68th International Astronautical Congress (IAC), Adelaide, Australia, 25-29 September 2017.

IAC-17-C1.5.11x336573

STATE MACHINE FAULT PROTECTION FOR AUTONOMOUS PROXIMITY OPERATIONS

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

[a] *Graduate Research Assistant, Space Systems Design Laboratory, Georgia Institute of Technology, Atlanta, Georgia, United States of America,* pzschulte@gatech.edu
[b] *Associate Professor, Space Flight Projects Laboratory, Purdue University, West Lafayette, Indiana, United States of America,* spencer@purdue.edu
* Corresponding Author

Abstract

The capability to recover gracefully from hardware or software faults is critical for many aerospace applications. This is particularly true for missions involving proximity operations, where multiple vehicles are operating at close range. Previous proximity operations missions have experienced faults that resulted in a failure to meet mission objectives. Fault protection systems are used to detect, identify the location of, and recover from faults. Typically, aerospace systems use a rule-based paradigm for fault protection, where telemetry values are monitored against logical statements such as static upper and lower limits. The model-based paradigm allows more complex decision logic to be used. The state machine approach for model-based fault protection has been explored by industry but has not yet been widely adopted for aerospace applications.

This study focuses on fault protection for the Guidance, Navigation, and Control vehicle subsystem, which is essential for any aerospace vehicle and has many complex and interrelated hardware and software components. Two separate case studies have been analyzed through this work, one for atmospheric flight and one for space flight. The first case involves detecting hardware faults on an unmanned aerial vehicle used for aerial surveying and mapping and is addressed in a previous paper. The second case is the focus of this paper and involves automated proximity operations during approach and capture of the orbiting sample canister for a Mars Sample Return mission.

For each case study, high-level failure modes are identified and linked to individual root cause events via fault tree analysis. The results of the fault tree analyses are developed into a generic and modular state machine fault protection architecture. This architecture will apply to a wide variety of aerospace applications and contains components that can be rearranged, added, or removed easily. The architecture facilitates export of the state machine logic to flight software via autocoding or other methods.

Keywords: fault protection; autonomy; proximity operations; state machines; model-based design; guidance, navigation & control

Acronyms/Abbreviations

FOV	Field-of-view
FP	Fault Protection
GN&C	Guidance, Navigation, & Control
I2C	Inter-Integrated Circuit
IR	Infrared
LIDAR	LIght Detection And Ranging
JPL	Jet Propulsion Laboratory
MATLAB	Matrix Laboratory
MAV	Mars Ascent Vehicle
MSR	Mars Sample Return
NASA	National Aeronautics & Space Administration
NeMO	Next Mars Orbiter
OS	Orbiting Sample container
ROCS	Rendezvous OS Capture System
SRO	Sample Return Orbiter
TBR	To Be Resolved
UAV	Unmanned Aerial Vehicle

1. Introduction

Fault protection (FP) systems for aerospace applications ensure that the vehicle can detect faults and respond autonomously to transition the vehicle into a safe state. This research develops a fault protection architecture that utilizes state machines for Fault Detection, Isolation, & Recovery. It builds on previous fault protection capabilities by incorporating state-based decision logic, model-based design, and autocoding to flight software, resulting in a generic, modular architecture that is portable to embedded systems. The architecture is broadly applicable for aerospace applications, with relative proximity operations as a specific case study. One important goal of this research is to increase autonomy in fault protection, which is vital for complex, time-constrained applications where real-time control by human operators is not feasible. Another goal is to advance model-based approaches to fault protection, in which the fault protection architecture keeps track of the state of the system via a

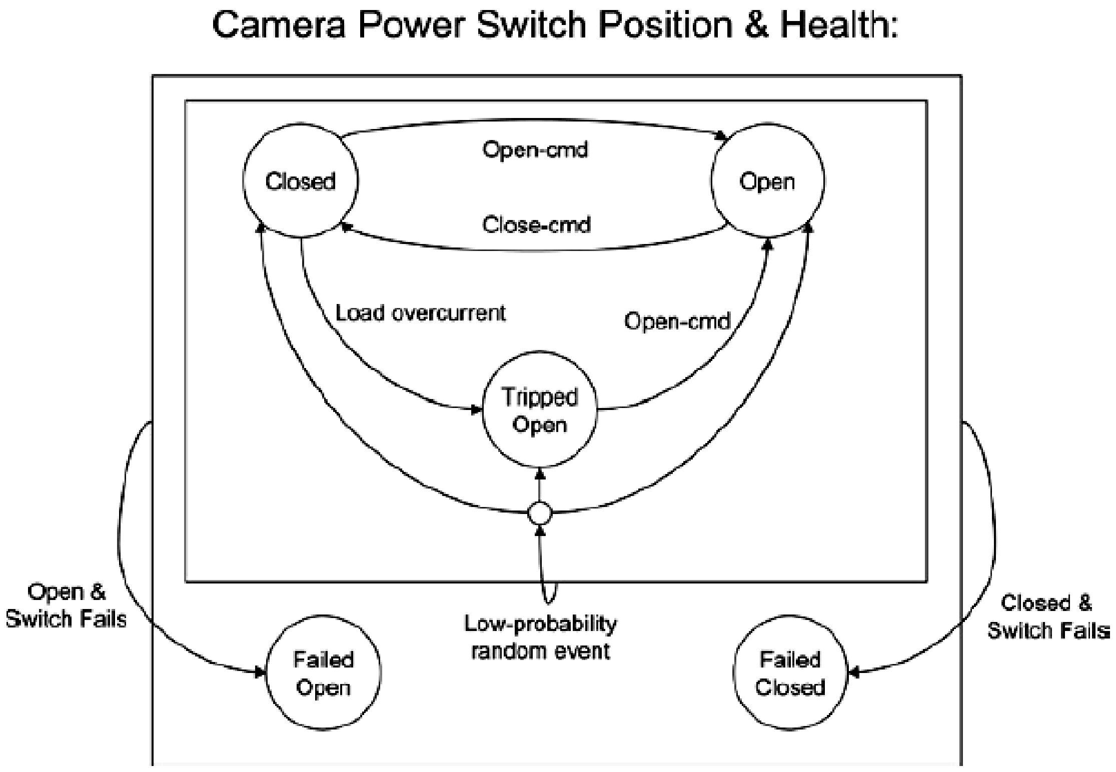


Fig. 1. Example usage of state machines for fault protection [4]

model of system behavior. Two primary applications are considered as case studies for this research: a rotorcraft Unmanned Aerial Vehicle (UAV) project with a start-up company called FalconViz [1] and a Mars Sample Return (MSR) project with the NASA Jet Propulsion Laboratory (JPL) at the California Institute of Technology [2].

1.1 State Machine Approach to Fault Protection

State machines are model-based representations of complex logical relationships. They provide a visual block-diagram development that is fairly straightforward to understand and is particularly useful for fault protection. 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 have yet to be widely adopted by industry for fault protection, even though they have been explored and used in some flight systems, as referenced in [1,3].

The example state machine in Figure 1 shows the possible states of a camera power switch. There are two primary states of the switch, “Open” and “Closed”. Transitions between the states are activated when an “Open-cmd” or “Close-cmd” command is sent to the camera. Additional states and transitions such as “Tripped Open” (a fault state) and “Load overcurrent” (a root cause of the fault state) are added to illustrate a known fault condition of the system. Another low-probability random event, such as a single-event upset caused by radiation, could allow an un-failed switch to arbitrarily change state. Finally, fault states “Failed Open” and “Failed Closed” are added into the system, showing how the state machine can be used to implement fault detection.

The “state” of a system includes any “aspects of the system that we care about for the purposes of control” [4]. Traditionally, state has included only continuous physical parameters such as position, velocity, attitude, temperature, and pressure. However, states can also include discrete quantities such as operating modes and device health. These discrete states can then be represented as state machines, as shown in Figure 1. 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 FP purposes can be developed in a way that represents all possible states relevant to mission success. Knowledge of the state is not the same as the state itself, and the status of a state machine representation at any time is only as accurate as the information that is provided to it. If input data is outdated or incorrect, the active state chosen by the state machine representation may be outdated or incorrect as well.

