

A Fault-Tree Approach for Identifying Causes of Actuator Failure in Attitude Control Subsystem of Space Vehicles

Amitabh Barua

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science in
Electrical and Computer Engineering at
Concordia University
Montreal, Quebec, Canada

July 2004

© Amitabh Barua, 2004



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 0-612-94691-6
Our file *Notre référence*
ISBN: 0-612-94691-6

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

A Fault-Tree Approach for Identifying Causes of Actuator Failure in Attitude Control Subsystem of Space Vehicles

Amitabh Barua

Any space exploration program demands a guarantee ensuring smooth and reliable operations of space vehicles. Due to unforeseen circumstances and naturally occurring faults, it is inevitable to have an intelligent on-board fault detection, isolation and recovery (FDIR) scheme in unmanned space vehicles. For such spacecrafts, it is desired that an on-board fault-diagnosis system is capable of detecting, isolating, identifying or classifying faults in the system. Unfortunately, none of the existing fault-diagnosis methodologies alone can meet all the requirements of an ideal fault diagnosis system due to variety of fault types and their severity, and handling mechanisms. However, it is possible to overcome these shortcomings through the integration of different existing fault-diagnosis methodologies. Additionally, it is important to have abilities to correctly identify potential causes of system failure in such systems to initiate isolation and reconfiguration process effectively.

In this thesis, we have proposed a novel approach which strengthens existing efficient fault-detection mechanisms with an additional ability to classify different types of faults to effectively determine potential causes of failure in a subsystem. This extra capability

ensures a quick and efficient recovery/reconfiguration from disruptions. Our developed diagnosis/analysis procedure exploits a widely used qualitative technique called fault-tree analysis for failure analysis in the Attitude Control Subsystem (ACS) of a spacecraft. The proposed fault-tree synthesis algorithm utilizes machine-learning techniques to classify and rank primitive events in terms of their severity for a particular system failure. The effectiveness of the fault-tree synthesis algorithm presented in this thesis has been demonstrated under different simulated ACS failure scenarios. Constructed fault-trees have been able to represent combinations of events leading to different failures resulting due to artificially injected faults in a *Simulink* model of ACS. It is important to emphasize that proposed technique has potentials for being integrated in an on-board spacecraft health monitoring and diagnosis tool.

Acknowledgements

I would like to take this opportunity to thank my supervisors Dr. Purnendu Sinha and Dr. Kash Khorasani for their expert guidance in carrying out this research. While Dr. Sinha has helped me to develop new ideas by providing his suggestions and feedbacks on my work on a regular basis, Dr. Khorasani has challenged me with difficult questions at different stages of this work through which my ideas took more concrete shape. My sincere thanks also go to Mr. Siamak Tafazoli of the Space Technology sector of Canadian Space Agency who has provided us with valuable information at different stages of this thesis. I would like to express my appreciation to the examiners Dr. L. Rodrigues and Dr. Hashtrudi Zad for their valuable comments for improving the presentation of this thesis.

Enormous encouragement and support from my parents have always been the key factor behind every success in my life. It has just got repeated one more time for this work. I would like to express my gratitude to my wife for her constant inspiration, support and understanding while carrying out this research.

My friends in Montreal – especially, Ajit, Ayush, Debasish and Harun – have extended their helping hands towards me whenever I needed. They have also been the sources of lots of fun in my life. All these have been very important to keep a balance in my life and carry out my research smoothly. Finally, I would like to thank all the members of my research group. The discussions I have had with these people have been really rewarding for me – technically and otherwise too.

Table of Contents

List of Figures.....	ix
List of Tables	xi
1 Introduction	
1.1 Problem Domain	1
1.2 Motivation	3
1.3 Current State-of-the-art of Fault Diagnosis	4
1.4 FDIR Goals and Limitations of the Existing Works	6
1.5 Proposed Approach for Fault Diagnosis	7
1.6 Contribution of the Thesis	10
1.7 Organization of the Thesis	11
2 Modeling of Attitude Control Subsystem	
2.1 Introduction to Attitude Control Subsystem (ACS)	12
2.1.1 ACS Modes.....	16
2.1.2 ACS Performance Requirements	17
2.1.3 ACS Computations	18
2.1.4 Attitude Control Methods	18
2.1.5 Attitude Motions for Three-axis Stabilized Spacecraft	22
Utilizing Reaction Wheels and With No Momentum Bias	

2.2	ACS Modeling in MATLAB- <i>Simulink</i>	24
2.2.1	Satellite Body Dynamics	26
2.2.2	Reaction Wheel Dynamics	27
2.3	Summary	35
3	Fault-Tree Synthesis and Analysis: An Overview and Proposed Approach	
3.1	Introduction to Fault-Trees	36
3.1.1	Automated Fault-Tree Synthesis Methodologies	39
3.1.2	Fault-Tree Analysis	44
3.2	IFT Approach for Fault-Tree Synthesis	46
3.2.1	The ID3 Algorithm	47
3.2.2	Feature Extraction	49
3.2.3	Attribute Selection Sequence	50
3.3	Proposed Algorithm for Fault-Tree Synthesis	52
3.4	Summary	55
4	Fault-Tree Synthesis from Simulated ACS-model Data	
4.1	Introduction	56
4.2	ACS Failure Scenarios	58
4.3	Fault-Tree Synthesis under Different ACS Failure Scenarios	69
4.3.1	Identification of Different Ranges of Feature Values	70
4.3.2	Generation Input for Fault-Tree Synthesis	71
4.3.3	Determination of Attribute Selection Sequence	73
4.3.4	Fault-Tree Construction	76

4.4 Interpretation of the Resulting Fault-Trees 85

4.5 Summary 89

5 Conclusion 90

Bibliography 95

Appendix 99

List of Figures

1.1	Proposed Framework for ACS Fault Diagnosis	8
2.1	Axes and Angles Used for Specifying Spacecraft's Attitude	12
2.2	ACS Block Diagram	14
2.3	Typical ACS Components	15
2.4	Functional Diagram of <i>Simulink</i> Model of ACS	24
2.5	Satellite Body Dynamics... ..	26
2.6	<i>Ithaco Type-A</i> Reaction Wheel Model	28
2.7	Motor Torque Control	29
2.8	Speed Limiter	30
2.9	EMF Torque Limiting	31
2.10	Friction Model	32
3.1	Basic Fault-Tree Structure	37
3.2	A Binary Decision Diagram (BDD)	46
3.3	A Simple Decision Tree	49
4.1	Pitch Error Vs Time under Failure Scenario-1	59
4.2	Pitch Error Vs Time under Failure Scenario-2	61
4.3	Pitch Error Vs Time under Failure Scenario-3	64
4.4	Different ACS Parameters under Failure Scenario-3	66

4.5	Pitch Error Vs Time under Failure Scenario-4	67
4.6	Fault-Tree for Failure Scenario-1	77
4.7	Fault-Tree for Failure Scenario-2 (Non-overlapping ranges of feature values)	79
4.8	Fault-Tree for Failure Scenario-2 (Overlapping ranges of feature values)	80
4.9	Fault-Tree for Failure Scenario-3	82
4.10	Fault-Tree for Failure Scenario-4 (for the selection sequence $V_c \rightarrow I_m \rightarrow T_m \rightarrow \omega_w \rightarrow V_b$)	84
4.11	Fault-Tree for Failure Scenario-4 (for the selection sequence $V_c \rightarrow T_m \rightarrow I_m \rightarrow \omega_w \rightarrow V_b$)	85
4.12	Fault-Tree for Failure Scenario-2, Case-1 (when the attribute selection sequence is reversed)	87
4.13	Fault-Tree for Failure Scenario-2, Case-2 (when the attribute selection sequence is reversed)	87
4.14	Fault-Tree for Failure Scenario-3 (with a different attribute selection sequence)	89

List of Tables

2.1	Disturbance Torques in Space	13
2.2	ACS Performance Requirements	17
2.3	Attitude Control Methods and Their Capabilities	19
2.4	<i>Ithaco Type-A</i> Reaction Wheel Constants	29
3.1	Training Set for Induction Task	48
4.1	Selected Attributes for Fault-Tree Synthesis and Feature Extraction Functions	57
4.2	Example Vectors and Ranges for Feature Values under Failure Scenario-1	60
4.3	Example Vectors and Ranges for Feature Values under Failure Scenario-2	62
4.4	Example Vectors and Ranges for Feature Values under Failure Scenario-3	65
4.5	Example Vectors and Ranges for Feature Values under Failure Scenario-4	68
4.6	Summary of the Ranges of Numeric Feature Values	70
4.7	Different Ranges of Feature Values under different failure scenarios	71
4.8	Input ('Existing Example' Set) for the proposed FTS Algorithm	72
4.9	Ranking Assigned to the Attributes Based on % Change in Entropy	74

Chapter 1

Introduction

This chapter starts with describing the problem domain of this thesis, providing an overview of the fault detection, isolation and recovery (FDIR) scheme required for unmanned spacecrafts and discussing the motivations behind the research works carried out in this thesis. It also presents a short review on the existing fault diagnosis methodologies and identifies the limitations of the existing works. After that, the proposed approach for fault diagnosis in the attitude control subsystem (ACS) of a space vehicle is presented. Finally, specific contributions of this thesis are discussed and the organization of the thesis is presented.

1.1 Problem Domain

Often for unmanned space vehicles, continuous communication with ground station may not be possible even during fault-free conditions. Further, in unforeseen environments, ground control could be interrupted for a long time. Moreover, especially for deep-space missions, round-trip communications delay – which may be of several hours – between ground and the spacecraft makes the operator intervention in controlling the spacecraft to

adapt to the changes in the environment in real-time more difficult and sometimes impossible. There is also an increased need for cost minimization in ground support systems, especially in case of long-duration space missions. Because of all these reasons, an on-board fault detection, isolation and recovery (FDIR) scheme is necessary for unmanned space vehicles. The development of a technology that would allow unmanned space vehicles to detect, diagnose and fix anomalies on-board is indeed a challenging problem.

Traditionally, the ground system has been almost entirely responsible for spacecraft planning and scheduling as well as spacecraft Health and Safety verifications. In the case of on-board anomalies, the ground-based flight operations team (FOT) has been responsible for finding out the possible cause(s) of such anomalies and developing solution(s) to overcome the problems. For an autonomous unmanned spacecraft, all these tasks are required to be performed on-board without any need for human interaction. In order to achieve smooth autonomous operations, the spacecraft needs to be designed such a way that the critical functions are executed autonomously without severe performance degradations even in the presence of undesired events. Therefore, the craft has to be empowered with fault-tolerance capabilities. In addition, there has to be decisional autonomy so that high-level planning and scheduling can be performed on-board.

A system which is capable of detecting, isolating, identifying or classifying faults is called a *fault-diagnosis system*. Development of such system for autonomous operations of spacecrafts involves the utilization of methods presently available in different fields

including those in system identification, robust and adaptive control, system health-monitoring, system modeling and analysis to name a few. The goal of an on-board FDIR function is to detect fault(s) at early stages and to take appropriate recovery actions before the fault(s) causes a failure. A rigorous failure analysis procedure within the diagnostic system can be a very useful feature for identifying the source of malfunction in order to determine a quick and correct recovery plan. While such failure analysis may not be easily performed within many of the existing diagnostic systems alone due to their limited capability of performing fault detection and diagnosis together, it can be achieved by a complementary procedure that incorporates fault-tree analysis based techniques in on-board fault diagnosis and recovery system. In this thesis, a framework for spacecraft fault-diagnosis has been proposed.

1.2 Motivation

In unmanned space vehicles, such as satellites, disturbances and anomalies in the reaction wheels or momentum wheels, i.e., in actuator mechanisms are often the main reasons behind the failures in vehicle's Attitude Control Subsystem (ACS). We have been aware of a similar problem that has been encountered by the Canadian Space Agency (CSA), where the pitch momentum wheel of a satellite had failed. Given this background, we had planned to investigate the possibility of correlated faults in the ACS behind such failures.

In the above-mention CSA-satellite, some anomalies were detected during its on-orbit operations. Consequently, the system we were interested in was not accessible. Therefore, the problem in hand was essentially about diagnosing a system for which there

had been little experience from our side and at the same time access to the system would be limited. The only way to analyze such system was to study a model of the system rather than experimenting with the actual system. For this purpose, a MATLAB-*Simulink* model of a generic ACS of a satellite that maintains its required attitude using reaction wheels was developed. *Simulink* was chosen as the modeling tool because of its wide acceptance and growing demand in system modeling, analysis and design. A detailed description of the ACS modeling is presented in Chapter 2 of this thesis. Finally, in order to perform fault-diagnosis and failure analysis studies on the above-mentioned satellite subsystem, a fault-tree based analysis technique, which is discussed in Chapter 3 of this thesis, was selected.

1.3 Current State-of-the-art of Fault Diagnosis

In this section a short review on fault-diagnosis methodologies are presented. These methodologies are applicable, but not limited to spacecraft fault diagnosis. Existing fault-diagnosis methodologies may be broadly classified into two categories based on the form of knowledge they require: (1) Process model-based methods (2) Process history-based methods.

As the name suggests, in process model-based methods, the system is represented by a mathematical model of the same. These methods are powerful but far more expensive than a simple rule-based technique in terms of computing power. Process model-based methods can be further divided into two sub-categories: qualitative and quantitative.

Qualitative methods include *fault-trees* and *signed graphs* [1]. Fault-tree analysis is a widely used technique for finding the cause of a failure. It usually uses ‘top-down’ or ‘back-tracking’ approach until the possible root cause(s) behind the anomaly is found. Though the manual synthesis of fault-trees is not uncommon in practice, for complex systems, automatic generation of fault-trees is desired and feasible. The quantitative methods include *Residual-based* methods and *Assumption-based* methods. *Residual-based* method [2] works in two steps: *residual generation* to detect faults and *decision process* to identify the cause. The residual generation technique may be based on *hardware redundancy (voting schemes)*, *state estimation* or *parameter estimation* methods. *Decision process* involves decision functions that are calculated using the residuals and some decision logics. In an *assumption-based* method [3], certain assumptions on the normal behavior of the system are made and diagnosis is based on the violation of these assumptions.

On the other hand, process history-based methods require large amount of process data. As in the case of process model-based methods, process history-based methods can also be divided further into two sub-categories: qualitative and quantitative. Qualitative methods include *Rule-based* methods and *Qualitative Trend Analysis*. A *Rule-based* method [4] utilizes an explicit mapping of known symptoms to root cause. Diagnostic *Expert Systems* are based on this method. In *Qualitative Trend Analysis* [5, 6] sensor values are measured for identification of trends and the identified trends are interpreted in terms of fault scenarios. Quantitative methods include *Neural Networks* and *Statistical Techniques* such as Multivariable Statistical Process Control (MSPC) [7] that provides a

diagnostic tool for process monitoring and malfunctions. *Neural Networks*-based fault detection methods [8] have learning and interpolation capabilities to handle unseen situations. However, they may not be very suitable for fault case analysis because of their 'black box' property.

As mentioned earlier, a fault-diagnosis system has to be capable of detecting, isolating, identifying or classifying faults. It appears from the short review of the existing diagnosis method that a single method is inadequate to meet all requirements of an ideal diagnosis system.

1.4 FDIR Goals and Limitations of the Existing Works

Due to the issues and shortcomings of using the ground-based controllers for unmanned autonomous space vehicles mentioned in Section 1.1, it is necessary and desired that the ground-based autonomy be shifted to autonomy on-board a spacecraft. The FDIR goals and requirements for such autonomous spacecrafts are of significant technical challenges. The main objectives of the FDIR problem at present time are:

- To enhance the existing on-board fault detection and diagnosis capability by powerful methods.
- To improve the mission reliability and survival by recovering from the upsets gracefully.
- To simplify the implementation of diagnosis procedures by developing software components that can be re-used in some future applications.

It is clear from the both from the above-mentioned goals of the FDIR problem and from the discussion presented in Section 1.3 that the on-board FDIR system should not only be able to detect the anomalies present in the system but also be able to diagnose the system for identifying the source(s) of malfunction(s) in order to recover from the upset(s). However, such analysis may not be performed easily within many of the existing diagnostic systems alone due to their limited capability of performing fault detection and diagnosis together. Most of the existing approaches and techniques, according to the open literature, are focused towards the detection of anomalies in the system. But an effective recovery highly depends on the system's knowledge on the anomaly and the reason behind the same. The existing approaches also lack a modular method of integrating the various types of existing techniques for building a strong on-board FDIR scheme. Therefore, there is a need for integrating existing fault detection and diagnosis techniques in order to develop a powerful on-board FDIR system.

1.5 Proposed Approach for Fault Diagnosis

The main objective of the approach presented here is to empower existing fault-detection techniques with a capability for identifying the cause(s) behind the detected anomaly. Consequently, in the proposed approach the existence of an efficient fault detection mechanism is assumed. The fault detection mechanism may be a process model-based method or a process history-based method (as discussed in Section 1.3). However, the fault-diagnosis or failure analysis part of the proposed approach utilizes the process model-based diagnosis using a *qualitative* technique called fault-tree analysis. Fault-trees

are be constructed by the proposed fault-tree synthesis algorithm developed in this thesis which is presented in Chapter 3. A detailed description of the fault-tree construction is given in Chapter 3 and 4.

The proposed diagnosis procedure does not depend on the design phase of the system (explained in details in Section 3.3). Figure 1.1 illustrates the proposed framework for fault diagnosis in the attitude control subsystem (ACS) of a satellite. Upon detection of any fault or anomaly in the ACS, the diagnosis system starts monitoring the pre-defined attributes (signals) on time-frames of X seconds with Y seconds overlap. The values of X and Y can be selected or set depending on the diagnosis requirement.

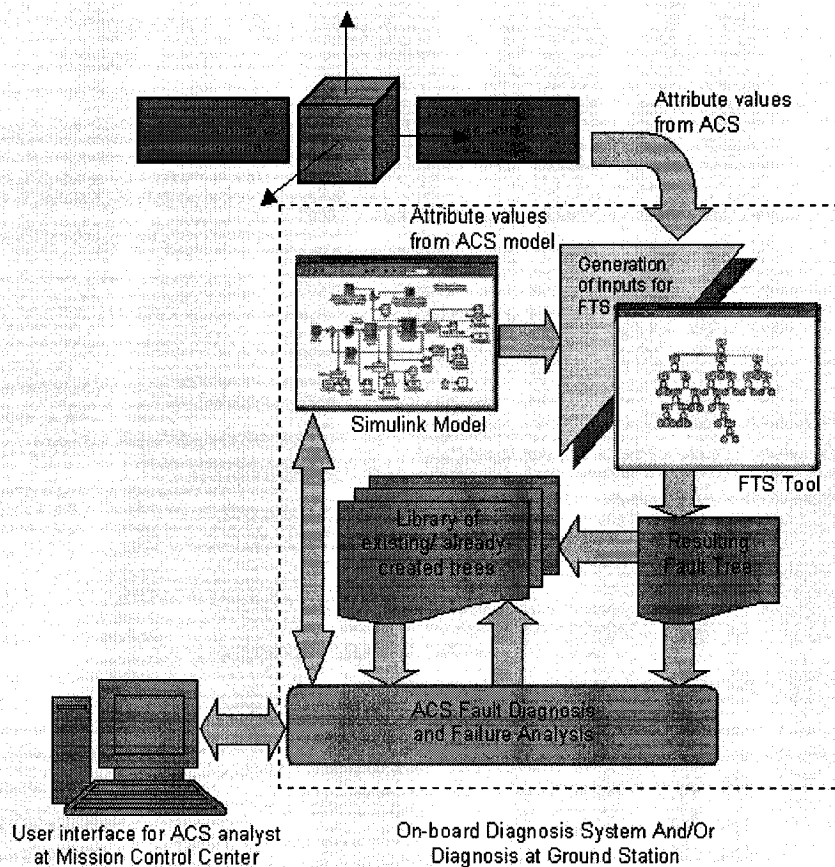


Figure 1.1: Proposed Framework for ACS Fault Diagnosis

After acquiring data for a particular attribute over the above-mentioned time-frame, the next step is to extract features from the data (will be discussed in Chapter 3 in detail). Subsequently, a vector of numeric feature-values for attributes is generated in which each feature value corresponds to an attribute over that time-frame. We call each value in this vector as 'current feature value' and the vector as 'current example'.

Once this 'current example' vector is created by feature extraction from the attributes, the next task is fault-tree synthesis. For this purpose, the current example is added to the 'existing' example set. The 'existing' example set consists of the vectors of numeric feature values for all attributes. This 'existing' example set is to be formed through the simulation of ACS (under fault-free condition as well as in presence of fault) and/or from the mission data after putting the spacecraft to the orbit in which it is intended to operate. The total example set (current plus previously existing) forms the input to the fault-tree synthesis tool, which constructs the tree using the proposed fault-tree synthesis algorithm. If the generated tree does not match with any of the existing trees in the database, this tree gets added to the existing library. After this, the constructed tree can be used for fault diagnosis.

Above computations can be performed in real time or near-real time on-board as well as at ground, if necessary. It is also possible identify and classify faults on-board by comparing the constructed trees with those existing in the library. Moreover, on-board generation of fault-trees provides better resolution on the required data compared to those available through telemetry points. This is due to the fact that telemetry data are usually sampled at lower frequencies and in most cases, sampling is done at every second, which

is not suitable if any frequency domain analysis is necessary for capturing the high frequency components of any signal. Finally, where fault-tree analysis is to be performed at ground, instead of downloading huge amount of data through telemetry, only relevant information such as constructed fault-trees can be transmitted to the ground.

1.6 Contribution of the Thesis

The objective of this work is to develop a procedure for fault diagnosis that empowers the existing fault detection techniques with the capability of identifying and classifying the source(s) of the anomalies in the system. The specific contributions of this thesis are as follows:

- (1) A framework for a fault diagnosis in the attitude control subsystem (ACS) has been developed.
- (2) In order to perform fault diagnosis and failure analysis related studies on unmanned spacecraft subsystems, model of a generic attitude control subsystem (ACS), with fault-injection capabilities, has been developed.
- (3) A new modified fault-tree synthesis algorithm for analyzing the cause(s) of failure(s) of the ACS has been developed from existing machine-learning based induction techniques for fault-tree synthesis. The proposed algorithm does not require detailed knowledge on design and construction of the system under consideration.
- (4) Fault-tree synthesis has been demonstrated using the proposed fault-tree synthesis algorithm under different ACS failure scenario which establishes that the

proposed framework has potential for automated spacecraft health monitoring and diagnosis.

1.7 Organization of the Thesis

In Section 1.5 of this chapter the proposed approach for fault-diagnosis has been presented. The organization of the remaining part of the thesis is as follows: modeling of generic attitude control subsystem (ACS) of a satellite is discussed in Chapter 2. This system model has been used to perform fault-diagnosis and failure analysis studies in this thesis. In Chapter 3, a review of existing fault-tree synthesis methodologies is provided and the algorithm developed in this thesis for fault-tree synthesis utilizing learning techniques is presented. In Chapter 4, different possible ACS failure scenarios are discussed and demonstration of fault-tree synthesis is provided under each failure scenario. Finally, the thesis is concluded in Chapter 5 with discussions and future research directions.

