Separable Architecture for Fault Isolation and Recovery

Matthew C. Ruschmann, John McGreevy
Emergent Space Technologies, Inc.
7901 Sandy Spring Road, Suite 511, Laurel, MD 20723; (301) 345-1535
matthew.ruschmann@emergentspace.com

ABSTRACT

Fault management is one of the key technologies that enable distributed and disaggregated mission architectures wherein multiple vehicles work cooperatively and autonomously in a cluster or formation, a typical mission concept involving small satellites. In this paper, we describe a software architecture, called Separable Architecture for Fault Isolation and Recovery (SAFIR), which addresses fault management for these types of missions. Although SAFIR is applicable to any system of systems, this paper demonstrates SAFIR for a cluster of spacecraft. The resulting fault detection, isolation, and recovery benefits from the SAFIR architecture because it is robust to intermittent communication and highly modular. The SAFIR software has been developed as apps for the Core Flight System (cFS) and has been demonstrated successfully on representative hardware using a high fidelity simulation of spacecraft in low earth orbit.

INTRODUCTION

In the near future, we will see the development of distributed and disaggregated space mission architectures wherein multiple small and large spacecraft work cooperatively as a cluster to achieve mission objectives, such as the cluster in Figure 1. Some of the anticipated advantages of cooperative missions are: more predictable development costs from reuse of existing technology, increased ability to augment functionality during the mission, and lower launch costs by replacing only failed pieces of the cluster.\(^1\) In order to leverage disaggregated architectures and exploit the advantages of small spacecraft, however, technology must be advanced in robust fault management.

Multi-spacecraft cooperative space missions are still relatively new, with only a few missions having been performed successfully, although there is great potential for growth of these types of missions. As such, there are few fault detection, isolation, and recovery (FDIR) methods that take advantage of the additional sensors and information available during cooperative space missions, let alone FDIR architectures designed for a distributed space environment. PRISMA is a recent European Space Agency mission that represents the state of the art in relative navigation for multi-spacecraft missions. It has successfully demonstrated Carrier Phase Differential GPS (CDGPS) navigation filters flying on orbit with large attitude changes and thruster accelerations. However, only the most basic FDIR techniques were implemented. These consisted of transmitting all software input, output, and meticulously enumerated status bytes (indicating anomalies or deficiencies detected during execution) to the ground.\(^2,3\)

Furthermore, while CDGPS is the current state of the art in relative navigation, it is fully dependent on access to GPS signals and receiving hardware.

Figure 1: Artist’s rendering of a multi-spacecraft concept mission that could benefit from enhanced FDIR techniques. Art courtesy of DARPA.

Another recent mission, the Synchronized Position Hold Engage Reorient Experimental Satellites (SPHERES), at the Massachusetts Institute of Technology (MIT) Space Systems Laboratory has flown clusters of autonomous fan powered modules on the International Space Station (ISS).\(^4\) SPHERES implemented basic sensor fault detection by monitoring the residuals in the extended Kalman filter (EKF) processing time of flight data from Ultrasonic receivers.\(^5,6\) SPHERES also implemented thruster fault
detection and isolation by comparing the accelerometer and gyro measurements with the expected values when a maneuver was performed.\textsuperscript{7} 

A survey of recent research does not reveal significant ongoing work on FDIR methods that are specific to clusters and formations. For monolithic spacecraft, federated navigation filters and GPS receiver autonomous integrity monitoring (RAIM) are two well-known FDIR methods. The federated approach to navigation system architecture decentralizes the estimation into multiple local sensors and estimates that are combined into a single estimate by a master.\textsuperscript{8,9} Using this approach, fault detection and isolation are performed by monitoring the accuracy of local estimators, which have a zero-mean distribution under nominal conditions. This approach has been used for FDIR within a specific INS/GPS/Doppler/CNS filter implementation.\textsuperscript{10} RAIM evaluates the parity of GPS pseudorange measurements when more than the minimum of four pseudoranges are available.\textsuperscript{11} This method was extended to CDGPS.\textsuperscript{12} Alternatively, FDIR can be performed by using an array of experts where each expert is an independent navigation system designed to accommodate a specific fault. The tasks of detection, isolation and recovery are accomplished by actively switching to the navigation system with the most accurate estimates. A common implementation of this approach is multiple model adaptive estimation (MMAE).\textsuperscript{13} MMAE has been used for low-cost GPS/INS.\textsuperscript{14} However, the computational requirements of the multiple filters required for this approach are discouraging.