1.2 Paper Organization

The paper is organized in the following manner. Section 1 introduces background for the topic, Section 2 presents a summary of the UAV case study, Section 3 presents an introduction to the ongoing MSR case study, Section 4 presents the methods that have been applied for MSR FP, Section 5 presents results and discussion for MSR, and Section 6 presents forward work.

2. Summary of UAV Case Study

One application of the state machine FP architecture has been developed for a multirotor UAV system in use by FalconViz. This FP system has been dubbed the "UAV Nervous System," and serves as a proof-of-concept of the state machine FP architecture. A flight test demonstrating successful detection of faults has been completed, and the results were published at the 67th International Astronautical Congress in Guadalajara, Mexico [1]. Note that much of the content of this section is expanded in detail in that paper. A FalconViz hexacopter (six rotors) was used as a proof-of-concept testbed for the UAV Nervous System. Fault protection provides a more reliable vehicle for performing aerial surveys and other tasks with FalconViz UAVs.

A fault tree analysis was performed, providing a systematic top-down symbolic approach to model chains of possible faults for the system [5]. The fault tree is made up of a top level event, which is a foreseeable, "undesirable event toward which all fault logic paths flow" [6]. The top event is connected to various intermediate events that could cause it. In turn, each intermediate event is connected to other lower-

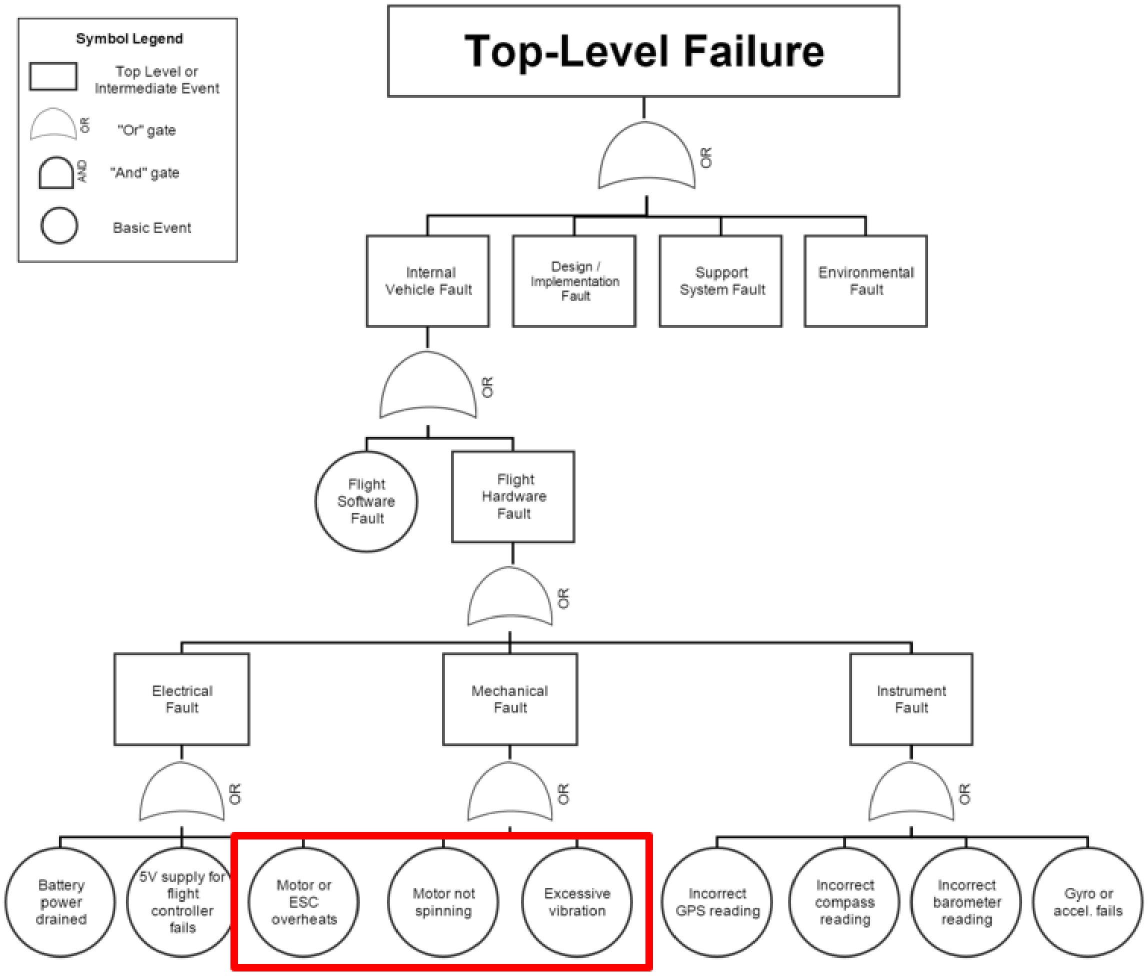


Fig. 2. Fault Tree for FalconViz UAVs, with most important faults highlighted

level events that could cause it. The bottom level is comprised of basic events or root cause events. These are initiating events whose cause is not analyzed further. Events are connected by "and" and "or" logic gates. High-level failure modes are identified and linked to individual root cause events.

The fault tree for the FalconViz UAVs is shown in Figure 2. This is an expansion of the Internal Vehicle Flight Hardware branch of a generic fault tree [1] and shows basic event faults identified by the FalconViz engineering team. The top-level failure for this analysis is "Loss of control" which can be traced back to each the identified faults. The three most important faults identified were "Motor not spinning," "Motor or ESC overheats" and "Excessive vibration," as indicated by the red box in Figure 2. In this study, each branch of the fault tree will be expanded to identify all possible root causes for the top event. One specific fault was addressed as a starting point: unbalanced propellers leading to excess vibration.

Vibration detection is accomplished by evaluating accelerometer data measured from the arms of the UAV that house the propellers. If a propeller is unbalanced, then there will be much more vibration in the system. A Simulink model records data from the accelerometer and feeds it through a supervised machine-learning classification algorithm. The algorithm determines the health of the system from the data. For testing, the propeller is unbalanced by adding a few pieces of electrical tape on one side. Flight test data is captured for both an unbalanced propeller (with tape) and a balanced propeller (without tape) and is used to train and validate the classification model in MATLAB on the ground.

Once model training and validation is complete, the system is ready for in-flight detection. Data collected from the accelerometer is fed into the Simulink model in real-time and reformatted. It is then fed into the MATLAB fault detection algorithm, which uses the trained model to determine if the propeller is balanced. If the live data is classified as "unbalanced" by the detection algorithm, then the vibration FaultDetected flag is set to 1; otherwise the flag is set to 0, indicating the propeller is "balanced." Then, a state machine shown in Figure 3 is used to determine the state of the system based on the persistence of fault detections.

The UAV Nervous System has been flight-tested and successfully indicates the state of vibrations during flight. Since publication of [1], additional sensor capability has been added for detection of current,

