Master's Dissertation 2015

Design of Adjustable Software for Fault Detection, Isolation and Recovery of Attitude Determination and Control System in MicroDragon Satellite

Nguyen Van Thuc

(Student ID number: 81334559)

Supervisor Hidekazu Nishimura

September 2015

Graduate School of System Design and Management
Keio University
Major in System Design and Management

SUMMARY OF MASTER’S DISSERTATION

Student Identification Number	81334559	Name	Nguyen Van Thuc

Title

Design of Adjustable Software for Fault Detection, Isolation and Recovery of Attitude Determination and Control System in MicroDragon Satellite

Abstract

Attitude Determination and Control System (ADCS) is one of the most important subsystem in every satellite. Without the well operating of this subsystem, satellites missions cannot be achieved. It is very important to ensure the safety operation of ADCS and FDIR (Fault Detection, Isolation and Recovery) is usually chosen to do that. However, according to statistical numbers of satellites in-orbit failures in the past, ADCS is the subsystem which has highest probability of failures, 32%. Therefore, the need of improving FDIR for this subsystem is very critical.

This research aims to propose a novel FDIR mechanism for ADCS of MicroDragon (MDG) satellite – the first Vietnamese micro satellite is being developed in Japan. The FDIR is proposed by using and adjustable software supported by an FIR (Fault Isolation and Recovery) library. The library is a text file in which, each line starts with a fault ID for all the predicted failure modes of ADCS, and is followed by the combinations of correspondent isolation and recovery functions ID. When a predicted fault is detected, the software searches for isolation and recovery functions ID in FIR library and execute the required functions. On the other hand, when ADCS faces with an unpredicted fault or there is need of software optimizations, operators on ground station send uplink data to update FIR library and adjust FDIR software. New combinations of isolation and recovery functions can be created by adding or removing function ID to existing lines or new lines can be added to FIR library by operators on the ground. Once FIR library is updated, software for FDIR of ADCS can be changed accordingly. The design of adjustable software for FDIR of ADCS is promising to shorten the time of fault recovery and process of handling in-orbit failures of ADCS becomes more convenient for operators.

Keyword (5 words)

Adjustable software; ADCS; FDIR; FIR library; MicroDragon Satellite;

Acknowledgements

This master's thesis could not have been finished without supports from some very important and special people. I would like to extend my sincere gratitude to all of them.

Firstly, I would like to thank my supervisor, Prof. Nishimura, for his guidance during 2 years of my master course in SDM. Without his unlimited support, this work would not have been completed.

Secondly, I am thankful to Prof. Shirasaka, Prof. Ioki and other professors in SDM. They have been following and giving me a lot of advises to improve my research.

Last but not least, I must thank to all of my colleagues, my friends in MDG project and my family for believing in me and supporting me during the time I studied in Japan.

Table of Contents

List of figures

Chapter 1. Introduction

1.1 MicroDragon (MDG) Satellite Background and Challenges

MicroDragon satellite project started in October 2014 and is planned to finish in the mid-year of 2017. This three years satellite project is founded by Vietnam National Satellite Center (VNSC) as a sub-contract in the big project to build Vietnam Space Center – a collaboration project between Vietnamese and Japanese Governments. Until the time of completion of this master dissertation, 22 researchers from VNSC have been sent to Japan to work on the project. At the same time, they also need to join master courses at 5 universities in Japan, include Keio University, The University of Tokyo, Tohoku University, Hokkaido University and Kyushu Institute of Technology. Scope of this project is mainly focused on space technology education for VNSC students, especially in micro satellite development process resulted in the first Vietnamese micro satellite named MicroDragon (MDG).

MDG satellite is one of the typical micro satellite in 50 cm cubic shape and 50 kg in total mass. The satellite is planned to be launched in 2018 as one of piggyback payloads in HII-A vehicle. Outer view of this satellite is shown in Figure 1.1. In term of system design, MDG satellite consists of seven subsystems, and among them, there is an important subsystem called ADCS (Attitude Determination and Control System). The main functions of this subsystem are to determine the orientation of the satellite and re-orient the satellite to the target direction.

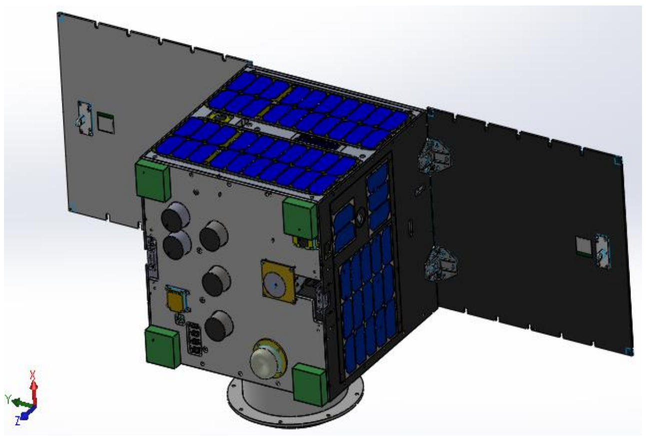


Figure 1.1 MicroDragon satellite in 3D model

As in every satellites, ADCS plays a very important role in ensuring the success of mission in MDG satellite. In this satellite, cameras are required to always point to the Earth to take pictures, solar panels are also needed to point to the Sun to generate power from sunlight and distribute to all satellite components. They are all controlled by ADCS. Without the successful operation of ADCS, satellite mission cannot be achieved.

Safety operation of ADCS in orbit is very significant. However, this subsystem is also well known to have the highest possibility of failures in orbit. In Ref. [1], ADCS is stated as the most sensible subsystem in satellite, and failures of this subsystem are the most critical. Ref. [2] proved those statements by statistic numbers. In Ref. [2], Mak Tafazoli conducted a study on 156 failures which occurred on 129 spacecraft from 1980 to 2005. Failures are categorized in several categories such as spacecraft size and mass, failures of subsystems, and time of failures after launch. A summary of failures in term of satellite subsystems is illustrated in Figure 1.2. As one can see clearly in this figure, ADCS is the most sensible subsystem which caused 32% of failures. Figure 1.3 shows the impact of ADCS failures to mission of satellites. Among those 156 failures, almost 60% of them contributed to degradation of mission and more critically, 30% of them led to the loss of whole mission. More failures effects can be found in Ref. [3]. Those statistical numbers are quite huge and, therefore, they are worth of considerations whenever satellite systems or subsystems are going to be developed.

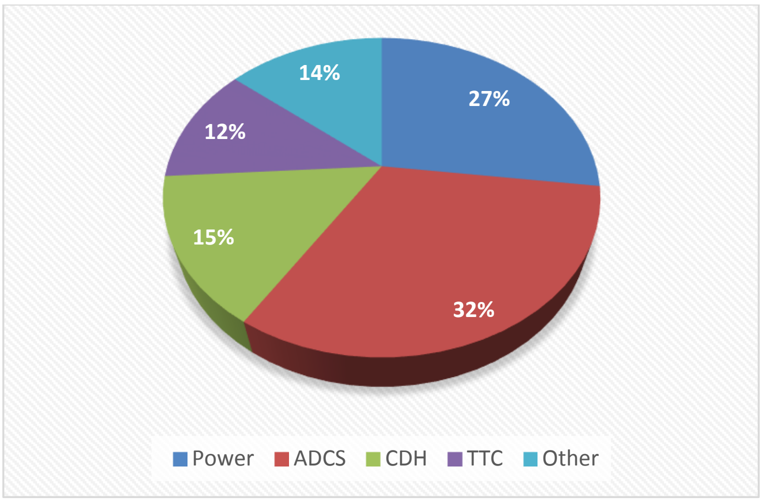


Figure 1.2 Failures in spacecraft subsystems

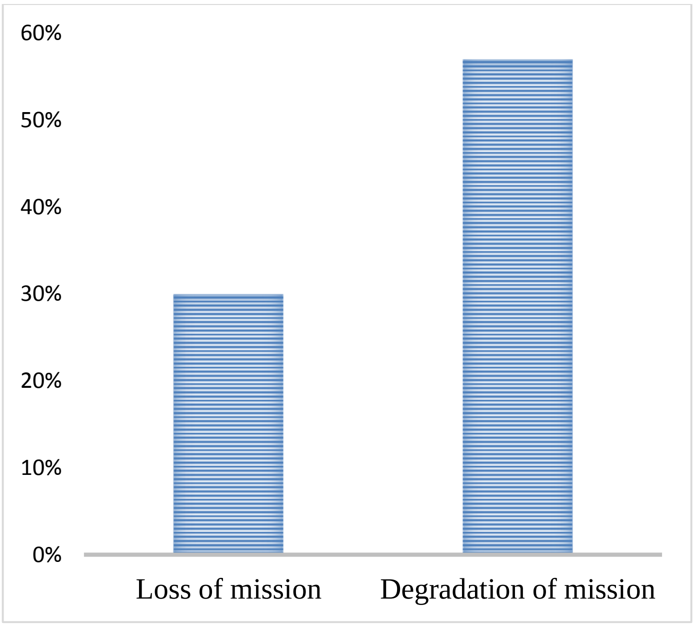


Figure 1.3 Impact of ADCS failures on mission of satellites

Since early phase of development process, the design of MDG satellite, except for payloads, has been decided to follow that of UNIFORM-1 satellite, one of the Japanese micro satellites launched in May 2014 [4]. All hardware components of MDG satellite, including ADCS, are identical with those in UNIFORM-1. The software needs to be developed by students. Using the same set of hardware with UNIFORM-1 satellite was expected to provide some benefits to the design of MDG because they are all space proven. However, actual operation of UNIFORM-1 shows that, after several weeks of operating in orbit, the satellite has faced failures in ADCS. It took several weeks to find out the causes of failures. Then, the problems are fixed by a set temporary solutions and they accepted to operate the satellite with degraded performance. This situation yield the concerns in ADCS team about the safety, and reliability operation of this subsystem when satellite is in orbit. There is necessity of an effective way to deal with failures of ADCS even when the satellite is already launched to space.

Learning from failures in ADCS of UNIFORM-1 and other satellites in the past (which mentioned above), the need of ensuring safety operation for this subsystem become more significant. Fault Detection, Isolation and Recovery (FDIR) is often used to carry out this job, some of them can be found in Ref. [1] [5] [6] [7]. In general, FDIR has functions to detect faults/ failures when they occurred, isolate them from propagations then recover the system operation.

In almost designs of ADCS as well as satellite system, when onboard FDIR failed to maintain satellite operation, the common way for satellite operators or engineers on ground to handle satellite in-orbit failures is by reprogramming for onboard computer (OBC) [8]. Example of this technique is explained in design of ETS-VIII Satellite [9], MDG satellite also has this function. The advantage of this technique is that onboard software can be fully renewed or modified to adapt to new situation of satellite in orbit. However, for micro satellites which usually come with very limited uplink speed (in MDG it is 4 kilobit per second), the process of uploading new code for reprogramming purpose could take a long time. It is not taken into account the satellite attitude error which defines antennas direction. When satellite attitude is lost due to failures of ADCS, that process can be worse. The time spending on reprogramming for OBC could be much longer. Therefore, operators on ground station needs to have more options when dealing with failures which occurred on ADCS in orbit.

1.2 Research Objective and Approach

Failures which occur on ADCS in orbit are critical and it is very important to have an effective way to handle those failures from ground station. This research aims to propose an FDIR mechanism which can provide operators on ground more options when dealing with in-orbit failures of ADCS. The FDIR mechanism of ADCS in MDG satellite is proposed by using an adjustable software supported by FIR (fault isolation and recovery) library.

Adjustable software is the software which can be adjusted during system operation. If onboard software is adjustable, satellite operators on ground station can add, remove software functions from execution or they can change orders (or sequence) of functions execution in the software, even when satellite has already launched into orbit. The main objective of introducing adjustability to the software is to gain the flexibility of FDIR which, as suggested in Ref. [2] is one of the keys to gain system safety as well as increase availability.

Having adjustable software for FDIR of ADCS, operators and engineers on ground can have more options or freedoms when dealing with ADCS failures in the future operation of satellite in orbit. In proposed FDIR mechanism, those options can be achieved by using the existing isolation and recovery functions in different orders which are represented by the combinations of fault ID, isolation, recovery function ID in FIR library.

In adjustable software for FDIR of ADCS, isolation and recovery actions are prepared as software functions or subroutines. Each of those functions or subroutines then be assigned with an ID and stored in a text file called FIR library corresponding to the predicted faults. Each fault also has a fault ID which stands at the beginning of the line in FIR library and followed by isolation, recovery function IDs. The text file then be stored in onboard memory of Onboard Computer before the satellite is launched to orbit so that FDIR software can access to find isolation, recovery instruction every time when a fault is detected. For the predicted failure modes of ADCS, the software as the part of FDIR mechanism can automatically detect then isolate the faults (or causes of failures) and recover ADCS by following instructions stored in FIR library, operation of ADCS can be maintained as desired.

In situation when a fault occurred but there is no appropriate instructions available in FIR library, or when an un-predicted failure occurred, the software then activates ADCS Safe Mode and waits for ground station interventions. At this point, operators or engineers on ground make adjustments to the software by sending uplink data to update FIR library based on results of failure analyses on the ground. New order or combination of isolation and recovery functions will be created in FIR library so that they can be performed onboard by the adjustable software. There is no needs of reprogramming for onboard computer even in these situations.

Adjustable software is also helpful in the cases when people on ground station found that a specific fault is actually not a fault anymore due to degradations of components; or in the situations when there is something needed to be modified to improve isolation, recovery process.

1.3 Dissertation Outline

This dissertation includes six chapter and followed by list of references and appendix.

Chapter 1 provides the overview of this dissertation. In this chapter, background of this research is explained as the introduction to MDG satellite and design challenges. Research objectives and approach are also specified in second part of this chapter.

Chapter 2 discusses about Fault Detection, Isolation and Recovery techniques and the applications of FDIR in satellite. Some software approaches for FDIR as well as satellite safety are also being reviewed in this chapter as the related works.

Chapter 3 then gives overview of MDG satellite system focused on Attitude Determination and Control system (ADCS). Several failure modes of ADCS are also identified in the contents of this chapter using Fault Tree Analysis (FTA) approach.

Chapter 4 is where the proposed FDIR mechanism for ADCS of MDG satellite is explained in detail. Then, the design of adjustable software for FDIR of ADCS is illustrated using process of model-based system engineering (MBSE) with supported by SysML (System Modeling Language).

Chapter 5 then shows the process of verification and validation. The design of FDIR software first is verified to be adjustable by updating FIR library. Then how the adjustability of software can support for FDIR of ADCS is demonstrated using a hardware in the loop simulation (HILS) with several failure scenarios. Validation of the designed software is done by evaluation from experts' opinions.

Finally, in chapter 6, results of this research are summarized for conclusion and followed by future works.

Chapter 2. Fault Detection, Isolation and Recovery

2.1 Fault Detection, Isolation and Recovery techniques

2.1.1 Overview

Fault detection, isolation and recovery (FDIR) is an important aspect in control engineering. The primary objective of an FDIR system is early detection of faults, isolation of their location, and then, enabling correction actions before additional consequences of faults apply to system operation and lead to the loss of missions.

In ISO/CD 10303-226, a fault is defined as an abnormal condition or defect at the component, equipment, or sub-system level which may lead to a failure. A failure is a permanent interruption of a system's ability to perform a required function under specified operating conditions. In each organization or each specific system, the definition of fault and failure may change a little compared to the standard. However, in general, a fault can be understand as the cause and failure is effect. Fault can occur in the individual unit of the sensors, actuators, or other devices and has effect to the unit, subsystem or overall behavior of the system. There are several types of faults and each of them affects on the system behavior in different ways:

- ✧ Abrupt fault: the sudden change in behavior of components. It could be the change in sensors output, actuator performance and so on.
- ✧ Incipient fault: a fault that not always present, it occurs occasionally in the system.
- ✧ Permanent fault: the fault that leads to total failure of a component, one they occurred, they do not disappear.
- ✧ Transient fault: is the temporary malfunction of components, it occurs in certain time duration then disappears.
- ✧ Intermittent fault: is the repeated occurrence of transition faults. It can occur in fix or variable time intervals.

- ✧ Hidden fault: is a type of fault that presents in components and visible only when components are activated.

There are several requirements (desired characteristics) for FDIR of a system, such as requirements for detection delay, sensitivity, and performance. The delay time for detect a fault which already occurred should be minimized to reduce the consequences as much as possible. Critical failures can be prevented if the faults (or causes) are detected early. Sensitivity, on the other hand, is the term to illustrate ability of FDIR mechanism to detect faults with high detection rate (ratio between occurrence and detected faults). A robust FDIR mechanism also has ability to distinguish noises, disturbances and faults. This is the meaning of FDIR performance which is also represented in term of false alarm rate (percentage of detect wrong faults when they actually not happened).

In process of FDIR, once a fault is detected the next step is to identify location of fault and isolate the fault from propagation to prevent further consequences. For critical system where safety and availability are the most important aspects, recovery is usually done by switching to redundant components, or systems. If there are two or more identical components in the system, faulty components can be ignored and the system directly switch to the spare components when the primary ones get failed. In such kind of system, 100% functionality of system can be recovered. For other systems, where there is no redundancy, recovery can be achieved by using alternative paths or functional degradations recovery [10].

2.1.2 Techniques for FDIR

Numerous of FDIR methods have been developed and applied since 1970 [11] [12]. In this section, overview of some common methods is introduced.

2.1.2.1 Fault - Detection

Figure 2.1 shows the decomposition of fault detection methods. There are two main categories in fault detection techniques: Detection with single signal and detection with multiple signals.

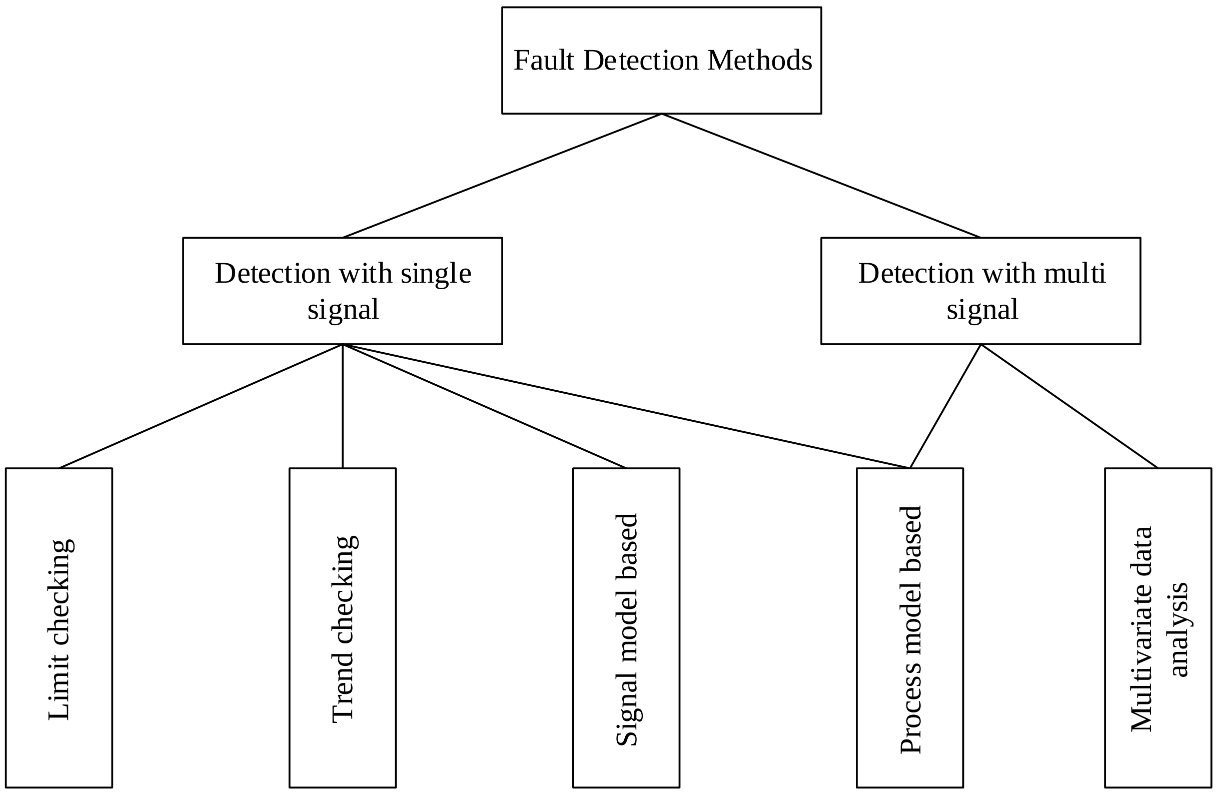


Figure 2.1 Fault – detection methods

In the category of detection with single signal, limit checking is the first one. This fault detection method is described in detailed in Ref. [13]. Two limit value called thresholds are set for purpose of fault detection: maximum value Y_{max} and minimum value Y_{min}. A normal status of the measured value, Y(t), is detected when is falls into this interval [Y_{min}; Y_{max}]. Otherwise, a fault is detected.

The principle of trend checking is very similar to limit checking. However, instead of directly use the measured value Y(t) and its thresholds, the derivation values are considered. A fault is detected when:

$$\dot{Y}(t) \notin [\dot{Y}_{\min};\dot{Y}_{\max}]$$

Signal model based and process model based are known as model based approach for fault detection and isolation, they are also described in very detailed in Ref. [13]. The general idea of those techniques is, based on dynamics model of the system, the detection mechanism calculate or estimate the future values of measured values to generate residual. The generated residuals then be used to detect faults of system or components by comparing with a fixed or adaptive thresholds. There are various types of model based fault detection method such as least square, extended least square, recursive estimators, parameter estimators, parity equations, state observer, etc.

Multivariate data analysis is a method for fault detection which mainly based on principal component analysis. Principal component analysis (CPA) is powerful fault detection and isolation method which have been used for very long time, especially for large scale system such as chemistry plant [14]. By projecting the original historical data onto a lower dimensionality space, PCA reduces the dimensionality it to simplify the system. It gets the principal causes of variability in a process to detect fault. Whenever some of these causes are changed, they can due to a fault in the process, fault alarm is triggered.

2.1.2.2 Fault – Isolation

As mentioned in above section, once fault is detected, the cause needs to be identified and isolated from system, operation. This process can be done automatically of manually depends on characteristics of the system.

Several methods for fault - isolation are already explained in above section when detection methods are describing. There are still some that needed to be discussed in this section and inference methods are the most common among them. Several methods in the category of inference are listed in Ref. [13], including Fault Trees, Approximate reasoning and Hybrid neuro-fuzzy systems. Each of them have its own advantages and disadvantages.

Fault Trees is a graphical tool to visualize the relationships in reliability and diagnosis. Each fault tree consists of events and gates connected with lines. The relationships are used in fault trees are binary relationships (mostly AND, OR) which along with hierarchical structure supports for human comprehension. Fault trees are usually used since the early phase of system design to identify the critical components, subsystems that contribute the most to system failure probability. In this research, fault trees are also used for failure prediction of ADCS (more details in section 3.2.2).

2.1.2.3 Fault – Recovery

In general, there are three types of recovery [10]:

- ✧ 100% functional recovery by using redundancy
- ✧ Functional recovery using alternatives paths
- ✧ Degraded functional recovery

Redundancy is the most effective way to deal with failures. Two or more identical components are implemented in the system, and when one get failed, the system will switch to the redundant one. There are three levels of redundancy [15]: hot redundancy, warm redundancy, and cold redundancy. For very critical system which must not go down for even a brief moment under any circumstance, hot redundancy is used. In systems that are using hot redundancy, the spare and the primary components work in parallel. The spare will continue to work when the primary one fails. Warm redundancy is similar, however, the additional components run in idle and will activated to take over the functions of the primary one when it fails. Cold redundancy, in the other hand, the redundant components are put into operation only when the primary one get failed. Depends on level of critical as well as requirements of the system, system designer choose one of the level or combination of those level of redundancy.

In most system, resources are very limited. Therefore, instead of using redundant components, units, an alternative path is used in case of failures. For example, once al the main actuators are failed, the system needs to use the sub-actuators. However, due to the differences in performance of the alternative which is usually lower, system performance cannot be fully recovered.

In the worst case when there is no redundant component available, to keep the system still in operation, operators have to decide to turn off/ or disable several functions of the system. For example, when some of solar array got damage, three – axis attitude controlled using reaction wheels is disabled and replaced by using magnetic torques with lower control accuracy.

2.2 Overview of FDIR in satellite

For such a system as satellites, safety, reliability, and availability are very important aspects. For almost satellite mission, once they are launched into orbits, there is no way to get back and repair. On ground testing and design of safety assurance are considered as critical parts in satellite development process. In term of FDIR, depends on satellite missions and architecture, each satellite has its own FDIR mechanism. In general, FDIR in satellites is implemented using both hardware and software and can be decomposed into several levels [16] [17] [18]. Some common levels of FDIR are explained following:

- ✧ Level 0 is the lowest level which handles failures entirely on unit level. Usually, FDIR in this level is ensured with the uses of components by default. Hardware component itself has functions of auto correction and protection. For example, a power distribution unit which supply electrical power to all components in satellite must have function to protect those components from electrical related hazards, such as over voltage, over current, etc. Since hardware components in satellites are usually COTS products, FDIR in this level is ensured by manufacturers.
- ✧ Level 1 covers failures being handled by subsystem onboard software. Failures are being handled within satellite subsystem and there is no need of interaction with other subsystem or higher level satellite system.
- ✧ Level 2 covers failures being handled by satellite onboard software. When subsystems cannot handle failures which occurred on them, or failures are severe and affected on other subsystem as well as the whole satellite, FDIR at level 2 will be triggered, satellite onboard software will take appropriate actions as predefined before launch to execute onboard.
- ✧ Level 3 comprises failures which need to be handled by hardware reconfiguration. As mentioned above in section 2.1.2, redundant units are implemented in satellites. Once

failures occurred on the primary components, redundant ones will be triggered. This can be done by software or hardware itself.

- ✧ Level 4 comprises failures which cannot be handled onboard and required ground station interventions. These failures include unpredicted failures modes, critical failures which affect overall satellite system. Failures analyses are done on the ground from telemetry data received from satellite, then appropriate isolation and recovery actions will be made by uplink commands. Reprogramming, as mentioned in section 1.2, is one of the recovery actions in this FDIR level [8] [9].

It is also important to design FDIR at each level so that failures cannot be isolated or recovered at that level can be transferred to the next higher level. Besides, safe mode is also needed to be defined. In case when on board FDIR mechanism cannot handle failures, safe mode should be automatically activated. Satellite system can be safe in safe mode with minimum function as possible. Power consumption in safe mode should be as low as possible so that satellite can save enough power until receive command actions from ground station. Communication between satellite and ground station also should be kept. Receiver must be turned on at any time so that ground station can send uplink command every time satellite flies above. Also, telemetry that consists of satellite system health should be transmitted to ground as much as possible to help the process of failures analysis from ground station become more effective.

For ADCS – the most sensible subsystem in satellite, the need of a robust, effective FDIR mechanism is critical. Many studies have been conducted in order to improve FDIR of this subsystem. Some can be found in [19] [20] [21]. Each of them focuses on some particular failures in ADCS such as failures of sensors (for example Gyroscope, IMU) or actuators (reaction wheels, momentum wheels). Various methods or techniques have been also applied to FDIR of ADCS and among them, model-based approaches which mentioned in section 2.1.2 are the most common one.

In the most design of big satellites where availability is the most important consideration, ADCS usually includes one or more redundant units, so that services that they provide are not interrupted even when failures occurred. In smaller satellites such as micro, nano satellites, size and weight are very limited. Having redundancy for every component is impossible. That is reason why only moving parts such as reaction wheels are usually considered as the most dangerous parts with higher probability of failures and are prepared with redundant one. Four reaction wheels are usually used instead of three, even size and mass are larger than other components. In such a system as micro satellite, more complicated FDIR functions, in the other hand, are usually implemented in software. Software has functions to detect, isolate faults and apply recovery actions to maintain ADCS performance. It makes software become heavier, more complicated and sometimes introduces more failures. Therefore, having a robust design software architecture for ADCS in general and FDIR in particular is very important.

Chapter 3. Attitude Determination and Control System (ADCS) in MDG Satellite

3.1 Overview of MDG satellite

3.1.1 Missions of MDG satelltie

Ocean color observation to assess coastal water quality and locate living resources is the main mission of MicroDragon satellite. The satellite will provide images data which can be used by researchers and scientists in fishery and oceanography fields to analyze then distribute necessary information to fishermen and environmental managers.

In addition to the main mission, there is a sub-mission which called Store and Forward (S&F) mission. Satellite will receive information transmitted from sensors modules located on surface of the sea and store those data onboard. The received data then will be transmitted to ground station when satellite is passing over ground station. The data provided in this sub-mission will be used to give additional information including assessments of chlorophyll and algae class to the main mission data.

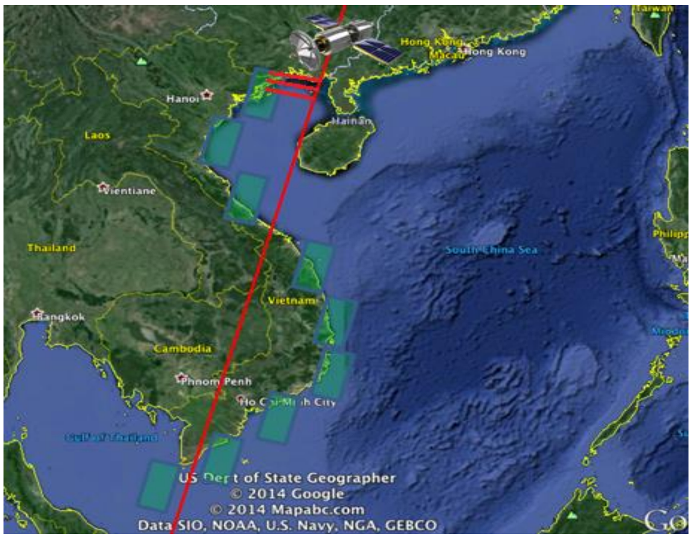


Figure 3.1 Main mission of MDG satellite [22]

© MicroDragon Development Team

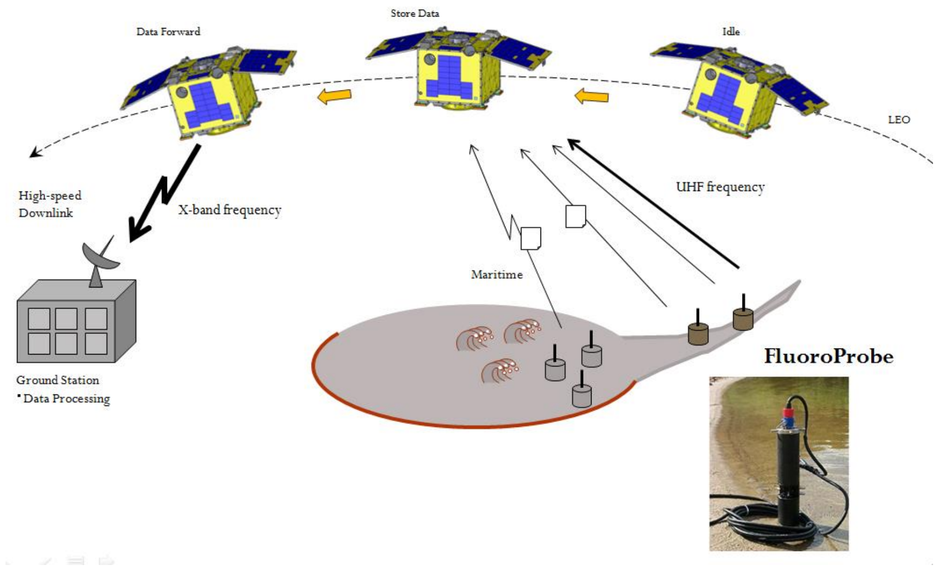


Figure 3.2 Store and Forward mission [22]

© MicroDragon Development Team

3.1.2 Overview of MDG system design

The MDG satellite consists of six sub-systems as shown in Figure 3.3 and the supporting structure. Electrical Power Subsystem (EPS) has the functions to supply power for all components in satellite using the generated power from sunlight. Thermal subsystem is utilized to ensure the temperature of components within the acceptable ranges. Communication (COM) subsystem which consists of an X-band transmitter and an S-band transceiver has the functions to provide the communication links between satellite and ground station so that mission data can be downlinked to ground and operator on ground can send command to satellite from ground station. Command and Data handling (C&DH) subsystem then has responsibility to handle those commands and manages the whole satellite system as well as data generated from satellite subsystems. Payloads are the most important subsystem which consists of three cameras and one onboard computer called SHU (Science data Handling Unit). SHU plays the role as the brain of payloads. It has functions to control cameras to take pictures, store images and send those image to ground when it is commanded.

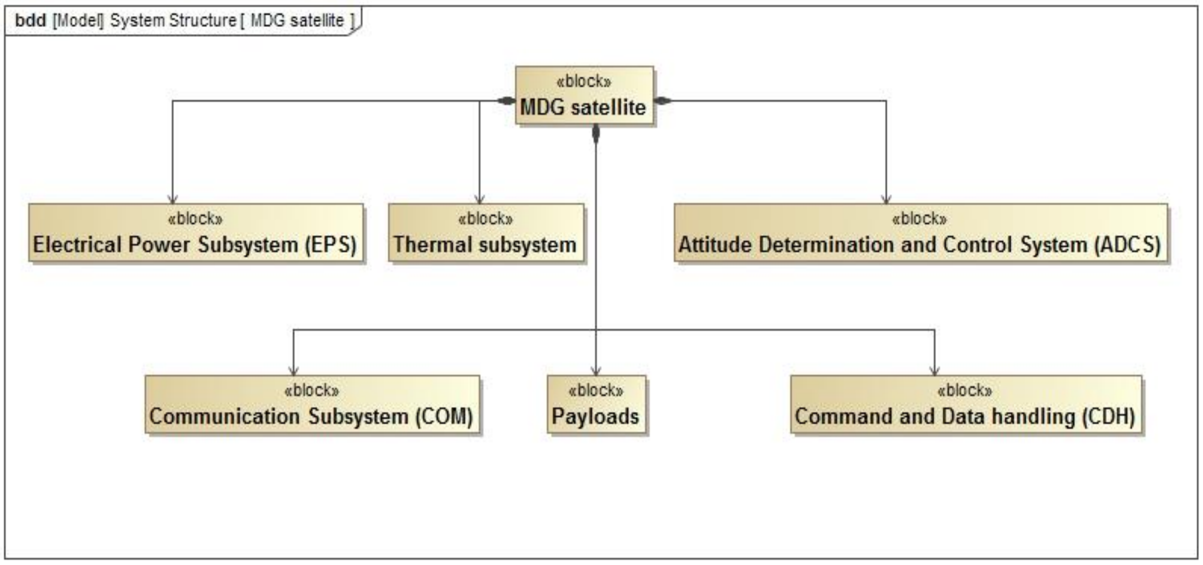

bdd [Model] System Structure [MDG satellite]
«block» MDG satellite
«block» Electrical Power Subsystem (EPS)
«block» Thermal subsystem
«block» Attitude Determination and Control System (ADCS)
«block» Communication Subsystem (COM)
«block» Payloads
«block» Command and Data handling (CDH)

Figure 3.3 Subsystems in MDG satellite

This research is focused on Attitude Determination and Control subsystem (ADCS), which is one of the most important supporting subsystems. ADCS controls satellite attitude corresponding to requirements from payloads as well as other subsystems. Payloads may have requirements to ADCS to point their lenses to certain target on the Earth and keep them stable when they are taking pictures. EPS may require ADCS to point its solar panels to the Sun to generate more power. COM also requires to direction of antenna to be faced to the Earth, so that communication with ground station can be ensured.

