AIAA Guidance, Navigation, and Control Conference
2 - 5 August 2010, Toronto, Ontario Canada

AIAA 2010-8301

Fault Tolerance for Spacecraft Attitude Management

Ali Nasir[*] and Ella M. Atkins.[†]
University of Michigan, Ann Arbor, Michigan, 48105

Abstract

We present an autonomy architecture called Fault Tolerant Remote Agent that integrates symbolic reasoning from AI planning/scheduling with physics-based fault-tolerant control. Application to spacecraft attitude management in the presence of diverse failure classes is studied. We first review fault tolerance in AI and control-theoretic contexts and introduce an architecture in which the capabilities of each can be integrated into a more comprehensive fault management framework. We then present fault identification and reconfiguration algorithms for a spacecraft attitude control case study. Simulation results demonstrate good recovery by the spacecraft for situations in which controllability is not lost. These simulations also illustrate how logic-based and physics-based algorithms cooperatively achieve a more comprehensive fault management capability than would be possible with either algorithm class alone.

Nomenclature

$\mathbf{q}$	=	Spacecraft attitude quaternion with respect to an inertial reference frame (4 elements)
$\mathbf{\Omega}$	=	Actual angular velocity vector of spacecraft in a body-fixed frame (3 elements)
$\mathbf{u}$	=	Control force vector in a body fixed frame (3 elements)
$\mathbf{J}$	=	Inertia matrix in a body fixed frame (3×3)
$\mathbf{C}$	=	Control gain matrix (3×3)
$\mathbf{K}$	=	Control gain matrix (3×3)
$\mathbf{x}$	=	State vector of the spacecraft in continuous time (7×1)
$\mathbf{x_n}$	=	Pre-computed nominal state vector in continuous time.
$\mathbf{I}$	=	Identity matrix (3×3)
$\mathbf{X_k}$	=	Discrete State Vector at time step *k*
$\mathbf{U_k}$	=	Internal Command Vector at time step *k*
$\mathbf{U\_x_k}$	=	Part of Internal command vector that contain desired position and velocities.
$\mathbf{U\_r_k}$	=	Part of Internal command vector that contain desired valve/switch modes.
$\mathbf{(Sup)_k}$	=	Vector of information from Supervisor to the Executive; $\mathbf{(Sup)_k = [F_k, P_k}, O_k, Q_k\mathbf{]}$
$\mathbf{(MI)_k}$	=	Vector of information from Mode ID to the Executive; $\mathbf{(MI)_k = [F\_MI_k , P\_MI_k]}$
$\mathbf{F_k}$	=	Fault vector from supervisor or fault detection scheme at time step *k*
$\mathbf{F\_MI_k}$	=	Fault vector from Mode Identification unit at time step *k*.
$\mathbf{P_k}$	=	Vector containing probability info for each component in $\mathbf{F_k}$
$\mathbf{P\_MI_k}$	=	Vector containing probability info for each component in $\mathbf{F\_MI_k}$
O_k	=	Variable containing observability information
Q_k	=	Variable containing controllability information
M_k	=	Variable containing mode information
$FLAG\_Sup_k$	=	Variable containing information about reconfiguration permition
$FAIL\_flag$	=	Variable indicating failure of recovery search by Mode Reconfiguration
$\mathbf{Cmd\_REC_k}$	=	Vector containing commands needed for recovery.
$\mathbf{Cmd\_MR_k}$	=	Contains information for recovery search; $\mathbf{Cmd\_MR_k = [F\_MI_k , X\_des_k, Constr_k]}$
$\mathbf{Constr_k}$	=	Vector encoding Constraints in the system.
$\mathbf{NV}$	=	Vector of zeros with appropriate size
$\mathbf{Y_k}$	=	Sensor data at time k (from all the sensors throughout the system)

Note: Subscript k everywhere signifies values of variables and vectors at infinitesimally small time step k.

[*] Graduate Student, Aerospace Engineering, Ann Arbor, MI 48109, email: techibro@umich.edu .
[†] Associate Professor, Aerospace Engineering, Ann Arbor, MI 48109, email: ematkins@umich.edu, Associate Fellow.

I Introduction

Once a spacecraft is launched, its hardware cannot be repaired. Therefore any component or system failure must be managed without replacement parts. To-date, spacecraft missions have been deployed with the ability to execute a "safe" mode transition in which a detected anomaly results in the spacecraft deactivating all thrusters and postponing science activities until the situation is resolved by engineers at the ground station. This solution has been sufficient to enable recovery from a variety of hardware and software failures so long as the spacecraft faces no immediate risks (e.g., collision) and provided the mission can be continued following a potentially extended "safe mode" episode.

With extended communication delays, increasingly complex science and maneuver schedules, and improved sensing and computational capabilities, future missions will support increased onboard autonomy, including autonomous fault management. Repeated safe mode transitions can be costly to the mission; safe mode is also not an option during critical mission phases such as one-time orbit insertions. As an Earth-orbiting example, consider a communication satellite tasked with providing coverage of a specific high-priority event. If this satellite goes to safe mode during the event (and there is no backup or the backup is already in safe mode), substantial loss of revenue could occur, or worse if providing communications for an area of unrest or disaster.

Two communities of researchers, from control systems and computer science, have studied spacecraft fault management with the common goal of robust fault tolerance but in slightly different contexts. Our long-term research goal is to capture and extend pertinent and complementary models and methods from both communities to establish a more comprehensive fault management protocol. In this initial work, we present an integrated approach to autonomous fault management that utilizes fault-tolerant control for post-failure system identification and feedback control coupled with a goal-based mode identification and reconfiguration module. We call our proposed architecture "Fault Tolerant Remote Agent" due to our adoption of both a fault-tolerant control module integrated with a goal-based mode identification and reconfiguration strategy inspired by the Remote Agent architecture successfully deployed on the Deep Space One spacecraft just over a decade ago. The "Remote Agent" component is responsible for establishing and adapting task-level plans based on goals and spacecraft state (including health), while the fault-tolerant controller is responsible for establishing and maintaining stable attitude (or more generally) flight control. Below, we first discuss previous work in spacecraft fault management and fault tolerant control. Next, we present the basic structure, functionality, and implementation of our fault-tolerant remote agent (FTRA) architecture. We describe its application to a spacecraft attitude controller and describe results from a case study in which a spacecraft's attitude maneuvering thruster system experiences one or more failure(s). Although we present a specific instantiation of FTRA algorithms, the notion of integrating task-level (logic-based) and physics-level adaptation algorithms is general.

II Background

Researchers in the artificial intelligence (AI) community have proposed a variety of architectures for planning/scheduling and plan execution[1]. Most represent state as a list of symbolic (discrete) feature/value pairs, enabling search-based algorithms to decompose, select, and sequence activities appropriate for the designated task-level goal and the observed system state. Reasoning, typically based on Markov Decision Process models, has become a standard method for building optimal policies that allow an agent to act with incomplete or uncertain information about itself or its environment. Although common in the literature, few AI systems have successfully been deployed on space systems due to their computationally-intensive deliberative and often difficult to validate nature. Rather than extensively trade the nontrivial set of AI architectures, we primarily reference Remote Agent[2,3,4] due to its emphasis on fault detection and reconfiguration and its focus on space applications. Below, we describe this architecture in more detail since, although 15 years old, it represents one of the most successful multi-layer AI architectures implemented and deployed on a spacecraft.[‡]

[‡] Scheduling protocols such as ASPEN developed by JPL have been successfully deployed. They, however, have focused on science data processing with little emphasis on real-time fault management. In such systems it remains the case that a spacecraft with ASPEN typically goes to safe mode rather than automatically classifying and adapting to faults.

Following our description of Remote Agent as a representative AI architecture focusing on fault management, we describe fault-tolerant control[5,6,7]. Fault-tolerant control represents more traditional guidance, navigation, and control (GNC) models in which physical continuous-valued state and control input vectors are related through physics-based models to describe the motion of a system through its environment. As described below, fault tolerance is then achieved through a mode-based supervisor and control law adaptation.

A. The Remote Agent

Researchers from JPL and NASA Ames developed the Remote Agent AI architecture for spacecraft mission management[2]. Remote Agent was tested on the DS-1 spacecraft and consisted of five main components including: 1) Planning Experts (PE), 2) Mission Manager (MM), 3) Planner and Scheduler (PS), 4) Smart Executive (EXEC), and 5) Mode Identification and Reconfiguration (MIR). Planning Experts (PE) are on-board software modules that assist a task planner/scheduler either by computing solutions or by requesting new goals. For example, a navigation PE might request updates to main engine thrust goals based on its determination of the spacecraft orbit, and the attitude PE might provide estimated duration of specified turns and resulting resource consumption.

The Mission Manager (MM) initiates planning/scheduling activities based on the long-term mission profile and execution status updates. The executive (EXEC) provides spacecraft status data and requests plans from the mission manager. The mission profile is provided at launch and can be updated from the ground. MM determines the goals to achieve during the next mission phase, and combines them with current spacecraft status. By adding constraints to the plan request, MM restricts PS to generate only plans that are coherent with the overall mission. This decomposition of planning into long-term mission planning and short-term task planning enables RA to undertake an extended mission with minimal human intervention. Such multi-resolution planning architectures have previously been used for applications such as telescope science scheduling.

The planner/scheduler performs Iterative Refinement Search (IRS)[8] and chronological backtracking to encode a set of methods applicable to extend the existing partial plan. A plan database input to the Heuristic Scheduling Testbed System (HSTS)[9] framework records the consequences of each problem-solving step and performs consistency maintenance and propagation. Domain constraints are specified in the Domain Description Language (DDL)[10] within HSTS. Throughout, system state is described as a finite set of symbolic state variables with tokens used to describe both action and state literals. PS, a generative planner, uses classical search-based planning and scheduling and is efficiently implemented with persistent parallel threads. PS is able to handle non-classical goal types such as periodic goals, accumulation goals, and default goals.