Chapter 2

Modeling of Attitude Control Subsystem

The first part (Section 2.1) of this chapter provides a review and background information on the Attitude Control Subsystem (ACS), which are necessary for better understanding of the MATLAB-*Simulink* model of the ACS presented in the second part (Section 2.2) of this chapter.

2.1 Introduction to Attitude Control Subsystem (ACS)

The main purpose of the Attitude Control Subsystem (ACS), which is commonly considered as momentum management system, is to orientate the main structure of the satellite at desired angle(s) within required accuracy. This ‘required accuracy’ is set by the payload, communication devices, etc. mounted on the main structure. Attitude of a spacecraft may be specified in a number of ways such as direction cosines, Euler’s angles etc. When Euler’s angles are used to specify the attitude, the information required for

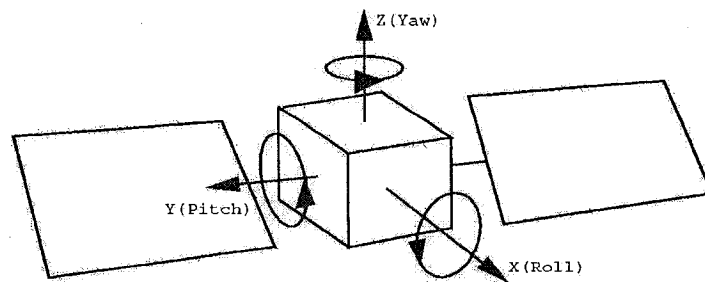


Figure 2.1: Axes and Angles Used for Specifying Spacecraft’s Attitude

specifying the attitude includes the three angles: ψ (roll), θ (pitch) and Φ (yaw), which are the measures of rotations about the x , y and z axes respectively.

An ACS is required because a body in space is subjected to small but persistent disturbance torques from a variety of sources as shown in Table 2.1 [9]. These torques would quickly re-orient the spacecraft unless restricted in some way. Therefore, it is required that the spacecraft determine its attitude using sensors and control the same using actuators. Disturbance torques may be external or internal to the spacecraft.

Table 2.1: Disturbance Torques in Space

External Torques	Internal Torques
Aerodynamic, Magnetic, Gravity Gradient, Solar Radiation, Thrust Misalignment.	Mechanisms, Fuel Movement, Astronaut Movement, Flexible Appendages, General Mass Movement

The major components of the ACS are [9]: the Attitude Control Processor (ACP) or on-board attitude controller, control torquers or actuators, (for example, reaction wheels (RW), momentum wheels (MW), magnetic torque bars (MTB), etc.), attitude sensors and the spacecraft body. The attitude sensors acquire spacecraft's attitude. The errors in angles are computed and based on these error signals the on-board ACP generates torque command voltages. Control actuators produce torques depending on the torque demand/command voltage inputs to them from the ACP. In this process, the required attitude is attained. A generic ACS block diagram with reaction wheel as the actuator for control along a single axis is shown in Figure 2.2:

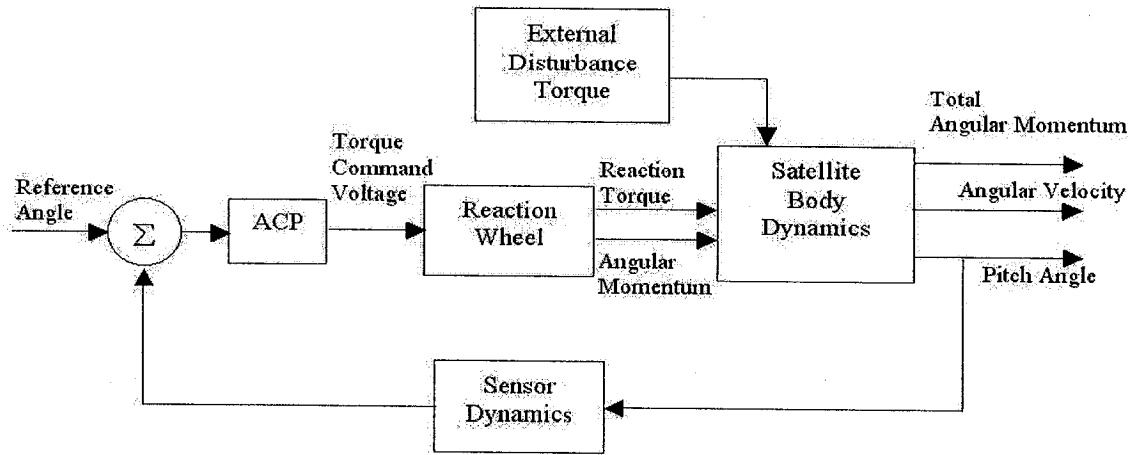


Figure 2.2: ACS Block Diagram

Spacecraft's attitude sensors can be of two categories [10]: *reference sensors* and *inertial sensors*. Reference sensors give attitude information with respect to some external references, which include the Sun, the Earth's IR horizon, local magnetic field direction and the stars [9]. Consequently, the most common reference sensors are: Sun Sensors, Earth Sensors (Horizon Scanners), Star Trackers and Magnetometers. However, there are normally periods of eclipse when reference sensor's attitude information is not available. To overcome this problem, inertial sensors (gyroscopes) are used to provide short-term attitude information between external updates or calibrations from reference sensors. Usually, the measurement system is formed using both reference sensors and inertial sensors to complement each other. Moreover, each vector measurement gives only two of the three pieces of information – as mentioned above – required for specifying spacecraft's attitude completely. This results in the need for using more than one type of sensors on-board.

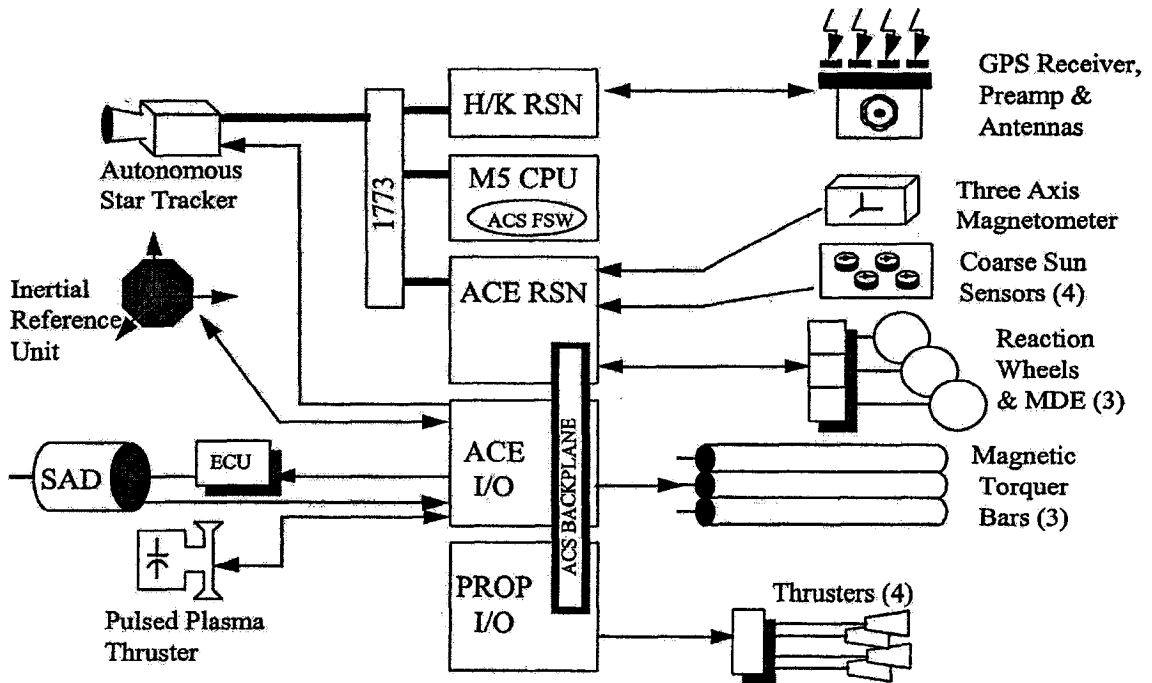


Figure 2.3: Typical ACS Components* [11]

As mentioned earlier, actuators may be of various types. Reaction wheels are electromechanical actuators and often a potential source of anomaly in the ACS. In the model that has been developed in this thesis, reaction wheel has been considered as the actuator in order to investigate ACS failure due to anomalies in the reaction wheel. However, in most cases, more than one type of actuators is used in satellites. Figure 2.3 shows the ACS components of NASA's EO-1 spacecraft. It should be mentioned here that often Solar Array Drive (SAD) and Spacecraft Antenna Mechanisms are also considered as the parts of ACS. A detailed description of the ACS architecture shown in Figure 2.3 is found on [11].

* SAD: Solar Array Drive, ECU: Electronic Control Unit (for SAD), RSN: Remote Services Node, M5: Mongoose 5 Processor, CPU: Central Processing Unit, FSW: Flight Software, 1773: Identification for a bus, ACE: Attitude Control Electronics, PROP: Propulsion, I/O: Input/Output, GPS: Global Positioning System, MDE: Motor Driver Electronics

2.1.1 ACS Modes

A spacecraft has a number of modes or states of operation [9]. The most common modes are as follows:

Orbit Insertion: This is the period during and after boost while spacecraft is brought to its final orbit.

Acquisition: Initial determination of attitude and stabilization of the spacecraft takes place in this mode. This mode is also used while recovering from upsets or emergencies.

Normal, On-station or Pointing: Spacecraft remains in this mode for the vast majority of its mission. Requirements of this mode are the key factors in system design.

Slew: Slew is a transitional mode for reorienting the spacecraft when required. Re-pointing requirements may demand for larger actuators than would be required for disturbance rejection alone.

Safe or Contingency: This mode, commonly called 'safe-hold' mode, is used in emergencies if regular mode fails or is disabled. Spacecraft may sacrifice normal operational performance requirements in this mode.

Others: There may be other operational modes depending on the additional requirement for pointing towards special targets or for some specific time periods such as eclipses. Consequently, most of the earth-orbiting spacecrafts or satellites have a mode 'Eclipse' during which the sun goes out of view of the spacecraft.

2.1.2 ACS Performance Requirements

ACS performance requirements may vary at different modes of operation of the spacecraft. Performance requirements are commonly specified in terms of *accuracy*, *jitter*, *settling time*, etc.

Table 2.2: ACS Performance Requirements [9]

Parameter	Definition	Examples/Typical Value
<i>Accuracy</i>	<u>Attitude Determination</u> : How well a spacecraft's orientation with respect to an absolute reference is known.	0.25 deg, $3\sigma^*$, all axes; may be real-time or post-processed on the ground.
	<u>Attitude Control</u> : How well the spacecraft's attitude can be controlled with respect to a commanded direction.	0.25 deg, 3σ , includes determination and control errors.
<i>Range</i>	<u>Attitude Determination</u> : Range of angular motion over which accuracy must be met.	Any attitude within 30 deg of nadir (earth).
	<u>Attitude Control</u> : Range of angular motion over which control performance must be met.	All attitudes, within 50 deg of nadir (earth), within 20 deg of sun.
<i>Jitter</i>	<u>Attitude Control</u> : A specified angle bound or angular rate limit on short-term, high frequency motion.	0.1 deg over 1 min, 1 deg/sec, 1 to 20 Hz; usually specified to keep spacecraft motion from blurring sensor data.
<i>Drift</i>	<u>Attitude Control</u> : A limit on slow, low frequency vehicle motion usually expressed as angle/time.	1 deg/ hr, 5 deg maximum. Used when vehicle may drift off target within infrequent resets.
<i>Settling Time</i>	<u>Attitude Control</u> : Specifies allowed time to recover from maneuvers or upsets.	2 deg max. motion, decaying to < 0.1 deg in 1 min; may be used to limit overshoot.

* Natural limit of random data variation produced by a process [12]

For a particular parameter, requirements for attitude determination and attitude control are different. Table 2.2 summarizes basic information about the parameters, which are used to specify ACS performance requirements.

2.1.3 ACS Computations

The ACS computer must perform reliably in the radiation environment in space. A number of such radiation-hardened computers now exist; further development is providing more power, speed and capability of being programmed in higher-level languages [10]. The availability of powerful computer means that spacecraft will be given greater autonomy and many of the sophisticated control techniques, which find applications in ground-based systems may be used on spacecraft. Robustness is another requirement for ACS and other on-board systems. For full autonomy or immediate response to any changes, adaptive control techniques are preferable.

2.1.4 Attitude Control Methods

Spacecraft attitude control methods may be broadly classified into three categories – *passive control*, *spin control* and *three-axis control* [9, 13].

Passive Control:

Gravity-gradient control utilizes the inertial properties of the spacecraft to keep it pointed towards the Earth, which is based on the fact that elongated object in a gravity field tends to align its longitudinal axis through the Earth's centre. Sometimes, a small, constant speed momentum wheel is added along the pitch axis. In another type of passive control,

permanent magnets are used on-board to force alignment along the Earth's magnetic field. Table 2.3 summarizes different attitude control methods and their capabilities.

Table 2.3: Attitude Control Methods and Their Capabilities [9, 13]

Control Type	Pointing Options	Typical Accuracy	Lifetime Expectancy
Gravity-gradient	Earth local vertical only	$\pm 5^\circ$ (2 axes)	Typically, > 10 years
Gravity-gradient and Momentum Bias Wheel	Earth local vertical only	$\pm 5^\circ$ (3 axes)	Limited by life of wheel bearings
Passive Magnetic	North/south only	$\pm 5^\circ$ (2 axes)	Typically, > 10 years
Pure Spin Stabilization	Inertially fixed any direction	$\pm 0.1^\circ$ to $\pm 1^\circ$ (2axes)	Limited by thruster propellant, if applies. Typically, 5–10 years
Dual-spin stabilization	Limited by articulation on de-spin platform	$\pm 0.1^\circ$ to $\pm 1^\circ$ (2axes)	Limited by thruster propellant, if applies and de-spin bearings Typically, 5–10 years.
Bias Momentum (1 Wheel)	Best suited for local vertical pointing	$\pm 0.1^\circ$ to $\pm 1^\circ$	Limited by thruster propellant, if applies. Also by sensors and wheel bearings
Zero Momentum (Thruster only)	No Constraints	$\pm 0.1^\circ$ to $\pm 5^\circ$	Limited by thruster propellant
Zero Momentum (3 Wheels)	No Constraints	$\pm 0.001^\circ$ to $\pm 1^\circ$	Limited by thruster propellant, if applies. Also by sensors and wheel bearings (Typically, 5–7 years)
Zero Momentum CMG	No Constraints	$\pm 0.001^\circ$ to $\pm 1^\circ$	Limited by thruster propellant, if applies. Also by sensors and wheel bearings

Spin Control:

In *spin control* is a passive control technique in which the entire spacecraft is spun. As a result, angular momentum vector remains approximately fixed in inertial space and gyroscopic stiffness provides stabilization about the transverse axes. The spinning spacecrafts possess inherent resistance to external disturbance torques. The downside is that extra fuel is required to re-orient the spacecraft because of the gyroscopic stiffness.

Dual-spin stabilization is a variation of *spin control* described above. In this case, the spacecraft has two sections spinning at different rates about the same axis. The major portion of the spacecraft is spun while only the payload or platform section is de-spun. By combining inertially fixed and rotating sections, *dual-spinners* can accommodate a variety of payloads in a simple vehicle.

Three-axis Control:

In three axis active control system, the major part of the spacecraft is de-spun. The payload is mounted on the main body or structure and the torques about the three axes required for attitude control come from the combinations of momentum wheels (MW), reaction wheels (RW), control moment gyros (CMG), thrusters, magnetic torque bars/rods (commonly known as MTB/ TR); as it was shown in Figure 2.2. Often, the same wheel can be used as a RW or MW. RWs have *zero* nominal speed whereas MWs have high nominal (typically, around 5000 – 6000 RPM). *Three-axis control/ stabilization* is necessary for achieving attitude control accuracy within 0.01 degrees.

In general, these *3-axis control* systems can be divided into two categories: (a) *Momentum-bias*, which has a momentum wheel along the pitch axis (b) *Zero-momentum* with a reaction wheel on each axis. *Momentum-bias* systems often have just one wheel along the pitch axis, normal to the orbit plane. The wheel is run nearly a constant high speed to provide gyroscopic stiffness to the spacecraft and attitude is controlled by varying the speed of the wheel slightly (typically, around $\pm 10\%$ of the nominal speed) which results in the variation in control torque.

In a *zero-momentum* system, RWs respond to disturbances on the spacecraft. Based on the attitude error(s), the ACS controller(s) generate signals which speed up the reaction wheel(s), which are initially at zero speed, in order to generate required ‘reaction’ torque(s) on the spacecraft body in an appropriate direction. These torque(s) correct vehicle’s attitude and leave the wheel spinning at a low speed until another pointing error (with same sign) speeds the wheel further or slows it down (error with opposite sign) again. While the RW may not reach its saturation speed for several orbits because of *cyclic disturbances*, *secular disturbances* may cause the wheel to reach its saturation speed. At this point, the wheel speed must be lowered back to *zero* by applying external torques using thrusters or magnetic torquers. This process is commonly known as *de-saturation*, *momentum unloading* or *momentum dumping*.

Thrusters may be used for momentum dumping and slewing at all attitudes. Control Moment Gyro (CMG) is an advanced technology utilizing a single wheel for attitude control along all three axes. A CMG is mounted on one or two gimbals depending on the required degrees of freedom and run at a constant speed [13]. Their control is somewhat complex; however, they have large torque capability with linearity at low power.

2.1.5 Attitude Motions for Three-axis Stabilized Spacecraft utilizing Reaction Wheels and with No Momentum Bias

The basic equations [10] related to the 3-axis stabilized zero-momentum systems utilizing reaction wheels will be presented in this section. The angular momentum \mathbf{H}_c of a single rigid body referred to its centre of mass C may be expressed as:

$$\mathbf{H}_c = [I_c] \cdot \omega \quad 2.1$$

Where ω is its angular velocity relative to an inertial (non-rotating) frame of reference and $[I_c]$ is the inertia matrix, based upon the centre of mass C , which can be expressed as:

$$[I_c] = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{yz} & I_{zz} \end{pmatrix} \quad 2.2$$

Where I_{xx} , I_{yy} , I_{zz} are the moments of inertia and I_{xy} , I_{yz} , I_{zx} are the products of inertia broadly representing a measure of the lack of mass symmetry, leading to a cross-coupled behavior. Using Equations 2.1 and 2.2, the angular momentum can be expressed as:

$$\mathbf{H}_c = \begin{pmatrix} (I_{xx}\omega_x - I_{xy}\omega_y - I_{zx}\omega_z) \\ (I_{yy}\omega_y - I_{yz}\omega_z - I_{xy}\omega_x) \\ (I_{zz}\omega_z - I_{yz}\omega_x - I_{yz}\omega_y) \end{pmatrix}$$

If principle axes are used, the products of inertia will be zero. Accordingly, \mathbf{H}_c can be expressed as:

$$\mathbf{H}_c = \{I_{xx}\omega_x, I_{yy}\omega_y, I_{zz}\omega_z\}^T$$

Angular momentum of a rigid body with spinning wheels, such as spacecraft fitted with momentum or reaction wheels, can be expressed as the sum of the angular momentum of the rigid body containing the wheels at their non-spinning rate, together with extra momentum due to the angular velocities of the wheels *relative to the body*. The angular momentum of a wheel will be equal to $J_w\omega_w$. For three wheels mounted on the three orthogonal axes, the additional momentum components due to the wheels will be $\{H_x, H_y, H_z\}$ and the total angular momentum of the spacecraft body plus wheels will be:

$$\mathbf{H}_c = \begin{pmatrix} (I_{xx}\omega_x - I_{xy}\omega_y - I_{zx}\omega_z) + H_x \\ (I_{yy}\omega_y - I_{yz}\omega_z - I_{xy}\omega_x) + H_y \\ (I_{zz}\omega_z - I_{yz}\omega_x - I_{yz}\omega_y) + H_z \end{pmatrix} \quad 2.3$$

If principal axes are used, Equation 2.3 becomes:

$$\mathbf{H}_c = \{(I_{xx}\omega_x + H_x), (I_{yy}\omega_y + H_y), (I_{zz}\omega_z + H_z)\}^T$$

It is also known that:

$$d(\mathbf{H}_c)/dt = \mathbf{T} ; \text{where, } \mathbf{T} \text{ is all external torques}$$

For an initially stationary spacecraft, a torque about a principal axis will produce a response about this axis, without any cross-coupling into the other axes. The response is an angular acceleration and for a particular axis, for example, pitch axis, it is given by:

$$\dot{\omega}_y = T_y / I_{yy}$$

2.2 Developed ACS Model in MATLAB-*Simulink*

In this thesis, a MATLAB-*Simulink* model of a generic ACS of a satellite has been developed. *Simulink* was chosen as the modeling tool because of its wide acceptance and growing demand in system modeling, analysis and design. Figure 2.4 shows the functional diagram for the MATLAB-*Simulink* model of the ACS which was described in Section 2.1 by Figure 2.1. Actual MATLAB-*Simulink* blocks for all the functional diagrams presented in this section are provided in the appendix of this thesis as noted next to the figure citations.

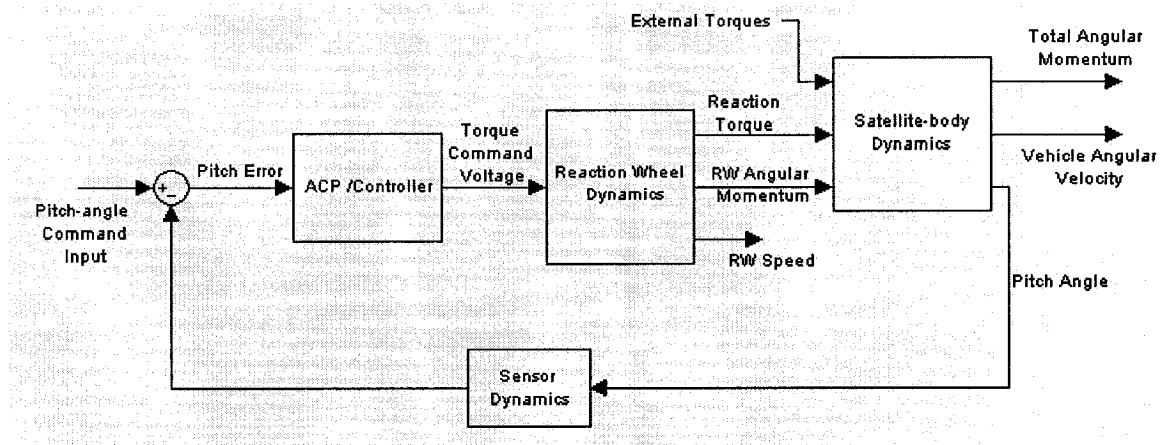


Figure 2.4: Functional Diagram of the *Simulink* Model of ACS (Figure A)

A *zero-momentum* system with reaction wheels has been modeled, which was described in details in Section 2.1.4 under *Three-axis Control*. The model has been developed for fault-diagnosis in the ACS along a single axis, i.e., along the pitch axis. Thus, it is not necessary to consider any *cross-coupling* effect. The actuator or reaction wheel block in the control loop of the developed ACS model is primarily based on the reaction wheel model presented in [14]. The model has been extended and modified in order to include

fault injection capabilities which will be presented in subsequent discussion. Also, an ideal dynamics for attitude sensors (described in Section 2.1) has been assumed, i.e., signals from sun sensors, horizon scanners, magnetometers, etc. have been fed back to ACP without any error or time delay. Therefore, the gain of the *Sensor Dynamics* block has been assumed to be 1.

