial P}\left[\frac{N}{I_{zz}} - \frac{PQ}{I_{zz}}\left(I_{yy} - I_{xx}\right)\right] \tag{C.63}$$

$$= \frac{q_T S b^2}{I_{zz} 2 V_T} C_{n_r} \tag{C.64}$$

$$\left.\frac{\partial \dot{R}}{\partial \phi}\right|_T = \frac{\partial}{\partial \phi}\left[\frac{N}{I_{zz}} - \frac{PQ}{I_{zz}}\left(I_{yy} - I_{xx}\right)\right] \tag{C.65}$$

$$= 0 \tag{C.66}$$

$$V_T \left.\frac{\partial \dot{\phi}}{\partial V}\right|_T = V_T \frac{\partial}{\partial \beta}\left[P + Q\sin\phi\tan\theta + R\sin\phi\tan\theta\right] \tag{C.67}$$

$$= 0 \tag{C.68}$$

$$\left.\frac{\partial \dot{\phi}}{\partial P}\right|_T = \frac{\partial}{\partial P}\left[P + Q\sin\phi\tan\theta + R\sin\phi\tan\theta\right] \tag{C.69}$$

$$= 1 \tag{C.70}$$

$$\left.\frac{\partial \dot{\phi}}{\partial R}\right|_T = \frac{\partial}{\partial R}\left[P + Q\sin\phi\tan\theta + R\sin\phi\tan\theta\right] \tag{C.71}$$

$$= tan\theta_T \tag{C.72}$$

$$\left.\frac{\partial\dot{\phi}}{\partial\phi}\right|_T = \frac{\partial}{\partial\phi}\left[P + Q\sin\phi\tan\theta + R\sin\phi\tan\theta\right] \tag{C.73}$$

$$= 0 \tag{C.74}$$

$$\left.\frac{\partial\dot{V}}{\partial\delta_-}\right|_T = \frac{\partial}{\partial\delta_-}\left[\frac{X}{m} + VR - WQ\right] \tag{C.75}$$

$$= 0 \tag{C.76}$$

$$\left.\frac{\partial\dot{V}}{\partial\Delta T_-}\right|_T = \frac{\partial}{\partial\Delta T_-}\left[\frac{X}{m} + VR - WQ\right] \tag{C.77}$$

$$= \frac{1}{m} \tag{C.78}$$

$$\left.\frac{1}{V_T}\frac{\partial\dot{W}}{\partial\delta_-}\right|_T = \frac{1}{V_T}\frac{\partial}{\partial\delta_-}\left[\frac{Z}{m} + VQ - VP\right] \tag{C.79}$$

$$= -\frac{q_T S}{mV_T}C_{L_{\delta_-}} \tag{C.80}$$

$$\left.\frac{1}{V_T}\frac{\partial\dot{W}}{\partial\Delta T_-}\right|_T = \frac{1}{V_T}\frac{\partial}{\partial\Delta T_-}\left[\frac{Z}{m} + VQ - UP\right] \tag{C.81}$$

$$= 0 \tag{C.82}$$

$$\left.\frac{\partial\dot{Q}}{\partial\delta_-}\right|_T = \frac{\partial}{\partial\delta_-}\left[\frac{M}{I_{yy}} - \frac{PR}{I_{yy}}\left(I_{xx} - I_{zz}\right)\right] \tag{C.83}$$

$$= -\frac{q_T S\bar{c}}{I_{yy}}C_{m_{\delta_-}} \tag{C.84}$$

$$\left.\frac{\partial\dot{Q}}{\partial\Delta T_-}\right|_T = \frac{\partial}{\partial\Delta T_-}\left[\frac{M}{I_{yy}} - \frac{PR}{I_{yy}}\left(I_{xx} - I_{zz}\right)\right] \tag{C.85}$$

$$= 0 \tag{C.86}$$

$$\left.\frac{\partial\dot{\theta}}{\partial\delta_-}\right|_T = \frac{\partial}{\partial\delta_-}\left[\frac{M}{I_{yy}} - \frac{PR}{I_{yy}}\left(I_{xx} - I_{zz}\right)\right] \tag{C.87}$$

$$= 0 \tag{C.88}$$

$$\left.\frac{\partial\dot{\theta}}{\partial\Delta T_-}\right|_T = \frac{\partial}{\partial\Delta T_-}\left[Q\cos\theta - R\sin\phi\right] \tag{C.89}$$

$$= 0 \tag{C.90}$$

$$\frac{1}{V_T}\frac{\partial \dot{V}}{\partial \delta_-}\bigg|_T = \frac{1}{V_T}\frac{\partial}{\partial \delta_-}\left[\frac{Y}{m} - UR + WP\right] \tag{C.91}$$

$$= \frac{1}{V_T}\frac{q_T S}{mV_T} C_{Y_{\delta_-}} \tag{C.92}$$

$$\frac{\partial \dot{P}}{\partial \delta_-}\bigg|_T = \frac{\partial}{\partial \delta_-}\left[\frac{L}{I_{xx}} - \frac{QR}{I_{xx}}\left(I_{zz} - I_{yy}\right)\right] \tag{C.93}$$

$$= \frac{q_T Sb}{I_{xx}} C_{L_{\delta_-}} \tag{C.94}$$

$$\frac{\partial \dot{R}}{\partial \delta_-}\bigg|_T = \frac{\partial}{\partial \delta_-}\left[\frac{N}{I_{zz}} - PQ\left(I_{yy} - I_{xx}\right)\right] \tag{C.95}$$

$$= \frac{q_T Sb}{I_{zz}} C_{n_{\delta_-}} \tag{C.96}$$

$$\frac{\partial \dot{\phi}}{\partial \delta_-}\bigg|_T = \frac{\partial}{\partial \delta_-}\left[P + Q\sin\phi\tan\theta + R\sin\phi\tan\theta\right] \tag{C.97}$$

$$= 0 \tag{C.98}$$

C.3 Decoupled Aircraft Dynamics

The values for these state space matrices can be calculated at trim in order to simulate the flight of the UAV.

