Proceedings of the
2005 IEEE Conference on Control Applications
Toronto, Canada, August 28-31, 2005

A Novel Fault-Tree Approach for Identifying Potential Causes of Satellite Reaction Wheel Failure

A. Barua, P. Sinha, K. Khorasani
Department of Electrical and Computer Engineering
Concordia University
Montreal, Quebec H3G 1M8 CANADA
E-mail: {a_barua,sinha,kash}@ece.concordia.ca

S. Tafazoli
Canadian Space Agency
6767, route de l'Aeroport
Saint-Hubert, Quebec, J3Y 8Y9 CANADA

Abstract— **Due to unforeseen circumstances and naturally occurring faults, it is desired that an on-board fault-diagnosis system of a space vehicle be capable of detecting, isolating, identifying or classifying faults in the system. In this paper, a novel approach is proposed which strengthens existing efficient fault-detection mechanisms with an additional ability to classify different types of faults to effectively determine potential failure causes 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, as a diagnostic aid, for failure analysis in the attitude control subsystem (ACS) of a spacecraft. Constructed fault-trees have been able to represent combinations of events leading to different failures resulting due to artificially injected faults in a MATLAB-*Simulink* model of the ACS. It is important to emphasize that proposed technique has potentials for being integrated in an on-board health monitoring and diagnosis tool for space vehicles.**

I. Introduction

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.

Development of a FDIR scheme 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. While the desired 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 through a complementary procedure that incorporates fault-tree analysis based techniques in on-board fault diagnosis and recovery system. 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.

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 this paper, we propose a framework for automated spacecraft fault-diagnosis.

The organization of this paper is as follows: in Section II, the proposed approach for fault-diagnosis has been presented. A brief review on fault-trees and the algorithm developed in this work for fault-tree synthesis utilizing learning techniques are presented in Section III. In Section IV, a generic attitude control subsystem (ACS) model of a satellite and different possible ACS failure scenarios based on this model are discussed. Demonstration of fault-tree synthesis under different ACS failure scenarios is provided in Section V. Finally, conclusions are stated in Section VI.

II. Proposed Approach

The proposed diagnosis procedure does not depend on the design phase of the system (explained in details in Section III). Figure 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 requirements. 'Attributes' are defined as the signals that are measured in the ACS subsystem; for example, the current drawn by the actuator or reaction wheel motor is an attribute. Similarly, the speed of the reaction wheel is another attribute and so on.

After acquiring data for pre-selected attributes over the above-mentioned time-frame, the next step is to extract features (such as a minimum or an average value) from the data (will be discussed in Section III-A in detail). Subsequently, a vector of numeric feature values for attributes is generated in which each feature value corresponds to an attribute over the pre-specified time-frame X mentioned above. We call this vector as *current example*.

Once the *current example* vector is created by feature extraction from the attributes, the next task is fault-tree synthesis. For this purpose, the information available from the *current example* is combined with the 'knowledge' available in an *example database* that had been generated before. This *example database* consists of the vectors of numeric feature values for all the pre-selected attributes. This means, each element in an example vector is the *feature value* of a particular attribute. Each example is tagged

This work was supported in part by the National Science and Engineering Research Council of Canada (NSERC) under the Strategic Grant Project # STPGP 258007

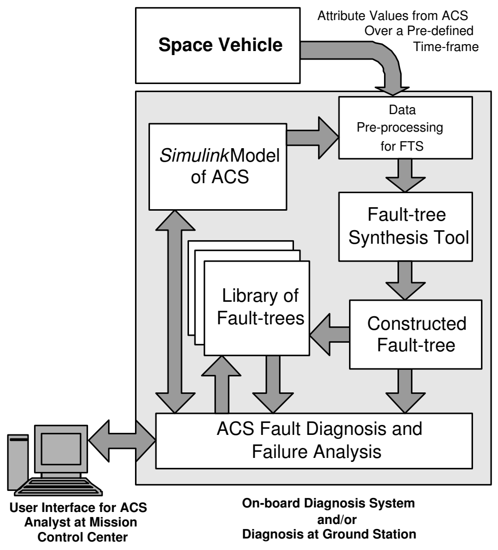

Space Vehicle
Attribute Values from ACS Over a Pre-defined Time-frame
Data Pre-processing for FTS
Simulink Model of ACS
Fault-tree Synthesis Tool
Library of Fault-trees
Constructed Fault-tree
ACS Fault Diagnosis and Failure Analysis
User Interface for ACS Analyst at Mission Control Center
On-board Diagnosis System and/or Diagnosis at Ground Station

