Copyright ©2018 by the International Astronautical Federation (IAF). All rights reserved.

IAC-18-C1.5.11x45016

On-Board Model-Based Fault Diagnosis for Autonomous Proximity Operations

Dr. Peter Z. Schulte[a]*, Dr. David A. Spencer[b]

[a] *School of Aerospace Engineering, Georgia Institute of Technology, 270 Ferst Drive, Atlanta, Georgia, United States of America, 30332*, pzschulte@gatech.edu
[b] *School of Aeronautics and Astronautics, Purdue University, 701 W. Stadium Avenue, West Lafayette, Indiana, United States of America, 47907*, dspencer@purdue.edu
* Corresponding Author

Abstract

Because of their complexity and the unforgiving environment in which they operate, aerospace vehicles often require autonomous systems to respond to mission-critical failures. Fault Detection, Isolation, and Recovery (FDIR) systems are used to detect, identify the source of, and recover from faults. Typically, FDIR systems use a rule-based paradigm for fault detection, where telemetry values are monitored against specific logical statements such as static upper and lower limits. The model-based paradigm allows more complex decision logic to be used for FDIR.

This study focuses on a state machine approach toward model-based FDIR. The state machine approach is increasingly utilized for FDIR of complex systems because it is intuitive, logic-based, and simple to interpret visually. In current practice, the detection of specific symptoms is directly mapped to the appropriate response for a pre-diagnosed fault, as determined by FDIR engineers at design time. This study advances the state-of-the-art in state machine fault protection by developing an on-board diagnostic system that will assess symptoms, isolate fault sources, and select corrective actions based on models of system behavior.

This state machine architecture for FDIR is applicable for a broad range of aerospace vehicles and mission scenarios. To demonstrate the broad applicability of the FDIR approach, two case studies are evaluated for scenarios in very different domains. The first is a terrestrial application involving the use of multi-rotor unmanned aerial vehicles (UAVs). The second is a space-based scenario involving autonomous proximity operations for orbital capture of a Mars Sample Return capsule. The efficacy of the state machine FDIR system is demonstrated via flight testing for the UAV case study and through software-in-the-loop testing in a flight-like simulation environment for the Mars Sample Return case. In each case, the FDIR system is focused on the Guidance, Navigation and Control subsystem.

This approach has been successfully shown to detect, diagnose, and respond to faults during testing. State machines allow the autonomous system to handle distinct faults with identical symptoms for initial detection. Each fault has a separate diagnosis and response procedure, and the proper procedure is selected by the state machine. This study demonstrates how a fault protection system may diagnose these faults on-board rather than relying upon a priori ground diagnosis.

Keywords: fault protection; state machines; guidance, navigation, and control; proximity operations; on-board diagnosis; fault detection, isolation, and recovery;

Acronyms/Abbreviations

APL	Applied Physics Laboratory
ESC	Electronic Speed Control
FOV	Field-of-view
FDIR	Fault Detection, Isolation, & Recovery
FSW	Flight Software
GN&C	Guidance, Navigation, & Control
JPL	Jet Propulsion Laboratory
KAUST	King Abdullah University of Science and Technology
KNN	K-nearest neighbors
LVLH	Local vertical local horizontal
MATLAB	Matrix Laboratory
MAV	Mars Ascent Vehicle
MSR	Mars Sample Return
NASA	National Aeronautics & Space Administration
OS	Orbiting Sample container
ROCS	Rendezvous OS Capture System
SRO	Sample Return Orbiter
UAV	Unmanned Aerial Vehicle
UML	Unified Modeling Language

1. Introduction

Aerospace vehicles are vulnerable to hardware and software faults that lead to mission-critical failures. Advances in on-board fault protection capability are necessary as both terrestrial and space vehicles increase in autonomy. In order to prevent failures, aerospace vehicles often employ Fault Detection, Isolation, and

Recovery (FDIR) or fault protection systems to sense, diagnose, and recover from faults. As more space missions travel to deep space destinations, autonomous operations will become more prevalent. There will be increased need for real-time prevention of failures through FDIR. These capabilities are especially vital for hazardous and time-critical activities such as rendezvous and proximity operations, which make extensive use of autonomous guidance, navigation, & control (GN&C). Deep space proximity operations applications require advanced autonomy and fault protection due to the significant round-trip light time from Earth.

The NASA Fault Management Handbook defines a failure as “the unacceptable performance of an intended function,” while a fault is defined as “a physical or logical cause, which explains a failure” [1]. Fault protection systems aim to perform the three-step process of fault detection, isolation, and recovery in order to prevent failures. Fault detection determines that something unexpected has occurred. Fault isolation (also connected to diagnosis) determines the possible source of a fault. Fault recovery is an action taken to attempt to retain or regain control of the system state and mitigate the impact of the fault. Typically, aerospace systems use a rule-based paradigm for FDIR where telemetry values are monitored against specific logical statements such as static upper and lower limits.

2. Background

This section presents background information on state machines and the current state-of-the-art in aerospace vehicle fault diagnosis.

2.1 State Machine Logic and Applications

Although the model-based paradigm for fault protection has been explored by industry, it has not yet been widely adopted. This study focuses on the state machine approach to model-based FDIR, which has been used in several flight projects and research studies because it is intuitive, logic-based, and simple to interpret visually. The “state” of a system includes any “aspects of the system that we care about for the purposes of control” [2]. Traditionally, state variables have included continuous physical parameters such as position, velocity, attitude, temperature, and pressure. However, state variables can also include discrete quantities such as operating modes, device health, and software filter convergence conditions. These discrete states can then be represented as state machines.

A state machine, or state chart, is a model-based tool that can be used to describe system behavior, including fault protection behavior [2]. Each block represents a specific state or sub-state of the system, and arrows between blocks represent transitions between states. A logical condition is associated with each transition, and if the condition associated with the transition becomes true, then the active state of the diagram will move from one state to another. State machines can be very simple, representing only a few possibilities, or they can involve complicated nested sets of states. State machine representations may be significantly simpler than the actual physical or software processes they represent, which is why they are considered models. However, a state machine for FDIR purposes can be developed in a way that represents all possible states relevant to mission success. FDIR systems expressed in terms of system state will be better able to protect the system in question [3]. Within MATLAB/Simulink, the Stateflow toolbox provides a simple graphical interface for developing state machines.

State machines offer several advantages over the rule-based FDIR paradigm. One significant advantage is the generation of a graphical product that is easier for designers, peer reviewers, and managers to understand and review. Other advantages include ease of accounting for subsystem interdependencies and implementing sequences with several decision points and/or path-dependent responses. The Johns Hopkins Applied Physics Laboratory (APL) conducted a formal trade study to determine whether their “ExecSpec” state-based fault protection system [4,5] or a more traditional rule-based system was more advantageous using the Solar Probe Plus mission as a case study [6]. They found that both methods were able to equivalently express all desired fault protection rules but that the state machine system is favored based on some of the advantages mentioned above. However, APL ultimately chose to continue using the rule-based system due to its extensive flight heritage. In addition to Stateflow and ExecSpec, another model-based software tool used for state machine design is MagicDraw, which uses the Unified Modeling Language (UML).

2.2 On-Board Model-Based Fault Diagnosis

In state-of-the-art fault protection practice, diagnosis is usually performed by FDIR engineers at design time, and the detection of a specific symptom is directly mapped to the appropriate response for the pre-diagnosed fault. This study develops an on-board diagnostic system that assesses symptoms, isolates fault sources, and selects corrective actions based on models of system behavior.

Though not typical for space missions, on-board fault diagnosis has been an area of research since the 1990s. Model-based fault diagnosis is considered a structured and mature field of research and many methods have been proposed and discussed in the control community using mathematical estimation methods for aeronautical vehicles [7,8]. Remote Agent was deployed as a technology demonstration (not as the primary control software) on the Deep Space 1 mission and featured model-based "mode identification" and "mode reconfiguration" for fault diagnosis, which identified components whose failures explained detected anomalies [9]. Cassini's Attitude Control Fault Protection is one of the few examples of a system where on-board fault diagnosis was performed in-flight [10,11]. One method for on-board diagnosis that has been used in research studies is called constraint suspension. It has been used to diagnose which component of a system is faulty [12,13].

3. Theory: State-Based Fault Protection Architecture

The FDIR architecture developed in this study collects data from the vehicle which is used to determine the likely fault state of the vehicle. This state can be classified as either "fault" or "no fault" based on how the decision logic is structured. The architecture isolates faults by performing diagnosis to determine their precise source and performs preventative actions to recover from faults before they become mission-critical failures. Outputs from the architecture can either send commands to the vehicle autonomously or notify ground operators to take corrective action.

3.1 Fault Protection Architecture Characteristics

This section focuses on three desired characteristics for the design of the architecture: generic, modular, and portable. The architecture itself is described in the following sections.