3.2 Attitude Determination and Control System

3.2.1 ADCS overview

3.2.1.1 ADCS definition and design drivers

As shown in Figure 3.3, ADCS is one of the most important subsystems in MDG satellite, which consists of various ranges of sensors and actuators. The main functions of ADCS are listed as following:

- ✧ Acquire data from attitude sensors

- ✧ Process sensors data to estimate the current attitude of satellite
- ✧ Compute the deviation of current attitude from target attitude
- ✧ Compute a control torque need to be applied to satellite to achieve target attitude
- ✧ Control actuators to apply control torque to the satellite

Practices show that attitude system design is an iterative process. Start from mission requirements to the design of ADCS control modes then identification of control algorithms for each mode and selections of sensors, actuators to satisfy the requirements. Figure 3.4 describes the impacts of mission requirements and subsystems on the design of ADCS in general. For MDG case, the main mission of this satellite is to observe Vietnam coastal sea to get assessments of water quality and also finding potential sources of fish to support to fisher mans and people in aquaculture filed. This specific mission requires satellite to have ability to always

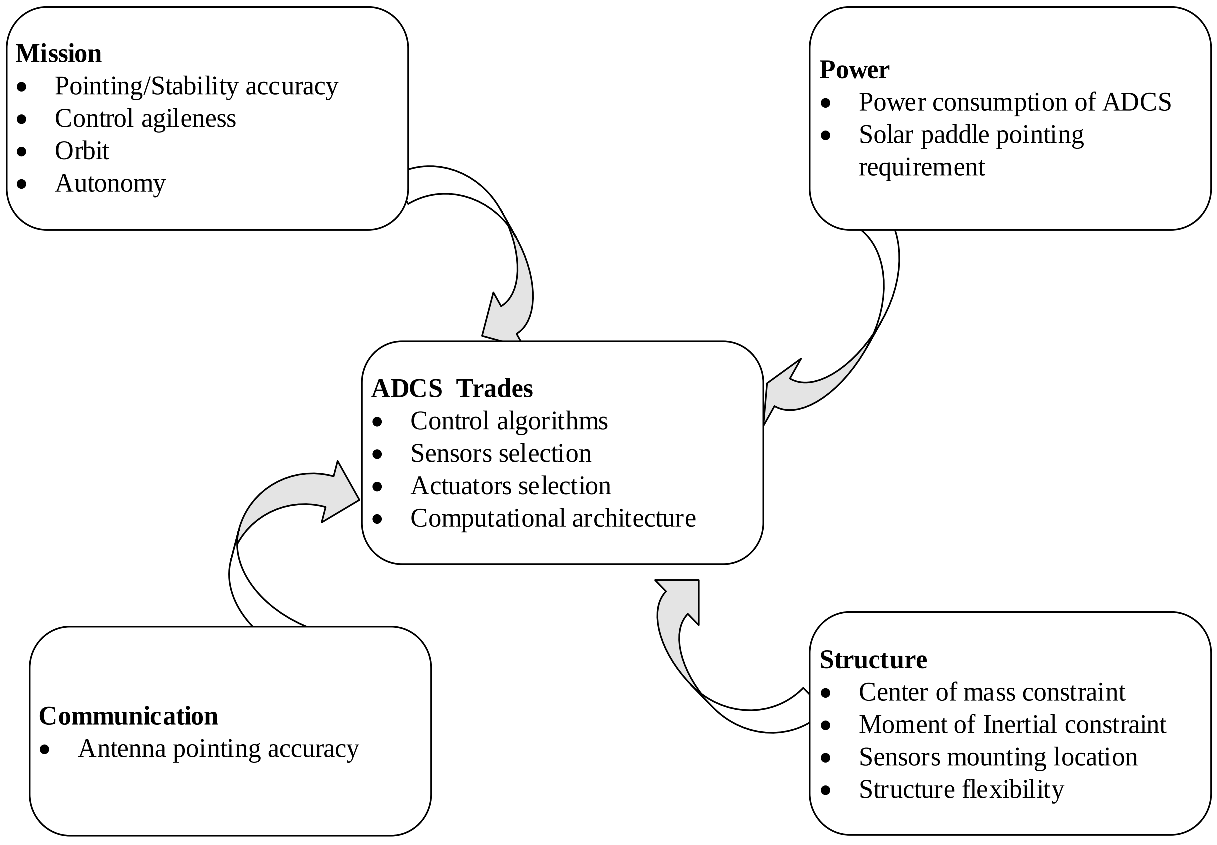

Mission
Pointing/Stability accuracy
Control agileness
Orbit
Autonomy
Power
Power consumption of ADCS
Solar paddle pointing requirement
ADCS Trades
Control algorithms
Sensors selection
Actuators selection
Computational architecture
Communication
Antenna pointing accuracy
Structure
Center of mass constraint
Moment of Inertial constraint
Sensors mounting location
Structure flexibility

Figure 3.4 Impacts of mission requirements and other sub-systems on ADCS

point mission cameras to nadir direction and/or to certain area of interest with very high pointing and stability accuracy. The mission also requires very agile pointing speed so that target areas can be changed in very short time and more images can be captured.

Design of ADCS also depends on other subsystems such as power, communication and satellite structure. From Electrical Power Subsystem (EPS), requirements are listed for ADCS as power consumption of all ADCS components and more importantly is the assurance of power generation from solar cells as the direction of solar panels corresponding to the Sun. Communication subsystem (COM) also gives requirement to ADCS to ensure the communication link between satellite and ground station, especially when satellite is facing with anomalies. Last but not least, ADCS design is impacted a lot by the satellite structure in term of satellite center of mass, moment of inertia (MOI), sensor mounting location and structure flexibility like deployable solar panels. Those kind of considerations need to be taken care during the design and development phase of ADCS as well as the whole satellite system.

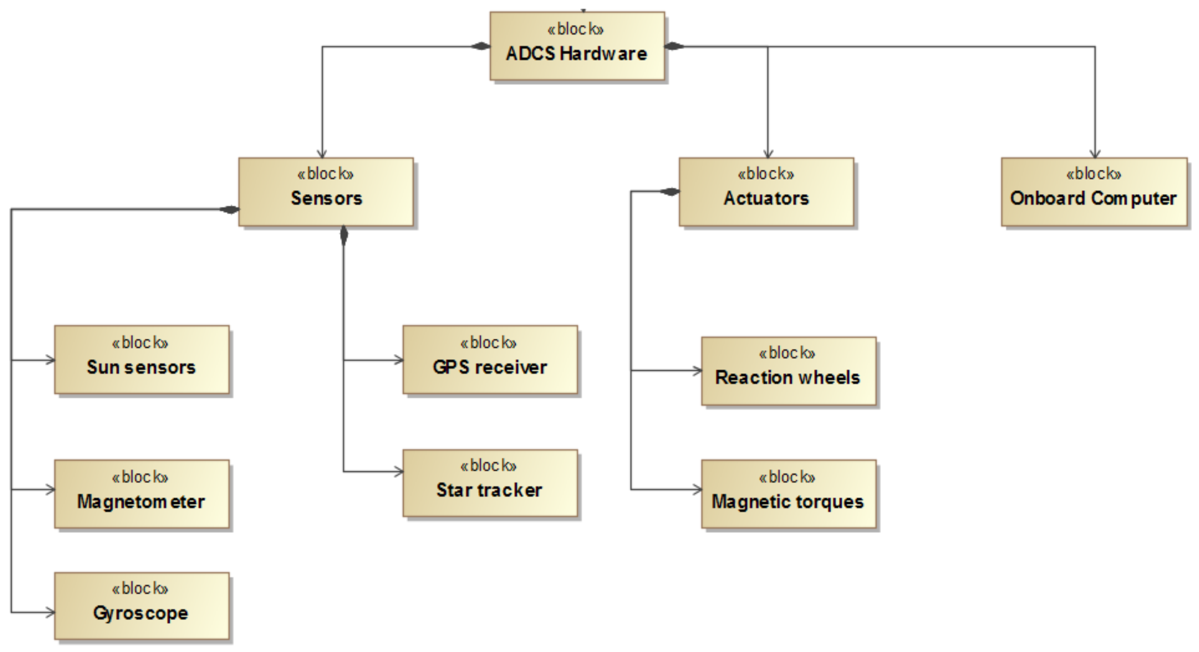


Figure 3.5 Hardware components of ADCS

3.2.1.2 ADCS hardware

The bus system of MDG satellite is decided to be identical with UNIFORM-1 satellite (more information can be found in [4]), and ADCS is a part of it. Figure 3.5 shows the hardware decomposition of ADCS hardware components. The interconnections between those components are shown in Figure 3.6.

ADCS consists of an Onboard Computer (OBC), sensors and actuators. Sensors include six sun sensors (NSAS), one 3-axis magnetometer (GAS), one 3-axis gyroscope (FOG), one GPS receiver (GPSR) and one star tracker (STT). Actuators include four reaction wheels (RWs) and three magnetic torques (MTQs). Different from UNIFORM-1 satellite, in MDG, there is only one OBC. That means ADCS has to share this computer with other subsystem such as Command & Data handling, Thermal, etc. A summary of all components specifications is shown in Table 3.1.

Table 3.1 Specification of ADCS components

Name	Weight &Size	Power	Specs
NSAS	46g, 31x50x22mm	5V, 150mW	RS422, ~1 deg accuracy
GAS	320g 95x95x45mm	5V, 30mA	Analog out +/- 100000nT range 4000nT/V
FOG	1kg 135x150x45mm	28V 120mA	RS422 and Analog +/- 5deg range 400mV/deg/s, <0.01de/s noise
STT	510g, 147x80x77mm	28V, 2.5W	RS422, 2Hz output Accuracy: 30 arcsec (Y, Z) 0.04 deg (X)
GPSR	400g 98x98x22mm	5V 180mA	RS422, 1Hz
RW	1.1kg 120x120x64mm	28V(motor), 5V(controller	RS422, >0.003Nm @4000rpm,
MTQ	375g 176x54x47mm	5V 70mA	5AM^2 +/- 20%

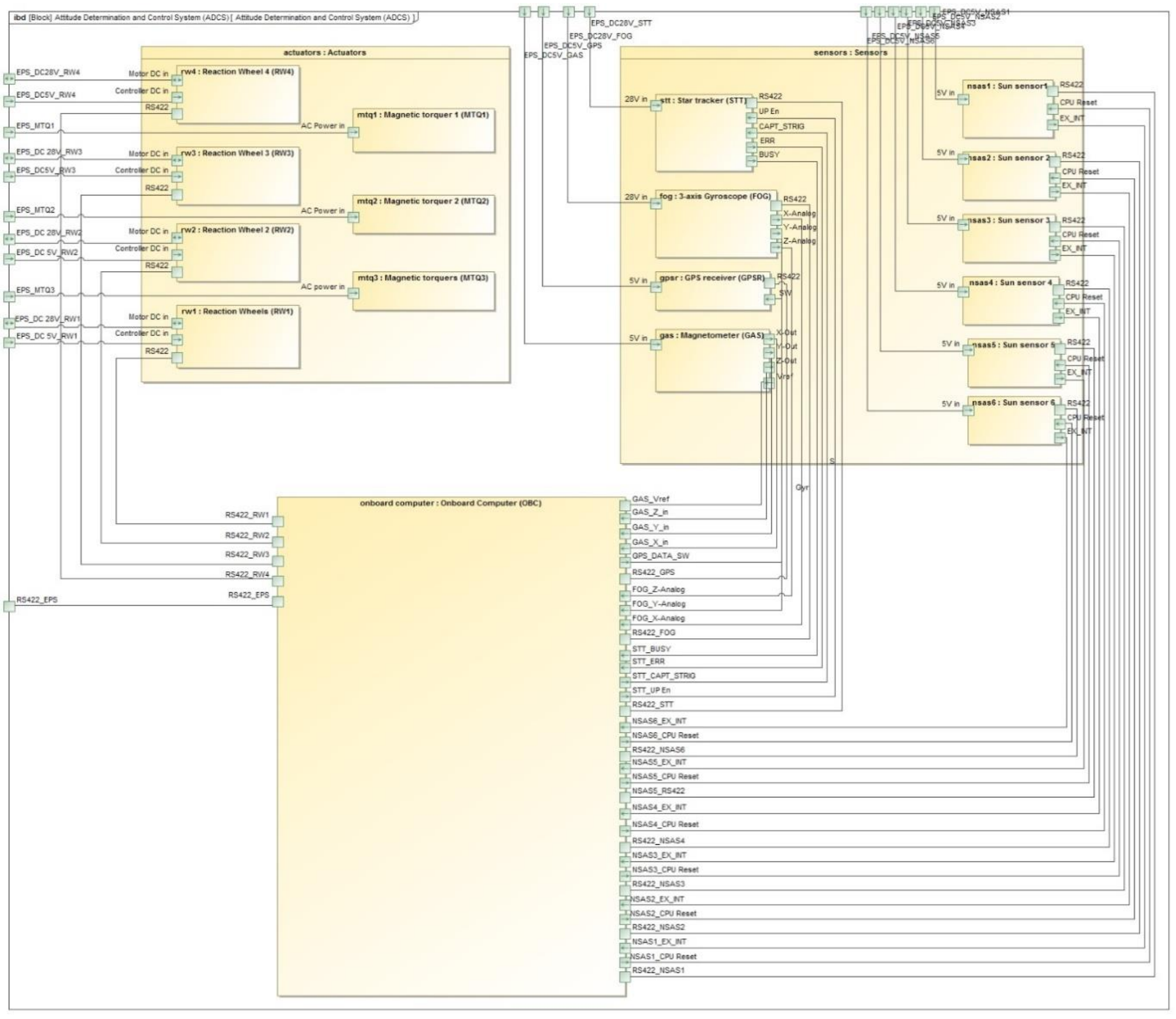


Figure 3.6 Internal block diagram of ADCS

3.2.1.3 *Attitude Control Modes in MDG satellite*

As mentioned above in section 3.2.1.1, attitude control modes are derived from mission requirements as well as requirements from other subsystems. For each specific set of requirements, an attitude control mode needs to be carefully defined. Then, hardware components and control algorithms to be used in each mode are selected. However, for ADCS of MDG satellite, all components are already decided to follow UNIFORM -1. These pre-defined components (shown in Figure 3.5) are beneficial in term of development time and components reliability since they are space proven products. As a result, however, design degree of freedom is reduced. There is constrain that all the modes and corresponding functions of ADCS have to be realized by that set of components.

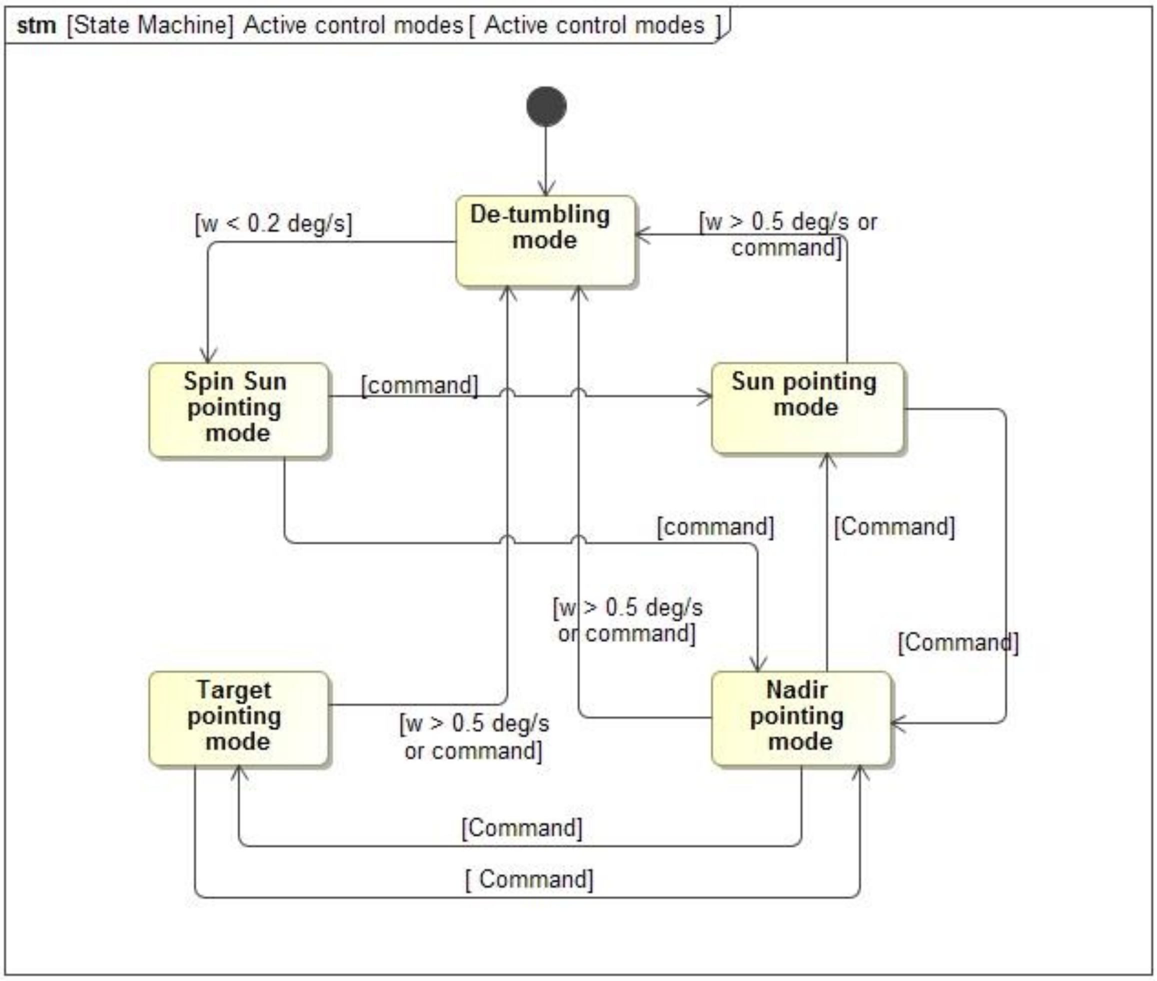


Figure 3.7 State machine diagram for Attitude control modes of MDG

Figure 3.7 describes the modes definitions of ADCS of MDG satellite. There are five active control modes including De-tumbling Mode, Spin Sun Pointing Mode (SSPM), Sun Pointing Mode (SPM), Nadir Pointing Mode (NPM), Target Pointing Mode (TPM) and one additional mode defined as ADCS Safe Mode when all the components of ADCS are turned off. The transitions of modes are defined as:

- ✧ De-tumbling mode is initiated automatically whenever satellite rotational rate is larger than 0.5 degree per second (except when ADCS is in Safe Mode). This mode is also can be activated by command received from ground station.

- ✧ Four other active control modes will be initiated according to the current operation mode of satellite system. Mode transitions also can be triggered by command received from ground station.
- ✧ Safe Mode can be triggered from any other mode whenever ADCS encounters emergency.

Safe Mode is the mode when all of ADCS components are turned off. Satellite initiates this mode when it is first separated from rocket because power consumption is needed to be kept as low as possible. Safe Mode is also activated when ADCS faces the failures that cannot be recovered on board. Mode transitions are designed in a way such that ADCS can automatically switch to Safe Mode from any other control mode in cases of emergency. During Safe Mode, OBC is still in operation. That means ADCS can still transmit telemetry data to ground every time the satellite pass over ground station so that operators can perform failure analyses then send appropriate commands to OBC to execute FIR actions.

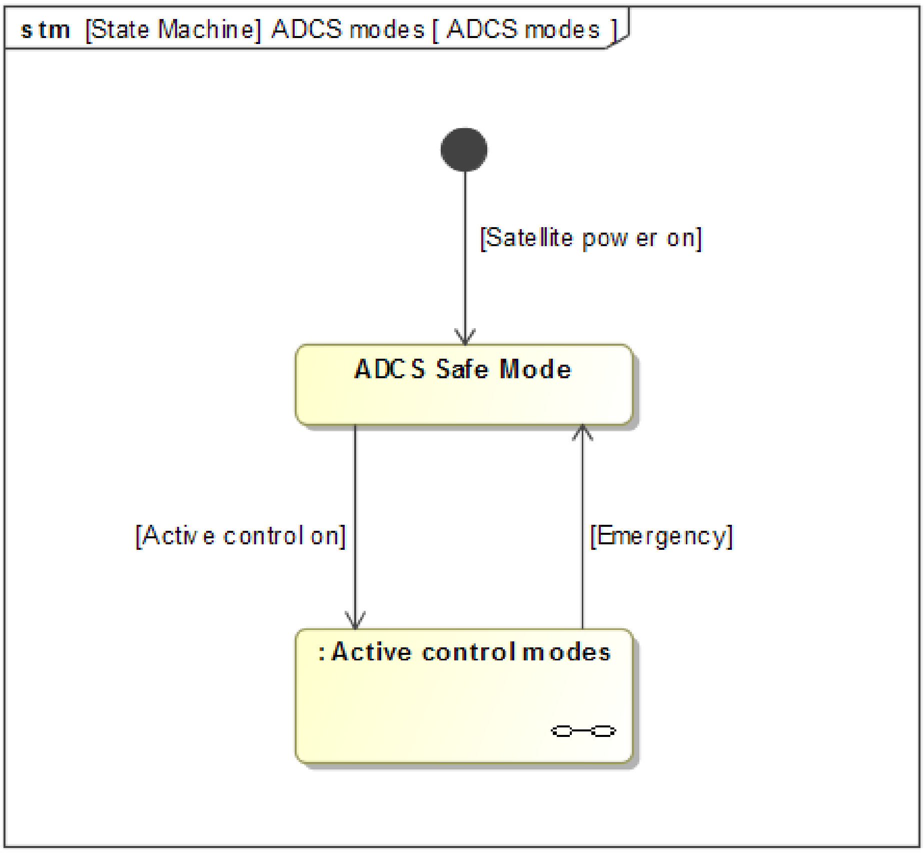


Figure 3.8 Transition to Safe Mode

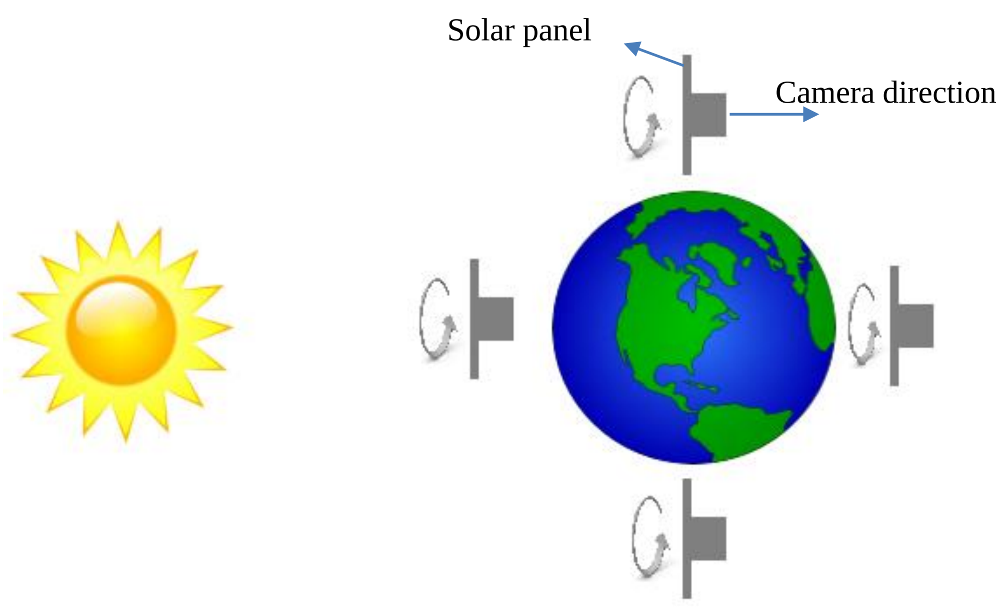


Figure 3.9 Spin Sun Pointing Mode

Spin Sun Pointing Mode is the most reliable active control mode of ADCS in MDG satellite. In this mode, satellite first rotates around Z – axis, then the rotating axis is moved so that solar panels are faced to the Sun. Sensors to be used in this mode are NSAS, FOG and GAS.

MTQs are the only actuators. This SSPM is also the mode in which ADCS consumes least power but solar cells can generate more power. That is the reason why SSPM is chosen to be the activated ADCS mode when MDG is in Safe Mode (note that satellite Safe Mode is different from ADCS Safe Mode).

Similar to Spin Sun Pointing Mode, in Sun Pointing Mode, satellite also needs to face solar panels to the direction of the Sun. However, 3-axis controlled algorithms is used to increase pointing accuracy so that solar cells can generate maximum power as designed from Sun light. This mode is helpful when satellite requires a large amount of electrical power generation in very short time period.

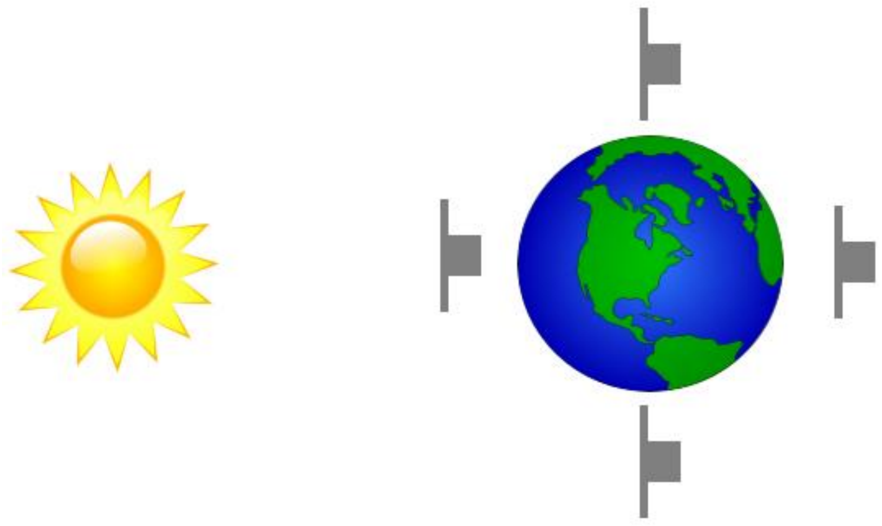


Figure 3.10 Sun Pointing Mode

Position of satellite in orbit as well as the direction of the Sun are calculated from GPS data. Attitude of satellite is calculated from outputs of FOG, NSAS and GAS using TRIAD algorithms. ADCS then computes required torque to be applied to satellite base on attitude error and rotational rate using PD controller. RWs are used for actuation and MTQs are needed to unload RWs momentum.

Control algorithms in Nadir Pointing Mode is the same with SPM, and it requires FOG, GAS, NSAS, GPSR, RWs and MTQs to be turned on. The only different for this mode is control target, when attitude of satellite is needed to be controlled so that mission cameras always point to center of the Earth, as shown in Figure 3.11. This mode is activated when satellite is in nominal operation mode. It is very typical for Earth observation satellite like MDG because in this mode, cameras can always take pictures of the Earth. Another reason for choosing this mode as nominal operation mode is that, when the mission data is required for specific area on the Earth, ADCS can quickly respond to the command received from ground station to change the control mode to target pointing mode and change the direction of camera to target. By doing this, efficiency of the satellite is increased, while the time that satellite is passing above target area is very limited and satellite is traveling with very high speed in the orbit.

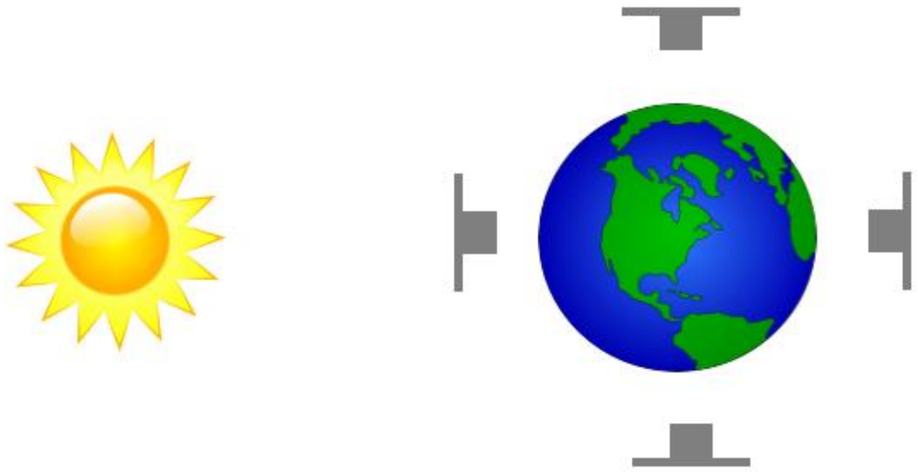


Figure 3.11 Nadir Pointing Mode

Target Pointing Mode is the mode in which the highest pointing accuracy (0.1 degree) is required. As shown in Figure 3.12, satellite is controlled to tilt to target direction on the Earth. List of components to be turned on are FOG, STT, GAS, GPSR, RWs and MTQs. Satellite attitude is calculated from outputs of FOG and STT. The use of STT in this mode is expected to provide the highest attitude determination accuracy to satisfy the requirements form mission payloads. Target attitude is calculated from GPS data and RWs are used as the main actuator. MTQs in other hand, are used for prevent momentum saturations of RWs in certain amount of time.

To this point, five control modes of ADCS are already explained. In addition to those mode, there is a mode that ADCS team defined in MDG satellite called de-tumbling mode. This mode is used in the case when satellite rotational speed to higher than a certain value (currently defined as 0.5 degree per second). MTQs are used as the only actuator to stabilize satellite.

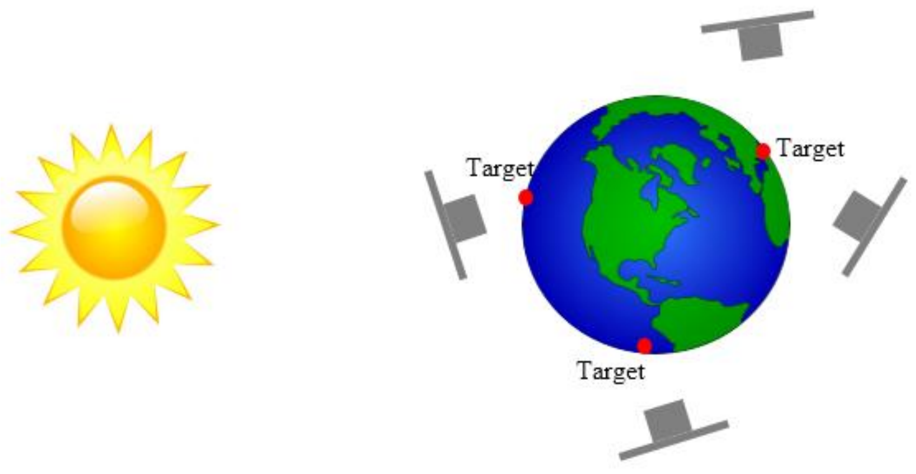


Figure 3.12 Target Pointing Mode

3.2.2 Failures Prediction for ADCS

Since all the design of bus system as well as hardware components of MDG satellite bus system are already decided to follow UNIFROM-1 satellite, the actual FDIR activities started from identify potential failure modes of each hardware component.

There are several approaches to identify potential failures have been applied to the design of FDIR in space systems, including satellites. Failures Modes and Effects Analysis (FMEA) and its variation Failure Mode Effect and Criticality Analysis (FMECA) are the most common approach that are adopted in many space missions including Apollo, Galileo and Skylab [23]. From the first introduction in 1940s, the two techniques are considered as the effective ways to identify potential failures modes of safety critical systems because of the ease of use. However, the disadvantage of FMEA and FMECAs is, while conducting analysis, engineers only focus on single component faults and their effects to the system but not the combinations of component faults which are the main contributions to failures.

In MDG project, Fault Tree Analysis approach is adopted to provide an estimate of failure probability and also to monitor and predict system behaviors, as it is recommended in Ref. [24]. For a critical system as Attitude Determination and Control, FTA is expected to provide

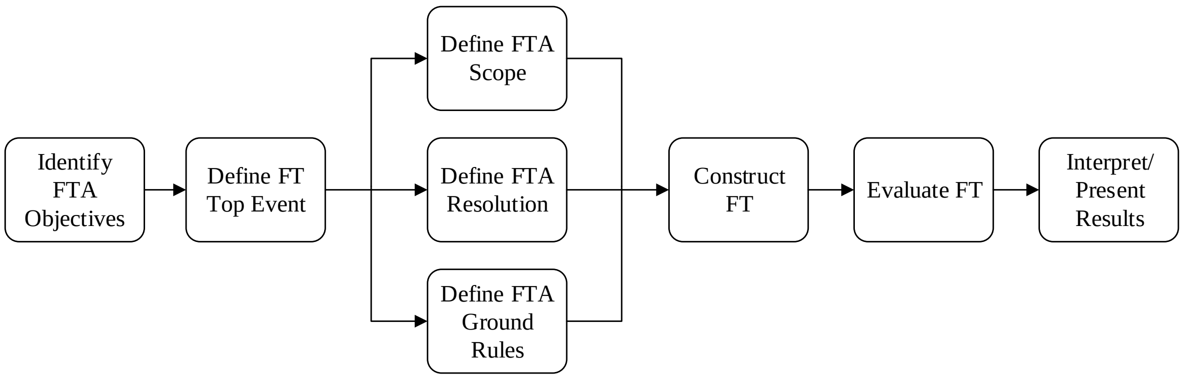


Figure 3.13 Process for conducting FTA

necessary information which can be used to prioritize the importance of each component failure mode to the effects or consequences of unexpected events. This also helps development team to understand about the system that they are going to develop, and more importantly to imagine all the realistic ways that failures can occur. Results of conducting FTA will be used to diagnose causes and plan for possible corrective actions as the part of FDIR.

The process of conducting FTA for ADCS of MicroDragon satellite is described in Figure 3.13, this is the typical process introduced in Ref. [24].

The first step in the process is to identity FTA objectives, one of the five steps to formulate a successful FTA. It is important to clearly define the objectives of FTA from very beginning of the process of designing FDIR so that they are satisfied when the actual FTA is performed. For ADCS in MDG project, FTA is conducted aims to: identify failure modes of each operation mode of ADCS; identify possible scenarios that failures actually happen; and finally, plan FDIR countermeasures for each failure mode.

After objectives are identified, the next step is to define Fault Tree (FT) top event to support those objectives. The top event in FT is the undesired event in which FDIR interventions are needed in order to maintain ADCS operation as well as prevent consequences of that event to the safety of whole satellite system. In this step, the development team decided to conduct different FTs corresponding to different operation modes of ADCS. Five of the important top events are listed as follow, in consideration of the main objective of ADCS in each mode:

- Spin Sun Pointing Mode (SSPM): Solar panels are not pointing to the Sun
- Sun Pointing Mode (SPM): Solar panels are not pointing to the Sun
- Nadir Pointing Mode (NPM): Camera, Antenna (+Z axis) direction is not faced to the Earth
- Target Pointing Mode (TPM): Camera, Antenna direction is not faced to the Earth; Satellite is not stable when camera is taking picture

In step 3, scope of analysis is identified. At this level, components failures are decided to be considered in fault tree and the others such as mechanical, software failures will not be included. All data interfaces and support systems (for example: power, thermal, etc.) are assumed to be working well and lies out of scope of this FTA for the subsystem.

Since top events are defined as the main functional failures of ADCS, it is suggested that FT will be conducted to major components failures level – the failure level when FDIR software plays the main roles to maintain system operation. The FT, if goes too deep will be not only waited time but also introduced additional uncertainties which contribute to the distractions of analysis. This is the result of the step to define FTA resolution.

In next step, ground rules are set among the team in order to create the understandable FT among the team:

- How to name event: using sentences
- List of abbreviations (for component name):
  - NSAS: Non- spin Sun Aspect sensor
  - GAS: Geomagnetic Aspect sensor
  - FOG: Fiber Optic Gyroscope
  - GPSR: GPS Receiver
  - STT: Star Tracker
  - RW: Reaction wheel
  - MTQ: Magnetic Torque
  - OBC: On-board Computer

Step 6 is the step in which actual construction of FT is performed. This requires ADCS team to work effectively together to produce the sufficient FT, which will be used in next step of designing FDIR mechanism and software as well. The results are presented in figures from Figure 3.14 to Figure 3.22. During the FT constructing several basic paradigms as listed in [24] are taken care of. After the construction of each FT, it is important to evaluate if the results are satisfied the objectives identified at the beginning of first step. The final step involves the interpretation and presentation of results. This helps ADCS team to have an overall view of the potential impacts of each failure mode and from that, plan for detection, isolation and recovery actions.

(1) FTA for the top event: Solar panels are not pointing to the Sun (in SSPM).

In Spin Sun Pointing Mode, ADCS first needs to rotate satellite body along z-axis, then it moves the rotating axis to direction of the Sun. The main purpose of this mode is to face solar panels direction to the Sun so that more electrical power can be generated. List of components to be turned on in this mode includes NSAS, FOG, GAS, and MTQ. The top event of FT shown in Figure 3.14 is defined as "ADCS fails to control satellite attitude to face solar panels to the Sun". This can be caused by either, the event that MTQs fail to generate required torque (which is discussed later in Figure 3.22), or the event when ADCS fails to determine Sun direction - the part of FT in the left side. Since the attitude in this mode is calculated by using sensors outputs from NSAS and GAS, so either one of one fails can lead to the event. For NSAS, failures modes are identified as no sensor output when the sensor fails to detect the Sun and fails to trigger Sun flag pin. Two other faults are actually known for MDG, because the same sensor has been using in UNIFROM-1 satellite and has faced those failures modes (see Figure 3.15).