Fig. 1. Proposed Framework for the ACS Fault Diagnosis.

with either a TRUE flag or a FALSE flag to indicate if the example is associated with a faulty or a non-faulty system condition respectively. *Example database* can be generated manually through the simulation of the ACS model and/or from the mission data — through a semi-automatic procedure— after the spacecraft reaches steady-state condition in the orbit in which it is intended to operate. The *example database* can be updated from time to time to enhance the knowledge on the system as time progresses and the operational condition of the system changes.

The *current example* and the *example database* are utilized by the fault-tree synthesis tool to construct fault-trees using the proposed fault-tree synthesis algorithm (presented in Section III-C). The constructed fault-tree can be used for fault diagnosis and added to a *library* for future reference.

Above computations can be performed in real time or near-real time, in parallel with the ACS subsystem when prompted by the fault-detection mechanism, on-board as well as at ground if necessary. It is possible to 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 data as compared to those available through telemetry points which are usually sampled at lower frequencies. 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.

III. Fault-Tree Synthesis and Analysis

Fault-trees have been extensively used in system safety and reliability analysis, as well as in system fault diagnosis, for more than 40 years [1], [2]. 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. In this work, fault-trees have been used as a diagnostic aid for fault diagnosis and health monitoring in the attitude control subsystem (ACS) of a satellite. This approach is different from the fault-tree construction where the aim is to perform system reliability and safety analysis

It is worthwhile to point out here that a fault-tree 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 for this work.

For our discussion, it is convenient to divide the whole problem of fault-tree into two parts—fault tree synthesis (FTS) or construction and fault tree analysis (FTA). Though the objective of this work is to develop a methodology for automatic FTS, it is worthwhile to discuss briefly about the FTA part. Fault-tree analysis can be performed mainly in two ways [2], [3]: (a) by *cut-set analysis*, which is a classical approach (b) through *Binary Decision Diagrams* (BDDs), which is relatively new. According to [2], a *cut set* is a combination of *basic events* that can cause the *top event* in a fault-tree. A *minimal cut set* (MCS) is the smallest combination of *basic events* that results in the *top event*. A fault-tree consists of a finite number of MCSs that are unique for a *top event*.

Fault-tree generation or synthesis (FTS) is considered to be a relatively difficult problem 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 construction can be extremely time-consuming and expensive and is likely to lead to human errors. Consequently, there has been a demand for automatic fault-tree synthesis from the point of cost reduction, result standardization and also for understanding the construction process.

The process of automating the fault-tree synthesis task was initiated in 1970's by J. B. Fussell [4]. 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. Most of them are focused towards system reliability analysis. Some of the other important works in this area are available in [5]–[11]. In all these methodologies, there has been a requirement for an in-depth knowledge on the system under consideration.

The authors in [12] described a machine-learning based 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, i.e., faults as well as under normal system condition. These example vectors are formed by extracting features from the data. The algorithm they developed was based on the ID3 algorithm [13] for induction of decision trees. The proposed algorithm for fault-tree synthesis has been developed by continuing along the same theme as that in [12] and [13]. But before presenting the algorithm, we would like to discuss the two most important factors in fault-tree construction by the proposed methodology — feature extraction and ordering of attributes.

A. Feature Extraction from Data

As mentioned above, the 'IFT' approach for fault-tree synthesis uses a set of example vectors which is formed by extracting features from the attributes. For fault-diagnosis applications, the main purpose of feature extraction is to find the symptoms of the presence of anomaly in the system. Another advantage of extracting features from the data is that it maps a large problem space into a smaller feature space by applying appropriate feature extraction function(s) on the available data. 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. and (c) Curve fitting techniques. In this work, 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 absolute value of time domain signals over a pre-defined time-frame (as discussed in Section II).