$$\boldsymbol{\Delta\dot{x}}_{long} = \begin{bmatrix} \frac{\rho V_T S C_X}{m} & \frac{q_T S}{m}\left(C_{L_\alpha}\alpha + C_l - \frac{2C_l C_{L_\alpha}}{\pi A R e}\right) & 0 & -g\cos\theta_T \\ \frac{-\rho S C_L}{m} & \frac{-q_T S C_{L_\alpha}}{mV} & 1 - \frac{q_T S\bar{c} C_{L_q}}{mV_T 2V_T} & \frac{-g\sin\theta_T}{V_T} \\ 0 & \frac{q_T S\bar{c} C_{m_\alpha}}{I_{yy}} & \frac{q_T S\bar{c} C_{m_q}}{I_{yy}2V_T} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v \\ \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0 & 0 & \frac{1}{m} & \frac{1}{m} \\ \frac{-q_T S C_{L_{\delta_{Er}}}}{mV_T} & \frac{-q_T S C_{L_{\delta_{El}}}}{mV_T} & 0 & 0 \\ \frac{q_T S\bar{c} C_{m_{\delta_{Er}}}}{I_{yy}} & \frac{q_T S\bar{c} C_{m_{\delta_{El}}}}{I_{yy}} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_{Er} \\ \delta_{El} \\ \Delta_{Tr} \\ \Delta_{Tl} \end{bmatrix} \tag{C.99}$$

$$\boldsymbol{\Delta\dot{x}}_{lat} = \begin{bmatrix} \frac{q_T S C_{Y_\beta}}{mV_T} & \frac{q_T S b C_{Y_p}}{mV_T 2V_T} & \frac{q_T S C_{Y_r}}{mV_T} - 1 & \frac{g\cos\theta_T}{V_T} \\ \frac{q_T S b C_{l_\beta}}{I_{xx}} & \frac{q_T S b^2 C_{l_p}}{I_{xx}2V_T} & \frac{q_T S b^2 C_{l_r}}{2I_{xx}V_T} & 0 \\ \frac{q_T S b C_{n_\beta}}{I_{zz}} & \frac{q_T S b^2 C_{n_p}}{I_{zz}2V_T} & \frac{q_T S b^2 C_{n_r}}{I_{zz}2V_T} & 0 \\ 0 & 1 & \tan\theta_T & 0 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} + \begin{bmatrix} \frac{q_T S C_{Y_{\delta_{Ar}}}}{V_T mV_T} & \frac{q_T S C_{Y_{\delta_{Al}}}}{V_T mV_T} & \frac{q_T S C_{Y_{\delta_{Fr}}}}{V_T mV_T} & \frac{q_T S C_{Y_{\delta_{Fl}}}}{V_T mV_T} & \frac{q_T S C_{Y_{\delta_{Rr}}}}{V_T mV_T} & \frac{q_T S C_{Y_{\delta_{Rl}}}}{V_T mV_T} \\ \frac{q_T S b C_{l_{\delta_{Ar}}}}{I_{xx}} & \frac{q_T S b C_{l_{\delta_{Al}}}}{I_{xx}} & \frac{q_T S b C_{l_{\delta_{Fr}}}}{I_{xx}} & \frac{q_T S b C_{l_{\delta_{Fl}}}}{I_{xx}} & \frac{q_T S b C_{l_{\delta_{Rr}}}}{I_{xx}} & \frac{q_T S b C_{l_{\delta_{Rl}}}}{I_{xx}} \\ \frac{q_T S b C_{n_{\delta_{Ar}}}}{I_{zz}} & \frac{q_T S b C_{n_{\delta_{Al}}}}{I_{zz}} & \frac{q_T S b C_{n_{\delta_{Fr}}}}{I_{zz}} & \frac{q_T S b C_{n_{\delta_{Fl}}}}{I_{zz}} & \frac{q_T S b C_{n_{\delta_{Rr}}}}{I_{zz}} & \frac{q_T S b C_{n_{\delta_{Rl}}}}{I_{zz}} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_{Ar} \\ \delta_{Al} \\ \delta_{Fr} \\ \delta_{Fl} \\ \delta_{Rr} \\ \delta_{Rl} \end{bmatrix} \tag{C.100}$$

Appendix D

AIRA Simulation

The following MATLAB scripts were used during the investigation of the AIRA.

D.1 Minimum Phase Zeros

The following MATLAB script was used to investigate a minimum phase system.

```
% Initialise state space matrices
SYS.x0 = [2 0.1 -1];
SYS.F = [0.6 -0.22 0.096; 0.5 0 0; 0 0.125 0];
SYS.G = [4; 0; 0];
SYS.H = [0 0.5 -2];
SYS.J = 0;

% Simulate the discrete system
SYS.r = 33;
SYS.l = rank(SYS.F);
SYS.k = 0:1:((SYS.r+1)*size(SYS.H,1))-1;
SYS.u = sin(SYS.k/4)+1;
SYS.x = zeros(3,size(SYS.k,2));
SYS.x(:,1) = SYS.x0;
SYS.y = zeros(1,size(SYS.k,2));
for n = 1:1:(((SYS.r+1)*size(SYS.H,1))-1)
    SYS.x(:,n+1) = SYS.F*SYS.x(:,n) + SYS.G*SYS.u(n);
    SYS.y(n+1) = SYS.H*SYS.x(:,n+1) + SYS.J*SYS.u(n+1);
end

% Generate pole plots
subplot(1,2,1);
pzmap(d2c(ss(SYS.F,SYS.G,SYS.H,SYS.J,1)));
subplot(1,2,2);
pzmap(ss(SYS.F,SYS.G,SYS.H,SYS.J,1));
axis([-2 2 -2 2]);
```