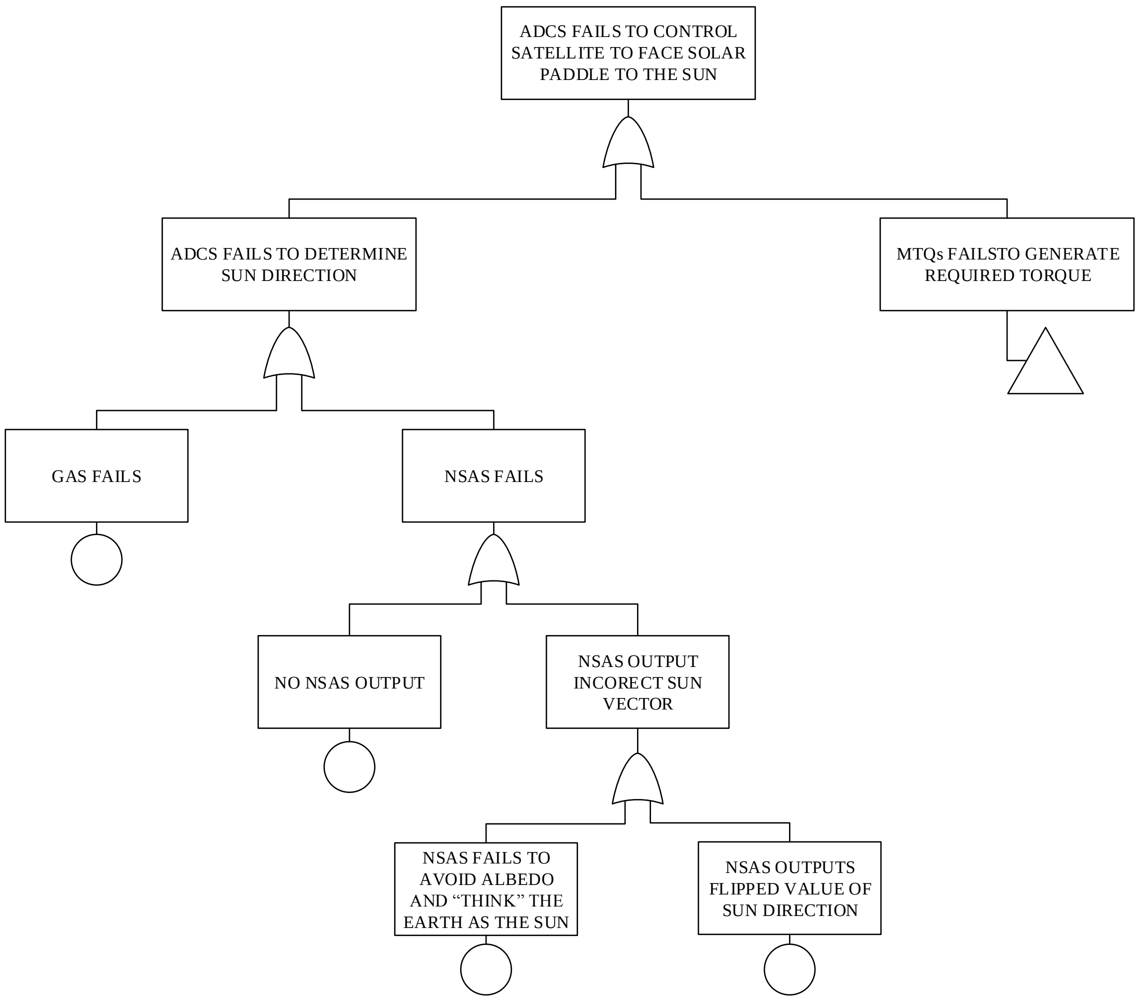

ADCS FAILS TO CONTROL SATELLITE TO FACE SOLAR PADDLE TO THE SUN
ADCS FAILS TO DETERMINE SUN DIRECTION
MTQs FAILSTO GENERATE REQUIRED TORQUE
GAS FAILS
NSAS FAILS
NO NSAS OUTPUT
NSAS OUTPUT INCORECT SUN VECTOR
NSAS FAILS TO AVOID ALBEDO AND "THINK" THE EARTH AS THE SUN
NSAS OUTPUTS FLIPPED VALUE OF SUN DIRECTION

Figure 3.14 FT for Spin Sun Pointing Mode

(2) FTA for the top event: Solar panels is not pointing to the Sun (in SPM)

In this mode, ADCS needs to control satellite attitude to the direction in which solar panels are pointing to the Sun with higher pointing accuracy and more stable than SSPM. Satellite attitude is calculated by using outputs from FOG, GAS and NSAS. Sun direction is calculated from satellite position in orbit using GPSR. Three – axis controlled algorithms is used to control RWs to generate torque, and MTQs are used for unloading RWs momentum. FT is constructed for ADCS in this mode as shown in Figure 3.16

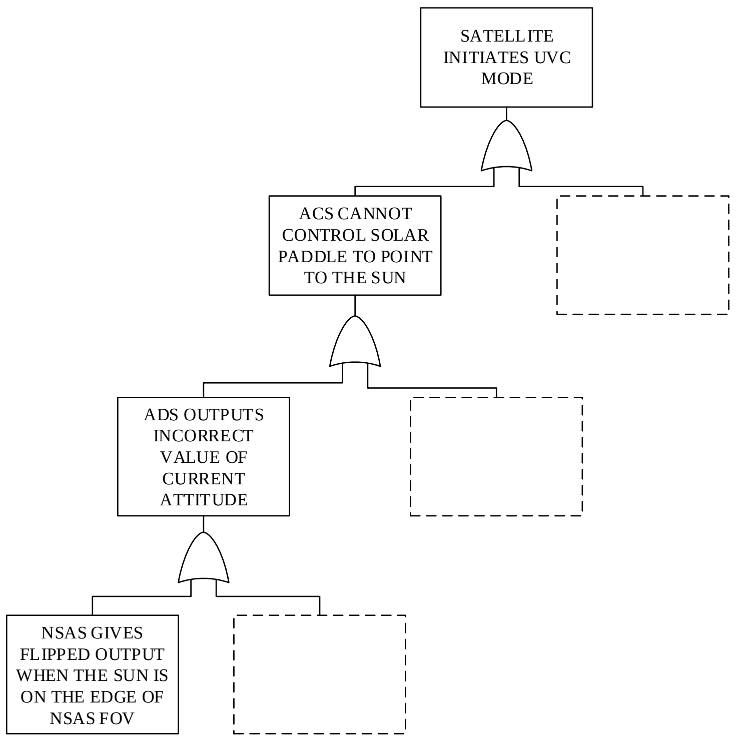

SATELLITE INITIATES UVC MODE
ACS CANNOT CONTROL SOLAR PADDLE TO POINT TO THE SUN
ADS OUTPUTS INCORRECT VALUE OF CURRENT ATTITUDE
NSAS GIVES FLIPPED OUTPUT WHEN THE SUN IS ON THE EDGE OF NSAS FOV

Figure 3.15 Failure caused by NSAS in UNIFORM-1

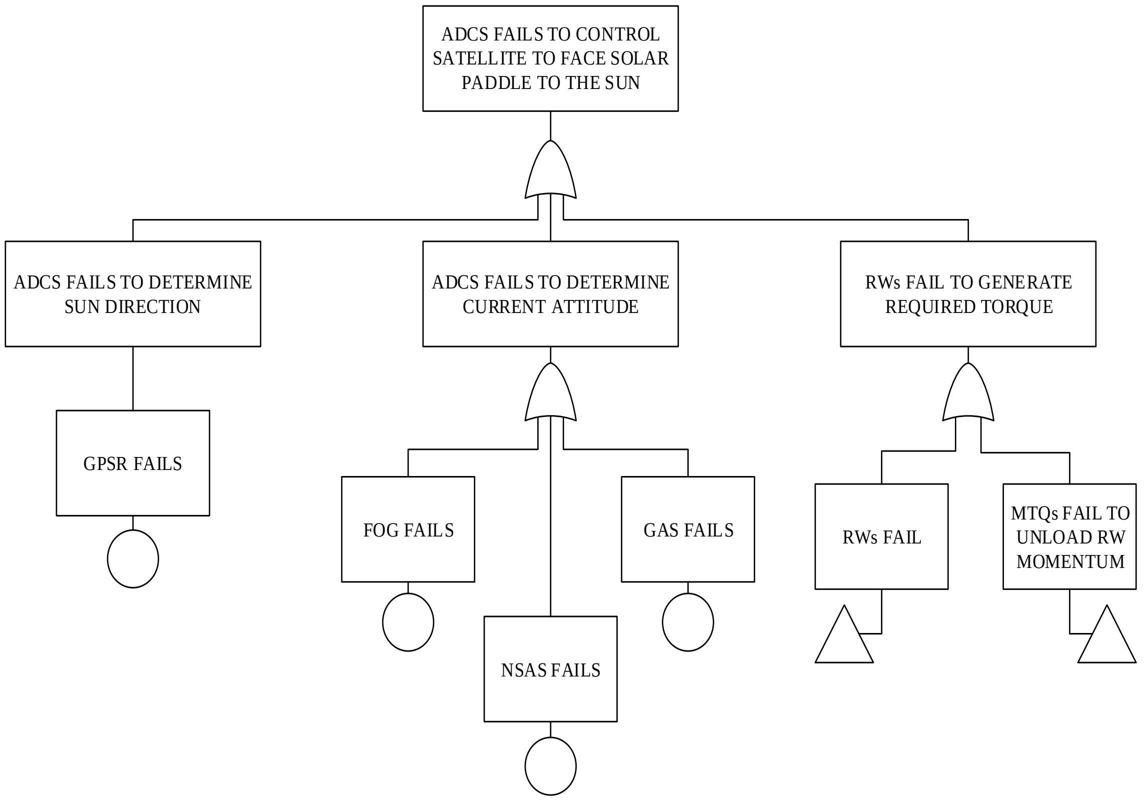

ADCS FAILS TO CONTROL SATELLITE TO FACE SOLAR PADDLE TO THE SUN
ADCS FAILS TO DETERMINE SUN DIRECTION
ADCS FAILS TO DETERMINE CURRENT ATTITUDE
RWs FAIL TO GENERATE REQUIRED TORQUE
GPSR FAILS
FOG FAILS
GAS FAILS
NSAS FAILS
RWs FAIL
MTQs FAIL TO UNLOAD RW MOMENTUM

Figure 3.16 FT for Sun Pointing Mode

As shown in Figure 3.16, the four basic events that could be the causes of this top event are the events caused by GPSR failure, FOG, GAS and NSAS failures. Failure modes of GAS and NSAS are already mentioned above in the FTA of SSPM. The main function of GPSR in this ADCS is to provide information about position of satellite in the orbit (longitude, latitude and altitude). It is also supposed to output such other data as time and date which will be used for orbit propagation on board. GPSR failures may involve the event when that sensor gives no output or provides incorrect outputs as the inputs for navigation algorithms.

RWs are used for actuation in this mode and the failure of them are defined as failed to generate required torque. This fault can be caused by RWs failure itself or because of the failures of MTQs to prevent RWs saturation. As the importance of RWs in ADCS of MDG, more detailed analysis of RWs failures will be discussed in Figure 3.21.

(3) FTA for the top event: Camera, Antenna (+Z axis) direction is not faced to the Earth in Nadir Pointing Mode (NPM)

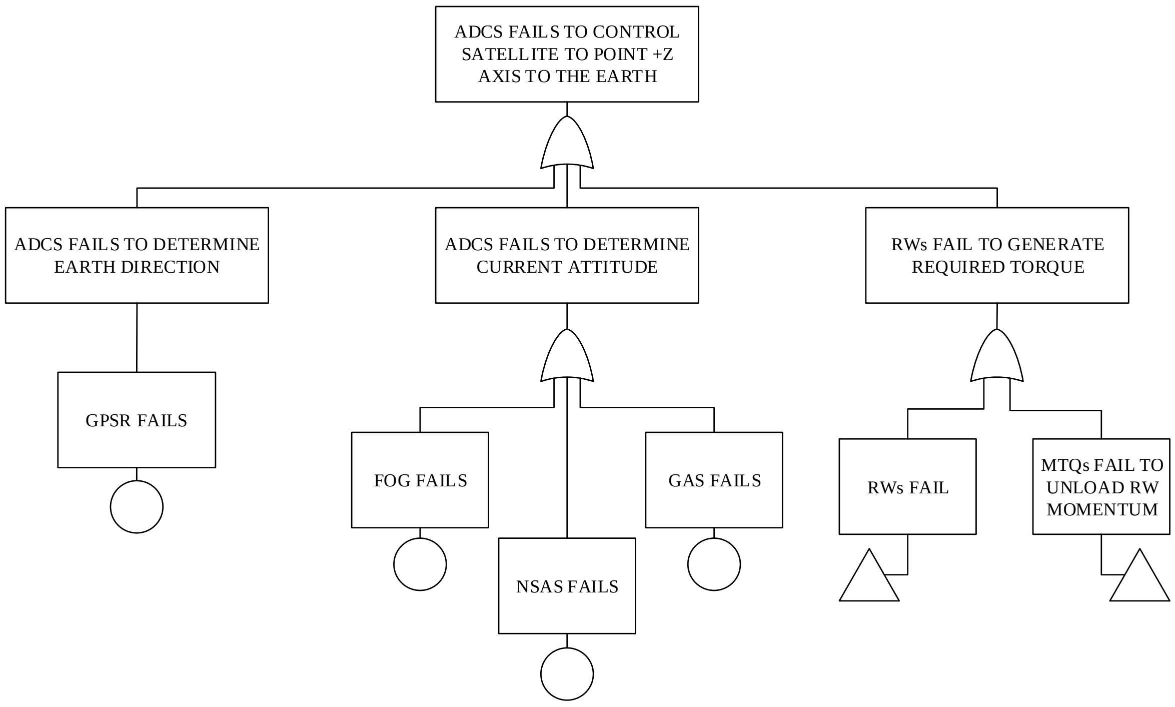

ADCS FAILS TO CONTROL SATELLITE TO POINT +Z AXIS TO THE EARTH
ADCS FAILS TO DETERMINE EARTH DIRECTION
ADCS FAILS TO DETERMINE CURRENT ATTITUDE
RWs FAIL TO GENERATE REQUIRED TORQUE
GPSR FAILS
FOG FAILS
NSAS FAILS
GAS FAILS
RWs FAIL
MTQs FAIL TO UNLOAD RW MOMENTUM

Figure 3.17 FT for Nadir Pointing Mode

In this mode, ADCS is required to control satellite attitude so that +Z axis, in which camera and X-Band antenna are located, is directed to center of the Earth. ADCS also needs to ensure the stabilization of satellite attitude in order to let camera to take pictures when it passes the commanded area on the Earth. List of components to be used in this mode includes GPSR, FOG, NSAS, GAS, RWs and MTQs. They are very similar as they are using in previous Sun Pointing Mode. Control algorithms is also the same.

The top event for constructing FTA in this mode is defined as the event when ADCS cannot control satellite attitude to direct cameras/ antenna direction to the Earth. FT as shown in Figure 3.17 presents the basic event for this failure mode in OR relationship as GPSR fails, FOG fails, NSAS fails, and GAS fails. Actuators failures modes is discussed in detail later.

FOG is used to measure satellite angular rate and it will be used for control satellite stabilization. The benefit of Fiber optic gyroscope is, it provides very accurate output with very low noise. However, it is well known for this kind of sensor to occasionally output error data so-called abruption, ramp or random error. Those data errors of FOG are represented in Figure 3.18.

Those three failures modes of FOG may lead to the situation when satellite is suddenly shaken when camera is doing mission. This may result to bad images quality such as blurry or localization error. Having either one type of those pictures transmitted to ground station then not be satisfied by customer is considered as mission failure.

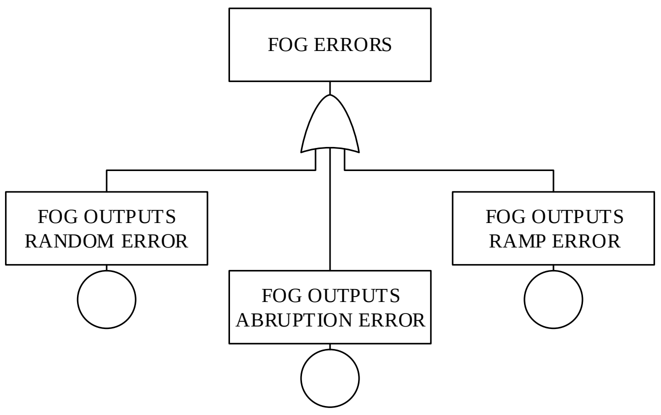


Figure 3.18 Three types of data errors in FOG

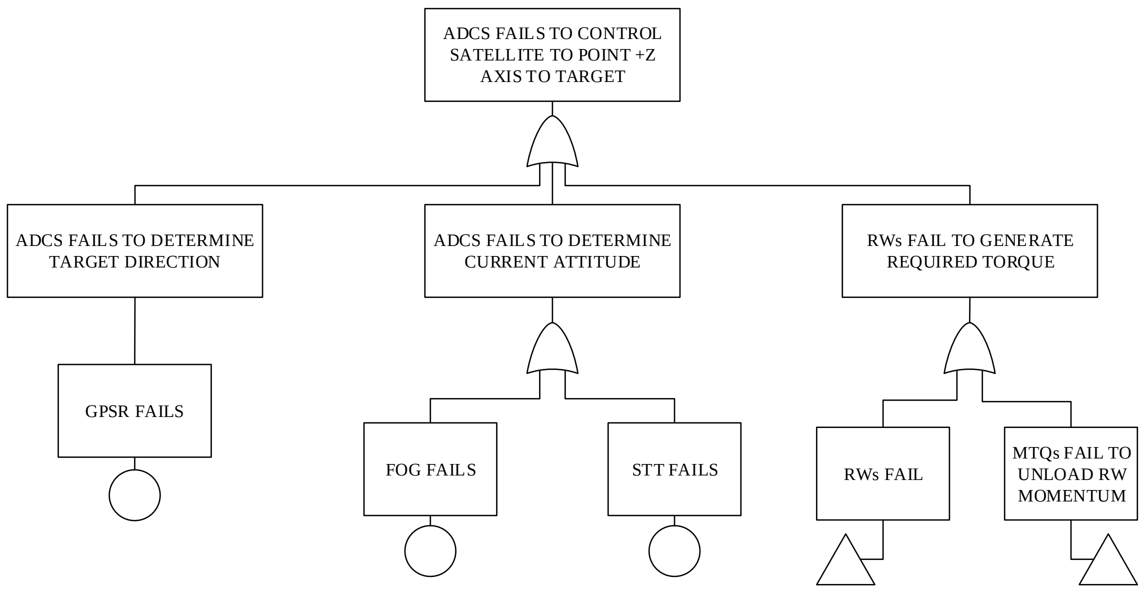

ADCS FAILS TO CONTROL SATELLITE TO POINT +Z AXIS TO TARGET
ADCS FAILS TO DETERMINE TARGET DIRECTION
ADCS FAILS TO DETERMINE CURRENT ATTITUDE
RWs FAIL TO GENERATE REQUIRED TORQUE
GPSR FAILS
FOG FAILS
STT FAILS
RWs FAIL
MTQs FAIL TO UNLOAD RW MOMENTUM

Figure 3.19 FT for Target Pointing Mode (1)

(4) FTA for top event: Camera is not faced to target direction (in TPM)

ADCS functions in this mode is very similar with previous Nadir pointing mode, however, it requires higher pointing accuracy since ADCS needs to control camera direction to exact the location received from ground station as the target. Once the target direction is achieved, ADCS will be required to keep it stable so that camera can acquire mission data by taking pictures of target. STT is used for determine current attitude instead of the combination of NSAS and GAS as used in NPM to gain the determination accuracy. Stabilization is also strictly required in this mode.

Top event for this FT is defined as ADCS fails to control satellite to point +Z axis to target direction. Result of the FT is very similar to the one in NPM, however, failure modes of GAS and NSAS are replaced by STT failures. This is noticeable that STT is one of the components that have higher possibility of fault than other because of its mechanism (like a camera) and stars identification algorithms used in that STT.

(5) FTA for the top event: Satellite is not stable when camera is taking pictures
As mentioned above, in addition to the function of controlling satellite attitude to point to target direction, ADCS also needs to keep satellite stable, especially when mission cameras are taking

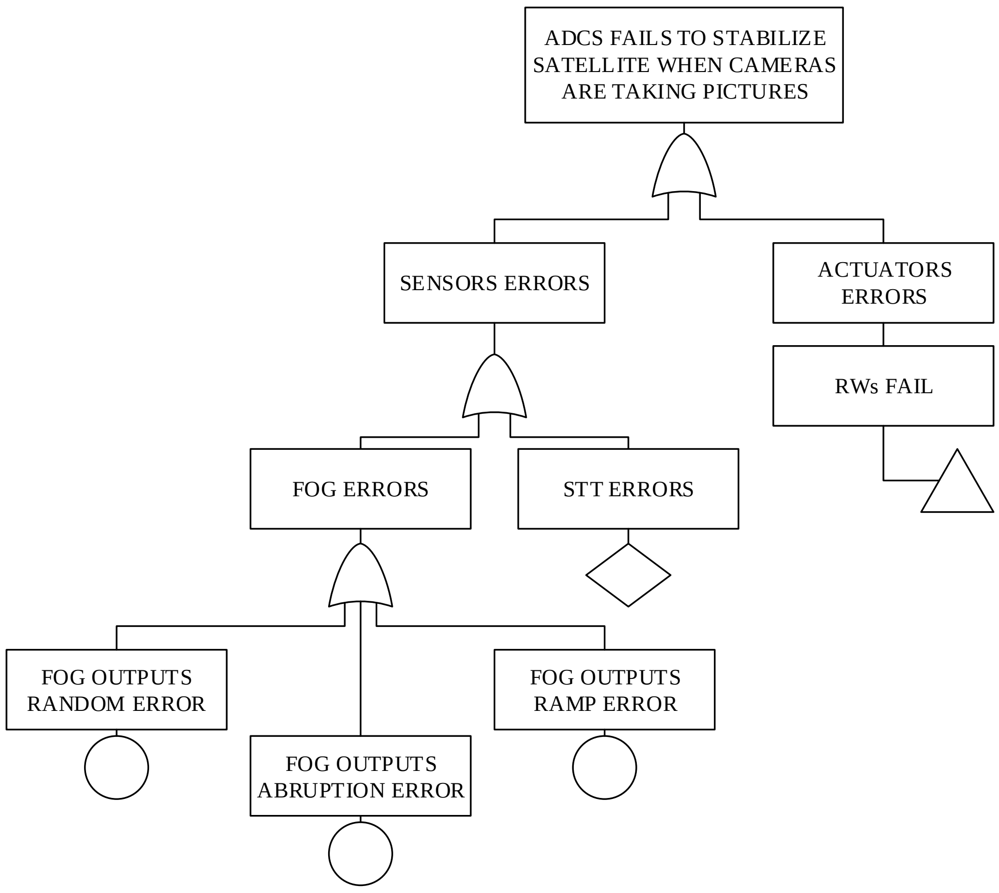


Figure 3.20 FT for Target Pointing Mode (2)

pictures. This is strictly required in target pointing mode when specific target attitude is given from ground station as the only area to observe.

FT for the top event of this aspect is shown in Figure 3.20. As in other FT of this mode, the undesired event can be caused by failure modes of sensors or actuators. FOG failure modes are decomposed in the FT, which is quite similar with what is shown before in Figure 3.18. Failures of RWs are also considered in as one of the mains contributes in this situation as they are the main actuators.

(6) RWs failures

Failures of RWs are now discussed in detailed in Figure 3.21. RW is the only component that is moving (rotating) in ADCS of MDG. It means RW is the component with highest possibility of fail. As the importance of RWs in ADCS, one more RW (which named here RW4) is used as redundant component, despite of its size and weight compared to others.

In Figure 3.21, RWs failures are divided into 4 levels, corresponding to the number of RWs have been failed.

Level 1 failures only consider failure of RW from 1 to 3. They are the main wheels which are used in nominal operation. When failure happens to one of them, the redundant one (RW4) shall be used immediately by FDIR mechanism.

It is quite easy to deal with RW failures at level 1 while ADCS control algorithms does not require to be changed very much. However, from level 2, it is totally different when two or more RWs are failed. FT conducted in Figure 3.21 shows the combinations of those failures. At level 2, when 2 RWs are failed, 9 scenarios are possible. Level 3 with the failures of 3 RWs, up to 24 scenarios defined, and same number are identified for the case when all of 4 RWs are failed (level 4). Several control algorithms have been proposed for this kind of situations (so-called under - actuated satellite) when 2 or 3 RWs have failed [25] [26] [21]. Those are known as the functional recovery using alternative paths and degraded functional recovery [10] in term of FDIR. Using MTQs with the modifications of control commands as the alternative path functional recovery is the sufficient way to deal with RWs faults. Since the technique is guaranteed in Ref. [21], ADCS team decided to apply the results to MDG satellite. This is supposed to reduce a lot of time for planning countermeasures actions as part of FDIR.

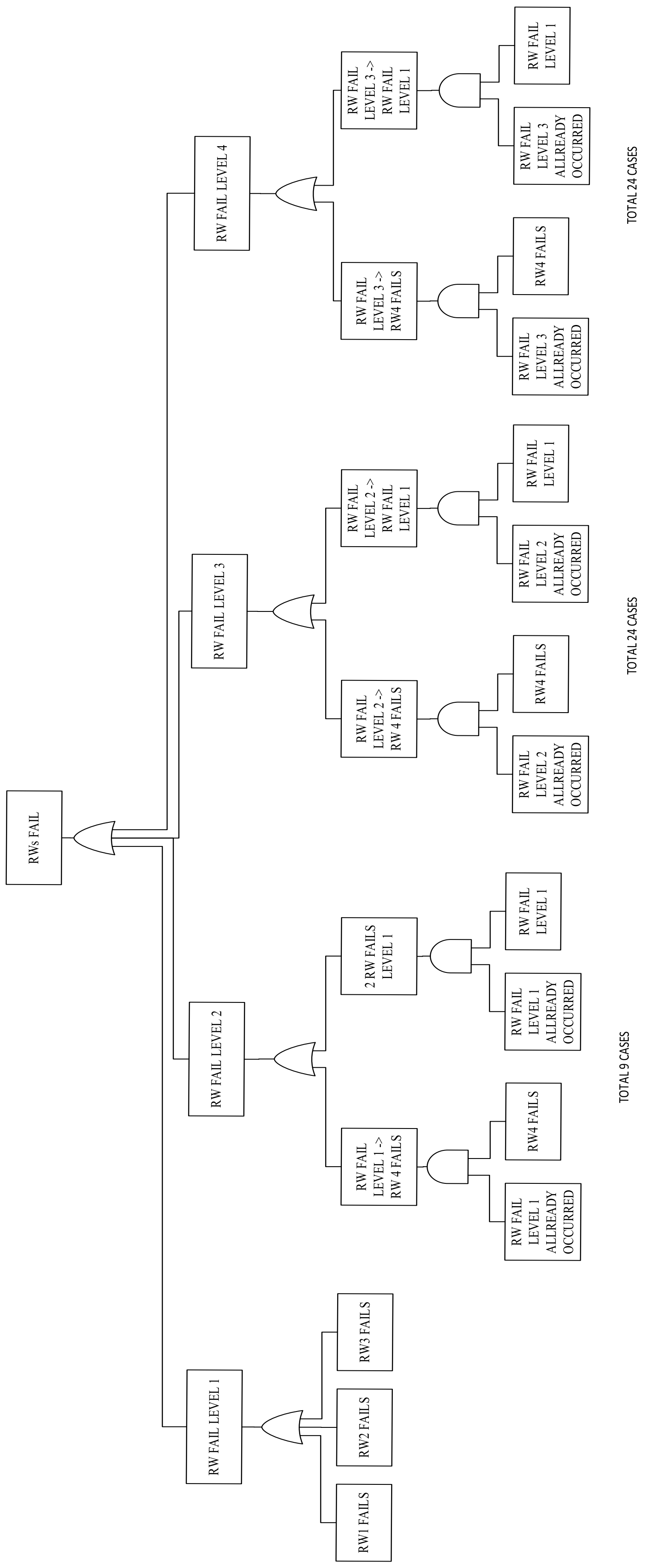

RWs FAIL
RW FAIL LEVEL 1
RW1 FAILS
RW2 FAILS
RW3 FAILS
RW FAIL LEVEL 2
RW FAIL LEVEL 1 -> RW 4 FAILS
RW FAIL LEVEL 1 ALLREADY OCCURRED
RW4 FAILS
2 RW FAILS LEVEL 1
RW FAIL LEVEL 1 ALLREADY OCCURRED
RW FAIL LEVEL 1
TOTAL 9 CASES
RW FAIL LEVEL 3
RW FAIL LEVEL 2 -> RW 4 FAILS
RW FAIL LEVEL 2 ALLREADY OCCURRED
RW4 FAILS
RW FAIL LEVEL 2 -> RW FAIL LEVEL 1
RW FAIL LEVEL 2 ALLREADY OCCURRED
RW FAIL LEVEL 1
TOTAL 24 CASES
RW FAIL LEVEL 4
RW FAIL LEVEL 3 -> RW4 FAILS
RW FAIL LEVEL 3 ALLREADY OCCURRED
RW4 FAILS
RW FAIL LEVEL 3 -> RW FAIL LEVEL 1
RW FAIL LEVEL 3 ALLREADY OCCURRED
RW FAIL LEVEL 1
TOTAL 24 CASES

Figure 3.21 FT of RW

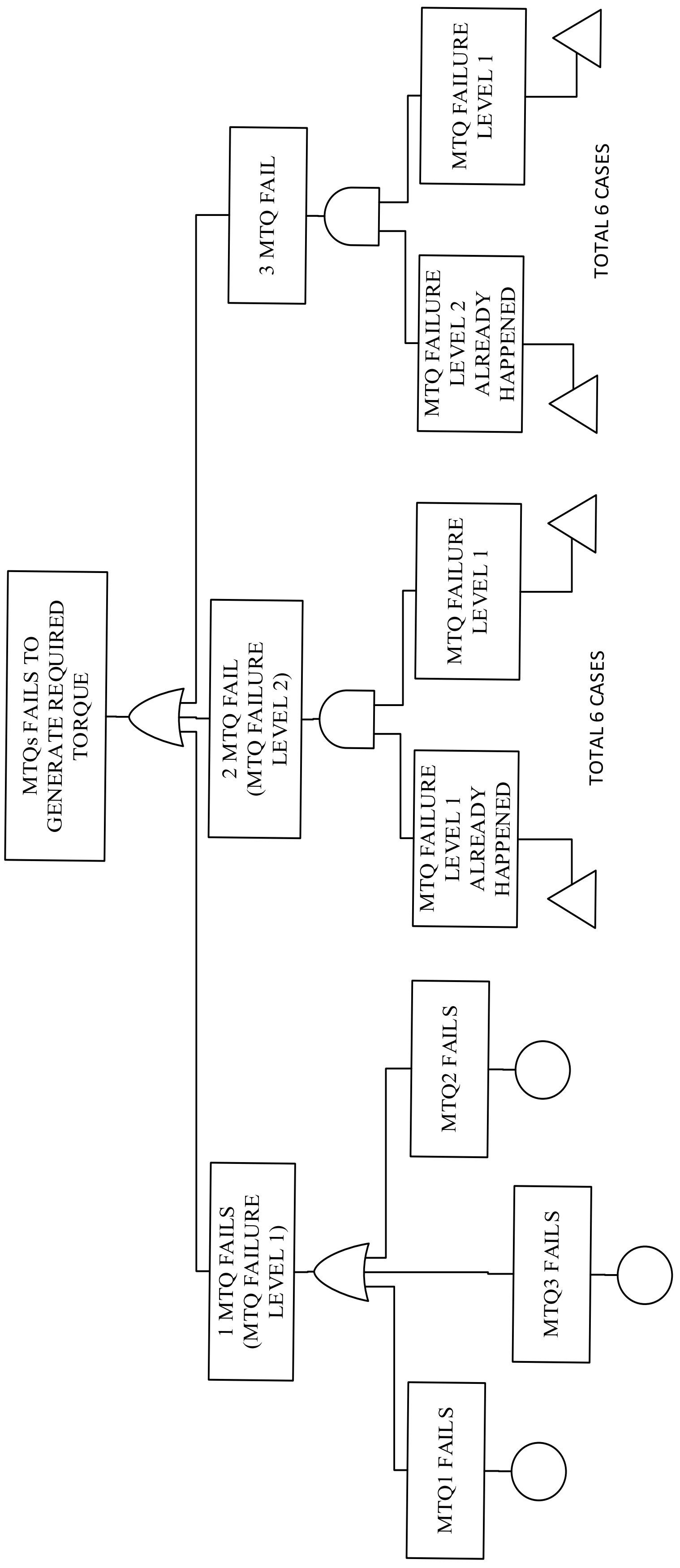

MTQs FAILS TO GENERATE REQUIRED TORQUE
1 MTQ FAILS (MTQ FAILURE LEVEL 1)
MTQ1 FAILS
MTQ2 FAILS
MTQ3 FAILS
2 MTQ FAIL (MTQ FAILURE LEVEL 2)
MTQ FAILURE LEVEL 1 ALREADY HAPPENED
MTQ FAILURE LEVEL 1
TOTAL 6 CASES
3 MTQ FAIL
MTQ FAILURE LEVEL 2 ALREADY HAPPENED
MTQ FAILURE LEVEL 1
TOTAL 6 CASES

Figure 3.22 FT for MTQs

(7) MTQs failures

Because of the simplicity in mechanism and also working principle, MTQs are supposed to be the most reliable component in ADCS of MDG. However, once MTQ gets failed, the effects of it is critical since MTQ is the only actuator used for tumbling and SSPM. FT is conducted for MTQs and shown in Figure 3.22.

To this point, potential failures modes of ADCS in MDG satellite are identified. It is very important step in designing ADCS, especially in term of FDIR while it makes the process of designing FDIR become less stressful. However, space environment is still unknown in many aspects, and because of that, having all potential failures modes predicted is impossible. As pointed out in Ref. [7], FTA is not the complete representation of all possible faults and failures in a system. Many satellites have failed before their missions are established [2] [3]. Therefore, no one can assure that the design of FDIR as well as the countermeasure plans they have made before satellites are launched to the orbit is competed or perfects. To MDG project, this problem is more critical since almost members in the development team are very new to space industry and MDG is their first satellite project. It is required to have an adjustable design for FDIR of ADCS so that unpredicted failure modes can be recovered even when satellite is in operational phase.

Chapter 4. FDIR Mechanism for ADCS and Design of Adjustable Software

4.1 Proposed FDIR mechanism for ADCS of MDG satellite

4.1.1 Finding from constructing FTA

Section 3.2.2 discussed about several of failure modes of ADCS. Constructed FTs show the possible ways or mechanisms that failures can occur during the operation of ADCS. Also, in the same section, results of conducting fault tree analysis pointed out the causes of those failure modes. A lot of faults, mainly for components, are pointed out for analyses. It is not so difficult to find out that, although the combinations of possible ways failures occur which are the so-called failure scenarios are so many, there are limited number of causes (as the basic events in FT) are identified and failure modes are sharing the common causes. Failures of one hardware component can lead to various types of failures in the system in different operation modes. For example, failures of FOG lead to the fluctuation of satellite in Target Pointing Mode which results in blurred images taken by mission cameras. It is considered as failure of ADCS since this subsystem cannot keep satellite stable when it is needed to be. The same failure of FOG also causes the failure of ADCS in Sun Pointing Mode when ADCS cannot get the correct values from sensor to determine current attitude so that solar panels are pointed to different direction rather than direction of the Sun.

When common causes are identified for different failure modes, it is possible that those failures can be fixed or recovered in the same ways. In other words, one can say that, a set of isolation and recovery actions can definitely support to fix several failure modes of the system. Continue the example of the failures that caused be failure of FOG, once they are detected and identified, there is no doubt that the failures can be simply fixed by performing the same action (reset the sensor, for example) or set of actions.

The challenge here is finding the appropriate actions to deal with those various type of failure mechanisms. It requires a lot of time, efforts as well as experiences which are not the advantages in MDG project when the satellite is developing by students. They are newcomers to space industry, and at the same time they have to study at schools as master students and work on this satellite development project. Finally, the satellite is required to be completed when students finish their master courses.

4.1.2 Proposed FDIR mechanism

Because of the time limitation as well as the lack of experiences, having a simple but efficient design of FDIR for ADCS becomes essential. Time limitation can lead to the un-finished implementation of the FDIR. The lack of experiences in ADCS team also yields the needs of making changes to the design as time goes and experiences are accumulated.

For MDG project, there are needs of adjustments for FDIR even after satellite is launched to the space, so FDIR is required to be changeable in some aspects. In such a system like satellite, making changes to hardware after launch is impossible because once satellite is launched, it cannot be got back. Making adjustments in software is the only way to change the way ADCS dealing with failures. Some suggestions for this situation are proposed in [2], software redundancy is one of them. Hardware redundancy is one of the most common techniques which been using to enhance system safety. In design of ADCS this technique is also adopted as the use of one redundant reaction wheel. However, when it comes to software, more redundant has same meaning with more complexity [27], and as the result, failure probability becomes higher.