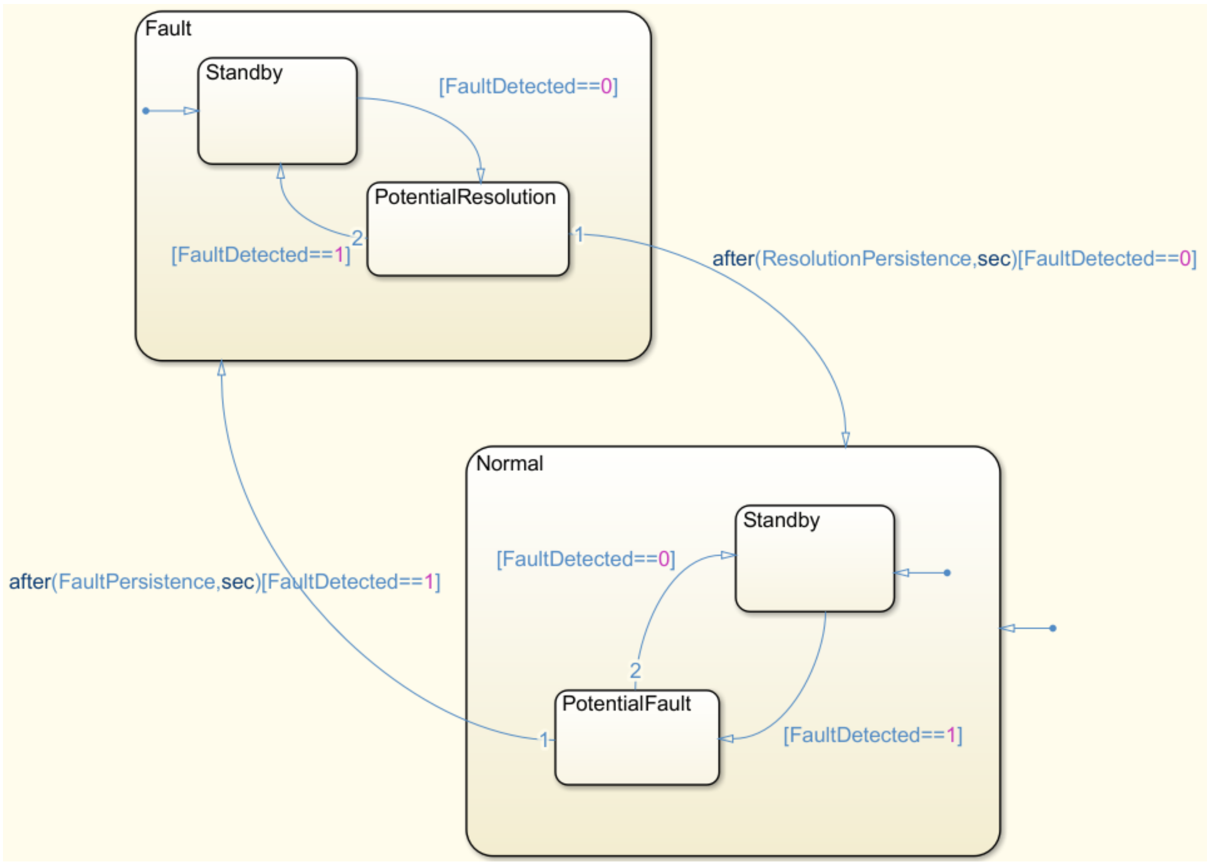


Fig. 3. Example of possible fault protection requirements [1]

voltage, and temperature anomalies. A second vibration sensor has also been added to another arm of the UAV, and additional sensors can be added easily using the I2C communication protocol. Similar state machines to Figure 3 have been added to the UAV Nervous System for each additional sensor. Data has been collected for all of these sensors in flight tests and will be used for verification and validation of the final fault protection architecture.

3. Introduction to Mars Sample Return Case Study

The top priority stated in the current planetary science decadal survey is to perform a Mars Sample Return mission [7]. In one mission concept, a Mars orbiter would rendezvous with a sample canister launched from the surface and capture it for return to Earth [2]. The last stage of this rendezvous operation, including capture, must be autonomous. During this autonomous phase, fault protection will be used to ensure that operations proceed under nominal conditions, take action to address certain fault conditions, or abort the capture phase if necessary. This research establishes a desired set of fault protection behaviors for the autonomous rendezvous and capture phase of the MSR orbiter mission. First, a fault tree analysis is performed to determine which faults should be considered based on input from subject matter experts. Several major areas are considered, including relative orbit determination, guidance & control, sequencing, and capture operations. Next, a selected subset of faults in each of these areas is expanded in detail. Criticality, detection, diagnosis, and response strategies are examined at various stages of the rendezvous and capture process. These details are used to define a set of potential fault protection requirements that accounts for different conditions in different stages of the process. Finally, a preliminary fault protection architecture is developed that shows how fault protection could be implemented during this unique and challenging mission phase. This architecture will utilize the state machine paradigm, a key tool of model-based design, rather than the current industry standard of rule-based fault protection.

3.1 Mars Sample Return Mission Context

In the current MSR mission concept, the Mars 2020 rover will collect soil and rock samples and cache them on the Martian surface. They would then be collected by a subsequent “fetch” rover (or other vehicle) and inserted into an Orbiting Sample container (OS). The OS would be placed on a Mars Ascent Vehicle (MAV) and launched into Mars orbit, as shown in Figure 4a. Once the MAV reaches orbit, it would release the OS. In one mission concept, the Sample Return Orbiter (SRO) would perform ground-in-the-loop rendezvous, as shown in Figure 4b, followed by autonomous approach and capture operations (also known as “terminal rendezvous and capture”) to collect the OS, as shown in Figure 4c. The fully autonomous portion would encompass approximately the last 100 meters of rendezvous and would occur during the sunlit portion of one orbit. Finally, the samples would be returned to Earth or cis-lunar space for recovery and laboratory study. The Guidance, Navigation, and Control (GN&C) process for the terminal rendezvous and capture phase is complex, with a number of risk areas that could result in

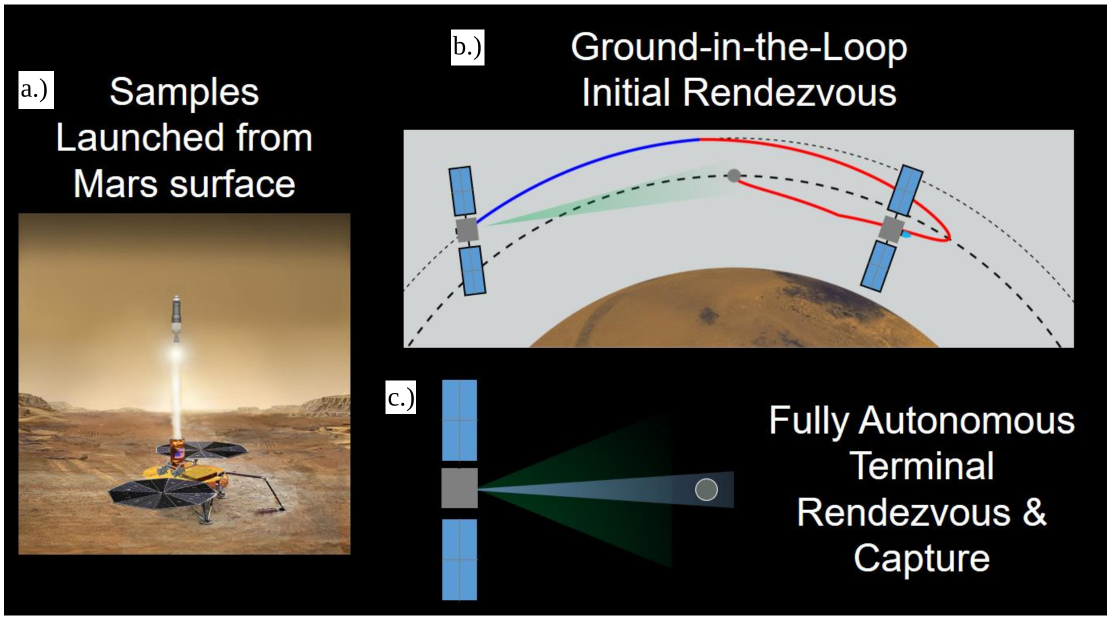

a.)
Samples Launched from Mars surface
b.)
Ground-in-the-Loop Initial Rendezvous
c.)
Fully Autonomous Terminal Rendezvous & Capture

Fig. 4. Mars Sample Return rendezvous concept

failure to capture the OS. This mission-critical autonomous activity presents the need for a comprehensive FP approach.

3.2 Desired Outcomes and Key Requirements

The terminal rendezvous and OS capture scenario provides an excellent case study for fault protection research. The main goal of this work is to develop a desired set of FP behaviors for autonomous rendezvous and capture of the OS. This is being done in collaboration with the Next Mars Orbiter (NeMO) mission formulation team at JPL, which is currently studying this problem. The NeMO rendezvous working group is made up of members from three disciplines: relative orbit determination, guidance & control, and sequencing. A separate team is developing concepts for the flight hardware subsystem that will perform the capture operation, known as the Rendezvous OS Capture System (ROCS). Inputs have also been sought from the NeMO flight systems working group about various aspects of spacecraft subsystem concepts.