Some general information about the ACP block in Figure 2.4 was provided in Section 2.1.3. The ACS control loop has been stabilized by utilizing a simple PID (Proportional, Integral and Derivative) control law. The PID controller inside the ACP block has been given the *pitch error signal* (θ_e) as the input. Controller parameters (K_p , K_i and K_d) have been tuned by trial and error in order to achieve best possible system response in terms of control *accuracy* (as described in Section 2.1.2 and Table 2.3), *overshoots* and *settling time*. The study has been performed assuming that the ACS is in its *Normal* or *Pointing* mode (as described in Section 2.1.1). The maximum allowable pitch error in this mode has been assumed to be ± 0.03 degrees. The output (Torque Command Voltage, v_c) of the controller is governed by the PID law:

$$v_c = K_p \cdot \theta_e + K_i \int \theta_e dt + K_d \cdot \dot{\theta}_e$$

The *External Disturbance Torque* block in Figure 2.4 has been implemented using the MATLAB-*Simulink* basic ‘source’ blocks to incorporate the possible external disturbance torques described in Table 2.1.

Subsequent discussion describes *Satellite Body Dynamics* and *RW Dynamics* (Reaction Wheel Dynamics) blocks of Figure 2.4 in details.

2.2.1 Satellite Body Dynamics

Figure 2.5 shows the functional diagram for the MATLAB-Simulink model of the satellite body or vehicle dynamics.

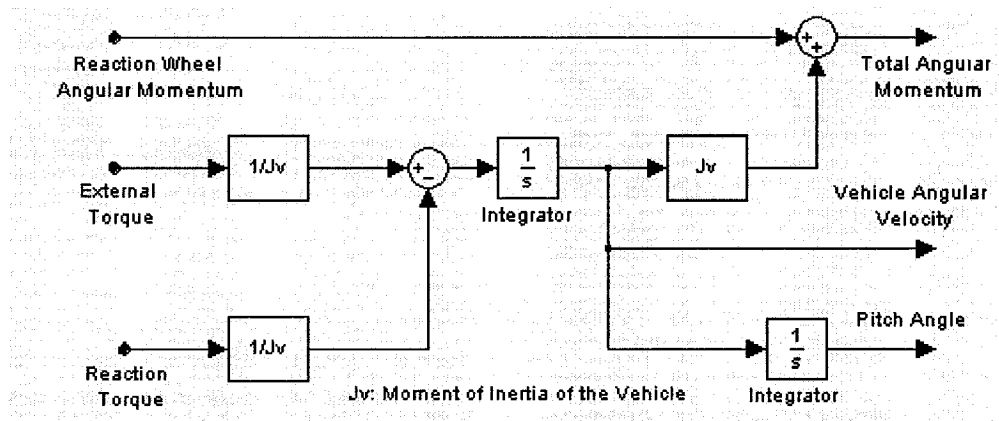


Figure 2.5: Satellite Body Dynamics (Figure B)

The equation of motion for the satellite body is:

$$\tau_{dy} = \tau_r + J_v (d\omega_v/dt)$$

where, τ_{dy} is the external disturbance torque, τ_r is the reaction torque from the wheel, J_v is the moment of inertia of the vehicle or satellite and ω_v is the angular velocity of the satellite. From this point onwards, moment of inertia will be denoted by J instead of I in order to avoid any confusion with *current* which is commonly denoted by ' I '. The above expression can be written in the form:

$$d\omega_v/dt = (1/J_v) \tau_{dy} - (1/J_v) \tau_r \tag{2.4}$$

From Equation-2.4, angular velocity ω_v of the satellite can be obtained by integration over time. Pitch angle can be obtained by integrating ω_v over time. Angular momentum of the satellite is:

$$H_v = J_v \cdot \omega_v$$

Total angular momentum will be equal to the angular momentum of the reaction wheel plus the angular momentum of the satellite body, i.e.,

$$H_{tot} = H_v + H_w$$

Where, $H_w = J_w \cdot \omega_w$; J_w is the moment of inertia of the reaction wheel and ω_w is the wheel speed. The Moment of inertia J_v of the satellite has been assumed to be 175 N-m-s²

2.2.2 Reaction Wheel Dynamics

Functional diagram for the MATLAB-*Simulink* model of the reaction wheel is shown in Figure 2.6. As shown in Figure 2.6, the net torque τ_n is equal to the motor torque τ_m minus and/or plus all the torques due to friction, torque noise and motor disturbances. Reaction torque τ_r is equal and opposite to net torque τ_n . Angular momentum stored in the reaction wheel can be expressed as:

$$H_w = J_w \cdot \omega_w$$

where, J_w is the moment of inertia of the reaction wheel and ω_w is the wheel speed. From the above relationships, we obtain:

$$\tau_r = -\tau_n = -\frac{dH_w}{dt} = -\frac{d(J_w \omega_w)}{dt} = -J \frac{d(\omega_w)}{dt}$$

$$\text{Therefore, } \omega_w = (1/J_w) \int \tau_n dt$$

Table 2.4 shows the various parameters for the *Ithaco Type-A* reaction wheel.

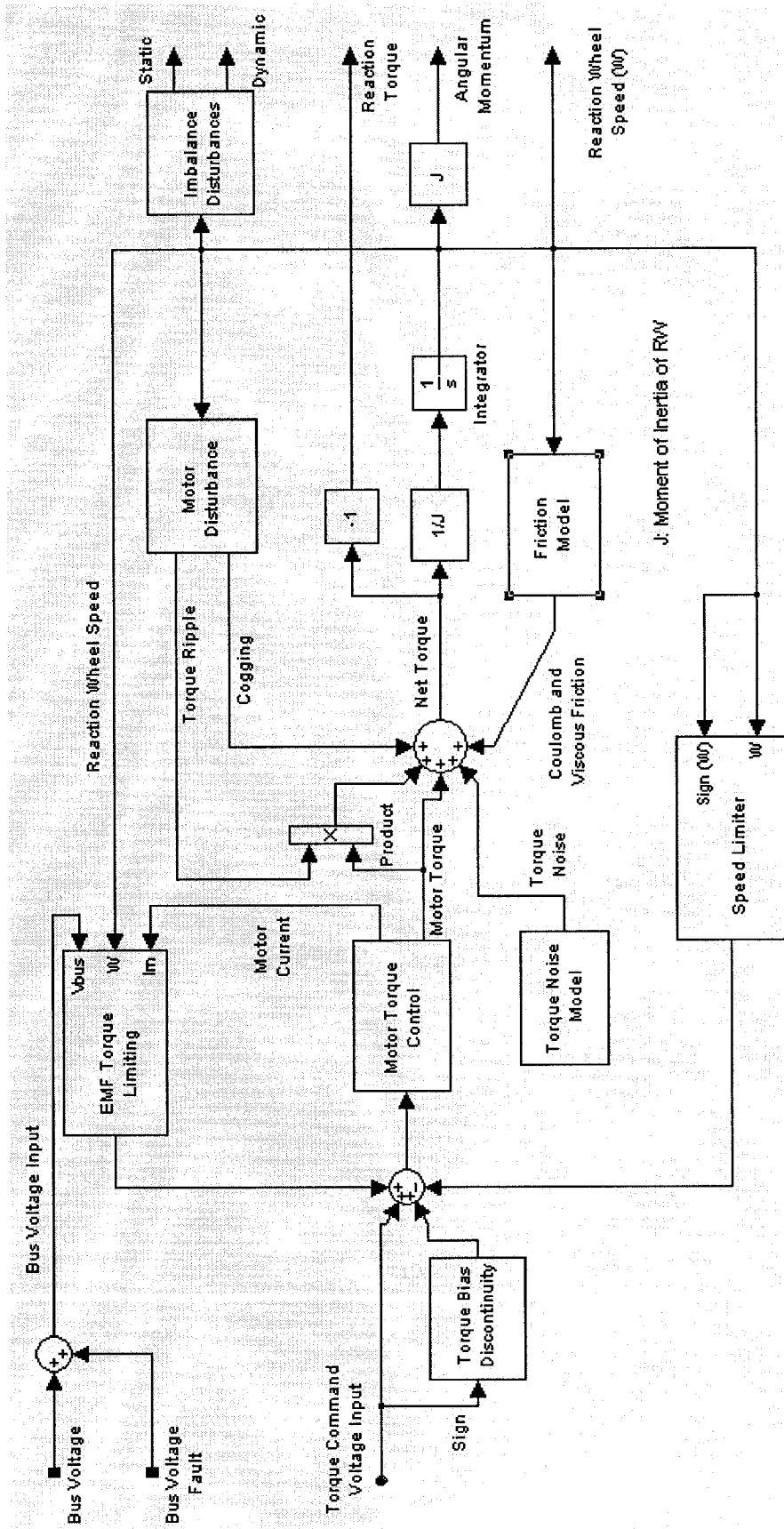


Figure 2.6: Ithaco Type-A Reaction Wheel Model (Figure C)

Table 2.4: Ithaco Type-A Reaction Wheel Constants [14]

Variable	Nomenclature	Unit	Value
τ_b	Torque Bias Discontinuity	N-m	zero
G_d	Motor Driver Gain	A/V	0.19
k_t	Motor Torque Constant	N-m/A	0.029
k_e	Motor Back-EMF Constant	V/rad/sec	0.029
k_s	Over-speed Circuit Gain	V/rad/sec	95
ω_d	Over-speed Circuit Threshold	rad/sec	690
		RPM	6600
τ_c	Coulomb Friction	N-m	0.002
J_w	Flywheel Inertia	N-m-s ²	0.0077
N	Number of Motor Poles	-	36
C	Cogging Torque Amplitude	N-m	zero
B	Motor Torque Ripple Coefficient	-	0.22
R_{IN}	Input Resistance	Ω	2.0
P_q	Quiescent Power	W	3.0
R_B	Bridge Resistance	Ω	2.0
-	Torque Command Range	V	± 5
-	Torque Command Scale Factor	N-m/V	0.0055
k_f	Voltage Feedback Gain	V/V	0.5
θ_a	Torque Noise Angle Deviation	rad	0.05
		degrees	3
ω_a	Torque Noise High Pass Filter Frequency	rad/sec	0.2
U_s	Static Imbalance	N-s ²	5×10^{-6}
U_d	Dynamic Imbalance	N-m-s ²	1×10^{-6}

Descriptions of the other blocks in Figure 2.6 are given below.

Motor Torque Control

Figure 2.7 shows the functional diagram for the *Motor Torque Control* block in Figure 2.6 in details.

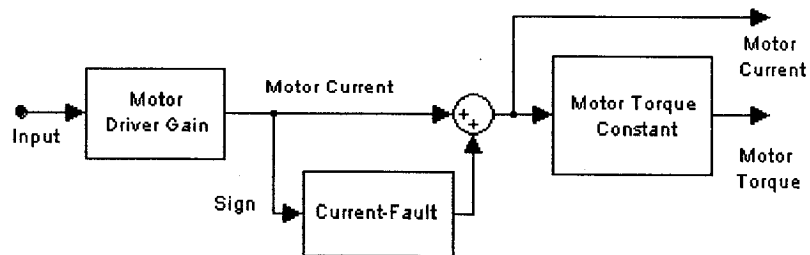


Figure 2.7: Motor Torque Control (Figure D)

It represents a voltage controlled current source with gain G_d and a motor with torque constant K_t (as given in Table 2.4). The function of this block is to generate motor current proportional to torque command voltage and to convert this current into torque by the motor torque constant K_t . A block has been added to incorporate fault injection into the motor current. This block takes the polarity of the motor current as input and injects error in the motor current using standard MATLAB-*Simulink* 'source' blocks such as *Random Source* and *Impulse Generator*.

Speed Limiter

Figure 2.8 shows the functional diagram for the *Speed Limiter* block in Figure 2.6 in details.

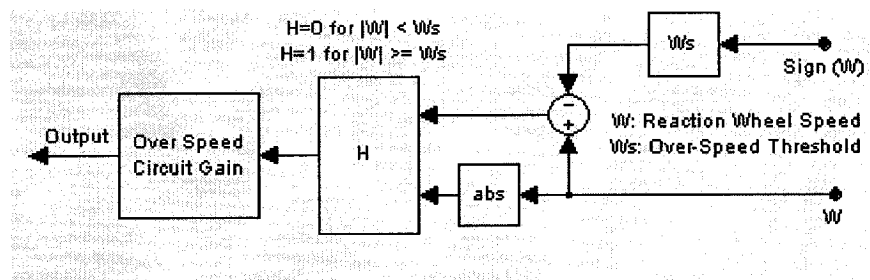


Figure 2.8: Speed Limiter (Figure E)

The function of the speed limiter block is to prevent the reaction wheel from rotating above maximum allowable speed. The actual wheel speed is compared with specified maximum allowable speed and an appropriate negative feedback is generated to lower the speed whenever the wheel speed exceeds the maximum allowable value.

EMF Torque Limiting

The purpose of this block is to model the limitation of the motor driver when the wheel runs at high speed at low bus voltage condition. At low bus voltage condition, if the wheel runs at high speed, motor torque may be limited because of the increasing back-EMF. From a disturbance point of view, the available motor torque at that point will be directly coupled with the bus voltage and any fluctuation in the bus voltage will be appearing as torque disturbance. Figure 2.9 shows the functional diagram for the *EMF Torque Limiting* block in Figure 2.6 in details.

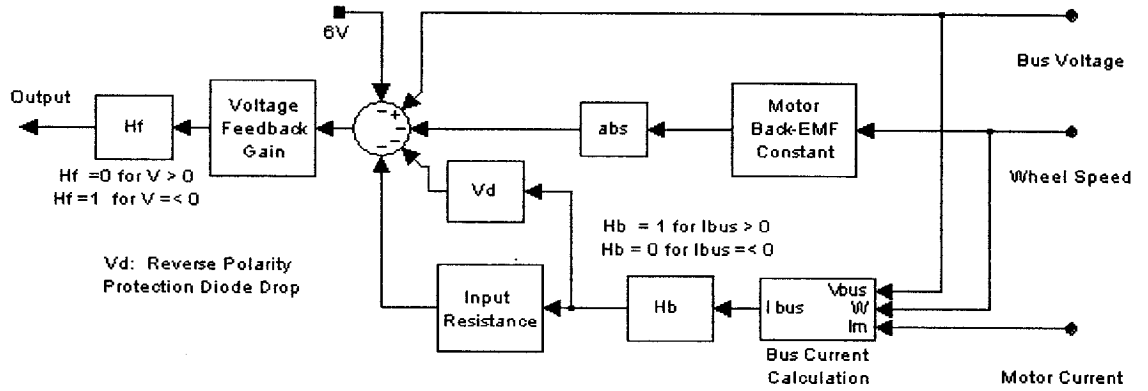


Figure 2.9: EMF Torque Limiting (Figure G)

Bus current I_{bus} , which is a function of bus voltage V_{bus} , motor current I_m and wheel speed ω , is given by [14]:

$$I_{bus} = \{1 / (V_{bus} - 1)\} \cdot \{I_m^2 R_B + 0.04 |I_m| V_{bus} + P_q + I_m k_e \omega\} \quad 2.5$$

The constants in this equation are given in Table 2.4. Equation-2.5 has been implemented inside the I_{bus} Calculation block in Figure 2.9. The block H_b has been included inside this block to eliminate the voltage drop when the power is not drawn from the bus. Detail of the *EMF Torque Limiting* block is found in [14].

Bus Voltage

Normal range for bus voltage was assumed to be 21 – 28 volts *D.C.* [15]. A block has been added to incorporate fault injection into the bus voltage. The function of the V_{bus} *Fault* block is to lower the amplitude of the bus voltage below the normal range *slowly* using a standard MATLAB-*Simulink* ‘source’ block: *Impulse Generator*.

Friction Model

Torque due to friction in the reaction wheel has two components: torque due to viscous friction τ_v and torque due to coulomb friction τ_c . τ_v is a function of wheel speed and temperature while τ_c is constant with polarity dependence on the direction of rotation of the wheel. MATLAB-*Simulink* friction block has been used to model these torques. Slope for the τ_v curve has been taken from Figure 4 of [14]. Figure 2.10 shows the functional diagram for the *Friction Model* block in Figure 2.6 in details.

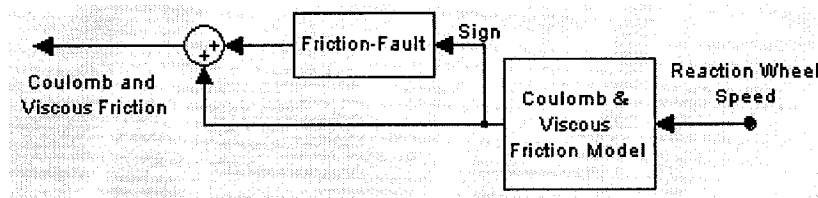


Figure 2.10: Friction Model (Figure F)

A block has been added to incorporate fault injection into the wheel bearing friction. This block takes the direction of the friction torque as input and makes the amplitude of this torque for some pre-defined time period (will be discussed in Chapter 4) using standard MATLAB-*Simulink* ‘source’ blocks such as *Impulse Generator*.

Torque Noise

Torque noise is a very low frequency torque variation from the bearings due to lubricant dynamics [14]. It is a function of lubricant behavior and it has the most significant effect on satellite pointing. It can be specified as a deviation from the ideal location of the rotor at any constant speed. Equation-2.6 [14] has been implemented inside the *Torque Noise* block in Figure 2.6.

$$\tau_a = J_w \Theta_a \omega_a^2 \sin \omega_a t \quad 2.6$$

The constants of the Equation-2.6 are specified in Table 2.4. Detail information on the *Torque Noise* block is available in [14]

Motor Disturbances

Motor disturbance has two parts: torque due to ripple in the motor torque and cogging torque. Torque ripple is amount of variation in the motor torque due to commutation method and shape of the back-EMF [14]. Cogging is always present in conventional brush-less DC motors because of the change in reluctance of the iron stator due to the rotation of the magnets inside them. However, it is possible to eliminate cogging disturbances completely by using ironless armature. *Ithaco Type-A* reaction wheel motor eliminate cogging disturbances completely. Equations 2.7 and 2.8 [14] have been implemented inside the *Motor Disturbance* block in Figure 2.6. The constants of the Equations 2.7 and 2.8 are specified in Table 2.4

$$\text{Ripple Torque:} \quad \tau_{rip} = B \sin 3N\omega_w t \quad 2.7$$

$$\text{Cogging Torque:} \quad \tau_{cog} = C \sin (N/2) \omega_w t \quad 2.8$$

Imbalance Disturbances

The imbalance of the flywheel causes imbalance disturbance in satellites. It is often considered as the most significant source of disturbance from the reaction wheel or momentum wheel. *Static Imbalance* U_s (specified in Table 2.4) is the offset of the center of gravity of the flywheel from the rotation axis and *Dynamic Imbalance* U_d (specified in Table 2.4) is the cross product of inertia of the flywheel, caused by angular misalignment of the principal inertia with the spin axis. Detail information on *static* and *dynamic imbalance* is available in [14]. Equations 2.9 and 2.10 have been implemented inside the ‘Imbalance Disturbances’ block in Figure 2.6.

$$\text{Static Imbalance Force:} \quad F_{x,y} = U_s \omega_w^2 \sin \omega_w t \quad 2.9$$

$$\text{Dynamic Imbalance Torque:} \quad \tau_{x,y} = U_d \omega_w^2 \sin \omega_w t \quad 2.10$$

Torque Bias Discontinuity

If the polarity of the commanded motor torque is reversed, or the wheel speed goes through *zero*, significant torque discontinuities and/or temporary torque drop-outs may result if there is any timing present in the quadrant detection schemes in the motor driver during quadrant changes. *Ithaco Type-A* reaction wheel motor drivers are designed to eliminate any need for quadrant detection. As a result, they provide seamless torque command polarity changes and smooth transitions through *zero* speed. Therefore, the gain of the *Torque Bias Discontinuity* block in Figure 2.6 has been set to *zero*.

2.3 Summary

In this chapter, a review on the Attitude Control Subsystem (ACS) of a satellite has been presented along with the description of the MATLAB-*Simulink* model of the ACS that has been developed in this thesis. In the next chapter, a review on fault-tree synthesis and analysis will be presented along with the proposed algorithm for fault-tree synthesis. Afterwards, in Chapter 4, different ACS failure scenarios will be presented and fault-trees will be constructed under each failure scenario using the proposed fault-tree synthesis algorithm.

Chapter 3

Fault-Tree Synthesis and Analysis: An Overview and Proposed Approach

The first part (Section 3.1) of this chapter gives a brief introduction to fault-trees and provides a review on fault-tree synthesis and analysis methodologies. The next part (Section 3.2) discusses an approach for fault-tree synthesis called 'IFT' along with a method for induction of decision trees known as 'ID3'. Finally, in Section 3.3, the fault-tree synthesis algorithm, which has been developed in this thesis based on IFT and ID3 concepts, is presented.

3.1 Introduction to Fault-trees

The concept of fault-tree evolved primarily within the U.S. aerospace and nuclear industries. Fault-trees have been extensively used in system safety and reliability analysis, as well as in system fault diagnosis, for more than 40 years [16]. The purpose of fault-trees is to translate the failure behavior of a physical system into a visual diagram in which a very simple set of rules, logics and symbols provides a mechanism for analyzing very complex systems.

The fault-tree for any failure analysis has a basic structure as shown in Figure 3.1. The top event in a fault-tree is the failure, which is to be analyzed. The basic events are the occurrences beyond which there is no further interest for analysis. The basic events (often

called leaves) are connected to the top events through some intermediate events (often called nodes) which can show how fault(s) propagated into the system and led to a failure. The top event in the fault-tree has to be foreseen by the analyst. The event(s) in one level of a fault-tree are connected to the event(s) at the next level of the tree through some logic gates (*AND*, *OR* etc.), as shown in Figure 3.1.