EXEC is a robust event-driven and multi-threaded plan execution system. It provides a framework in which specific mission goals and spacecraft state can be used to customize control, diagnosis, and reconfiguration capabilities autonomously. It can request and execute plans involving concurrent and interdependent activities potentially with uncertain timing and outcomes. EXEC decomposes planned tasks into primitive commands executed closed loop (i.e., as a function of state). This enables the planner to reason at a higher level of abstraction. EXEC's design also supports close integration between activity decomposition and fault response. EXEC is built on the Execution Support Language (ESL)[11] providing parallel execution, synchronization, error handling, and property locks[10]. EXEC loads and executes each plan while monitoring execution (spacecraft) status through Mode Identification (MI). When a plan is completed successfully, EXEC provides current status to MM and asks for a new plan. If task execution fails, EXEC puts the spacecraft into safe mode but autonomously asks MM rather than a ground operator for an alternate plan (unless MM can no longer resolve the problem in which case ground operators must be involved). EXEC achieves robustness by exploiting flexibility to create and modify plans based on goals and observations and by handling execution failures using deductive plan repair (Mode Reconfiguration (MR)).

Livingstone[12] provided the Mode Identification and Reconfiguration (MIR) functionality of Remote Agent. Livingstone is a discrete model-based controller inserted between high-level feed-forward reasoning and low level feedback control layers in a physical system. MIR proposes activities to migrate a system (spacecraft) to a configuration that achieves a configuration goal. It has a sensing component, Mode Identification (MI), and a commanding component, Mode Reconfiguration (MR). Its model is declarative, compositional, and stochastic with concurrency support. Mode Identification tracks changes in spacecraft status using input from EXEC as well as a spacecraft system model. It predicts state values and compares them with monitored values. In case of discrepancy, it predicts the malfunction or fault most likely to explain the discrepancy. Mode reconfiguration (MR) assists EXEC in generating recovery procedures. On the occurrence of a fault, EXEC invokes MR with current fault information from MI. EXEC also provides MR with global constraints and goals. MR performs deduction and search in a reactive loop using fast propositional reasoning through unit propagation along with conflict directed best-first search.

B. Fault Tolerant Control Systems

While the MIR capability of RA can identify and respond to faults via discrete event state (mode) models, MIR does not itself regulate continuous force/torque commands, nor does it adapt to properties of physics-based models except by switching between pre-specified modes. The control systems community has studied fault management primarily in the context of adapting physical models and control commands. A class of architectures known as Fault Tolerant Control System (FTCS) has emerged. A typical FTCS has three layers[5,6,13]. The lowest layer is a reconfigurable feedback control law with state estimator. The middle layer is a fault detection and diagnosis (FDD)[7] scheme. At the top level is a supervisor that manages reconfiguration of the FDD and control layers.

FTCS have adopted numerous control law formulations. In this manuscript we focus on a scheme applicable to spacecraft attitude control. Two types of fault tolerant controllers exist: passive strategies (robust control)[6] and active strategies (controllers for which reconfiguration is based on projection or on-line controller adaptation). Passive fault-tolerant control for a spacecraft uses a robust controller, providing a baseline upon which an active fault tolerant scheme could be implemented. Since the robustness of a controller has an effect on fault detection efficiency, a tradeoff between the two must be established. An active control approach can be implemented using projection or adaptive feedback control approaches[6]. The main purpose of this layer of FTCS is to adapt to anomalous situations and either to restore nominal performance, if possible, or gracefully degrade[14].

Fault Detection and Diagnosis (FDD) predicts faults from residual signals[15] based on sensor measurements and fault effect models. When the system is fault-free, all residuals should be driven to zero by the controller. An FDD scheme could be based on state or parameter estimation or a mixture of both. FDD can be classified into two groups: model-based approaches and data-based approaches[6]. A fault detection and diagnosis scheme should be robust, especially when model-based[16]. With sound but imperfect FDD models or incoming data, missed detections or a false alarm can occur. The decision-making in FDD can be made robust using various robustness methods such as statistical data processing, averaging, fuzzy decision-making, and adaptive thresholds[16,17]. Another issue is to distinguish between disturbances, noise, and faults. Disturbance decoupling methods can also be applied. For a complex system such as a spacecraft, decoupling of residuals from a set of integrated disturbances sometimes makes the residual completely or partially insensitive to some faults. From the point of view of a spacecraft, an FDD scheme should be able to detect multiple simultaneous faults, both abrupt and incipient.

Supervision is the top FTCS layer and is responsible for reconfiguration decisions based on information from FDD and its own reasoning algorithms. Supervision schemes have been developed to manage diagnostic information and on-line controller restructuring. Supervision modules have been implemented with methods[6,5] including Failure Mode Effect Analysis (FMEA)[5], Intelligent Computing, Fuzzy Logic, Neuro-Computing, Genetic algorithms, and Probabilistic reasoning. FMEA models the effect of faults on observable system parameters by providing data on how each fault impacts these parameters. The supervisor can be implemented using extended state machine[5] or parallel state machine logic with transition probabilities based on knowledge of the system.

C. Representational Gaps

We seek to establish the most complete fault management architecture possible, but we also seek a tractable solution that avoids (or minimizes) duplication of calculations. As a start, we compare the AI-derived Remote Agent architecture and fault-tolerant control for a spacecraft. Remote Agent models a spacecraft as a concurrent transition system with multiple components and operating modes, while FTCS models the spacecraft as a rigid or flexible body with associated kinematics and dynamics. Both rely on sensor data fused into state estimates and translated to control output, where “control” is defined for Remote Agent as general action primitives and for a FTCS as a vector of physical servo/motor commands. RA reasons about spacecraft components and their interactions but typically does not manipulate physical dynamics/kinematics parameters. On the other hand, FTCS can reason on the basis of equations of motion but is unable to reason about component interactions and task-level algorithm or software implementation properties. This difference establishes *representational gaps* in both architectures. One might be tempted to bridge these gaps by extending the capabilities of either RA or FTCS. Incorporating dynamics and kinematics reasoning in symbolic models is possible but difficult due to the tradeoff in [discrete model] resolution versus search-space tractability. On the other hand, incorporating qualitative component and interconnection details in FTCS not only requires a state-based supervision architecture that can reason about system-wide interactions but also the ability to reconfigure (replan) based on component failures and events associated with components such as communication channels, processing elements, the payload, etc. We propose an alternative in which representational gaps are filled by integrating remote agent with fault tolerant control, allowing each to fill gaps in the other architecture.

Below, we introduce the resulting architecture i.e., Fault Tolerant Remote Agent (FTRA). We then present a simple example of spacecraft attitude management illustrating how the integrated system synergistically models and reasons about component and control-centric faults.

III Fault Tolerant Remote Agent

A. The Architecture

Figure 1 shows the Fault Tolerant Remote Agent (FTRA) architecture. Since a Remote Agent-class AI architecture would be attached to the feedback control system of a spacecraft, the most natural way of incorporating FTCS is to replace the single traditional controller with a three layer FTCS, illustrated in detail in Figure 2. In the new architecture, the commands from Remote Agent's EXEC are passed to the FTCS supervisor. Fault information from the FTCS FDD is forwarded to EXEC through the supervisor. The monitored values from the sensors along with FDD information are conveyed to MI. Since MI and FDD have different model representations, data must be translated before it is delivered from FDD to MI and vice versa.

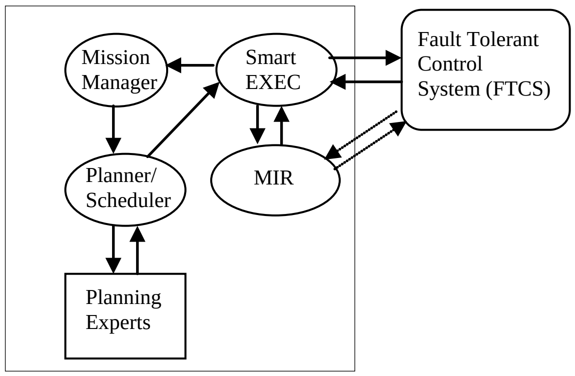

Figure 1: Fault Tolerant Remote Agent

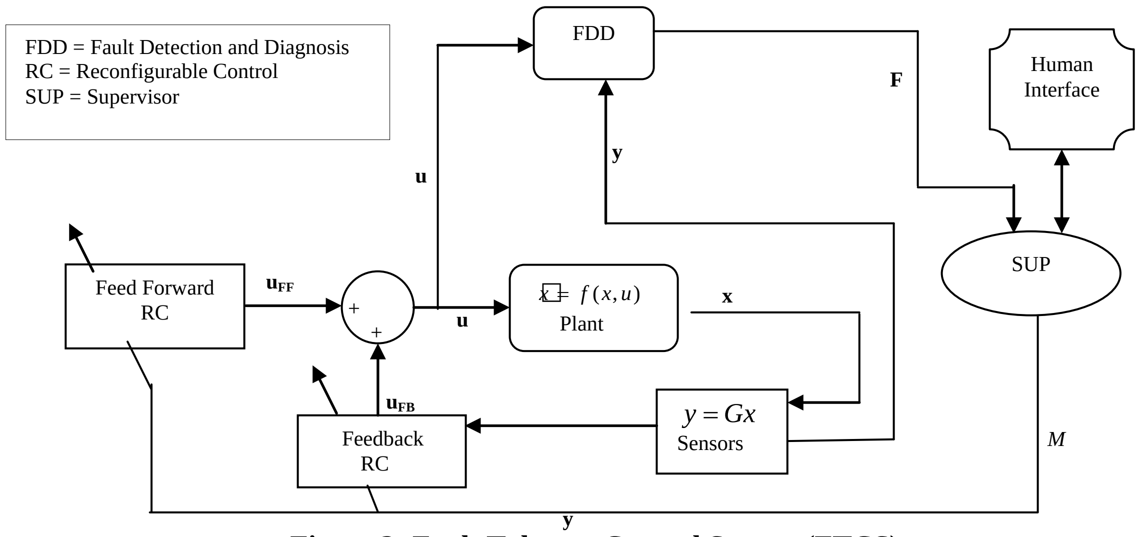

Figure 2: Fault Tolerant Control System (FTCS)

B. Functionality

The merger of FTCS and RA introduces new communication channels. The first is between EXEC and supervisor; the second is between FDD and MI. EXEC transmits task-level commands to the supervisor in cases where such commands can influence FTCS operational mode (e.g., selecting a different control mode due to a different rate or mode of motion). EXEC can also communicate MIR results to the supervisor, guiding the supervisor's choice of control options based on failed or reconfigured components or goals. For this discussion we presume the supervisor is implemented with FMEA extended state machine logic. FMEA determines and transfers its state (describing the fault's effects) to EXEC. It can also transfer the probabilities of other faulty or normal states based on the current state of the spacecraft and the FMEA uncertainty model. Since EXEC now has access to two sources of fault information and two models capable of reconfiguration, it has the choice of allowing or disallowing reconfiguration of the supervisor control loop, or altering invocation of or parameters in MIR. Ideally MIR and the FTCS supervisor would handle a comprehensive and non-overlapping fault set, but in practice there will likely be overlap to be studied here and in our future work. Below we present initial interfaces and demonstrate their use for spacecraft attitude control fault management. While overlapping fault sets between MIR and the supervisor may be difficult to manage, as discussed below these independent modules also provide a means of independently validating solutions that are consistent between the two modules.