There are three desired outcomes of this study. First, a set of initial fault protection requirements should be defined. The requirements may then be used to drive the initial design of the SRO rendezvous and capture system. Next, the NeMO team desires to integrate fault protection with mission concept development, influencing design decisions during Pre-Phase A based on fault protection considerations. Finally, the NeMO team desires to apply the fault protection process used in this study to other aspects of the NeMO design.

Several key requirements guided this study. First, mission success is vital. Fault protection should be designed to ensure that the SRO mission to capture the OS can be completed or aborted without ground intervention, even under fault conditions. As stated earlier, autonomy is a key feature, since the terminal rendezvous and capture will occur fully autonomously. Safety is a key concern, and fault protection should prevent the spacecraft from colliding with the OS. Finally, time criticality should be taken into account. For example, a fault response may be quite different at the beginning of the autonomous rendezvous sequence 100 m from the target than in the last 5 or 10 meters.

4. Methods for Mars Sample Return Case Study

Several different tasks were undertaken in order to define the fault protection behavior for autonomous rendezvous and capture. Some of these are standard fault protection practices and others were customized for this study. All tasks have been completed at the preliminary level only, since detailed design has yet not begun for this mission concept.

4.1 Fault Tree Analysis

The first step in developing fault protection requirements is to perform a fault tree analysis. Through discussions with the NeMO rendezvous working group, a fault tree was defined that captures faults that could result in failure of the terminal rendezvous phase, as

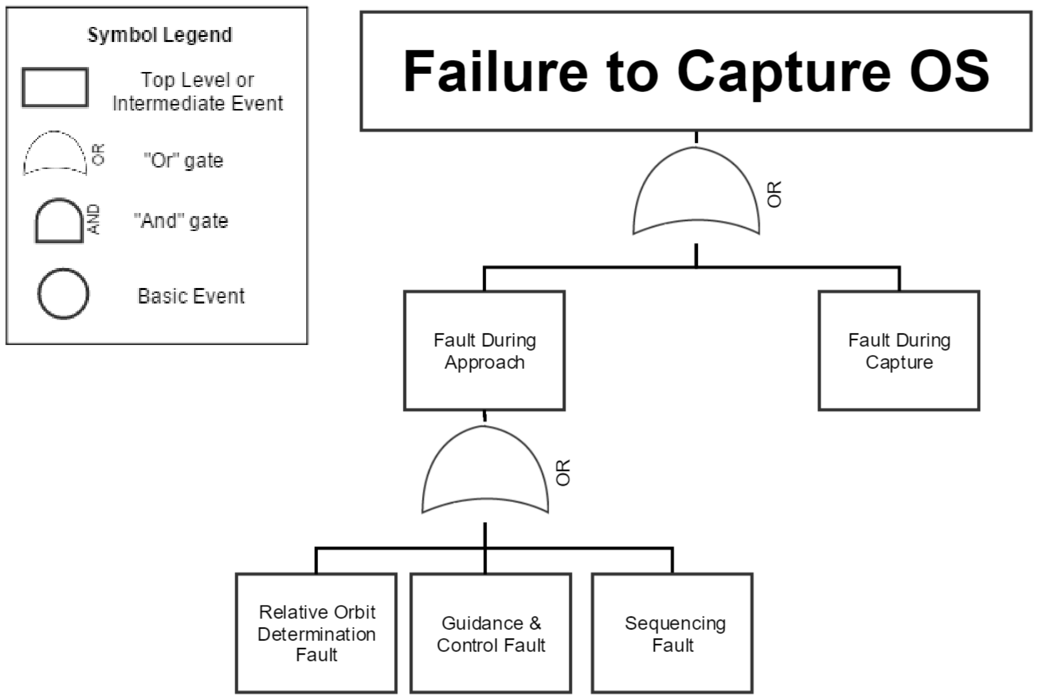


Fig. 5. High-level fault tree for autonomous rendezvous and capture

shown in Figure 5.

The first discipline considered is Relative Orbit Determination, which involves calculating relative position & velocity from rendezvous sensor data. Next is Guidance & Control, which is responsible for attitude determination/control and trajectory control during rendezvous. In addition, sequencing uses the Virtual Machine Language (VML) to direct the autonomous process based on state machines that are developed on the ground and loaded onboard [8]. Finally, Capture deals with the mechanical and logical components within ROCS.

An initial fault tree was developed prior to consulting the NeMO team. However, in order to capture the inputs from JPL experts representing each discipline, various breakout meetings were held to revise and expand this initial fault tree. These meetings were designed to simply brainstorm, add, remove, rearrange, or rename potential faults from the fault tree. Figure 6 shows one example of the results of these

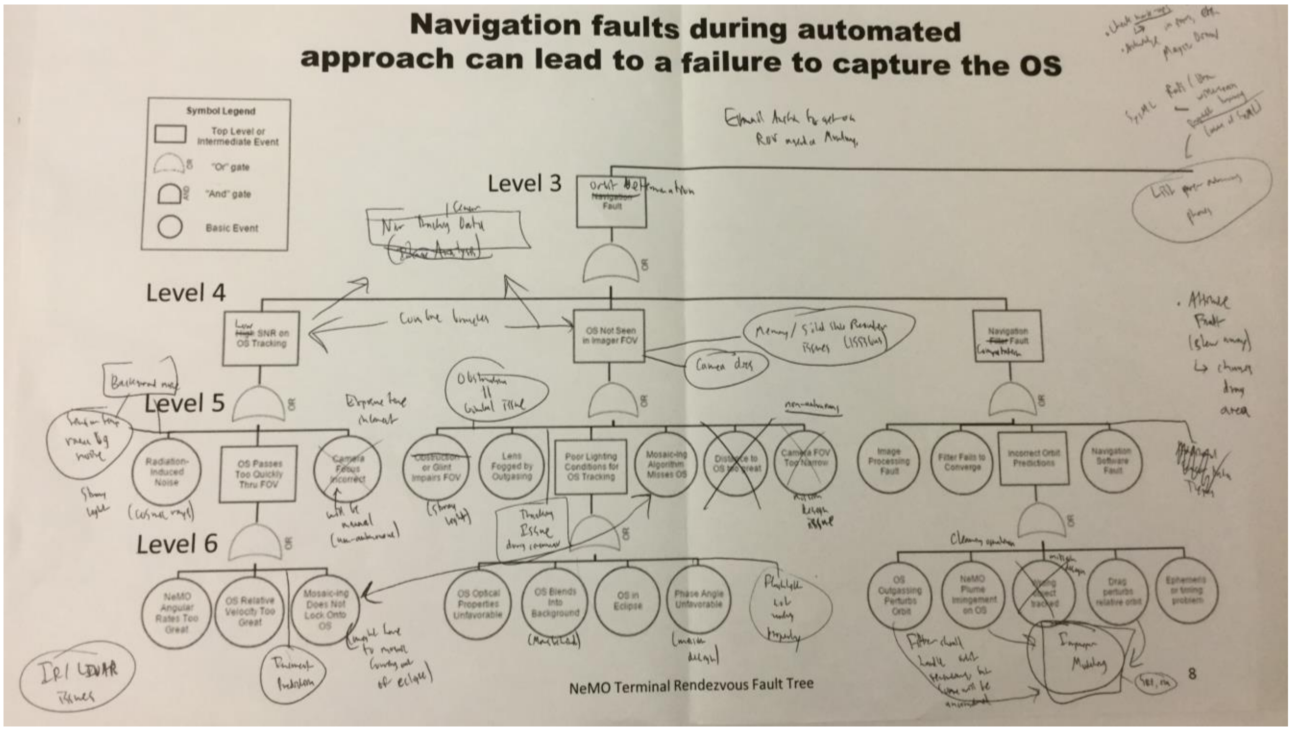

Navigation faults during automated approach can lead to a failure to capture the OS
Symbol Legend
Top Level or Intermediate Event
"Or" gate
"And" gate
Basic Event
Level 3
Orbit Determination
~~Navigation~~ Fault
Level 4
Low ~~High~~ SNR on OS Tracking
OS Not Seen in Imager FOV
Navigation ~~Filter~~ Fault Computation
Combine branches
Camera dirty
Level 5
Radiation-Induced Noise
OS Passes Too Quickly Thru FOV
~~Camera Focus Incorrect~~
~~Obstruction~~ or Glint Impairs FOV
Lens Fogged by Outgassing
Poor Lighting Conditions for OS Tracking
Mosaic-ing Algorithm Misses OS
~~Distance to OS too great~~
~~Camera FOV Too Narrow~~
Image Processing Fault
Filter Fails to Converge
Incorrect Orbit Predictions
Navigation Software Fault
Level 6
NeMO Angular Rates Too Great
OS Relative Velocity Too Great
Mosaic-ing Does Not Lock Onto OS
OS Optical Properties Unfavorable
OS Blends Into Background
OS in Eclipse
Phase Angle Unfavorable
OS Outgassing Perturbs Orbit
NeMO Plume Impingement on OS
~~Wrong object tracked~~
Drag perturbs relative orbit
Ephemeris or timing problem
Improper Modeling
IR / LIDAR
NeMO Terminal Rendezvous Fault Tree
8
OR