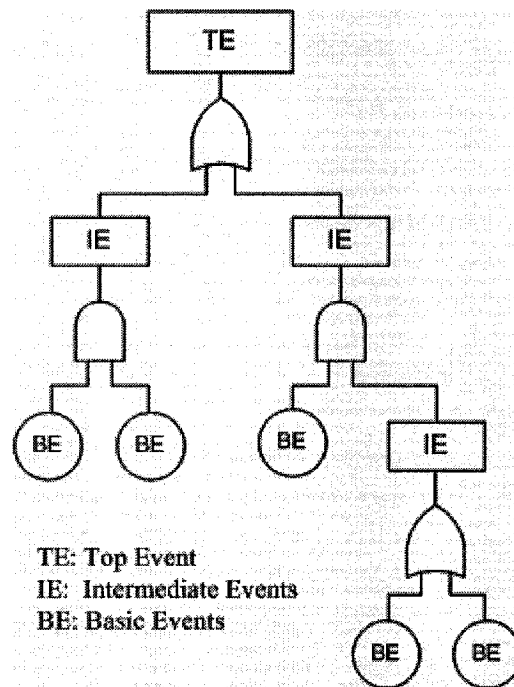


Figure 3.1: Basic Fault-Tree Structure

Fault-trees, that use the traditional logic gates mentioned above, are not capable of representing the temporal relationships among the events and dynamic behavior of the system under consideration. Where the sequences of the occurrences of events are necessary for analysis, dynamic fault-trees can be constructed. Usually, the kind of dynamic system behaviors that make it necessary to use the dynamic fault-trees for computer based systems include: sequence dependencies, shared pool of resources,

‘warm’ and ‘cold’ spares, etc. [17]. Dynamic fault-trees are essentially an extension of traditional or static fault-trees with special-purpose gates to capture the system dynamics.

The whole problem of fault-tree* can be divided into two parts – fault tree synthesis (FTS) or construction and fault tree analysis (FTA). As mentioned above, FTS and FTA find their application in System Reliability Analysis, System Safety Analysis and System Fault-diagnosis. In this thesis, fault-trees have been used as a diagnostic aid for fault diagnosis and health monitoring in the Attitude Control Subsystem (ACS) of a satellite.

It is important to point that fault-tree is not a complete representation of all possible faults and failures in a system. It is usually capable of representing the combinations of events for a failure, which have been foreseen by the analyst. It should also be noted here that FTA is primarily a means for analyzing the cause of a failure. Therefore, the top event in a fault tree should be detected by other mechanism. Hence, the existence of an efficient fault detection mechanism has been assumed here.

Fault-tree generation or synthesis is considered to be a relatively difficult problem. This is mainly due to the fact that a very good understanding of the system is often required for FTS. Though manual synthesis of fault tree is still common in today’s industries, for large and complex systems, manual synthesis is often not feasible. For complex systems, manual construction of fault-trees can be extremely time-consuming and expensive and is likely to lead to human errors. This is due to the fact that if two persons analyze a system,

* In the remaining part of this thesis, the word ‘fault-trees’ will be used to mean static fault-trees unless otherwise specified.

the results are never the same. One may overlook some analysis the other has performed. Moreover, the forms of the resulting fault-trees may be different; and the terminology used to describe the failure may also be different. Therefore, the automation of fault-tree synthesis is necessary from the point of cost reduction, result standardization and also for understanding the construction process. In order to automate the fault-tree synthesis, it is necessary to build a consistent methodology which can be implemented in computer programs. Existing methodologies for automatic fault-tree construction is presented in the next section.

3.1.1 Automated Fault-Tree Synthesis (FTS) Methodologies

The process of automating the fault-tree synthesis task was initiated in 1970's by J. Fussell [16]. Since then, numerous attempts have been made by many researchers to create computer programs for automatic fault-tree construction using the different tree-synthesis approaches they had developed. In this section, a short review of the existing dominant fault-tree synthesis methodologies will be presented.

The author in [18] presented a formal methodology for automatic synthesis of fault-trees, commonly known as the *synthetic tree model*, for electrical systems. This method uses a set component-failure transfer functions as models for the components and their various failure modes to construct the final fault-tree in a 'top-down' fashion. The authors in [19] addressed the issues related to the formal synthesis methods for fault-trees.

The Fault-tree Handbook [20] provides basic concepts of fault-trees. It contains information on fault-tree construction fundamentals, including basic rules for tree-synthesis, fault-tree evaluation techniques and the application of probability and statistics theory as well as Boolean algebra for fault-tree analysis.

In order to synthesize fault-trees automatically, authors in [1] represented the system under consideration using digraphs (directed graphs). A digraph is a set of nodes connected by edges. In fault-tree synthesis, nodes of digraphs represent process variable and certain type of failures. If a deviation in one variable causes a deviation in another variable, then a directed edge is drawn from the node representing the first variable to the node representing the second one. Also, a number is assigned to the edge depending on the direction and magnitude of the second deviation relative to the first. Finally, fault-tree synthesis is formulated using a state-space representation. The initial state in the fault-tree synthesis is a definition of the top event along with a description of the process (in the form of a digraph). The 'goal state' is a fault-tree connecting the top event to the events (basic events or *primal* events), which are not developed further. Transformation from one state to another is accomplished by using appropriate operators.

Another classical paper on automatic synthesis of fault-tree is [21], where the author introduced the concept of mini fault-tree models for system components. One big advantage of this approach is that the mini fault-tree models can be used repeatedly in different failure analysis studies. In this approach, a component functional and failure model consists of a set of mini fault-trees. Each mini fault-tree consists of an input event, a set of component conditions, and a set of output and state-change events. The algorithm

starts by taking a localized hazard event within a particular component selected by the user. This event is considered as the top event in the tree. Component mini fault-trees with this event as output are searched in the library and added to the fault-tree (or *failure-tree*, as called by the author) as inputs to an *OR* gate. Then each of these mini fault-trees is selected in turn. The process continues by adding mini fault-trees to the *failure-tree* until only spontaneous events and normal states exist as the leaves of the tree. The algorithm in [21] for fault-tree synthesis differs from the one in [1] in that the former works on a component by component basis rather than loop by loop basis. Also, the former works directly from the *system flow sheet*. The treatment of loops on a component by component basis is a new feature of this algorithm which gives the capability of treating multiple loops of unlimited complexity.

The authors in [22] presented another new approach for constructing fault-trees for electrical systems or circuits by representing them by graphs in which connections between components are represented by vertices and paths through the components are represented by arcs. This approach is quantitative and uses backtracking. Bigger circuits are decomposed into sub-circuits using a decomposition technique utilizing graph theory. This breaks down the problem into smaller pieces and solves the problem of handling loops in the circuits. Technical descriptions of the top events are derived and the inputs to the circuits are determined. The backtracking approach considers one component at a time going from output to input where each level in the tree corresponds to one of these components. The tree generation algorithm is to be consistent with this approach. Finally, in order to handle the time, the authors proposed that a time range or interval can be specified on the voltage and current for specifying the time requirements for the top

event. In the backtracking process, when the inputs to the circuit are reached, the required input and the actual input must be compared to determine if the time requirements on the voltage and current are met as well as their magnitude requirements.

The authors in [23] pointed out that in most of the above-mentioned methodologies, well-grounded discrete event system formalism to represent component models had not been adopted and timing concept had been introduced only in a 'rough sense'. To overcome these problems, they presented an automated fault-tree generation methodology using symbolic DEVS (Discrete Event System Specification) simulation. In contrast to the conventional backward (back-track) approaches, their method is a forward approach because is generated through simulation. Symbolic DEVS is an extension of conventional DEVS. In conventional DEVS, time is expressed in real numbers; whereas in symbolic DEVS, time is represented by linear polynomials over the real numbers. This allows manipulation of expressions for time with symbols representing unspecified event times. Finally, the precise time information can be obtained by setting its symbol to a real value. The symbolic simulation starts by injecting a given input command into the model structure. After that, every possible event – both faulty and non-faulty – is investigated by propagating events through the next components depending on its coupling and timing relations until it satisfies the goal condition i.e., the top event. In this way, this method intrinsically represents timing effects.

In [24], the authors described a dynamic fault-tree modeling techniques for handling some difficulties in the reliability analysis of fault-tolerant computer systems. Since the modeling is dynamic, they incorporated four special types of gates in the fault-tree

models to capture the dynamic behavior of the system. Continuing along the same theme, the authors developed a methodology for automatic synthesis of fault-trees for computer-based systems in [25] where dynamic fault-trees were constructed from a RIDL (Reliability Imbedded Design Language) system model. Their goal was to perform the reliability analysis at an early stage in order to avoid costly design changes. Afterwards, in [26], the authors presented a methodology for automatic generation of a diagnostic expert system for system fault-diagnosis.

The authors in [27, 28] presented a methodology for fault-tree synthesis from MATLAB-*Simulink* models in order to assist the safety and reliability analysis for model-based product development in the automotive industry.

Another case where fault-trees were used for fault-diagnosis is found in [29]. The authors describe a methodology to generate fault-trees automatically for operational fault-diagnosis. The diagnosis system was developed for on-line diagnosis of modern trains based on an expert system approach for off-line generation and optimization of fault-trees using *case-based* reasoning.

The authors in [30] presented an approach for automatic fault-tree construction called *Induction of Fault-Trees* (IFT). In their approach, a detailed knowledge of the system design, construction and operation is not necessary. The fault-tree synthesis is performed from the database of example vectors representing system behaviors under faulty and non-faulty conditions. Their fault-tree synthesis algorithm utilizes machine learning technique and is based on Quinlan's ID3 algorithm [31] for induction of decision trees. In

[32], the authors developed a diagnostic system using constructed fault-trees. The IFT approach will be discussed in detail in Section 3.2

3.1.2 Fault-Tree Analysis

Once the fault-tree synthesis is complete, both qualitative and quantitative analysis [1, 21, 33, 34] can be performed. However, since this thesis is mainly focused towards fault-tree synthesis (FTS), only a brief discussion on fault-tree analysis (FTA) is presented in this section for the sake of completeness.

Qualitative Analysis: The purpose of performing qualitative analysis on a fault-tree is to reduce the constructed tree to a logically equivalent one showing the specific combinations of basic events which are sufficient to cause the top event. Therefore, in qualitative analysis, the intermediate events are removed and only the relationships between top event and basic events are shown. These are commonly known as the *cut sets*. The ultimate goal of the qualitative analysis is to find the *minimal cut sets*.

A *minimal cut set* in a fault tree represents a collection of basic events all of which are necessary and sufficient for the top event to take place by that minimal cut set. Therefore, even if one basic event in the cut set does not occur, the top event in the tree will not take place. If there are OR gates in the tree, same primary events usually occur in more than one of the minimal cut sets. Consequently, minimal cut sets are generally not independent of each other. A medium size fault-tree can have millions of minimal cut sets [34]. Consequently, computer programs are necessary to calculate minimal cut sets.

Quantitative Analysis: The objective of performing quantitative analysis on a fault-tree is to calculate, using the minimal cut sets, the probability of the occurrence of the top event from the probability of the basic events. So, information on the events occurring in the tree is required for performing such analysis.

Cut-set analysis is a classical approach for fault-tree analysis. The authors in [35] pointed out that the sometimes the determination of minimal cut sets can be very time consuming process even on modern computers. They described an alternative approach that uses Binary Decision Diagrams (BDD) for fault-tree analysis and ways through which the technique can be implemented on a computer. A BDD, as described in [36], is a directed acyclic graph in which all paths begin at the root vertex and terminate in one of the two states: (1) a '0 State' (system success) (2) a '1 State' (system failure). A BDD is composed of *terminal* and *non-terminal* vertices connected by branches. In contrast to the basic events representation in the fault-trees, the *non-terminal* vertices, as represented by $X1, X2, X3, X4$ in Figure 3.2, of a BDD correspond to the basic events of the fault-tree from which it is derived. In short, the aim the approach is to convert fault-trees into BDDs for increasing the efficiency in the computation of both the minimal cuts and the probability of its root events. However, a major problem with the BDDs is that variable ordering has a crucial effect of the size of the BDD. Finding an appropriate ordering scheme that is capable of producing BDDs for all fault-trees is a difficult task. In order to handle the complexity of this ordering problem, genetic algorithm, machine learning and neural network based techniques are used [36, 37].

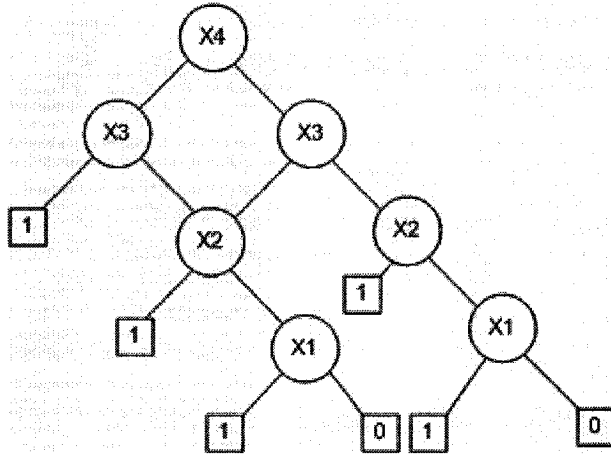


Figure 3.2: A Binary Decision Diagram (BDD) [36]

The authors in [17] presented a unified methodology for solving the fault-trees. A large system level fault-tree is decomposed into static and dynamic fault-trees (sub-trees). The disadvantage of converting the entire system fault-tree into an equivalent *Markov chain* is that the size of the resulting Markov model, which is used to solve dynamic trees, becomes too large. They used Binary Decision Diagrams (BDD) for solving the static sub-trees and *Markov methods* for solving the dynamic sub-trees.

3.2 IFT Approach for Fault-tree Synthesis

As mentioned in Section 3.1.1, the authors in [30] described an alternative to the traditional approaches for fault-tree synthesis. It is a machine-learning method for automatic generation of fault-trees for simulated incipient faults in dynamic systems. A significant aspect of this approach is that detailed knowledge or analysis of the system under consideration is not required. The algorithm constructs trees from a database of example vectors which represent the system behavior in presence of undesired events as well as under normal system condition. These example vectors are formed by extracting

features from the data. The authors [30] used Fast Fourier Transform (FFT) based feature extraction in their study of a servomechanism for machine tool applications. The algorithm they developed was based on the ID3 algorithm [31] for induction of decision trees. A decision tree is a way of representing classification rules. The resulting fault-trees by the algorithm in [30] are essentially a type of decision trees that classify systems on the basis of whether the systems do or do not exhibit symptoms of a particular fault. Generated fault-trees were used to identify the system states by classifying the simulated data on the basis of the severity of the injected fault.

In the remaining part of this section, basic idea of the ID3 algorithm, from which the IFT approach was derived, is described for better understanding of this concept. Also, two important issues – *Feature Extraction* and *Attribute Selection Sequence* – have been addressed. Finally, in Section 3.3, the fault-tree synthesis algorithm which has been developed in this thesis based on these concepts will be presented.

3.2.1 The ID3 Algorithm

The ID3 algorithm [31] was designed for constructing reasonably good decision trees without much computation where many attributes and many objects are there in the data/training set. The purpose of decision trees was to classify objects that are described in terms of a collection of attributes. In this method, decision trees are constructed by an induction task from a ‘training set’ which contains collections of attribute values for different classes. The tree construction process starts from the root of the tree and proceeds down to its leaves. Leaves of such decision trees represent class names and the

other nodes represent attribute-based tests. Each branch from these nodes represents the outcome of the test. Though ID3 was found to construct simple decision trees, the approach it uses cannot rule out the possibility of overlooking better trees [31].

The induction task which is performed by ID3 should be explained in more detail. The induction task is being explained here through the example of the object *Saturday Mornings* [31]. Table 3.1 contains the attributes for a *Saturday morning*. Fourteen example vectors in this table were formed by considering different ‘values’ for the four attributes: outlook, temperature, humidity and wind which might be sufficient to classify the object i.e., a particular Saturday morning. Depending on the attribute values, *Saturday morning* has been classified into two classes: ‘N’ (negative instances) and ‘P’ (positive instances) or simply ‘bad’ and ‘good’ respectively.

Table 3.1: Training Set for Induction Task [31]

Attributes				Class
Outlook	Temperature	Humidity	Windy	
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

A simple decision tree which classifies the information given in Table 3.1 is shown in Figure 3.3.

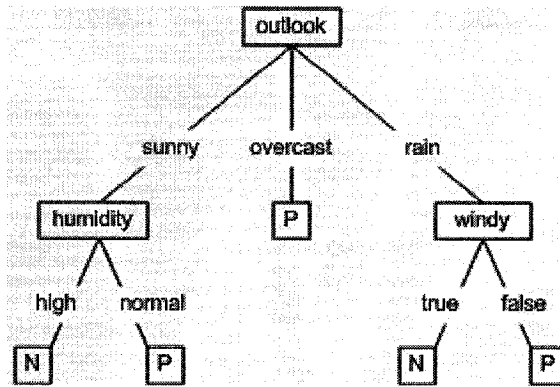


Figure 3.3: A Simple Decision Tree [31]

Given adequate attributes, it is always possible to construct a decision tree that classifies the data correctly. However, if the example set contains two or more classes that have identical values for each attribute but belong to different classes, it will never be possible to differentiate between the classes based on the given attributes. In such cases, attributes are considered to be ‘inadequate’ for the induction task.

3.2.2 Feature Extraction

As mentioned in Section-3.2, the IFT approach for fault-tree synthesis uses a set of example vectors which is formed by extracting features from the attribute. For fault-diagnosis applications, the main purpose of feature extraction is to find the symptoms of the presence of anomaly in the system. Symptoms and evidences are necessary for fault-diagnosis, failure analysis and any accident or incident investigation. Essentially, in the tree-synthesis process, attributes are ‘tested’ sequentially for symptoms of one or more anomaly in the system. Another advantage of extracting features from the data is that it drastically reduces the amount of data to be handled; i.e., feature extraction, in general,

maps a large problem space in into a smaller feature space. This mapping is done by applying appropriate feature extraction function(s) on the available data.

Selection of an appropriate feature extraction function is a difficult task and can be done in several ways. The most commonly used techniques include (a) Spectrum analysis by N-point DFT (Discrete Fourier Transform) (b) Standard Control Systems specifications such as natural frequency, peak value, percentage overshoot, settling time etc. (c) Curve fitting techniques. In this thesis, the second approach has been followed and feature extraction has been performed by using simplest feature extraction functions such as peak value (*MAX*) and minimum value (*MIN*) of the time domain signals or attributes (see Chapter 4) to demonstrate fault-tree synthesis by the proposed algorithm.

3.2.3 Attribute Selection Sequence

In order to construct decision trees by ID3 or fault-trees by the IFT approach described earlier in Section 3.2, the sequence in which the attributes are selected, plays a crucial role on the size of the resulting trees. This is due to the fact that some attributes divide the data more clearly than the others. It appears from the reviewed research works that there are no rule-based means of determining the ‘best way’ for attribute selection. In this thesis, a rigorous study on attribute selection techniques has not been performed. A very common and simple attribute selection criterion based on ‘Entropy’ (will be discussed later in this section) has been used (see Chapter 4) to demonstrate fault-tree synthesis by the proposed algorithm.

The author in [31] described ‘information-based’ attribute selection heuristics. One of them calculates the *Information Gain* for each attribute in the unit of bits and selects the attribute with the largest information gain as the first attribute; i.e., attribute selection is done in descending order of the information gains. Detail of this calculation can be found in [31]. The authors in [30] used this information gain heuristics for determining the attribute selection sequence in their study. However, in another study it was reported that the *Information Gain* criteria ‘tends to favor’ attributes with many values [31], i.e., the criterion is biased by such attributes. Another criterion called *Gain Ratio* criterion suggested in [31] overcomes this problem.

Another widely used technique for attribute selection is to select them based on their entropy value. Entropy is a common concept in many fields of research. One way to calculate the entropy $E(A)$ of an attribute A over some interval is to use well-known *Shannon’s* entropy:

$$E(A) = - \sum_i A_i^2 \log(A_i^2) \quad 3.1$$

Where, $\log(X)$ is the natural logarithm of X . In this thesis, the percentage change in entropy between non-faulty and faulty scenario is considered as an attribute selection criterion. Entropy is calculated using Equation 3.1 for each attribute over a pre-defined time window (as discussed in Section 1.5) during normal operating condition of the system and under presence of anomaly. Then the percentage changes between these two entropies are calculated and attributes are selected in the descending order of the percentage changes.

3.3 Proposed Algorithm for Fault-Tree Synthesis

In this section the automatic fault-tree synthesis algorithm developed in this thesis is presented. The algorithm has been developed by continuing along the same theme as that in [30] and [31]. Consequently, the diagnosis process utilizing the proposed algorithm does not depend on the design phase of the system, i.e., in depth design and construction knowledge is not necessary.

The proposed algorithm for FTS adopts and transforms the one presented in [30] to suit our requirements. This algorithm also generates fault trees from vectors of numeric feature values of the attributes by induction. However, there are some important differences between these two algorithms. First, instead of using a single threshold point for a numeric feature value, we have used more than one range of values for the features of some of the attributes. The advantage here is that a range is relatively easy to determine compared to an exact threshold point for a feature value. In order to specify such thresholds, one has to know exactly at what point the transition will take place from a non-faulty to a faulty state. For practical purposes, specifying ranges for attributes are much easier and for operational fault-diagnosis purpose, the operators or engineers involved in analysis will be able to specify these ranges from their operational experience on the system.

Secondly, the algorithm in [30], on selecting a feature value V_x of an attribute A_x with threshold T_x , considers both the cases when $V_x < T_x$ and $V_x \geq T_x$. This results in a generalized tree for a particular top event which allows one to evaluate all possible

Proposed FTS Algorithm:

- A. Form a *TOP NODE* with the undesired event.
- B. 1. Select an attribute A_x . Read its current feature value. Determine the pre-defined range i which the current feature value belongs to.
2. a. (Only if executing step-B for the first time):
If $F(A_x)$ belongs to a single range, connect an *AND* gate with the Top Node and place a node $(F(A_x) \in i)$ at one input branch of the *AND* gate.
- If $F(A_x)$ belongs to more than one range, connect an *OR* gate with the *Top Node* and place all $(F(A_x) \in i)$ nodes at the inputs of the *OR* gate. Number of $(F(A_x) \in i)$ nodes will be equal to number of pre-defined ranges $F(A_x)$ belongs to.
For each $(F(A_x) \in i)$ node, replace the node by an *AND* gate place $(F(A_x) \in i)$ at one input branch of the *AND* gate
- b. (Only if back from B.3a):
Place $(F(B_x) \in i)$ at one input of the *AND* gate.
- If $F(B_x)$ belongs to more than one range, connect an *OR* gate with the *AND* gate and place all $(F(B_x) \in i)$ nodes at the input of the *OR* gate.
For each $(F(B_x) \in i)$ node, replace the node by an *AND* gate and place $(F(B_x) \in i)$ at one input branch of the *AND* gate
- c. (Only if back from B-3c):
For each $(F(B_x) \in j)$ node, replace $(F(B_x) \in j)$ by an *AND* gate and place $(F(B_x) \in j)$ at one input of the *AND* gate.
3. To develop the other branch of each *AND* gate, consider all the examples in the example-set where, $F(A_x) \in i$.
- a. If all examples are *FALSE*, remove the $(F(A_x) \in i)$ node. Select next attribute B_x by going back to step-B.1 and executing the algorithm recursively.
If no more attribute is available, connect an 'Unknown Event' leaf with the remaining input of the *AND* gate and STOP.
- b. If all examples are *TRUE*, then STOP.
- c. Otherwise, to develop the other branch of the *AND* gate, select next attribute B_x and place a node $(F(B_x) \in i)$, (where i is range for $F(B_x)$ in the current example), at the other input of the *AND* gate. Repeat step-B.2 onwards recursively for B_x .
- If $F(B_x)$ belongs to more than one range, connect an *OR* gate with the remaining input of the *AND* gate and place all $(F(B_x) \in i)$ nodes at the input of the *OR* gate. Repeat step-B.2 onwards recursively for B_x .
- If no more attribute is available, connect an 'Unknown Event' leaf with the remaining input of the *AND* gate and STOP.
- C. Remove all single-input *OR* and *AND* gates.

combinations of events behind a failure that may provide useful information into future design to avoid other unforeseen combination of events that could lead to the failure scenario. However, in this thesis, the objective has been to identify the exact combination of events behind a failure. In order to achieve this, different and distinct (non-overlapping) ranges for numeric feature values have been defined whenever it is necessary to do so. Upon selection of the numeric feature value for a particular attribute, the proposed algorithm considers only that subset of examples for which the current value of the feature and existing feature values (as mentioned in Section 1.5) are in the same range. It is also important to note that when the algorithm considers an attribute B (for the event $F(B) \in y$; where y is a pre-defined range for this attribute) after considering an attribute A (for an event $F(A) \in x$; where x is a pre-defined range for this attribute), it considers only those examples in the database which satisfy both the properties: ' $F(A) \in x$ ' and ' $F(B) \in y$ '. Finally, we are using an 'Unknown Event' leaf in the tree, which represents that a branch in the fault-tree could not be developed beyond that point due to lack of information (examples).