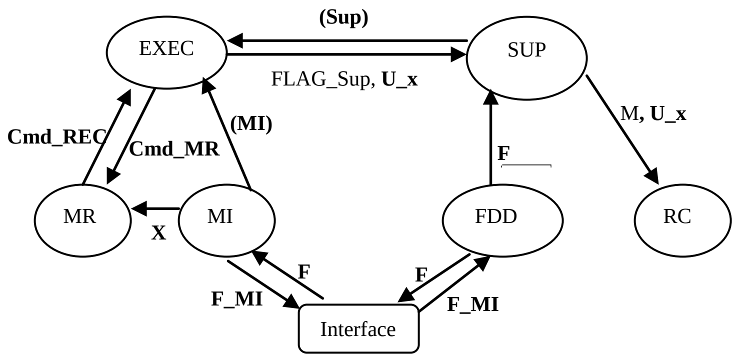

Figure 3: Communication links in FTRA

Figure 3 illustrates the integration of the Remote Agent (RA) and fault-tolerant control (FTC) layers. For simplicity the planning/scheduling (PS) layer of RA is omitted in the above figure since its only direct connection to layers in Figure 3 is through the RA executive (EXEC). Recall that RA included executive (EXEC), mode/fault identification (MI), and mode/fault reconfiguration (MR) components. Similarly, recall that FTC consisted of supervisor (Sup), fault detection and diagnosis (FDD), and reconfigurable control (RC) layers. Connections of RC to sensors and actuators in its environment are omitted in the figure for simplicity. The two communication layers connecting RA and FTC layers are a focus of our work. The FDD/MI interface maximizes information redundancy to facilitate fault diagnosis/identification decisions and minimize the potential of reaching contradicting conclusions with FDD vs. MI. When MI detects an anomaly by comparing monitored with predicted data, it uses a declarative compositional probabilistic concurrent transition model to determine the fault set that justifies the anomaly. With additional data from FDD, it can double-check the identified fault by comparing its conclusion with that of FDD. Similarly, FDD can incorporate fault information from MI into its fault detection algorithm to justify a decision of fault. With this strategy, each is using the output of the other but neither MI nor FDD is aware of the other's inference process. We consider this a plausible first implementation, particularly given that computed probabilities are shared between the two modules. We recognize that correlation of internal process may improve efficiency and accuracy in future extensions, although substantial correlation of process might compromise solution independence and the associated system-level robustness.

There are several issues to address in MI/FDD integration. First, even in cases where MI and FDD would independently diagnose consistent fault(s), they may not reach their conclusions simultaneously.

Next, either one could be wrong in its conclusion, resulting in contradictory results. Third, some faults (e.g., related to changes in kinematics/dynamics) may only be detectable by FDD, while MI might exclusively detect failures not related to physical motion sensing or actuation. Finally, since, EXEC is responsible for the resolution of discrepancies, it must be maximally-informed by both the MI and FDD processes.

Information exchange between MI and FDD can be useful in a number of ways. Fault information from one layer can strengthen certainty of conclusions in the other, e.g. when FDD residuals are close to the detection threshold, or when two or more MI anomalies have similar probabilities. As discussed above, agreement between MI and FDD conclusions certainly will strengthen confidence of the diagnosis. Also, each can alert the other upon anomaly detection, which may impact the decision-making parameters as well as the decision itself. Together, information exchange supports resolution of discrepancy between the two schemes, exploits faster detection of a fault by one layer, and builds confidence/robustness of the decision in cases of agreement between the two schemes. During normal operation, EXEC will execute the current plan and receive system health and state updates from MI and the FTCS supervisor. MI can detect a significant spectrum of faults observable from discrepancies between predicted and actual state due to its flexible compositional constraint-based modeling with concurrent transition support. On the other hand, FDD concentrates on discrepancies in continuous-time state, not in components or logical state. For example, the FTCS will have model a thruster in terms of the force it can apply and fuel it consumes, but it would have no information about the valves used to supply fuel to the thrusters.[§] Other examples FDD would be unlikely to detect include component overheating, or science equipment defects.

Regardless of which identifies the fault, both MI and FTCS send fault data to EXEC. EXEC invokes MR by default, except in cases where only FDD has identified the control-centric fault condition. EXEC will monitor the FTCS supervisor's reconfiguration activities and invoke planning/scheduling activities in cases where its original task set may no longer be valid given supervisor/FDD fault management decisions (e.g., a reduced control authority mode will no longer support completion of a particular task). To arbitrate, EXEC must have a mapping of faults MI and FDD can detect, along with a strategy (e.g., a simple priority or confidence measure) for arbitrating between two contradicting results. This situation generally can occur either due to MI or FDD being incapable of detecting the fault, or due to discrepancies in the MI vs. FDD models. For example, the former case would be present with a stuck thruster valve. In this case MI would conclude that the valve is stuck, while FDD would conclude that the thruster is not functioning (without further diagnosis). While neither conclusion is incorrect, in this case EXEC's confidence/conclusion mapping would indicate the "stuck valve" diagnosis was a more descriptive (thus useful) diagnosis. MR would then be invoked to respond appropriately.

EXEC (with feedback from MR) will invoke the planner/scheduler module in cases where existing goal-level tasks become infeasible due to a diagnosed fault, or more generally when a particular set of goals has been accomplished. In cases where EXEC needs a new plan and the planner/scheduler requires additional deliberation time, EXEC itself will be capable of transitioning the spacecraft into safe mode. Rather than involving ground personnel (which of course would still be an option in some situations), the planner/scheduler then re-plans, considering the effects of identified faults as well as the mission goals and domain model. In cases where MI and FDD conclusions cannot possibly be consistent even after consideration of shared data and with tolerance of uncertainties, EXEC will request ground operator input.

IV Case Study: Fault Tolerant Attitude Management

The FTRA architecture described above has been translated into a prototype implementation to demonstrate FTRA functionality. In this section we present a simple example of spacecraft attitude control using FTRA to diagnose and reconfigure to a small but diverse set of anomalies. Attitude control is required for most spacecraft and is perhaps one of the most critical real-time control functions executed onboard a spacecraft. Attitude control is also of relevance for our case study because fault-tolerant performance has previously been studied separately in the context of MIR and FTCS. For our scenario, a simple cubical spacecraft (CubeSat)[**] structure is presumed. An onboard telescope is capable of imaging the solar system and distant celestial objects. We study a particular mission segment in which EXEC has initiated a pointing task commanding the spacecraft to reorient toward a particular celestial object (e.g., a

[§] A spacecraft thruster valve application was in fact one of the original Livingstone case studies[12].

[**] We use the term CubeSat as a general name for small-scale satellites potentially benefiting from a FTRA capability; our research is not connected to any particular CubeSat programs.

mapped asteroid) and take a series of images. We assume spacecraft attitude is adjusted only through low-magnitude thrusters affixed to each CubeSat face, and that CubeSat is a rigid body with constant mass and inertia.

A. Spacecraft Model

The dynamics and kinematics spacecraft models for FTCS are formulated below, followed by presentation of a compositional qualitative model used by the RA layers. Spacecraft attitude state is represented by a four-element quaternion vector **q** and angular velocity **Ω**. Equation (1) describes the spacecraft attitude kinematics:

$$\dot{q} = \frac{1}{2}\left(q_4\Omega - [\Omega \times q]\right)$$

$$\dot{q}_4 = -\frac{1}{2}\left(\Omega^T q\right) \qquad (1)$$

or

$$\dot{\mathrm{R}} = \begin{bmatrix} 0 & \Omega_3 & -\Omega_2 \\ -\Omega_3 & 0 & \Omega_1 \\ \Omega_2 & -\Omega_1 & 0 \end{bmatrix} \mathrm{R}$$

Equation (2) describes a spacecraft's rigid-body attitude dynamics:[19]

$$\dot{\Omega}_1 = \frac{1}{J_1}\left[(J_2 - J_3)\Omega_2\Omega_3 + u_1\right]$$

$$\dot{\Omega}_2 = \frac{1}{J_2}\left[(J_3 - J_1)\Omega_1\Omega_3 + u_2\right] \qquad (2)$$

$$\dot{\Omega}_3 = \frac{1}{J_3}\left[(J_1 - J_2)\Omega_2\Omega_1 + u_3\right]$$

Above, **Ω = [**Ω_1 Ω_2 Ω_3]^T is the angular velocity vector of the spacecraft in a body-fixed frame, **q** = [q_1 q_2 q_3 q_4]^T is the quaternion vector with q = [q_1 q_2 q_3]^T and scalar component q_4. R in alternate of equation (1) is 3 by 3 rotation matrix. Note that in simulations we verified that a rotation matrix formulation yields identical results, providing some validation of our simulation code. Control inputs are body-axis torques u_1, u_2 and u_3 generated by thruster pairs imparting opposite forces presumed to have no translation component. Moment of inertia matrix is **J** = diag(J_1, J_2, J_3). The control law in nominal no-fault conditions is

$$u = -Kq - C\Omega$$

$$K = \frac{k}{q_4^3} J \qquad (3)$$

$$C = diag(c_1, c_2, c_3)J$$

or

$$u = -K\begin{bmatrix} r_{23} - r_{32} \\ r_{31} - r_{13} \\ r_{12} - r_{21} \end{bmatrix} - C\Omega$$

$$K = \frac{k}{-0.5(1 + trace(R))^{\frac{3}{2}}} J$$

$$C = diag(c_1, c_2, c_3)J$$

where **u** = [u_1 u_2 u_3]^T, R is 3 by 3 rotation matrix and r_{ij} is the element of R belonging to column *j* and row *i*.