A generic architecture is applicable to any type of aerospace vehicle or mission. The FDIR architecture is comprised primarily of several generic diagrams that are described in the following sections. The MATLAB/Simulink simulation environment in which the architecture is developed allows setting vehicle parameters including physical dimensions and trajectory. It is applicable to a multitude of possible mission scenarios and permits alternate configurations, such as individual vehicles or multiple cooperative or non-cooperative vehicles. The simulation environment also contains generic modules for commonly used components such as sensors and actuators. The simulation environment has previously been adapted for use with many scenarios, missions, and vehicles, including the Prox-1 small satellite mission [14,15], various proximity operations scenarios with hardware such as a modular attitude determination system CubeSat avionics board and a Mars communication relay CubeSat constellation [16]. Each of the diagrams described in the following sections are implemented without focusing on any particular application or vehicle. Section 4 demonstrates how the architecture can be adapted for two distinct and very different applications. While the generic architecture presented here is focused particularly on FDIR for faults related to the GN&C subsystem, the same principles and design can be applied to any other faults and subsystems on an aerospace vehicle.

A modular architecture allows components to be easily added, removed, or rearranged. The visual block diagram environment offered by MATLAB/Simulink can be altered and reconfigured easily and allows for testing of many combinations of software modules and hardware components. For example, the investigator could replace the sensor/actuator suite and GN&C software modules. Also, various initial conditions, environmental scenarios, and physical vehicle properties can be easily redefined in a MATLAB initialization script and edited or rearranged in Simulink. These include spacecraft orbit and attitude dynamics, spacecraft properties such as mass and moment of inertia, relative dynamics for multiple spacecraft, sensor and actuator properties such as field of view and resolution, GN&C software components, and central body or environment properties. Parameters for FDIR algorithms can also be adjusted, such as fault injection times, wait times, and trigger thresholds. The diagrams described in the following sections can also be easily adjusted and rearranged to adapt them for various vehicles and missions.

A portable architecture allows straightforward conversion of its design implementation for a particular mission to code that is used onboard the vehicle. The FDIR architecture allows rapid transition from development to flight. The computational requirements of the FDIR architecture match the capability generally available on flight processors. The architecture has the ability to make the kinds of complex decisions normally required for autonomous flight software (FSW) and is evaluated by testing its response to realistic conditions rather than "canned" scenarios. It is well-integrated with other hardware and software components, allowing new components to be quickly evaluated. Finally, the architecture features the capability to easily convert its logic into FSW code via autocoding, a process which has been used with the Prox-1 mission as described in [14]. In this process, algorithms developed in MATLAB/Simulink are converted to C code and integrated with other FSW code in C. Autocode performance is validated via a "day-in-the-life" test on flight hardware. Although the autocoding process is not demonstrated directly in this study, technical memos written by the Prox-1 team are included in Appendix A of [17] to provide guidance for future researchers or engineers desiring to reproduce it.

3.2 Generic Functional State Machine

A functional state machine is a model that describes the behavior of a system by tracking the "mode state" of the system [18]. Mode states are high-level descriptions of the overall system behavior and are distinct from dynamic states (such as position & velocity) or vehicle component states (such as battery level or processor temperature). Each vehicle and mission will have a distinct state machine describing how these modes change and the logical conditions to switch between them. The generic functional state machine shown in Fig. 1 provides a template for constructing this diagram. It features generic modes that may be present in many different contexts. The initial state in the bottom left is "Standby," which is a passively safe mode where the vehicle waits for further commands to proceed.

If no faults have been detected (`FaultDetectedMode=0`) then a command (`BeginComplexProcess=1`) allows a "complex process" to begin. Complex processes could include either autonomous or piloted operations. An optional transition phase occurs before the complex process state begins. The complex process has several sub-states. First is "Standoff" (`ArriveAt Standoff=1`), which is a phase where the complex process is "armed" but not initiated and the vehicle is awaiting permission to proceed. Standoff is distinct from Standby because the vehicle may *not* necessarily be in a passively safe dynamic state during Standoff. If no faults are detected (`FaultDetectedMode=0`) the complex process begins when a command is provided (`ReadyToGo=1`). At this point the NominalZone state begins. This is a nominal region where faults are acceptable and can generally be detected and responded to safely while still continuing nominal operations.

At some point, based on the dynamic state of the system, safe operation under fault conditions may no longer be possible (`EnterAbortZone=1`). When this occurs, the AbortZone state begins, and at any time if a fault detection is triggered or a human operator decides conditions are unsafe, an Abort can be commanded (`Abort=1`). The abort stops the complex process and moves the vehicle to a safe dynamic state, eventually returning to the Standby state (`ArriveAtStandby =1`). Additionally, an "Interact" state allows the vehicle to interact with other vehicles, target objects, or the environment. A pre-interaction region called the InteractZone is entered from the AbortZone when (`EnterInteractZone=1`). The Interact state can be entered from either AbortZone or InteractZone when a command is received (`BeginInteraction=1`). The vehicle *cannot* enter the Interact state directly from NominalZone because interaction almost always involves hazardous conditions. If a fault or other hazard occurs during Interact, an abort can be triggered (`Abort=1`). If no anomalies occur, the vehicle will return to passively safe standby after the interaction is complete (`ArriveAtStandby=1`).

3.3 Generic Diagnostic State Machine

To implement on-board model-based fault diagnosis, the generic diagnostic state machine shown in Fig. 2 has been developed. The diagnostic state machine consists