D.2 Nonminimum Phase Zeros

The following MATLAB script was used to investigate a non minimum phase system.

```
% Initialise state space matrices
SYS.x0 = [2 0.1 -1];
SYS.F = [0.6 -0.22 0.096; 0.5 0 0; 0 0.125 0];
SYS.G = [4; 0; 0];
SYS.H = [0 0.5 -6];
SYS.J = 0;

% Simulate the discrete system
SYS.r = 33;
SYS.l = rank(SYS.F);
SYS.k = 0:1:((SYS.r+1)*size(SYS.H,1))-1;
SYS.u = sin(SYS.k/4)+1;
SYS.x = zeros(3,size(SYS.k,2));
SYS.x(:,1) = SYS.x0;
SYS.y = zeros(1,size(SYS.k,2));
for n = 1:1:(((SYS.r+1)*size(SYS.H,1))-1)
    SYS.x(:,n+1) = SYS.F*SYS.x(:,n) + SYS.G*SYS.u(n);
    SYS.y(n+1) = SYS.H*SYS.x(:,n+1) + SYS.J*SYS.u(n+1);
end

% Generate pole plots
subplot(1,2,1);
pzmap(d2c(ss(SYS.F,SYS.G,SYS.H,SYS.J,1)));
subplot(1,2,2);
pzmap(ss(SYS.F,SYS.G,SYS.H,SYS.J,1));
axis([-2 2 -2 2]);
```

D.3 Zeros on Unit Circle

The following MATLAB script was used to investigate a system with zeros on the unit circle.

```
% Initialise state space matrices
SYS.x0 = [2 0.1 -1];
SYS.F = [0.6 -0.22 0.096; 0.5 0 0; 0 0.125 0];
SYS.G = [4; 0; 0];
SYS.H = [0 0.5 -4];
SYS.J = 0;

% Simulate the discrete system
SYS.r = 33;
SYS.l = rank(SYS.F);
SYS.k = 0:1:((SYS.r+1)*size(SYS.H,1))-1;
SYS.u = sin(SYS.k/4)+1;
SYS.x = zeros(3,size(SYS.k,2));
SYS.x(:,1) = SYS.x0;
SYS.y = zeros(1,size(SYS.k,2));
for n = 1:1:(((SYS.r+1)*size(SYS.H,1))-1)
    SYS.x(:,n+1) = SYS.F*SYS.x(:,n) + SYS.G*SYS.u(n);
    SYS.y(n+1) = SYS.H*SYS.x(:,n+1) + SYS.J*SYS.u(n+1);
end

% Generate pole plots
subplot(1,2,1);
pzmap(d2c(ss(SYS.F,SYS.G,SYS.H,SYS.J,1)));
subplot(1,2,2);
pzmap(ss(SYS.F,SYS.G,SYS.H,SYS.J,1));
axis([-2 2 -2 2]);
```

Appendix E

Reliability Analysis

E.1 FMEA Documentation

E.1.1 Failure Modes

On the following page is a table containing the failure modes for the Meraka Modular UAV.

Table E.1: *Component Failure Modes*

Failure Mode Number	Failure Mode	Failure Cause
1.1.1	Degraded	Several recharge cycles and aging
1.1.2	Discharged	Battery needs charging or is damaged
1.1.3	Unknown	Battery behaves unexpectedly
1.2.1	Degraded	Several recharge cycles and aging
1.2.2	Discharged	Battery needs charging or is damaged
1.2.3	Unknown	Battery mode undetectable
1.3.1	Degraded	Several recharge cycles and aging
1.3.2	Discharged	Battery needs charging or is damaged
1.3.3	Unknown	Battery mode undetectable
2.1.1	Range error	RC Controller is out of range
2.1.2	System failure	RC receiver stops working
2.2.1	Miscalibration	Incorrect mixing matrix or gains
2.2.2	Unknown	Servo board behaves unexpectedly
2.3.1	Power failure	ESC looses power connection
2.3.2	Miscalibration	Incorrect PWM settings
2.4.1	Stuck failure	Servo halts or gets jammed
2.4.2	Hardover failure	Servo rotates in one direction
2.4.3	Floating failure	Loose mechanical connection
2.4.4	Loss of effectiveness failure	Slipping mechanical connection
2.4.5	Unknown	Unexpected actuator behaviour
2.5.1	Stuck failure	Servo halts or gets jammed
2.5.2	Hardover failure	Servo rotates in one direction
2.5.3	Floating failure	Loose mechanical connection
2.5.4	Loss of effectiveness failure	Slipping mechanical connection
2.5.5	Unknown	Unexpected actuator behaviour
2.6.1	Stuck failure	Servo halts or gets jammed
2.6.2	Hardover failure	Servo rotates in one direction
2.6.3	Floating failure	Loose mechanical connection
2.6.4	Loss of effectiveness failure	Slipping mechanical connection
2.6.5	Unknown	Unexpected actuator behaviour
2.7.1	Stuck failure	Servo halts or gets jammed
2.7.2	Hardover failure	Servo rotates in one direction
2.7.3	Floating failure	Loose mechanical connection
2.7.4	Loss of effectiveness failure	Slipping mechanical connection
2.7.5	Unknown	Unexpected actuator behaviour
2.8.1	Stuck failure	Servo halts or gets jammed
2.8.2	Hardover failure	Servo rotates in one direction
2.8.3	Floating failure	Loose mechanical connection
2.8.4	Loss of effectiveness failure	Slipping mechanical connection
2.8.5	Unknown	Unexpected actuator behaviour
2.9.1	Stuck failure	Servo halts or gets jammed
2.9.2	Hardover failure	Servo rotates in one direction
2.9.3	Floating failure	Loose mechanical connection
2.9.4	Loss of effectiveness failure	Slipping mechanical connection