Below a compositional qualitative spacecraft model is presented for the thruster/valve system in which faults will be injected. Note that the full spacecraft RA model is beyond the scope of this paper. As shown in Figure 4, we assume our spacecraft is equipped with six thrusters (one per CubeSat face) arranged such that pairs of thrusters provide torques about x, y, and z axes. The spacecraft has nine thruster valves, three per thruster pair, and nine pressure sensors, one for each valve. Each propulsion system component is a state variable with different modes and associated constraints. Appropriate states, constraints, and sample mode transition probabilities are provided for our spacecraft attitude control domain in Tables 1-3 below.

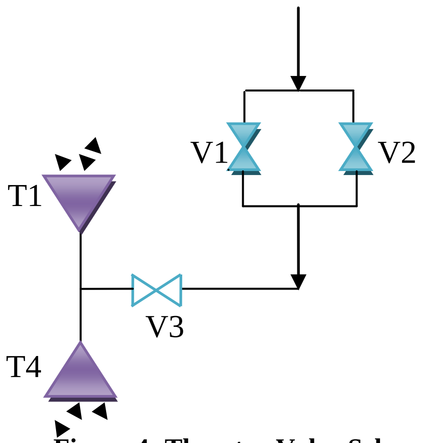

Figure 4: Thruster-Valve Scheme

Table 1: Example Spacecraft States

States	**Modes**
T_i	Normal {On, Off}, failed {No Thrust}
V_i	Normal {Open, Shut}, Failed {Stuck open, Stuck shut}
P_i	Normal {0-5V}, Failed {Constant 0V or 5V}

Table 2: Example Thruster Constraints

Modes	**Constraints**
T_1(On)	Valve V_1(On) or V_2(On) and, V_3(On)
T_1(Off)	Valve V_3(Off)

Table 3: Example Mode Transition Probabilities

Component	**Mode Transition Probabilities**
T_i	$P_{FF} = 0.99$, $P_{FN} = 0.01$, $P_{NN} = 0.95$, $P_{NF} = 0.05$
V_i	$P_{FF} = 0.95$, $P_{FN} = 0.05$, $P_{NN} = 0.80$, $P_{NF} = 0.2$
P_i	$P_{FF} = 0.8$, $P_{FN} = 0.2$, $P_{NN} = 0.85$, $P_{NF} = 0.15$

B. Fault Detection and Diagnosis (FDD)

Fault detection is implemented using a multi level residual generation and testing technique. On the first level, we compute the residuals from nominal versus observed state trajectories. For this purpose, we have nominal trajectories based on normal and faulty modes. On the second level, we generate fault rejection number N_{ij} for each fault case *i* and each state variable *j*. The fault rejection number is 1 if the residual from the first level crosses a threshold (*h*) which in our case is five times the anticipated disturbance and additive noise together (i.e. $h = 5\times10^{-6}$) and 0 otherwise. Note that N_{ij} is 0 when state *j* of the actual spacecraft agrees with the corresponding state of nominal mode *i*. On the third level, a fault hypothesis F is generated if the fault rejection number for a particular fault is less than a second threshold ($v \in \{0.9, 1.9, \ldots m\text{-}0.1\}$), where m is the number of states, 7 in our case study. *v* depicts how many states are in agreement with a given nominal model. In our case a value *v* of 0.9 means that all seven spacecraft states agree (i.e., fewer than 0.9 states disagree with values predicted from a nominal model). In this case we hypothesize that particular model to be the true model of the spacecraft. Equation (4) shows these computations:

$$
\begin{aligned}
e_{ij} &= \left|x_{ij}\text{-}x_{nj}\right| \\
N_{ij} &= \left(\frac{1\text{-}\dfrac{h\text{-}e_{ij}}{\left|h\text{-}e_{ij}\right|}}{2}\right) : \left|h\text{-}e_{ij}\right| \neq 0 \\
F_i &= \left(\frac{1\text{-}\dfrac{N_{ij}-v}{\left|N_{ij}-v\right|}}{2}\right) : \left|N_{ij}-v\right| \neq 0
\end{aligned}
\qquad (4)
$$

where $j \in \{1,2,3,4,5,6,7\}$ and $i \in \{0,1,2,3\ldots n\}$ with *n* the number of anticipated faults (0 means normal). Finally, if a hypothesis persists for a selected time window (in our case 1 sec), the fault is confirmed.

C. Reconfigurable Control Strategy

Our reconfigurable controller is comprised of a database of pre-computed control laws selectable from fault models as shown in Figure 5. This design is essentially the multiple model reconfigurable control scheme [16].

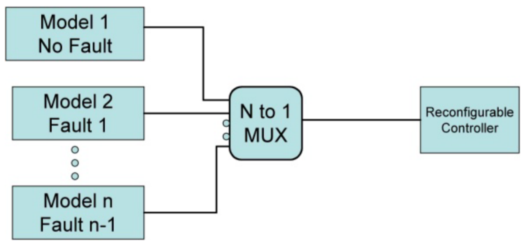

Figure 5: Multiple models Reconfiguration

Figure 5 shows our scheme is projection-based i.e., we have pre-computed models of the system with expected faults. When an FDD-identified fault is confirmed by the supervisor and EXEC concurs with its reconfiguration, the dynamic model is replaced and the controller is redesigned according to the new dynamics. There are eight possible combinations of nominal and failed thruster states representing the fact that each of the three thruster pairs can be either normal or failed. Here, we will show simulations for only one case that covers the three cases of loss-of-torque (thrust) about one of the three axes due to thruster valve failure. These three faults are characterized by $\mathbf{u} = [u_1\ u_2\ 0]^T$, $\mathbf{u} = [0\ u_2\ u_3]^T$, and $\mathbf{u} = [u_1\ 0\ u_3]^T$. Our attitude control strategy is derived in Appendix 1. In this appendix we provide control laws for a sequence of maneuvers that can reorient the spacecraft from an arbitrary initial state to a desired attitude and zero angular velocities given each of the specified fault scenarios. It might appear to the reader that these maneuvers take the spacecraft from arbitrary state to the equilibrium state i.e. $[0\ 0\ 0\ 1\ 0\ 0\ 0]^T$ but in fact we can use the same laws for any arbitrary desired pointing by defining quaternion components in the control law to be the error between initial and desired quaternion components (see [29] for detail).

D. The Supervisor

The supervisor is implemented using FMEA combined with extended state machine (ESM) logic. The example transition probabilities used in our simulations are given in Table 4. In this table, Faults 1, 2, and

3 correspond to loss of one, two and then three torque directions respectively. We assume identical probabilities of failure for the different torque axes, but indicate an increased likelihood of a second and third torque failure once the first is observed. The probability numbers used are equivalent for any ordering in which progressive loss-of-torque failures occur. This table is not complete in a sense that it does not cover all seven failure states but it covers all three categories of those seven failure modes i.e, single, double and triple failures.

Table 4: Example Fault Transition Probabilities

	Normal	**Fault 1**	**Fault 2**	**Fault 3**
Fault1	0.01	0.97	0	0
Fault 2	0.005	0.02	0.96	0
Fault 3	0.002	0.01	0.04	1
Normal	0.983	0	0	0

Each fault column represents probabilities of possible transitions from the corresponding fault to other faults or to itself. Occurrence of a fault is an event that prompts the supervisor to possibly switch "health" states. When FDD detects a fault, the supervisor checks for fault convergence. If the fault persists for appreciable time, it sends a message to EXEC with indicating the fault and the level of confidence in this fault state. EXEC will continue to monitor the supervisor and manage any associated reconfiguration, while the supervisor simply switches the system model and control strategy according to the next fault condition. Figure 6 shows the supervisor algorithm. Here, subscript *k* indicates the time step. $\mathbf{F_k}$ represents the fault vector, $\mathbf{y_k}$ and $\mathbf{u_k}$ are output and input vectors of the dynamic system respectively at time step *k*, *M* is the mode or finite state corresponding to no fault (0) or one of the faults (1-3) described above. Variable **x** is the continuous state vector, and $\mathbf{P_k}$ represents the probabilities associated with faults at time step *k* using Table 4 as the initial probability distribution. Q_k and O_k are variables carrying local controllability and observability properties respectively at time step k, where 0 indicates no controllability/observability, 1 indicates accessibility/distinguishability and 2 indicate full controllability/observability). FMEA observability and controllability calculations are defined in the Appendix 2.

```
Supervisor:
Input:  $F_k=[F1_k,F2_k,F3_k]$,  $y_k$, $u_k$, $FLAG\_Sup_k$
Output:  $M_k=\{0,1,2,3\}$, $U\_x_k = [x\_des]$, $P_k=[P_k(F1), P_k(F2), P_k(F3)]$
$[O_k, Q_k] \leftarrow$ FMEA ($F_k$, $y_k$, $u_k$)
if ($FLAG\_Sup_k$ == 1)
        send_RC ($M_k$, $U\_x_k$)
else
        send_RC (0, $U\_x_k$)
If ($F_k$ != [0,0,0])
        $P_k \leftarrow$ Get_Probability (ESM, $F_k$)
Else
        $P_k \leftarrow [0,0,0]$
End
$(SUP)_k \leftarrow [F_k, P_k, Q_k, O_k]$
End Main
FMEA:
Input:  $F_k=[F1_k,F2_k,F3_k]$, $y_k$, $u_k$
Output:  $O_k=\{0,1,2\}$, $Q_k=\{0,1,2\}$
Set $B_k$ and $G_k$ for $F_k$
$O_k$ = observability($y_k$, $u_k$ )
$Q_k$ = controllability($u_k$)
End
```

Figure 6: Supervisor Algorithm for the Spacecraft Attitude Control Case Study

E. EXEC Implementation

EXEC is implemented using the algorithm shown below in Figure 7. At the start of an execution cycle, the current partial plan is loaded into EXEC. At each iteration or time step *k*, EXEC receives system updates and health status through MI ($\mathbf{(MI)_k}$) and the Supervisor ($\mathbf{(Sup)_k}$). In case of a fault, two situations may take place: either there is a conflict between FDD fault and MI fault, or there is no conflict. In case of conflict, fault transition and mode transition probabilities provided by the Supervisor and MI respectively are utilized along with the qualitative EXEC spacecraft model (e.g., confidence in FDD vs. MI to correctly diagnose each fault) to deduce the most likely fault(s). Recovery actions may be executed through MR ($\mathbf{U\_Xr_k}$), the Supervisor ($\mathbf{U\_x_k}$, $FLAG\_sup_k$), or both. In case of an un-reconcilable situation, EXEC puts the spacecraft in safe mode and invokes the Mission Manager (planner/scheduler) for an alternate plan based on the current situation. If no appropriate plan can be found, the worst case, ground operators are contacted to resolve the conflict. In case of no fault or consistent diagnoses by MI and FDD, the normal or fault-accommodation execution cycle is continued, i.e., EXEC commands are executed according to the current plan.