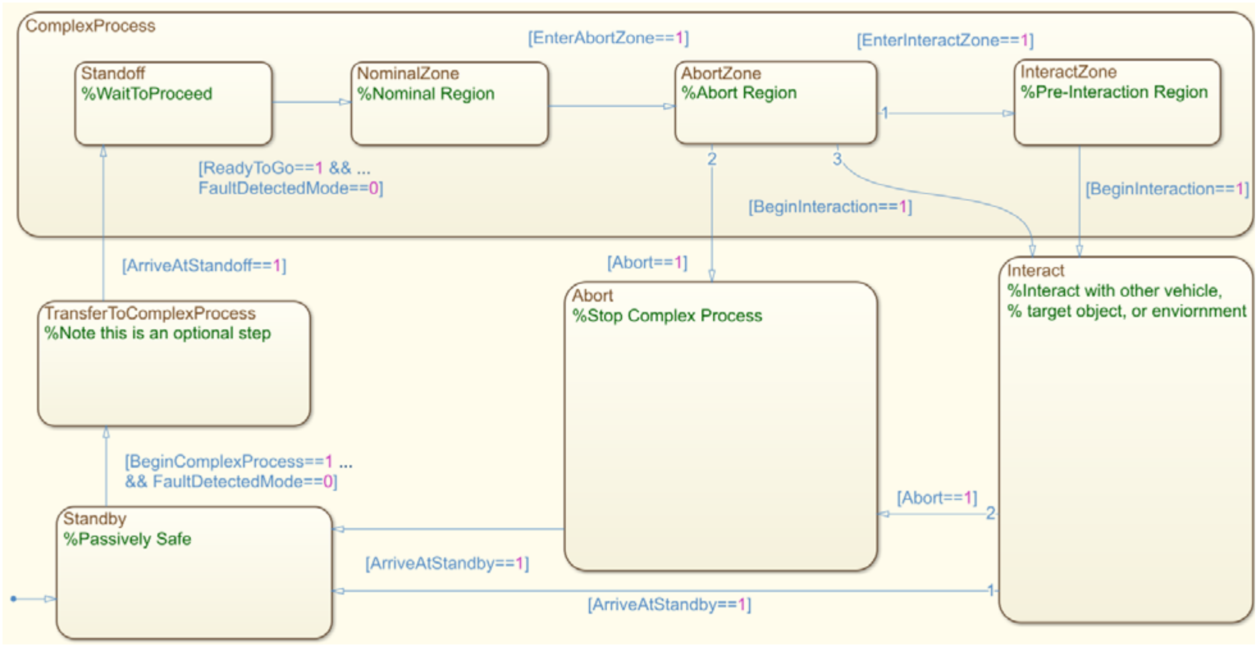


Fig. 1. Generic functional state machine

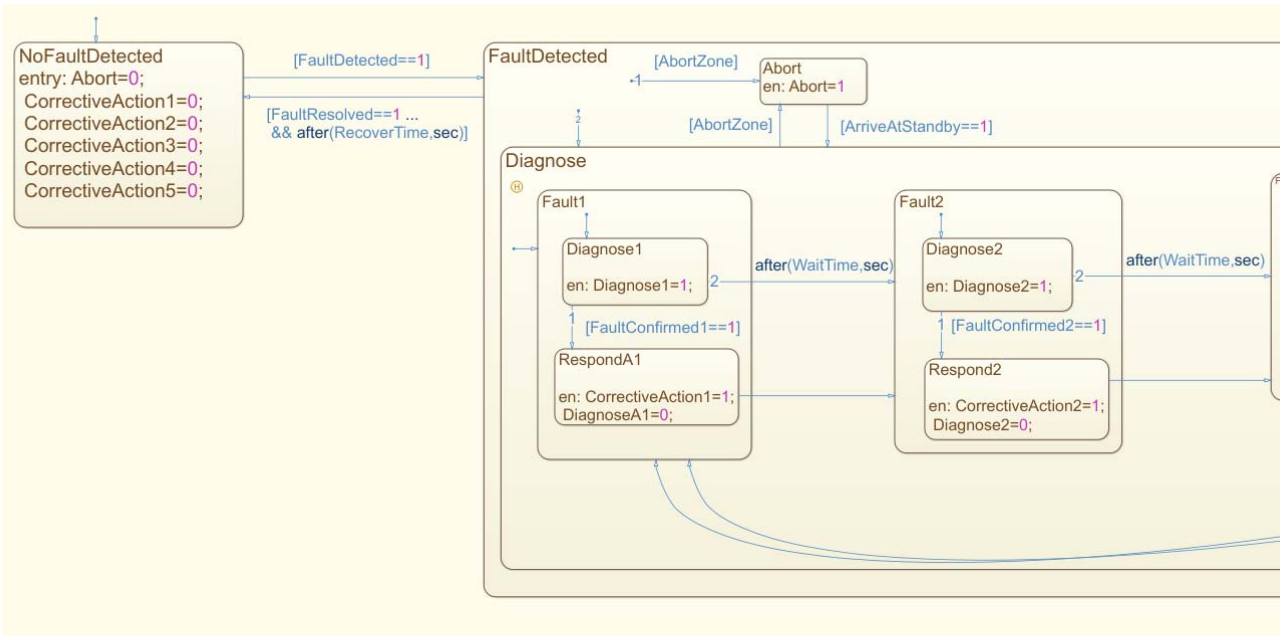


Fig. 2. Generic diagnostic state machine (only a portion of this diagram is shown for readability)

of two primary states: NoFaultDetected and Fault Detected. During all nominal mission phases, NoFaultDetected is activated, but when a fault detection trigger is observed (`FaultDetected=1`), the Fault Detected state is activated. If the functional state machine is in any state other than AbortZone, then the diagnostic state machine enters "Diagnose" immediately when a fault is detected. If the functional state machine is in the Abort Region (`AbortZone=1`) when a fault is detected, the diagnostic state machine does not attempt to determine which fault has occurred. An abort maneuver is commanded immediately, returning the vehicle to a passively safe dynamic condition before entering the Diagnose state.

The Diagnose state consists of sub-states for each possible fault. Each sub-state begins by running a diagnostic routine to determine if that particular fault has occurred. If the diagnostic routine returns `Fault Confirmed=1`, then the appropriate fault response routine is called and the diagnostic sub-state for the next fault is activated while the response runs in the background. If the diagnosis does not result in fault confirmation within a user-defined wait time, then the active sub-state moves to the next possible fault and the process repeats. Once all possible faults have been evaluated, the active sub-state returns to the first fault until the fault has been resolved by one of the corrective actions.

Note that fault *diagnostic* checks are distinct from fault *detection* checks. None of the diagnostic checks are performed unless they are called by the diagnostic state machine, which is only activated once the fault detection triggers are activated. Thus, a fault will *not* be detected if one of the fault *diagnosis* conditions is met but the fault *detection* conditions have not been met. Once a fault has been diagnosed, the diagnostic state machine calls the appropriate fault response routine. When the fault is resolved and a user-specified recovery time has passed, the active state returns to NoFaultDetected.

3.4 Integration in MATLAB/Simulink

The functional state machine and diagnostic state machine described in the previous two sections are designed to work together in the fault protection architecture along with several additional components in the MATLAB/Simulink environment. The fault & mode portions of the architecture are the main focus of this study, and these components are shown in the example Simulink diagram in Fig. 3, which illustrates how each of the components interacts with the others.

The two primary components are the functional state machine and the diagnostic state machine. These are Stateflow blocks which have been described in the previous two sections. Most of the inputs to the functional state machine are produced by the generic mode management block, a MATLAB function which takes in vehicle state information and ground commands and calculates the logical variables that are evaluated in functional state machine transitions. `AbortZone` is output from the functional state machine to the diagnostic state machine and describes whether the AbortZone state is active. `FaultDetectedMode` and `Abort` are generated by the diagnostic state machine.

The inputs to the diagnostic state machine come from several sources. `ArriveAtStandby` is

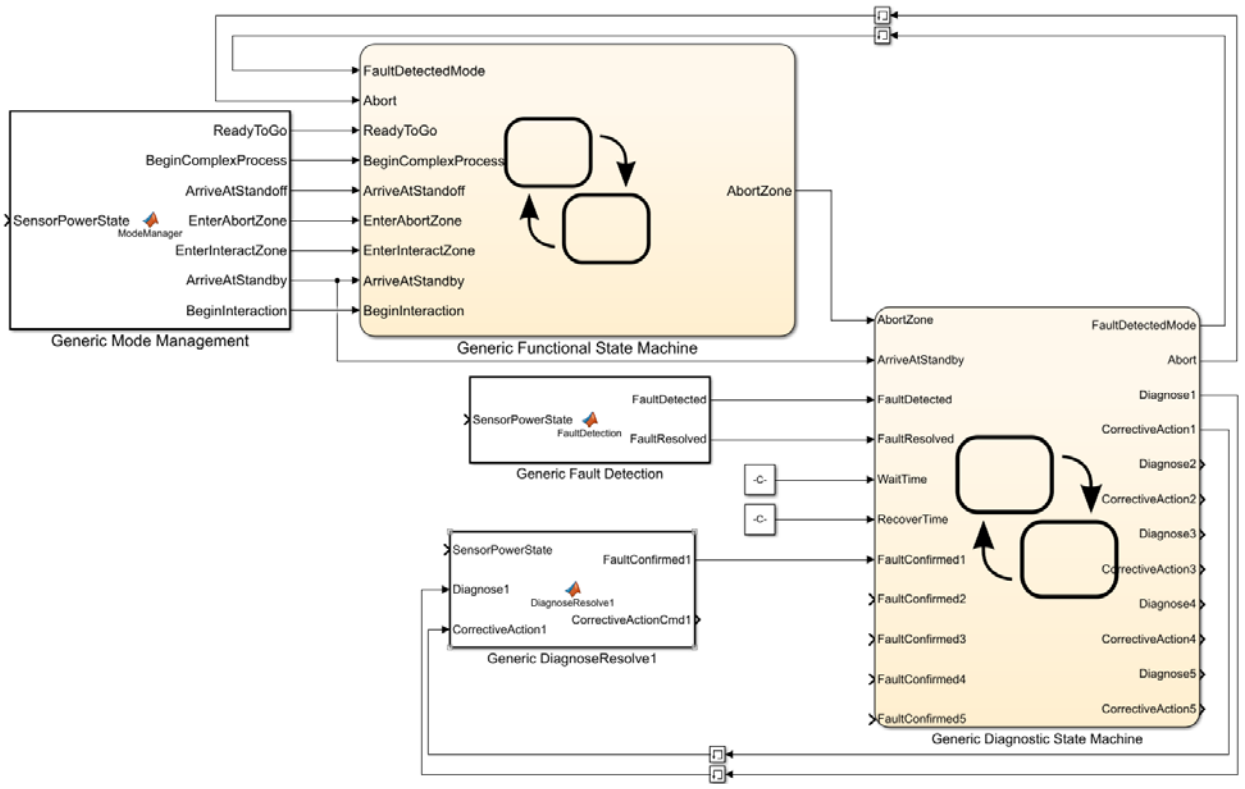


Fig. 3. Generic diagnostic state machine (only a portion of this diagram is shown for readability)

generated by the mode management MATLAB function block. `AbortZone` is generated by the functional state machine. Fault detection checks are performed by a MATLAB function block and result in `Fault Detected` and `FaultResolved`. Two variables are input as constants (`WaitTime` and `RecoverTime`), and a set of variables indicating fault confirmation (`FaultConfirmed1,2,3,` etc.) are input from the fault diagnosis/resolution MATLAB function block for each fault. Two output commands for each fault (`Diagnose1,2,3,` etc. and `Corrective Action1,2,3,` etc.) are fed into the respective fault diagnosis/resolution function blocks. Note that only one diagnosis/resolution function is shown for clarity but most systems will consider more than one fault and will have a diagnosis/resolution function for each fault

4. Results: Evaluation in Simulation

Each of the generic architecture diagrams described in the preceding section can be adapted for particular applications. This section provides examples for two very different scenarios. It is important to note that the generic diagrams usually provide more or less detail than necessary, depending on the application. Detail can be added or removed in each diagram as needed.

4.1 UAV Nervous System Example

One application of the state machine FDIR architecture has been developed for a multirotor UAV system. This "UAV Nervous System," serves as a proof-of-concept of the state machine FDIR architecture and has been developed in collaboration with FalconViz, a startup company based at the King Abdullah University of Science and Technology (KAUST). FalconViz uses multi-rotor and fixed-wing UAVs for scanning and 3D mapping, among other applications. The FalconViz team recognized a need for fault protection because small problems with their UAV hardware or software would often cause mission critical failures.