2.9.5	Unknown	Unexpected actuator behaviour
2.10.1	Stuck failure	Servo halts or gets jammed
2.10.2	Hardover failure	Servo rotates in one direction
2.10.3	Floating failure	Loose mechanical connection
2.10.4	Loss of effectiveness failure	Slipping mechanical connection
2.10.5	Unknown	Unexpected actuator behaviour
2.11.1	Stuck failure	Servo halts or gets jammed
2.11.2	Hardover failure	Servo rotates in one direction
2.11.3	Floating failure	Loose mechanical connection
2.11.4	Loss of effectiveness failure	Slipping mechanical connection
2.11.5	Unknown	Unexpected actuator behaviour
2.12.1	Stuck failure	Motor rotation halts
2.12.2	Slipping failure	Loose mechanical connection
2.12.3	Unknown	Unexpected motor behaviour
2.13.1	Stuck failure	Motor rotation halts
2.13.2	Slipping failure	Loose mechanical connection
2.13.3	Unknown	Unexpected motor behaviour
3.1.1	Communication failure	Microcontroller doesn't acknowledge
3.1.2	System failure	Microcontroller halts
3.1.3	Unknown	Unexpected microcontroller behaviour
3.2.1	Communication failure	Microcontroller doesn't acknowledge
3.2.2	System failure	Microcontroller halts
3.2.3	Unknown	Unexpected microcontroller behaviour
3.3.1	Range error	RC Controller is out of range
3.3.2	System failure	RC receiver stops working
3.4.1	Node failure	Node fails to acknowledge
3.4.2	Communication failure	Multiple devices access bus
3.4.3	System failure	CAN bus malfunctions
4.1.1	Bias	Temperature or mechanical changes
4.1.2	Stuck	Mechanical error in the detection mechanism
4.1.3	Drift	High rates for physical quantities detected with significant noise
4.1.4	Scale error	Sensor detects high rates for physical quantities
4.1.5	Unknown	Sensor behaves unexpectedly
4.2.1	Bias	Temperature or mechanical changes
4.2.2	Stuck	Mechanical error in the detection mechanism
4.2.3	Drift	High rates for physical quantities detected with significant noise
4.2.4	Scale error	Sensor detects high rates for physical quantities
4.2.5	Unknown	Sensor behaves unexpectedly
4.3.1	Bias	Temperature or mechanical changes
4.3.2	Stuck	Mechanical error in the detection mechanism
4.3.3	Drift	High rates for physical quantities

		detected with significant noise
4.3.4	Scale error	Sensor detects high rates for physical quantities
4.3.5	Unknown	Sensor behaves unexpectedly
4.4.1	Bias	Temperature or mechanical changes
4.4.2	Stuck	Mechanical error in the detection mechanism
4.4.3	Drift	High rates for physical quantities detected with significant noise
4.4.4	Scale error	Sensor detects high rates for physical quantities
4.4.5	Unknown	Sensor behaves unexpectedly
4.5.1	Bias	Temperature or mechanical changes
4.5.2	Stuck	Mechanical error in the detection mechanism
4.5.3	Drift	High rates for physical quantities detected with significant noise
4.5.4	Scale error	Sensor detects high rates for physical quantities
4.5.5	Unknown	Sensor behaves unexpectedly

E.1.2 Failure Effects

On the following page is a table containing the failure effects analysis for the failure modes identified for the Meraka Modular UAV.

Table E.2: *Failure Effects Analysis*

Failure Mode Number	Failure Effects (Local)	Failure Effects (Next Level)	Failure Effects (End Effect)	Severity Category
1.1.1	Non ideal voltage supply	On board components behave erratically	Inaccurate flight data	C
1.1.2	Low voltage supply	OBC and Servo Board looses avionics battery power	Controlled emergency landing	B
1.1.3	Unknown battery fault	OBC and Servo Board malfunctions	Controlled emergency landing	B
1.2.1	Non ideal voltage supply	RC Receiver and Servo Board behave erratically	Uncontrolled emergency landing	A
1.2.2	Low voltage supply	RC Receiver and Servo Board looses avionics battery power	Uncontrolled emergency landing	A
1.2.3	Unknown battery fault	RC Receiver and Servo Board malfunctions	Uncontrolled emergency landing	A
1.3.1	Non ideal voltage supply	Speed controller behaves erratically	Controlled emergency landing	B
1.3.2	Low voltage supply	Speed controller and motors loose power	Controlled emergency landing	B
1.3.3	Unknown battery fault	Speed controller looses malfunctions	Controlled emergency landing	B
2.1.1	Control signals received erratically	Servo board receives commands erratically	Uncontrolled emergency landing	A
2.1.2	Control signals not received	Servo board stops receiving commands	Uncontrolled emergency landing	A
2.2.1	Servo board distorts commands	Faulty commands sent to actuators	Uncontrolled emergency landing	A
2.2.2	Servo board malfunctions	Faulty commands sent to actuators	Uncontrolled emergency landing	A
2.3.1	ESC stops working	No commands sent to motors	Controlled emergency landing	B
2.3.2	ESC distorts	Faulty commands	Controlled	B

	commands	sent to motors	emergency landing	
2.4.1	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.4.2	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.4.3	Control surface starts flapping	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.4.4	Control surface looses effectiveness	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.4.5	Control surface behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.5.1	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.5.2	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.5.3	Control surface starts flapping	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.5.4	Control surface looses effectiveness	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.5.5	Control surface behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.6.1	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.6.2	Control surface gets stuck	Aircraft possibly looses trim or does dangerous	Uncontrolled emergency landing	A

		manoeuvre		
2.6.3	Control surface starts flapping	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.6.4	Control surface looses effectiveness	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.6.5	Control surface behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.7.1	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.7.2	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.7.3	Control surface starts flapping	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.7.4	Control surface looses effectiveness	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.7.5	Control surface behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Uncontrolled emergency landing	A
2.8.1	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.8.2	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.8.3	Control surface starts flapping	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.8.4	Control surface looses effectiveness	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B