In Figure 7, matrix $\mathbf{U_k}$ represents EXEC commands at time step *k*. Commands might be part of the normal execution sequence part of a recovery sequence, depending upon the situation. In both cases, command format is $\mathbf{U_k = [U\_x_k, U\_r_k]}$ where $\mathbf{U\_x_k}$ specifies continuous-time reference state for the attitude controller and second $\mathbf{U\_r_k}$ specifies commands to discrete controllers such as switches, valves, etc. $\mathbf{Cmd\_MR_k}$ contains information such as Fault information, desired state, and constraints ($\mathbf{Cmd\_MR_k = [F\_MI_k , X\_des_k, Constr_k]}$) that MR uses in finding recovery procedure if there is one. $\mathbf{Cmd\_REC_k}$ specifies commands that can take the spacecraft back to normal mode, these commands are essentially given to the switches or valves in the form of $\mathbf{U\_r_k}$. $\mathbf{X_k}$ represents the discrete state vector in representing component states, both in terms of commanded position (e.g., open/shut) and failure state (e.g., OK/failed).

```
Algorithm for EXEC:
Inputs: $(Sup)_k = [F_k, P_k, O_k, Q_k], (MI)_k = [F\_MI_k , P\_MI_k], Cmd\_REC_k$
Outputs: $FLAG\_sup_k$, $Cmd\_MR_k$, $U_k$
Start_Plan  (t_des, $X_k$)
loop: do
        $Is\_FAULT_k \leftarrow$ Check_Faults ($(MI)_k, (Sup)_k$)
        If (Is_Empty($Is\_FAULT_k$):
                Send_Commands (Compute_Commands($X\_des_k$, t_des))  // sends $U_k = [U\_x_k, U\_r_k]$
        Else:
                $CONF_k \leftarrow$ Check_Conflict ($F_k, F\_MI_k$)
                [$FLAG\_sup_k$ ,$Cmd\_MR_k$]$\leftarrow$ Resolve_for_Recovery($F_k, F\_MI_k$ , $CONF_k$)
                $Cmd\_REC_k \leftarrow$ Invok_MR ($Cmd\_MR_k$)
                Send_Rec_Commands ($Cmd\_REC_k$, $FLAG\_sup_k$)
End of loop
```

Figure 7: EXEC's Fault Recovery Algorithm

F. MIR Implementation

MIR is implemented using the algorithm set illustrated in Figure 8 and described in Figure 9. MI predicts spacecraft states based on the spacecraft model and EXEC inputs. Predicted states are then compared with the current state to identify any anomalous situation. In the case of fault, the MI result is compared with data from FDD. . MR receives fault information from EXEC along with constraints and goals. It performs model-based search for the best reconfiguration strategy with real-time updates from MI and forwards the proposed solution to EXEC

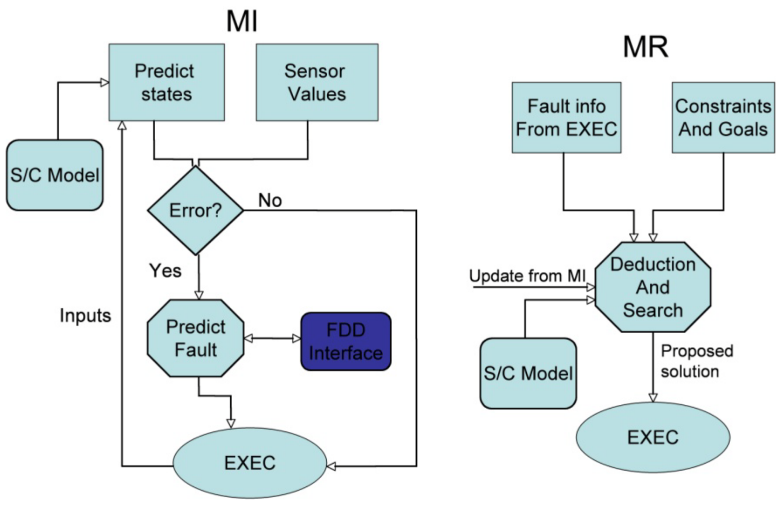

Figure 8: MIR Block Diagram

The Figure 9 variables have the same meanings as those defined in Figures 6 and 7; additionally, **y** is the output vector at time step *k* from all the sensors in the spacecraft including attitude control sensors and valve pressure sensors and **NV** means null values or zeros. In this work we presume valves are either open or shut, functioning or stuck open or shut, so valves are controlled with discrete rather than continuous reference state commands.

```
Algorithm for MR:
Inputs: $Cmd\_MR_k = [F\_MI_k, X\_des_k, Constr_k], (MI)_k, X_k$
Outputs: $Cmd\_REC_k$, FAIL_flag
$Cmd\_REC_k \leftarrow$ Search_Algorithm($Cmd\_MR_k, X_k$)
If (Is_Solved ($Cmd\_MR_k, Cmd\_REC_k$)):
        Send_EXEC ($Cmd\_REC_k$)
Else:
        Send_EXEC (FAIL_flag)
End
```

```
Algorithm for MI:
Inputs: $Y_k, U_k$
Outputs: $(MI)_k = [F\_MI_k, P\_MI_k, X_k]$
$X\_est_k \leftarrow$ PredictState ($X_{k-1}, U_{k-1}$)
$F\_MI_k \leftarrow$ Compare ($X\_est_k, Y_k, F_k$)
If (Not_Empty ($F\_MI_k$)):
        $X_k \leftarrow$ Update_State ($F\_MI_k, X_{k-1}$)
        $P\_MI_k \leftarrow$ Update Probabilities $F\_MI_k, X_k, X_{k-1}$)
        $(MI)_k \leftarrow [F\_MI_k, P\_MI_k, X_k]$
Else:
$X_k \leftarrow X\_est_k$
$(MI)_k \leftarrow [X_k, NV]$
Send_EXEC ($(MI)_k$)   //End
```

Figure 9: MIR Algorithms

G. Results

Below, we present simulation results for our selected fault case. We implemented all algorithms in MATLAB for this work. In our simulation, the rigid CubeSat is commanded to conduct an attitude maneuver from $q = [0.5\ 0.5\ 0.5]^T$, $q_4 = -0.5$ to $q = [0\ 0\ 0]^T$, $q_4 = -1$. For our satellite $J_1 = 1700$ kg·m^2, $J_2 = 2000$ kg·m^2, and $J_3 = 1400$ kg·m^2. We constrain the time allowed to complete the maneuver to 500 sec, and the initial velocities are presumed to be $\mathbf{\Omega}_0 = [0.1\ 0.15\ 0.2]^T$. EXEC sends the desired maneuver to the FTCS reconfigurable controller, but in our initial case study there is no thrust in u3 (i.e., no torque about the z axis) due to a valve being stuck. This fault is detectable by both MI and FDD, but FDD detects it as thruster failure while MI detects it as a valve failure. EXEC receives fault information from the Supervisor and MIR. EXEC sends the potential "valve failure" information to MR, given that this diagnosis is consistent with and more expressive than the "thruster failure" result from FDD. MR in return suggests valve reconfiguration. EXEC reconfigures the thruster values per the MR result, thereby regaining control authority. In the plots shown in Figures 10 and 11, reconfiguration is initiated at t = 5 sec. Since the mission-level goal was to achieve the desired pointing attitude within 500 sec, our results indicate that, with our proposed scheme, the goal has been achieved. Note that the control gains were not changed during this process since MR was able to restore functionality without Supervisor reconfiguration. This illustrates MIR utility distinct from FTCS functionality. In the Figure 10-12 plots, the green lines indicate reconfigured trajectories, while red lines illustrate behavior without valve reconfiguration. For comparison, the normal no-fault trajectory is plotted in blue. Although the failure does introduce additional transient, reconfiguration enables the system to regain performance comparable to the no-fault condition in this case. We have also presented the results using an alternative rotation matrix formulation (Figure 11), yielding results consistent with quaternion-based trends.

To complete the demonstration of our idea, we also present the case where the thrusters related to *u3* fail. In that case, both MI and FDD are still able to detect the fault and this time there is no conflict between their diagnoses i.e. both schemes detect this situation as thruster failure. Now MR has no way of bringing the spacecraft to any graceful configuration, so EXEC commands controller reconfiguration and the new control laws shown in Appendix 2 are invoked. Results are shown in Figures 13-16. Note that we used a discontinuous control strategy because although the spacecraft is locally controllable, it cannot be asymptotically stabilized to any equilibrium attitude using time invariant continuous feedback[29]. Another noticeable difference here is that the rotation matrix based controller completes the maneuver slightly earlier than the quaternion based controller. This is due to the presence of alternate ways to maneuver to the desired orientation.