The primary goal of the UAV nervous system is to detect, diagnose, and respond to excess vibration in flight. An accelerometer is placed on the arm of the copter below the propeller and electrical tape is added to the propeller to simulate unbalance. The K-nearest neighbors (KNN) supervised machine learning algorithm is trained and used for fault detection [19]. The output of the KNN algorithm is sent into the diagnostic state machine for fault diagnosis and confirmation. When the fault has been diagnosed as "confirmed," an audio signal is sent to the pilot via the radio controller. The pilot can then land the copter to investigate the fault. The complete system involves a suite of sensors on two arms of the copter. In addition to an accelerometer on the second arm, temperature sensors are installed at the base of the motors and current/voltage sensors are installed between the lithium polymer battery and the Electronic Speed Controls (ESCs). The nervous system is able to detect and respond to faults from all eight sensors simultaneously. For all faults, the response is to send a signal to the pilot to land. Several flight tests demonstrating successful detection of faults have been completed. The results of one set of flight tests were published at the 67th International Astronautical Congress [19].

After the generic FDIR architecture described in Section 3 was created, the UAV Nervous System case study was revisited and adjusted to match the generic architecture. First, the generic functional state machine was adapted for the FalconViz UAV test flight, as shown in Fig. 4. Many of the states from the generic diagram were unnecessary because of the relative simplicity of the UAV test flight. The Standby state at the bottom is the starting state and represents the copter sitting stationary on the lab bench at the beginning of the test flight when the system is activated and data recording begins. When the copter is being carried outside (`CarryingCopter=1`), the Transfer state begins. The Transfer state ends when the copter is set down on the ground outside (`ArriveAt Standoff=1`), which begins the Piloted phase. During Standoff, the first sub-state of the Piloted phase, the copter is sitting on the ground, waiting for the pilot's command to proceed. When the pilot begins throttling up the motors to launch the copter (`Liftoff=1`), the NominalZone sub-state begins, indicating that the copter is flying. When the copter lands and the motors are powered down (`Landed=1`), the active state returns to Standoff. Note that no additional abort states are included in this functional state machine because the standard abort procedure when a fault is detected is for the pilot to land the copter.

Next, the generic diagnostic state machine was adapted for the UAV Nervous System as shown in Fig. 5. In this case, the trigger for fault detection (`FaultDetected=1`) is set to the output of the machine learning algorithm for vibration detection. This is done without regard to persistence, so whenever the machine learning algorithm indicates a fault detection, the FaultDetected state and Diagnose sub-state become active. Because only one fault could cause this particular detection, only one fault diagnosis sub-state is present in the diagnostic state machine. The "Propeller Unbalanced" fault is diagnosed if the fault detection trigger remains active for the "Time to Detect" of 0.025 sec, *and* if the current state of the functional state machine is NominalZone (indicating the copter is flying). If the fault detection is only intermittent, fault diagnosis will be inconclusive. In either case, when the fault detection flag from the machine learning algorithm is set to zero for the "Time to Resolve" length of 0.03 sec, the state machine returns its active state to NoFaultDetected.

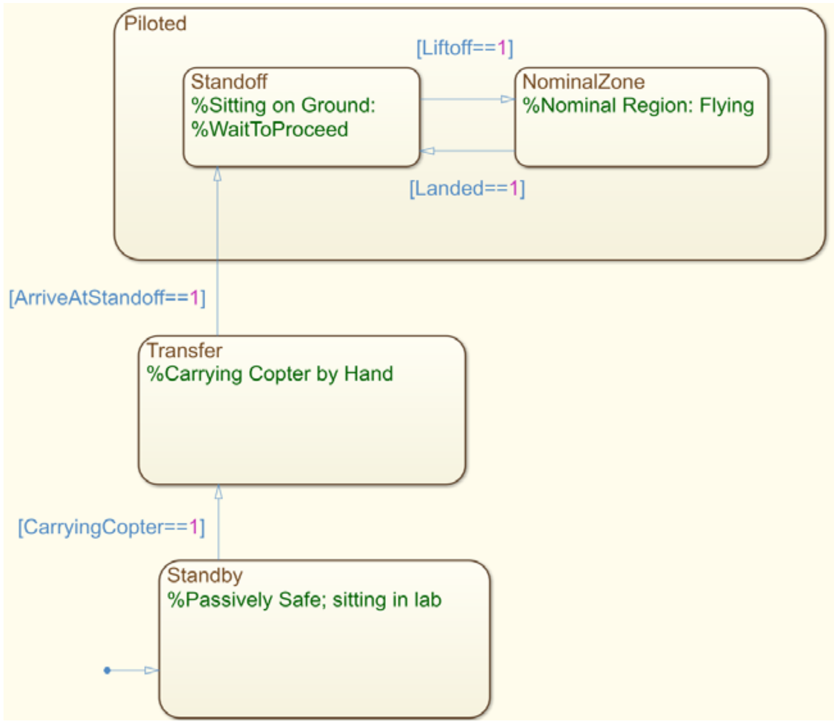


Fig. 4 Functional state machine for FalconViz UAV

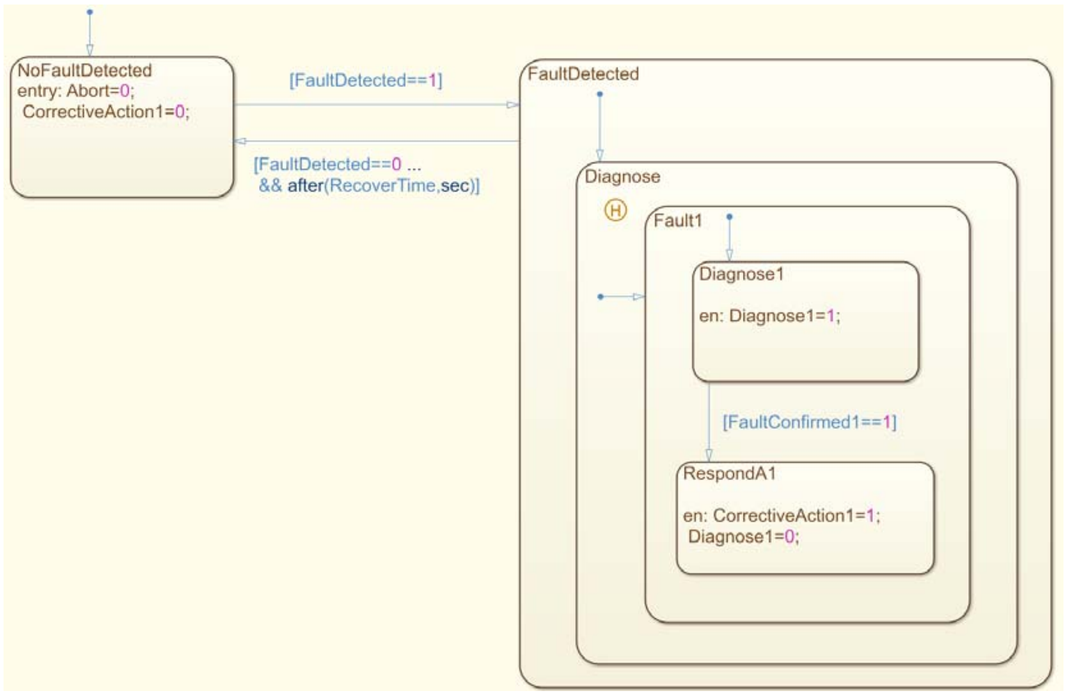


Fig. 5 Diagnostic state machine for UAV

To demonstrate the FDIR architecture, recorded flight test data was loaded from a data file and replayed in Simulink. The functional and diagnostic state machines described above were added to a Simulink model, and MATLAB functions were written to calculate mode management inputs to the functional state machine and fault diagnostic inputs to the diagnostic state machine. The recorded flight test data and FDIR output are shown in Fig. 6. Note that this flight test data is similar but not exactly the same as the data shown in [29]; the flights were performed on different dates and at different stages of development of the UAV Nervous System. The top three plots show accelerometer data in three axes and the bottom plot shows the `FaultConfirmed1` signal output by the diagnostic function.

The data begins with the copter on the lab bench in segment A, and tape is placed on the propeller to unbalance it. The copter is carried outside from the lab during segment B. The copter takes off and flies with an unbalanced propeller in segment C. The FDIR architecture quickly detects the imbalance and outputs a `FaultConfirmed` status of 1 at around 3 sec, shortly after segment C begins. At the end of segment C the copter lands, and the FDIR architecture immediately resets the `FaultConfirmed` status to 0. During segment D the copter is on the ground, and the tape is removed to restore the propeller balance. Segment E shows balanced flight, and the copter lands again at the end of segment E. During segment F, tape is added again while the copter is on the ground. Unbalanced flight resumes during segment G, and between 11 and 12 sec the FDIR architecture quickly detects the imbalance and outputs a `FaultConfirmed` status of 1. The copter lands at the end of segment G, and the FDIR architecture immediately resets the `Fault Confirmed` status to 0.