2.8.5	Control surface behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.9.1	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.9.2	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.9.3	Control surface starts flapping	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.9.4	Control surface looses effectiveness	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.9.5	Control surface behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.10.1	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.10.2	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.10.3	Control surface starts flapping	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.10.4	Control surface looses effectiveness	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.10.5	Control surface behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.11.1	Control surface gets stuck	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.11.2	Control surface	Aircraft possibly	Controlled	B

	gets stuck	looses trim or does dangerous manoeuvre	emergency landing	
2.11.3	Control surface starts flapping	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.11.4	Control surface looses effectiveness	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.11.5	Control surface behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.12.1	Motor stops rotating	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.12.2	Motor rotates with varied speed	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.12.3	Motor behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.13.1	Motor stops rotating	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.13.2	Motor rotates with varied speed	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
2.13.3	Motor behaves unexpectedly	Aircraft possibly looses trim or does dangerous manoeuvre	Controlled emergency landing	B
3.1.1	Microcontroller looses sensor data	Aircraft switches to manual control	Inaccurate flight data	C
3.1.2	Microcontroller halts execution programs	Aircraft switches to manual control	Inaccurate flight data	C
3.1.3	Microcontroller applies erroneous commands	Aircraft switches to manual control	Inaccurate flight data	C
3.2.1	Microcontroller looses sensor data	Communication with GPS and OBC	Inaccurate flight data	C

		is lost		
3.2.2	Microcontroller halts execution programs	GPS module behaves erratically	Inaccurate flight data	C
3.2.3	Microcontroller applies erroneous commands	GPS module malfunctions	Inaccurate flight data	C
3.3.1	Control signals received erratically	GS receives commands & sensor data erratically	Inaccurate flight data	C
3.3.2	Control signals not received	GS stops receiving commands & sensor data	Inaccurate flight data	C
3.4.1	Node doesn't receive message	Node doesn't receive command	Inaccurate flight data	C
3.4.2	All devices don't receive message	Data is corrupted	Inaccurate flight data	C
3.4.3	CAN bus halts	AP fails and data is corrupted	Inaccurate flight data	C
4.1.1	Sensor data gets biased	OBC receives distorted sensor data	Inaccurate flight data	C
4.1.2	Sensor data halts at value	OBC receives distorted sensor data	Inaccurate flight data	C
4.1.3	Sensor data distorted by drift	OBC receives distorted sensor data	Inaccurate flight data	C
4.1.4	Sensor data gets scaled	OBC receives distorted sensor data	Inaccurate flight data	C
4.1.5	Sensor behaves unexpectedly	OBC receives distorted sensor data	Inaccurate flight data	C
4.2.1	Sensor data gets biased	OBC receives distorted sensor data	Inaccurate flight data	C
4.2.2	Sensor data halts at value	OBC receives distorted sensor data	Inaccurate flight data	C
4.2.3	Sensor data distorted by drift	OBC receives distorted sensor data	Inaccurate flight data	C
4.2.4	Sensor data gets scaled	OBC receives distorted sensor data	Inaccurate flight data	C
4.2.5	Sensor behaves	OBC receives	Inaccurate flight	C

	unexpectedly	distorted sensor data	data	
4.3.1	Sensor data gets biased	OBC receives distorted sensor data	Inaccurate flight data	C
4.3.2	Sensor data halts at value	OBC receives distorted sensor data	Inaccurate flight data	C
4.3.3	Sensor data distorted by drift	OBC receives distorted sensor data	Inaccurate flight data	C
4.3.4	Sensor data gets scaled	OBC receives distorted sensor data	Inaccurate flight data	C
4.3.5	Sensor behaves unexpectedly	OBC receives distorted sensor data	Inaccurate flight data	C
4.4.1	Sensor data gets biased	OBC receives distorted sensor data	Inaccurate flight data	C
4.4.2	Sensor data halts at value	OBC receives distorted sensor data	Inaccurate flight data	C
4.4.3	Sensor data distorted by drift	OBC receives distorted sensor data	Inaccurate flight data	C
4.4.4	Sensor data gets scaled	OBC receives distorted sensor data	Inaccurate flight data	C
4.4.5	Sensor behaves unexpectedly	OBC receives distorted sensor data	Inaccurate flight data	C
4.5.1	Sensor data gets biased	OBC receives distorted sensor data	Inaccurate flight data	C
4.5.2	Sensor data halts at value	OBC receives distorted sensor data	Inaccurate flight data	C
4.5.3	Sensor data distorted by drift	OBC receives distorted sensor data	Inaccurate flight data	C
4.5.4	Sensor data gets scaled	OBC receives distorted sensor data	Inaccurate flight data	C
4.5.5	Sensor behaves unexpectedly	OBC receives distorted sensor data	Inaccurate flight data	C

E.1.3 Detection Methods

On the following page is a table containing the detection methods for the failure modes identified for the Meraka Modular UAV.

Table E.3: *Fault Detection Methods*

Failure Mode Number	Failure Detection Method	Compensation Action
1.1.1	Battery precision voltage sensor	Monitor battery level, notify ground station operator
1.1.2	Battery precision voltage sensor	Prevent start up of on board components then activate backup battery power for select components
1.1.3	Battery precision voltage sensor	Activate backup battery for select components
1.2.1	Battery precision voltage sensor	Monitor battery level, notify ground station operator
1.2.2	Battery precision voltage sensor	Prevent start up of on board components, flash indicator
1.2.3	Battery precision voltage sensor	Attempt to activate avionics battery
1.3.1	Battery precision voltage sensor	Monitor battery level, notify ground station operator
1.3.2	Battery precision voltage sensor	Prevent start up of ESD and flash indicator
1.3.3	Battery precision voltage sensor	Prevent start up of ESD and flash indicator
2.1.1	Check range level from RC receiver	Halt actuator commands until communication resumes
2.1.2	Add a watchdog timer to receive signals from RC receiver	Halt actuator commands and attempt to restart RC receiver
2.2.1	Compare transmitted and received commands	Notify ground station operator, try comparison again
2.2.2	Compare transmitted and received commands	Notify ground station operator, try comparison again
2.3.1	Measure battery voltage	Attempt to divert power, notify ground station operator
2.3.2	Monitor actual and desired thrust	Notify ground station operator
2.4.1	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.4.2	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.4.3	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.4.4	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.4.5	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC

2.5.1	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.5.2	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.5.3	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.5.4	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.5.5	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.6.1	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.6.2	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.6.3	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.6.4	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.6.5	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.7.1	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.7.2	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.7.3	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.7.4	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.7.5	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.8.1	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.8.2	Compare desired and actual	Notify ground station operator,

	actuator deflection	send reconfiguration commands to OBC
2.8.3	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.8.4	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.8.5	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.9.1	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.9.2	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.9.3	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.9.4	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.9.5	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.10.1	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.10.2	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.10.3	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.10.4	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.10.5	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.11.1	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.11.2	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.11.3	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands

		to OBC
2.11.4	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.11.5	Compare desired and actual actuator deflection	Notify ground station operator, send reconfiguration commands to OBC
2.12.1	Compare desired and actual rotation, measure motor voltage, measure motor current	Notify ground station operator, send reconfiguration commands to OBC
2.12.2	Compare desired and actual rotation, measure motor voltage, measure motor current	Notify ground station operator, send reconfiguration commands to OBC
2.12.3	Compare desired and actual rotation, measure motor voltage, measure motor current	Notify ground station operator, send reconfiguration commands to OBC
2.13.1	Compare desired and actual rotation, measure motor voltage, measure motor current	Notify ground station operator, send reconfiguration commands to OBC
2.13.2	Compare desired and actual rotation, measure motor voltage, measure motor current	Notify ground station operator, send reconfiguration commands to OBC
2.13.3	Compare desired and actual rotation, measure motor voltage, measure motor current	Notify ground station operator, send reconfiguration commands to OBC
3.1.1	Check message acknowledge	Retry communication, switch to manual control
3.1.2	Send message and check acknowledge	Have watchdog time reset OBC
3.1.3	Send message and check acknowledge	Have watchdog timer reset OBC
3.2.1	Check message acknowledge	Retry communication, switch to manual control
3.2.2	Send message and check acknowledge	Have watchdog time reset OBC
3.2.3	Send message and check acknowledge	Have watchdog timer reset OBC
3.3.1	Check range level from RC receiver	Halt actuator commands until communication resumes
3.3.2	Add a watchdog timer to receive signals from RC receiver	Halt actuator commands and attempt to restart RC receiver
3.4.1	Monitor message acknowledge	Attempt to reset CAN bus, notify ground station operator
3.4.2	Attempt to send message and monitor acknowledge	Attempt to reset CAN bus, notify ground station operator
3.4.3	Check bus voltages	Attempt to reset CAN bus, notify ground station operator
4.1.1	Compare expected and actual	Notify OBC and ground station

	sensor data, apply signal processing filters to identify fault	operator
4.1.2	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.1.3	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.1.4	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.1.5	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.2.1	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.2.2	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.2.3	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.2.4	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.2.5	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.3.1	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.3.2	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.3.3	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.3.4	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.3.5	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.4.1	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.4.2	Compare expected and actual sensor data, apply signal	Notify OBC and ground station operator

	processing filters to identify fault	
4.4.3	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.4.4	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.4.5	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.5.1	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.5.2	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.5.3	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.5.4	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator
4.5.5	Compare expected and actual sensor data, apply signal processing filters to identify fault	Notify OBC and ground station operator

E.2 FTA Documentation

E.2.1 Component Failure Rates

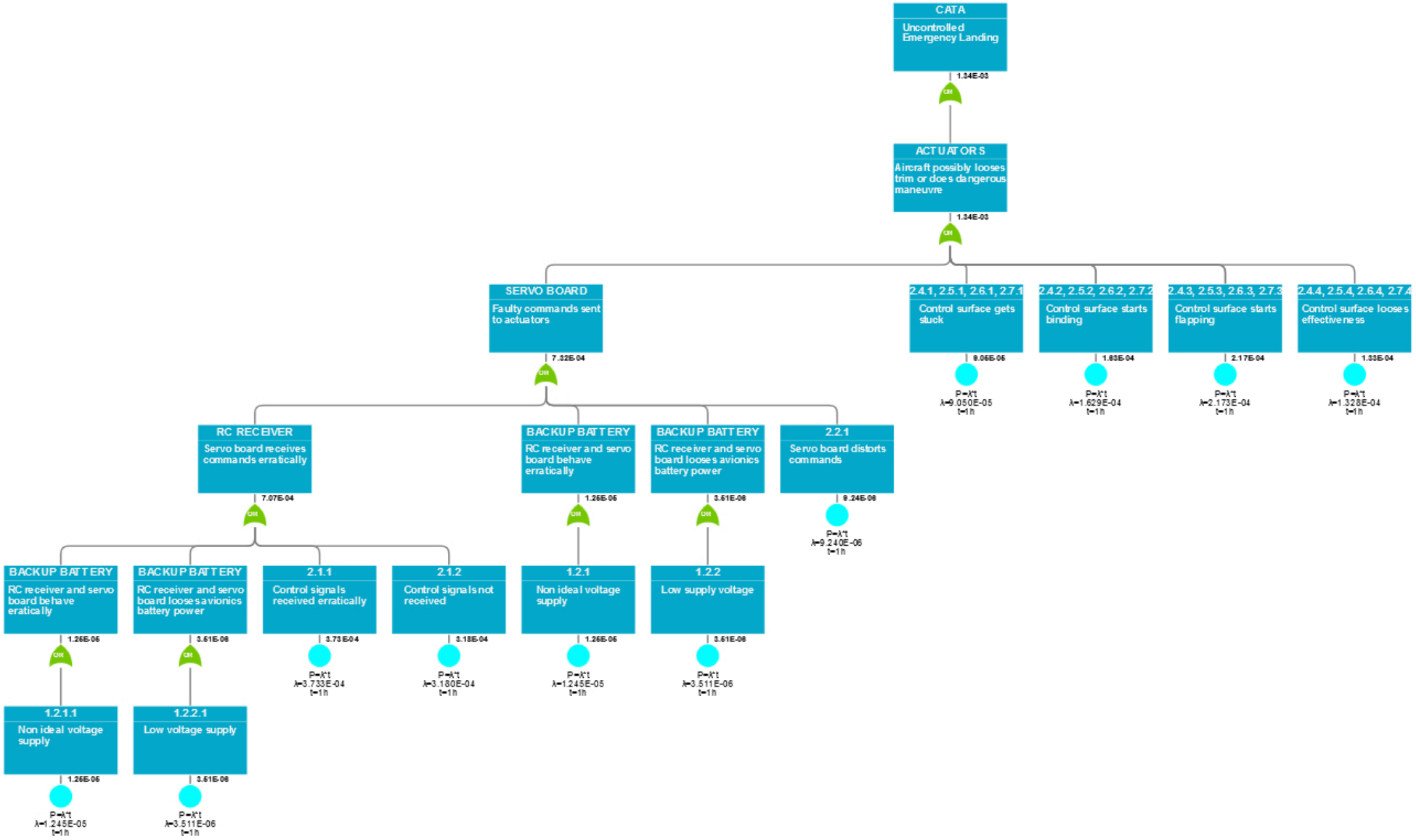
Category	Component	Failure Rate
Aircraft		
B	Avionics Battery	1.596e-5
A	Backup Battery	1.157e-5
B	Motor Batteries	1.596e-5
Actuation		
A	RC Receiver	69.133e-5
A	Servo Board	1.2e-5
B	Speed Controller	3.49e-6
A	Left Rudder	0.97e-4
A	Right Rudder	0.97e-4
A	Left Elevator	0.97e-4
A	Right Elevator	0.97e-4
B	Left Aileron	1.206e-4
B	Right Aileron	1.206e-4
B	Left Flap	1.206e-4
B	Right Flap	1.206e-4
B	Left Motor	5e-5
B	Right Motor	5e-5
Control		
C	PICA	1.6e-5
C	PICB	1.6e-5
C	RF Transmitter	69.133e-5
C	CAN Bus	1.0787e-5
Sensors		
C	GPS	2.5e-5
C	IMU	1.13e-4
C	Magnetometer	7.2e-5
C	Dynamic Pressure	3.3e-5
C	Static Pressure	3.55e-5