The proposed FDIR mechanism for attitude determination and control subsystem of MDG satellite is shown in Figure 4.1. This is a general mechanism for fault detection, isolation and recovery of any system. What makes it original is the use of FIR library. Detailed explanation of this mechanism is as follow:

Fault detection is performed by using model – based technique. This technique is very simple and straightforward. An observer is implemented for collecting necessary information to

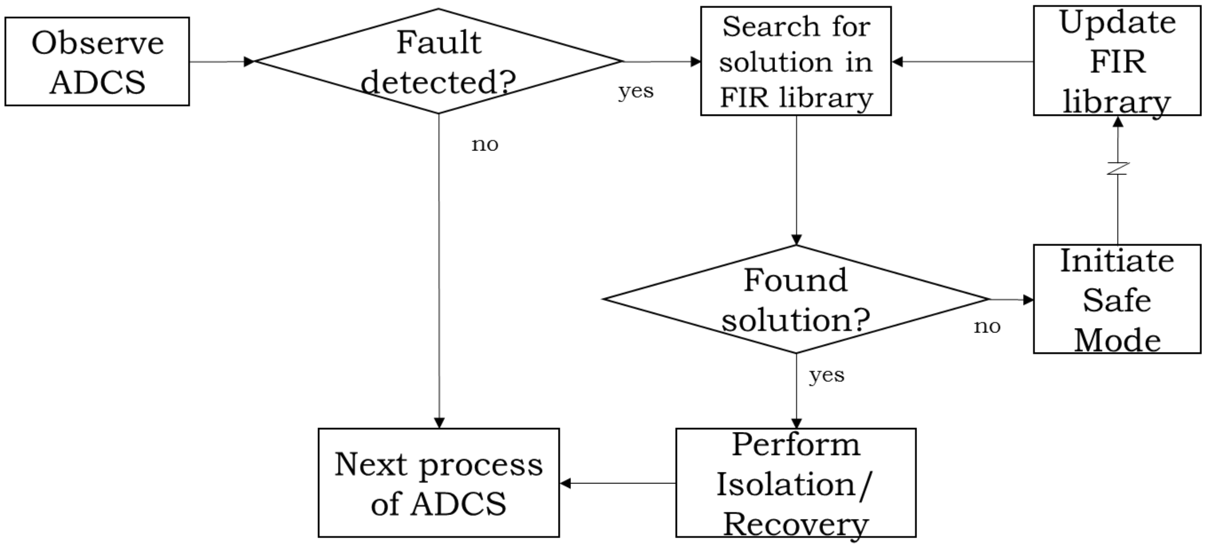


Figure 4.1 Proposed FDIR mechanism

calculate and provide the estimations of sensors measurements. These estimated values then be compared to the actual values output from sensors for purposes of faults detection. Residuals are generated from comparisons. A threshold is set so that generated residuals are compared to this value to give the conclusion about components.

There are two kinds of thresholds are used for fault detection in ADCS of MDG satellite which are fixed thresholds and adaptive thresholds. A fixed threshold is a constant value corresponding to each type of sensor. Fixed thresholds are used for fault detection because they are simple to use and also very reliable [28]. Adaptive thresholds, on the other hand, are less reliable and the uses of them requires to add additional computations to onboard software. The big advantage with adaptive thresholds is the enhancement of sensitivity for fault detection. Depends on current state of system, adaptive threshold provides the optimal magnitude to detect failure more accurately. They are especially helpful in the cases of components performance degradations.

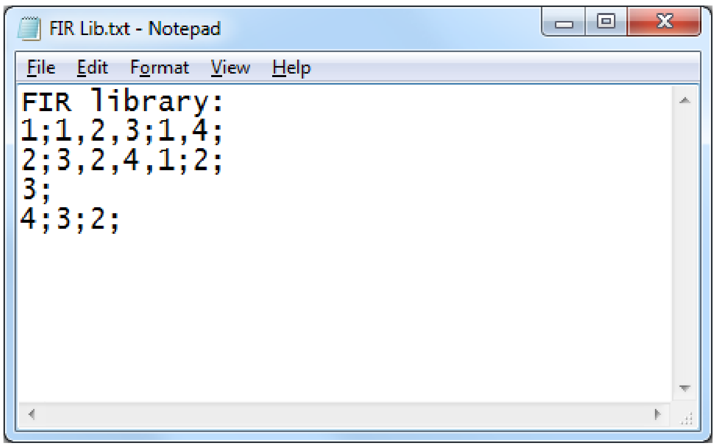
Once a fault is successfully detected, the next step is to identify the appropriate cause. In this step, the residuals which are generated in fault detection are also expected to provide additional information in such a way that a specific fault can be determined easier. However, only residuals are not enough. This process also requires the more information about system such as on/off state of components in ADCS (sensors, actuators) in order to give the conclusion about faulty component.

The next step after the causes are identified is to take appropriate actions for fault isolation and to recover ADCS system operation. Usually, a list of actions and their logical relationship with each failure mode are defined ahead in software [6]. FDIR software performs those actions by deploying corresponding pre-defined functions as they are planned ahead and fixed in the software source code. In this research, a library called FIR library is introduced to contain those relationships.

FIR library which contains a list of fault IDs and isolation and recovery functions corresponding to each fault ID, is a text file saved in onboard memory before satellite is launched to the orbit. This text file is created based on the results of fault tree analysis and countermeasures planning actions. Example of FIR library is shown in Figure 4.2.

There are several important rules for making FIR library to ensure the well functional of FDIR software when failure occurs. They are defined as following:

- ✧ First line in text file contains the phrase "FIR library" to indicate this is the library for FIR purposes



```
FIR library:
1;1,2,3;1,4;
2;3,2,4,1;2;
3;
4;3;2;
```

Figure 4.2 Example of FIR library

- ✧ From second line, each line indicates one failure mode and its isolation and recovery solutions. Format of each line is defined as follow:
  - ➢ Fault_ID;Isolation_func_01_ID,Isolation_func_02_ID,…;Recovery_func_01_ID,Recovery_Func_02_ID,…;<CR>
  - ➢ Fault ID, Isolation functions, and Recovery functions are separated by one semicolon sign, (;)
  - ➢ If there are more than one Isolation or Recovery functions, they need to be delimited by a comma sign, (,)
  - ➢ There is one carriage return, <CR>, at the end of line

Once a fault of ADCS is identified, a fault ID is assigned as it is defined beforehand in software. The software, then, searches in library for correspondent isolation and recovery functions and performs those functions. For example, with FIR library in Figure 4.2, once fault ID number one "1" is assigned to current fault of ADCS, software can easily find isolation functions with ID: 1, 2, 3 and recovery function with ID: 1, 4 to perform. All the functions with respected to the ID in FIR library are needed to be defined beforehand in software source code. One more thing that needs to be notice for the use of FIR library is that, the software will perform isolation and recovery functions in the order as they are defined inside the library. If "1, 4" are defined in library, function with ID number 4 shall be performed after function number 1 has been completed.

When there is no isolation and recovery function available in FIR library for a detected fault, the software will take action to activate ADCS Safe Mode and wait for ground interventions.

The introducing of FIR library, first, supposes to reduce the efforts for making software of FDIR since the logical relationship between ADCS fault and isolation recovery functions as have been defined in software now are represented in text file library which is very short and simple. In the proposed FDIR mechanism of ADCS, it is not software guiding itself to deal with failures when they happen but the FIR library. FIR library will have responsibility to provide software necessary instructions to deal with every fault of ADCS so that those they can be isolated and failure can be prevented or ADCS operation can be restored. Secondly, as it is text file, it is very easy to be updated and modified as time goes and more failure modes are identified or more precise order of actions are investigated. Last but not least, the most important contribution of this FIR library for this research is it enables the design of adjustable software that will be explained in the next section.

4.2 Adjustable Software Design for FDIR of ADCS

In this section, the design of an adjustable software for fault detection, isolation and recovery of ADCS is going to be explained.

4.2.1 Requirements Definition

As mentioned in section 4.1 the software is required to be able to accept changes even when satellite is launched to the orbit but still, the functions of fault detection, isolation and recovery have to be ensured. The detailed requirements are identified as follow:

- ✧ For the predicted failure modes, the software shall be detect, isolate the fault and recover ADCS operation automatically
- ✧ If un-predicted failure modes occur, the software shall initiate ADCS Safe Mode and can be adjusted to perform isolation, recovery solution by operators from ground station.

4.2.2 Context Analysis

Text – based requirements for the software to be designed are captured in System Modeling Language (SysML) requirement diagram and illustrated in Figure 4.3.

In additional to requirements, there are also several constrains for the design of software. For computation and processing time, it should not be exceeded OBC cycle time as it is already defined. Also, as mentioned in section 3.2, the design of software must not require any changes in hardware of ADCS since they are not subjected to change.

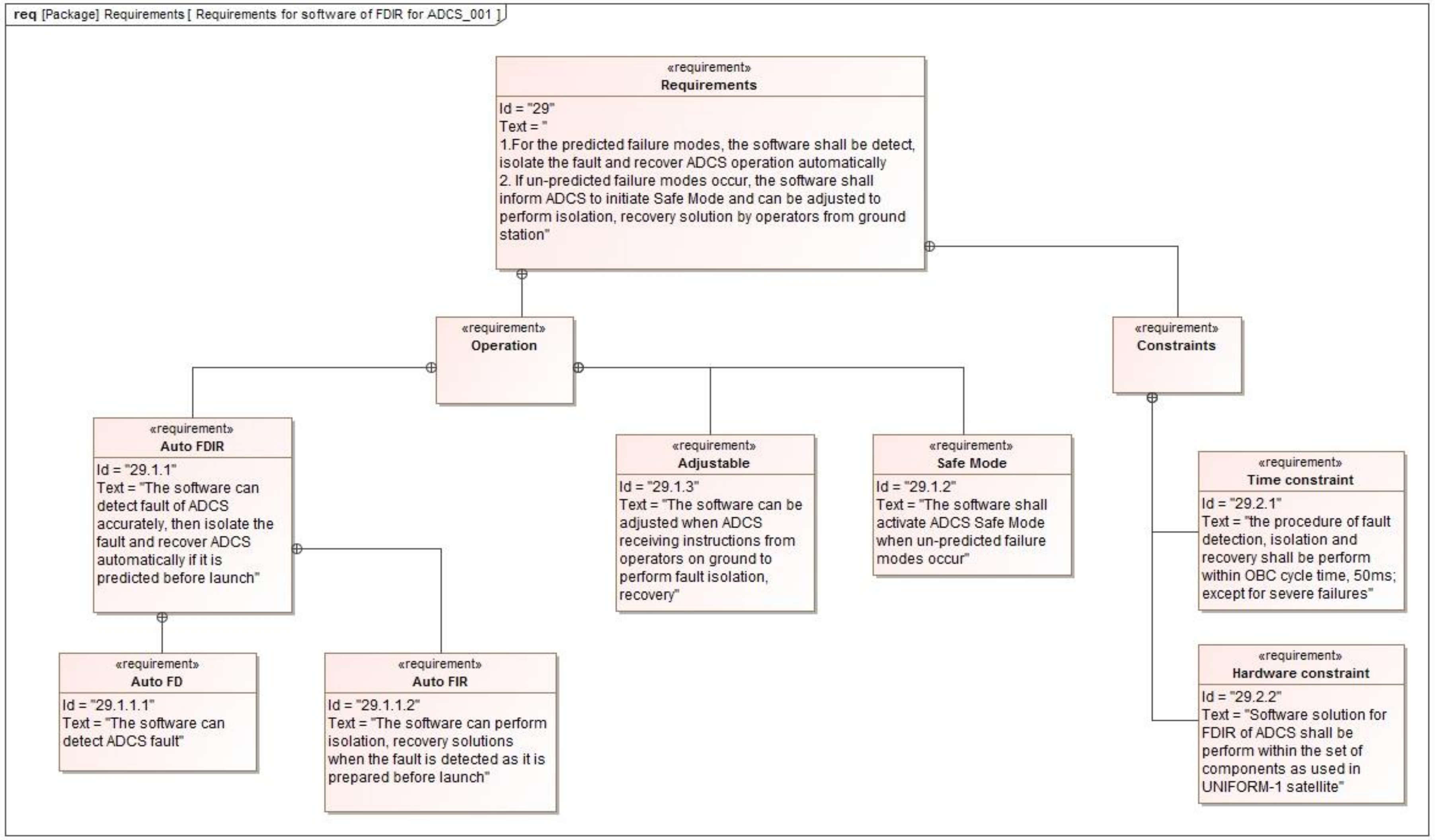

req [Package] Requirements [Requirements for software of FDIR for ADCS_001]
«requirement» Requirements
Id = "29"
Text = "
1.For the predicted failure modes, the software shall be detect, isolate the fault and recover ADCS operation automatically
2. If un-predicted failure modes occur, the software shall inform ADCS to initiate Safe Mode and can be adjusted to perform isolation, recovery solution by operators from ground station"
«requirement» Operation
«requirement» Constraints
«requirement» Auto FDIR
Id = "29.1.1"
Text = "The software can detect fault of ADCS accurately, then isolate the fault and recover ADCS automatically if it is predicted before launch"
«requirement» Adjustable
Id = "29.1.3"
Text = "The software can be adjusted when ADCS receiving instructions from operators on ground to perform fault isolation, recovery"
«requirement» Safe Mode
Id = "29.1.2"
Text = "The software shall activate ADCS Safe Mode when un-predicted failure modes occur"
«requirement» Time constraint
Id = "29.2.1"
Text = "the procedure of fault detection, isolation and recovery shall be perform within OBC cycle time, 50ms; except for severe failures"
«requirement» Auto FD
Id = "29.1.1.1"
Text = "The software can detect ADCS fault"
«requirement» Auto FIR
Id = "29.1.1.2"
Text = "The software can perform isolation, recovery solutions when the fault is detected as it is prepared before launch"
«requirement» Hardware constraint
Id = "29.2.2"
Text = "Software solution for FDIR of ADCS shall be perform within the set of components as used in UNIFORM-1 satellite"

Figure 4.3 Requirements for FDIR software

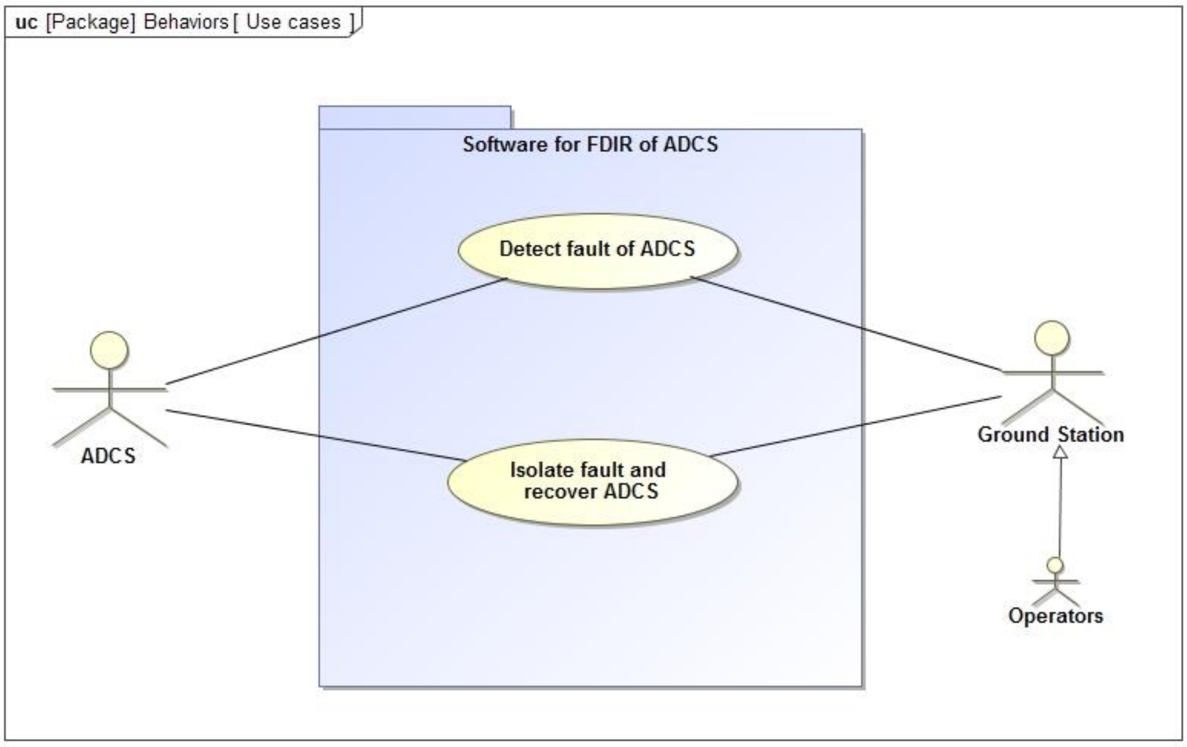

uc [Package] Behaviors [Use cases]
Software for FDIR of ADCS
Detect fault of ADCS
Isolate fault and recover ADCS
ADCS
Ground Station
Operators

Figure 4.4 Use case diagram for top – level functionality

4.2.2.1 Top - level functionality

Continue the sequence of model – based system engineering, the next step is to define top-level functionalities of the software using SysML use case diagrams. Two core functions of the any FDIR software in general are defined as:

- Detect fault of ADCS
- Isolate fault and recover ADCS

Two use cases correspondent with the two top-level functionalities are defined in Figure 4.4 for the FDIR software of ADCS.

4.2.2.2 Functional requirements

Sequence of two top-level functions of the software are represented in Figure 4.5. After a fault of failure is detected, isolation and recovery functions need to be performed.

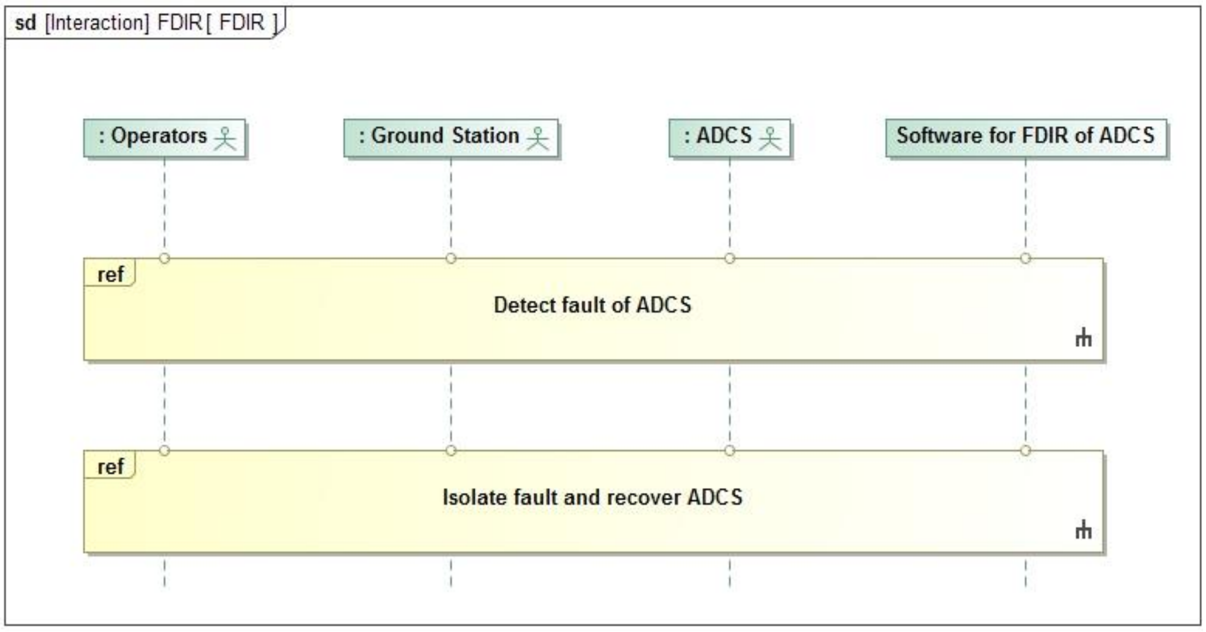

sd [Interaction] FDIR[FDIR]
: Operators
: Ground Station
: ADCS
Software for FDIR of ADCS
ref
Detect fault of ADCS
ref
Isolate fault and recover ADCS

Figure 4.5 Sequence diagram of FDIR

Each use case that mentioned in Figure 4.4, then be described in more detailed by using SysML sequence diagram. This step helps ADCS team to identify the interactions between FDIR software and attitude determination and control system as well as ground station and operators. Constructing sequence diagram then helps to identify functional requirements for the software system.

Figure 4.6 illustrates the sequence diagram for the use case: detect fault ADCS. As already explained in section 4.1.2, to detect fault of ADCS, observers are implemented in the software. ADCS status such as the variables in ADCS software as well as sensors outputs will be monitored by observers. From collected information, fault will be detected onboard. Two functional requirements for the software are obtained from this sequence diagram are:

- ✧ Observe ADCS status
- ✧ Detect fault (of ADCS)

Second use case of FDIR software is described in sequence diagram shown in Figure 4.7. Once a fault is detected, the software will trigger second function to perform isolation and recovery actions. Identify fault (or failure cause) is the first step in this sequence. Then if faulty component is successfully identified, the software next will perform isolation and recovery actions. Detailed sequence diagram for this is shown in Figure 4.8. If the cause is not identifiable, ADCS Safe Mode is initiated then software will wait for ground station interventions. This is one of the situations when adjustability of software will play the main role to support restoring the normal operations of ADCS.

Functional requirements obtained from sequence diagrams in Figure 4.7 and Figure 4.8 are:

- ✧ Identify fault
- ✧ Search isolation, recovery functions in FIR library
- ✧ Activate ADCS Safe Mode
- ✧ Execute isolation, recovery functions

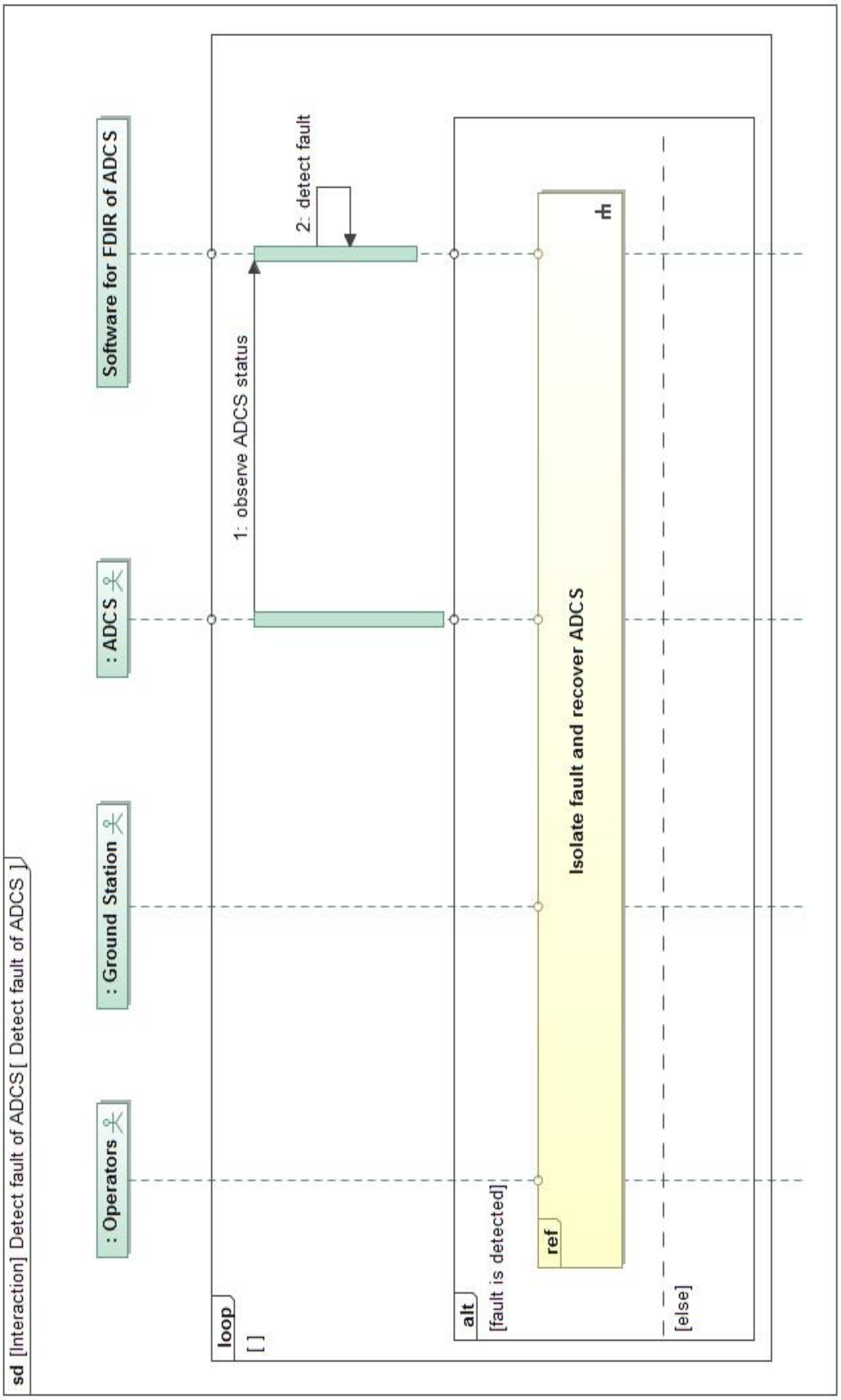

sd [Interaction] Detect fault of ADCS [Detect fault of ADCS]
: Operators
: Ground Station
: ADCS
Software for FDIR of ADCS
loop
[ ]
1: observe ADCS status
2: detect fault
alt
[fault is detected]
ref
Isolate fault and recover ADCS
[else]

Figure 4.6 Sequence diagram for detect failures of ADCS (context level)

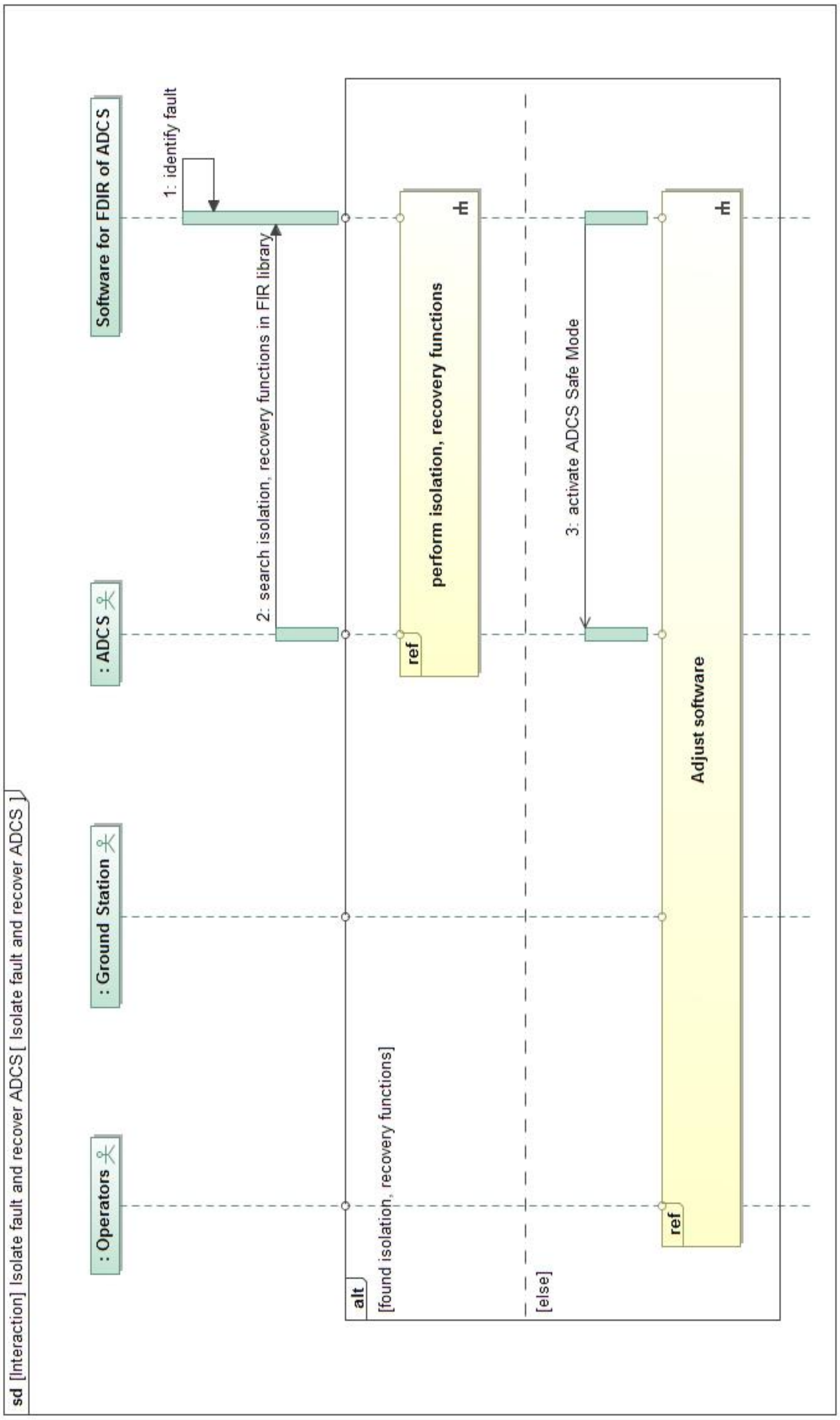

sd [Interaction] Isolate fault and recover ADCS[Isolate fault and recover ADCS]
: Operators
: Ground Station
: ADCS
Software for FDIR of ADCS
1: identify fault
2: search isolation, recovery functions in FIR library
alt
[found isolation, recovery functions]
ref
perform isolation, recovery functions
[else]
3: activate ADCS Safe Mode
ref
Adjust software

Figure 4.7 Sequence diagram for isolate failures and recover ADCS (context level)

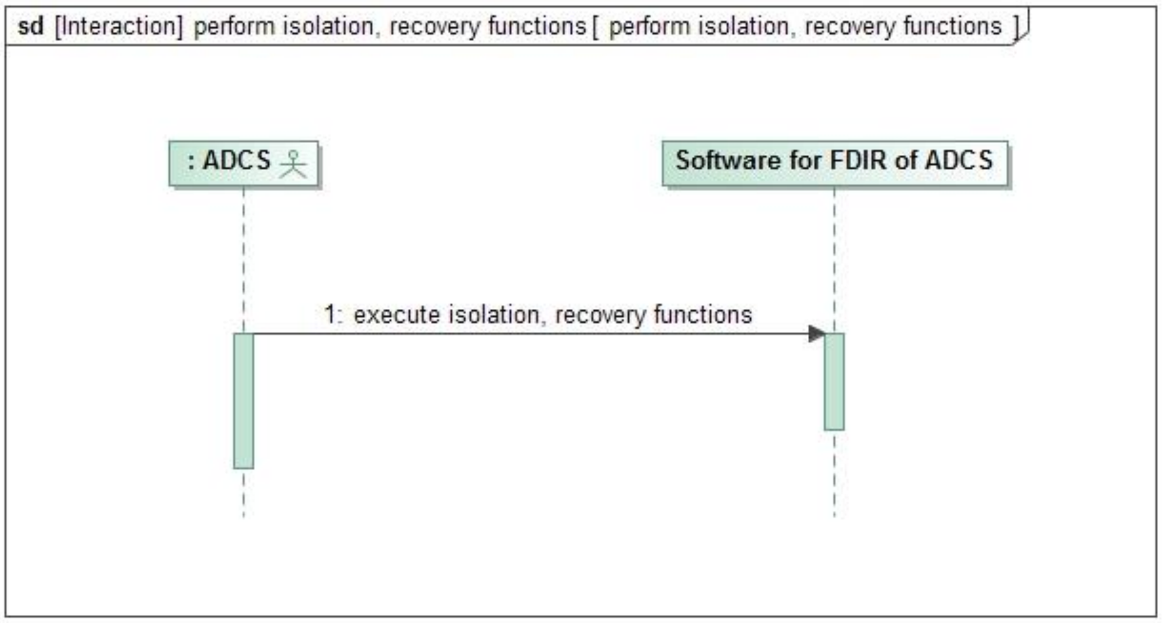

sd [Interaction] perform isolation, recovery functions [perform isolation, recovery functions]
: ADCS
Software for FDIR of ADCS
1: execute isolation, recovery functions

Figure 4.8 Sequence diagram for perform FIR functions (context level)

Sequence of how software can be adjusted from ground station is shown in Figure 4.9. It starts from when operators on ground receive telemetry data from satellite and conduct telemetry data analysis. Obviously, this kind of analysis is conducted when satellite has faced failure mode that cannot be recovered onboard, to find out what led to that failure. However, even when satellite is in normal operation, the work of analyzing telemetry is performed frequently first to ensure satellite is working well in orbit, secondly to find out or forecast and plan for future of operation. For example, from analyzing telemetry received from EPS subsystem, operators can decide what mode of ADCS should be initiated for ensure power safety for whole satellite system. Also, from those data, they can decide the operation of satellite in the next orbit cycle, such as how long mission payloads can be turned on to do the missions, or how long transmitter can be turned on to send mission data to ground station.

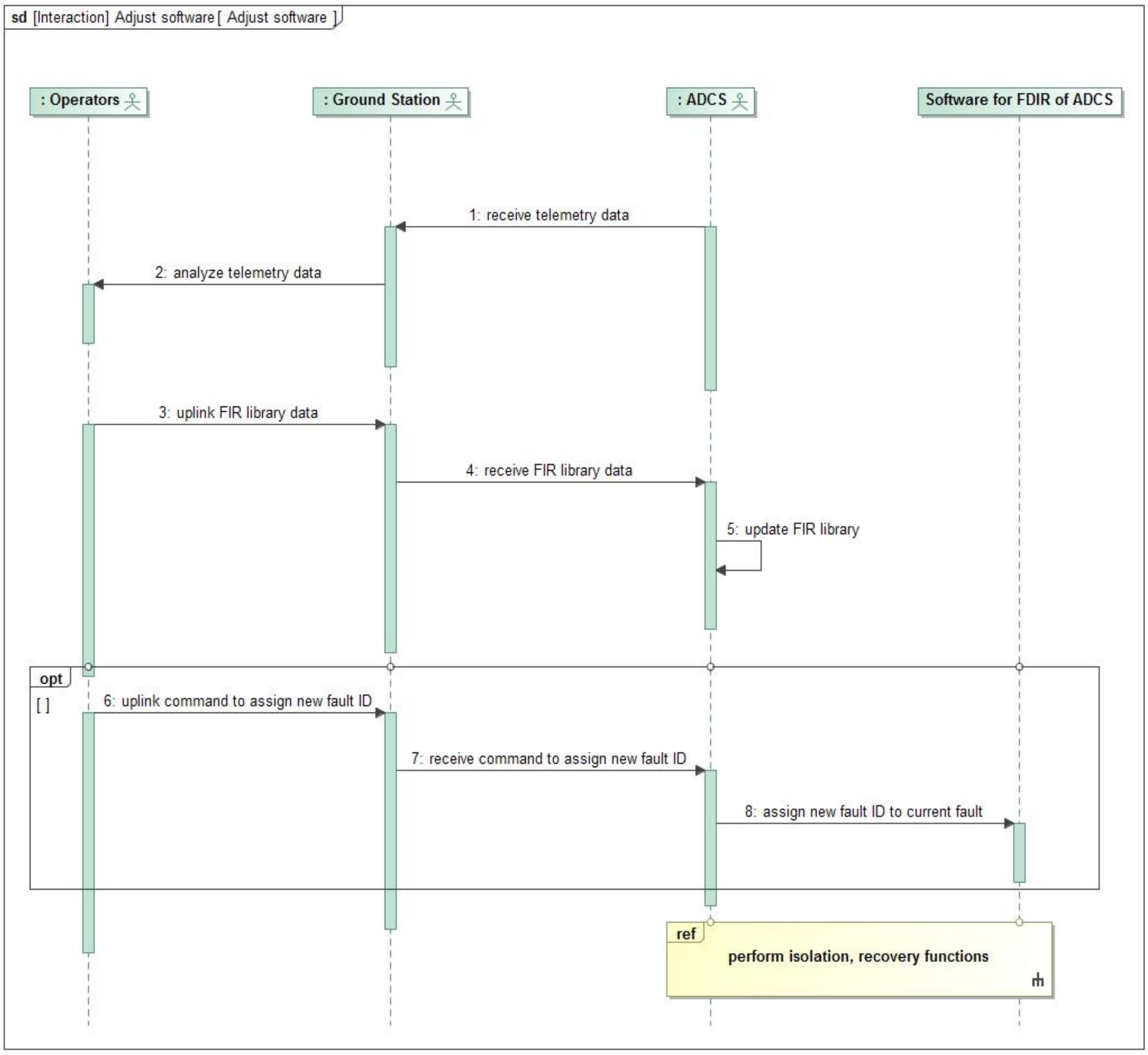

sd [Interaction] Adjust software [Adjust software]
: Operators
: Ground Station
: ADCS
Software for FDIR of ADCS
1: receive telemetry data
2: analyze telemetry data
3: uplink FIR library data
4: receive FIR library data
5: update FIR library
opt
[ ]
6: uplink command to assign new fault ID
7: receive command to assign new fault ID
8: assign new fault ID to current fault
ref
perform isolation, recovery functions

Figure 4.9 Sequence diagram for adjust software (context level)

If unusual parameter is detected as the result of analysis, further analysis will be conducted. The final goal is to investigate the root cause of that abnormal and take appropriate actions to restore the normal operation. For ADCS of MDG satellite, corrective actions will be send to satellite by uplink data to update FIR library. ADCS software has function to save all the FIR library data it received from ground to the existing text file in onboard memory, so that FIR library can be updated from ground station. There are several type of FIR library updating including modifying, adding and deleting. Depends on the situation when failure occurs, operator can decide which type of updating is the appropriate one to take.