The UAV Nervous System has been developed and tested for a terrestrial rotary wing UAV. The system proof-of-concept has been shown through flight testing. The generic FDIR architecture has been successfully adapted for use with the UAV Nervous System and has been demonstrated in MATLAB/Simulink using recorded flight test data. Since the new system is operating on similar flight data to the previous system, improved performance is attributable to the addition of a state machine monitoring the state of the UAV from telemetry. For example, by monitoring ESC current measurements, the updated architecture is able to determine whether the UAV is flying and takes this into account when diagnosing whether a fault is present.

Copyright ©2018 by the International Astronautical Federation (IAF). All rights reserved.

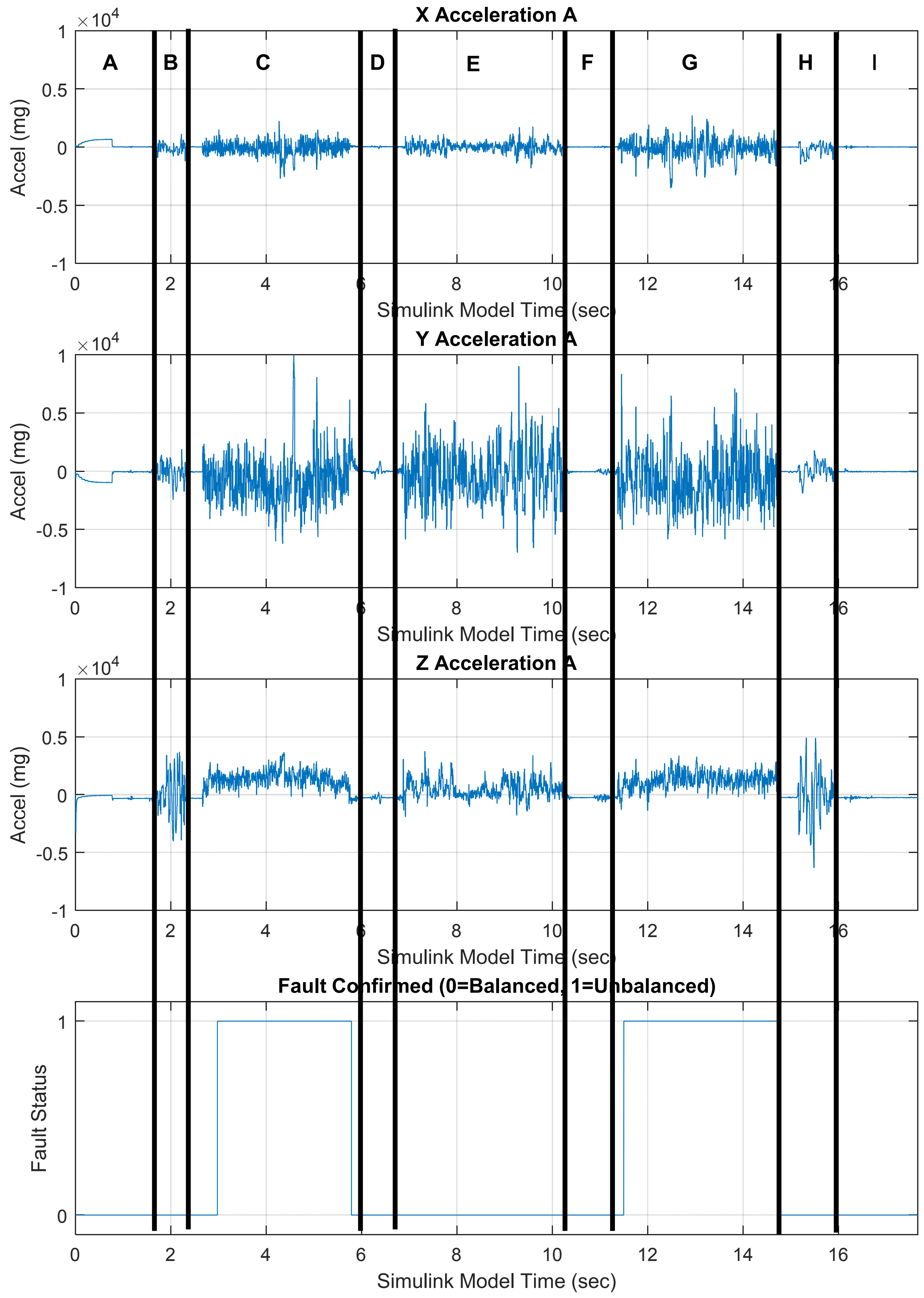


Fig. 6 Results of test flight replay with UAV Nervous System FDIR Architecture

4.2 Mars Sample Return Rendezvous Example

A second use of the state machine FDIR architecture has been developed for an automated relative proximity operations application. This work supports development of a Mars Sample Return (MSR) mission. The process for development of fault protection requirements was published at the 68th International Astronautical Congress [20] and a detailed concept of operations, trajectory control strategy, and complete simulation results have been submitted in an article to the Journal of Spacecraft and Rockets [21].

One key feature of the Mars Sample Return concepts currently under consideration is that they require autonomous rendezvous and capture. Samples collected by the Mars 2020 rover will be placed into an Orbiting Sample container (OS), launched into orbit around Mars, and intercepted by a Sample Return Orbiter (SRO). The SRO performs ground-directed rendezvous until it is about 100 meters away from the OS. Finally, terminal rendezvous and capture of the OS are performed autonomously.

A nominal approach trajectory is shown in Fig. 7 in the Local Vertical Local Horizontal (LVLH) frame and involves the following phases. First, out-of-plane natural motion occurs in the passively safe standby trajectory before any control is activated; this is the blue portion of the trajectory. Once trajectory control is activated (at the start of the black portion of the trajectory), the controller allows the SRO to continue in natural motion. When the *xy*-plane is reached, a planar hop maneuver is commanded to remove all out-of-plane motion and the red portion of the trajectory begins. The controller allows the SRO to continue coasting until the along-track axis is reached and a hold position maneuver is commanded to hold the SRO at a fixed relative position. A small maneuver is commanded to begin the v-bar approach (light blue portion of the trajectory), and subsequent hops are performed until the SRO is near to the OS. In the final v-bar hop, the green portion of the trajectory begins and the controller allows the SRO to coast until it reaches the point of closest approach (the red x) at a range of 1.08 m, where another hold position maneuver is performed to represent OS capture.

During the v-bar approach, three "zones of criticality" are defined to alter fault protection behavior based on distance to the target, as shown in Fig. 7. Note that durations and distances shown here are dependent on the rendezvous approach strategy, so the transition conditions between these zones may change, but the