Table E.4: *Component Failure Rates*

E.2.2 Category A

On the following page is the Category A fault tree developed using [47].

Figure E.1: *Category A Fault Tree*



Bibliography

[1] A guide to fault detection and diagnosis. `http://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/faultdiagnosis.htm`, 19 April 2014 at 3:23 PM.

[2] *A generalized inverse for matrices*, volume 51. Proceedings of the Cambridge Philosophical Society, 1955.

[3] *On best approximate solution of linear matrix equations*, volume 52. Proceedings of the Cambridge Philosophical Society, 1956.

[4] Hybrid diagnostics for the falconsat-5 sciencecraft. Technical Report 29-31, St Louis, Missouri, March 2011.

[5] Djamel Benazzouz; Samir Benammar; Smail Adjerid. Fault detection and isolation based on neural networks case study: Steam turbine. *Energy and Power Engineering, Scientific Research*, 3(513-516), 2011.

[6] Gordon B. Aaseng; Honeywell Inc.; Glendale AZ. Blueprint for an integrated vehicle health management system. *IEEE*, 2001.

[7] Lionel Basson. Control allocation as part of a fault-tolerant control architecture for uavs. Master's thesis, University of Stellenbosch, November 2011.

[8] Willem Basson. Fault tolerant adaptive control of an unmanned aerial vehicle. Master's thesis, University Stellenbosch, December 2011.

[9] Haoyun Fu; Siddharth Kirtikar; Elena Zattoni; Harish Palanthandalam-Madapusi; Dennis S. Bernstein. Approximate input reconstruction for diagnosing aircraft control surfaces. *AIAA Guidance, Navigation, and Control Conference*, 2009.

[10] Mark Schwabacher; Jeff Samuels; Lee Brownston, editor. *The NASA Integrated Vehicle Health Management Technology Experiment for X-37*, MS-269-3, Mottfett Field, CA, 94035, 2002. NASA Ames Research Center.

[11] Sriram Narasimhan; Lee Brownston. Hyde – a general framework for stochastic and hybrid model-based diagnosis. Technical report, University of California, Santa Cruz; Perot Systems Government Services, Inc, M/S 269-3, NASA Ames Research Center, Moffett Field, CA-94035.

[12] Jeffrey Schein; Steven T. Bushby. A hierarchical rule-based fault detection and diagnostic method for hvac systems. *American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.*, 12(1), 2005.

[13] Reliability Analysis Center. *RAC Failure Mode / Mechanisms Distributions*. IIT Research Institute, 1991.

[14] Ming Yu; Danwei Wang; Ming Luo; Danhong Zhang; Qijun Chen. Fault detection, isolation and identification for hybrid systems with unknown mode changes and fault patterns. *Expert Systems with Applications*, 39(9955–9965), 2012.

[15] Alexander Feldman; Tolga Kurtoglu; Sriram Narasimhan; Scott Poll; David Garcia; Johan de Kleer; Lukas Kuhn; Arjan van Gemund. Empirical evaluation of diagnostic algorithm performance using a generic framework. *International Journal of Prognostics and Health Management*, 1:24, 2010.

[16] Dr. Mark Klein; Prof. Chrysanthos Dellarocas. A knowledge-based approach to handling exceptions in workflow systems. *Journal of Computer Supported Collaborative Work, Special Issue on Adaptive Workflow Systems*, January 2000.

[17] OSMA Dr. Michael Stamatelatos, NASA HQ. *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance, NASA Headquarters Washington, DC 20546, 2002.

[18] Lee S. Brownston Edward Balaban; Howard N. Cannon; Sriram Narasimhan. Model-based fault detection and diagnosis system for nasa mars subsurface drill prototype. *Aerospace Conference, IEEE*, 3-10 March 2007.

[19] Electrical Engineering Department, Louisiana State University. *Trending for Aircraft Fault Detection*, Balton Rouge, LA, USA, 2004.

[20] JJ Gentler. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker Inc, 1998.

[21] M Gopal. *Digital Control and State Variable Methods*. Mc Graw Hill, 2nd edition, 2004.

[22] Adi Ben-Israel; Thomas N.E. Greville. *Generalized Inverses, Theory and Applications*. Springer-Verlag New York, Inc, 2nd edition, 2003.

[23] Jim H An; Paul Wai Hing Chung; Joe McDonald; Madden. Automated cause and effect analysis for process plants. In *Conference Papers and Presentations (Computer Science)*, Rome, Italy, 10-13 May 2009. International Conference on Chemical and Process Engineering.