The Separable Architecture for Fault Isolation and Recovery (SAFIR) addresses the issues imposed by FDIR for decentralized systems with many coupled or loosely coupled subsystems such as a cluster of spacecraft implementing a service-oriented architecture and using a hierarchical design. The flexible, service-oriented architecture easily adapts to new and changing mission requirements. Appropriate fault detection solutions can be selected from a library of interchangeable algorithms – the number and types of algorithms used would depend on the application of the FM architecture, and higher-level isolation and recovery services can be easily configured to implement new health information provided by new fault detection services. This promotes easy adaptation to new systems and future missions such as unmanned vehicles, cloud servers, and networks of biomechanical devices.

The hierarchical design of SAFIR addresses the challenge of fusing FDIR results from many different subsystems that may be distributed among a cluster, which is loosely defined as multiple independent modules, or systems of systems, connected via intermittent communications. Such a hierarchy is advantageous even for traditional monolithic satellite missions.\textsuperscript{15} Recent research analyzed several different centralized and decentralized architectures for distributing fault management responsibilities across a cluster of spacecraft.\textsuperscript{16} The architecture defined by SAFIR allows expedient FDIR on a single spacecraft, or any other type of independent module, while promoting robust FDIR solutions across the cluster of modules. The well-defined structure clarifies service roles for engineers new to SAFIR, and it eases the burden of verification and validation for the integrated system.

The rest of the paper is organized as follows. We first introduce the hierarchical and service-based architecture of SAFIR. Then we describe the configuration of SAFIR for guidance, navigation, and control (GN&C) of multiple spacecraft. This example was selected for demonstration because more robust FDIR techniques are crucial for autonomous operation during such multi-spacecraft missions especially during communication interruptions or outages. We also describe the simulation platform used to validate and test SAFIR in this scenario followed by a presentation of the test results. A conclusions section draws closing remarks and expands on future work.

**DECENTRALIZED ARCHITECTURE FOR FDIR**

The SAFIR architecture and design utilize technological concepts that are popular in fault management. In particular, SAFIR is a service-based architecture with a hierarchical design that enables portability, creates modularity, enables flexibility, and simplifies validation. The hierarchical design defines decentralized roles for FDIR services within a cluster composed of systems of systems. The benefits of the service-based architecture and hierarchical design are described below.

**Benefits of Service-Based Architecture**

SAFIR is composed of services with well-defined roles, which are defined by the hierarchy, and a well-defined interface, the system health message. Since their roles are well-defined, individually validating each service is straightforward. The service-based architecture enables adaptation to new and changing mission requirements.

The flexibility of SAFIR comes from the fact that the FDIR algorithms are readily customizable to meet the unique requirements of a mission. While our reference FDIR system employs algorithms designed to detect, isolate and recover from major types of faults that
affect multi-spacecraft operations, additional FDIR algorithms can be added to or removed for future applications. The service-oriented architecture eases portability to new platforms and subsystems by providing flexibility, scalability and robustness. In fact, a library of FDIR algorithms that are already developed is available for reuse as applications for the Core Flight System (CFS).

SAFIR can be expanded with any number of detection algorithms that use the health message as their communications paradigm. The health message is defined using a generic format that is extendable for new systems with different definitions of health; different versions of the health messages maintain compatibility. Fault isolation and recovery services are implemented and customizable to account for new states in the health messages resulting from additional FDIR algorithms and new types of systems.

**Benefits of Hierarchical Design**

SAFIR benefits from a hierarchical design that divides the requirements of the system among components with well-defined interfaces and functionality. This simplifies the testing process and improves human readability of the design. Validation of the integrated system is simplified by limiting communication paths within the hierarchical architecture, which will also be discussed in the next subsection.