FIR library modifying and deleting are performed in the case when the existing FIR solutions that defined in library is not effective in term of solving the problem or in case when the pre-defined faulty conditions are not the fault anymore due to components performance degradation. In those situation, FIR library is updated by replacing the existing function ID by another one, change the order of functions, or deleting the existing one if it is found to be needed.

Library adding is also divided into two actions. The first one is adding function ID to existing solution in library. This action is recommended in the situation when satellite operators find that the current existing isolation, recovery functions are not enough, there should be more functions to ensure ADCS operation can be restored. The second action, on the other hand, is the action to add new line to FIR library. This is extremely necessary in situation when unpredicted failure mode occurs. After conducting failure analysis based on data received from satellite, failure cause is identified and isolation, recovery actions are defined (of course using existing functions that are already defined in software source code). Satellite operators then need to uplink the solutions to satellite to update FIR library. New line with new fault ID and new combination of isolation, recovery functions will be added at the end of text file. However, after FIR library is updated, software has no idea what is the new data represented for, because it does not know about new fault ID. In this situation, satellite operators, one again need to send the command to assign that new fault ID to current fault or failure mode of ADCS. Once fault ID is successfully assigned, software then search again in FIR library to find correspondent isolation and recovery functions to be executed.

Functional requirement that is obtained from sequence diagram shown in Figure 4.9 is:

✧ Assign new fault ID to current fault

At this point, all three use cases of the software for FDIR of ADCS are already described using SysML sequence diagram. Seven functional requirements are obtained from those diagrams as listed after each explanation. Those functional requirements then be updated to the requirement diagram shown in Figure 4.3.

4.2.3 Software architecture

After functional requirements are defined, the next step is to develop software structure as well as identify its components to satisfy those requirements. Base on the required functions of software associated with use cases developed in previous phase, internal components of FIDR software need to be defined. Result of this process is illustrated in Figure 4.10 as SysML block definition diagram.

The software for FDIR of ADCS is decomposed into four components:

- ADCS observer software
- ADCS fault detection software
- ADCS fault identification software
- ADCS fault isolation and recovery software

Sequence diagrams then need to be constructed for each required function to verify the assumed software components in Figure 4.10 will satisfy all required functions as derived in section 4.2.2. The main purpose of this step is to confirm the defined software components can realize the functional requirements. In additional, when this activity is carrying out, ADCS team can also find out what they should be done to improve the design. For example, if the assumed components are pointed out to be cannot provide the required functions. They can go back

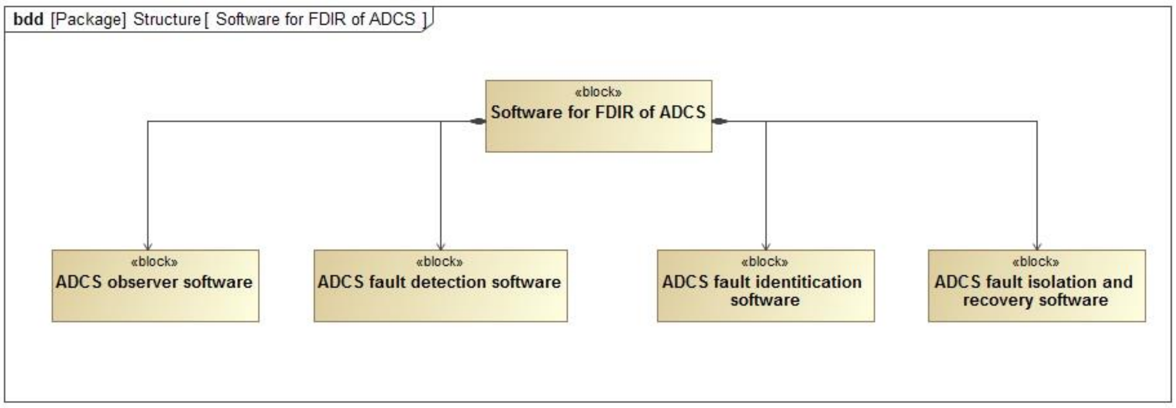

bdd [Package] Structure [Software for FDIR of ADCS]
«block»
Software for FDIR of ADCS
«block»
ADCS observer software
«block»
ADCS fault detection software
«block»
ADCS fault identitication software
«block»
ADCS fault isolation and recovery software

Figure 4.10 Block definition diagram of FDIR software

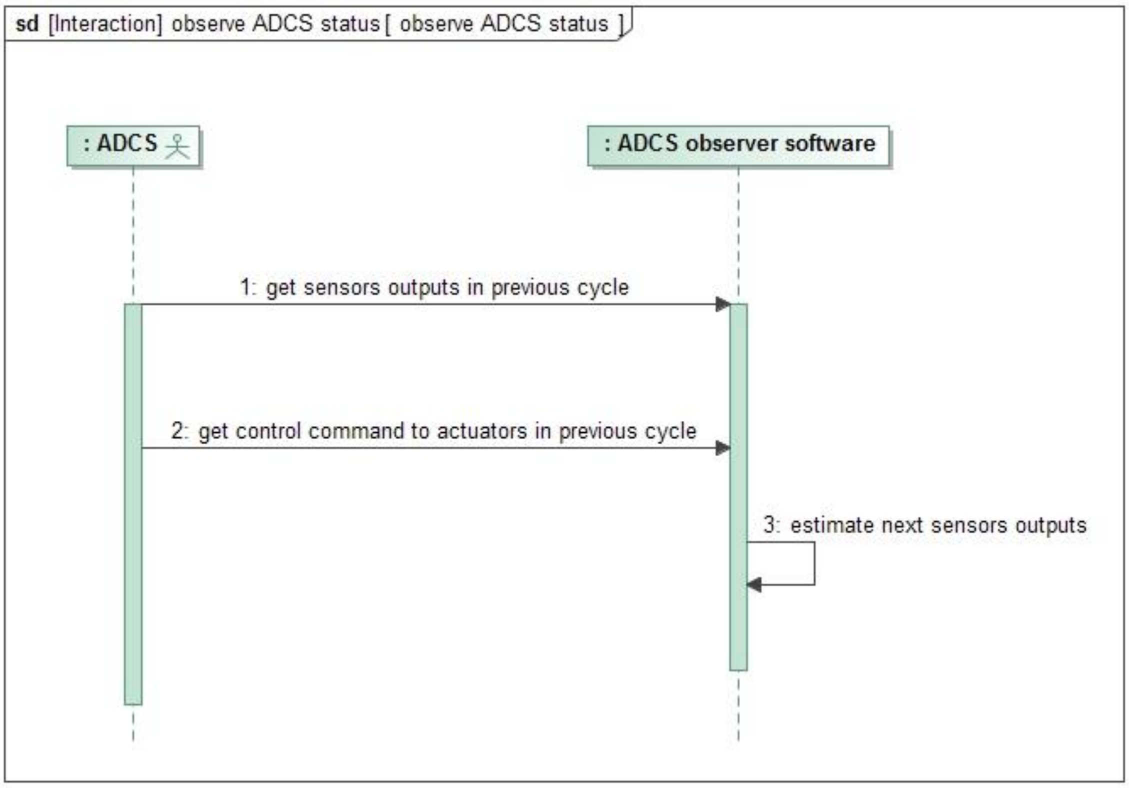

sd [Interaction] observe ADCS status [observe ADCS status]
: ADCS
: ADCS observer software
1: get sensors outputs in previous cycle
2: get control command to actuators in previous cycle
3: estimate next sensors outputs

Figure 4.11 Sequence diagram for observe ADCS status (analysis level)

and re-think about it, or try to find others. This step by step verification is conducted during the design phase of the software as well as the whole satellite system, so that the consistent of the design can be verified even during designing phase. At the end, the software is confirmed to do what it is supposed to do. Other benefits of this model – based approach are about time and cost for reworks. They can be significantly reduced.

In sequence diagrams shown in figures from Figure 4.11 to Figure 4.16, the required functions that derived from context level are described and each of those functions is decomposed into lower level. These functions, finally, are the functions which each software component represented in Figure 4.10 (by blocks) needs to perform. This process is also expected to support for the next step which is allocating functions to components. Allocation is a term used in system engineering which illustrate the process of allocating functions to components. Functions allocations of FDIR software are described clearer in Figure 4.18 in the form of SysML activity diagram.

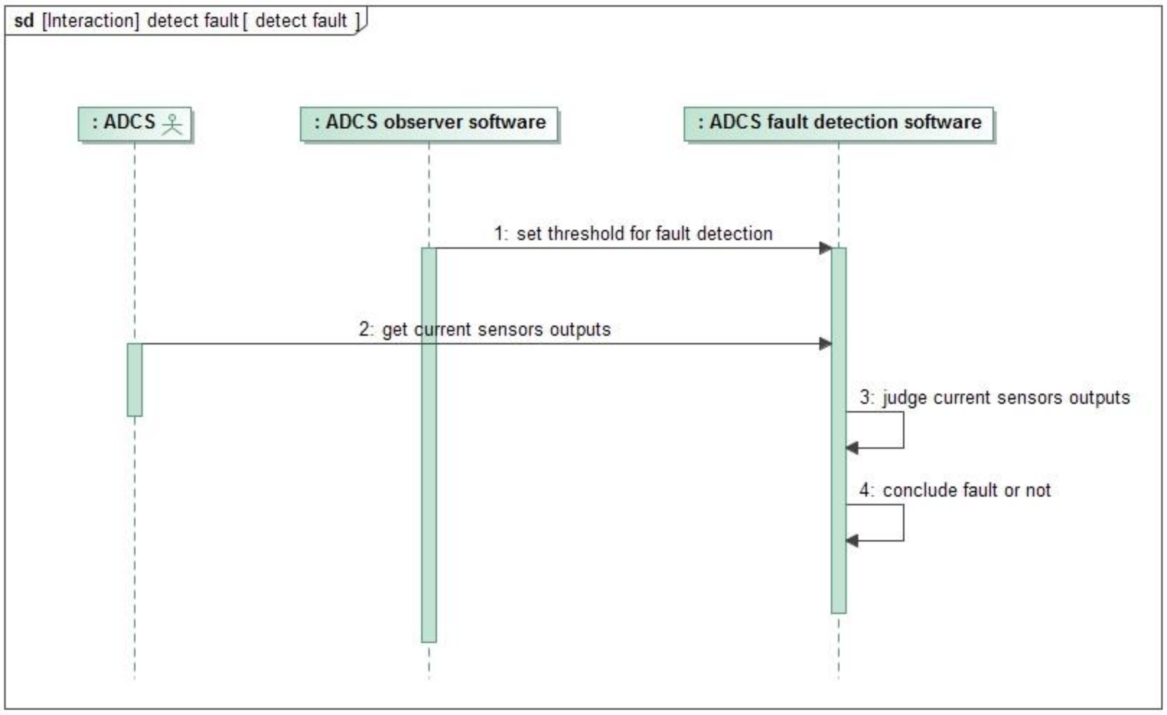

sd [Interaction] detect fault [detect fault]
: ADCS
: ADCS observer software
: ADCS fault detection software
1: set threshold for fault detection
2: get current sensors outputs
3: judge current sensors outputs
4: conclude fault or not

Figure 4.12 Sequence diagram for detect fault (analysis level)

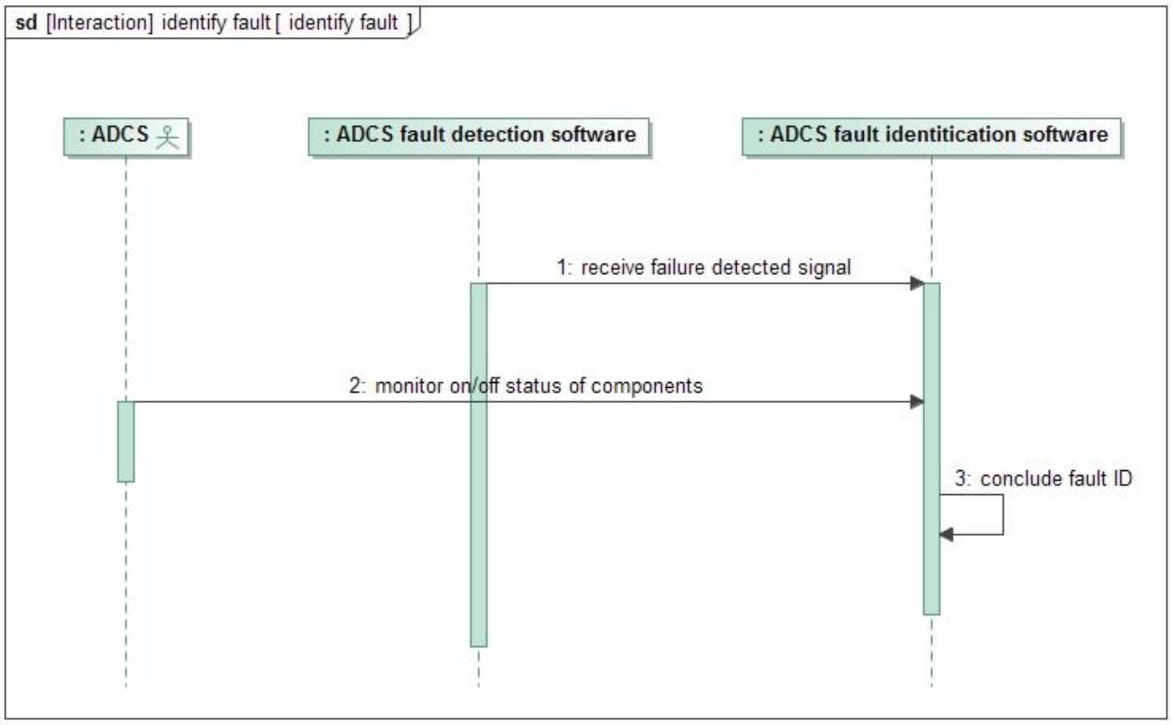

sd [Interaction] identify fault [identify fault]
: ADCS
: ADCS fault detection software
: ADCS fault identitication software
1: receive failure detected signal
2: monitor on/off status of components
3: conclude fault ID

Figure 4.13 Sequence diagram for identify fault (analysis level)

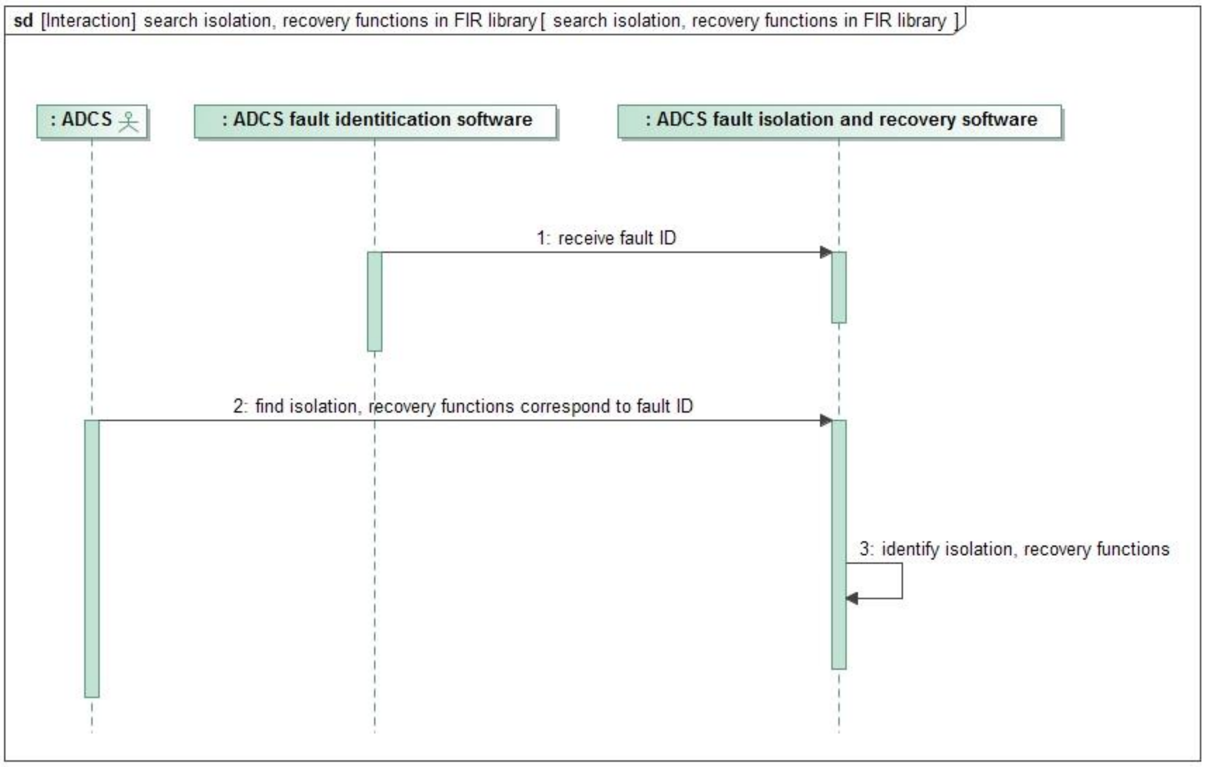

sd [Interaction] search isolation, recovery functions in FIR library [search isolation, recovery functions in FIR library]
: ADCS
: ADCS fault identitication software
: ADCS fault isolation and recovery software
1: receive fault ID
2: find isolation, recovery functions correspond to fault ID
3: identify isolation, recovery functions

Figure 4.14 Sequence diagram for search isolation, recovery functions in FIR library (analysis level)

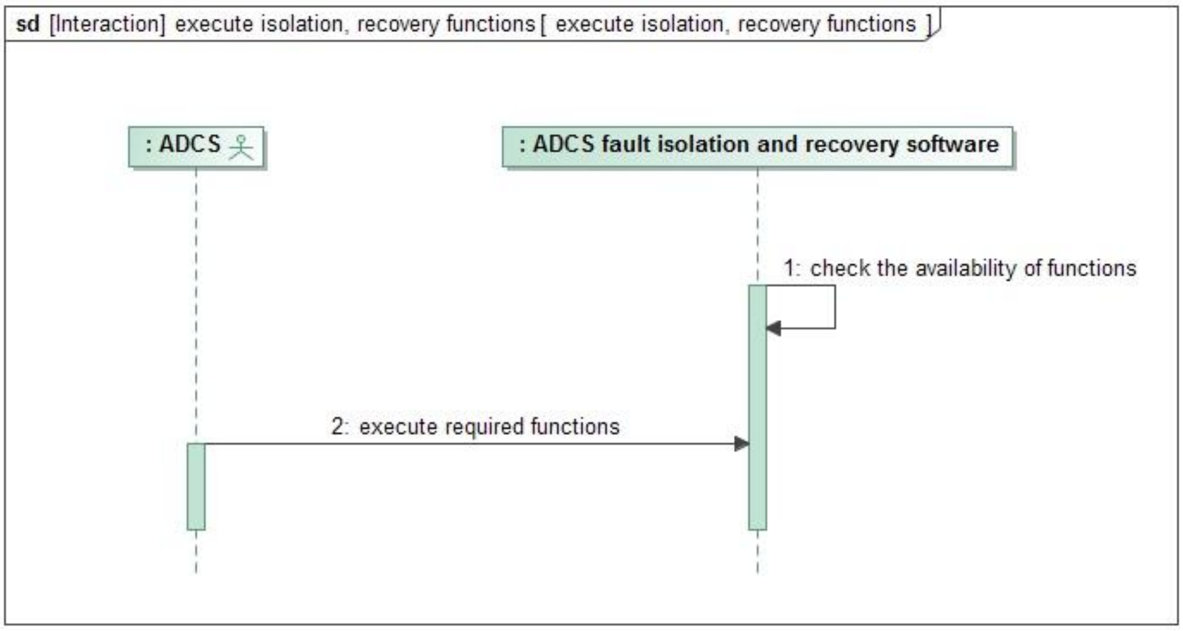

sd [Interaction] execute isolation, recovery functions [execute isolation, recovery functions]
: ADCS
: ADCS fault isolation and recovery software
1: check the availability of functions
2: execute required functions

Figure 4.15 Sequence diagram for execute isolation, recovery functions (analysis level)

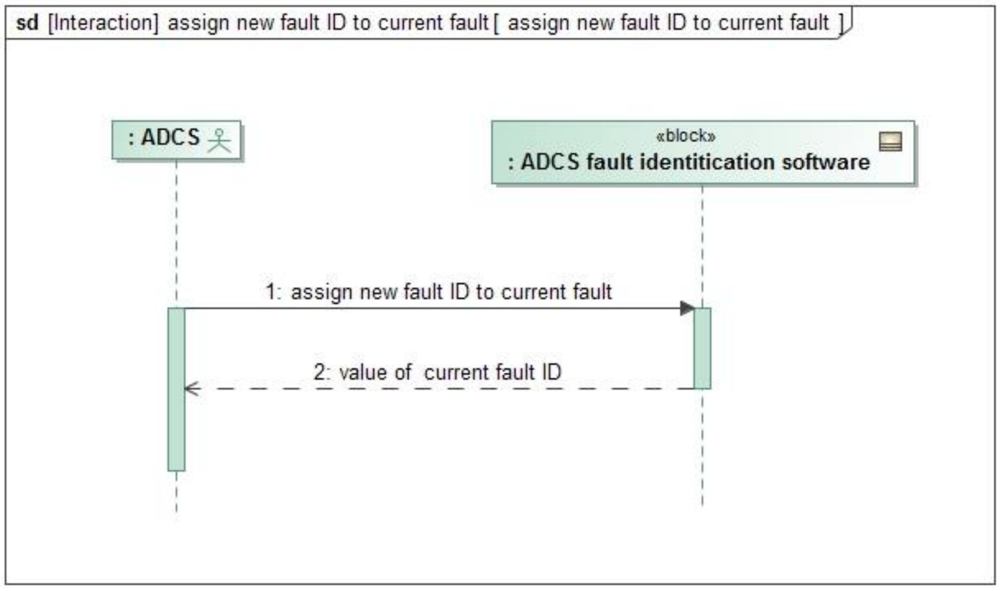

sd [Interaction] assign new fault ID to current fault [assign new fault ID to current fault]
: ADCS
«block»
: ADCS fault identitication software
1: assign new fault ID to current fault
2: value of current fault ID

Figure 4.16 Sequence diagram for assign new fault ID to current fault (analysis level)

Once the assumed components are confirmed to be able to realize functions, it is also necessary to model the interconnections between them. Interconnections and interfaces between all four components of the software are defined in Figure 4.17. This diagram also shows the connections of software with external system which is ADCS in this case.

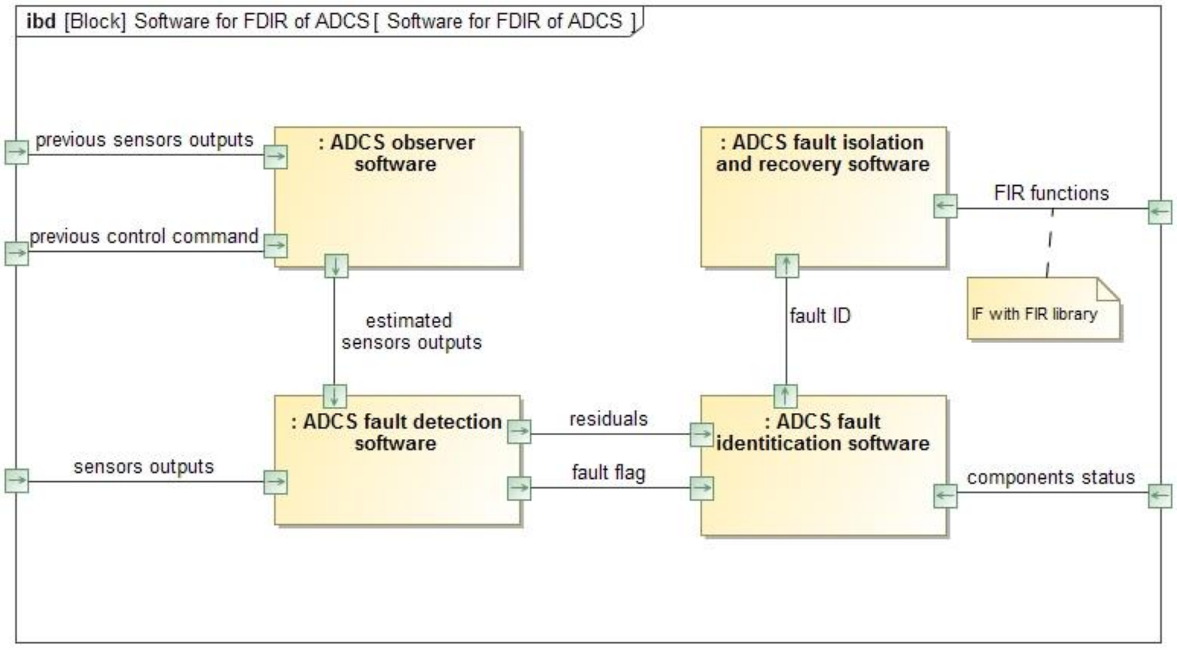

ibd [Block] Software for FDIR of ADCS [Software for FDIR of ADCS]
previous sensors outputs
previous control command
: ADCS observer software
: ADCS fault isolation and recovery software
FIR functions
IF with FIR library
estimated sensors outputs
fault ID
: ADCS fault detection software
residuals
fault flag
: ADCS fault identitication software
sensors outputs
components status

Figure 4.17 Internal block diagram of FDIR software

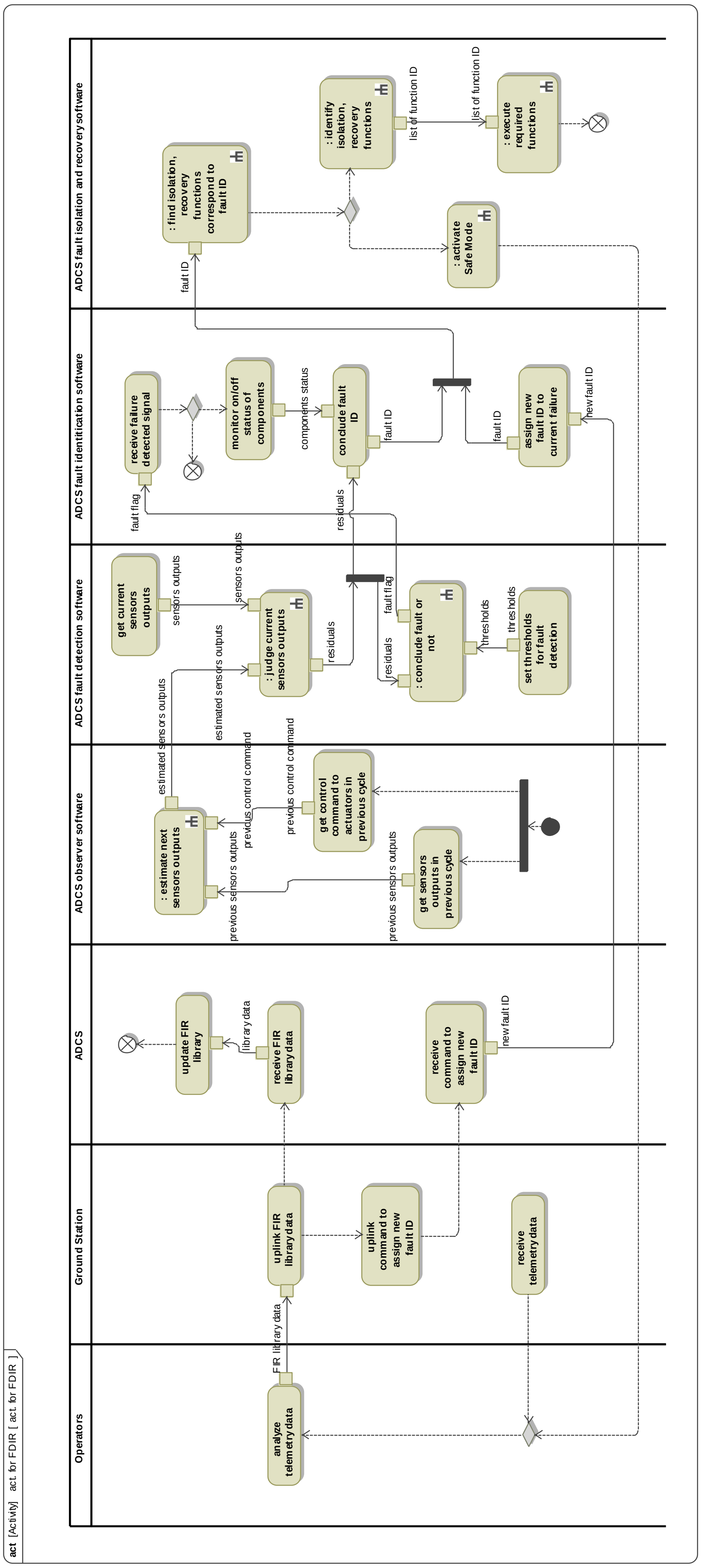

act [Activity] act. for FDIR [act. for FDIR]
Operators
Ground Station
ADCS
ADCS observer software
ADCS fault detection software
ADCS fault identitication software
ADCS fault isolation and recovery software
analyze telemetry data
FIR library data
uplink FIR library data
uplink command to assign new fault ID
receive telemetry data
receive FIR library data
library data
update FIR library
receive command to assign new fault ID
new fault ID
: estimate next sensors outputs
get control command to actuators in previous cycle
get sensors outputs in previous cycle
previous sensors outputs
previous control command
estimated sensors outputs
get current sensors outputs
sensors outputs
: judge current sensors outputs
residuals
: conclude fault or not
fault flag
thresholds
set thresholds for fault detection
receive failure detected signal
monitor on/off status of components
components status
conclude fault ID
fault ID
assign new fault ID to current failure
: find isolation, recovery functions correspond to fault ID
: identify isolation, recovery functions
list of function ID
: execute required functions
: activate Safe Mode

Figure 4.18 Activity diagram for FDIR of ADCS

The activity diagram shown in Figure 4.18 not only describes interactions between software components but it also represents the flow of actions or functions of the software. According to Ref. [29], it is very similar to the traditional function flow block diagram (FFBD). The process of fault detection, isolation and recovery starts with the function of ADCS observer software to get sensors outputs and control commands from previous cycle (each cycle is 50ms). The collected data then be used to calculate and estimate the values of next sensors outputs. Depends on ADCS mode, the software get specific sensors values and estimates the future values output from those sensors. For example, in target pointing mode, sensors to be used to determine current attitude of satellite include STT and FOG which output satellite rotational rate and quaternion. The software then needs to estimate outputs from those sensors. The principle how that function could be performed is shown in Figure 4.19.

The estimate outputs then be received by ADCS fault detection software along with the current outputs from sensors to generate residuals. Process of residuals generating is presented in Figure 4.20. As mentioned in section 4.1.2, the residuals that generated as the results of this will be used to detect failure by comparing those values to thresholds. The designed software also has ability to set thresholds for fault detection since they are adaptable.

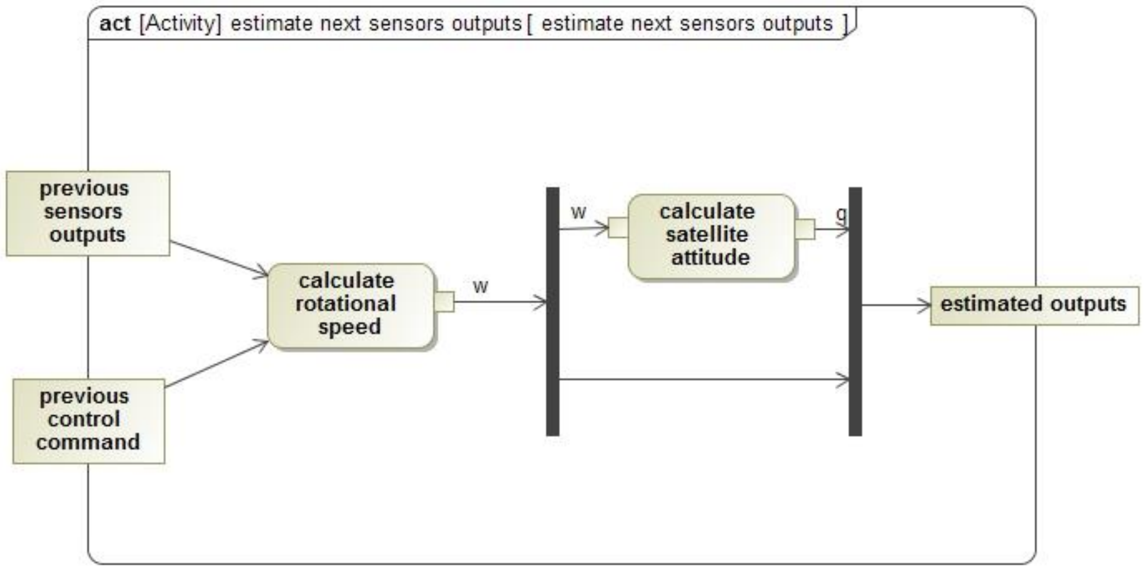

act [Activity] estimate next sensors outputs [estimate next sensors outputs]
previous sensors outputs
previous control command
calculate rotational speed
w
w
calculate satellite attitude
q
estimated outputs

Figure 4.19 Activity diagram for estimate next sensors outputs

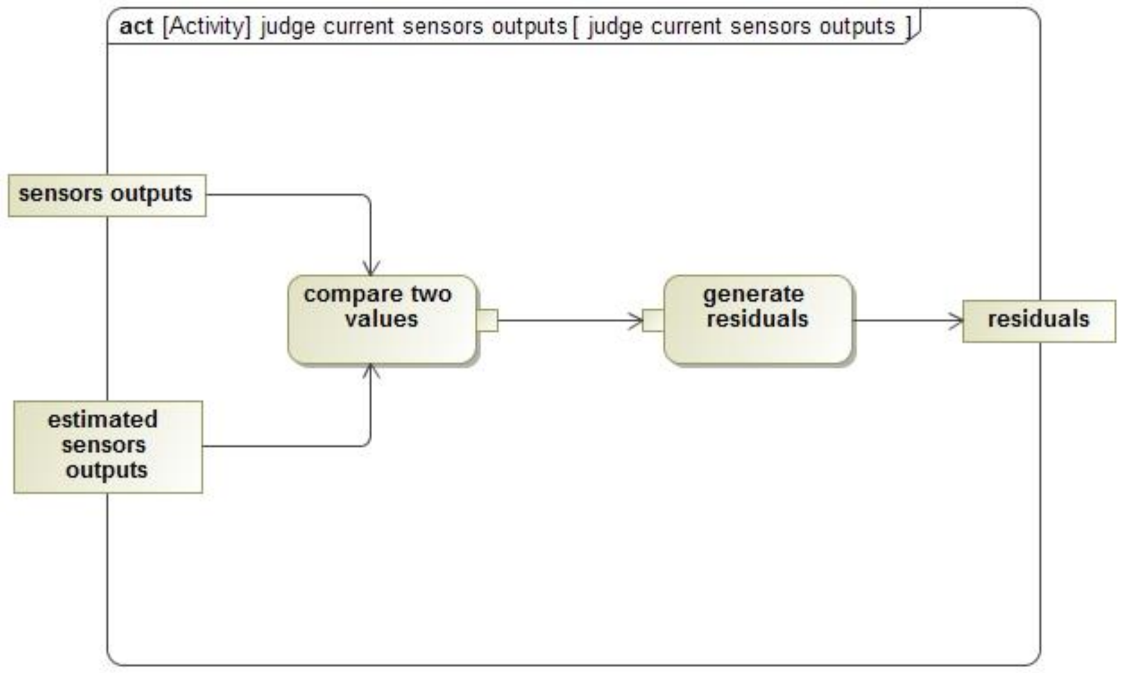

act [Activity] judge current sensors outputs [judge current sensors outputs]
sensors outputs
compare two values
generate residuals
residuals
estimated sensors outputs

Figure 4.20 Activity diagram for judge current sensors outputs

If fault is detected, then a fault flag will be set in the software, by assigning the true value for a Boolean variable. If not, a false value will be set. This process is carried out by the function to conclude failure or not in the software (Figure 4.21).