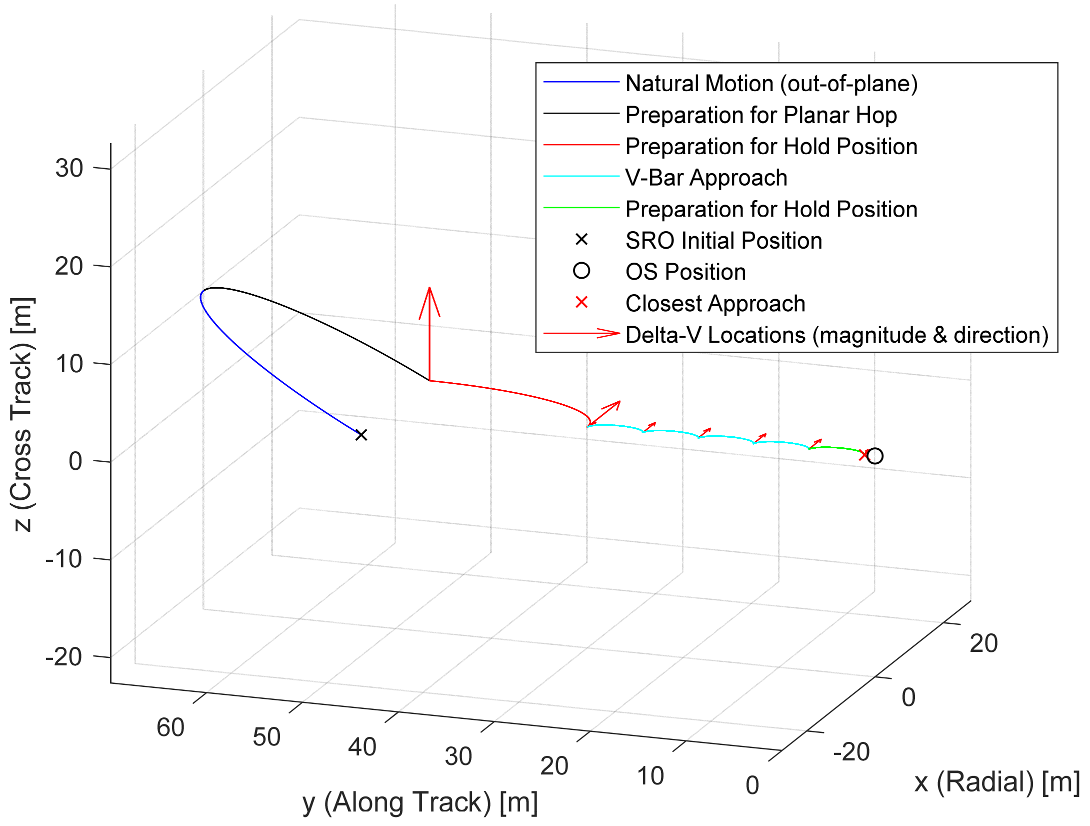
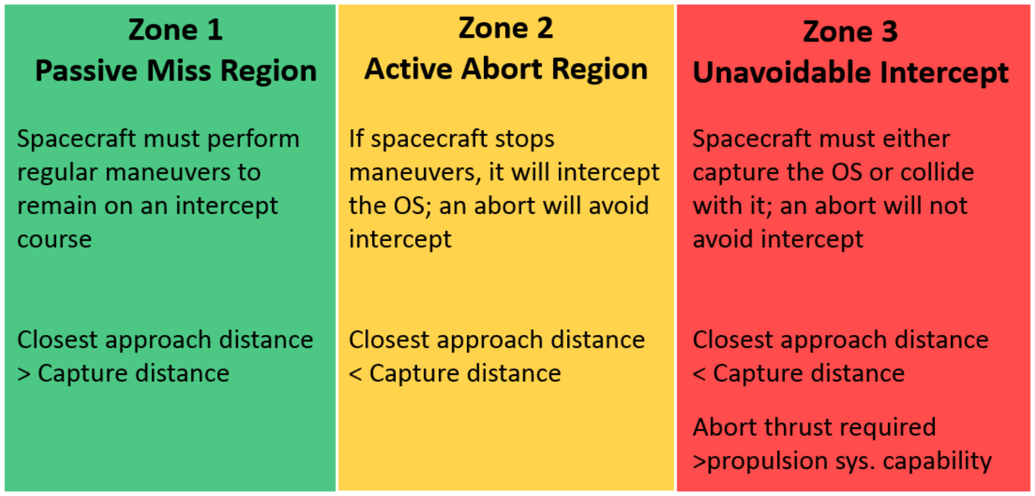


Fig. 7 Relative orbit three dimensional view (LVLH) for nominal trajectory



Zone 1 Passive Miss Region	Zone 2 Active Abort Region	Zone 3 Unavoidable Intercept
Spacecraft must perform regular maneuvers to remain on an intercept course	If spacecraft stops maneuvers, it will intercept the OS; an abort will avoid intercept	Spacecraft must either capture the OS or collide with it; an abort will not avoid intercept
Closest approach distance > Capture distance	Closest approach distance < Capture distance	Closest approach distance < Capture distance
		Abort thrust required >propulsion sys. capability

Fig. 8 Notional "zones of criticality"

criticality (and thus impact on FDIR behavior) of the zones will endure regardless of the implementation selected. In order to capture how this behavior fits into the overall terminal approach, a functional state machine was created, shown in Fig. 9. This model represents both nominal and off-nominal processes for rendezvous and capture and is referenced by the diagnostic state machine.

At the end of ground-in-the-loop rendezvous, the system begins in a passively safe trajectory that will not impact the OS even if it drifts. A ground command initiates the autonomous sequence, and a Final Hop moves the spacecraft from the passively safe trajectory to the v-bar approach plane. The Final Hop ends at a "standoff" position (no longer passively safe) at the start of the v-bar approach. When proper conditions are achieved, the "closed-loop" v-bar approach begins.

The system then enters the "Passive Miss Region," which requires the SRO to perform regular maneuvers in order to remain on an intercept course. If a fault is detected at any point in this region, the SRO stops maneuvers and enters Passive Abort, passing by the OS harmlessly and returning to Passive Standby. If no faults occur, the system enters the "Active Abort Region" when the dynamic boundary is crossed. This zone ends in an intercept unless an Abort maneuver is commanded to return to Passive Standby via the Active Abort mode. The final zone, called the "Unavoidable Intercept Region" occurs at the very end of the rendezvous sequence, when the SRO can no longer avoid an intercept; it must either capture the OS or collide with it.

If capture is unsuccessful and the OS does not enter the capture volume, the system enters the LocateOS state. It attempts to determine where the OS is located

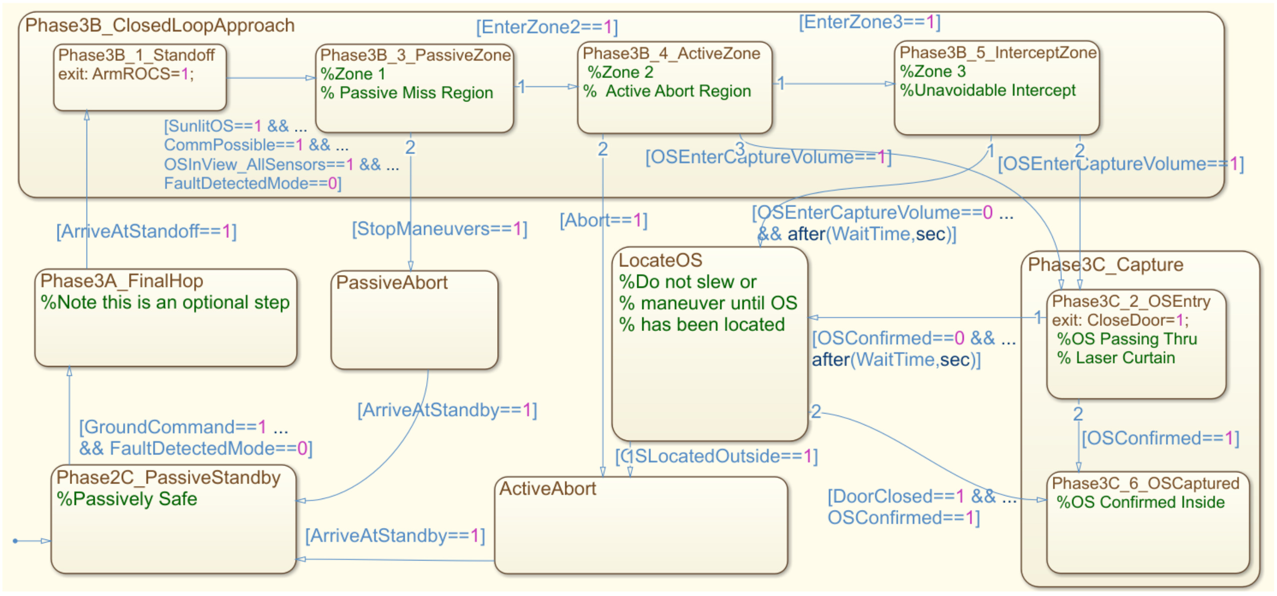


Fig. 9 State machine for terminal rendezvous and capture process

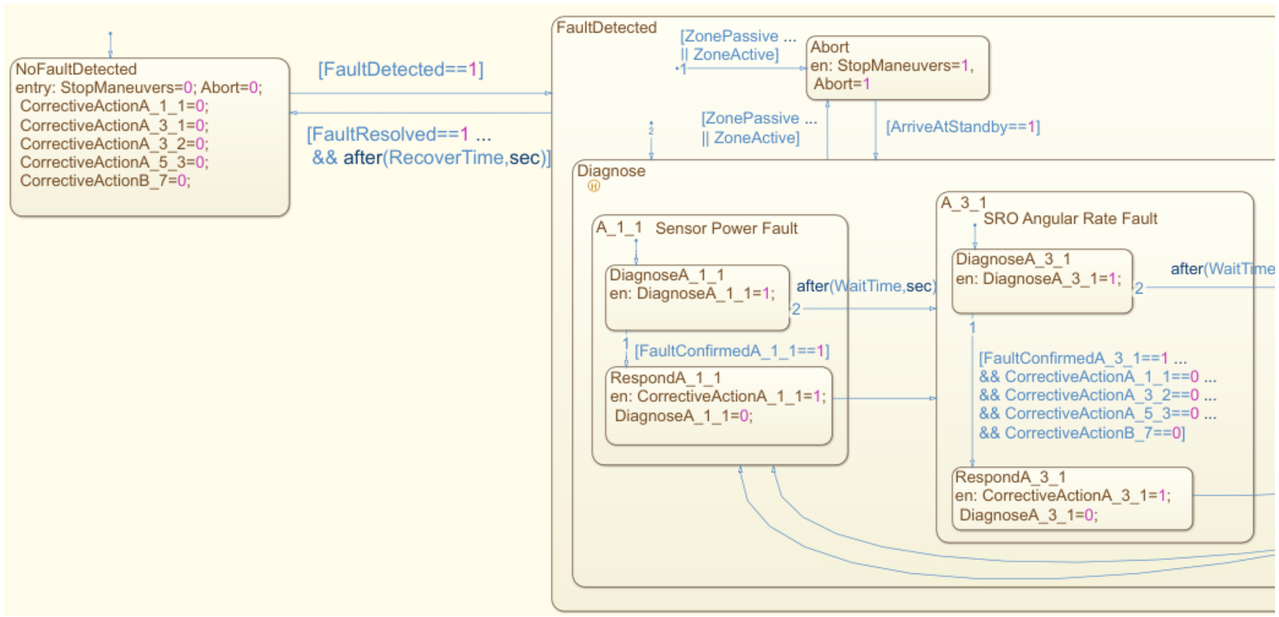


Fig. 10 Diagnostic state machine for MSR fault protection architecture
(Only a portion of this diagram is shown for readability)

before performing any slew or thrust maneuvers. Once the OS is found, an abort maneuver is commanded. If the OS enters the capture volume successfully, the capture process begins. The OS passes by a sensor such as a laser curtain and the door is closed. If the OS cannot be confirmed inside the capture volume after the door has closed, the system also enters the FindOS state and commands an Abort unless the OS is found inside.