B. Ordering of Attributes

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 available 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. Recently, to handle the problem of detrmining the attribute selection sequence for binary decision diagrams (BDDs), genetic algorithm and neural-network based techniques have been suggested [14], [15]. A rigorous research on attribute selection has not been performed in this work. In order to have a rational for attribute selection, a very common and simple attribute selection criterion based on 'entropy' has been used for demonstraing fault-tree synthesis by the proposed algorithm. One way to calculate the entropy $E(A)$ of an attribute A over some interval is to use the well-known entropy formula:

$$E(A) = -\sum_{i=1}^{n} A_i^2 \log(A_i^2)$$

where, n is the number of samples in that time interval and $\log(A)$ is the natural logarithm of A. In this work, the percentage change in entropy between non-faulty and faulty system conditions has been considered as an attribute selection criterion. Specifically, the entropy is calculated using the above equation for each attribute over a pre-defined time-frame (as discussed in Section II) during normal operating condition of the system and under presence of anomaly. The percentage changes between these two entropies are calculated and attributes are selected in the descending order of the percentage changes, i.e., the attribute with largest change in entropy is selected as the first attribute. For the entropy calculation, data for all the attributes are normalized to a same scale.

C. Proposed Algorithm for Fault-Tree Synthesis

The proposed algorithm for FTS is an extended and generalized version of the algorithm that was introduced in [16], i.e., it adopts and transforms the one presented in [12] to suit our fault diagnosis requirements. The goal of the proposed algorithm is to generate a fault-tree for a detected (by the fault-detection mechanism) failure utilizing machine-learning from the given inputs to the algorithm. Readers may refer to Section II for better understnading of the terminologies used in the proposed algorithm.

PROPOSED ALGORITHM:

A. *Form a TOP NODE with the undesired event.*

B. 1. *Select an attribute A_x. Read its feature value $F(A_x)$ in the current example. Determine the pre-defined range i which this 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 that $F(A_x)$ belongs to.

For each $(F(A_x) \in i)$ node, replace the node by an AND gate and place the same $(F(A_x) \in i)$ at one input branch of the AND gate.

2.b. (Only if called from 3.a.:)

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 the same $(F(B_x) \in i)$ at one input branch of the AND gate

2.c. (Only if called from 3.c.:)

For each $(F(B_x) \in j)$ node, replace $(F(B_x) \in j)$ by an AND gate and place the same $(F(B_x) \in j)$ 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, $F(A_x) \in i$.

3.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 execute the algorithm iteratively.*

If no more attribute is available, connect an 'Unknown Event' node with the remaining input of the AND gate and STOP.

3.b. *If all examples are TRUE, then STOP.*

3.c. *Otherwise, to develop the other branch of the AND gate, select next attribute B_x in the current example 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 iteratively 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 iteratively for B_x.

If no more attribute is available, connect an 'Unknown Event' node with the remaining input of the AND gate and STOP.

C. *Remove all single-input OR and AND gates.*

The algorithm can be implemented to generate a matrix that represents a fault-tree. A row in this matrix would describe the position of an object (nodes and gates) in the fault-tree with respect to it's parent, i.e., the object above it to which it is connected.

It is important to note the differences between the proposed algorithm and the algorithm presented in [12]. The algorithm in [12], after selecting an attribute A_x with feature value $F(A_x)$ and threshold T_x, considers both the cases when $F(A_x) < T_x$ and $F(A_x) \geq T_x$. But in our work, for all attributes under consideration, different faulty and non-faulty ranges for numeric feature values have been specified or defined. The advantage gained is that a range is relatively easy to determine as compared to an exact threshold point for a feature value. Upon selection of the numeric *feature value* in the *current example* for a particular attribute, the proposed algorithm considers only that subset of examples in which the *feature value* in the *current example* and the feature values in the *example database* for that attribute are in the same range. Further, when the algorithm considers an attribute A_2 (for the event $F(A_2) \in R_1$; where R_1 is a pre-defined range for attribute A_2) after considering an attribute A_1 (for an event $F(A_1) \in R_2$; where R_2 is a pre-defined range for attribute A_1), it considers only those examples in the database which satisfy both the properties: '$F(A_2) \in R_1$' and '$F(A_1) \in R_2$'.

From a practical point of view, sometimes it may be difficult to specify distinct and non-overlapping ranges for a particular feature. In such cases, it is possible that a feature value may belong to more than one class (range). Our proposed algorithm is capable of handling such cases by putting an *OR* gate in the tree.

IV. Attitude Control Subsystem of Space Vehicles

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 of the spacecraft. Attitude of a spacecraft may be specified in a number of ways such as quaternions, direction cosines, Euler's angles etc. When Euler's angles are used to specify the attitude, the information required for 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.