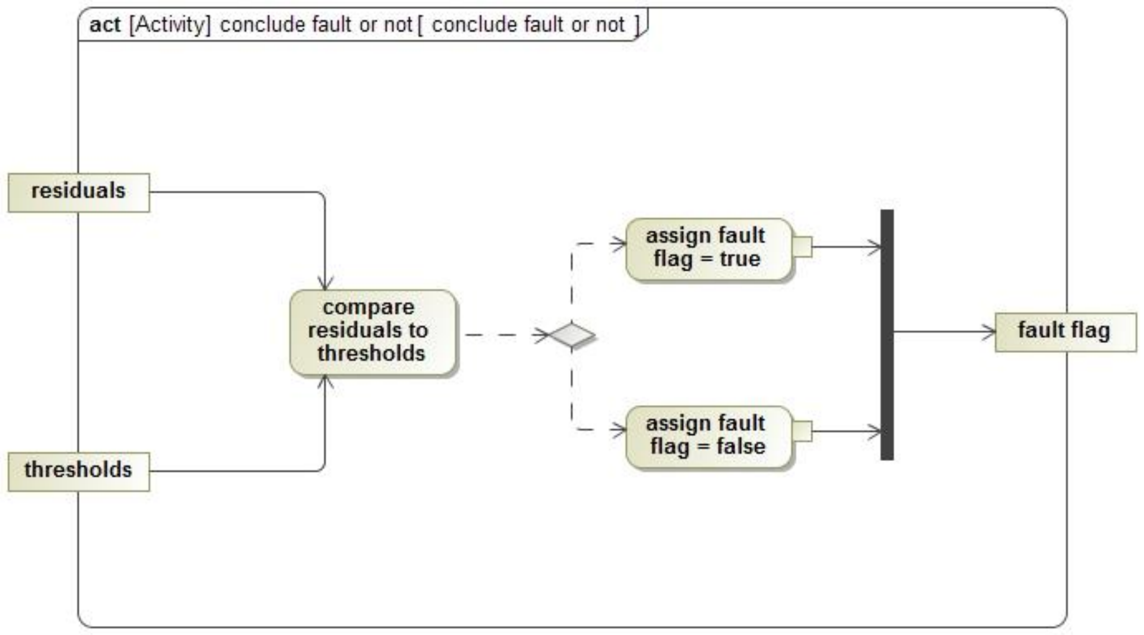

act [Activity] conclude fault or not [conclude fault or not]
residuals
compare residuals to thresholds
assign fault flag = true
assign fault flag = false
fault flag
thresholds

Figure 4.21 Activity diagram for conclude fault or not

The fault flag is then used in ADCS fault identification software as the condition to call out its functions. If the flag is set to true, this software then execute the function to monitor on/off status of components in ADCS. Similar to observer software, this function will monitor status of components respected in ADCS mode. The results of this and information from generated residuals then be used to give the conclusion about faulty component in form of a fault ID if it is predicted beforehand.

When a fault is identified and a fault ID is given, ADCS fault isolation and recovery software will has responsibility to receive that ID then find the correspondence one in FIR library. After fault ID is found in library, the software finds appropriate action as are defined in FIR library to perform. The process of this function is described in Figure 4.22. Output of this function is a text line (or string) which consist of isolation and recovery functions ID. This string then be used in the next function in sequence to identify the required functions to be executed. Figure 4.23 shows what need to be done in the software to perform this function. List of separated isolation and recovery function ID will be outputted as the result when the function is successfully executed. Figure 4.23 also describes the flow of this function. Isolation and recovery functions as string is split up into isolation functions and recovery functions. List of required function ID as in array then can be outputted as the result of this action.

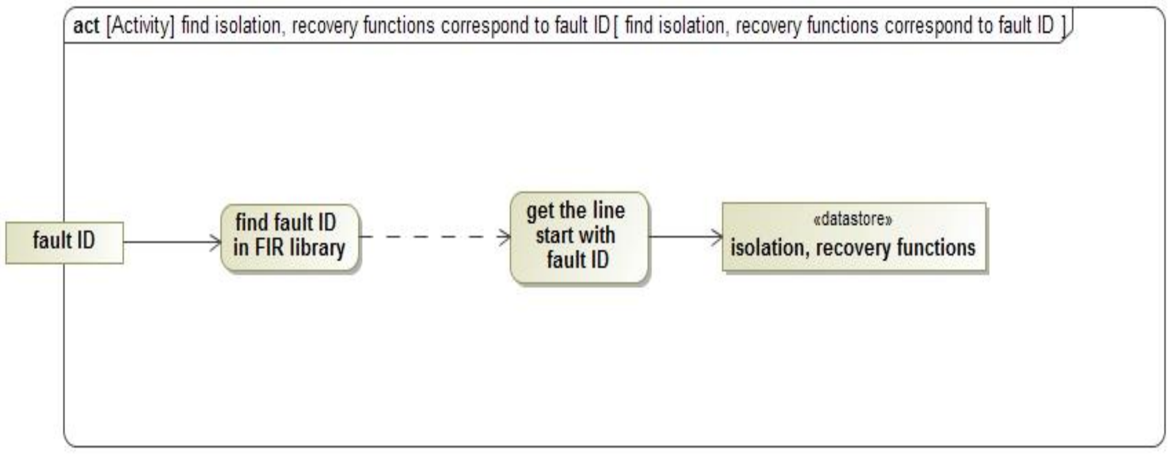

act [Activity] find isolation, recovery functions correspond to fault ID [find isolation, recovery functions correspond to fault ID]
fault ID
find fault ID in FIR library
get the line start with fault ID
«datastore»
isolation, recovery functions

Figure 4.22 Activity diagram for find isolation, recovery functions in FIR library

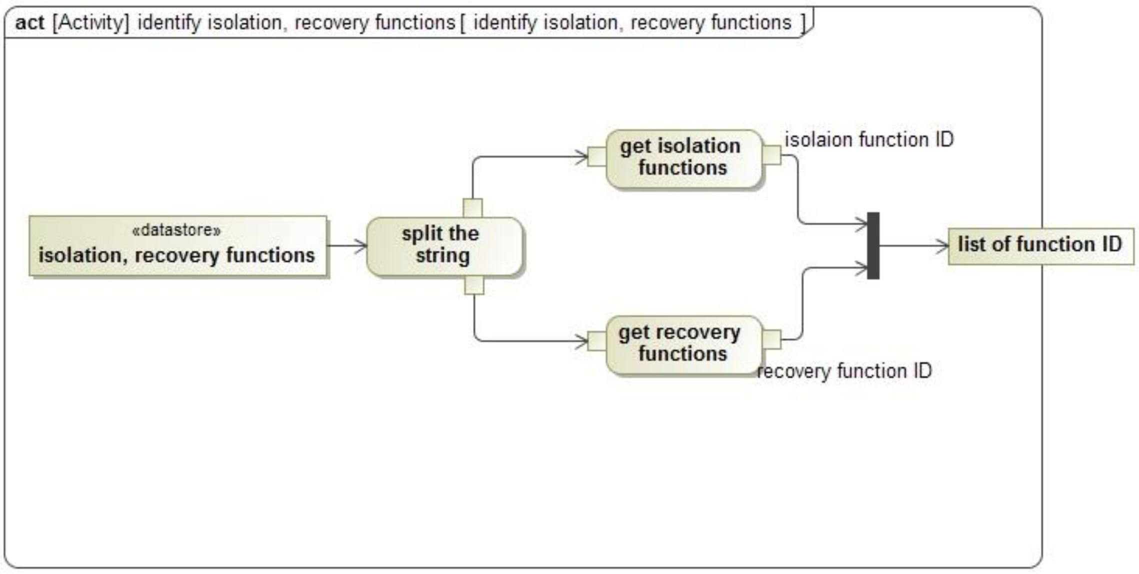


Figure 4.23 Activity diagram for identify isolation, recovery functions

List of required function IDs then become input for the next function: execute required functions. As shown in Figure 4.24, required functions will be executed one by one until all of them are done. The process of isolation, recovery now is successfully performed onboard.

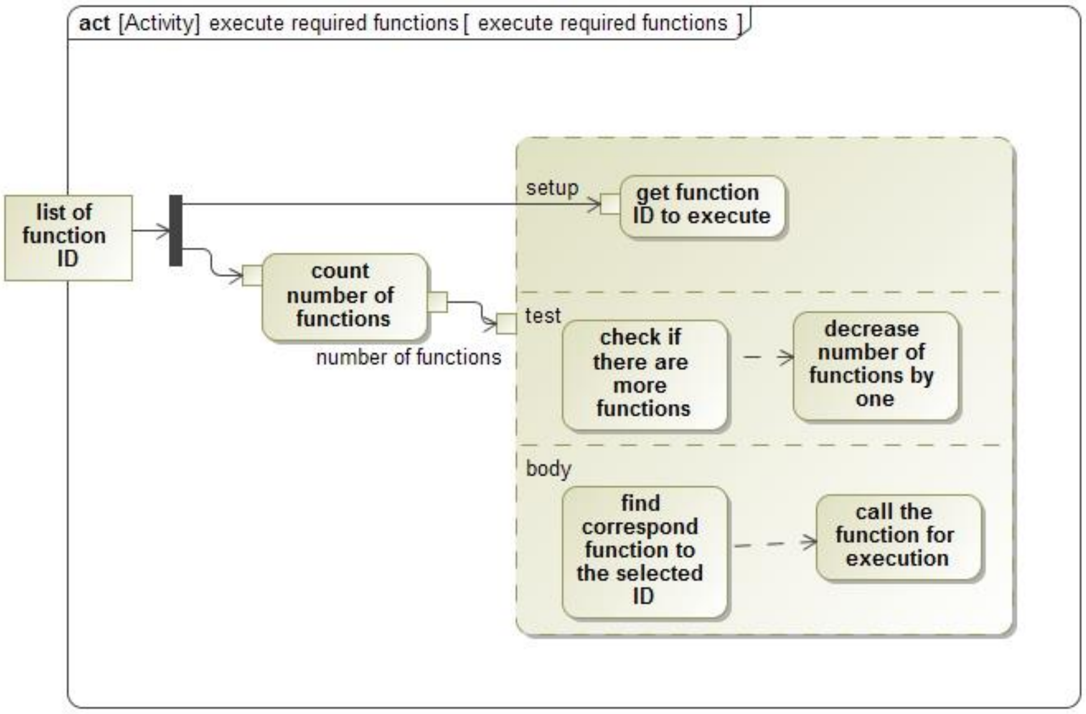

act [Activity] execute required functions [execute required functions]
list of function ID
count number of functions
number of functions
setup
get function ID to execute
test
check if there are more functions
decrease number of functions by one
body
find correspond function to the selected ID
call the function for execution

Figure 4.24 Activity diagram for execute required functions

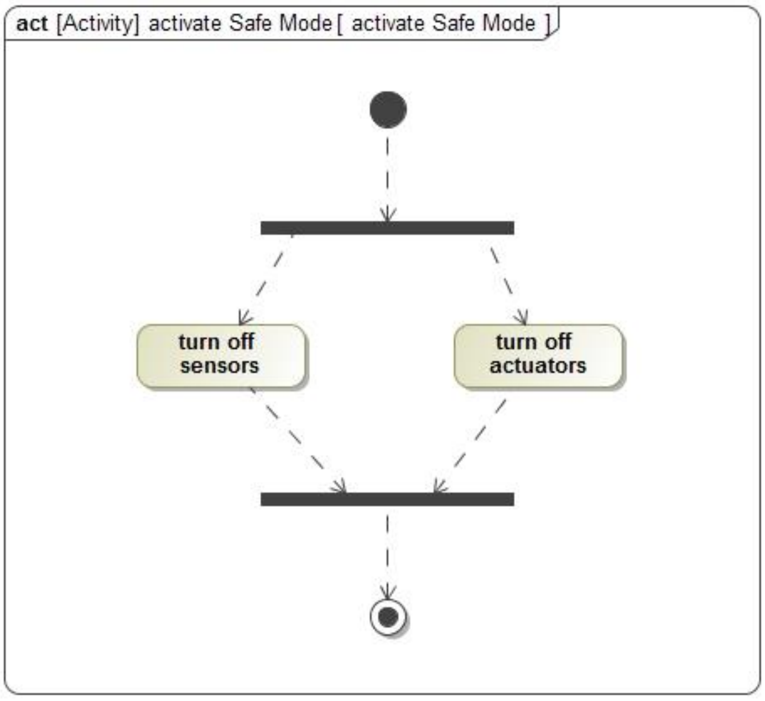


Figure 4.25 Activity diagram for activate Safe Mode

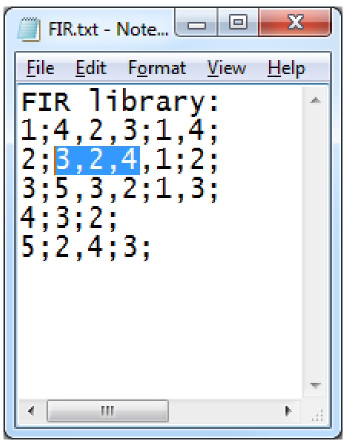
In the case when isolation, recovery functions are not found in FIR library, the software then has a function to activate ADCS Safe Mode. As already explained in section 3.2.1 of Chapter 3, in Safe Mode, all components including sensors and actuators will be turned off, except for OBC. Function flow is described in Figure 4.25. The role of FDIR now be given to satellite operators on ground station side.

Ground station has function to receive telemetry data generated from ADCS as well as all satellite subsystem to monitor satellite state. Telemetry data is extremely helpful in situations when failure occurs. Base on received data with supports from ground station facilities and experiences, failure analyses are conducted and solutions must be pointed out. The operator then need to identify isolation, recovery functions ID and the right order of them in order to correct the failure. List of isolation and recovery functions then will be sent from ground station so that ADCS can receive and update FIR library. Three types of library updating are mentioned in previous section of this chapter. Depends on the actual situation, operators must choose the right type to update. Failure mode also decides the need of command to assign fault ID. If it is known for FDIR software, command is not necessary. However, if failure is un-predicted, after updating FIR library, sending command to assign new fault ID to that failure is needed.

4.2.3.1 *Three types of adjustments and their applications*

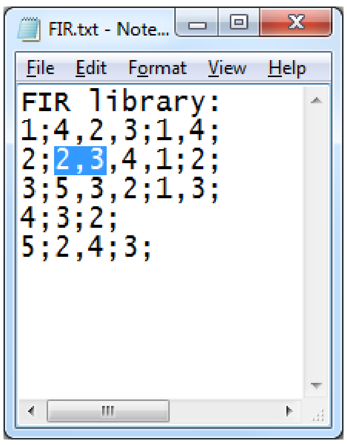
Corresponding to each type of update in FIR library, there is a set software adjustments and each of them has their own applications which share the final goal - to support for FDIR mechanism to work more effective. Having the software with ability to be adjusted on board, operators on ground can have more degree of freedom when dealing with failures of ADCS.

The first type of software adjustments is swap/change functions. This type is associated with FIR library modifying. As mentioned in the previous section, FIR library modification is required in case when existing isolation, recovery functions are found to be not effective in actual operation of satellite in orbit. Example of library modification is shown in Figure 4.26. By analyzing received telemetry, engineers or operator on ground find out that, the existing functions, such as isolation ID number 2, 3 are not in the right order. They send command to change that order, for example reverse order of those functions to 3, 2 (see Figure 4.26 b). Another example of FIR library modifications is changing functions. It is required when functions are found that not in right places and needed to be replaced or changed by other one. Figure 4.26 c) shows an example of this modification type.



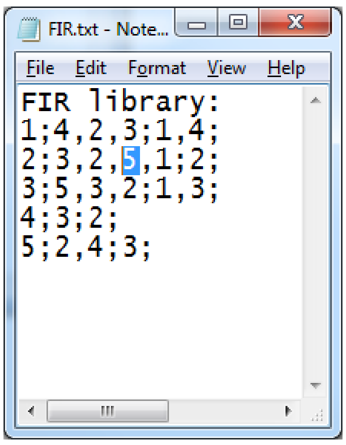
```
FIR library:
1;4,2,3;1,4;
2;3,2,4,1;2;
3;5,3,2;1,3;
4;3;2;
5;2,4;3;
```

a) Original library



```
FIR library:
1;4,2,3;1,4;
2;2,3,4,1;2;
3;5,3,2;1,3;
4;3;2;
5;2,4;3;
```

b) After swapping



```
FIR library:
1;4,2,3;1,4;
2;3,2,5,1;2;
3;5,3,2;1,3;
4;3;2;
5;2,4;3;
```

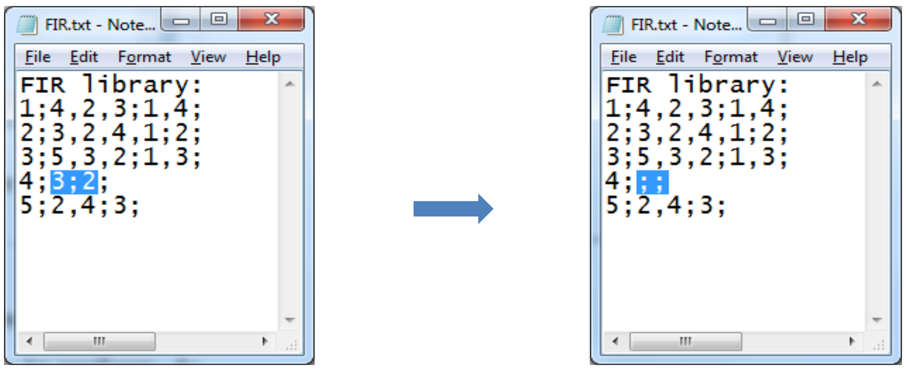
c) After changing

Figure 4.26 Examples of FIR library modification

Once FIR library is modified, every time when ADCS is facing with the same failures, ADCS isolation and recovery software can find the modified solutions in library to perform. As results, the software now has ability to swap functions orders as well as change functions.

Second type of software adjustment is functions eliminating. This type, basically is very similar to the first type and corresponding to deleting type in FIR library (an example is in Figure 4.27). If an isolation or recovery function ID is deleted in FIR library. The software then reacts in the same way to eliminate that function when a fault is detected. This type of software adjustment is critical in case of component performance degradation. For example, according to specification, FOG (Fiber Optic Gyroscope) has output noise less than 0.01 degree per second, so that a fixed threshold is set for detect fault of FOG if it outputs values out range. However, after a certain time of operation in orbit, due to effects from space environment (such as radiation), FOG frequently outputs values with noise magnitude out of range. In this case, the defined fault is not the actual fault any more so there is no isolation and recovery function is required. The existing functions in FIR library need to be deleted. Software then do nothing when that specified failure mode is detected.

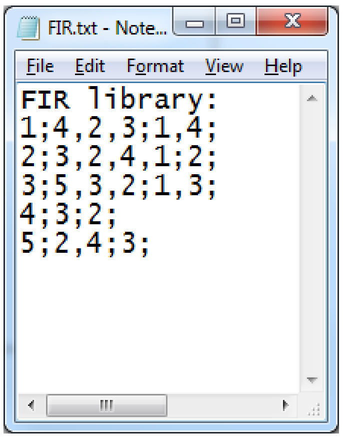
Functions adding is the third type of software adjustments which is divided in to two smaller actions. As mentioned in previous section, operators on ground can add one or more functions ID to existing line corresponding to a fault ID. Because of that, software then has ability to automatically call those functions to execution when failure mode is identified. Example if adding functions to existing failure mode is in Figure 4.28 b).



```
FIR library:
1;4,2,3;1,4;
2;3,2,4,1;2;
3;5,3,2;1,3;
4;3;2;
5;2,4;3;
```

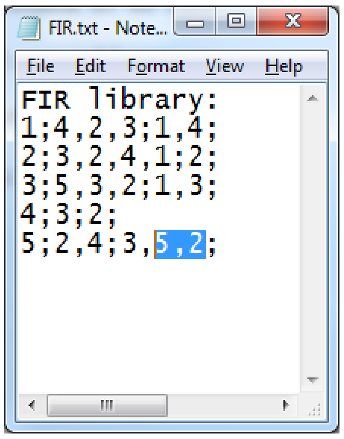
```
FIR library:
1;4,2,3;1,4;
2;3,2,4,1;2;
3;5,3,2;1,3;
4;;;
5;2,4;3;
```

Figure 4.27 Example of FIR functions deleting



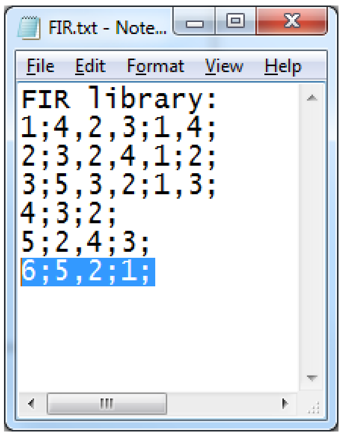
```
FIR library:
1;4,2,3;1,4;
2;3,2,4,1;2;
3;5,3,2;1,3;
4;3;2;
5;2,4;3;
```

a) Original library



```
FIR library:
1;4,2,3;1,4;
2;3,2,4,1;2;
3;5,3,2;1,3;
4;3;2;
5;2,4;3,5,2;
```

b) Functions added to existing line



```
FIR library:
1;4,2,3;1,4;
2;3,2,4,1;2;
3;5,3,2;1,3;
4;3;2;
5;2,4;3;
6;5,2;1;
```

c) New line is added

Figure 4.28 Example of adding FIR functions

The second action, on the other hand, is the action to add new line to FIR library. This is extremely necessary in situation when unpredicted failure mode occurs. When a new line with new fault ID and new combination of isolation, recovery functions is added at the end of text file, the software then can be adjusted to be able to identify the new fault ID and find the corresponded isolation and recovery functions to perform. Figure 4.28 c) shows an example when new line is added which contents a new fault ID 6 and isolation functions ID 5, 2 and recovery function ID 1. However, in this situation, FDIR software does not have any information about fault ID 6, so that it cannot perform isolation and recovery function automatically. This requires operators on ground station to send command to assign that fault ID to current fault. The software then searches again in FIR library to find appropriate actions to execute. Now, isolation function with ID 5, 2, and recovery function 1 can be executed onboard.

To this point, the design of adjustable software for fault detection, isolation and recovery of ADCS in MDG satellite is explained. With the introduction of an updateable FIR library and ability to be adjusted, the design of FDIR software is expected satisfy all the requirements as defined in section 4.2.1 and be able to handle both known and un-known failure modes, and ensure the safety operation of ADCS.

4.2.4 Traceability of requirements

Traceability of requirements for software of FDIR is shown in Figure 4.29 by SysML requirement diagram. Requirements relationships such as derivation, satisfaction are used to support this process.

Process of maintaining requirements traceability supports ADCS team to confirm that the design of software satisfies all the requirements identified in context analysis phase. Functional requirements are derived from system requirements for the software and they are all satisfied by software components. For example, functional requirements observe ADCS status and detect fault are derived from requirement for auto fault detection of the software. Then they are satisfied by two software components including ADCS observer software and ADCS fault detection software.

It is very important to maintain requirements traceability during design phase so that early verification and validation can be carried out from early phase of system development. In process of software design for FDIR of ADCS using SysML that explained earlier in this chapter, traceability of requirements is also ensured by step-by-step refinement during modeling. Model – based design also helps to improve and optimize the design of software, problems can be identified and corrected during modeling so that time and cost for reworking can be significantly reduced.

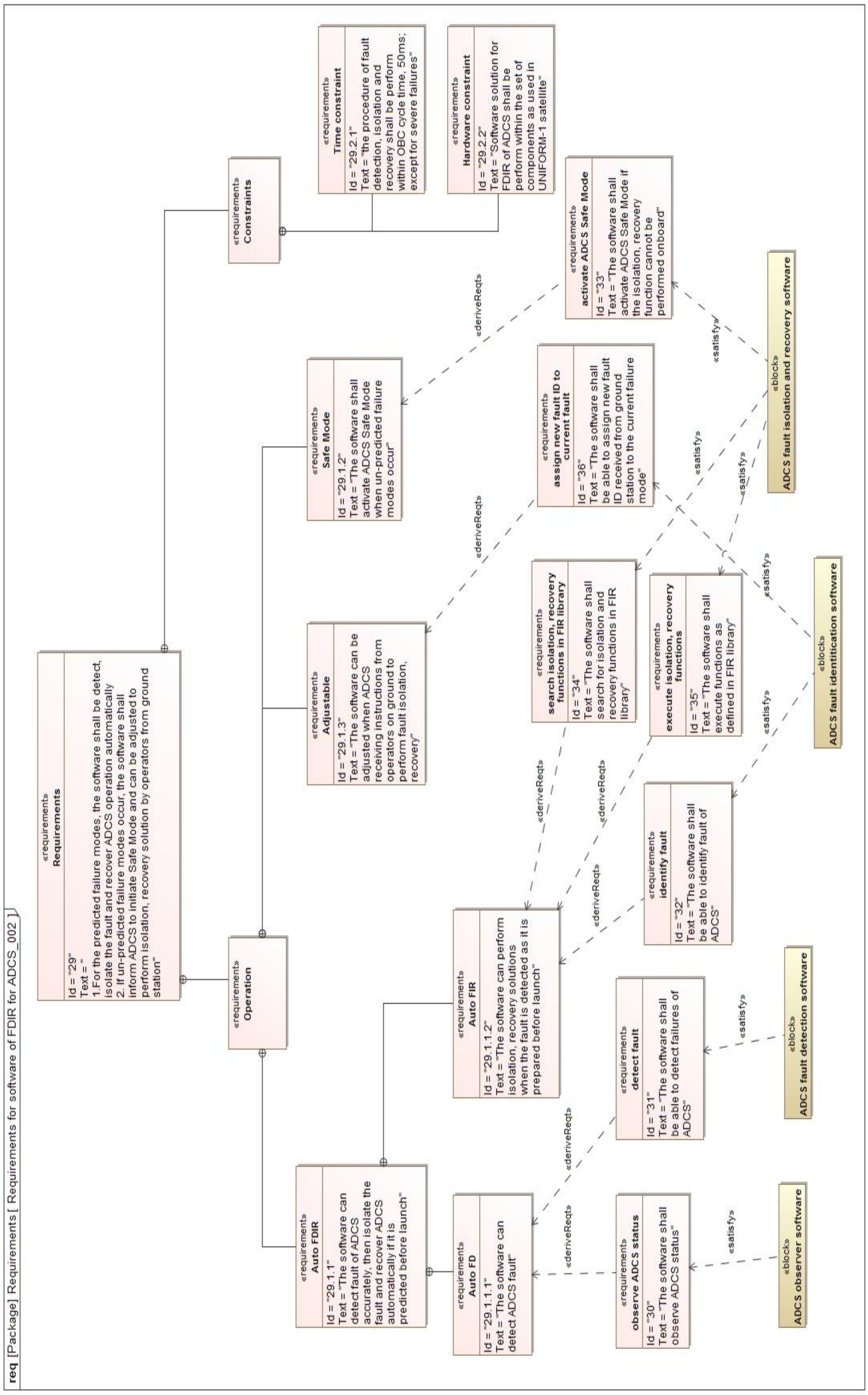

req [Package] Requirements [Requirements for software of FDIR for ADCS_002]
«requirement» Requirements
Id = "29"
Text = "
1.For the predicted failure modes, the software shall be detect, isolate the fault and recover ADCS operation automatically
2. If un-predicted failure modes occur, the software shall inform ADCS to initiate Safe Mode and can be adjusted to perform isolation, recovery solution by operators from ground station"
«requirement» Constraints
«requirement» Time constraint
Id = "29.2.1"
Text = "the procedure of fault detection, isolation and recovery shall be perform within OBC cycle time, 50ms; except for severe failures"
«requirement» Hardware constraint
Id = "29.2.2"
Text = "Software solution for FDIR of ADCS shall be perform within the set of components as used in UNIFORM-1 satellite"
«requirement» Operation
«requirement» Safe Mode
Id = "29.1.2"
Text = "The software shall activate ADCS Safe Mode when un-predicted failure modes occur"
«requirement» Adjustable
Id = "29.1.3"
Text = "The software can be adjusted when ADCS receiving instructions from operators on ground to perform fault isolation, recovery"
«requirement» activate ADCS Safe Mode
Id = "33"
Text = "The software shall activate ADCS Safe Mode if the isolation, recovery function cannot be performed onboard"
«requirement» assign new fault ID to current fault
Id = "36"
Text = "The software shall be able to assign new fault ID received from ground station to the current failure mode"
«requirement» search isolation, recovery functions in FIR library
Id = "34"
Text = "The software shall search for isolation and recovery functions in FIR library"
«requirement» execute isolation, recovery functions
Id = "35"
Text = "The software shall execute functions as defined in FIR library"
«requirement» Auto FDIR
Id = "29.1.1"
Text = "The software can detect fault of ADCS accurately, then isolate the fault and recover ADCS automatically if it is predicted before launch"
«requirement» Auto FIR
Id = "29.1.1.2"
Text = "The software can perform isolation, recovery solutions when the fault is detected as it is prepared before launch"
«requirement» Auto FD
Id = "29.1.1.1"
Text = "The software can detect ADCS fault"
«requirement» identify fault
Id = "32"
Text = "The software shall be able to identify fault of ADCS"
«requirement» detect fault
Id = "31"
Text = "The software shall be able to detect failures of ADCS"
«requirement» observe ADCS status
Id = "30"
Text = "The software shall observe ADCS status"
«block» ADCS fault isolation and recovery software
«block» ADCS fault identification software
«block» ADCS fault detection software
«block» ADCS observer software
«deriveReqt»
«satisfy»

Figure 4.29 Traceability of requirements

Chapter 5. Verification and Validation

5.1 Demonstration of Software Adjustment

5.1.1 Demonstration setup

Demonstrations are conducted in order to verify the design. A prototype of software for FDIR as well as attitude control is developed and implemented on a micro controller board called Mbed LPC1768.

A hardware in the loop simulation (HILS) environment is set up for purpose of demonstration. Overview of the simulation setup is shown in Figure 5.1. To run the simulation, two computers and one electronics board are needed.

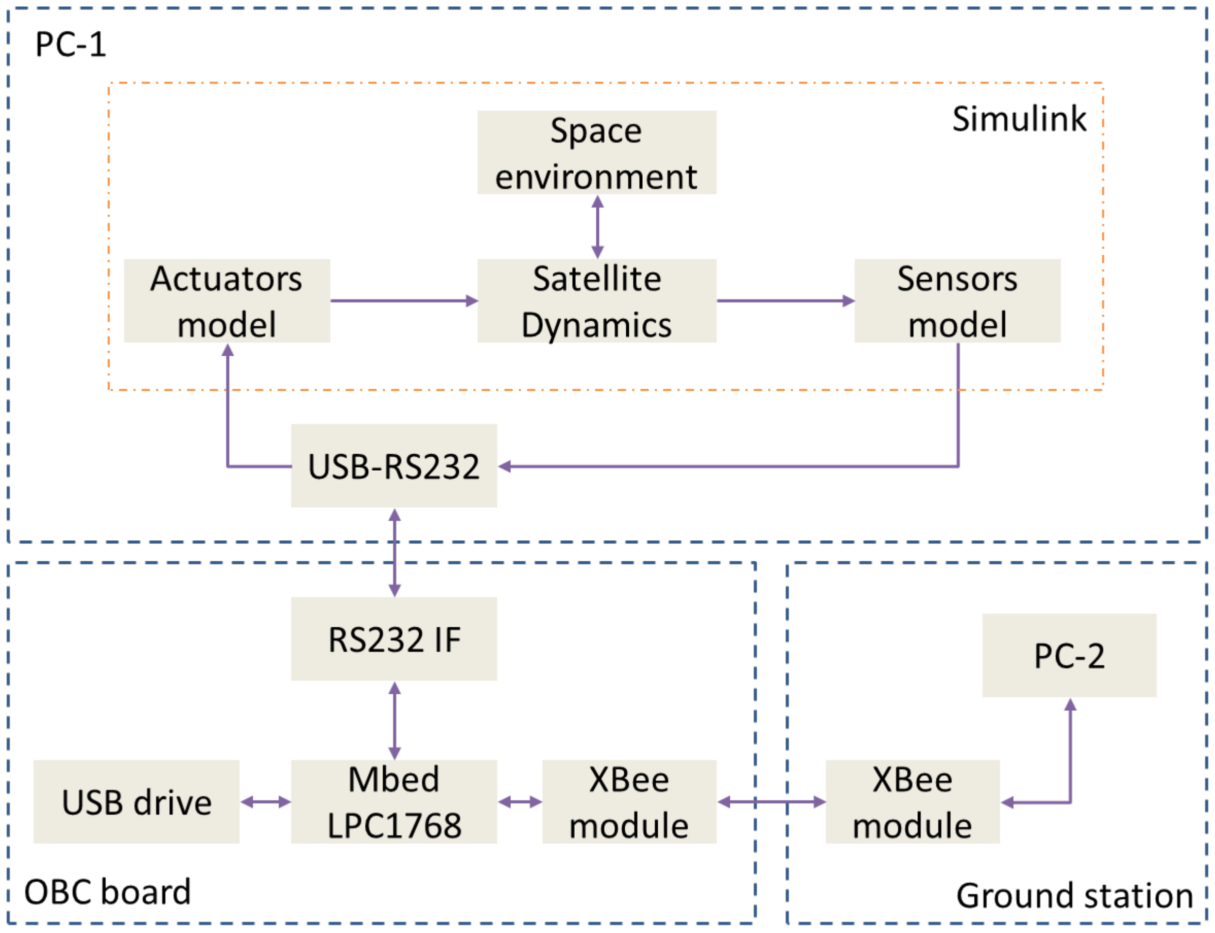


Figure 5.1 Configuration of HILS

A Simulink model of ADCS (shown in Figure 5.3) is developed and run on PC-1, which is shown in the upper part of Figure 5.1. The model can be divided into four smaller model. Actuators model (or block) consists of three reaction wheels with functions to generate required torque as commanded from OBC. The generated torque from actuators then be the inputs for satellite dynamics block. In this block, attitude of satellite as well as angular rotational rate is calculated by solving the kinematic and dynamic equations of motion of a rigid body:

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

$$T = I\dot{\omega} + \omega \times (I\omega + h_{RW})$$

Where q is satellite attitude represented in quaternion,

ω is angular rotational rate,

T is generated torque,

I is satellite moment of inertia,

and h_{RW} is reaction wheel momentum.

Space environment block consists of an Earth model and a satellite orbit model. Satellite position in orbit and target attitude of ADCS is calculated from outputs of this block. Sensor model in this Simulink model is very simple with one gyroscope and one star tracker. Noises are added to the input signal to simulate the sensors outputs.

Gyroscope model:

$$\omega_m = \omega_t + w_1 + \dot{r}$$
$$\dot{r} = w_2$$

Where:

ω_m is gyroscope output,

ω_t true value of satellite rotational rate,

r is random walk,

w_1, w_2 are white Gaussian noise with $1\sigma = 0.01$ and 0.001 respectively.

Star tracker model:

$$q_m = q_t + v$$

Where:

q_m is star tracker output

q_t is true value of quaternion

v is white Gaussian noise with $1\sigma = 0.001$

On ground station side, there is a computer (PC-2). A free software for serial communication is installed on this computer. All the interactions with OBC from ground station are made via this software interface. An XBee module is connected to PC-2 via RS232 interface and plays the role of a transceiver. For purpose of demonstration, this wireless transceiver can be referred as the real transceiver of ground station. The computer PC-2 is also used for the debugging purpose. All the information generated from software will be transmitted to this computer and displayed on screen.

As mentioned above, software prototype (see appendix) is implemented in an Mbed LPC1768 board. This board is developed based on a 32-bit ARM cortex M3 microprocessor. However, compare to the real OBC on MicroDragon satellite, performance of this board is much lower. Control cycle is set to 100 ms (millisecond) instead of 50 ms as in real OBC. As shown in the bottom left of Figure 5.1, there also a USB drive connected to the Mbed. FIR library is saved in this drive. Another XBee module is implemented to represent for communication transceiver inside the satellite and do the communication jobs with ground station. Data from OBC board will be transmitted to ground station via this XBee module. Commands and data from ground station will be transmitted via XBee module connected to PC-2 and received by XBee module on this OBC board.

To verify the design of software, two types of demonstrations are made: demonstration of the adjustability of software and demonstration of FDIR functions. Results of these are shown in the next sections.