The hierarchy is divided into six levels in Table 1 with ground operators acting on top of the other five levels. At the lowest levels are subsystems, which are components that interface with SAFIR but are not part of the SAFIR implementation. The second layer of the hierarchy is composed of system level components, which interface only with subsystems on a single module (typically the host vehicle or spacecraft). The system-level is responsible for managing recovery when consent of the cluster is not required. The third layer is composed of cluster level components, which interface with multiple modules to detect faults at a cluster level. These modules are not assumed to have reliable communication, which means that data may be missing or late. In order to maintain performance as communications links are broken and restored, identical configurations of each cluster-level component can exist on multiple modules. The responsibility of fusing the results of the first three layers, including fault isolation and interpretation of inconsistent results from multiple cluster-level components, belongs to the diagnosis & recovery layers of the hierarchy. In this manner, the system has a distributed architecture. The recovery layer commands automated responses across the cluster when necessary. If an automated response is not available for an evaluated system health state, recovery responsibilities are passed to the ground.

Level 1 through Level 3 is composed of interchangeable services. Subsystems and systems level detection algorithms and cluster level detection algorithms can be selected based on the type of systems and the mission requirements. Diagnosis and recovery are implemented by a dedicated Diagnosis service and a dedicated Recovery service, respectively, which are configurable to handle new aspects of the health message.

The structure of the hierarchical design also contributes to validation of the integrated system. For example, consider deadlocks where multiple services lock up waiting for responses from one another and race conditions when multiple services must perform their operations in the proper sequence to obtain the correct result. Both situations can be avoided by enforcing communication in one direction along the hierarchy. The only exception to this, which must be handled with care in SAFIR, is recovery commands which span from Recovery to subsystem level.

**Table 1: Levels of the Service Hierarchy in SAFIR Starting at the Lowest Level**

<table>
<thead>
<tr>
<th>Level of Hierarchy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Subsystem Level</td>
<td>Monitors diagnosed health and commands recovery actions</td>
</tr>
<tr>
<td>2 System Level</td>
<td>Detection and isolation components monitoring data with a very high guarantee of arrival</td>
</tr>
<tr>
<td>3 Cluster Level</td>
<td>Detection and isolation components monitoring data subject to transmission outages</td>
</tr>
<tr>
<td>4 Diagnosis</td>
<td>Fuses health reports into a single diagnosis of cluster health</td>
</tr>
<tr>
<td>5 Recovery</td>
<td>Monitors diagnosed health and commands recovery actions</td>
</tr>
<tr>
<td>6 Ground Operator</td>
<td>Provides support for health conditions that have ambiguous causes that prevent safe automatic recovery</td>
</tr>
</tbody>
</table>

**SAFIR FOR SPACECRAFT GN&C**

In this section, we describe how SAFIR is configured for FDIR of GN&C in a multi-spacecraft mission so that common faults do not end missions. For example, a guidance fault could result in a collision if spacecraft are working in close proximity. Each individual spacecraft of the cluster has fault probabilities similar to a monolith, which further compounds the probability of a single fault ending the cluster’s mission if it is not immediately addressed. Therefore, spacecraft clusters, including formation flight and proximity operations, without fault management can quickly become impractical. Automatic FDIR reduces the effort
required by the ground crew when faults occur, reduces the chance of collision by quickly recovering from faults, and makes clusters more practical for missions.

The implementation of SAFIR for spacecraft GN&C includes FDIR algorithms, deployed on each spacecraft, that employ common sensors to detect faults. The implementation develops and configures the five levels of the SAFIR hierarchy as shown in Figure 2: Navigation Monitor performs local navigation FDIR; Thrust Monitor performs actuation FDIR; Cluster Monitor implements unique algorithms for cluster navigation FDIR which is robust to communication interruptions or outages; Diagnosis provides this robustness by fusing health information from the monitors and performs fault isolation; and Recovery commands actions within the system when health information changes.

Definition of Subsystem Level Components
As the target of this demonstration is FDIR for spacecraft GN&C, the subsystem components are spacecraft sensors and actuators. The scope of the demonstration is limited to 3 degrees-of-freedom in translational dynamics only. The spacecraft are assumed to have GPS, inter-spacecraft ranging, accelerometers, and a single thruster. Thruster pointing is assumed to be handled by a fully independent attitude dynamics and control system.

Figure 2: Diagram of SAFIR configuration for GN&C FDIR for a cluster of four spacecraft. The diagram shows selection of FDIR algorithms at the System Level and Cluster Level, configuration of Diagnosis and Recovery, and definition of the health message interface.