From a practical point of view, sometimes even it may be difficult to specify distinct and non-overlapping ranges for a particular feature. Consequently, there may be a need for specifying overlapping ranges. In that case, a feature value may belong to more than one class (range). The proposed algorithm fulfills such requirement by putting an *OR* gate in the tree whenever a feature value belongs to more than one class and placing all the classes at the input of that *OR* gate and finally, executing the remaining part of the algorithm for each *OR* gate input, i.e., for each range of the feature. It is easy to see that

the more the number of such *OR* gate in the fault tree, the more generic will be the resulting tree. Interpretation and usefulness of the resulting fault-trees will be discussed in Section 4.4 of Chapter 4.

3.4 Summary

In this chapter different fault-tree synthesis (FTS) and analysis (FTA) methodologies have been reviewed. Also, at the end of this chapter the fault-tree synthesis algorithm developed in this thesis has been presented. In the next chapter, different failure scenarios in the Attitude Control Subsystem (ACS) (as discussed in Chapter 2) will be presented and fault-trees will be constructed under each failure scenario using the fault-tree synthesis algorithm that has been proposed in Section 3.3 of this chapter.

Chapter 4

Fault-Tree Synthesis from Simulated ACS Model Data

4.1 Introduction

In this chapter, fault-tree synthesis from the data obtained by the simulation of ACS model (as discussed in Chapter 2) is presented. As discussed in Section 3.3, the proposed algorithm for fault-tree synthesis constructs fault-trees from a set of example vectors, which can be classified based on different ranges of the feature values of the attributes/signals. In addition, each example vector is required to be tagged with a T (true) / F (false) flag that indicate whether the example is associated with a non-faulty or faulty system behavior. Therefore, for fault-tree construction from the data – which may be simulated or actual – the first step is to determine the ranges of feature values for each attribute under consideration under different type of faults. Different failure scenarios, for which fault-trees have been constructed, are presented in Section 4.2.

As discussed in Section 3.2, for an attribute/ signal A_i , if the extracted feature value is $F(A_i) = F_i$, where F is the feature extraction function, then a range (f_1, f_2) has to be defined for F_i when there is fault present in the system as well as when the system is fault-free. As a result, for different faults in the system, F_i will have different ranges. These defined ranges may be overlapped in some cases. Moreover, feature extraction functions may be different for different attributes. From now onwards, ' $F(A_i)$ ' will be used to indicate the

extracted feature value of the attribute A_i . Also, T and F will be used to indicate whether or not a particular example is associated with any failure.

Fault-trees have been constructed based on 5 (five) attributes of the ACS. For each attribute, simple feature extraction functions have been used. This demonstrates that the input generation process for the proposed algorithm may be very simple. While using a complex feature extraction may be efficient in many case, a big advantage of using simple feature extraction functions, wherever possible, is that the resulting tree would be able to convey information on the failure in terms of easily understandable functions. This is particularly important when there is an option of diagnosing the autonomous spacecraft by an operator at ground station. Table 4.1 shows the feature extraction functions that have been applied on attributes under consideration over a pre-defined time window (as discussed in Section 1.5) for each attribute under consideration.

Table 4.1: Selected Attributes for Fault-tree Synthesis and Feature Extraction Functions

Attribute	Feature Extraction Function (abs = absolute value)
Reaction Wheel motor current (I_m)	$F(I_m) = \max(\text{abs}(I_m))$
Reaction Wheel motor torque (T_m)	$F(T_m) = \max(\text{abs}(T_m))$
Reaction Wheel speed (ω_w)	$F(\omega_w) = \max(\text{abs}(\omega_w))$
Bus Voltage input to the Reaction Wheel (V_b)	$F(V_b) = \min(V_b)$
Torque Command input voltage to the Reaction Wheel (V_c)	$F(V_c) = \max(\text{abs}(V_c))$

Determination of Ranges under Each Failure Scenario:

For each feature, feature range (f_1, f_2) discussed above can be determined by the following factors:

1. Minimum feature value of the attribute required for a particular fault to take place.
2. Maximum or worst case possible feature value of the signal/attribute.

One of the f_1 and f_2 can be determined by injecting fault into the attribute/signal and by finding out the minimum amount of fault injection that may lead to a failure by simulation; finally, by applying the feature extraction function to find out f_1 or f_2 for that minimum amount of fault injection. Once f_1 (or f_2) is determined, the other bound of the range can be determined by assuming the worst case failure scenario i.e., by injecting maximum-probable fault (maximum possible anomaly) in the signal and applying the feature extraction function for that worst fault injection into the signal.

4.2 ACS Failure Scenarios

Since this thesis has been based on simulated data, it has been necessary to assume some failure scenarios utilizing the concept of fault injection into the system. Four failure scenarios for which fault-trees have been constructed are presented in the subsequent sections. It should be noted here that while using the actual system for analysis, the results obtained by assuming such scenarios may be utilized (as discussed in Section 1.5). In the developed ACS model, it has been assumed that a maximum attitude error of 0.03° in the pitch angle can be tolerated (as discussed in Section 2.2). Consequently, for all of the ACS failure scenarios, the top events in the fault-trees have been identified as ‘Pitch Error $> 0.03^\circ$ ’.

Out of the four ACS failure scenarios that are presented in the subsequent discussion, three of them correspond to a initial condition, when the reaction wheel (RW) runs near

zero speed whereas the rest corresponds to a different initial condition when the RW runs near maximum allowable speed.

4.2.1 ACS Failure Scenario-1

Random increase in reaction wheel motor current

This type of fault may take place because of some hardware level failure in the motor driver unit (MDU) in the reaction wheel. This fault has been injected when the RW was running near zero speed. The purpose of this fault is to represent failure under a surge in current. System behavior (pitch angle error) during fault-free condition as well as under the presence of this fault can be observed in Figure 4.1. Fault has been injected between $t=2500$ and $t=3500$ seconds. System behaved normally outside this time range. The location where this fault-injection has been injected was presented in Figure 2.7 of Chapter 2.

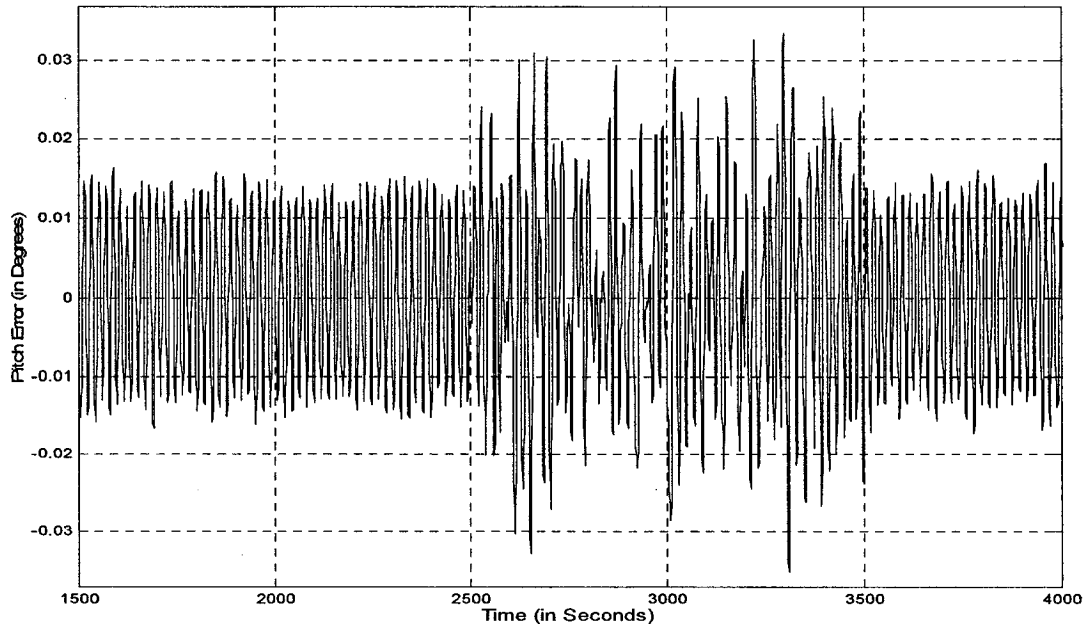


Figure 4.1: Pitch Error Vs Time under Failure Scenario-1

Example-1 in Table 4.2 represents the system condition under fault free condition. Examples 4 to 10 show the example vectors when the fault is injected into the system. These examples have both T and F flags associated with them. T indicates that the injected fault led the top event, i.e., 'Pitch Error > 0.03°'

From Example 1 we see that under fault-free condition, the feature value of motor current varies within an approx. range of +/-0.25 Ampere. For determining the ranges for feature values for this case, it was observed that ACS met the attitude requirement with a random error of 0.18 Ampere (around 72% of the normal peak value) or below present in the system. Example 10 shows the example vector when random error of +/-0.18 A was injected at the MDU output.

Table 4.2: Example Vectors and Ranges for Feature Values under Failure Scenario-1

Example Number	FLAG	Feature Values For:				
		Motor Current (I_m) Ampere	Motor Torque (T_m) N-m	Bus Voltage (V_b) Volts	RW Speed (ω_w) RPM	Torq. Comd. Voltage (V_c) Volts
1	F	0.2532	0.0073	28	21.04	1.2104
2	F	0.2187	0.0063	21	19.85	1.1510
3	F	0.2187	0.0063	19.5	19.84	1.1510
4	T	0.8532	0.0247	28	47.49	3.0363
5	T	0.7541	0.0219	28	41.37	2.6817
6	T	0.7212	0.0209	28	44.73	2.6404
7	T	0.6447	0.0187	28	38.52	2.3971
8	T	0.7227	0.0210	28	39.04	2.4471
9	T	0.5974	0.0173	28	33.63	2.2878
10	F	0.5492	0.0159	28	31.99	2.0792
Approx. Ranges:		0.55 – 0.86	0.0160 – 0.0249	21 – 28	31 – 48	2.08 – 3.04

In order to determine the upper limit, we need to assume a worst-case scenario. One can also use past experience/ fault data in this case. It has been assumed that the surge would lead to maximum 0.375 ampere (150% of the normal peak value) random error in the

MDU output. Example 2 shows the vector corresponding to this situation. Based on the above information, the approximate ranges for the feature values of the different attributes for this failure scenario are presented in the last row of Table 4.2. Ranges for V_b are discussed in Section 4.2.3.

4.2.2 ACS Failure Scenario-2

Increase in Friction in the reaction wheel

This fault has been injected when the RW was running near zero speed. The purpose of this fault is to represent failure if the friction is increased in the wheel bearings because of wear of bearing material over time or some problem in the lubricant flow. System behavior (pitch angle error) during fault-free condition as well as under the presence of this fault can be observed in Figure 4.2.

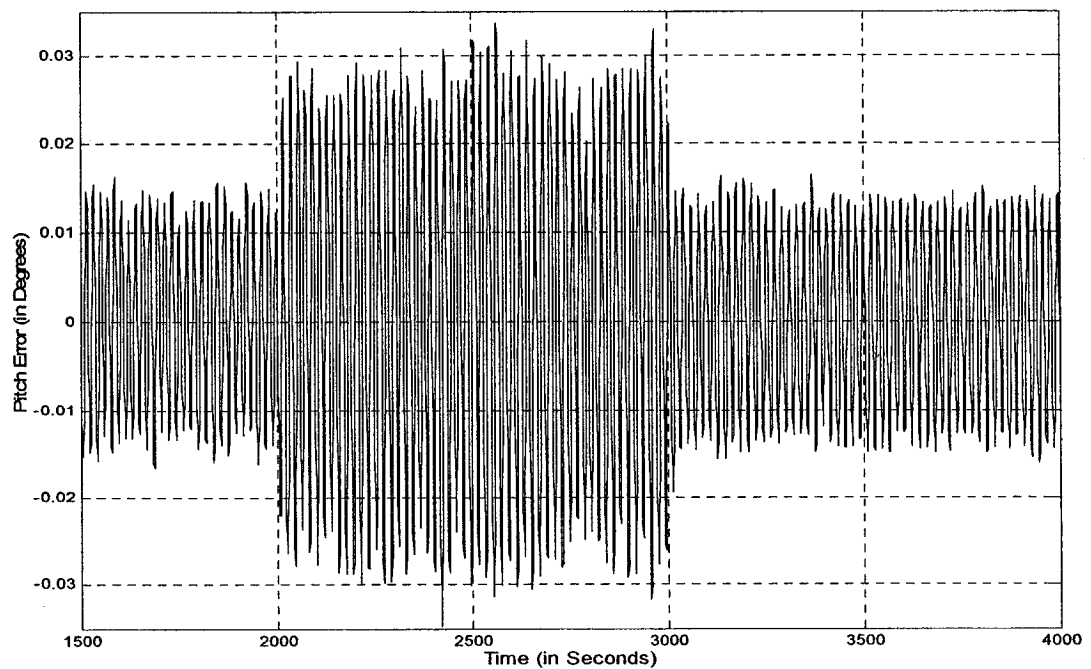


Figure 4.2: Pitch Error Vs Time under Failure Scenario-2

Fault has been injected between $t=2000$ and $t=3000$ seconds. System behaved normally outside this time range. The location where this fault-injection has been injected was presented in Figure 2.10 of Chapter 2.

Example 1 (which is same as the Example 1 in Table 4.2) in Table 4.3 represents the system condition under fault free condition. Examples 11 to 18 show the example vectors when the fault is injected into the system. These examples have both T and F flags associated with them. T indicates that the injected fault led the top event, i.e., ‘Pitch Error $> 0.03^\circ$ ’.

Table 4.3: Example Vectors and Ranges for Feature Values under Failure Scenario-2

Example Number	FLAG	Feature Values For:				
		Motor Current (I_m) Ampere	Motor Torque (T_m) N-m	Bus Voltage (V_b) Volts	RW Speed (ω_w) RPM	Torq. Comd. Voltage (V_c) Volts
1	F	0.2532	0.0073	28	21.04	1.2104
11	F	0.5469	0.0159	28	48.57	2.8784
12	F	0.4679	0.0136	28	42.04	2.4629
13	T	0.4907	0.0142	28	42.17	2.3240
14	T	0.5003	0.0145	28	40.00	2.3189
15	T	0.4218	0.0122	28	36.89	2.2198
16	F	0.3783	0.0110	28	34.41	1.9909
17	F	0.3957	0.0115	28	34.40	2.0824
18	F	0.3699	0.0107	28	32.67	1.9468
Approx. Ranges:		0.4 – 0.549	0.0121 – 0.0159	21 – 28	35 – 49	2.0 – 2.9

From the ACS model it is known that that the coulomb friction in the wheel is 2 mN-m and near zero wheel speed, coulomb plus viscous friction is approximately equal to the coulomb friction. Also, from simulation, it was observed that the ACS met the attitude requirement (Pitch Error $< 0.03^\circ$) with an increase of 1.5 mN-m (approx. 75% of nominal

value) or below in friction. Example 16 shows the example vector when the friction was increased by 1.5mN-m.

Now, in order to determine the other the other limit, it is necessary to assume a worst-case scenario. We can safely assume 3mN-m (150% of the nominal value) increase as the worst case because bearing wear or other similar faults are usually developed gradually over the time and it is very likely that the fault due to this type of anomaly will be detected by the fault detection scheme before the friction reaches an extreme value. Example 11 shows the vector corresponding to this assumed worst-possible situation. Based on the above information, the approximate ranges for the feature values of the different attributes for this failure scenario are presented in the last row of Table 4.3.

4.2.3 ACS Failure Scenario-3

Bus Voltage Failure at High Speed

This fault has been injected when the RW was running near maximum allowable speed. This type of fault may take place at low bus conditions when large back-EMF, developed in the reaction wheel motor operating at a high speed, limits the motor current, consequently the motor torque. System behavior (pitch angle error) during fault-free condition as well as under the presence of this fault can be observed in Figure 4.3. Fault has been injected between $t=2000$ and $t=3500$ seconds. System behaved normally outside this time range. The location where this fault-injection has been injected was shown in Figure 2.6 of Chapter 2.

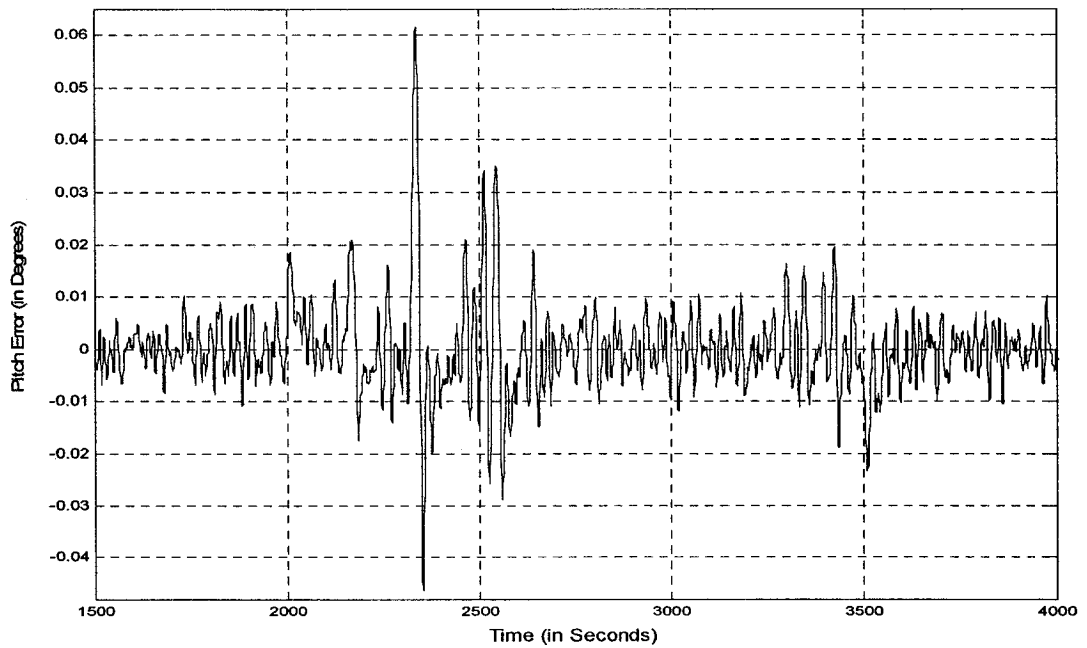


Figure 4.3: Pitch Error Vs Time under Failure Scenario-3

Example-19 in Table 4.4 represents the system condition under fault free condition when the RW runs near the maximum allowable speed. Examples 22 to 29 show the example vectors when the fault is injected into the system; i.e., bus voltage is lowered below the minimum limit of 21 volts. These examples have both T and F flags associated with them. T indicates that the injected fault led the top event, i.e., ‘Pitch Error > 0.03°’.

It should be clear at this stage that failure due to this type of fault may take place only at high operational speed of the reaction wheel. At low or near zero speed, even if the bus voltage drops to a value as low as 10 volts, the torque may not be limited because of small back-EMF developed in the motor. And it is very unlikely that bus voltage will ever reach such low value. As a result, in this case, we should only consider scenarios at high speed. It has been assumed that the bus voltage level is of 21 –28 volts under normal

system conditions (as it is in EO-1 satellite). Consequently, it has also been assumed that the maximum allowable reaction wheel speed is 5095 RPM which is very close to 5100 RPM (as in the case of the EO-1 satellite).

Table 4.4: Example Vectors and Ranges for Feature Values under Failure Scenario-3

Example Number	FLAG	Feature Values For:				
		Motor Current (I_m) Ampere	Motor Torque (T_m) N-m	Bus Voltage (V_b) Volts	RW Speed (ω_w) RPM	Torq. Comd. Voltage (V_c) Volts
19	F	0.8931	0.0259	28.00	5095.2	4.70
20	F	0.9257	0.0268	21.50	5100.2	4.87
21	F	0.9002	0.0261	21.00	5097.4	4.96
22	F	0.8632	0.0250	20.50	5096.7	5.00
23	F	0.8275	0.0240	20.10	5094.7	5.00
24	F	0.7919	0.0230	19.70	5097.3	5.00
25	T	0.7761	0.0225	19.50	5106.9	5.00
26	F	0.7321	0.0212	19.50	4077.6	3.8442
27	F	0.8981	0.0260	21.00	5137.4	4.9959
28	T	0.7708	0.0224	19.40	5103.5	5.00
29	T	0.7559	0.0219	19.20	5119.5	5.00
Approx. Ranges:		0.75 – 0.78	0.0217 – 0.0226	19.0 - 20.99	5100 onwards	4.99 onwards

From ACS simulation, it is observed that at around 5095 RPM of peak speed, minimum bus voltage drop to 19.5 volts is to be there in order for any failure in attitude to take place. Example 25 shows the example vector corresponding to this situation. In order to determine the other limit, a worst-case scenario has to be assumed as it was done for scenario-1 and scenario-2. One can also use past experience/ fault data in this case. However, from ACS simulation it has been found that when the bus voltage drops to 19.1 volt or below, the ACS control loop becomes totally unstable. Based on the above information, the approximate ranges for the feature values of the different attributes for this failure scenario are presented in the last row of Table 4.4. Figure 4.4 shows the detail response of the four system parameters or attributes under this failure scenario. It can be

observed from the Figure 4.4 that the system behaved normally when the fault was removed at $t=3500$ seconds.