The major components of the ACS [17], [18] are: 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 (for example, sun sensors, Earth sensors or horizon scanners, star trackers, magnetometers, etc.) 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 and specifications are attained.

A. Modeling of the ACS

In the above-mentioned CSA-satellite (see Section I), 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 access 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.

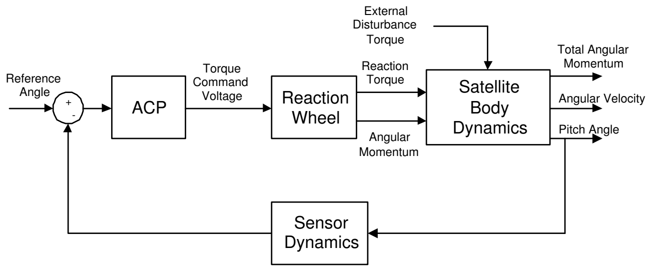


Fig. 2. Functional Diagram of the ACS.

Figure 2 shows the functional diagram for the developed MATLAB-$Simulink$ model of the ACS using a 'feedback control' loop. As mentioned above, the 'ACP' is the controller and the 'Reaction Wheel' is the actuator used for maintaining the required attitude or orientation of the 'Satellite Body'. A *zero-momentum* system with reaction wheels has been modeled in $Simulink$ 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 effects. In a *zero-momentum* system, RWs respond to disturbances on the spacecraft. Based on the attitude error(s), the 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) [17] 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. The actuator or the 'reaction wheel' block in the feedback control loop of the developed ACS model has been primarily based on the high fidelity mathematical model of the *Ithaco Type-A* reaction wheel presented in [19]. This reaction wheel model has been extended and modified in order to include fault injection capabilities. Also, an ideal dynamics (without any error or time delay) for attitude sensors has been assumed.

B. ACS Failure Scenarios

Since our study has been based on a simulated ACS model, for fault diagnosis purposes it has been necessary to assume some failure scenarios which resembles some real-life faults and failures by utilizing the concept of fault injection into the system. In the developed ACS model, it has been assumed that a maximum attitude error of $0.03\,^\circ$ in the pitch angle can be tolerated. Consequently, for all of the ACS failure scenarios the top events in the fault-trees have been identified as 'Pitch Error $> 0.03\,^\circ$'. Figure 3 shows the pitch angle error response under failure scenario-3. Responses for other scenarios are not presented due to a limited space. The failure scenarios are described in the sebsequent discussion.

1) Failure Scenario-1: Random increase in reaction wheel motor current. The purpose of this fault is to represent failure when there is a surge in the motor current due to some hardware level failure in the motor driver unit (MDU) in the reaction wheel (RW).This fault has been injected when the RW was running near zero speed.

2) 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 increases in the wheel bearings due to the wear of bearing material over time or due to some problems in the lubricant flow.

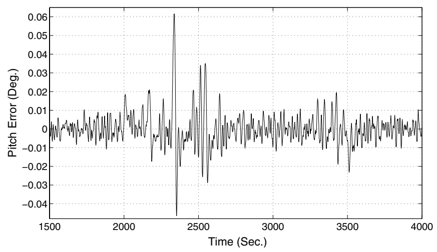


Fig. 3. Pitch Error Response under Failure Scenario-3

3) 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-voltage condition when a large back-EMF, developed in the reaction wheel motor operating at a high speed, limits the motor current and consequently the motor torque. Fault has been injected between $t = 2000$ and $t = 3500$ seconds (Figure 3).

4) Failure Scenario-4: Small error in motor current together with increase in wheel-bearing friction. In this case, we show that though a small error in motor driver unit (MDU) output or small

TABLE I

SPECIFIED RANGES FOR FEATURE VALUES OF THE ATTRIBUTES

Attribute	Ranges of Feature Value	Assigned Name for Range	Corresponding Failure Scenario
I_m	0.55 – 0.86	A	1 & 3
	0.4 – 0.549	B	2
	0.313 – 0.470	D	4
T_m	0.0160 – 0.0249	A	1 & 3
	0.0121 – 0.0159	B	2
	0.0090 – 0.0136	D	4
ω_w	30 – 50	A	1 , 2 & 4
	5100 and above	B	3
V_b	21 – 28	A	1 , 2 & 4
	19 – 20.99	B	3
V_c	2.0 – 3.5	A	1 , 2 & 4
	4.99 and above	B	3

increase in friction does not lead to any failure individually, when both of these take place together, ACS may fail to maintain the required attitude. This scenario has been simulated when the RW was running near zero speed.