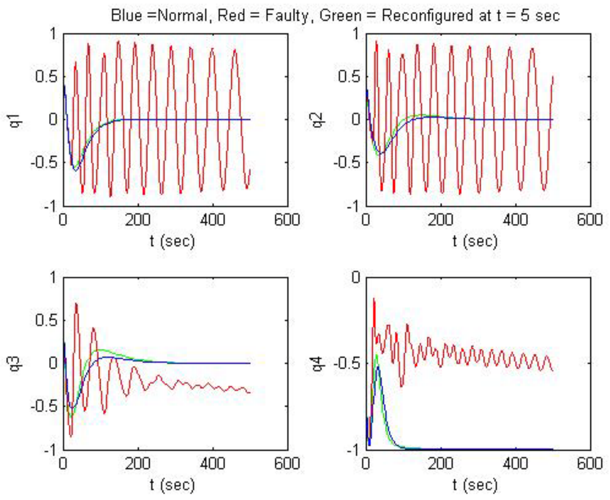

Figure 10: Quaternion Values for Valve Failure and Reconfiguration case

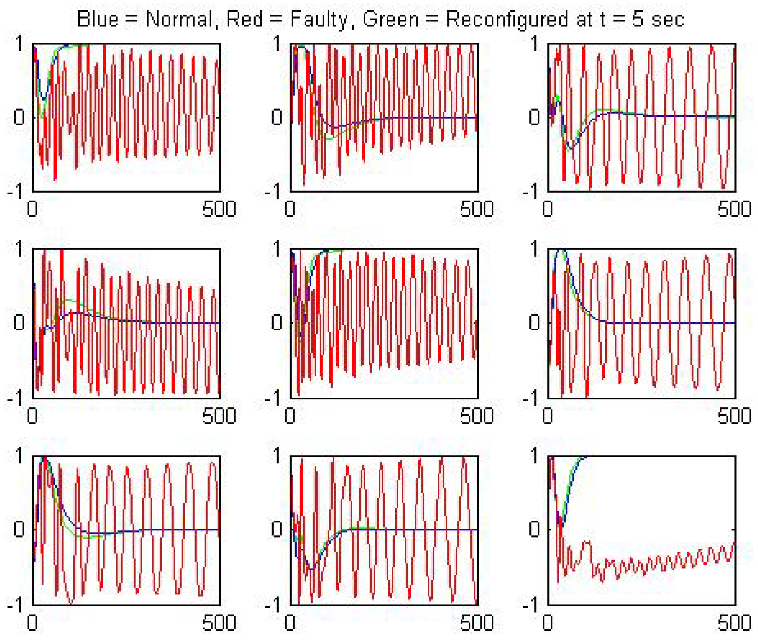

Figure 11: Values of Rotation matrix for valve reconfiguration case

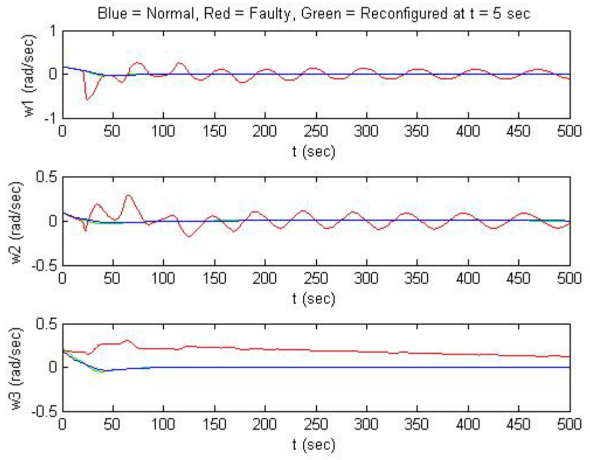

Figure 12: Angular Velocities for Valve Failure and Reconfiguration case

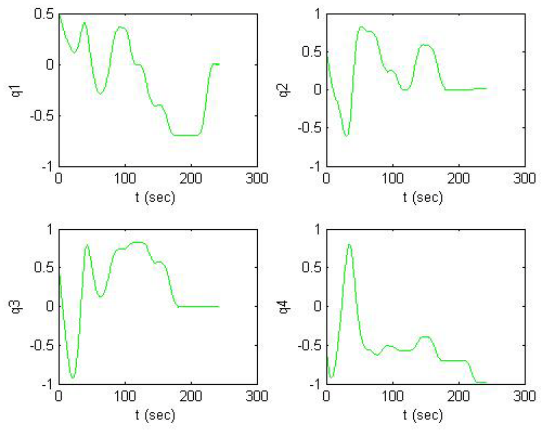

Figure 13: Quaternion Values for Thruster Failure Case

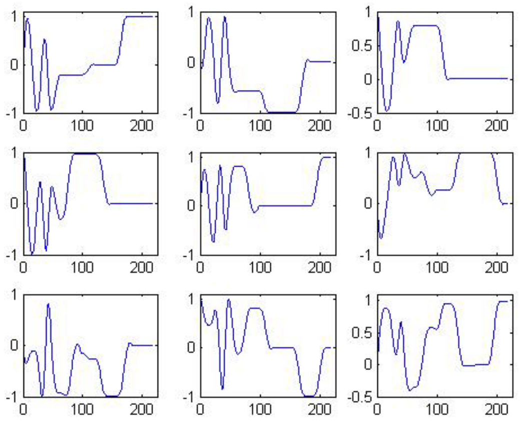

Figure14: Rotation matrix values for thruster failure case

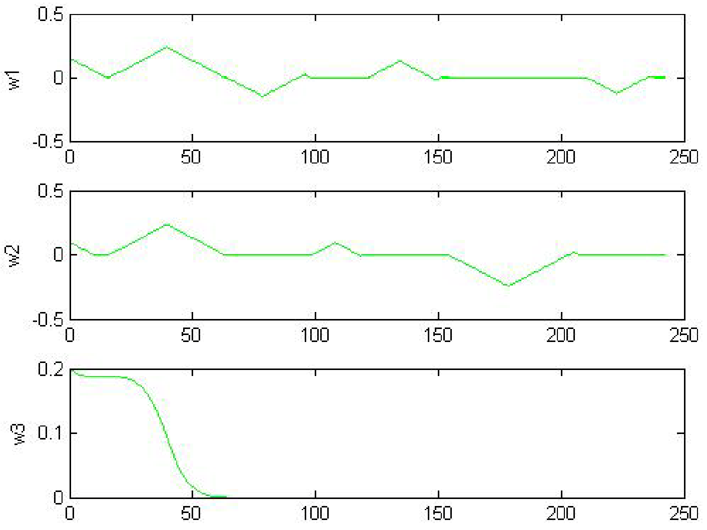

Figure 15: Angular Velocities for Thruster Failure Case (using quaternion based control)

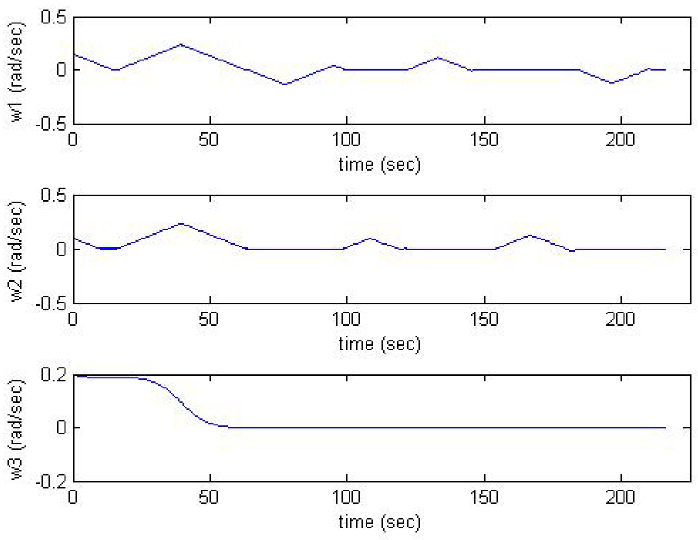

Figure16: Angular Velocities for Thruster Failure Case (using Rotation matrix based control)

V Conclusions and Future Work

In this paper we have proposed the integration of Remote Agent, an AI planning/scheduling architecture with fault management capability, with a fault-tolerant feedback control system to provide a general fault management solution capable of identifying and executing both logic-based and continuous state reconfiguration strategies. We have demonstrated application to fault-tolerant spacecraft attitude control. The proposed scheme offers a number of advantages, at a cost of increased architectural and model complexity. Some advantages include increased fault coverage, increased reliability through redundancy, handling of fault propagation, ability to autonomously recover, and handling of concurrent faults. Computational complexity tradeoffs are dependent on model complexity and overhead and must be further analyzed in future work. Both RA and FTC may require computationally-intensive search to identify or optimize a solution, but particularly if executed in parallel overhead may not be substantial compared to either RA or FTC executing alone since complexity *order* does not increase given independent execution threads. Additionally, RA and FTC models may be marginally simpler since faults most naturally expressed in the other's representation need not be fit into the alternate architecture (except when needed for redundancy).

The benefits to the integrated fault-tolerant remote agent (FTRA) versus either RA or FTC managing faults alone depend on several factors. First, if the level of autonomous decision-making is low, an AI approach may not be warranted; it may instead be feasible for a fault-tolerant controller to make adaptations to ensure stability while the human operator manages other reconfiguration decisions. Conversely, if system dynamics are trivial (e.g., a stationary plant where "decisions" are in terms of data, not actuation) there is no role for FTC. Increasingly-sophisticated Aerospace platforms will incorporate all the decision layers found in FTRA thus we are confident an integrated architecture such as what has been proposed in this work is broadly applicable.

In practice, a variety of algorithms could be implemented in the different FTRA layers provided they are capable of computing the designated output. Although we have investigated the FTRA architecture in the context of specific algorithms, its structure is generalizable to others. The problem of validation and verification (V&V) becomes ever more difficult as module and architecture complexity increases. Even when software implementing algorithms internal to each layer completes a V&V process, the new communications, including arbitration between potentially-disparate conclusions, introduces a new challenge. From a practical implementation standpoint, while the integration of deliberative AI and FTC algorithms enables capture and management of a more comprehensive fault set, increased complexity also increases risk of unanticipated execution sequences due to unpredicted interactions. A formidable but surmountable challenge is then to ensure the integrated system is validated.