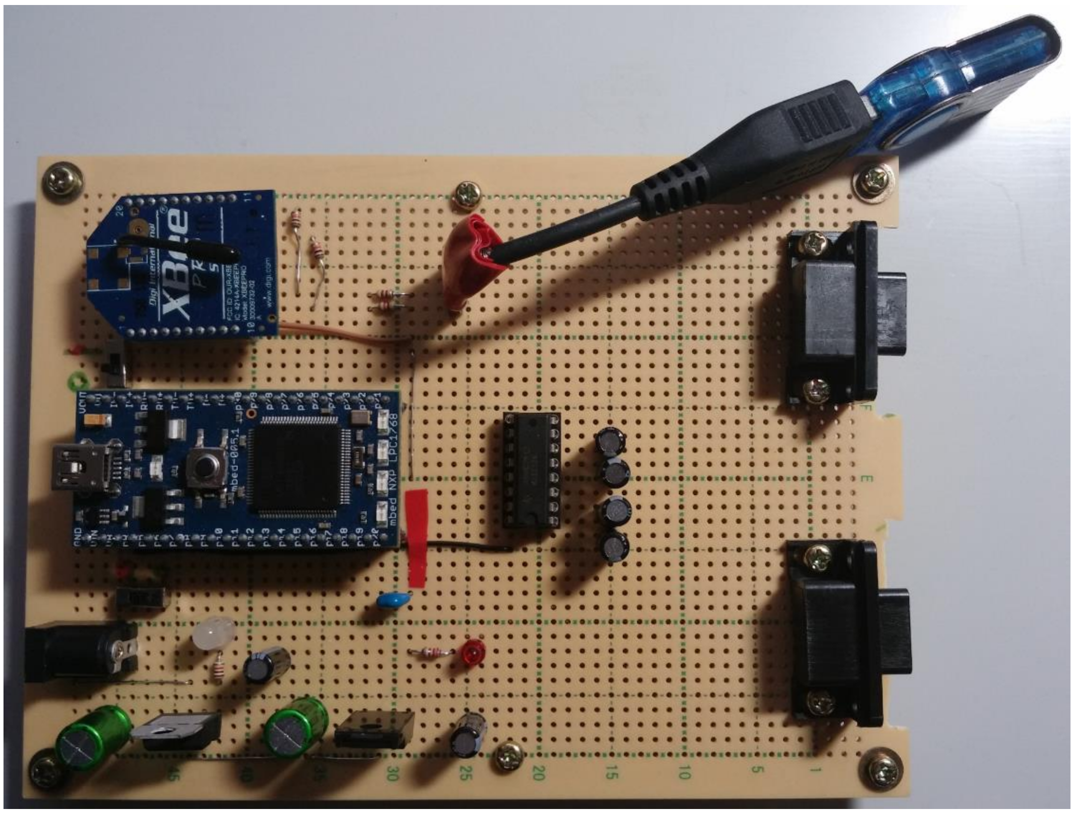


Figure 5.2 OBC board used for HILS

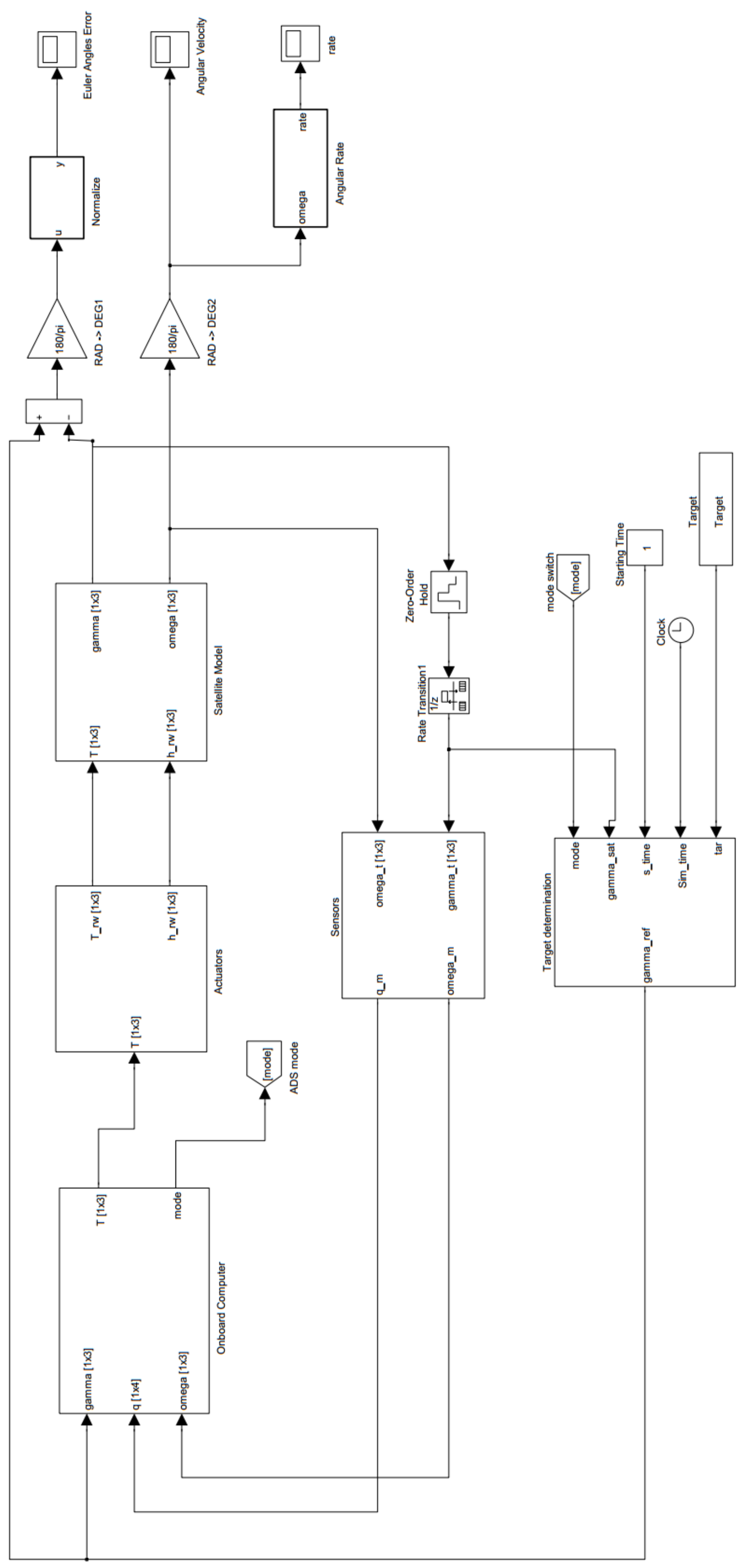


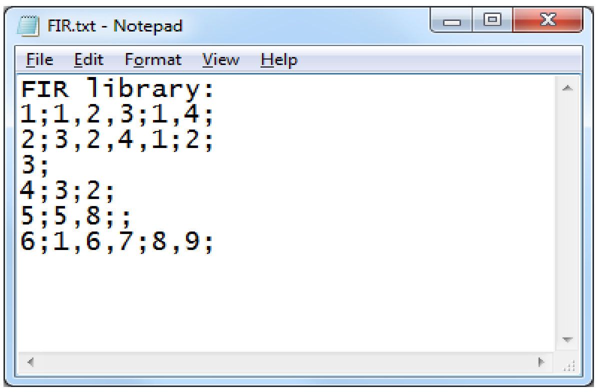
Figure 5.3 ADCS model in Simulink

5.1.2 Demonstration of software adjustments

This demonstration is made in order to show the adjustability of the designed software. Three types of software adjustments as mentioned in section 4.2.3 are demonstrated in this section. Results of each adjustment is verified by looking at the debug screen on ground station side (PC-2). When a software function is going to be executed, a message is displayed on PC-2 to show ID of that function. For example, if isolation function 1 is executed, people on ground station side can see the message "Isolation: function 1 is executed".

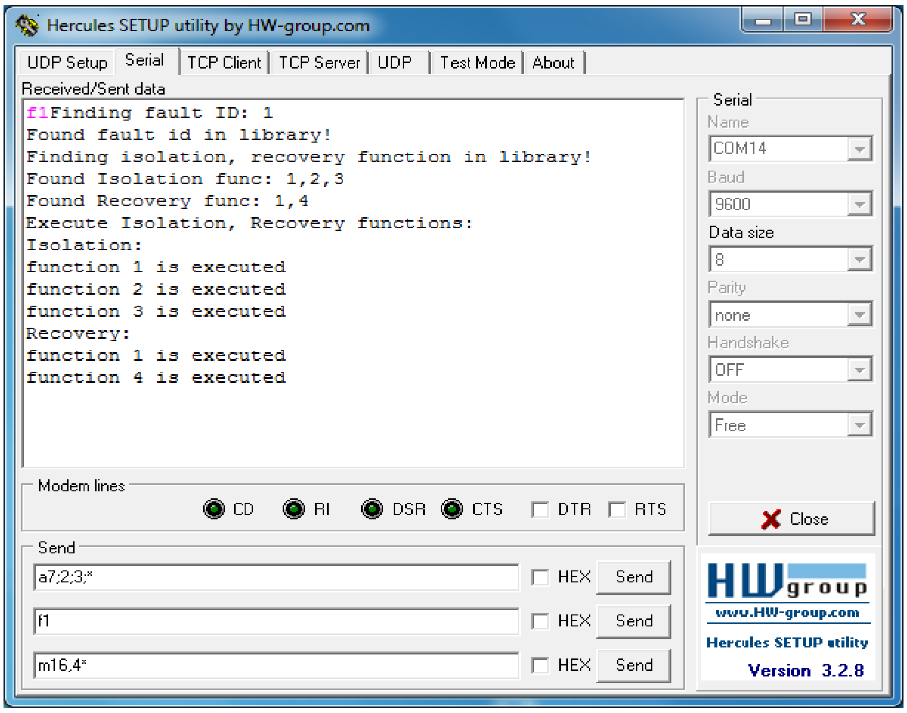
FIR library is created and store in the USB drive which connected to Mbed (as shown in Figure 5.2) with following contents:

First demonstration is to verify that prototype software can find isolation and recovery functions as they are already defined in library. Note that in this demonstration, failure detection and identification are not going to be tested. Fault IDs are sent from ground station PC-2 using a free software called Hercules terminal software (http://www.hw-group.com/products/hercules/index_en.html). Figure 5.5 illustrates the result when fault ID number 1 is sent from ground station. The software first receives fault ID from ground station then starts looking in FIR library for that ID. A "Found fault id message in library" is displayed when fault ID is found. The software then starts finding corresponding isolation, recovery functions in library, a message is displayed on debug screen for this action. As already defined in



```
FIR library:
1;1,2,3;1,4;
2;3,2,4,1;2;
3;
4;3;2;
5;5,8;;
6;1,6,7;8,9;
```

Figure 5.4 Original FIR library for demonstration



```
f1Finding fault ID: 1
Found fault id in library!
Finding isolation, recovery function in library!
Found Isolation func: 1,2,3
Found Recovery func: 1,4
Execute Isolation, Recovery functions:
Isolation:
function 1 is executed
function 2 is executed
function 3 is executed
Recovery:
function 1 is executed
function 4 is executed
```

Serial
Name: COM14
Baud: 9600
Data size: 8
Parity: none
Handshake: OFF
Mode: Free

Send: a7;2;3;*
Send: f1
Send: m16,4*

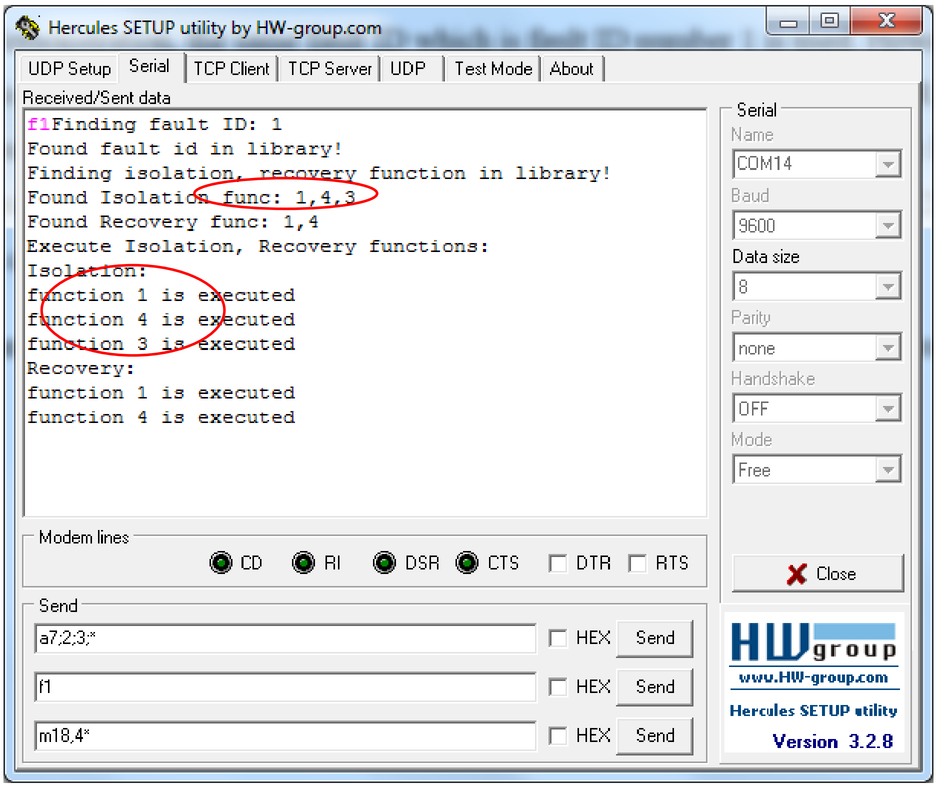
Figure 5.5 Result of demonstration #1

FIR library, isolation functions 1, 2, 3 and recovery functions 1, 4 are found. The five messages in the bottom of Figure 5.5 tell that those functions then are executed one by one. Process of isolation and recovery is done at this point.

Result of this demonstration verify that the prototype software has ability to find appropriate isolation and recovery functions in FIR library to execute onboard.

In second demonstration, the same fault ID which is fault ID number 1 is used. However, in this scenario, isolation function is going to be changed. The type of change here is FIR library modification. First, a command is sent from ground station computer to modify the library. Software receives the command and call the function to change isolation id number 2 in line 2 (which corresponds to fault ID 1) to 4.

Now the same process as in the first demonstration is carried out. Result of this process is shown in Figure 5.6. As seen in this figure, after modifying FIR library, the software was be able to change the way it reacts to fault ID 1. Differences between Figure 5.5 and Figure 5.6 are



```
f1Finding fault ID: 1
Found fault id in library!
Finding isolation, recovery function in library!
Found Isolation func: 1,4,3
Found Recovery func: 1,4
Execute Isolation, Recovery functions:
Isolation:
function 1 is executed
function 4 is executed
function 3 is executed
Recovery:
function 1 is executed
function 4 is executed
```

Serial: Name COM14, Baud 9600, Data size 8, Parity none, Handshake OFF, Mode Free

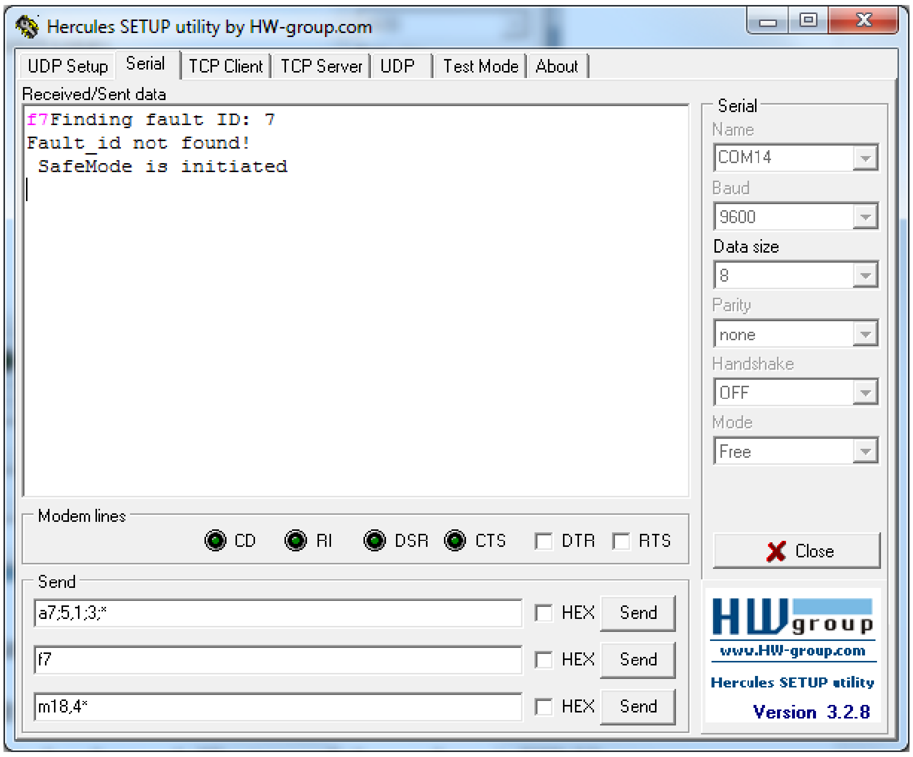
Send: a7;2;3;* | f1 | m18,4*

Figure 5.6 Result of demonstration #2

represented by red line circles in Figure 5.6. Instead of executing isolation function number 1, 2, 3, the software did execute function 1, 4, 3 as they are defined in FIR library after the modification was made.

The third demonstration is about adding new fault ID with isolation and recovery functions to library. As in the original FIR library shown in Figure 5.4, there is no line start with number 7. Therefore, when software received command to start looking for fault ID number 7, it could not find it. Figure 5.7 a) shows the messages on debug screen. Safe Mode as initiated.

After FIR library is updated by adding new line to the end of this text file ("7;5,1;3"), the same command was sent from ground station computer to let software know the current fault ID. The software then started finding fault ID and the corresponded isolation, recovery functions in library. As the result, it found the solutions. Then the functions could be executed successfully.

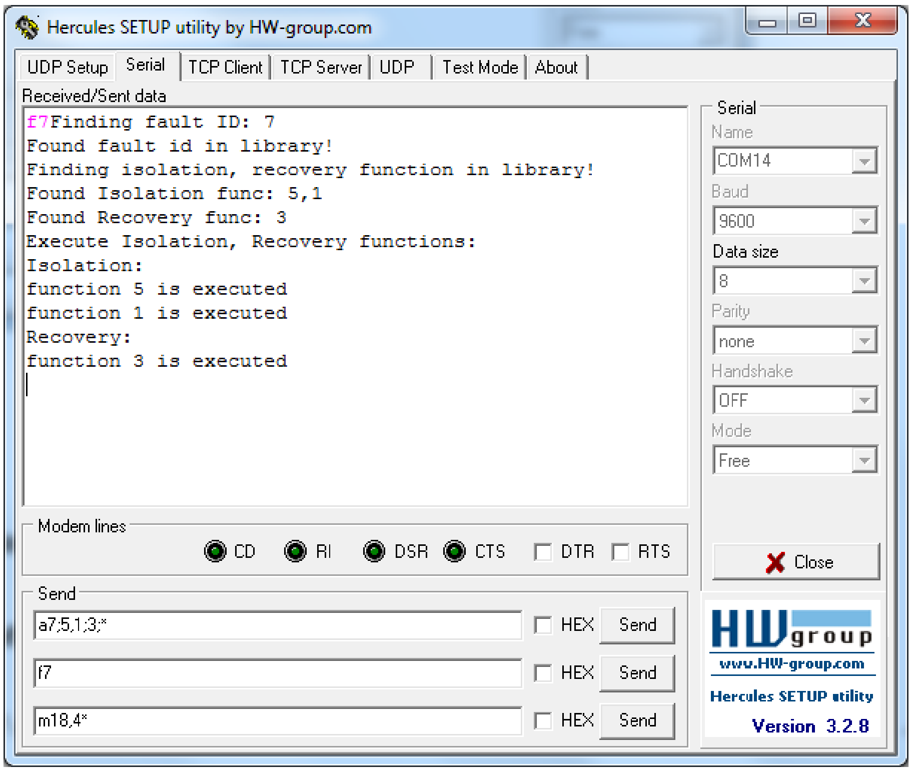


```
f7Finding fault ID: 7
Fault_id not found!
 SafeMode is initiated
```

Serial: Name COM14, Baud 9600, Data size 8, Parity none, Handshake OFF, Mode Free

Send: a7;5,1;3;* / f7 / m18,4*

a) Result before software adjustment



```
f7Finding fault ID: 7
Found fault id in library!
Finding isolation, recovery function in library!
Found Isolation func: 5,1
Found Recovery func: 3
Execute Isolation, Recovery functions:
Isolation:
function 5 is executed
function 1 is executed
Recovery:
function 3 is executed
```

Serial: Name COM14, Baud 9600, Data size 8, Parity none, Handshake OFF, Mode Free

Send: a7;5,1;3;* / f7 / m18,4*

b) Result after software adjustment

Figure 5.7 Result of demonstration #3

Through results of three demonstrations above, the prototype software is verified that has ability to be adjusted according to changes of FIR library save in the USB drive. In the next section, this adjustability of software is going to be verified to support for fault detection, isolation and recovery of ADCS.

5.1.3 Demonstration of FDIR

5.1.3.1 Predicted failure of ADCS

Failure scenario of ADCS is simulated for purpose of verification and results are shown in term of attitude error (the gap between target attitude and true satellite attitude at a time) for both situations when software is not adjusted and after adjusted.

In this simulation, a faulty gyroscope is simulated. As shown in Figure 5.8, a step function is added to output of gyroscope sensor in y-axis. This step function produces an error value with magnitude of 0.02 degree per second at the simulation time t = 200 seconds. Simulation is running for 2000 seconds and gyro output is presented in Figure 5.9. The curve in purple color shows the value of y-axis which is suddenly goes up at t = 200s and maintains till the end of simulation time.

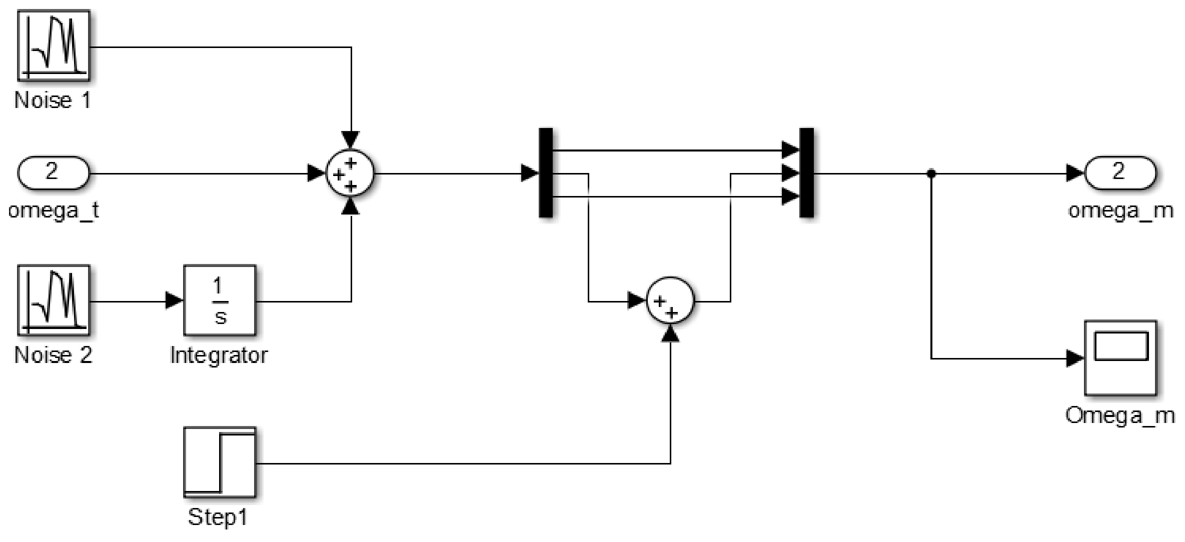


Figure 5.8 Simulink model of a faulty gyroscope

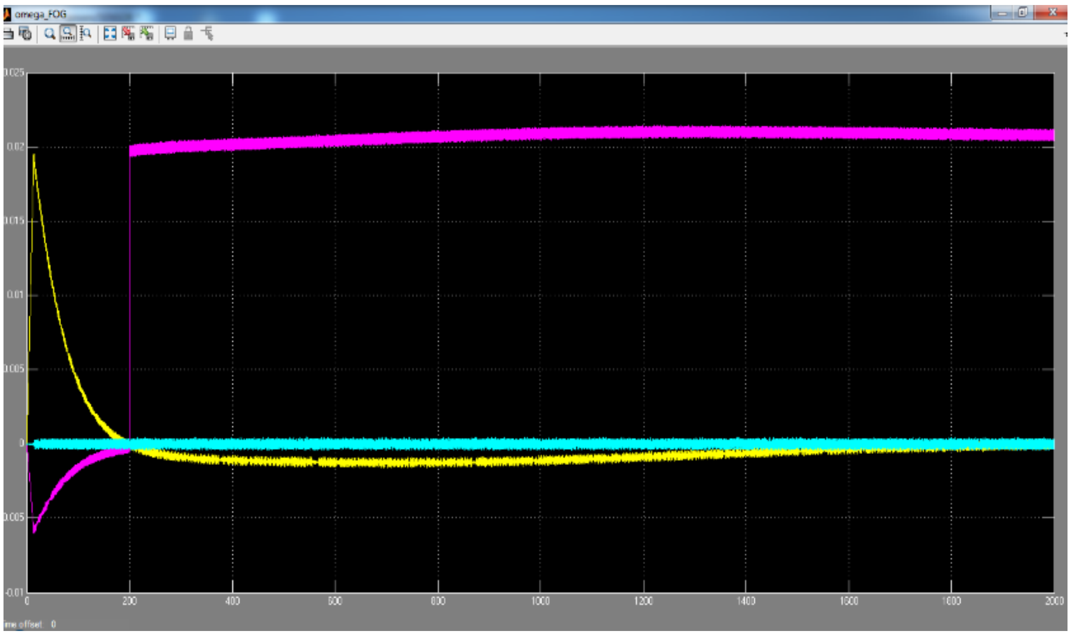


Figure 5.9 Output from faulty gyroscope (1)

This simulation demonstrated one of the most typical type of gyroscope failure modes. Result of attitude control in form of three Euler angles errors (attitude error) is illustrated in Figure 5.10. Vertical axis shows attitude error in degree and horizontal axis shows simulation time. Due to the introduced malfunction of gyroscope, satellite lost its attitude from the time t = 200 seconds and ADCS was not able to restore the target attitude.

In the next step of this simulation, FDIR library is enabled and after that, the same simulation is conducted and result of satellite attitude control is shown in Figure 5.11. As seen clearly in this figure, after FDIR mechanism is enabled, target attitude is achieved with acceptable error.

Result of this simulation verified that FDIR software was able to determine the appropriate isolation and recovery functions from library and execute those functions. Once failure is predicted and isolation and recovery functions are prepared, FDIR software can work to prevent that failure.

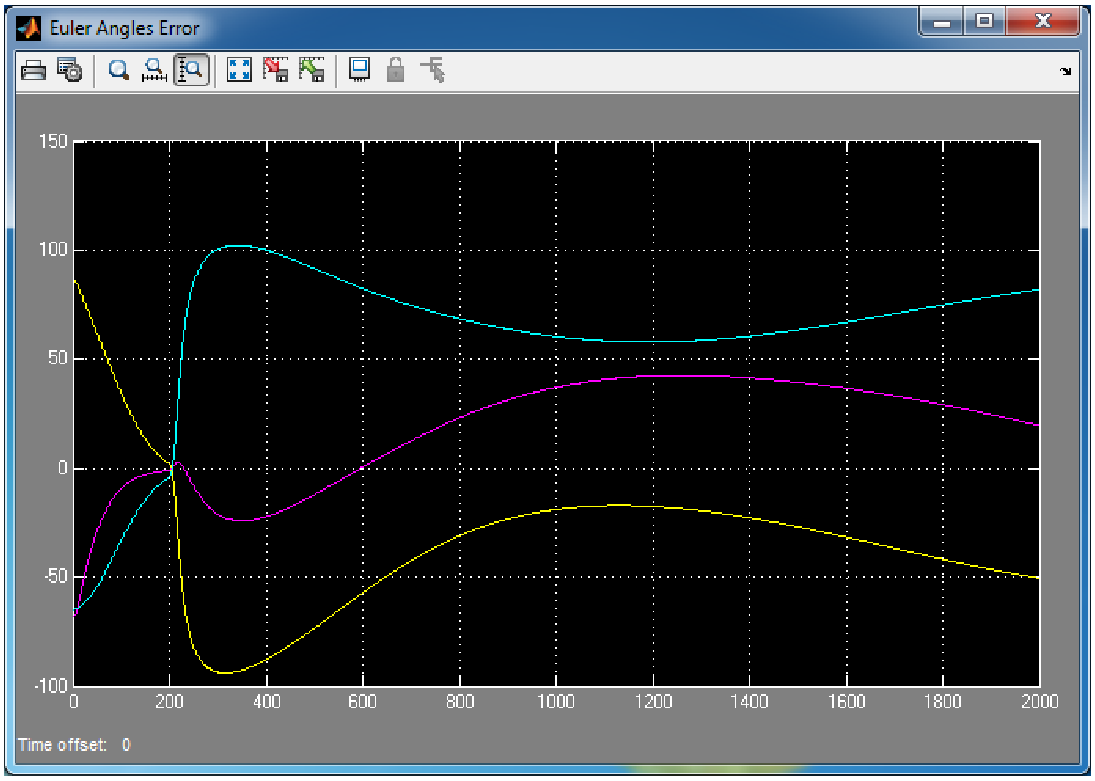


Figure 5.10 Attitude error when FDIR is disabled

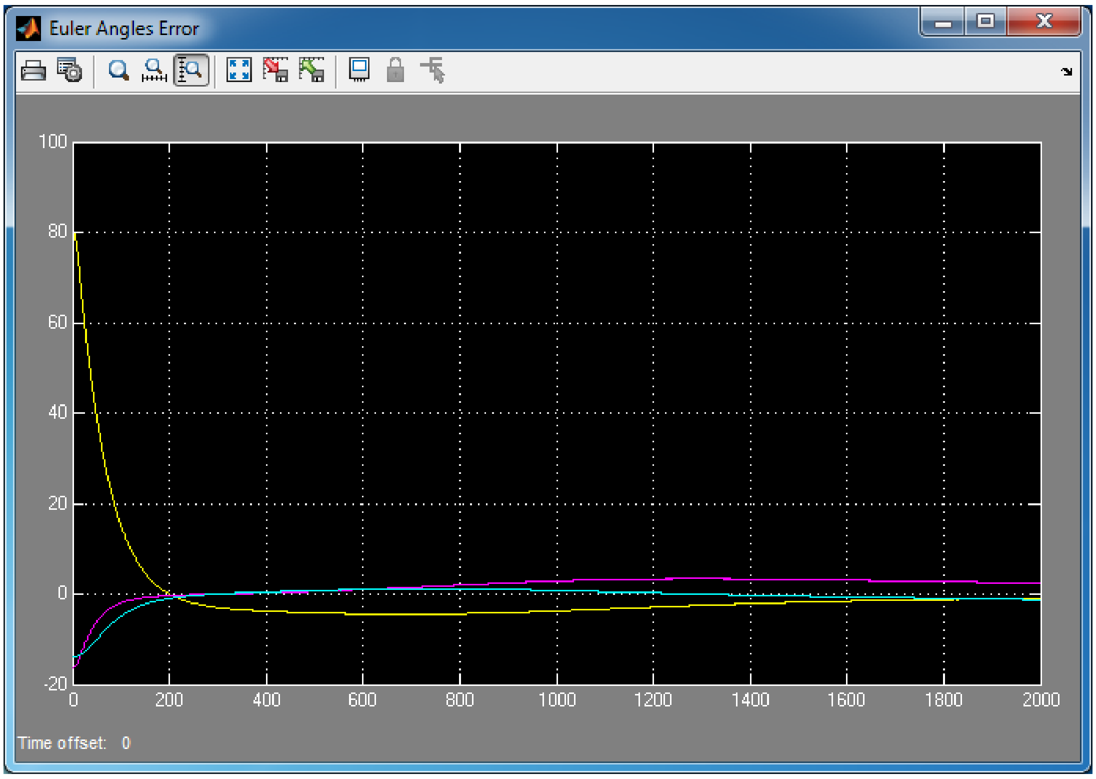


Figure 5.11 Attitude error when FDIR is enabled

5.1.3.2 Unpredicted failure of ADCS

Simulation scenario and assumptions are shown in Figure 5.12. After 10 minutes from starting time, fault occurred in gyroscope. Output of the faulty gyroscope is represented in Figure 5.13. Ground station can receive telemetry (TLM) data from ADCS 5 minutes after fault occurred. Then, 8 minutes later, operators can send uplink data to update FIR library in order to recover ADCS.

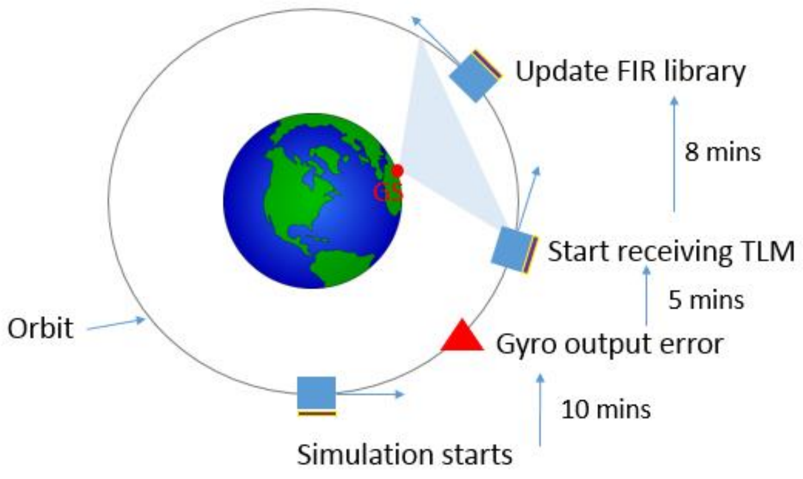


Figure 5.12 Simulation scenario

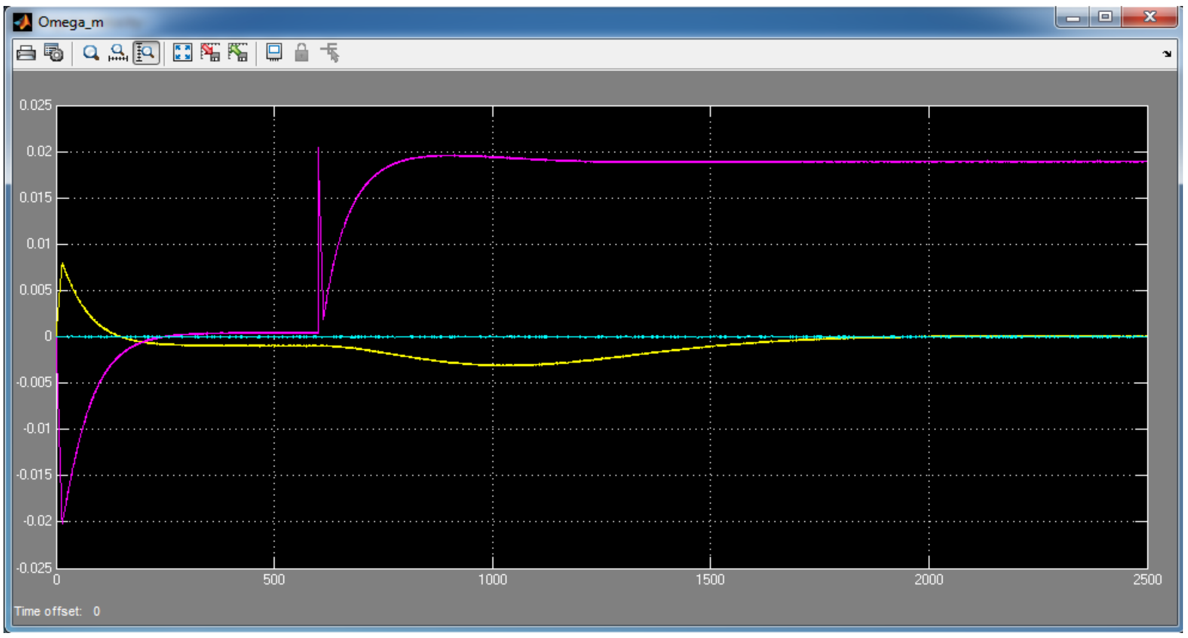


Figure 5.13 Output from faulty gyroscope (2)

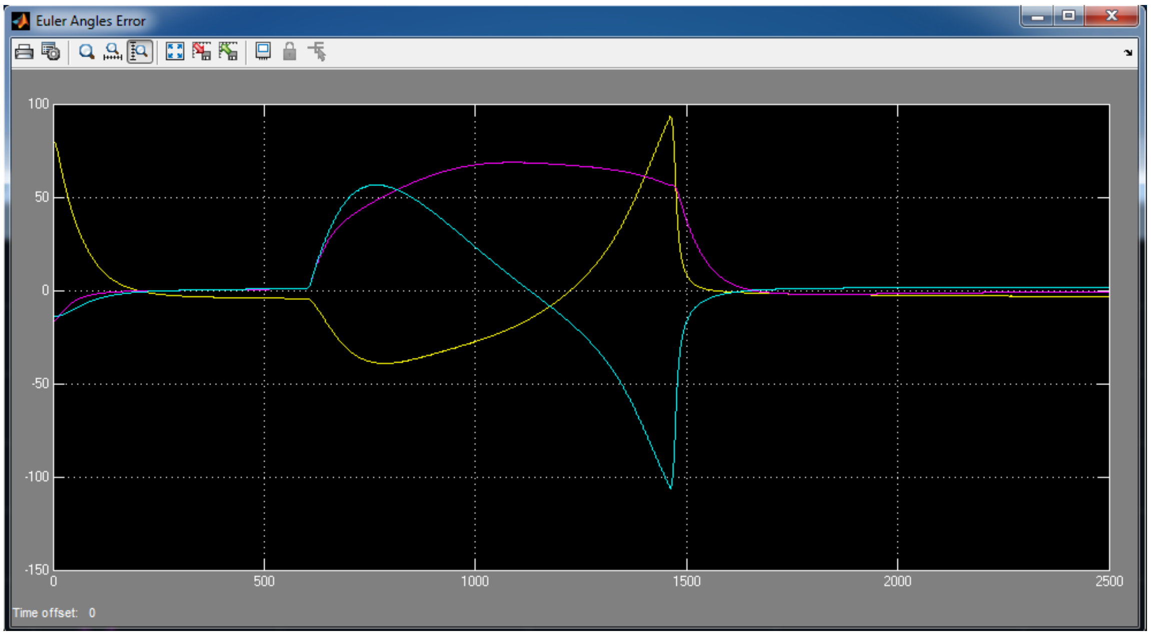


Figure 5.14 Simulation result

Result of this simulation is shown in Figure 5.14. As clearly seen in that figure, satellite attitude was lost due to the fault in gyroscope. Satellite is fluctuated from t = 600s. Right after FIR library was updated – 13 minutes after failure occurred, satellite attitude was starting to be converged again.