Fig. 6. Result of a fault tree brainstorming session for relative orbit determination

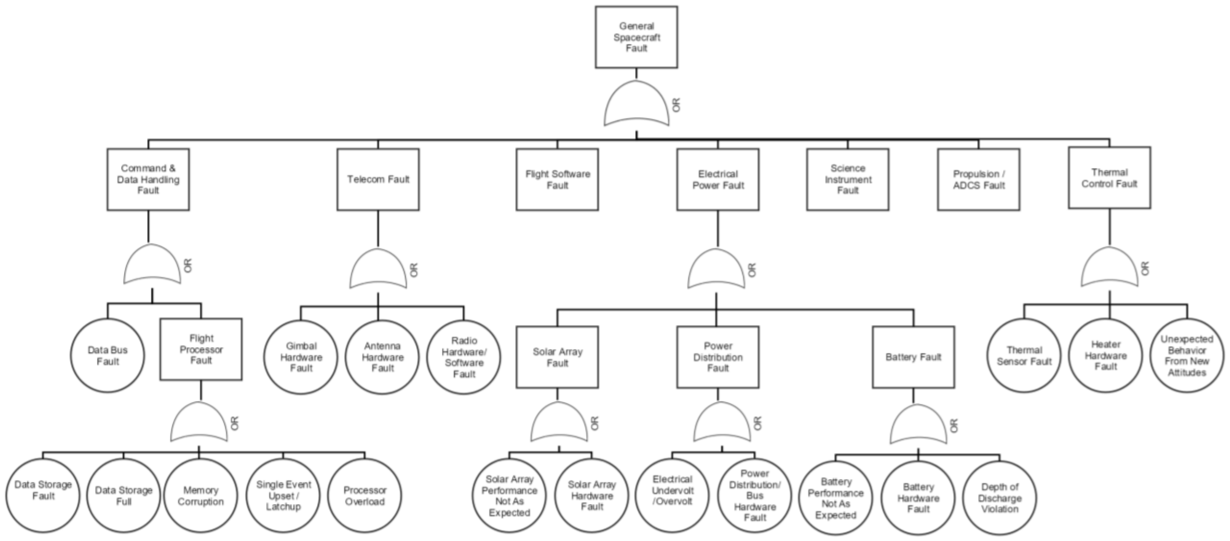


Fig. 7. General spacecraft subsystem fault tree

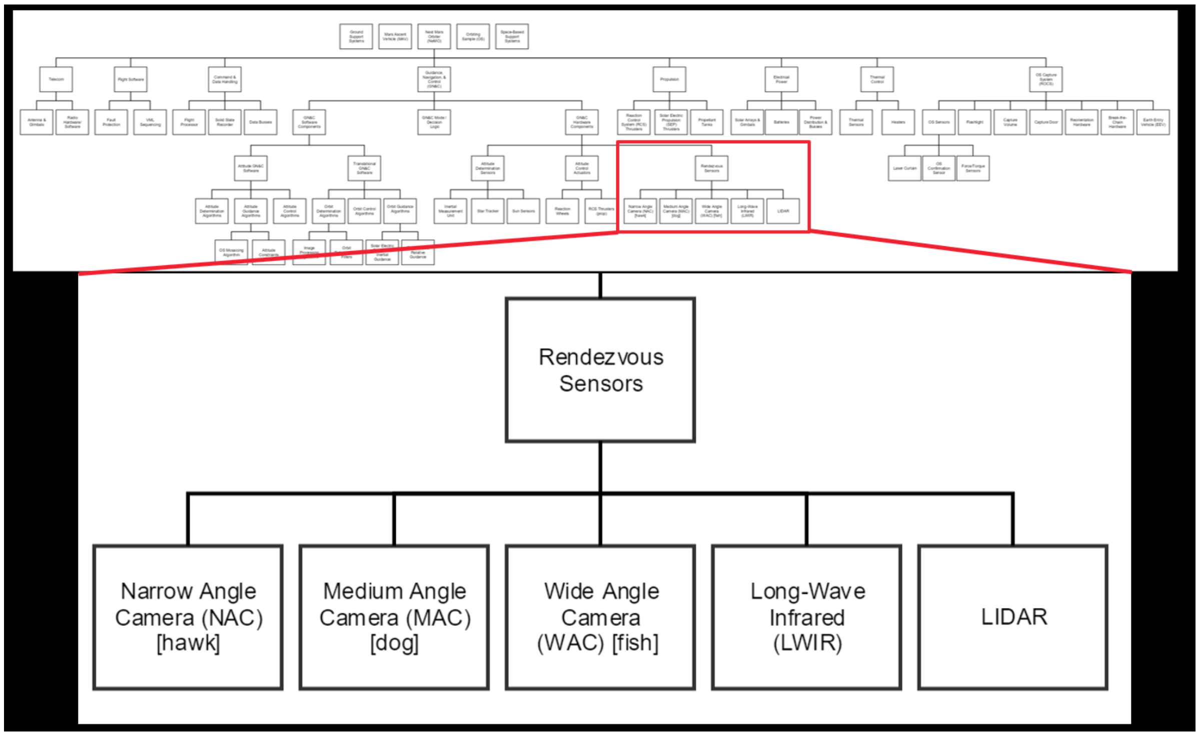

Rendezvous Sensors
Narrow Angle Camera (NAC) [hawk]
Medium Angle Camera (MAC) [dog]
Wide Angle Camera (WAC) [fish]
Long-Wave Infrared (LWIR)
LIDAR

Fig. 8. Subsystem taxonomy, with an example expanded

breakout sessions.

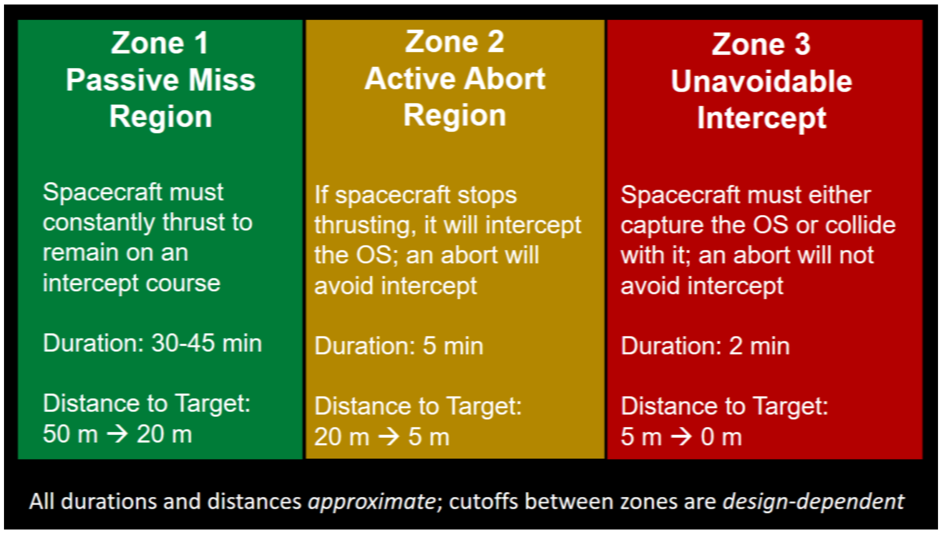
Finally, the results of all the breakout discussions were compiled to create comprehensive fault trees. A complete fault tree including over 50 root cause events (shown in the Appendix in text form because of space limitations) was constructed to capture all faults specific to rendezvous and capture. A second fault tree shown in Figure 7 was used to capture generic spacecraft subsystem faults that could occur during rendezvous.