V. FTS FROM SIMULATED ACS MODEL DATA

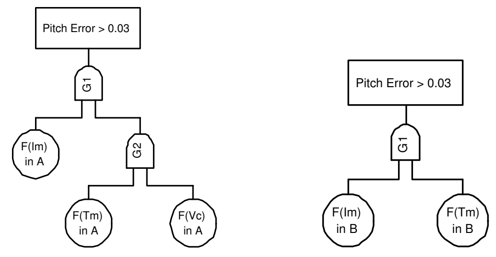
Fault-trees have been constructed based on 5 (five) attributes of the ACS. They are: reaction wheel *motor current* (I_m), reaction wheel *motor torque* (T_m), *wheel speed* (ω_w); *bus voltage input* (V_b) to the reaction wheel and *torque command input voltage* (V_c) to the reaction wheel.

Basic building blocks of fault-trees are available in [2]. In this work, five types of nodes have been used for tree synthesis: *Top Event* nodes (represented by rectangles describing the failure to be analyzed), *AND* gates (represented by a standard *AND* logic symbol), *OR* gates (represented by a standard *OR* logic symbol), *Basic Event* nodes (represented by circles showing appropriate basic events) and *Unknown Event* nodes (represented by square boxes with the letter *U* inside). While constructing fault trees, if insufficient number of attributes are used and/or if insufficient examples are given to the algorithm from which the algorithm is unable to determine a finite set of *basic events*, it places an *unknown event* node in the tree saying that a branch in the fault-tree could not be developed further due to the lack of information on the system.

Different types of faults (as discussed in Section IV-B) have been injected in the developed ACS model and different *current examples* have been generated. It should be noted that all the fault-tree results presented here have been constructed for unseen data, i.e., the data used for generating the *current example* has not been used for generating any example vector in the *example database*. Under each faulire scenario, as well as under fault-free condition, a number of example vectors were generated and the *example database* was created by merging all of these resulted example vectors.

Next, the *ranges* for each attribute under consideration were determined for all the four failure scenarios. Table I summerizes the ranges of different feature values under different failure scenarios.

As discussed in Section III-B, we need to order the attributes in an appropriate sequence. Every time we construct a fault-tree, we need to do this. For different failure scenarios, the attribute selection sequences were determined as follows: under failure scenarios 1 and 2, the attribute selection sequence have been: $I_m \rightarrow T_m \rightarrow V_c \rightarrow \omega_w \rightarrow V_b$; under scenario 3, the attribute selection sequence has been: $V_b \rightarrow V_c \rightarrow I_m \rightarrow T_m \rightarrow \omega_w$; and under scenario 4, the attribute selection sequence has been: $V_c \rightarrow I_m \rightarrow T_m \rightarrow \omega_w \rightarrow V_b$. Figure 4(a) shows the fault-tree that resulted after a fault similar to the one under failure scenario-1 was injected into the system and with the attribute selection sequence mentioned above. In this case, each feature value in the *current example* belonged to only one specified class or *range*. The constructed fault-tree successfully



(a) Scenario 1 (b) Scenario 2

Fig. 4. Fault-trees for Failure Scenarios 1 and 2.