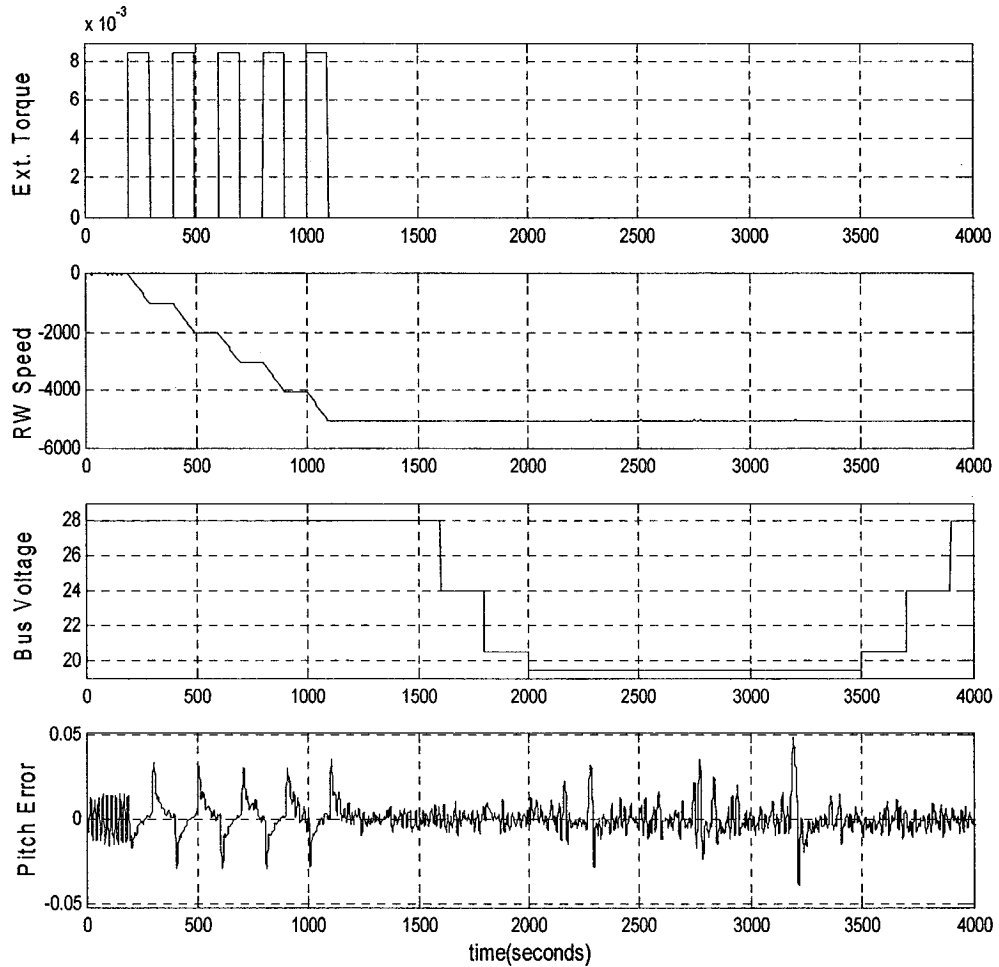


Figure 4.4: Different ACS Parameters under Failure Scenario-3

(Top to bottom) (a) External Torque, shown in $N\text{-}m$, which was applied on the satellite body to make the RW run near maximum allowable speed (b) Increase in RW Speed, shown in RPM , when the external torque was applied on the satellite body (c) Bus voltage, shown in $Volts$, was reduced gradually after $t=1500$ sec. when the ACS became stable after overcoming the effect of external torque applied on the satellite body (d) Pitch Error Response – shown in $Degrees$

4.2.4 ACS Failure Scenario-4

Small Error in Motor Current together with Increase in Wheel-bearing Friction

In this case, we show that while a small error in motor driver unit (MDU) output or small increase in friction does not lead to any failure individually, when both of these take place together, ACS may fail to maintain required attitude. This scenario has been created under the initial condition when the RW was running near zero speed. System behavior (pitch angle error) during fault-free condition as well as under the presence of this fault can be observed in Figure 4.5. The fault related to the motor current has been injected between $t=2000$ and $t=3000$ seconds and the fault related to the friction has been injected between $t=2500$ and $t=3500$ seconds. The locations where these faults have been injected were shown in Figures 2.7 and 2.10 of Chapter 2. It can be observed from the figure that ‘Pitch Error $> 0.03^\circ$ ’ when both the faults take place together between $t=2500$ and 3000 seconds. ACS meets the attitude requirement outside this time range.

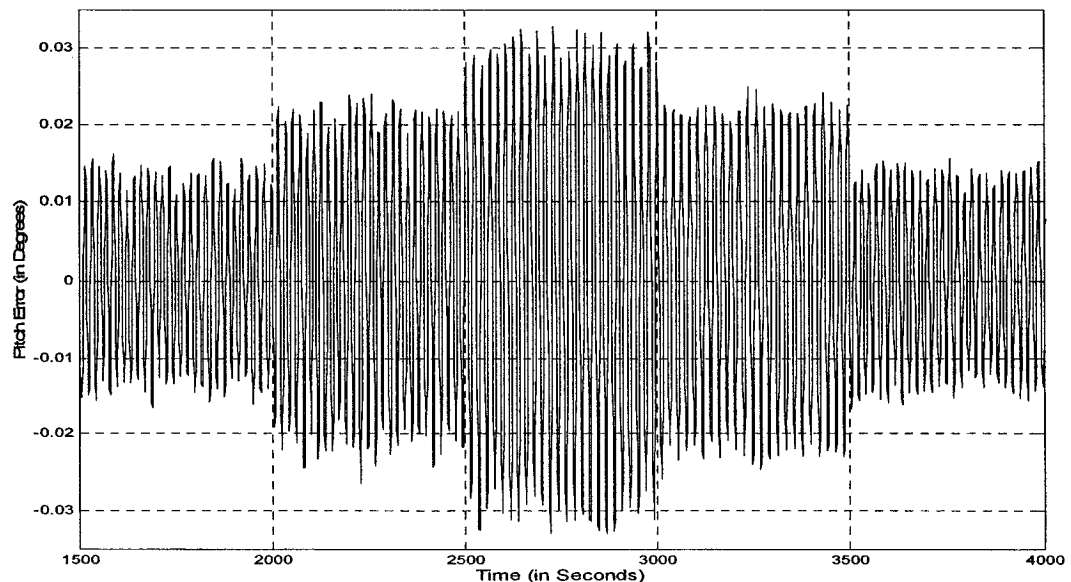


Figure 4.5: Pitch Error Vs Time under Failure Scenario-4

Example-1 in Table 4.5 (which is same as the Example 1 in Table 4.2 and Table 4.3) represents the system condition under fault free condition. Examples 30 to 39 show the example vectors when the fault is injected into the system. These examples have both T and F flags associated with them. T indicates that the injected fault led the top event, i.e., ‘Pitch Error > 0.03°’

Example 33 shows the case when an error of only 0.01Ampere (approx. 4% of the nominal peak value) is introduced at the MDU output with 1.5mN-m increase in friction. Example 37 shows the case when only 0.1mN-m increase in friction (approx. 5% of the nominal peak value) is there with 0.09 Ampere error in MDU output.

Table 4.5: Example Vectors and Ranges for Feature Values under Failure Scenario-4

Example Number	FLAG	Feature Values For:				
		Motor Current (I_m) Ampere	Motor Torque (T_m) N-m	Bus Voltage (V_b) Volts	RW Speed (ω_w) RPM	Torq. Comd. Voltage (V_c) Volts
1	F	0.2532	0.0073	28	21.04	1.2104
30	T	0.4489	0.0130	28	44.70	2.8363
31	T	0.4618	0.0134	28	44.05	2.7992
32	T	0.4350	0.0126	28	41.01	2.5528
33	T	0.4066	0.0118	28	36.64	2.1928
34	T	0.3735	0.0108	28	34.79	2.2290
35	T	0.3454	0.0100	28	34.26	2.2915
36	T	0.3276	0.0095	28	32.09	2.1976
37	T	0.3149	0.0091	28	31.82	2.1311
38	F	0.3120	0.0090	28	30.83	2.1159
39	F	0.3089	0.0090	28	30.62	2.0468
Approx. Ranges:		0.313 - 0.470	0.0090 - 0.0136	21 - 28	31 - 45	2.1 - 2.9

Examples 30 and 31 show some worst-possible results for this failure scenario – when a fixed error of 0.09 Ampere is introduced together with a 1.5mN-m increase in friction.

From the ACS model we know that the coulomb friction in the wheel is 2 mN-m and near zero wheel speed, coulomb plus viscous friction will be approximately equal to the coulomb friction. From ACS simulation, it has been observed that an increase in friction of 1.5 mN-m (approx. 75% of nominal value) or below does not lead to any failure. It has also been observed from the simulation that an increase in error in motor current by 0.09 Ampere (approx. 35% of nominal peak value) or below does not lead to any failure. Example 38 shows the example vector when a fixed error of 0.09 Ampere has been introduced at MDU output. Based on the above information, the approximate ranges for the feature values of the different attributes for this failure scenario are presented in the last row of Table 4.5.

It should be noted here that for motor current, above 0.31 Ampere (example 37), there could be another failure scenario where fixed error in MDU output itself would lead to a failure. In that fault, we could define another range with lower limit as 0.32 Ampere and the upper limit would have been bounded by a worst-case scenario.

4.3 Fault-tree Synthesis under Different ACS-failure Scenarios

Once the failure scenarios are available, the fault-tree construction requires four main steps to be followed, which will be presented in the subsequent discussion:

1. Identification of different ranges of feature values under different failure scenarios.
2. Generation of input for fault-tree synthesis
3. Determination of attribute selection sequence.
4. Fault-tree construction using the fault-tree synthesis algorithm proposed in Chapter 3.

4.3.1 Identification of Different Ranges of Feature Values

The reason behind the determination of different ranges of feature values has been discussed in Section 3.3. Table 4.6 summarizes the ranges of different feature values of the attributes for the four failure scenarios under consideration which have been presented before in the last rows of tables 4.2, 4.3, 4.4 and 4.5.

Table 4.6: Summary of the Ranges of Numeric Feature Values

Attribute	Range under Scenario-4	Range under Scenario-2	Range under Scenario-1	Range under Scenario-3
I_m	0.313 – 0.470	0.4 – 0.549	0.55 – 0.86	0.75 – 0.78
T_m	0.0090 – 0.0136	0.0121 – 0.0159	0.0160 – 0.0249	0.0217 – 0.0226
ω_w	31 – 45	35 – 49	31 – 48	5100 onwards
V_b	21 – 28	21 – 28	21 – 28	19.0 – 20.99
V_c	2.1 – 2.9	2.0 – 2.9	2.08 – 3.04	4.99 onwards

It has already been mentioned in Section 4.2 that out of the four ACS failure scenarios that have been presented above, three of them correspond to an initial condition, when the reaction wheel (RW) runs near zero speed while the rest corresponds to a different initial condition when the RW runs near maximum allowable speed. As discussed in Section 2.2.2, at high speed, high back-EMF developed in the motor may limit the torque if the bus voltage becomes very low. Due to this fact, two ranges were assumed for the feature values of the attribute bus voltage (V_b) – one was the normal operational range ‘21- 28 volts’ (as discussed in Section 2.2) and the other range representing the low bus voltage condition discussed under the ‘ACS Failure Scenario-3’ in Section 4.2.3. It should be noted here that for some other failure scenarios, where the RW runs in an intermediate speed between *zero* and maximum allowable speed, the normal operational range of 21 – 28 volts may be divided into different ranges, if required.

Using the information in Table 4.6, it is easy to define different ranges of feature values under different failure scenarios as presented in Table 4.7. For the sake of clarity in presentation, these ranges have been assigned names. In subsequent sections, especially in fault-tree synthesis, these names will be used for quick and easy reference.

Table 4.7: Different Ranges of Feature Values under different failure scenarios

Attribute	Range for Feature Value	Given Name of the Range	Corresponding Failure Scenario
Motor Current (I_m)	0.55 – 0.86	A	Failure Scenarios 1 & 3
	0.4 – 0.549	B	Failure Scenario-2
	0.313 – 0.470	D	Failure Scenario-4
Motor Torque(T_m)	0.0160 – 0.0249	A	Failure Scenarios 1 & 3
	0.0121 – 0.0159	B	Failure Scenario-2
	0.0090 – 0.0136	D	Failure Scenario-4
Wheel Speed (ω_w)	30 – 50	A	Failure Scenarios 1, 2 & 4
	5100 onwards	B	Failure Scenario-3
Bus Voltage (V_b)	21 – 28	A	Failure Scenarios 1, 2 & 4
	19.0 – 20.99	B	Failure Scenario-3
Torque Command Voltage (V_c)	2.0 – 3.5	A	Failure Scenarios 1, 2 & 4
	4.99 onwards	B	Failure Scenario-3

It should be noted here that ranges B and D are overlapping both for motor current (I_m) and motor torque (T_m).

4.3.2 Generation of Input for Fault-tree Synthesis

It should be clear at this point that in the proposed approach, fault-trees are constructed from example vectors. We call the set of such examples as the input to the Fault-tree Synthesis (FTS) algorithm. As mentioned in Sections 1.5 and 3.3, when fault is detected in the system, *Current Example* vector (which is assigned a TRUE flag always) is generated and added with the existing example set to form an input to the FTS algorithm. Therefore, input to the algorithm is *Existing Example Set* plus 1 (one) *Current Example Vector*.

Table 4.8: Input ('Existing Example' Set) for the proposed FTS Algorithm

Example Number	FLAG	Motor Current (I_m) Ampere		Motor Torque (T_m) N-m		Bus Voltage (V_b) Volts		RW Speed (ω_w) RPM		Torq. Comd. Voltage (V_c) Volts	
		Range(s) and Feature Value		Range(s) and Feature Value		Range(s) and Feature Value		Range(s) and Feature Value		Range(s) and Feature Value	
1	F	*	0.2532	*	0.0073	A	28	*	21.04	*	1.2104
2	F	*	0.2187	*	0.0063	A	21	*	19.85	*	1.1510
3	F	*	0.2187	*	0.0063	B	19.5	*	19.84	*	1.1510
4	T	A	0.8532	A	0.0247	A	28	A	47.49	A	3.0363
5	T	A	0.7541	A	0.0219	A	28	A	41.37	A	2.6817
6	T	A	0.7212	A	0.0209	A	28	A	44.73	A	2.6404
7	T	A	0.6447	A	0.0187	A	28	A	38.52	A	2.3971
8	T	A	0.7227	A	0.0210	A	28	A	39.04	A	2.4471
9	T	A	0.5974	A	0.0173	A	28	A	33.63	A	2.2878
10	F	*	0.5492	B	0.0159	A	28	A	31.99	A	2.0792
11	T	B	0.5469	B	0.0159	A	28	A	48.57	A	2.8784
12	T	B, D	0.4679	D, B	0.0136	A	28	A	42.04	A	2.4629
13	T	B	0.4907	B	0.0142	A	28	A	42.17	A	2.3240
14	T	B	0.5003	B	0.0145	A	28	A	40.00	A	2.3189
15	T	B, D	0.4218	D, B	0.0122	A	28	A	36.89	A	2.2198
16	F	D	0.3783	D	0.0110	A	28	A	34.41	*	1.9909
17	F	B	0.3957	D	0.0115	A	28	A	34.40	A	2.0824
18	F	D	0.3699	D, B	0.0107	A	28	A	32.67	*	1.9468
19	F	*	0.8931	*	0.0259	A	28.00	*	5095.2	*	4.70
20	F	*	0.9257	*	0.0268	A	21.50	B	5100.2	*	4.87
21	F	*	0.9002	*	0.0261	A	21.00	*	5097.4	*	4.96
22	F	*	0.8632	*	0.0250	B	20.50	*	5096.7	B	5.00
23	F	A	0.8275	A	0.0240	B	20.10	*	5094.7	B	5.00
24	F	A	0.7919	A	0.0230	B	19.70	*	5097.3	B	5.00
25	T	A	0.7761	A	0.0225	B	19.50	B	5106.9	B	5.00
24	F	A	0.7321	A	0.0212	B	19.50	*	4077.6	*	3.8442
27	F	*	0.8981	*	0.0260	A	21.00	B	5137.4	B	4.9959
28	T	A	0.7708	A	0.0224	B	19.40	B	5103.5	B	5.00
29	T	A	0.7559	A	0.0219	B	19.20	B	5119.5	B	5.00
30	T	D	0.4489	D	0.0130	A	28	A	44.70	A	2.8363
31	T	D	0.4618	D	0.0134	A	28	A	44.05	A	2.7992
32	T	D	0.4350	D	0.0126	A	28	A	41.01	A	2.5528
33	T	D	0.4066	D	0.0118	A	28	A	36.64	A	2.1928
34	T	D	0.3735	D	0.0108	A	28	A	34.79	A	2.2290
35	T	D	0.3454	D	0.0100	A	28	A	34.26	A	2.2915
36	T	D	0.3276	D	0.0095	A	28	A	32.09	A	2.1976
37	T	D	0.3149	D	0.0091	A	28	A	31.82	A	2.1311
38	F	*	0.3120	D	0.0090	A	28	A	30.83	A	2.1159
39	F	*	0.3089	D	0.0090	A	28	A	30.62	A	2.0468

The *Existing Example Set* can be created by merging the resulted vectors under each failure scenario. Table 4.8 shows the example set that has been created by merging all the

example vectors of tables 4.2, 4.3, 4.4 and 4.5. However, while generating the input i.e., the example set, it must be ensured that the example set contains sufficient number of example vectors to represent the failure scenarios clearly. An asterisk mark in the range column indicates that the feature value does not belong to a pre-defined range in that example. However, if more numbers of failure scenarios are considered, they may fall into some other pre-defined range(s).

4.3.3 Determination of Attribute Selection Sequence

As mentioned in Section 3.2.3, the size of the resulting tree may vary as the attribute selection sequence varies which is due to the fact that some attributes divide or classify data more clearly than the others. Different approaches for the determination of an appropriate attribute selection sequence were also discussed in Section 3.2.3. In this section, determination of the attribute selection sequence will be demonstrated using the concept of 'Entropy'.

When a fault is detected in the system, according to the proposed framework, features is extracted from the attributes over a pre-defined time window (as discussed in Section 1.5) and an example vector is formed which is used in fault-tree synthesis. For fault-tree synthesis using the proposed algorithm, attributes has to be selected sequentially. In order to find an appropriate sequence, entropies for each attribute have been calculated under each failure scenario over the same time frame mentioned above. These calculations have been based on the entropy equation presented in Section 3.2.3 with normalized data. The concept is quite simple – entropy is calculated for each attribute

under non-faulty conditions as well as in presence of faults in the system. The more is the percentage change in entropy between faulty and non-faulty cases, the important the attribute is considered to be. Consequently, attributes are ranked according to the % change in their entropy. Since the entropies under non-faulty system condition are taken as ‘references’ and the % changes are calculated with respect to these ‘references’, while determining entropies under non-faulty conditions, it must be ensured that the system is in same initial condition.

Table 4.9: Ranking Assigned to the Attributes Based on % Change in Entropy

Attribute	Entropy at Non-faulty Condition (E_{NF})	Failure Scenario-1			Failure Scenario-2			Failure Scenario-4		
		Absolute Entropy (E_1)	% ΔE	RANK	Absolute Entropy (E_2)	% ΔE	RANK	Absolute Entropy (E_4)	% ΔE	RANK
I_m	-2.5186e+006	-7.0611e+006	180.3588	1	-7.7292e+006	206.8863	1	-5.7999e+006	130.2835	2
T_m	-2.5186e+006	-7.0611e+006	180.3588	2	-7.7292e+006	206.8863	2	-5.7999e+006	130.2835	3
V_b	-4.6230e+007	-4.6230e+007	6.82e-004	5	-4.6230e+007	6.82e-004	5	-4.6230e+007	6.82e-004	5
ω_w	291.2832	332.9672	14.3105	4	412.1233	41.4854	4	432.9105	48.6218	4
V_c	-2.5186e+006	-4.3393e+006	72.2913	3	-7.7292e+006	206.8863	3	-7.8539e+006	211.8353	1

Attribute	Entropy at Non-faulty Condition (E_{NF})	Failure Scenario-3		
		Absolute Entropy (E_3)	% ΔE	RANK
I_m	-9.1500e+007	-9.1549e+007	0.0538	3
T_m	-9.1500e+007	-9.1549e+007	0.0538	4
V_b	-4.6230e+007	-8.7750e+007	89.8129	1
ω_w	-1.7474e+008	-1.7474e+008	5.4808e-005	5
V_c	-9.1500e+007	-1.4069e+008	53.7578	2

It should be clear at this point that out of the four failure scenarios presented in this chapter, ‘Failure Scenario-3’ has a different initial condition compared to that of other three failure scenarios. Table 4.9 shows the ranks of the attributes that have been assigned based on the percentage change in their entropy when a fault took place.






It should be noted here that the developed ACS is simple compared to the actual model and the relation among I_m , T_m and V_c is linear. As a result, the changes in entropy also come out to be same for these three attributes when simulated data from the model is used which will not be the case for actual ACS or more complex ACS model. In order to break the ties, preference has been given to these attributes in the order: I_m , T_m and V_c .

In summary, under different failure scenarios, the attribute selection sequences were found out to be as follows:

<u>ACS Failure Scenario</u>	<u>Attribute Selection Sequence</u>
Scenario-1	$I_m \rightarrow T_m \rightarrow V_c \rightarrow \omega_w \rightarrow V_b$
Scenario-2	$I_m \rightarrow T_m \rightarrow V_c \rightarrow \omega_w \rightarrow V_b$
Scenario-3	$V_b \rightarrow V_c \rightarrow I_m \rightarrow T_m \rightarrow \omega_w$
Scenario-4	$V_c \rightarrow I_m \rightarrow T_m \rightarrow \omega_w \rightarrow V_b$

4.3.4 Fault-tree Construction

Fault-trees that have been constructed for different failure scenarios presented in Section 4.2. Following types of nodes have been used for tree synthesis, which were discussed in Section 3.1:

<u>Node Symbol</u>	<u>Description</u>
	<i>Top Event</i> node. The top event is ‘Pitch Error > 0.03°’ in all failure scenarios under consideration.
	<i>Basic Event</i> node
	<i>AND</i> gate
	<i>OR</i> gate
	<i>Unknown Event</i> node

It should be mentioned here that our intention here is to construct a tree when the attribute feature values are within these pre-defined ranges to demonstrate the feasibility of the proposed fault tree synthesis algorithm. While constructing fault trees, if the feature value of any attribute is found outside these pre-defined ranges and/or, if insufficient examples are given to the algorithm from which the algorithm is unable to reach a conclusion, it places an ‘unknown event’ node in the tree saying that a branch in the fault-tree could not be developed further because of the lack of example.