4.2 Subsystem Taxonomy

To aid in clarifying terminology, a subsystem taxonomy (or system block diagram), shown in Figure 8, was constructed to list out various elements, subsystems, and components of the system. This block diagram was also used to help team members understand conceptually what components should be considered for the fault protection process. An example of the terminology clarification is the naming of various rendezvous cameras, shown in the expansion of Figure 8. Because the terms "Narrow Angle Camera" and "Wide Angle Camera" have different meanings in different contexts, the rendezvous team developed animal names for each camera. The "hawk" is a camera that can see far away, the "dog" is a shorter-range camera with a wider field of view, and the "fish" is a very wide-angle camera with a short range.

4.3 Concept of Operations

The next challenge was to link fault protection behavior to the state of the spacecraft within the rendezvous process. Along with members of the NeMO rendezvous team, a detailed process was defined for terminal rendezvous and capture. Defining the rendezvous process included specifying "Zones of Criticality" for the terminal approach. These zones are then used to alter fault protection behavior based on distance to the target and time to intercept. The zones are shown in Figure 9. Note that the durations and distances shown are very dependent on the rendezvous approach strategy selected, so the transition conditions between these zones may change, but the criticality (and thus impact on FP behavior) of the zones will endure regardless of the implementation selected.



Zone 1 Passive Miss Region	Zone 2 Active Abort Region	Zone 3 Unavoidable Intercept
Spacecraft must constantly thrust to remain on an intercept course	If spacecraft stops thrusting, 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
Duration: 30-45 min	Duration: 5 min	Duration: 2 min
Distance to Target: 50 m → 20 m	Distance to Target: 20 m → 5 m	Distance to Target: 5 m → 0 m

All durations and distances *approximate*; cutoffs between zones are *design-dependent*

Fig. 9. Notional "zones of criticality"

The first zone is called the "passive miss region". During this zone the SRO must constantly thrust to remain on an intercept course with the OS. If the SRO stops thrusting (a passive abort), then it will pass by the OS harmlessly. Zone 1 should last about 30-45 minutes and the distance to the target will close from 50 m to about 20 m. The second zone is called the "active abort region". During this zone, if the SRO stops thrusting it will intercept the OS, but if an active abort maneuver is performed, intercept can be avoided. Zone 2 should last around 5 minutes, and the distance from the target will close from 20 m to about 5 m. The third zone is called the "unavoidable intercept region". During this zone, the SRO can no longer avoid an intercept even if an abort maneuver is performed; it must either capture the OS or collide with it. Zone 3 should last around 2 minutes and the distance from the target will close from 5 m to zero.

The rendezvous process was developed into a state machine that will be used by the fault protection system to determine how to respond to various faults as they are detected. Fault responses will be calibrated based on the relative risk to the mission in each zone. The state machine, shown in Figure 10, represents both nominal and off-nominal processes. The system begins in Passive Standby at the end of the ground-in-the-loop

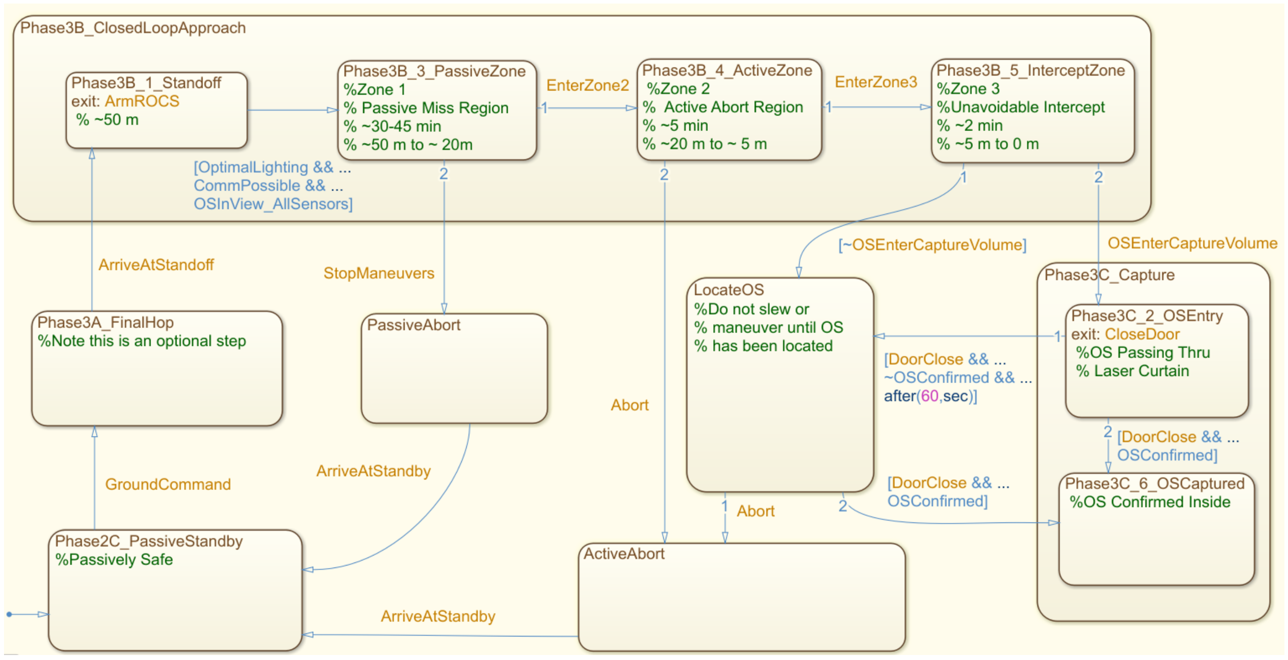


Fig. 10. State machine for rendezvous and capture process

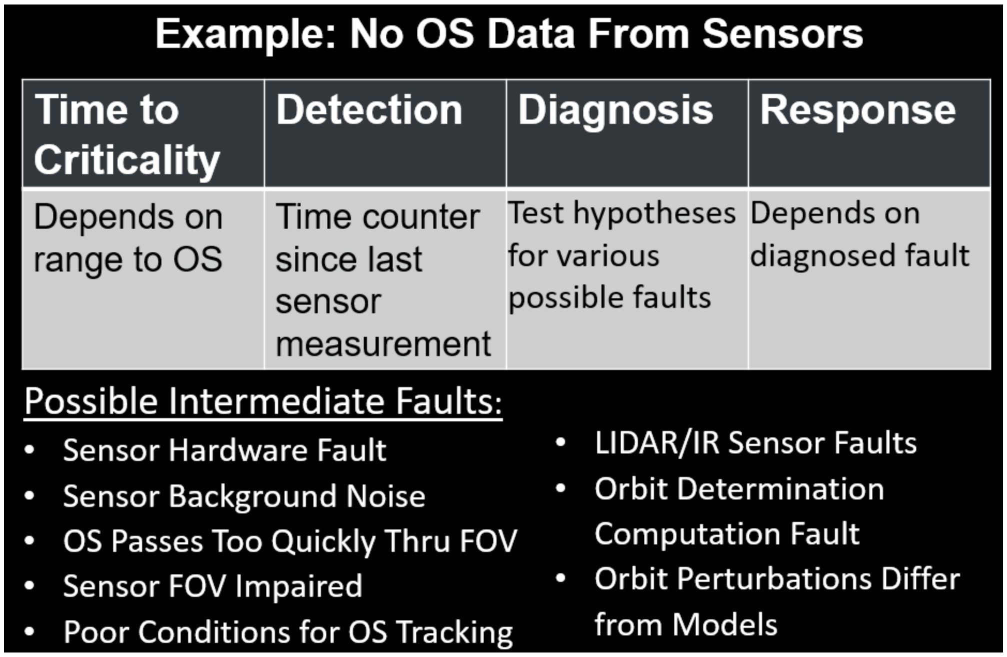
rendezvous process; this is the state in the bottom left of Figure 10. This is a passively stable trajectory that will not impact the OS even if it drifts. A ground command will be given to initiate the autonomous sequence. In one strategy, this would include a Final Hop from the passively stable trajectory to the terminal approach corridor. This optional step may be removed later.