[24] Redouane Hallouzi. *Multiple-Model Based Diagnosis for Adaptive Fault-Tolerant Control*. PhD thesis, Delft University of Technology, 2008.

[25] Harish J; Palanthandalam-Madapusi; Dennis S. Bernstein Haoyun Fu; Jin Yan; Mario A. Santillo. Fault detection for aircraft control surfaces using approximate input reconstruction. In *American Control Conference*. Hyatt Regency Riverfront, St. Louis, MO, USA, 2009.

[26] Ing. Zizung Yoon Ing. Karsten Groÿekatthöfer. Quaternion-based adaptive attitude tracking controller without velocity measurements. *Journal of Guidance, Control, and Dynamics*, 24(6):1214–1222, 2001.

[27] Youmin Zhang; Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual reviews in control*, 32(2):229–52, 2008.

[28] Stephen B. Johnson. Introduction to system health engineering and management in aerospace. *Advanced Sensors and Health Management Systems Branch, EV23*.

[29] Brian C. Williams; Michael Hofbaur; Thomas Jones. Mode estimation of probabilitic hybrid systems. *International Journal of Control and Automation*, 2001.

[30] Anna Hagenblad; Fredrik Gustafsson; Inger Klein. A comparison of two methods for stochastic fault detection: the parity space approach and principal component analysis. 2004.

[31] Eugene L. Duke; Robert F. Antoniwicz; Keith D. Krambeer. *Derivation and Definition of a Linear Aircraft Model*. NASA, 1988.

[32] LoganFTA. Logan fault and event tree analysis. `http://loganfta.com/index.html`, November 2015.

[33] John G. Proakis; Dimitris G. Manolakis. *Digital Signal Processing, Principles, Algorithms, and Applications*. Pearson Education, Inc, 4th edition, 2007.

[34] Rasul Mohammadi. *Fault Diagnosis of Hybrid Systems with Applications to Gas Turbine Engines*. PhD thesis, Concordia University, Montreal Quebec, Canada, April 2009.

[35] Peter Bunus; Olle Isaksson; Beate Frey; Burkhard Münker. Rodon - a model-based diagnosis approach for the dx diagnostic competition. *Proc. DX'09*, pages 423–430, 2009.

[36] Absal Nabi. Application of least squares parameter estimation techniques in fault detection. *International Journal of Engineering Trends and Technology*, 2013.

[37] Kihoon Choi; Satnam Singh; Anuradha Kodali; Krishna R. Pattipati; John W. Sheppard; Setu Madhavi Namburu. Novel classifier fusion approaches for fault diagnosis in automotive systems. *AUTOTESTCON, Baltimore, MD, USA*, pages 260–269, September 2007.

[38] Hendrik Mostert Odendaal. An analysis and comparison of two methods for uav actuator fault detection and isolation. Master's thesis, University of Stellenbosch, October 2012.

[39] Office of the Secretary of Defense. Umanned aircraft systems roadmap. Technical report, Department of Defense, 2005.

[40] Stephen B. Johnson; Thomas J. Gormley; Seth S. Kessler; Charles D. Mott; An Patterson-Hine. *System Health Management with Aerospace Applications*. A John Wiley and Sons, Ltd., Publication, 2011.

[41] Charles Pecheur. The livingstone2 model-based diagnosis system. `http://ti.arc.nasa.gov/tech/rse/vandv/livingstone/`.

[42] Regardt Busch; Ian K Peddle. Active fault detection for open loop stable lti siso systems. *International Journal of Control, Automation and Systems*, 12(2):324–332, 2014.

[43] Nicholas Chung Propes. *Hybrid System Diagnosis and Control reconfiguration for Manufacturing Systems*. PhD thesis, Georgia Institute of Technology, April 2004.

[44] Eliahu Khlastchi; Meir Kalech; Lior Rokach. A hybrid approach for fault detection and diagnosis in autonomous systems.

[45] ALD Service. Ram commander. `http://www.aldservice.com`, 2015.

[46] Shawn Reimann; Jeremy Amos; Erik Bergquist; Jay Cole; Justin Phillips; Simon Shuster. Uav for reliability. 2013.

[47] Fault Tree Analysis Software. Online fta tool. `http://www.fault-tree-analysis-software.com`, November 2015.

[48] Rajiv Sreedhar; Timothy D. Hill; Gregory M. Stanley. Intelligent management for large networks.

[49] Imtiez Fliss; Moncef Tagina. Multiple faults diagnosis using causal graph. *Imtiez Fliss and Moncef Tagina UNIVERSITY OF MANOUBA, NATIONAL SCHOOL OF COMPUTER SCIENCES, SOIE LABORATORY*, 2010.

[50] Halim Alwi; Christopher Edwards; Chee Pin Tan. *Fault Detection and Fault-Tolerant Control Using Sliding Modes*. Springer, 2011.

[51] Alexander Feldman; Jurryt Pietersma; Arjan van Gemund. All roads lead to fault diagnosis: Model-based reasoning with lydia. In *BNAIC 2006: 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium, 5-6 October 2006*, 2006.

[52] Dario Vieira; Elisangela Rodrigues Vieira. *Software Engineering Research, Management and Applications 2009*, volume 253 of *Studies in Computational Intelligence*, pages 203–216. 2009.

[53] J Marzat; H Piet-Lahanier; F Damonget; E Walter. Model-based fault diagnosis for aerospace systems: a survey. In *Proceedings of the Institution of Mechanical Engineers*, volume Part G of *Journal of Aerospace Engineering*, page 1329, 2012.

[54] William Denson; Greg Chandler; William Crowell; Rick Wanner. *Nonelectronic Parts Reliability Data*. Reliability Analysis Center, Reliability Analysis Center PO BOX 4700, 1991.

[55] Ying Guo qing Wei Xue. Application of kalman filters for the fault diagnoses of aircraft engine. 2010.

[56] Jeffrey Humpherys; Preston Redd; Jeremy West. A fresh look at the kalman filter. *Society for Industrial and Applied Mathematics*, 54(4), 2012.