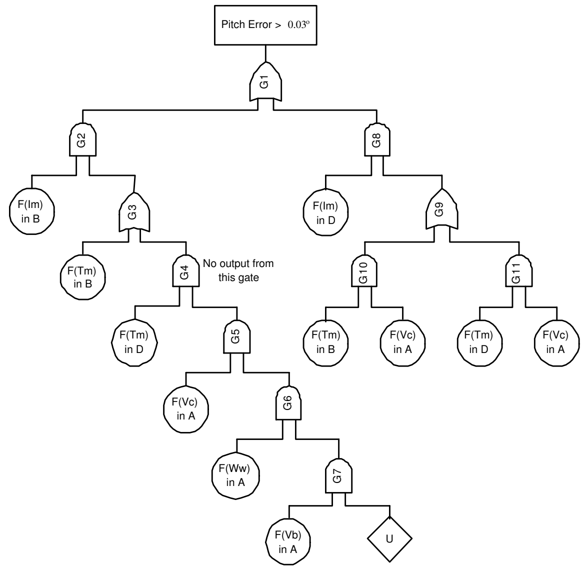
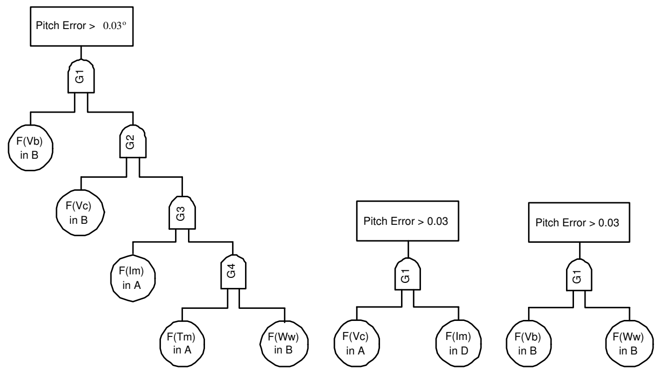


Fig. 5. Fault-tree for Failure Scenario-2 (for overlapping ranges).

classifies the injected fault by putting appropriate *basic events*. The tree provides information that matches with those in Table I. The fault-tree points towards motor current I_m by a node in the tree. From the developed ACS model point of view, in presence of fault, the anomalous behavior of the reaction wheel motor current is directly translated into motor torque. Since there is a closed-loop control in the ACS, the fault propagates 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 appear in the resulting fault-tree as the sources of anomaly in the system.

Figure 4(b) shows the fault-tree that resulted under failure scenario-2 with the attribute selection sequence mentioned above. In this case, each feature value in the *current example* belonged to only one specified class or range. It is clear from the fault-tree in Figure 4(b) 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. Moreover, in order to overcome the increased 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. Consequently, the motor current I_m, also appears in the resulting fault-tree as the source of anomaly in the system.

Figure 5 shows the fault-tree that resulted under failure scenario-2 with the same attribute selection sequence but some feature values in the *current example* belonged to more than one specified *ranges*. It is clear from the fault-tree in Figure 5 that the tree represents four possible combinations of events behind the top event. If the *cut-sets* (as discussed in Section III) of this tree are determined, it is found that one of the *cut-sets* formed using the two events $I_m \in B$



(a) Scenario 3 (b) Scenario 4 (c) Scenario 3

Fig. 6. Fault-trees for Failure Scenarios 3, 4 and 3(with a modified Attribute Selection Sequence).

and $T_m \in B$ of the tree represents events which are exactly the same as the ones represented by the fault-tree in Figure 4(b). Out of the remaining three branches, one branch (in which $I_m \in B$ and $T_m \in D$) does not converge and terminates by putting an *unknown event* node as discussed earlier. The next branch (in which $I_m \in D$ and $T_m \in D$) represents the failure scenario-4. The last branch (in which $I_m \in D$ and $T_m \in B$) gives a new combination of events, 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 database* used for fault-tree synthesis but was unknown until the fault-tree had been constructed.

Figure 6(a) shows the fault-tree that resulted under failure scenario-3 with the attribute selection sequence mentioned above. The constructed fault-tree has been able to identify the two basic events $V_b \in B$ and $\omega_w \in B$ that are responsible for the failure. Similar results, as shown in Figure 6(b), can be obtained for failure scenario-4 which is not discussed in details due to space limitation.

It is important to point out that the constructed trees are not always the minimum or 'best trees'. This important observation can be made by changing the attribute selection sequence under failure scenario-3 to: $V_b \rightarrow \omega_m \rightarrow ...$ and so on, i.e., if ω_w is selected right after V_b. In that case, the fault-tee was found to have only two nodes, that represents the failure-causes more clearly as shown in Figure 6(c). Clearly, other efficient criterion for deciding the attribute selection sequence are likely to provide better results in such cases.

VI. Conclusions and Future Works

In this paper, a framework 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 types 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 proposed approach utilizes a widely used qualitative technique known as fault-tree analysis. 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. Constructed fault-trees have been found to be able to represent appropriate combinations of events leading to different failures resulting due to artificially injected faults in the ACS model developed using MATLAB-$Simulink$. 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 [9], [10] were however applied to a mission avionics system which is not too close to the application domain of the work presented in this paper. A more complete version of the outcome of this research with an improved version of our FTS algorithm as well as more detailed results and discussions has been submitted for publication [20].