We leveraged previous work on the development and test of the System F6 Cluster Flight Application (CFA) whenever possible. The CFA is a prototype of flight software for guidance and control of a cluster of satellites.17 We integrated SAFIR with the CFA to realistically demonstrate the effectiveness of recovery actions and demonstrate the response of SAFIR after the recovery action is taken. Therefore, CFA GN&C functionality is treated as a subsystem-level component.

Development of Module Level Fault Detection

Navigation Monitor
Navigation Monitor implements two fault detection algorithms: Receiver Autonomous Integrity Monitoring (RAIM) of GPS pseudorange measurements and limit checking to monitor whether CFA navigation service has switched to using relative range measurements instead of GPS.

RAIM is a well-known algorithm for GPS solution integrity monitoring that detects pseudorange measurements that are statistical outliers.11 It uses the estimate of the position from the overdetermined system to find the difference between the measured pseudoranges and the expected pseudorange values based on the estimated position. This difference is passed through a parity transformation to create a parity vector that is used to determine the presence of an error.
The parity transformation can also isolate the faulty GPS pseudorange measurement. The RAIM algorithm is executed onboard the spacecraft receiving GPS measurements whenever more GPS satellites are visible than is needed for constructing a single position estimate. If a fault is detected, the algorithm attempts to isolate the pseudorange that is the outlier.

With the second method, Navigation Monitor detects if the spacecraft has permanently switched to relative range measurements by monitoring CFA telemetry. If the measurement source changes to relative ranges for a specified interval of time without returning to GPS, then it declares a fault.

Navigation Monitor performance was evaluated with a high fidelity GPS receiver simulation. It is the same simulation that was used for development of CFA application for System F6. The receiver model includes ionosphere errors, troposphere errors, tracking loop errors, and oscillator errors. These error sources have not caused significant issues.

In simulations that included GPS faults of constant bias and sinusoidal errors, the implementation of RAIM here was able to correctly detect 99.8% of the faults and was able to correctly isolate the fault source 97.9% of the time. The detection threshold is selected for 0.1% probability of false alarm based on the theoretical distribution of the test statistic. To verify that the distribution of the test statistic matches the chi-squared distribution, the value of the test statistic when exactly 10 GPS space vehicles (SVs) were visible is compared to the theoretical distribution in Figure 3. This, however, is for a simulation that neglected the ionospheric effects. The ionosphere causes a bias that skews the distribution away from the ideal chi-squared distribution. To further verify the false alarm rate, a simulation with just over 1,740,000 data points (300 orbits with GPS data received every second) was run and no false alarms were detected.

Figure 3: Comparison of actual distribution of the test statistic and the theoretical chi-squared distribution with six degrees of freedom for 10 SVs.

**Thrust Monitor**

Thrust Monitor detects faults in the onboard thrusters. Its algorithm uses a probabilistic approach to estimate how likely it is that a fault is present with the thrusters, which was first introduced by Wilson et al. It also implements a fault isolation scheme that estimates the likelihood of a specific fault mode being the cause of the fault. To determine if there is a fault, it compares measurements from the accelerometer to the expected acceleration caused by the nominal command thrust. It also evaluates fault modes, such as loss of effectiveness (LOE) or stuck thruster, by comparing the acceleration measurements to the accelerations expected in those fault modes. Once a fault is detected, it isolates the most likely fault mode in the list of evaluated fault modes, and it declares the fault in a health message.

The performance of Thrust Monitor was evaluated using our high fidelity dynamics simulation that includes multiple noise/error sources for the burn. Noise is assigned to the thruster on-time, the thruster off-time, the magnitude of the thrust, and the direction of the thrust. We explored the sensitivity of the detection algorithm to these noise sources. Errors in the start and stop time of the burn were most problematic, which prompted additional features to avoid false alarms when the timing is not perfect. Thrust Monitor is preconfigured with numerous LOE levels that may occur during the mission. We explored its ability to detect these LOE levels as well as its sensitivity to LOE levels that were not preconfigured, and we also added the capability to detect a stuck thruster that is firing when it should be inactive.

Several case studies were run through Thrust Monitor to determine the detection time for recognizing that a fault has occurred, to determine the isolation time for identifying the most likely fault mode, and to determine the delay for clearing a fault that is resolved. Three different window sizes were studied along with six different failure modes. The magnitudes of the nominal accelerations ranged from 0.029 m/s^2 to 3.53 m/s^2, and the duration of the thrust ranged from 0.1 seconds to 9.1 seconds. The noise on the accelerometer measurements is set to be zero mean Gaussian with a standard deviation of 0.02 m/s^2.