Four faults were selected for simulation based on the scenario where the OS is no longer visible in the imager field of view (FOV), but further information is necessary to determine which fault occurred. This scenario provides a suitable case study to demonstrate the diagnostic capability of the FDIR architecture. The diagnostic state machine shown in Fig. 11 is used to perform on-board model-based fault diagnosis. It behaves exactly the same as the generic diagnostic state machine, except that an abort is commanded upon fault detection if the SRO is in either the Passive Miss Region or the Active Abort Region. In all other regions, the Diagnose state calls diagnostic and response functions for each candidate fault. Only two diagnose sub-states are shown for clarity, but there is one sub-state for each possible fault.

Six simulation cases have been evaluated to demonstrate the capabilities of the FDIR architecture for Mars Sample Return autonomous rendezvous and capture [21]. A summary of three representative scenarios is presented here. In the first case, shown in Fig. 11, an angular rate fault results in the loss of the OS from the imager FOV; a fault is injected at 1,050 sec during the planar hop by turning off the attitude tracking

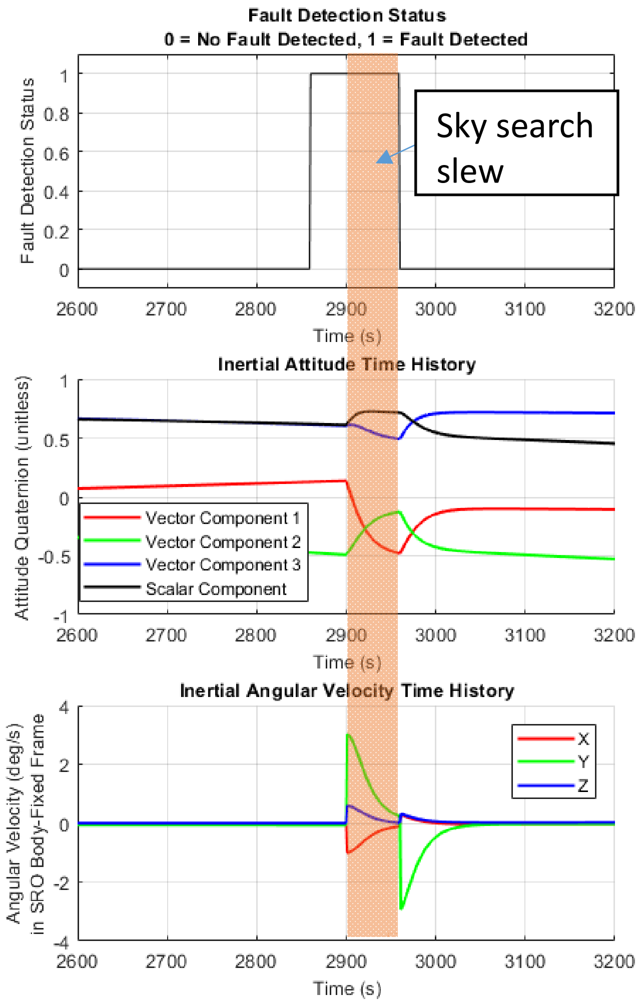


Fig. 11 Simulation results for angular rate fault recovery

controller. The OS slowly drifts out of the imager FOV until it is no longer visible, triggering a fault detection.

The fault protection system then initiates a sky search slew, which scans the sky and quickly finds and tracks the OS again. After reacquiring the OS in the imager the SRO continues to capture at 20,534 sec and a minimum range of 1 m. Similar responses have been demonstrated for cases with the OS in eclipse and an unconverged relative orbit filter prior to the beginning of the v-bar approach.

In the next case, shown in Fig. 12, a fault is injected at 6,000 sec indicating that the relative orbit determination filter is unconverged. Unlike the previous scenario, in this scenario the planar hop has already been completed and the v-bar approach has begun *before* the fault is injected. The fault protection system detects this fault and immediately commands a passive abort because the SRO is in the Passive Miss Region (Zone 1) of the v-bar approach. The SRO then stops maneuvers and begins drifting; it passes through a minimum range of 24.23 m at 8,706 sec (about 30 minutes after the fault time). After this minimum range, the SRO drifts away from the OS in the negative along-track direction. Once the along-track distance reaches 50 m, the SRO injects cross-track motion and returns to a passively safe standby.

In the final case, shown in Fig. 13, a fault is injected at 19,000 sec at a range of 4.32 m, indicating that the camera has lost power. The FDIR system detects this fault and immediately commands an active abort because the SRO is in the Active Abort Region (Zone 2) of the v-bar approach. The SRO then injects out-of-plane motion and moves away from the OS. After entering an out-of-plane ellipse, the SRO drifts away from the OS in the negative along-track direction. Once the along-track distance reaches 50 m, the SRO freezes the drift and returns to a passively safe standby.

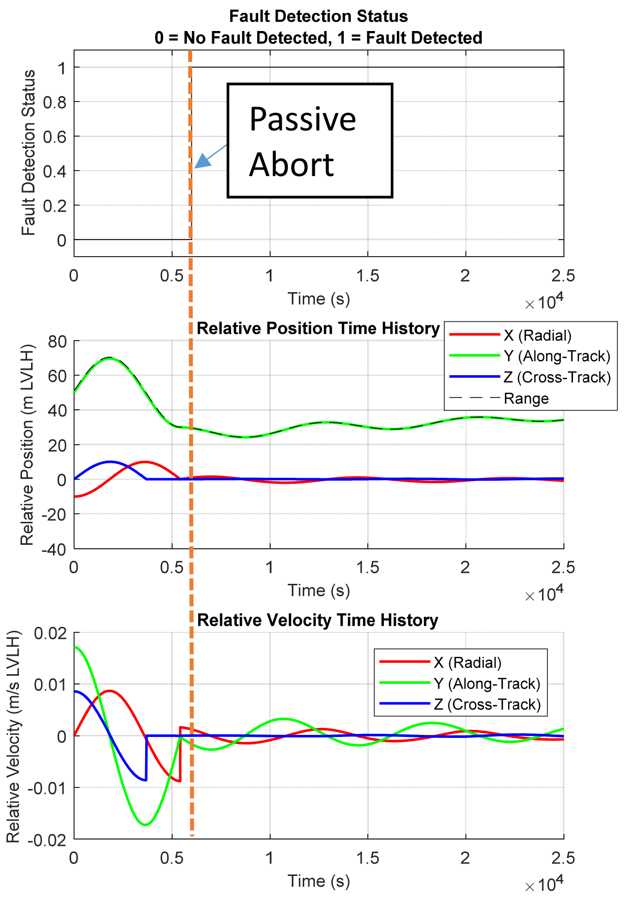


Fig. 12 Simulation results for unconverged relative orbit filter resulting in passive abort

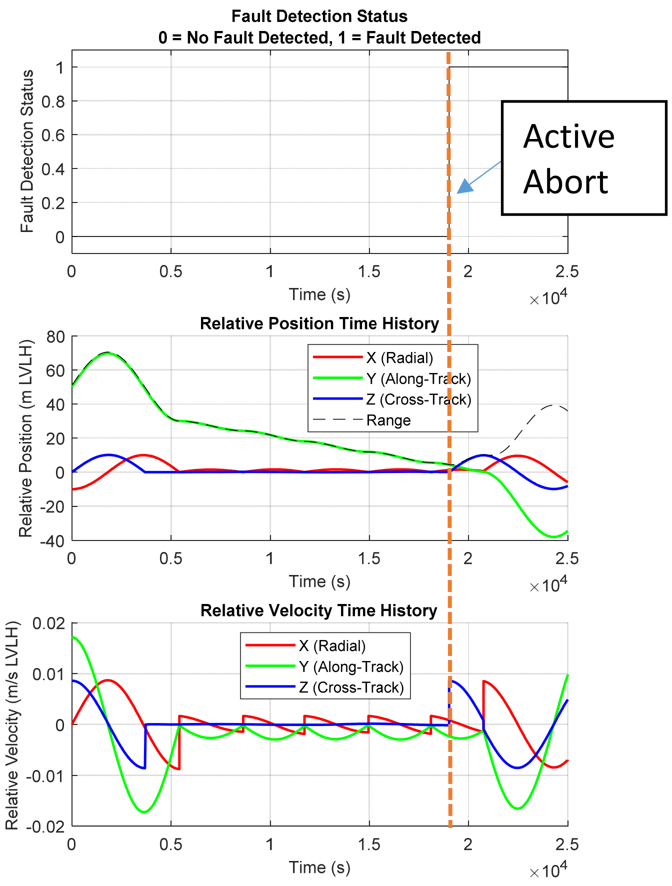


Fig. 13 Simulation results for camera power fault resulting in active abort

Each of the tasks described above has been completed successfully for an initial treatment of defining fault protection behavior for autonomous rendezvous and capture of the OS. A detailed rendezvous and capture process concept of operations has been created, accounting for safety concerns. The architecture has been demonstrated in simulation for several fault cases with fault responses dependent on the mode state of the system.