Having been encouraged by the results obtained from this work, as a part of our on-going research, we are developing a more complex ACS model by including cross-coupling effects and sensor dynamics. We are also hopeful to develop a mehodology for automatic synthesis of dynamic fault-trees in order to incorporate causal relationships among the events, which may be more suitable for fault-diagnosis in complex dynamical systems.

References

[1] C. A. Ericson, "Fault Tree Analysis – A History," in *Proc. 17th International System Safety Conference*, Florida, USA, August 1999.

[2] W. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback, "Fault Tree Handbook with Aerospace Applications," NASA Office of Safety And Mission Assurance, NASA Headquarters, Washington, DC 20546, Tech. Rep. Version 1.1, August 2002.

[3] R. M. Sinnamon and J. D. Andrews, "Fault Tree Analysis and Binary Decision Diagrams," in *Proc. Annual Reliability and Maintainability Symposium (International Symposium on Product Quality and Integrity*, Las Vegas, USA, January 1996, pp. 215–222.

[4] J. B. Fussell, "Synthetic Tree Model," Aerojet Nuclear Company, Tech. Rep. ANCR 1098, March 1973.

[5] J. B. Fussell, J. G. Powers, and R. G. Bennetts, "Fault Trees – A State of Art Discussion," *IEEE Transactions on Reliability*, vol. R-23, No.1, pp. 2–12, April 1974.

[6] J. R. Taylor, "An Algorithm for Fault-Tree Construction," *IEEE Transactions on Reliability*, vol. R-31, No.2, pp. 137–146, June 1982.

[7] S. A. Lapp and J. G. Powers, "Computer-aided Synthesis of Fault-Trees," *IEEE Transactions on Reliability*, vol. R-26, pp. 2–12, April 1977.

[8] R. C. Vries, "An Automated Methodology for Generating a Fault Tree," *IEEE Transactions on Reliability*, vol. 39, No.1, pp. 76–146, April 1990.

[9] J. B. Dugan, S. J. Bavuso, and M. A. Boyd, "Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems," *IEEE Transactions on Reliability*, vol. 41, No.3, pp. 363–377, September 1992.

[10] K. K. Vemuri, J. B. Dugan, and K. J. Sullivan, "Automatic Synthesis of Fault Trees for Computer-Based Systems," *IEEE Transactions on Reliability*, vol. 48, No.4, pp. 394–402, December 1999.

[11] Y. Papadopoulos and M. Maruhn, "Model-Based Synthesis of Fault Trees from Matlab-Simulink Models," in *Proc. The International Conference on Dependable Systems and Networks*, Sweden, July 2001, pp. 72–82.

[12] M. G. Madden and P. J. Nolan, "Generation of Fault Trees from Simulated Incipient Fault Case Data," in *Proc. 9th International Conference on Applications of Artificial Intelligence in Engineering*, Pennsylvania, USA, 1994, pp. 567–574.

[13] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[14] L. M. Barlett and J. D. Andrews, "Efficient Basic Event Orderings for Binary Decision Diagrams," in *Proc. Annual Reliability and Maintainability Symposium*, January 1998, pp. 61–68.

[15] ——, "Choosing a Heuristic for the 'Fault Tree to Binary Decision Diagram' Conversion," *IEEE Transactions on Reliability*, vol. 51, No.3, pp. 344–349, September 2002.

[16] A. Barua, P. Sinha, and K. Khorasani, "On the Fault Diagnosis and Failure Analysis in the Satellite Attitude Control Subsystem," in *Proc. 8th International Conference on Space Operations (SpaceOps2004)*, Montreal, Canada, May 2004.

[17] W. J. Larson and J. R. Wertz, Eds., *Space Mission Analysis and Design*. USA: Kluwer Academic Publishers, 1999.

[18] P. C. Hughes, *Spacecraft Attitude Dynamics*. USA: John Wiley & Sons Inc., 1986.

[19] B. Bialke, "High Fidelity Mathematical Modeling of Reaction Wheel Performance," *Advances in the Astronautical Sciences*, vol. 98, pp. 483–496, 1998.

[20] A. Barua, P. Sinha, and K. Khorasani, "A Fault Tree Approach for Identifying Failure Causes in the Attitude Control Subsystem of Space Vehicles," *Submitted for Journal Publication*.