As a multidisciplinary architecture, FTRA will require team-based adaptation to any domain, with composition of experts in both symbolic inference and adaptation (AI) and physics-based control systems. This may be a negative for small projects, although the authors argue any goal-based system with nontrivial dynamics requires participation from both communities today, just in segregated modules presumed to work with little to no knowledge of the other. From the Aerospace perspective, even if/when a FTRA-class implementation matures, the community must accept the increased level of autonomy offered for the FTRA to find its way into missions. NASA's now decade-old demonstration of Remote Agent (RA) illustrated both the power and challenge of onboard deliberation. Time delay and limited communication bandwidth (e.g., from deep space) will ultimately drive this migration. Only by building the algorithms and interfaces, including sound and tractable means of validation, will we be prepared to transition to a level of onboard autonomy that truly makes a ground station link optional even when faults occur.

As a start, this paper has mapped "AI deliberation and execution" processes to the Remote Agent architecture, motivated in part by its capabilities and in part by its previous use in a spacecraft. While this choice has been a convenient beginning to integration of AI deliberation and FTC, we also encourage and hope to conduct additional work to map other deliberation strategies into analogous integrated structures promoting synergistic task-level and motion-level fault management.

In our future work, we will more formally address arbitration between FDD and MI, particularly when independent conclusions are not consistent. This will likely require sharing additional statistical data as well as the list of conclusions each module has reached. Longer-term, we plan to build benchmark comparisons of FTRA performance with respect to RA alone, FTC alone, and ground-based supervision (user reconfiguration). Metrics will include fault diagnosis delay and correctness, reconfiguration delay, and post-reconfiguration performance. Our goal is to not only demonstrate increased capabilities but also to understand the tradeoffs and challenges associated with an integrated architecture of the class we propose in this paper.

VI. References

[1]Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach* 2nd Edition 2006**.**

[2]Muscettola, N., Nayak, P., et al "Remote Agent: To Boldly Go Where No AI System Has Gone Before". *Artificial Intelligence*, 103(1-2):5--48, 1998.

[3]Pell, Barney. et al, "An Autonomous Spacecraft Agent Prototype". *Autonomous Robots* 5, 29-52(1998) Kluwer Academic Publishers.

[4]Havelund, K., Lowry, M. et al "Formal Analysis of the Remote Agent Before and After Flight." In *Proceedings of the 5th NASA Langley Formal Methods Workshop, June 2000.*

[5]Blanke, M., Izadi-Zamanabadi, R., Bogh, S.A. and Lunau, C.P. "Fault-Tolerant Control Systems – A Holistic View ".*Control Engineering Practice* 5 (1997), no. 5, 693–702.

[6]Patton, J.R. "Fault-Tolerant Control Systems: The 1997 Situation". *IFAC Safeprocess'97 (Hull, United Kingdom), August 1997, pp. 1033–1055.*

[7]Zhang, Y., Jiang, J. "Bibliographical review on reconfigurable fault-tolerant control systems." *IFAC SAFEPROCESS*, 2003, pp 265-276.

[8]Muscettola, N., Smith, B., Chien, C., Fry, C., Rabideau G., Rajan K., Yan, D. "On-board planning for autonomous spacecraft". *in: Proc. 4th International Symposium on Artificial Intelligence, Robotics. And Automation for Space (i-SAIRAS)*, Tokyo, Japan, August 1997.

[9]Zweben, Monte and Mark S. Fox *Intelligent Scheduling* Chapter 6 Morgan Kaufman Publishers 1994

[10]Dean, T.L., McDermott, D.V. "Temporal data base management", *Artificial Intelligence 32 (1987) l-55.*

[11]Gat, E., "ESL: a language for supporting robust plan execution in embedded autonomous agents", *in: L. Pryor (Ed.), Proc. AAAI Fall Symposium on Plan Execution, AAAI Press, 1996.*

[12]Williams, C**.**B**.,** and Nayak, P.P. "A Model-Based Approach to Reactive Self-Configuring Systems". *In Proceedings of AAAI-96*, pages 971-978, Cambridge, Mass., 1996. AAAI, AAAI Press

[13]Zhou, D.H. and Frank, P.M. "Fault Diagnostics and Fault Tolerant Contro"*l. IEEE Transactions on Aerospace and Electronic Systems* Vol. 34, No. 2 April 1998.

[14]Zhang, Y.M. and Jiang, J. "Fault Tolerant Control System Design with Explicit Consideration of Performance Degradation*". IEEE Transactions on Aerospace and Elctronic Systems* 2003, 37 (3)

[15]Chamseddine, Abbas, Noura, Hassan and Ouladsine, M. "Sensor Fault Detection, Identification and Fault Tolerant Control: Application to Active Suspension". *1-4244-0210-7/06 2006 IEEE.*

[16] Zhang, Y.M. and Jiang, J. "Integrated Active Fault Tolerant Control Using IMM Aproach". *IEEE Transactions on Aerospace and Electronic Systems* Vol. 37, No. 4 October 2001.

[17]Wu, N.E., Zhang, Y.M. and Zhou, K. "Detection, estimation, and accommodation of loss of control effectiveness". *International Journal of Adaptive Control and Signal Processing* 2000; 14:775}795.

[18]Jiang J "Design of Reconfigurable control systems using Eigenstructure Assignment" *International J. Control*, 1994, 59(2): pages 395-410

[19]Wie, Bong *Space Vehicle Dynamics and Control"* AIAA Education Series 1998

[20]Chen, C.T. *Linear System Theory and Design* Harcourt Brace College Publishers 1984

[21]Ogata K. *Modern Control Engineering* 5th edition 2009

[22]Hermann, R. and Krener, A.J. "Nonlinear Controllability and Observability" *IEEE Transaction on Automatic Control*, Vol AC-22, No. 5, October 1977.

[23]Zhang, Y., Chen, X.Q., Chen, M. and Geng, Y.H. "A Novel Approach for Satellite Attitude Reconfigurable Fault-tolerant Control". *: Industrial Electronics and Applications, 2007. ICIEA 2007. 2nd IEEE Conference*, 23-25 May 2007 page(s): 2612-2616

[24]Smith, B., Feather, Martin S. and Muscettola, N. "Challenges and Methods in Testing the Remote Agent Planner". *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO*

[25]Pell, B. et al "A Hybrid Procedural/Deductive Executive for Autonomous Spacecraft". *Autonomous Agents and Multi Agent Systems*, 2, 7-22(1999) Kluwer Acamedic Publishers.

[26]Gamble, Edward B. et al "The Impact of Autonomy Technology on Spacecraft Software Architecture: A Case Study". IEEE Experts, USA, August 1998

[27] Zhang, Y.M. and Jiang, J. "Active Fault-Tolerant Control System Against Partial Actuator Failures". *IEE Proc.-Control Theory Appl.*, Vol. 149 No. 1, January 2002.

[28] Zhang, Y.M. and Jiang, J. "Design of Integrated Fault Detection, Diagnosis and Reconfigurable Control Systems". *Decision and Control, 1999. Proceedings of the 38th IEEE Conference* Volume: 4, On page(s): 3587-3592 vol.4, 1999

[29] Krishnan, H. McClamroch, N.H. and Reyhanoglu, M. "Attutude Stabilization of a Rigid Spacecraft Using Two Control Torques: A Nonlinear Control Approach Based on the Spacecraft Attitude Dynamics". *Automatica, Vol. 30, No. 12, December, 1994, 1885-1897.*

Acknowledgement

The authors are grateful to Professor N. H. McClamroch for reviewing this paper and providing useful comments and suggestions.

Appendix 1: Reconfigured Control[29]

This appendix presents an attitude stabilization method based on eight maneuvers using control torques along two principal axes only. The post-fault equations of motion are same as given by (1) and (2) except u_3 is now identically zero. When executed sequentially these eight maneuvers can transition a spacecraft from an arbitrary initial state to any desired final attitude. Note that each succeeding maneuver inherits initial conditions from the terminal conditions of the preceding maneuver.

Maneuver 1:

$$\begin{aligned} u_1 &= -k[sign(\Omega_1)]J_1 - \frac{1}{J_1}[(J_2 - J_3)\Omega_2\Omega_3] \\ u_2 &= -k[sign(\Omega_2)]J_2 - \frac{1}{J_2}[(J_3 - J_1)\Omega_1\Omega_3] \end{aligned} \tag{5}$$

Where k > 0 (in our simulations k = 0.01). Execute this maneuver until approximately zero angular velocity is reached along the controlled axis and an arbitrary angular velocity, say Ω_3^*, is achieved along the uncontrolled axis (in the above simulation, this took 16 sec).

Maneuver 2:

$$u_1 = -k\left[sign(\Omega_1 - \Omega_1^*)\right]J_1 - \frac{1}{J_1}\left[(J_2 - J_3)\Omega_2\Omega_3\right]$$
$$u_2 = -k\left[sign(\Omega_2 - \Omega_2^*)\right]J_2 - \frac{1}{J_2}\left[(J_3 - J_1)\Omega_1\Omega_3\right]$$
$$\Omega_1^* = \left(\frac{3k\left|\Omega_3^*\right|}{2\left|\frac{(J_1 - J_2)}{J_3}\right|}\right)^{1/3}, \Omega_2^* = -\Omega_1^*\left|\frac{(J_1 - J_2)}{J_3}\right|\left|\Omega_3^*\right| \tag{6}$$

Execute this maneuver until angular velocity $\mathbf{\Omega_3}^*/2$ is achieved along the uncontrolled axis (in the above simulation, this required 23.6 sec). Meanwhile, the other two velocities will reach a constant but potentially nonzero value.

Maneuver 3:

Execute maneuver 1 again until all three velocities become zero (In the above simulation, this took 24 sec).