As mentioned in Sections 1.5 and 3.3, when fault is detected in the system, *Current Example* vector (which is assigned a TRUE flag always) is generated and added with the *existing example set* (as presented in Table 4.8) to form an input to the FTS algorithm.

Different types of faults have been injected in the ACS and example vectors ('Current Example') have been generated. In the subsequent discussion, fault-trees will be constructed by adding different *current example* vectors with the *existing example set* given in Table 4.8. These *current example* vectors correspond to different ACS failure scenarios mentioned in Section 4.2.

Fault-tree for Failure Scenario-1:

A fault similar to the one under failure scenario-1 was injected into the system. *Current example* vector generated by fault injection and feature extraction is as follows:

FLAG	Feature Value for:				
	I_m	T_m	V_b	ω_w	V_c
T	0.7013 (A)	0.0203 (A)	28 (A)	44.17 (A)	2.3742 (A)

The pre-defined ranges which the feature values belong to are shown in the parenthesis below each feature value in the generated vector. When this vector is added with the example sets in Table 4.8 to generate input for fault-tree construction and proposed fault-tree synthesis algorithm is applied on this input by following the attribute selection sequence mentioned in Section 4.3.3, i.e., $I_m \rightarrow T_m \rightarrow V_c \rightarrow \omega_w \rightarrow V_b$, the resulting fault tree is the same as the one in Figure 4.6

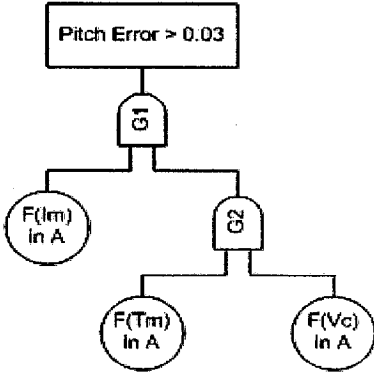


Figure 4.6: Fault-tree for Failure Scenario-1

It is clear from the fault-tree in Figure 4.6 that the tree points towards motor current I_m by a node in the tree. From the ACS-model point of view, in presence of fault, the anomalous behavior of reaction wheel motor current is directly translated into motor torque. Since there is closed-loop control in the ACS, the fault is propagated through the loop and the torque command voltage V_c is also affected. Consequently, in addition to the motor current I_m , motor torque T_m and torque command voltage V_c appears on the resulting fault-tree as the sources of anomaly in the system.

Fault-tree for Failure Scenario-2:

Case-1: A feature value belong to one range

A fault similar to the one under failure scenario-2 was injected into the system. *Current example* vector generated by fault injection and feature extraction is as follows:

FLAG	Feature Value for:				
	I_m	T_m	V_b	ω_w	V_c
T	0.4927 (B)	0.0143 (B)	28 (A)	44.15 (A)	2.5933 (A)

The pre-defined ranges which the feature values belong to are shown in the parenthesis below each feature value in the generated vector. When this vector is added with the example sets in Table 4.8 to generate input for fault-tree construction and proposed fault-tree synthesis algorithm is applied on this input by following the attribute selection sequence mentioned in Section 4.3.3, i.e., $I_m \rightarrow T_m \rightarrow V_c \rightarrow \omega_w \rightarrow V_b$, the resulting fault tree is the same as the one in Figure 4.7

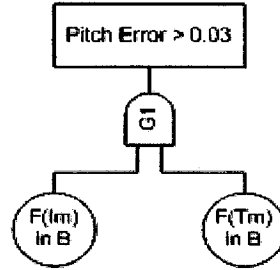


Figure 4.7: Fault-tree for Failure Scenario-2 (Non-overlapping ranges of feature values)

It is clear from the fault-tree in Figure 4.7 that the tree points towards motor torque T_m , which is most effected by the increase in friction in the system, by a node in the tree. From the ACS-model point of view, in order to overcome the increase in friction due to the presence of fault in the system, the reaction wheel motor draws more current which is necessary to maintain satellite's desired orientation. As a result the behavior of motor current is also altered. Consequently, the motor current I_m , also appear on the resulting fault-tree as the source of anomaly in the system.

Case-2: A feature value belong to more than one range

A fault similar to the one under failure scenario-2 was injected into the system. *Current example* vector generated by fault injection and feature extraction is as follows:

FLAG	Feature Value for:				
	I_m	T_m	V_b	ω_w	V_c
T	0.4394 (B, D)	0.0127 (D, B)	28 (A)	38.30 (A)	2.3127 (A)

The pre-defined ranges which the feature values belong to are shown in the parenthesis below each feature value in the generated vector. When this vector is added with the example sets in Table 4.8 to generate input for fault-tree construction and proposed fault-tree synthesis algorithm is applied on this input by following the attribute selection

sequence mentioned in Section 4.3.3, i.e., $I_m \rightarrow T_m \rightarrow V_c \rightarrow \omega_w \rightarrow V_b$, the resulting fault tree is the same as the one in Figure 4.8

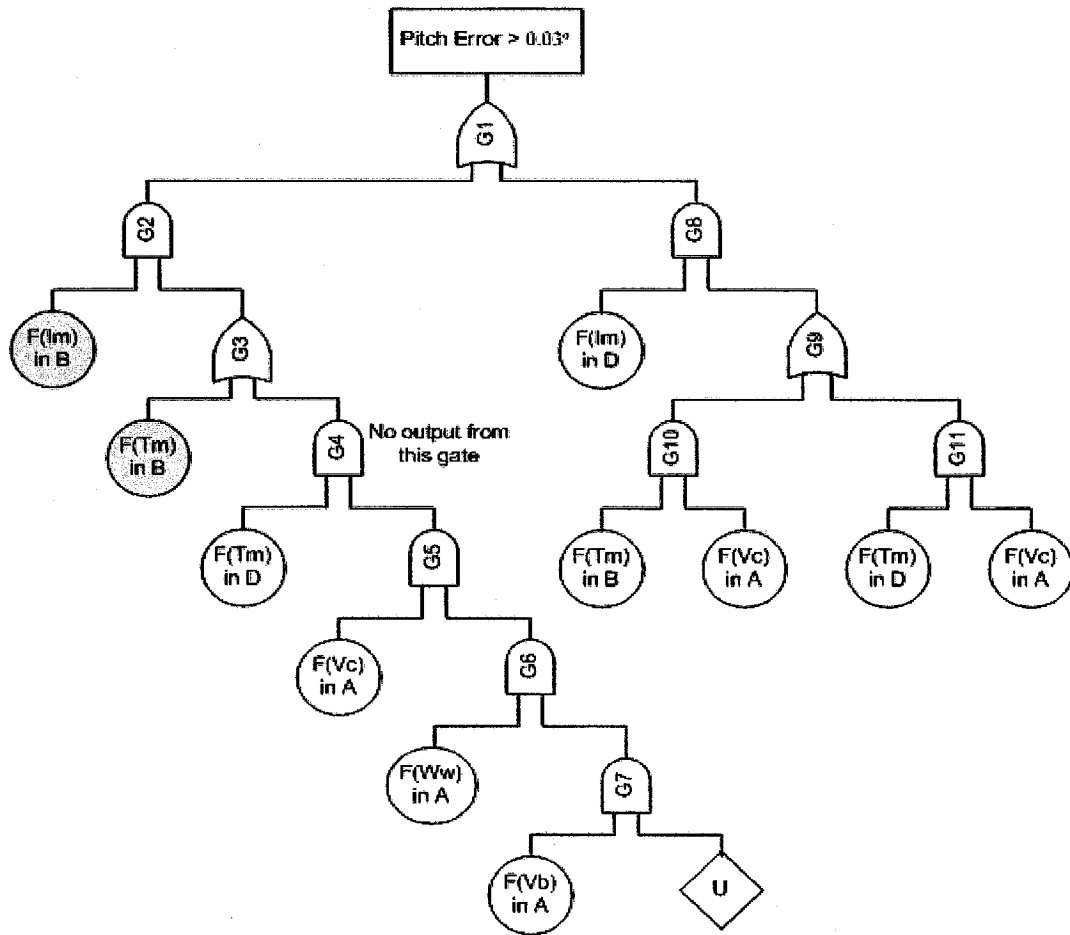


Figure 4.8: Fault-tree for Failure Scenario-2 (Overlapping ranges of feature values)

It is clear from the fault-tree in Figure 4.8 that the tree represents four possible combinations of events behind the top event. If the *minimal cut-sets* (as discussed in Section 3.1.2) of this tree are determined, it is found that one of the *minimal cut-set* formed using the two shaded nodes of the tree ($I_m \in B$ and $T_m \in B$) represents events which are exactly same as the ones represented by the fault-tree in Figure 4.7. Out of the remaining three branches one branch (in which $I_m \in B$, $T_m \in D$) does not converge and

terminates by putting an *unknown event* node (as discussed in Sections 3.3). The next branch (in which $I_m \in D$, $T_m \in D$) represents the failure scenario-4. The last branch (in which $I_m \in D$, $T_m \in B$) gives a new combination of event, i.e., a combination which was not foreseen and can lead to the top event. This particular combination of events, which may cause the *top event* to take place, was inherent in the example set used for fault-tree synthesis but was unknown until the fault-tree had been constructed.

It is also clear from the fault-tree in Figure 4.8 that the tree points towards motor torque T_m , which is most effected by the increase in friction in the system, by a node in the tree. As discussed under ‘Case-1’, from the ACS-model point of view, in order to overcome the increased in friction due to the presence of fault in the system, the reaction wheel motor draws more current which is required to maintain satellite’s desired orientation. As a result the behavior of motor current is also altered. Consequently, the motor current I_m , also appear on the resulting fault-tree as the source of anomaly in the system. Moreover, as discussed under Failure Scenario-1, since there is closed-loop control in the ACS, the fault is propagated through the loop and the torque command voltage V_c is also affected. Consequently, in addition to the motor current I_m , motor torque T_m and torque command voltage V_c appears on three branches in the resulting fault-tree as the sources of anomaly in the system.

Fault-tree for Failure Scenario-3:

A fault similar to the one under failure scenario-3 was injected into the system. *Current example* vector generated by fault injection and feature extraction is as follows:

FLAG	Feature Value for:				
	I_m	T_m	V_b	ω_w	V_c
T	0.7730 (A)	0.0224 (A)	19.45 (B)	5101.3 (B)	5 (B)

The pre-defined ranges which the feature values belong to are shown in the parenthesis below each feature value in the generated vector. When this vector is added with the example sets in Table 4.8 to generate input for fault-tree construction and proposed fault-tree synthesis algorithm is applied on this input by following the attribute selection sequence mentioned in Section 4.3.3, i.e., $V_b \rightarrow V_c \rightarrow I_m \rightarrow T_m \rightarrow \omega_w$, the resulting fault tree is the same as the one in Figure 4.9

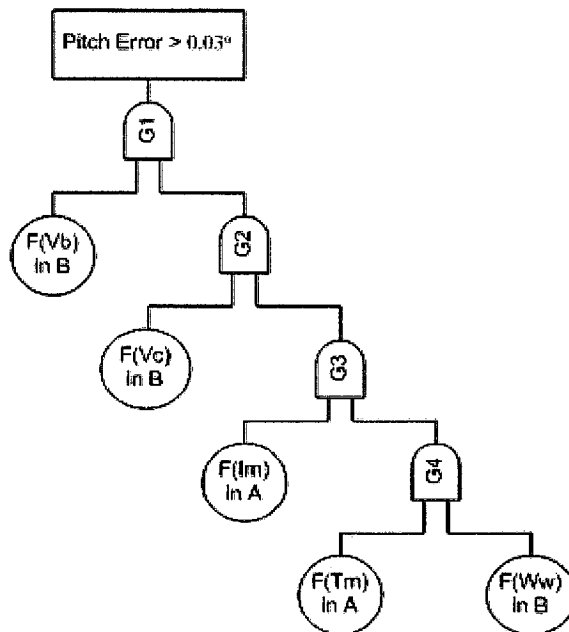


Figure 4.9: Fault-tree for Failure Scenario-3

It is clear from the fault-tree in Figure 4.9 that the tree points towards the bus voltage V_b , which is the cause of failure in this case, by a node in the tree. As mentioned earlier, this fault can take place only at high speed of the reaction wheel. Under such condition, the increasing back-EMF of the reaction wheel motor limits the motor current I_m . Consequently, in the closed-loop system, motor torque T_m and torque command voltage V_c are also affected. As a result, it is observed from the constructed tree that the algorithm includes all these attributes in the fault-tree as shown in Figure-4.9 and stops when it finds the necessary condition of high wheel speed (ω_w) in the last right-hand node.

Fault-tree for failure Scenario-4:

A fault similar to the one under failure scenario-4 was injected into the system. *Current example* vector generated by fault injection and feature extraction is as follows:

FLAG	Feature Value for:				
	I_m	T_m	V_b	ω_w	V_c
T	0.3940 (D)	0.0114 (D)	28 (A)	38.75 (A)	2.5210 (A)

The pre-defined ranges which the feature values belong to are shown in the parenthesis below each feature value in the generated vector. When this vector is added with the example sets in Table 4.8 to generate input for fault-tree construction and proposed fault-tree synthesis algorithm is applied on this input by following the attribute selection sequence mentioned in Section 4.3.3, i.e., $V_c \rightarrow I_m \rightarrow T_m \rightarrow \omega_w \rightarrow V_b$, the resulting fault tree is the same as the one in Figure 4.10

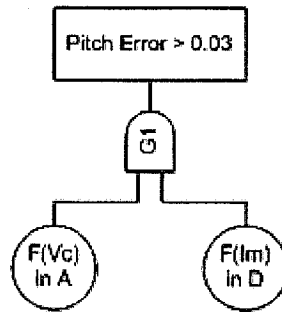


Figure 4.10: Fault-tree for Failure Scenario-4
 (for the selection sequence $V_c \rightarrow I_m \rightarrow T_m \rightarrow \omega_w \rightarrow V_b$)

It is clear from the fault-tree in Figure 4.10 that the tree points towards the torque command voltage V_c and motor current I_m , which are most effected by the error in motor driver unit (MDU) output and increase in friction in the system, by a node in the tree. As mentioned under failure scenarios 1 and 2, from the ACS-model point of view, under this failure scenario, the effect of the anomalous behavior of reaction wheel motor current together with the effect of increased friction is propagated through the closed-loop and the torque command voltage V_c is also affected.

It is obvious that under this failure scenario, the motor torque T_m should also appear on the resulting fault-tree. This is due to the fact that any change in current in the reaction wheel mechanism directly gets translated into torque. It should be mentioned here that from entropy point of view, between I_m and T_m , the tie has been broken by selecting I_m before T_m . In an actual ACS, if the change in entropy is more for T_m as compared to I_m , the resulting fault-tree would look like the one in Figure 4.11 where in addition to the torque command voltage V_c and motor current I_m , motor torque T_m is also identified as a source of anomaly.

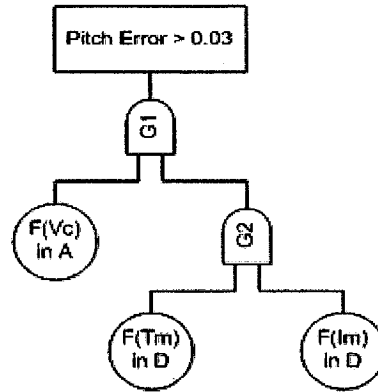


Figure 4.11: Fault-tree for Failure Scenario-4
 (for the selection sequence $V_c \rightarrow T_m \rightarrow I_m \rightarrow \omega_w \rightarrow V_b$)

4.4 Interpretation of the Resulting Fault-trees

In the previous section fault-trees have been constructed under different ACS failure scenarios. The ultimate objective of the proposed framework is to generate a fault tree for a particular failure (top event in the tree). The generated trees display the combination(s) of events leading to the top event. Each event (other than the top event) of the tree will represent an attribute's feature value and its range when the failure took place. Attributes are nothing but different measured signals, for example, current through a component, speed of a rotating body etc.

If the feature extraction function is simple 'min' and/or 'max' values during the time interval for which the fault tree analysis has been performed, the resulting tree will give us information on the failure as follows: *'The top event took place when: the peak value of motor current was within the range (x_1, y_1) ampere AND maximum speed of the reaction wheel was within the range (x_2, y_2) RPM AND minimum bus voltage was within*

the range (x_3, y_3) and so on. Looking at each node of the generated tree, the ACS analyst will be able to figure out what parameter(s) in the system is/are not within the expected range of operation and take action accordingly. In many cases, the generated fault tree may not pin point the exact cause of the failure; but certainly, it will save significant amount of time of the analyst spends on going through each attribute and wondering what all did go wrong within the system. Moreover, the resulting fault-trees can be used during future system design to avoid certain combinations of event that are potential cause of system failure. Finally, if it is necessary to know what exactly components of the system had caused the failure, any ‘basic event’ node in the generated fault-tree may be considered as the ‘top event’ and fault-trees can be constructed to find out what combination of events led to such condition.

If the attribute selection is done arbitrarily, the resulting trees may be bigger in size with more number of nodes. Though such trees are also correct trees, they contain some nodes which are not the exact representation of the cause(s) behind the system failure. For example, under failure scenario-2, if the attribute selection is reversed, the resulting tree looks like the one shown in Figure 4.12 which is different from the one in Figure 4.7. The exact reason behind the failure, in this case, is I_m being in its range ‘B’ AND T_m being in its range ‘B’. The additional three nodes in Figure 4.12 are not the reasons behind the failure – they simply represent the range or condition in which the feature values of the attributes were during anomaly. Though these additional or ‘redundant’ nodes do not lead to an incorrect tree, for large number of attributes, the tree may get unnecessarily big in size and complicate the failure analysis process.

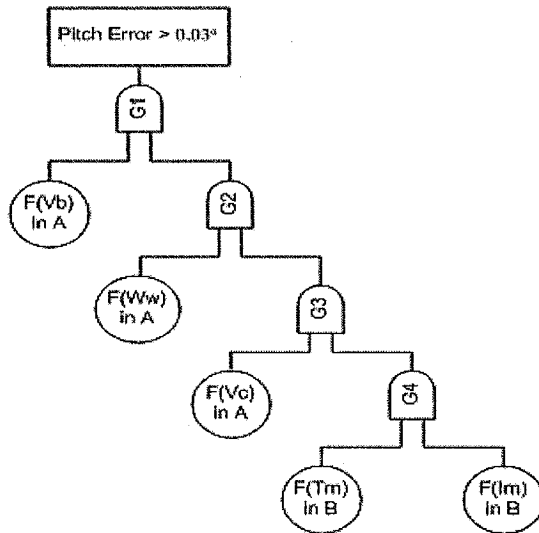


Figure 4.12: Fault-tree for Failure Scenario-2, Case-1 (when the attribute selection sequence is reversed)

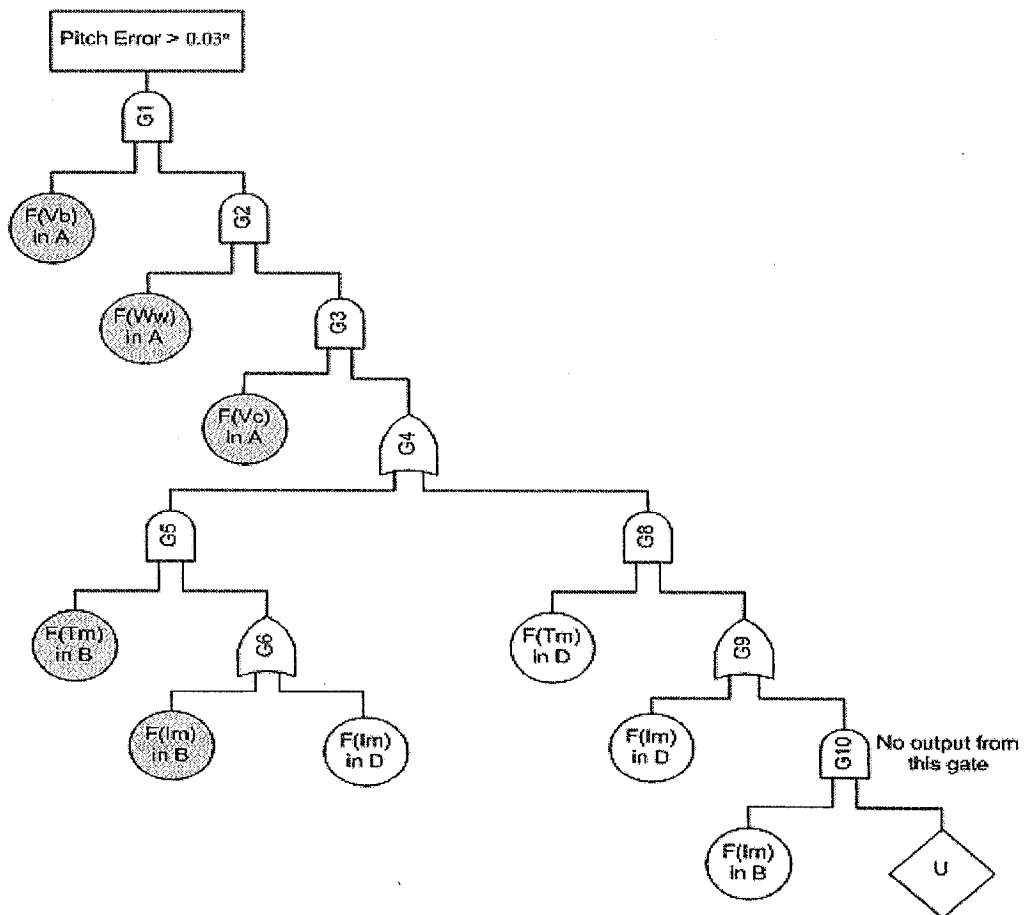
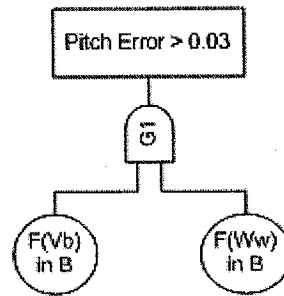


Figure 4.13: Fault-tree for Failure Scenario-2, Case-2 (when the attribute selection sequence is reversed)

Figure 4.13 shows the fault-tree that has been constructed under failure scenario-2 (Case-2, as mentioned in Section 4.3.3) with attribute selection sequence reversed. As in the case of the tree in Figure 4.12, the shape of this tree also differs from the one in Figure 4.8. The shaded nodes in this tree show the *minimal cut-set* representing the combination of events under the failure scenario-2. Both Figure 4.13 and Figure 4.8 convey same information. However, while in the case of Figure-4.8, three ‘redundant’ nodes representing attributes V_b , ω_w and V_c appear in only one branch, in Figure 4.13 they appear in all four branches in the tree. It should be noted here that in reality, number of attributes would be much higher. Consequently, if the selection sequence is not proper, the cut-sets are likely to contain many such ‘redundant’ events and may complicate the analysis process.