If the Final Hop step is utilized, that step would end at a standoff position at the start of the terminal corridor. When proper lighting and communication conditions are achieved and the OS has been acquired by all rendezvous sensors, the "closed-loop" approach will begin. The system then enters the "Passive Miss Region". If at any point in this region something goes wrong, the system simply stops maneuvers and enters Passive Abort. Once the vehicle is safe from collision with the OS, it would return to the Passive Standby state and await ground analysis and command to resume autonomous operations. If no problems occur, the system will enter the "Active Abort Region", when the dynamic boundary is crossed. If at any point something goes wrong, an abort can be commanded to return to Passive Standby via the Active Abort mode. Finally, just before intercept the system enters the "Unavoidable Intercept Region".

If capture is unsuccessful and the OS does not enter the capture volume, the system enters the "LocateOS" state. It will attempt to determine where the OS is located 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. A confirmation sensor will then verify that the OS is inside. If the OS cannot be confirmed inside the capture volume after the door has closed, the system also enters the LocateOS state and will abort unless the OS is found inside the capture volume.

4.4 Fault Protection Requirements

Next, a subset of faults (bolded in the fault tree shown in the Appendix) was selected from the completed fault tree. Several representative faults were chosen from each area (relative orbit determination, guidance & control, sequencing, and capture). The selected faults are challenging to detect, diagnose, or respond to in a quick, efficient, and safe way. These faults were expanded in detail, and time-to-criticality, detection methods, diagnosis methods, and response strategies were examined at various stages of the rendezvous and capture process. These details were then used to define a set of potential fault protection requirements that accounts for different conditions in different stages of the rendezvous process. One example strategy for a single fault is shown in Figure 11, and the related possible fault protection requirements are shown in Figure 12. Finally, a second round of breakout meetings was held with technical experts in each area to share the results and seek direction for the next steps.



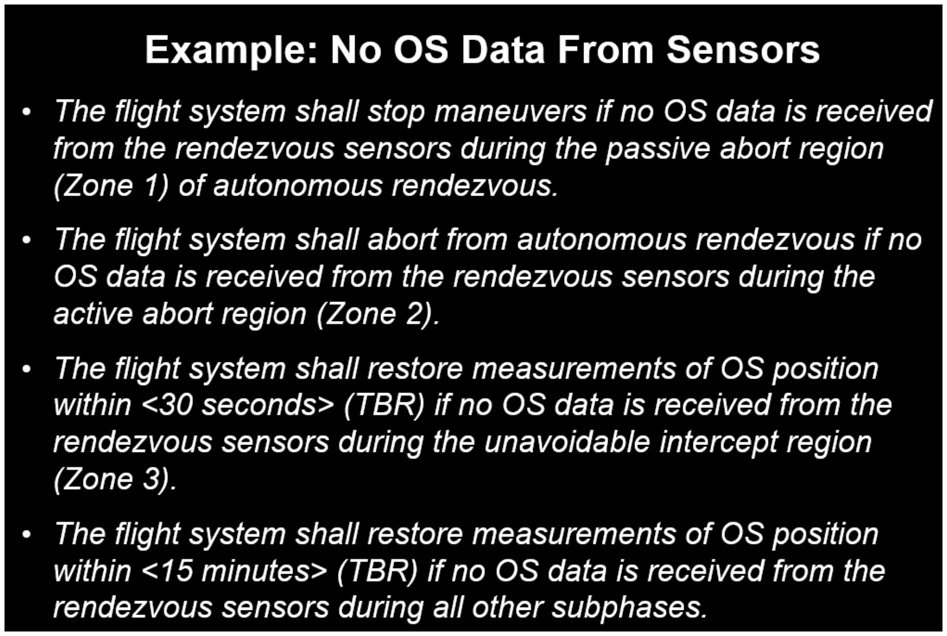
Example: No OS Data From Sensors

Time to Criticality	Detection	Diagnosis	Response
Depends on range to OS	Time counter since last sensor measurement	Test hypotheses for various possible faults	Depends on diagnosed fault

Possible Intermediate Faults:

- Sensor Hardware Fault
- Sensor Background Noise
- OS Passes Too Quickly Thru FOV
- Sensor FOV Impaired
- Poor Conditions for OS Tracking
- LIDAR/IR Sensor Faults
- Orbit Determination Computation Fault
- Orbit Perturbations Differ from Models

Fig. 11. Example fault protection strategy



Example: No OS Data From Sensors

- *The flight system shall stop maneuvers if no OS data is received from the rendezvous sensors during the passive abort region (Zone 1) of autonomous rendezvous.*
- *The flight system shall abort from autonomous rendezvous if no OS data is received from the rendezvous sensors during the active abort region (Zone 2).*
- *The flight system shall restore measurements of OS position within <30 seconds> (TBR) if no OS data is received from the rendezvous sensors during the unavoidable intercept region (Zone 3).*
- *The flight system shall restore measurements of OS position within <15 minutes> (TBR) if no OS data is received from the rendezvous sensors during all other subphases.*

Fig. 12. Example of possible fault protection requirements

5. Results and Discussion

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 fault tree has been defined, along with a system block diagram and a detailed rendezvous and capture process concept of operations. Initial fault protection strategies and requirements have been generated for a total of about 20 key faults from the various discipline areas.

5.1 Observations

One goal of introducing fault protection earlier in the design cycle (during Pre-Phase A mission formulation) has been to help guide mission design considerations. One major observation is that the process of fault protection has been a forcing function for the NeMO mission formulation team to clarify some of their architecture and concept of operation decisions. In some cases, mission design and concept of operations assumptions have been documented for the first time. This has been an unexpected but welcome result, showing the value of fault protection not just as an add-on to a space mission design but as an essential component of the system design from the beginning.

5.2 Lessons Learned

In retrospect, there are a few things that could be done differently based on lessons learned. First, it would make sense to build the fault tree with a more functional structure rather than one based on rendezvous discipline areas. For example, if the OS is not seen in the rendezvous sensor's field of view, there could be an issue with relative orbit determination, attitude control, or sequencing that could cause this. The current version of the fault tree places this fault under relative orbit determination and not the other two branches. A more functional structure was suggested by a JPL fault protection expert, but the work was already far enough along with the process that it was decided to leave the fault tree in its current format.

Defining terminology clearly is very important. There have been miscommunications at several meetings because of different understandings for the definitions of certain terminology. For example, although terms like "guidance, navigation, & control" have fairly standard definitions, they may have different connotations in different contexts. Even the term "fault protection" means different things to different people. This challenge has been addressed by inviting open discussion and feedback in group meetings and by attempting to clarify any terms that could be confusing or misunderstood when they are presented. When developing tools like a fault tree, it is important to anticipate how terms will be understood by engineers from various disciplines and define any terms that may be misinterpreted.

Another challenge has been determining what to do next when each step is completed. Since a new method of fault protection design is being experimented with, there is not a defined process to follow. A final challenge has been a backlog in the communication of progress throughout the project. Because of the cadence of meeting cycles, work is often completed several weeks before it can be communicated to all relevant stakeholders. These challenges have been addressed by seeking additional direction and advice of fault protection experts and rendezvous/capture subject matter experts. Their suggestions have helped refine the direction of the study.

6. Forward Work

This work will be continued under a JPL contract with NeMO at Georgia Tech. The first step will involve developing a high-level flowdown of the fault protection architecture. This will include a conceptual design that identifies the arrangement of various components within the fault protection system, including how they will interact with each other and with other hardware and software components of the SRO. The next step is to create a preliminary implementation of the fault protection architecture in state machine form using the Stateflow toolbox within MATLAB/Simulink (or similar state machine tools). This system will be tested using simple simulated inputs (i.e. step functions) to demonstrate base-level functionality.

An example case ("No OS data received from sensors") has been selected to demonstrate how this fault protection system may diagnose faults on-board rather than relying on prior ground diagnosis assigning certain symptoms to specific faults. The fault protection architecture will utilize the state machine paradigm, a key tool of model-based design, rather than the current industry standard of rule-based fault protection. In this way, fault responses will be tied to the state of the system rather than simply as a reaction to the detection of symptoms.