Result of this simulation verified that the FDIR software is adjustable by updating FIR library and also, that adjustability was able to support for FDIR of ADCS.

5.2 Interviews with Satellite Developers and Operators

Purposes of interview:

- ✧ To evaluate the proposed FDIR mechanism and the design of adjustable software for FDIR of ADCS base on experts' opinions
- ✧ To get feedbacks/comments to improve the design in future

Questions in the interviews include both closed and open-ended types. They are divided into three groups. The first group is about general information about the interviewees including name, organization, and their experiences in satellite development and operation. The second group includes questions to collect interviewees' opinions and experiences in safety operation of satellite system both in general and in their particular projects they have joined. Experiences of interviewees on failures of satellite system, subsystem and components after launch and their countermeasures action as well as the effectiveness of each action are also gathered by those questions. In addition, to answer questions in second group, interviewees also need to identify difficulties when they was dealing with failures from ground station. The third group of questions is where the proposed FDIR mechanism and the design of adjustable software for FDIR of ADCS are explained and evaluated base on interviewees' opinions and experiences. Comments and feedbacks for design improvements (if any) are also required in this group.

The interviews are conducted with two people who have experiences in satellite development and operation. Both of them have joined several satellite projects with different satellite size, from pico satellite (SwampSat), to micro satellites (Hodoyoshi - 3, 4, UNIFORM, PROCYON) and big satellites (such as EOS AURA, NPOESS, NIRCam, etc.) with several different roles such as test engineer, member in subsystem team, project manager and advisor. Details of the interviews are listed in following part.

Question 1: In your organization, or in the projects you have joined,

1. How do people think about safety of satellites?

a) Very important

b) Important

c) Not so important

d) Don't care

e) Depend:

<u>Answers:</u> Both of interviewees chose the option a) which is very important.

2. ***How do people think about failures of satellite systems/ subsystems?***

a) Acceptable

b) Unacceptable

c) Depend:

<u>Answer 1:</u> It depends. If there is redundant system or subsystem, it is acceptable.

<u>Answer 2:</u> We never say it is ok or not. Depends on what kind of failure occurred. For example, if failure is temporary, and if there duration is short, we may accept. However if failure is permanent, it is unacceptable. If there is redundancy or a mechanism to recover from that failure mode.

Question 2: In your experienced projects, what are the ways (technique/ mechanism, etc.) to handle failures and increase safety of satellites? What are the advantages/ disadvantage?

<u>Answer 1:</u> I worked for NASA when I was in United Sate. Usually, for big satellites, redundant components are almost mandatory, we need to put other identical components to satellite and if one of them fails the system switch to redundant component. By that way, failure can be recovered 100%, that is the big advantage. However, disadvantages is the system becomes complicated, heavier, and more expensive. That is reason why it is only available in big satellites. For nano, micro satellite, size and mass are very limited to put redundant system in, so usually, only very important components such as reaction wheel has redundancy. In many cases, we do not have redundant for other components.

<u>Answer 2:</u> In my experience in microsatellite projects, we utilized one redundant reaction wheel. At the same time, it was not intentional but we also used magnetic torque (MTQ) to control attitude of satellite. In both ways, we can achieve attitude controlled. However, it does not mean by using MTQ, we can achieve required accuracy, agility.

We also able to reprogram onboard computer by rewrite and compile it then break it into several parts to upload to satellite via uplink commands. Once all binary file is uploaded, we send another command to swap the existing flight software. By that way we can upload the whole new software to deal with unpredicted or unexpected failures that occur in orbit. The only issue is it take so long time. As we calculated, if we have to upload the same size of the binary file of the whole software, it could take more than one week, even when we utilize both day and night communication opportunities.

Question 3: Have you or your team ever experienced with any failure in satellite after launch? (Any kind of failure such as components, subsystem, system, etc.)

a) No.

b) Yes. (Provide more details if possible)

Luckily, both answers for this question was "yes". And more details about failure they have experienced with are provided.

<u>Answer 1:</u> EOS AURA HIRDLS has failed just after launch

<u>Answer 2:</u> Under voltage controlled, unexpected attitude, loss of telemetry from attitude control computer, and unexpected temperature increased.

1: What are the considerations when dealing with those failures?

a) Recovery time

b) Performance of the system after recovered

c) Ease of recovery process

d) Other:

<u>Answer 1:</u> chose a) and b)

<u>Answer 2:</u> chose d)

2: What did you or your team do to handle the failures? Effectiveness of countermeasure actions?

<u>Answer 1:</u> We tried to recover functionality of instrument by spending couple of months using every way to recover. However, at the end, we could not. Only 30% of performance was recovered. Customer was unhappy because they are scientists and had been waiting for years to get it launched.

Answer 2: One example is that, we realized it could be risky to turn on star tracker. Because it could cause more problems. We were not able to get precise attitude control or determination. We considered about how to still control satellite attitude without star tracker. Because of that, we utilized MTQ which is not so complicated. About effectiveness of the countermeasure action, as you know without star tracker, we cannot get precise attitude control. It is related to mission success and the mission required pictures take of ground. The picture turned out to be not so degraded in quality. However, it is difficult to utilize those pictures for scientific purpose because of the lack or less accurate of housekeeping data added in the pictures. Scientists need to come up with the way for image mapping.

3: Is there any difficulty when dealing with failures from ground station? What needed to be improved? Is there any need to improve the way ground station interact with satellite in case of failures?

Answer 1: The data we could have is very limited, we tried to gather both data before and after launch. If we could have a camera inside it would be easy.

Answer 2: It is not really related, but we always need to improve communication link between ground station and satellite.

4: Do you think using existing software functions onboard will help? Was the design of the satellite enable this (using existing software functions to handle failures)?

Answer 1: In my case, since the failure occurred in one instrument, so it is not applicable.

Answer 2: I would say "half", because in the case of failure I mentioned above, we decided not to turn on star tracker, by sending command from ground. If there is no such kind of functions onboard, we could not tell satellite to do so.

Question 4: We have proposed an FDIR mechanism and designed the software for FDIR of ADCS in the way that enable operators on ground station to use existing functions onboard to handle new failure mode. (Explain about the design).

1: How do you think about the proposal? (In term of recovery time, convenient for operators, etc.)

<u>Answer 1:</u> Recovery time compared with the other way such as reprogramming onboard software can be shorter. Because, we only need to upload small amount of data as a part of FIR library.

<u>Answer 2:</u> I think the recovery time can be short enough and good. For second question, if you only think about uploading library, it will be convenient for operators. However, we also need to think about how easy is it to come up with alternatives combination of functions as a part of FIR library. Overall, my expression is that it is more convenient for us.

2: If this FDIR mechanism was applied to the design of satellites you have developed, do you think it would help when failures occur?

<u>Answer 1:</u> General speaking, it would be the good idea. But, it hard to answer in more detailed since the failures I have experienced are about hardware which cannot be solved by software.

<u>Answer 2:</u> In the satellite that we have developed, having this kind of software would help in certain kind of functions. We set several modes in software. In each mode, certain functions are enabled and certain functions are disabled. If we have capability to have more flexibility in choosing which function to enable in each mode, it could add more freedom in operation. Probably the only tricky part is how to co-operate with existing mode and this kind of flexibility because having mode which was very useful we did not have to worry about individual function. Therefore, if there is a nice way of mixing software functions into existing control mode, that would definitely, I believe, help to improve satellite performance.

Question 5: What could to be improved in the proposal?

<u>Answer:</u> If you can come up with the way or make the guide line to apply failure prediction or failure analysis such as FTA or FMEA, it could be better. Currently, you have utilized FTA to

your project to analyze failure of ADCS, but if you can provide general guidance for apply the design to other subsystem or other project, it could mean a lot.

Conclusions from results of interviews:

- ✧ Safety and reliability of satellites in orbit are considered as a very important aspect in every satellite development project.
- ✧ Various type of failures prevention, recovery was applied to satellites such as hardware redundancy. However those techniques are not applicable for nano, micro satellite due to limitations of size and mass. Failures still occurred when satellites are already in orbits. There are the needs to improve the way people handle failures from ground station.
- ✧ Based on experts’ opinions, the proposed FDIR mechanism and design of the adjustable software for FDIR of ADCS can be able improve the situation by shortening the time of recovery and they are convenient for operators when dealing with failures.
- ✧ The concept of proposal can be improved and apply in other subsystems as well as other satellite projects.

Chapter 6. Conclusions and Future Works

6.1 Overall summary

In this research, the design of adjustable software for FDIR of ADCS was done using SysML (System Modeling Language). The software was designed in the way so that operators on ground can utilize existing isolation, recovery functions onboard to deal with failures occur to ADCS in orbit. Also, a novel FDIR mechanism was proposed for ADCS of MDG satellite, by using the design of adjustable software. Instead of spending a long time to reprogram onboard computer, satellite operators only need to send one or two commands (depends on failure mode) which consist of the combination of fault ID, isolation function ID and recovery function ID, to update the FIR library. The FDIR software then be adjusted according to changes in FIR library to perform isolation and recovery functions. Recovery time can be shortened, and effects of failures on ADCS as well as on satellite system can be reduced very much, thanks to the adjustability of software. In addition, with the introduction of adjustable software for FDIR, operators can have more freedoms when dealing with in-orbit failures, the recovery process will become more convenient for operators on the ground.

To verify the design as well as software functionality, simulations are conducted. First, the software is verified to be adjustable when operators send uplink data to update FIR library by software functions demonstrations. Second, a hardware-in-the-loop simulation is conducted with satellite dynamics in Simulink model and adjustable software is implemented in Mbed board. The software is verified to be able to detect, isolate the fault of gyroscope and restore ADCS performance after adjustments are made (by updating FIR library).

The proposed FDIR mechanism and design of adjustable software then be evaluated based on experts' opinions. Satellite developers and operators who have experiences in on-orbit failures of satellites are interviewed. All the interviewees shared the same expression that the proposal as well as the design of adjustable software would help to improve the effectiveness of recovery process from ground station. Operators can have more freedoms when dealing with failures and recovery time can be reduced (compare with the existing technique, such as reprogramming onboard computer).

6.2 Future works

Further works should be conducted in order to improve and implement the design of adjustable software during development phase of MDG satellite project. The design of FDIR software as well as the proposed FDIR mechanism should be validated during development process and in the future operation of satellite in orbit.

In this research, the proposed FDIR mechanism and the design of adjustable software for FDIR only focus on ADCS of MDG satellite. Further researches could be done in order to apply for FDIR of other subsystems and whole satellite system.

References

[1] X. Olive, "FDI(R) for Satellite: How to Deal With High Availability and Robustness in The Space Domain?," *Int. J. Appl. Math. Comput. Sci,* vol. 22, pp. 99-107, 2012.

[2] Mak Tafazoli, "A study of on-orbit spacecraft failures," *Science Direct*, 2007.

[3] David M. Hardland, Ralph D. Lorenz, Space system failures, Springer, 2006.

[4] Shusaku Yamaura, Seiko Shirasaka, Takashi Hiramatsu, Miki Ito, Yuta Araki, Kikuko Miyata, Tomomi Otani, "UNIFORM-1- First Micro-Satellite of Forest Fire Monitoring Constellation project," in *Conference on Small Satellites*, 2014.

[5] F.N. Pirmoradi, F. Sassani, C.W. de Silva, "Fault detection and diagnosis in a spacecraft attitude determination system," *Acta Astronautica*, vol. 65, pp. 710-729, 2009.

[6] Maurício N. Pontuschka, Ijar M. da Fonseca, "FDIR for the IMU Component of AOCS Systems," *Mathematical Problems in Engineering*, 2014.

[7] Amitabh Barua, Purnendu Sinha, Kash Khorasani, "On the Fault Diagnosis and Failure Analysis in the Satellite Attitude Control Subsystem," 2004.

[8] Wilfried Ley, Klaus Wittmann, Willi Hallmann, Handbook of Space Technology, John Wiley & Sons, 2009.

[9] YONEZAWA Katsuo and HOMMA Masanori, "Overview of ETS-VIII Satellite," *Journal of the National Institute of Information and Communications Technology*, vol. 50, pp. 14-22, 2003.

[10] NASA Technique DFE-7, "Fault - Detection, Fault - Isolation and Recovery (FDIR) Techniques".

[11] Isermann, Rolf., "Process fault detection based on modeling and estimation methods—a survey," *Automatica*, vol. 20, no. 4, pp. 387-404, 1984.

[12] Frank, Paul M., and Xianchun Ding., "Survey of robust residual generation and evaluation methods in observer-based fault detection systems," *Journal of process control*, vol. 7, no. 6, pp. 403-424, 1997.

[13] R. Isermann, Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance, Springer, 2006.

[14] Yvon THARRAULT, Gilles MOUROT, Jose´ RAGOT, "Fault detection and isolation with robust principal component analysis," in *Mediterranean Conference on Control and Automation*, Ajaccio, France, 2008.

[15] Wallace R. Blischke, D. N. Prabhakar Murthy, Reliability: Modeling, Prediction, and Optimization, Jonh Wiley and Sons, Inc, 2000.

[16] Fatemeh SalarKaleji, Aboulfazl Dayyani, "A survey on Fault Detection, Isolation and Recovery (FDIR) Module in Satellite Onboard Software," *Recent Advances in Space Technologies (RAST)*, pp. 545 - 548, 2013 .

[17] Alexandra Wander, Roger Forstner, "Innovative Fault Detection, Isolation and Recovery on-board Spacecraft: Study and Implementation using Cognitive Automation," in *2013 Conference on Control and Fault-Tolerant Systems (SysTol)*, Nice, France, 2013.

[18] Vinod Kumar, A.K.Kulkarni, K.Parameshwaran, R. Pandiyan and N.K.Malik, "ON-BOARD AUTONOMY FOR ISRO GEOSYNCHRONOUS SPACECRAFT," *Advances in the Astronautical Sciences*, vol. 145, pp. 109-118, 2012.

[19] Walton R. Williamson, Jason L. Speyer, Vu T. Dang, and James Sharp, "Fault Detection and Isolation for Deep Space Satellites," *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, vol. 32, no. 5, pp. 1570-1584, 2009.

[20] G. N. Ashtiani, "Fault Detection and Isolation in Spacecraft Attitude Control System Using Parity Equation Method," 2007.

[21] Hossein Bolandi, Mostafa Abedi and Mehran Haghparast, "Fault detection, isolation and accommodation for attitude control system of a three - axis satellite using interval linear parametric varying observers and fault tree analysis," *Aerospace Engineering*, 2014.

[22] MicroDragon Development team, "Mission Definition Doccumentation," 2015.

[23] National Aeronautics and Space Administration, "Failure Modes, Effects and Criticality Analysis (FMECA)," 2010.

[24] Michael Stamatelatos, William Vesely, Joanne Dugan, Joseph Fragola, Joseph Minarick, Jan Railsback, Fault Tree Handbook with Aerospace Applications, 2002.

[25] Nadjim M. Horri, Phil Palmer, "Practical Implementation of Attitude-Control Algorithms for an Underactuated Satellite," *Journal of Guidance, Control and Dynamics,* vol. 35, 2012.

[26] ZHENG Min-jie, XU Shi-jie, "Backstepping Control for Attitude Control System of an Underactuated Spacecraft," *Journal of Astronautics,* 2006.

[27] N. G. Leveson, "Software Challenges in Achieving Space Safety," *Journal of the British Interplanetary Society,* vol. 62, 2009.

[28] Ali Zolghadri, David Henry , Jer ´ ome Cieslak, Denis Efimov, Philippe Goupil, Fault Diagnosis and Fault-Tolerant Control and Guidance for Aerospace Vehicles: From Theory to Application, London: Springer, 2014.

[29] Sanford Friedenthal, Alan Moore, Rick Steiner, A Practical Guide to SysML: The System Modeling Language, Elsevier Inc., 2012.

[30] JR Wertz, DF Everett, JJ Puschell, Space mission engineering: The new SMAD, Microcosm Press, 2011.

Appendix

Simulation source code

```
/******************************************************************************************
//
//
//
//
//
//
******************************************************************************************/
#include "mbed.h"
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "MSCFileSystem.h"

/******************************************************************************************
DigitalOut myled(LED1);
Serial sat(USBTX, USBRX);  //satellite model
Serial com2(p9, p10);
Serial com3(p13,p14);
Serial com(p28,p27);       //communication

//LocalFileSystem local("local");
MSCFileSystem msc("usb");

/******************************************************************************************
// Reaction wheel
#define I_rw 0.0007

// OBC cycle time
#define obc_dt 0.1

//PD Control gains
#define Kp 1.0
#define Kd 30.0

#define fog_er 0.005

#define Ix 2.08
#define Iy 2.08
#define Iz 1.88

//Constants
#define pi 3.14159265359
#define rad2deg 180.0/pi
#define deg2rad pi/180.0

/******************************************************************************************
FILE* fp;
char msg[20];
int pos[3];
char isolation[20], recovery[20];
char * pch;
char d, comm, fault_id;
int c, f_active;
int indicator, fun;

float q1, q2, q3, q4, q1_ref, q2_ref, q3_ref, q4_ref;
float phi, theta, psi, theta_ref, phi_ref, psi_ref;
float w1, w2, w3, w1_dot, w2_dot, w3_dot;
float V1, V2, V3, V1_dot, V2_dot, V3_dot;
float p_w1, p1_w1, p2_w1, p3_w1;
float p_w2, p1_w2, p2_w2, p3_w2;
float p_w3, p1_w3, p2_w3, p3_w3;
float T1, T2, T3, T_rw1, T_rw2, T_rw3;
float norm_q, h_rw1, h_rw2, h_rw3;
float eq1, eq2, eq3, eq4, wrw1, wrw2, wrw3;
float w1_est, w2_est, w3_est;
float w_rw1, w_rw2, w_rw3;
float ph_rw1, ph_rw2, ph_rw3;
int fault_flag;
float q1_e, q2_e, q3_e, q4_e;

int i = 0;
float t = 0.0;
char msg1[256];
int mode;
int t_count;

/*********************************************************************************************
int MidCopy(char* outString, char* inString, int startPos, int stopPos)
{
   int index = 0;
   int i;
   for(i  = startPos - 1; i < stopPos; i++) {
      outString[index] = inString[i];
      index++;
   }

   //NULL terminate it
   outString[index] = 0;
   return 0;
}
//isolation_1
int disable_fog_x()
{
   w1 = 0;
   return 0;
}
//isolation_2
int disable_fog_y()
{
   w2 = 0;
   return 0;
}
//isolation_3
int disable_fog_z()
{
   w3 = 0;
   return 0;
}

//recovery_1
void get_est_value_fog_x()
{
   w1 = w1_est;
```

```
    //return 0;
}
//recovery_2
void get_est_value_fog_y()
{
    w2 = w2_est;
    //return 0;
}
//recovery_3
void get_est_value_fog_z()
{
    w3 = w3_est;
    //return 0;
}

/************************************************************************************
int find_fault_id(char fault_id)
{
    int current_pos = 0;

    //com.printf("Finding fault ID: %c\n",fault_id);

    //rewind (fp);
    //fp = fopen("/local/FIR.txt", "r");
    fp = fopen("/usb/FIR.txt", "r");
    d = 0;
    c = fgetc(fp);
    while (c != EOF) {
        //printf("On the way!\r\n");
        c = fgetc(fp);
        if (c == '\n') {
            //printf("start looking in a new line");
            d = fgetc(fp);
        }
        if (d == fault_id) { //found fault ID that we are looking for
            //printf("Hello!");
            current_pos = ftell(fp);
            *msg = NULL;
            //fsetpos(fp, &current_pos+1);
            fseek ( fp , current_pos -1, SEEK_SET );
            fgets(msg,20,fp);
            //printf("%s",msg);
            //com.printf("Found fault id in library!\n");
            fclose (fp);
            return (1);
        }
    }
    //f_active = 0; ///temporary
    fclose(fp);
    //com.printf ("Fault_id not found!\n SafeMode is initiated\n");
    return (0); //fault ID not found
}

/************************************************************************************
int find_IR_function()
{
    //com.printf("Finding isolation, recovery function in library!\n");
    //looking for position of ';' in the line
    int j = 0;
    for (int i = 0; i < 20; i++) {
        if (msg[i] == ';') {
            //printf("possition of ; %d",i);
            pos[j] = i;
            j++;
        }
    }
    //printf("possition of ; %d %d %d\n",pos[0], pos[1],pos[2]);
    if (j < 2) {
        //com.printf("No existing solution is found in library\n");
        f_active = 0;
        return (1); //no existing solution is found in library
    }

    //split string msg to find isolation, recovery func
    MidCopy(isolation,msg,pos[0]+2,pos[1]);
    MidCopy(recovery,msg,pos[1]+2,pos[2]);

    //com.printf("Found Isolation func: %s\nFound Recovery func: %s\n",isolation, recovery);
    //com.printf("Execute Isolation, Recovery functions:\n");
    return (0);
}

/*****************************************************************************************
int perform_isolation()
{
    //com.printf("Isolation:\n");
    pch = strtok(isolation,",");
    while (pch != NULL) {
        //printf("Func %s, ",pch);
        char func = *pch;
        switch (func) {
            case '1':
                //com.printf("function 1 is executed\n");
                disable_fog_x();
                break;

            case '2':
                //com.printf("function 2 is executed\n");
                disable_fog_y();
                break;

            case '3':
                //com.printf("function 3 is executed\n");
                disable_fog_z();
                break;

            case '4':
                com.printf("function 4 is executed\n");
                break;

            case '5':
                com.printf("function 5 is executed\n");
                break;

            case '6':
                com.printf("function 6 is executed\n");
                break;

            case '7':
                com.printf("function 7 is executed\n");
                break;
            case '8':
                com.printf("function 8 is executed\n");
                break;

            case '9':
                com.printf("function 9 is executed\n");
                break;

            default:
                com.printf("No such isolation function is available\n");
                break;
        }
        pch = strtok (NULL, ",");
    }
    return (0);
}

/*****************************************************************************************
int perform_recovery()
{
    //com.printf("Recovery:\n");
    pch = strtok(recovery,",");
    while (pch != NULL) {
        //printf("Func %s, ",pch);
        char func = *pch;
        switch (func) {
            case '1':
                com.printf("function 1 is executed\n");
                get_est_value_fog_x();
                break;

            case '2':
                com.printf("function 2 is executed\n");
                get_est_value_fog_y();
                break;

            case '3':
                com.printf("function 3 is executed\n");
                get_est_value_fog_z();
                break;

            case '4':
                com.printf("function 4 is executed\n");
                break;

            case '5':
                com.printf("function 5 is executed\n");
                break;

            case '6':
                com.printf("function 6 is executed\n");
                break;

            case '7':
                com.printf("function 7 is executed\n");
                break;

            case '8':
                com.printf("function 8 is executed\n");
                break;
            case '9':
                com.printf("function 9 is executed\n");
                break;

            default:
                com.printf("No such recovery function is available\n");
                break;
        }
        pch = strtok (NULL, ",");
    }
    return (0);
}

/*******************************************************************************************
// Function: modify_lib
// To modify the FIR library by replace existing id by new id
// Input: index - index (location) of id in FIR lib
//          c - new id want to put in lib
// Return; NONE
void modify_lib(int index, int c)
{
    fp = fopen("/usb/FIR.txt", "r+");
    fseek (fp, index, SEEK_SET);
    fputc (c, fp);
    fclose(fp);
}

//Function: add_new_line
// To add new line to FIR lib
//input: data in new line
//Return NONE
void add_new_line(char * data)
{
    //fp = fopen("/local/FIR.txt", "a");
    fp = fopen("/usb/FIR.txt", "a");
    //fseek (fp, 0,SEEK_END);
    fputc('\n',fp);
    fputs (data,fp);
    *msg = NULL;
    fclose(fp);
}

/*******************************************************************************************
void getline()
{
    for(int i=0; i<20; i++) {
        msg[i] = com.getc();
        if(msg[i] == '*') {
            msg[i] = 0;
            return;
        }
    }
    //error("Overflowed message limit");
}

void get_input() {
    while(sat.getc() != '$');    // wait for the start of a line
    for(int i=0; i<256; i++) {
        msg1[i] = sat.getc();
        if(msg1[i] == '*') {
            msg1[i] = 0;
            //pc.printf("%s\r\n",stt_msg);
            return;
        }
    }
    //error("Overflowed message limit");
}

/**************************************************************************************
void RW_control()
{
    sat.printf("%f %f %f %d\n",T1,T2,T3,mode);
}
//
float sign(float value)
{
    if (value >= 0)
        return 1.0;
    else return -1.0;
}
//end of sign

void omega2quat()
{
    q1 = q1 +   0.5 * (     q2 * w3 - q3 * w2 + q4 * w1)  * obc_dt;
    q2 = q2 +   0.5 * (   - q1 * w3 + q3 * w1 + q4 * w2)  * obc_dt;
    q3 = q3 +   0.5 * (     q1 * w2 - q2 * w1 + q4 * w3)  * obc_dt;
    q4 = q4 + (- 0.5)*(     q1 * w1 + q2 * w2 + q3 * w3)  * obc_dt;

    norm_q = sqrt (q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4) ;

    q1 = q1 / norm_q;    q2 = q2 / norm_q;    q3 = q3 / norm_q;    q4 = q4 / norm_q;
}

/**************************************************************************************
int main()
{
    com.baud(9600);
    sat.baud(115200);

   //init values
    //
    //Time count = 0 at  the begining
    t_count = 0;

    //set fault flag = 0, since there is no dectected failure
    fault_flag = 0;

    //ADCS Default Mode
    mode = 2;

    //Initialization
    //Omega satellite
    w1 = 0.;    w2 = 0.;    w3 = 0.;

    //Estimation of omega satellite
    w1_est = 0.;    w2_est = 0.;    w3_est = 0.;

    //No momentum at begining
    ph_rw1 = 0.;    ph_rw2 = 0.;    ph_rw3 = 0.;

    //RW rotation speed
```

```
w_rw1 = 0.;    w_rw2 = 0.;    w_rw3 = 0.;

//Quaternion init
q1 = 0.;    q2 = 0.;    q3 = 0.;    //q4 = 1.;

f_active = 1;

com.printf("Connected!\r\n");
//fp = fopen("/local/FIR.txt", "r");
fp = fopen("/usb/FIR.txt", "r");
//printf("File opened!\r\n");
//fprintf(fp,"1;1,2,3;1;\n2;3;4;\n");
//rewind (fp);
//fseek ( fp , 16, SEEK_SET );
//fputc('3',fp);
//modify_lib(16,'5');//**********************************CONFIRMED_OK
//add_new_line("\n4;3;2;"); //****************************CONFIRMED_OK
fclose(fp);

//find_fault_id('4');
//find_IR_function();
//perform_isolation();
//perform_recovery();
while (1) {
    if (com.readable()) {
        comm = com.getc();
        //com.putc(comm);
        if (comm == 'f') fault_id = com.getc();
        if (comm == 'a') {
            getline();
            add_new_line(msg);
            //com.printf("%s\r\n",msg);
        }
        if (comm == 'm') {
            getline();
            sscanf(msg,"%d,%s", &indicator, &fun);
            modify_lib(indicator, fun);
        }
    // control functions
    if (f_active == 1) {
        //STT_get_output();
        //FOG_get_output();
        //ADS_get_ref();

        //p3_w3 = p2_w3; p2_w3 = p1_w3; p1_w3 = p_w3; p_w3 = w3;
        //p3_w2 = p2_w2; p2_w2 = p1_w2; p1_w2 = p_w2; p_w2 = w2;
        //p3_w1 = p2_w1; p2_w1 = p1_w1; p1_w1 = p_w1; p_w1 = w1;

        //Information input from dynamics simulation computer
        get_input();
                sscanf(msg1,"D,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f", &w1, &w2, &w3, &q1, &q2, &q3,
                &q4, &phi_ref, &theta_ref, &psi_ref, &w_rw1, &w_rw2, &w_rw3);

        //Detect,
        if (t_count >= 5) {
            if (abs(w1 - w1_est) > fog_er) {
                //w1 = w1_est;
                fault_id = '1';
                //fault_flag = 1;
            }
            if (abs(w2 - w2_est) > fog_er) {
```

```
            //w2 = w2_est;
            fault_id = '2';
            //fault_flag = 1;
        }
        if (abs(w3 - w3_est) > fog_er) {
            //w3 = w3_est;
            fault_id = '3';
            //fault_flag = 1;
        }
    }

    // fault isolation, recovery
    if (fault_id != 0) {
        if (find_fault_id(fault_id)) {
            find_IR_function();
            perform_isolation();
            perform_recovery();
            *msg = NULL;
            fault_id = 0;
        } else {
            fault_id = 0;       //temporary///
        }
    }

    //com2.printf("%f,%f,%f,%f,%f,%f,%f\r\n", w1, w2, w3, q1, q2, q3, q4);

    //sscanf(fog_msg,"D,%f,%f,%f", &w1, &w2, &w3);
    //sscanf(stt_msg,"D,%f,%f,%f,%f", &q1, &q2, &q3, &q4);
    //sscanf(stt_msg,"D,%f,%f,%f,%f", &q1_ref, &q2_ref, &q3_ref, &q4_ref);

    //Target direction calculation
    //quaternion target is calculated from set of Euler angle
    //provided by ADS
    q4_ref = cos (phi_ref / 2.0) * cos(theta_ref / 2.0) * cos (psi_ref / 2.0) +
                             sin (phi_ref / 2.0) *sin (theta_ref / 2.0)  * sin (psi_ref / 2.0);
    q1_ref = sin (phi_ref / 2.0) * cos(theta_ref / 2.0) * cos (psi_ref / 2.0) –
                             cos (phi_ref / 2.0) *sin (theta_ref / 2.0)  * sin (psi_ref / 2.0);
    q2_ref = cos (phi_ref / 2.0) * sin(theta_ref / 2.0) * cos (psi_ref / 2.0)  +
                             sin (phi_ref / 2.0) *cos (theta_ref / 2.0)  * sin (psi_ref / 2.0);
    q3_ref = cos (phi_ref / 2.0) * cos(theta_ref / 2.0) * sin (psi_ref / 2.0) –
                              sin (phi_ref / 2.0) *sin (theta_ref / 2.0)  * cos (psi_ref / 2.0);

    //Quaternion positive
    if (q4_ref <= 0) {
        q1_ref = - q1_ref;
        q2_ref = - q2_ref;
        q3_ref = - q3_ref;
        q4_ref = - q4_ref;
    }

    //PD controller

    //Calculate quaternion error
    //
    eq1 =   q4_ref * q1 + q3_ref * q2 - q2_ref * q3 - q1_ref * q4;
    eq2 = - q3_ref * q1 + q4_ref * q2 + q1_ref * q3 - q2_ref * q4;
    eq3 =   q2_ref * q1 - q1_ref * q2 + q4_ref * q3 - q3_ref * q4;
    eq4 =   q1_ref * q1 + q2_ref * q2 + q3_ref * q3 + q4_ref * q4;

    // fprintf (' % f % f % f %f\r\n', eq1, eq2, eq3, eq4);
```

```
//Define travel trajectory
eq1 = sign (eq4) * eq1;
eq2 = sign (eq4) * eq2;
eq3 = sign (eq4) * eq3;

//Required torque need to be generated by actuators to achieve control target

T1 = - Kp * eq1 - Kd * w1;
T2 = - Kp * eq2 - Kd * w2;
T3 = - Kp * eq3 - Kd * w3;

//Use required torque to control RWs
//Send control command to RWs

RW_control();

//Satellite dynamics simulation for FDIR purposes

//Reaction wheel model
//Angular momentum generated from RW
h_rw1 = I_rw * w_rw1;
h_rw2 = I_rw * w_rw2;
h_rw3 = I_rw * w_rw3;

//Torque generated
T_rw1 = -(h_rw1 - ph_rw1)/obc_dt;
T_rw2 = -(h_rw2 - ph_rw2)/obc_dt;
T_rw3 = -(h_rw3 - ph_rw3)/obc_dt;

//Save it for the next cyle
//use previous value to calculate derivation
ph_rw1 = h_rw1;
ph_rw2 = h_rw2;
ph_rw3 = h_rw3;

// Satellite model
//
//Derivative of omega sat
w1_dot = (T_rw1 + w3_est * (Iy * w2_est + h_rw2) - w2_est * (Iz * w3_est + h_rw3)) / Ix;
w2_dot = (T_rw2 - w3_est * (Ix * w1_est + h_rw1) + w1_est * (Iz * w3_est + h_rw3)) / Iy;
w3_dot = (T_rw3 + w2_est * (Ix * w1_est + h_rw1) - w1_est * (Iy * w2_est + h_rw2)) / Iz;

// Estimate next state of satellite omega
w1_est = w1_est + w1_dot * obc_dt;
w2_est = w2_est + w2_dot * obc_dt;
w3_est = w3_est + w3_dot * obc_dt;

// fprintf (' % f % f % f \r\n', w1, w2, w3);

//Estimate next state of quaternion
//Estimate the normalize

q1_e = q1_e +   0.5 * (   q2_e * w3_est - q3_e * w2_est + q4_e * w1_est) * obc_dt;
q2_e = q2_e +   0.5 * ( - q1_e * w3_est + q3_e * w1_est + q4_e * w2_est) * obc_dt;
q3_e = q3_e +   0.5 * (   q1_e * w2_est - q2_e * w1_est + q4_e * w3_est) * obc_dt;
q4_e = q4_e + (- 0.5)*(   q1_e * w1_est + q2_e * w2_est + q3_e * w3_est) * obc_dt;

norm_q = sqrt (q1_e * q1_e + q2_e * q2_e + q3_e * q3_e + q4_e * q4_e);
q1_e = q1_e / norm_q;
q2_e = q2_e / norm_q;
q3_e = q3_e / norm_q;
```

```
        q4_e = q4_e / norm_q;

        //Telemetry
        //com1.printf("%f %f %f %d\n",T1,T2,T3,mode);
        //com3.printf("%f %f %f\r\n",w1-w1_est, w2-w2_est, w3-w3_est);
        //com3.printf("%f %f %f\r\n",T1-T_rw1, T2-T_rw2, T3-T_rw3);

               //TLM_CMD.printf("%f %f %f %f %f %f %f %f %f %f %d\r\n",w1,w2,w3,q1,q2,q3,q4,T1,T2,T
               3,fault_flag);

              //com3.printf("%f %f %f %f %f %f %f %f %f %f %d\r\n",w1,w2,w3,q1,q2,q3,q4,T1,T2,T3,fault_fl
              ag);
        //com.printf("%d\r\n",fault_flag);

        myled = !myled;

        t_count = t_count + 1;
        if (t_count > 254) t_count = 5;
      }
    }
    return 0;
}
/*******************************************************************************************/
```