Maneuver 4:

For this and subsequent maneuvers, we define function $Q_{z1,z2}$ as:

$$Q_{z1,z2} = \begin{cases} k \ldots if, \left\{z_1 + \frac{z_2|z_2|}{2k}\right\} > 0, or, \left\{z_1 + \frac{z_2|z_2|}{2k}\right\} = 0, and, z_2 > 0 \\ -k \ldots if, \left\{z_1 + \frac{z_2|z_2|}{2k}\right\} < 0, or, \left\{z_1 + \frac{z_2|z_2|}{2k}\right\} = 0, and, z_2 < 0 \\ 0 \ldots if, z_1 = 0, and, z_2 = 0 \end{cases} \tag{7}$$

We define the control law for this maneuver as:

$$u_1 = -(Q_{z1,z2})J_1 - \frac{1}{J_1}\left[(J_2 - J_3)\Omega_2\Omega_3\right]$$
$$z_1 = a\tan 2(2(q_4q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)), z_2 = \Omega_1$$
$$u_2 = -\frac{1}{J_2}\left[(J_3 - J_1)\Omega_1\Omega_3\right] \tag{8}$$

Execute this maneuver until the function given by z_1 above reaches zero (this corresponds to zero roll angle of the spacecraft if we convert the quaternion vector into Euler angles. In our case $\mathbf{\Omega}_1$ is the roll angular velocity). In above simulations, this maneuver took 35 sec.

Maneuver 5:

For this maneuver,

$$u_1 = -\frac{1}{J_1}\left[(J_2 - J_3)\Omega_2\Omega_3\right]$$
$$u_2 = -(Q_{x1,x2})J_2 - \frac{1}{J_2}\left[(J_3 - J_1)\Omega_1\Omega_3\right]$$
$$z_1 = \sin^{-1}(2(q_4q_2 - q_1q_3)), z_2 = \Omega_2 \tag{9}$$

Execute this maneuver until the function given by z_1 above reaches zero (this corresponds to zero pitch angle of the spacecraft if we convert the quaternion vector into Euler angles. In our case Ω_2 is the pitch angular velocity). In above simulations, this maneuver took 23 sec.

Maneuver 6:

For this maneuver,

$$u_1 = -(Q_{x1,x2})J_1 - \frac{1}{J_1}\left[(J_2 - J_3)\Omega_2\Omega_3\right]$$
$$z_1 = a\tan 2(2(q_4q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)) - \pi/2, z_2 = \Omega_1$$
$$u_2 = -\frac{1}{J_2}\left[(J_3 - J_1)\Omega_1\Omega_3\right] \tag{10}$$

Execute this maneuver until the function given by z_1 above reaches zero; this corresponds to roll angle of the spacecraft equal to pi/2, if we convert the quaternion vector into Euler angles. In our simulations, this maneuver took 32.5 sec.

Maneuver 7:

For this maneuver,

$$u_1 = -\frac{1}{J_1}\left[(J_2 - J_3)\Omega_2\Omega_3\right]$$
$$u_2 = -(Q_{x1,x2})J_2 - \frac{1}{J_2}\left[(J_3 - J_1)\Omega_1\Omega_3\right]$$
$$z_1 = a\tan 2(2(q_4q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)), z_2 = \Omega_2 \tag{11}$$

Execute this maneuver until the function given by z_1 above reaches zero (this corresponds to yaw angle of the spacecraft equal to zero, if we convert the quaternion vector into Euler angles. In above simulations, this maneuver took 56 sec (the same maneuver took 30 seconds for the rotation matrix based controller as can be noticed from figures 14 and 16).

Maneuver 8:

Finally, execute maneuver 4 until the desired final state, $\boldsymbol{x} = [\boldsymbol{q} \quad \boldsymbol{\Omega}] = [0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0]$, is reached. Note that, if we replace **q** by $\tilde{\boldsymbol{q}}$ in above control laws**,** where $\tilde{\boldsymbol{q}}$ is given by the relation below, our final attitude would be $\boldsymbol{x_d} = [\boldsymbol{q_d} \quad \boldsymbol{\Omega}] = [q_{d1} \quad q_{d2} \quad q_{d3} \quad q_{d4} \quad 0 \quad 0 \quad 0]$ whereas after 8 maneuvers we will have $\tilde{\boldsymbol{q}} = [\boldsymbol{0} \quad \boldsymbol{0} \quad \boldsymbol{0} \quad \boldsymbol{-1}]$**.** This means we can achieve any arbitrary attitude with two independent torques[29].

$$\begin{bmatrix} \tilde{q}_1 \\ \tilde{q}_2 \\ \tilde{q}_3 \\ \tilde{q}_4 \end{bmatrix} = \begin{bmatrix} q_{4d} & q_{3d} & -q_{2d} & -q_{1d} \\ -q_{3d} & q_{4d} & q_{1d} & -q_{2d} \\ q_{2d} & -q_{1d} & q_{4d} & -q_{3d} \\ q_{1d} & q_{2d} & q_{3d} & q_{4d} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

Appendix 2: FMEA Definitions[22]

Below is a summary of the mathematics used to calculate observability and controllability for our non-linear spacecraft attitude dynamics model. Consider two vector fields *f(x)* and *g(x)* in *n*-dimensional Euclidean space. The Lie Bracket operation generates a new vector field:

$$[f, g] \equiv \frac{\partial g}{\partial x} f - \frac{\partial f}{\partial x} g \tag{12}$$

Also, higher order Lie Brackets can be defined where *ad* represents the adjoint operator:

$$
\begin{aligned}
&(ad_f^1, g) \equiv [f, g] \\
&(ad_f^2, g) \equiv [f, [f, g]] \\
&\ldots \\
&(ad_f^k, g) \equiv [f, (ad_f^{k-1}, g)]
\end{aligned}
\tag{13}
$$

Definition 1:

The *n*-dimensional system defined by:

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x) u_i \quad (m \le n) \tag{14}$$

is locally accessible *about an arbitrary state* x_o if the accessibility distribution *C* spans *n* space, where *n* is the dimension of x and C is defined by:

$$C = [g_1, g_2, \ldots, g_m, [g_i, g_j], \ldots, [ad_{g_i}^k, g_j], \ldots, [f, g_i], \ldots [ad_f^k, g_i], \ldots] \tag{15}$$

The g_i terms are analogous to the **B** terms in linear dynamic system $\dot{x} = Ax + Bu$ and the $[f,g_i]$ terms correspond to the **AB** terms in the corresponding controllability matrix for linear systems. Nonzero $[g_i,g_j]$ terms arise from the nonlinear system properties. Note that if *f(x)* = 0 then *C* having rank *n* implies controllability.

Our spacecraft attitude dynamics are already in the form required by the above definition with:

$$
\begin{aligned}
g_1 &= [0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0]^T \\
g_2 &= [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]^T \\
g_3 &= [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]^T \\
f &= [f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7]^T
\end{aligned}
\tag{16}
$$

where

$$
\begin{aligned}
f_1 &= \frac{q_4 \Omega_1}{2} - \Omega_2 q_3 + \Omega_3 q_2 \\
f_2 &= \frac{q_4 \Omega_2}{2} - \Omega_3 q_1 + \Omega_1 q_3 \\
f_3 &= \frac{q_4 \Omega_3}{2} - \Omega_1 q_2 + \Omega_2 q_1 \\
f_4 &= -\frac{1}{2}[\Omega_1 q_1 + \Omega_2 q_2 + \Omega_3 q_3] \\
f_5 &= \frac{1}{J_1}[(J_2 - J_3)\Omega_2 \Omega_3] \\
f_6 &= \frac{1}{J_2}[(J_3 - J_1)\Omega_1 \Omega_3] \\
f_7 &= \frac{1}{J_3}[(J_1 - J_2)\Omega_2 \Omega_1]
\end{aligned}
\tag{17}
$$

In equation (10) J, Ω, q represent inertia, angular velocity and quaternion components.

For observability we first define a Lie Derivative. Notice that, any actuator failure will reduce control input vector rank *(m)* but the form of the system remains the same. Let *f* be an *n*-dimensional vector field, specifically the equations in (10) for our spacecraft, and let *h* be a smooth scalar function such that:

$$f : \Re^n \to \Re^n$$

$$h : \Re^n \to \Re$$

Then the Lie derivative of h with respect to f is:

$$L_f h = \nabla h . f = \frac{\partial h}{\partial x} . f = \sum_{i=1}^{n} \frac{\partial h}{\partial x_i} . f_i \tag{18}$$

where, *f* is given by

$$f = [f_1 \quad f_2 \quad f_3 \quad . \quad . \quad . \quad f_n]^T$$

and scalar h is h(x) where x is the n dimensional state vector).
Also note that the Lie derivative of h with respect to f is a scalar. We can also define higher order Lie derivatives:

$$L_f^k(h) = \frac{\partial}{\partial x}\left[L_f^{k-1}(h)\right] f \quad \text{where } k \geq 1 \qquad (19)$$
$$Note: L_f^0(h) = h$$

We assume a nonlinear system of the form:

$$\begin{aligned} \dot{x} &= f(x,u) \\ z &= h(x) = [h_1(x), h_2(x), ..., h_p(x)]^T \end{aligned} \quad \text{for } p \leq n \qquad (20)$$

Definition 2:

Let G denote the set of all finite linear combinations of the Lie derivatives of $h_1,\ldots,h_p$ with respect to f for various values of u = constant. Let ∇G denote the set of all their gradients. If we can find n linearly independent vectors within ∇G, then the system is locally observable. We can put the above theorem in mathematical form as follows. The system is locally observable at any state x_o if there exists some neighborhood of this state such that within that neighborhood:

$$\nabla G(x_o, u^*) \equiv \left.\frac{\partial l(x_o, u^*)}{\partial x}\right|_{x=x_o} \quad \text{has rank n, the dimension of x} \qquad (21)$$

where

$$l(x_o, u^*) \equiv \left[L_f^0(h_1) \quad \ldots \quad L_f^0(h_p) \quad \ldots \quad L_f^{n-1}(h_1) \quad \ldots \quad L_f^{n-1}(h_p)\right]^T$$

Note that, two states x_o and x_1 are *distinguishable* if there exists an input function u* such that:

$$z(x_o) \neq z(x_1).$$

In our case study p = 7 and,

$$\begin{aligned} h_1 &= q_1 \\ h_2 &= q_2 \\ &\ldots \\ h_7 &= \Omega_3 \end{aligned} \qquad (22)$$

Notice that the effect of sensor failure would be a reduction in p.