Finally, it should be noted that the proposed fault-tree synthesis algorithm has been derived from the ID3 algorithm for induction of decision trees. ID3 was designed to be efficient in a situation where there are many attributes and the training set contains many objects, but where a reasonably good decision tree is required without much computation (as mentioned in Section 3.2.1). It has generally been found to construct simple decision trees, but the approach it uses cannot guarantee that better trees have not been overlooked. As a result, fault-tree synthesis using the proposed algorithm and use of ‘change in entropy’ as the attribute selection criteria do not always guarantee the best tree. For example, under failure scenario-3, if ω_w is selected immediately after V_b the resulting tree becomes as the one in Figure 4.14 which is better than the one in Figure 4.9 in a sense that the earlier one has less number of nodes.



**Figure 4.14: Fault-tree for Failure Scenario-3
(with a different attribute selection sequence)**

4.5 Summary

In this chapter, the four ACS failure scenarios, which have been considered in this thesis, have been explained and fault-tree construction under each failure scenario using the proposed fault-tree synthesis algorithm is demonstrated. It has been shown that the proposed fault-tree synthesis algorithm is capable of synthesizing fault-trees under different failure scenarios. Finally, how the constructed fault-trees should be interpreted is also explained in this chapter.

Chapter 5

Conclusion

A rigorous fault detection, isolation and recovery (FDIR) system is necessary on-board an unmanned spacecraft to encounter undesired events without any need for human intervention from the ground. Consequently, an on-board *fault-diagnosis system* that is capable of detecting, isolating, identifying or classifying faults is required. Development of a strong FDIR scheme for on-board applications is a problem that cannot be classified under a single research area. Moreover, none of the existing fault-diagnosis methodologies alone can meet all the requirements of an ideal fault-diagnosis system. An approach to develop such fault-diagnosis system may be to develop a new methodology that can guarantee all necessary requirements. However, from practical point of view, integration of different existing fault-diagnosis methodologies that may result in a strong unified FDIR scheme seems to be a much better option.

In this thesis, an approach for spacecraft fault diagnosis has been proposed which aims to empower any efficient fault-detection mechanism with a powerful capability of identifying or classifying different type of faults rather than simply determining their presence in the system. The purpose of this extra capability is to ensure a quick and efficient recovery from upsets. The fault-diagnosis approach proposed in this thesis

utilizes a widely used *qualitative* technique called *fault-tree analysis*. Fault-tree synthesis (FTS) and analysis (FTA) methodologies have been used as a diagnostic aid for fault diagnosis in the Attitude Control Subsystem of a satellite.

A generic attitude control subsystem (ACS) model has been developed in this thesis using MATLAB-*Simulink*. The model represents a *zero-momentum* attitude control system with reaction wheel as the actuator. In practice, such actuators have been found to be a potential source of anomalies in the system. Ideal dynamics of attitude sensors has been assumed and the model has been restricted to control along a single axis. In addition, the model allows fault-injection into the system based on the assumed failure scenarios. This purpose of developing this model has been to generate simulated system data under faulty and non-faulty system conditions which is an essential part of the fault-diagnosis study carried out in this thesis.

In order to support the proposed fault-diagnosis approach, a new modified fault-tree synthesis algorithm has been developed from existing machine-learning based induction techniques for fault-tree synthesis. The advantage of this fault-tree synthesis algorithm is that a detailed knowledge of the design and construction of the system under consideration is not required. Given the system parameters under faulty and fault-free conditions, the algorithm can be implemented to induce fault-trees from a set of example vectors that represent the system. Each example in a vector represents a feature-value of a particular system parameter. In the feature extraction task, simple feature extraction functions have been used. An example vector consists of feature values for all the system

parameters or attributes under consideration and a flag to indicate whether or not the vector corresponds to a faulty system behavior.

Four probable ACS failure scenarios have been assumed. Different types of faults have been injected into the ACS model and data for five pre-selected attributes have been collected for faulty as well as non-faulty system behavior. These faults are similar to the ones that people have encountered in practice. Next, features have been extracted from the data and a small set of example vectors has been formed. To demonstrate fault-tree synthesis, different types of faults have been injected into the system. Under each failure scenario, an example vector has been generated. This example vector has been added with the small set of example vectors mentioned above to form the 'input' for the proposed fault-tree synthesis algorithm. Finally, the algorithm has been applied on this generated 'input' and fault-trees have been constructed under four failure scenarios. Each node in the constructed fault-trees represents a basic event (see Chapter 3). These basic events essentially provide information on the attributes when anomalies take place in the system. It has been found from the constructed trees that the proposed algorithm can successfully determine the combination of basic events leading to a failure. Generation of such fault-trees establishes that the proposed framework based on fault tree analysis and synthesis techniques has potential for automated spacecraft health monitoring and diagnosis applications.

It should be noted here that though the fault-tree synthesis using the proposed algorithm has been demonstrated for the attitude control subsystem (ACS), it can be applied to other spacecraft subsystems such as thermal subsystem, electrical power subsystem, etc.

Finally, it may be pointed out here that, to the best of our knowledge, in open literature, there is no published research work for fault-diagnosis using fault-trees applicable to the attitude control subsystem of space vehicles. The automatic fault-tree synthesis methodologies presented in [24, 25] were applied to a mission avionics system which is close to the application domain of the work presented in this thesis.

The contribution of this thesis can be summarized as follows:

- (1) Developed a framework for a fault diagnosis in the attitude control subsystem (ACS) of an unmanned spacecraft. This approach for fault diagnosis does not depend on the design phase of the system.
- (2) Developed a MATLAB-*Simulink* based generic attitude control subsystem (ACS) model with fault-injection capabilities for performing fault diagnosis and failure analysis related studies on unmanned spacecraft subsystems.
- (3) Developed a new modified fault tree synthesis algorithm for analyzing the cause(s) of failure(s) in the ACS. The algorithm utilizes existing machine-learning based induction techniques for fault-tree synthesis. The proposed algorithm is capable of constructing fault-trees without a detailed knowledge on the design and construction of the system under consideration.
- (4) Demonstrated fault-tree synthesis using the proposed fault-tree synthesis algorithm under different ACS failure scenarios which establishes that the proposed framework has potential for automated spacecraft health monitoring and diagnosis.

As a part of the future research work along this direction, some parts of this thesis can be extended in order to obtain better results that can support practical implementation of the proposed approach. First, the ACS model can be made more realistic by incorporating more system dynamics such as sensor dynamics and also considering the cross-coupling effects by extending the developed model to represent control along all three axes. Secondly, in order to capture the behavior of such highly dynamical system, the proposed fault-tree synthesis methodology needs to be modified by including special types of gates that are capable of representing the dynamics of the system as well as temporal relationships among the events. In addition, in order to determine best selection sequence for attributes, it may be necessary to explore more efficient techniques or heuristics, which would always guarantee 'best' fault-trees.

Bibliography

- [1] S. A. Lapp, G. J. Powers “*Computer-aided Synthesis of Fault-trees*” IEEE Transactions on Reliability, Vol.: R-26, pp. 2 – 12, April 1977.
- [2] R. J. Patton, P. M. Frank, R. N. Clark “*Fault Diagnosis in Dynamic Systems: Theory and Applications*” Prentice Hall, 1989
- [3] T. F. Petti, J. Klein, P. S. Dhurjati “*Diagnostic Model Processor: Using Deep Knowledge for Process Fault Diagnosis*” AIChEJ, Vol.:36, No.:4, pp.: 565 – 575 , 1990
- [4] B. Fenton, M. McGinnity , L. Maguire “*Fault Diagnosis of Electronic System using Artificial Intelligence*” IEEE Instrumentation & Measurement Magazine, Vol.: 5 , Issue: 3 , pp.:16 – 20, September 2002
- [5] J. T. Y. Cheung, G. Stephanopoulos, “*Representation of Process Trends – Part I: A Formal Representation Framework*”, Computers & Chemical Engineering, Vol.: 14, No. 4/5, pp. 495–510, 1990
- [6] B. R. Bakshi, G. Stephanopolous “*Representation of Process Trends – IV: Induction of Real-Time Patterns from Operating Data for Diagnosis and Supervisory Control*” Computers and Chemical Engineering, Vol.: 18, No.:4, pp.:303 – 332, 1994
- [7] J. F. Macgregor, J. Christina, K. Costas, M. Kotoudi “*Process Monitoring and Diagnosis by Multi-block PLS Methods*” AIChEJ, Vol.:40, No.:5, pp.: 826 – 838 , 1994
- [8] K. Madani “*A Survey of Artificial Neural Networks Based Fault Detection and Fault Diagnosis Techniques*” International Joint Conference on Neural Networks, 1999. (IJCNN '99), Vol.: 5, pp.:3442 – 3446, July 1999.
- [9] W. J. Larson, J. R. Wertz (Eds.) “*Space Mission Analysis and Design*” Second Edition, Kluwer Academic Publishers, 1999.

- [10] P. Fortescue , J. Stark (Eds.) “*Spacecraft Systems Engineering*” Second Edition, John Wiley & Sons Inc., 1995.
- [11] D. Speer, P. Sanneman “*Attitude Determination and Control for the New Millennium EO-1 Spacecraft*” Proceedings of IEEE Aerospace Conference, Vol. 1, pp. 93 – 109, March 1998.
- [12] A. Papoulis, S. U. Pillai “*Probability, Random Variables, and Stochastic Processes*” Fourth Edition, McGraw-Hill, 2002.
- [13] V.A. Chobotov “*Spacecraft Attitude Dynamics and Control*” Original Edition, Krieger Publishing Company, 1991.
- [14] B. Bialke “*High Fidelity Mathematical Modeling of Reaction Wheel Performance*” Advances in the Astronautical Sciences, Vol. 98. pp. 483 –496, 1998.
- [15] M. E. Perry, P. Alea, M. J. Cully, M. McCullough, P. Sanneman, N. Teti, B. Zink “*Earth Observing-1 Spacecraft Bus*” Swales Aerospace. Web-link: www.swales.com/pdf/eosb.pdf (as on April 9, 2004).
- [16] C. A. Ericson II “*Fault Tree Analysis – A History*” Proceedings of the 17th International System Safety Conference, 1999.
- [17] L. L. Pullum, J.B. Dugan “*Fault Tree Models for the Analysis of Complex Computer-based Systems*” Proceedings of Annual Reliability and Maintainability Symposium, 'International Symposium on Product Quality and Integrity', pp:200 – 207, January 1996.
- [18] J. B. Fussell “*Synthetic Tree Model*” Report ANCR 1098, Aerojet Nuclear Company, March 1973.
- [19] J. B. Fussell, G. J. Powers, R.G. Bennetts “*Fault Trees – A State of Art Discussion*” IEEE Transactions on Reliability, Vol.: R-23, No. 1, April 1974.
- [20] W. E. Vesley, F.F. Goldberg, N.H. Roberts, D.F. Haasl “*The Fault Tree Handbook*” Technical Report USNRC NUREG 0492, United States’ Nuclear Regulatory Commission, January 1981.
- [21] J. R. Taylor “*An Algorithm for Fault Tree Construction*” IEEE Transactions on Reliability, Vol. R-31, No. 2, pp. 137- 146, June 1982.
- [22] R. C. Vries “*An Automated Methodology for Generating a Fault Tree*” IEEE Transactions on Reliability, Vol. 39, No. 1, pp.76- 146, April 1990.

- [23] S. Chi, S. Lee, S. Park “*Automated Generation of Fault Tree using the Symbolic DEVS Simulation*” Proceedings of AIS, 1993.
- [24] J.B. Dugan, S. J. Bavuso, M. A. Boyd “*Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems*” IEEE Transactions on Reliability, Vol. 41, Issue: 3, pp: 363 – 377, September, 1992.
- [25] K. K. Vemuri, K.K.; J.B. Dugan, K.J. Sullivan “*Automatic Synthesis of Fault Trees for Computer-Based Systems*” IEEE Transactions on Reliability, Vol. 48, Issue: 4, pp:394 – 402, December 1999
- [26] T. Assaf, J.B. Dugan “*Automatic Generation of Diagnostic Expert Eystems from Fault Trees*” Annual Reliability and Maintainability Symposium, 2003, pp: 143 – 147, January 2003.
- [27] Y Papadopoulos, M. Maruhn “*Model-Based Synthesis of Fault Trees from Matlab – Simulink models*” Proc. The International Conference on Dependable Systems and Networks, 200, pp: 77– 82, July 2001.
- [28] Y Papadopoulos, C. Grante “*Techniques and Tools for Automated Safety Analysis & Decision Support for Redundancy Allocation in Automotive Systems*” Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC’03), pp: 105-110, November 2003.
- [29] A.G.T. Raaphorst, B.D. Netten, R. A. Vingerhoeds “*Automated Fault-tree Generation for Operational Fault Diagnosis*” International Conference on Electric Railways in a United Europe, 1995, pp.:173 – 177, March 1995.
- [30] M.G. Madden, P.J. Nolan “*Generation of Fault Trees from Simulated Incipient Fault Case Data*” Proc. 9th International Conference on Applications of Artificial Intelligence in Engineering, Pennsylvania, USA. pp: 567-574, 1994.
- [31] J. R. Quinlan “*Induction of Decision Trees*” Machine Learning, Vol.:1, pp. 81-106, 1986.
- [32] M.G. Madden, P.J. Nolan “*Monitoring and Diagnosis of Multiple Incipient Faults Using Fault Tree Induction*” IEE Proceedings on Control Theory and Applications, Vol. 146, Number 2. March 1999.
- [33] G. Kocza, A. Bossche “*Automatic Fault Tree Synthesis and Real-Time Tree Trimming, Based on Computer Models*” Proceedings of Annual Reliability and

Maintainability Symposium, pp. 71– 75, 1997.

- [34] N. G. Leveson “*SAFWARE: System Safety and Computers*” Addison-Wesley Publishing Company, 1995.
- [35] R. M. Sinnamon, J.D. Andrews “*Fault Tree Analysis and Binary Decision Diagrams*” Proceedings of Annual Reliability and Maintainability Symposium, 'International Symposium on Product Quality and Integrity', pp: 215 – 222, January 1996.
- [36] L.M. Bartlett, J.D. Andrews “*Choosing a Heuristic for the 'Fault Tree to Binary Decision Diagram' Conversion, using Neural Networks*” IEEE Transactions on Reliability, Vol.: 51, Issue: 3, pp: 344 – 349, September, 2002.
- [37] L.M. Bartlett, J.D. Andrews “*Efficient Basic Event Orderings for Binary Decision Diagrams*” Proceedings of Annual Reliability and Maintainability Symposium, pp: 61 – 68, January 1998.

Appendix

Main MATLAB-Simulink blocks of the developed ACS Model:

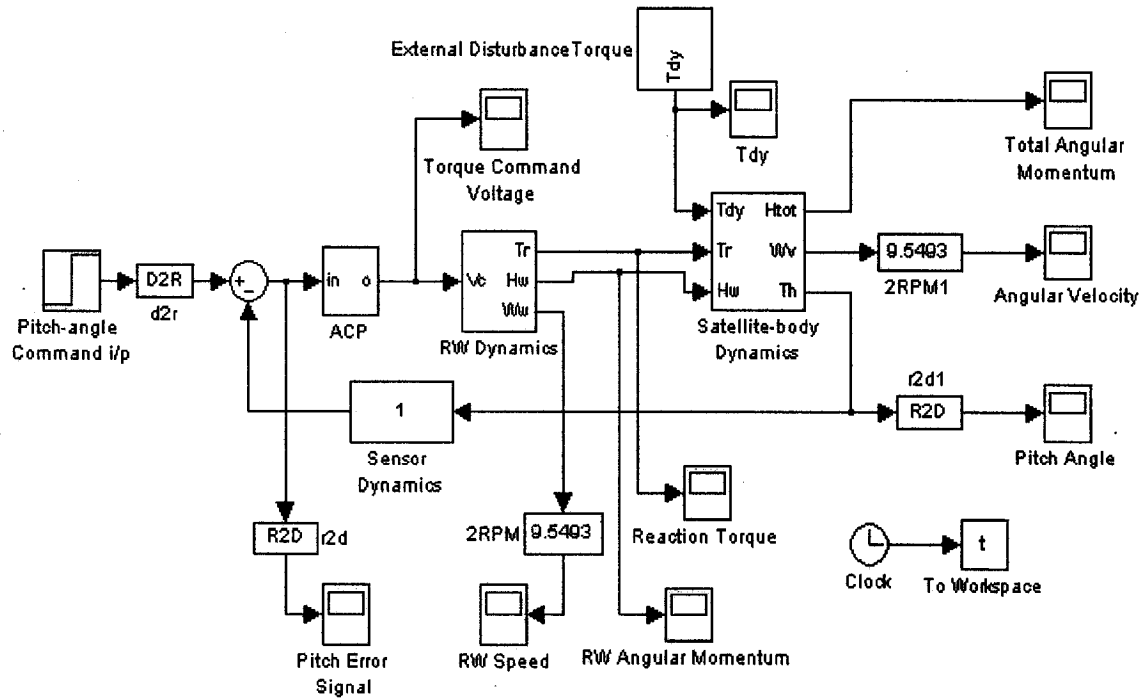


Figure A: Simulink Model of ACS

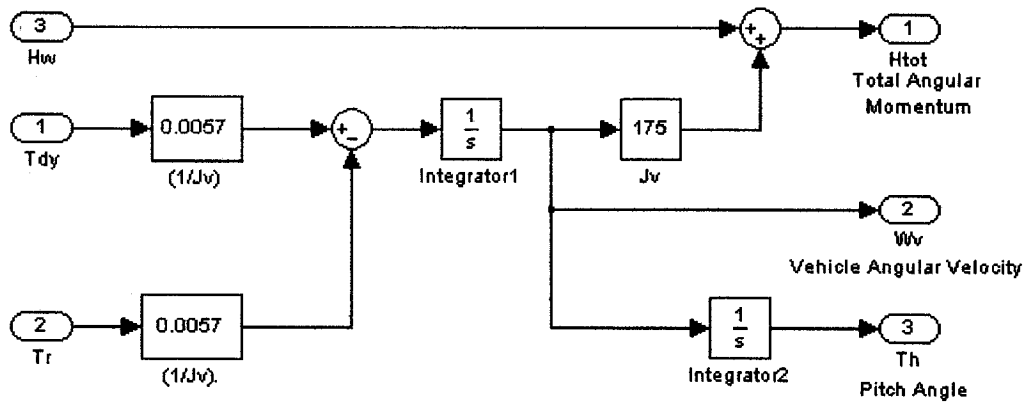


Figure B: Satellite Body Dynamics

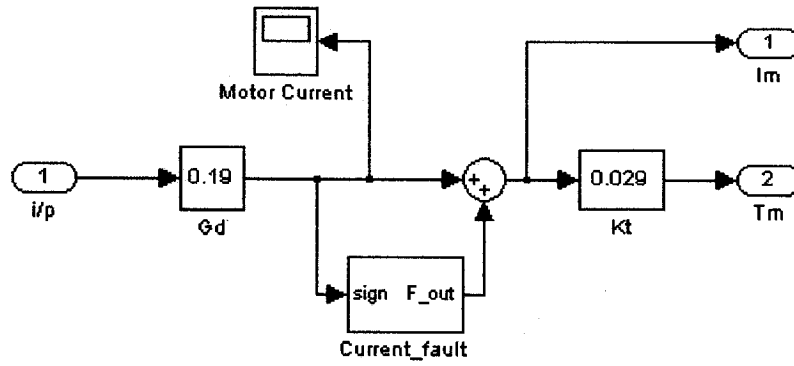


Figure D: Motor Torque Control

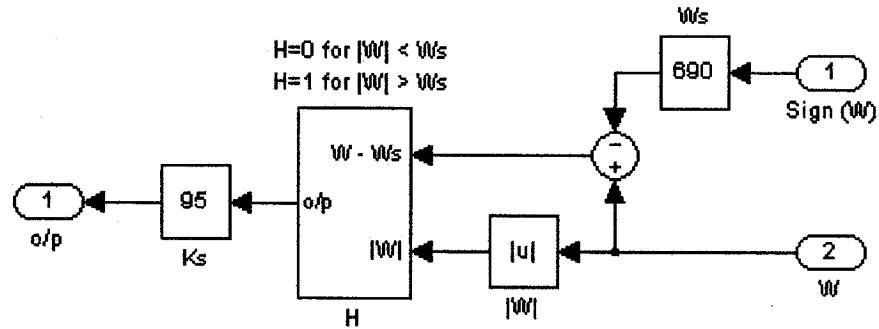


Figure E: Speed Limiter

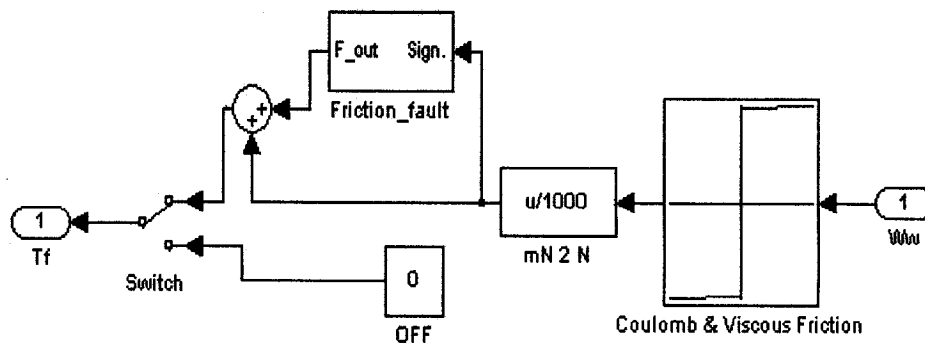


Figure F: Friction Model

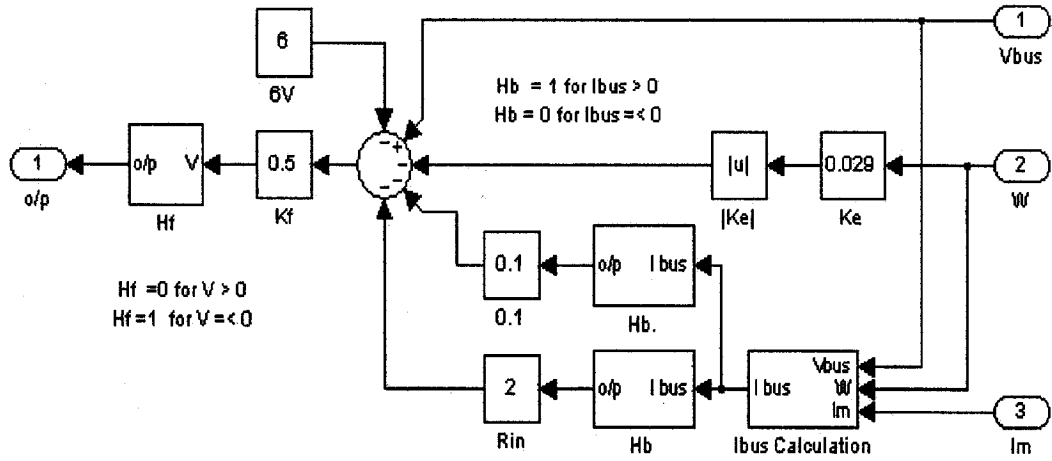


Figure G: EMF Torque Limiting