5. Conclusion

As aerospace vehicles become more complex and require increased automation, the design of FDIR systems must evolve to keep pace with vehicle advancements. A generic, modular, and portable architecture has been developed for aerospace vehicle fault protection. The architecture has been adapted to two distinct scenarios and has demonstrated the ability to successfully detect, diagnose, and respond to a variety of faults in real time using a state-based on-board system. Flight testing and detailed simulation have been used to thoroughly develop, verify, and validate this capability for two distinct case studies in very different regimes.

Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1148903. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation. FalconViz and the King Abdullah University of Science and Technology (KAUST) provided funding support and technical guidance for development of the UAV Nervous System. The Mars Sample Return fault protection work was completed under contract with the NASA Jet Propulsion Laboratory (JPL) at the California Institute of Technology.

McClain Goggin at Purdue University completed most of the MSR trajectory design work. Dr. Neil Smith at KAUST provided ideas, mentorship, personal support, and technical guidance that were invaluable to the UAV portions of this study. Special thanks to Rob Lock (Mars Program Office) and Peter Meakin (Fault Protection and Autonomy group supervisor) at JPL for their mentorship and assistance with the MSR portion of the study. Also, the JPL MSR study team and Rendezvous Working Group provided many inputs by participating in breakout discussions. Many engineers at JPL were consulted to solicit ideas for fault protection research.

References

[1] Fesq, L., Dennehy, N., Barth, T., Clark, M., Day, J., Fretz, K., Johnson, S., Hattis, P., McComas, D., Newhouse, M., Melcher, K., Rice, E., West, J., Zinchuk, J., "Fault Management Handbook", NASA Technical Handbook, NASA-HDBK-1002, Apr. 2012, https://www.nasa.gov/pdf/636372main_NASA-HDBK-1002_Draft.pdf [retrieved 12 Dec. 2016].

[2] Ingham, M.D., Rasmussen, R.D., Bennett, M.B., and Moncada, A.C., "Engineering Complex Embedded Systems with State Analysis and the Mission Data System", *Journal of Aerospace Computing, Information, and Communication*, Vol. 2, No. 12, Dec. 2005, pp. 507-536. doi: 10.2514/1.15265.

[3] Rasmussen, R.D., "Thinking Outside the Box to Reduce Complexity in National Aeronautics and Space Administration Flight Software", NASA Study on Flight Software Complexity, Jet Propulsion Laboratory, Pasadena, CA, Mar. 2009, https://www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf [retrieved 26 Oct. 2017].

[4] Cancro, G.J., Turner, R.J., Monaco, C.C., Wilson, Nguyen, L., Pekala, M.J., Olson, C.C., Kahn, E.G., "Emphasizing Understandability, Flexibility, and Verifiability in a Spacecraft Fault Management Autonomy System", *American Institute of Aeronautics and Astronautics Infotech@Aerospace Conference,* Seattle, WA, Apr. 2009. doi: 10.2514/6.2009-2029.

[5] Cancro, G.J., "APL Spacecraft Autonomy: Then, Now, and Tomorrow", *Johns Hopkins APL Technical Digest*, Vol. 29, No. 3, 2010, pp. 226-233, http://www.jhuapl.edu/techdigest/TD/td2903/Cancro_Autonomy.pdf [retrieved 19 Apr. 2018].

[6] Van Besien, B., "Investigating Model-Based Autonomy for Solar Probe Plus", Johns Hopkins Applied Physics Laboratory, Dec 2013, http://flightsoftware.jhuapl.edu/files/2013/talks/FSW-13-TALKS/BVB-FSW-Presentation.pdf [retrieved 13 Dec. 2016].

[7] Wander, A., Förstner, R., "Innovative Fault Detection, Isolation, and Recovery Strategies On-Board Spacecraft: State of the Art and Research Challenges", *Deutscher Luft- und Raumfahrtkongress 2012*, Berlin, Germany, Sept. 2012, http://www.dglr.de/publikationen/2013/281268.pdf [retrieved 19 Apr. 2018].

[8] Marzat, J., Piet-Lahanier, H., Damongeot, F., Walter, E., "Model-based fault diagnosis for aerospace systems: a survey", *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 226, No. 10, Jan 2012, 1329-1360. doi: 10.1177/ 0954410011421717.

[9] Muscettola, N., Nayak, P.P., Pell, B., Williams, B.C., "Remote Agent: to boldly go where no AI system has gone before", *Artificial Intelligence*, Vol. 103, No. 1–2, 1998, pp. 5–47. doi: 10.1016/S0004-3702(98)00068-X.

[10] Brown, M.B., Johnson, S.A., "An Overview of the Fault Protection Design for the Attitude Control Subsystem of the Cassini Spacecraft", *American Control Conference*, Philadelphia, PA, June 1998. doi: 10.1109/ACC.1998. 703535.

[11] Meakin, P.C., "Cassini Attitude Control Fault Protection Design: Launch to End of Prime Mission Performance", *AIAA Guidance, Navigation, & Control Conference and Exhibit*, Honolulu, HI, Aug. 2008. doi: 10.2514/6.2008-6809.

[12] Fesq, L.M., "MARPLE: An Autonomous Diagnostician for Isolating System Hardware Failures", Ph.D. Dissertation, Department of Computer Science, University of California Los Angeles, 1993.

[13] Kolcio, K.O., "Model-Based Fault Detection and Isolation System for Increased Autonomy", *AIAA Space 2016*, Long Beach, CA, Sept. 2016. doi: 10.2514/6.2016-5225.

[14] Schulte, P. Z. and Spencer, D.A., "Development of an Integrated Spacecraft Guidance, Navigation, & Control Subsystem for Automated Proximity Operations," *Acta Astronautica*, Vol. 118, Jan-Feb 2016, pp. 168-186.

doi: 10.1016/j.actaastro.2015.10.010.

[15] Spencer, D.A., Chait, S.B., Schulte, P. Z., Okseniuk, K.J., and Veto, M., "Prox-1 University-Class Mission to Demonstrate Automated Proximity Operations," *Journal of Spacecraft and Rockets*, Vol. 53, No. 5, July 2016, pp. 847-863. doi: 10.2514/1.A33526.

[16] Chait, S.B., Spencer, D.A, "Georgia Tech Small Satellite Real-Time Hardware-in-the-Loop Simulation Environment: SoftSim6D", Master's Project Report, Georgia Institute of Technology, Atlanta, GA, Dec 2015, http://ssdl.gatech.edu/sites/default/files/papers/mastersProjects/ChaitS-8900.pdf [retrieved 19 Apr. 2018].

[17] Schulte, P.Z., "A State Machine Architecture for Aerospace Vehicle Fault Protection", Ph.D. Dissertation, School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA, 2018, http://www.ssdl.gatech.edu/sites/default/files/papers/phdTheses/SchulteP-Thesis.pdf [retrieved 13 June 2018].

[18] H.J Kim, W. E. Wong, et. al, "Bridging the Gap Between Fault Trees and UML State Machine Diagrams for Safety Analysis", *2010 Asia Pacific Software Engineering Conference*, Sydney, Australia, Dec. 2010, 196-205. doi: 10.1109/APSEC.2010.31.

[19] Schulte, P.Z., Spencer, D.A., Smith, N.G., McCabe, M.F., "Development of a Fault Protection Architecture Based Upon State Machines", *67th International Astronautical Congress*, Guadalajara, Mexico, Sept. 2016, IAC-16-D1.IP.2x32540.

[20] Schulte, P.Z., Spencer, D.A., "State Machine Fault Protection for Automated Proximity Operations", *68th International Astronautical Congress*, Adelaide, Australia, Sept. 2017, IAC-17-C1.5.11x36573.

[21] Schulte, P.Z., Spencer, D.A., Goggin, M., "Mars Sample Return Terminal Rendezvous Fault Protection", *Journal of Spacecraft and Rockets*, submitted Apr. 2018.