Finally, the MSR work will be used as a case study for the broader research on fault protection. The FP state machine system developed for the MSR application will be made more generic, and the UAV flight test data will be used for verification and validation of the generic FP system. There is potential for some of the MSR work to continue at JPL by adding more detail for the NeMO rendezvous and capture FP strategy and by extending these FP concepts to other aspects of MSR concept development. Some of these FP methods could also be fed back into JPL's general fault protection process to continue improving them for current and future missions.

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 authors and do not necessarily reflect the views of the National Science Foundation.

FalconViz and 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 and will be continued under contract with the NASA Jet Propulsion Laboratory at the California Institute of Technology. Special thanks to Rob Lock (NeMO Project System Engineer) and Peter Meakin (Fault Protection and Autonomy group supervisor) for their assistance. Also, the NeMO team provided many inputs by participating in breakout discussions:

NeMO Rendezvous team: Austin Nicholas, Alan Didion

- Navigation: Ed Riedel, Rob Haw, Eric Gustafson, Drew Vaughan, Dylan Boone
- Guidance/Control: George Chen, Jack Aldrich
- Sequencing: Chris Grasso, PJ Guske

ROCS team: Joe Parrish, Paulo Younse, Marco Dolci

NeMO Flight System team: Kristina Larson, Ny Sou Okon, Travis Imken

Many engineers around JPL were consulted to solicit ideas for fault protection research. Their insights will be valuable as this work moves to the next stage: Lorraine Fesq, Kevin Barltrop, Jeff Levison, John Day, Bob Rassmussen, Mike Sievers, Magdy Bareh, Gene Lee, Garth Watney, Len Reder, Rob Bocchino, Tracy Neilson, Rajeev Joshi, Jean-Francois Castet, Ken Starr, Lermont Khachikyan, Bobak Ferdowsi, and John West.

References

[1] P.Z. Schulte, D.A. Spencer, N.G. Smith, M.F. McCabe, Development of a Fault Protection Architecture Based Upon State Machines, IAC-16-D1.IP.2x32540, 67th International Astronautical Congress, Guadalajara, Mexico, 2016, 26-30 September.

[2] J.E. Riedel, J. Guinn, et al., A Combined Open-Loop and Autonomous Search and Rendezvous Navigation System for the CNES/NASA Mars Premier Orbiter Mission, 26th Annual AAS Guidance and Control Conference, Breckenridge, Colorado, USA, Feb 2003.

[3] P.Z. Schulte, D.A. Spencer, Development of an Integrated Spacecraft Guidance, Navigation, & Control Subsystem for Automated Proximity Operations, Acta Astronautica, 118 (Jan-Feb 2016), 168-186, doi:10.1016/j.actaastro.2015.10.010.

[4] M.D. Ingham, R.D. Rasmussen, M.B. Bennett, and A.C. Moncada, Engineering Complex Embedded Systems with State Analysis and the Mission Data System, Journal of Aerospace Computing, Information, and Communication, 2 (Dec. 2005).

[5] B.E. Goldberg, K. Everhart, et al., Systems Engineering "Toolbox" for Design-Oriented Engineers, NASA Reference Publication 1358, Dec. 1994, http://www.hq.nasa.gov/office/codeq/doctree/rp1358.pdf, (accessed 13.12.16).

[6] R.J. Simmons, Fault Tree Analysis, Tunghai University, Feb 2009, http://www2.nuu.edu.tw/~er/reportfile/saminar/Fault_Tree_Analysis.pdf, (accessed 13.12.16).

[7] "Vision and Voyages for Planetary Science in the Decade 2013-2022," Committee on the Planetary Science Decadal Survey, National Research Council of the National Academies, 2011, https://solarsystem.nasa.gov/docs/131171.pdf (accessed 04.09.17).

[8] C.A. Grasso, VML 3.0 Reactive Rendezvous and Docking Sequencer for Mars Sample Return, AIAA SpaceOps Conference, Pasadena, California, USA, 2014, 5-9 May.

Appendix: Full Fault Tree for Mars Sample Return Terminal Rendezvous and Capture Phase

- Failure to Capture the OS
  - Fault During Approach
    - Relative Orbit Determination Fault
      - Rendezvous Sensor Data Fault
        - Sensor Hardware Fault
          - Sensor Loses Power
          - Sensor Settings Incorrect
          - Solid-State Recorder Malfunction
        - Sensor Background Noise
          - Radiation-Induced Noise
          - Temperature-Induced Noise
          - **Stray Light Glint**
        - OS Passes Too Quickly Through Imager FOV
          - SRO Angular Rates Too Great
          - OS Relative Velocity Too Great
        - Sensor FOV Impaired
          - Lens Fogged Due to Outgassing
          - Spacecraft Component in FOV
          - Debris in FOV
        - Poor Conditions for OS Tracking
          - OS Surface Properties Unfavorable
          - OS Blends Into Background
          - OS in Eclipse or Shadow (visual only)
          - Phase Angle Unfavorable
          - Flashlight Malfunction (visual only)
        - LIDAR/IR Sensor Faults
        - **No OS Data from Sensors**
      - Orbit Determination Computation Fault
        - Image Processing Fault
        - Navigation Software Fault

        - Mosaicing Algorithm Misses OS
        - Orbit Perturbations Differ from Models
            - OS Outgassing Perturbs Orbit
            - **SRO Plume Impingement on OS**
            - Atmospheric Drag Perturbs Relative Orbit
            - Other Orbit Perturbation Mismodeling
        - Incorrect Model Parameters (i.e. OS optical properties)
        - Ephemeris or Timing Fault
        - Filter Does Not Converge
- Guidance & Control Fault
    - Attitude Fault
        - Degraded Attitude Knowledge
            - Inertial Measurement Unit Fault
                - No Data Output
                - Reset/Excessive Reset
                - **Bias/Scale Factor Offset**
                - Measurement Drift
            - Star Tracker Fault
                - No Output
                - Bias/Incorrect Output
                - Excessive Current Draw
                - Optics Contamination
                - Optics Coating Degradation
                - False/Intermittent Star ID
                - **Temporary Lock-Up**
                - Noisy Measurements
                - Sun Sensor Fault
            - **Attitude Filter Does Not Converge**
        - Degraded Attitude Control
            - Reaction Wheel Fault
                - Wheel Stuck/Seized/Not Rotating
                - Increased Drag/Friction
                - Excessive Current Draw
                - Excessive Vibration
                - Tachometer Fault
                - Drive Electronics Fault
                - **Wheel Momentum Saturated**
            - Reaction Control System Thruster Fault
                - Thruster Fails to Actuate
                - Thruster Stuck On
                - Tank heater fault
                - Propellant line freezing
    - Trajectory Fault
        - Maneuver Fault
            - **Incorrect Timing, Direction, or Delta-V for Burn**
            - **Deadband Violation Does Not Trigger Manuever**
        - Degraded Translational Control

- Unable to place OS inside capture cone
  - **OS not aligned with capture cone**
  - **Rotation Rate Too High**
  - **Relative Velocity Too High**
- Guidance & Control Software Fault
- Sequencing Fault
  - Spacecraft Reboot
  - Unable to Meet Conditions that Allow Transfer to a Key State
    - **Tolerances on parameters too tight**
    - Unexpected configuration
    - Telemetry Reporting Fault
    - Logical Error in Sequence
  - Premature Entry into any State
    - Logical Error in Sequence
    - Sequencing Software Coding Fault
  - **Ground Command Halts or Unloads Sequence**
  - Incorrect Config File Version Loaded
- Fault During Capture
  - Capture Door Closure Fault
    - Door Close Timing Fault (Early/Late Closure)
    - **Door Close Signal Does Not Activate**
    - Door Mechanism Fault
  - Unexpected OS Dynamics
    - **OS Spin Rate Exceeds Capture Requirement**
    - OS Energy Exceeds Capture Capability
    - OS Impacts ROCS Components
  - Capture Detection Sensor Fault
    - Door Sensor Fault
    - OS Confirmation Sensor Fault
    - Force/Torque Sensor Fault
    - **Sun Interference/Spoofing**