Figure 4 demonstrates Thrust Monitor detecting a fault during a maneuver that is 9 seconds long. Before the maneuver begins, the values of the likelihood ratio in the plots on the right are $10^0 = 1$ because it is equally likely that there is and is not a fault when the thruster is off as there is insufficient information to decide either way. A fault is declared the first instance that a likelihood ratio drops below the detection threshold marked by the thick dashed line in the upper plot, and
the two inactive fault modes are excluded when both the blue and red plots in the lower half exceed their threshold as well. This leads to the fault mode being identified as having a 75% LOE. At the end of the plot, even though all the likelihood ratios have risen above the lower threshold, the fault is not cleared because all likelihood ratios would have to exceed the upper threshold to clear the fault. The plots on the left show a nominal thrust as all likelihood ratios rise above the upper threshold, indicating that the thruster is performing at 100% effectiveness. If thruster faults were previously declared, they would be removed starting at the time all likelihood ratios are above the threshold.

With a window size of 0.50 seconds (50 measurements in each window), the thrust monitor took an average of 0.105 seconds to detect the presence of a fault and 0.170 seconds to isolate the fault when the fault exactly matched a preconfigured LOE. When the fault LOE was 0.05 off of the closest preconfigured LOE level, detection of a fault took an average of 0.047 seconds and 0.104 seconds to isolate the fault. It missed the detection of one faulty thrust because the window size is closer to the duration of this short thrust. Decreasing the window duration to 0.10 seconds (10 measurements in each window) changed the above results to an average of 0.040 seconds to detect the presence of a fault and 0.088 seconds to isolate the fault when the fault exactly matched a preconfigured LOE level. When the fault LOE was 0.05 off of the preconfigured LOE level, detection of a fault took an average of 0.047 seconds and 0.104 seconds to isolate the fault. It missed the detection of one faulty thrust because the window size is closer to the duration of this short thrust. Decreasing the window duration to 0.05 seconds (5 measurements in each window) led to further reductions in the detection time. However, the small window size made the algorithm unable to isolate half of the fault modes for the detected faults.

**Development of Cluster Level Fault Detection**

Fault detection algorithms that utilize data from multiple systems across an unreliable data channel should be implemented at the cluster level. For our multi-spacecraft GN&C problem, two cluster-level fault detection algorithms were implemented in Cluster Monitor that utilize the parity between GPS measurements and range measurements, as shown in Figure 5, to detect faults. They are Filter/Range Parity (FLT-RP) and Receiver Autonomous Integrity
Monitoring Augmented with Relative Range Measurements (RAIM-RELNG).

GPS Filter Solution

Range Measurement

Figure 5: GPS can be used alone for satellite navigation. Clustered spacecraft frequently have range data available as well, which can provide additional redundancy without adding size, weight, and power.

**FLT-RP in Cluster Monitor**

The FLT-RP algorithm uses the relative range measurements to verify the navigation estimate by comparing the measured relative ranges of the spacecraft with the expected ranges based on the estimated positions from the navigation filter. If a fault is detected, then the algorithm tries to find the source of the error. The source could be either with the navigation filter estimate, the GPS data, or the relative range measurements.

The FLT-RP algorithm first needs to ensure that the timestamps of the navigation filter estimates correspond to the same timestamps as the relative range measurements. If this is not the case, then the filter estimates are propagated to align the times. Next the relative ranges are calculated from the propagated states for each spacecraft pair that has a range measurement. The range residuals are then found from the difference between the range measurements by the two end points. These residuals are weighted by their variance and the sum of the squared weighted residuals for the test statistic for the $\chi^2$ test. The degrees of freedom used for the $\chi^2$ test is the number of relative range measurements used in calculating the residuals. If the test statistic exceeds the threshold corresponding to the selected confidence level, a fault is declared. Otherwise no fault is declared, and any previous fault is cleared. The algorithm attempts to isolate the spacecraft responsible for the fault by removing the residuals corresponding to each spacecraft and checking if the new test statistic for the $\chi^2$ test falls below the fault threshold.

**RAIM-RELNG in Cluster Monitor**

RAIM-RELNG operates when there are more than the minimum of four GPS range measurements, like the RAIM algorithm does, and augments RAIM by including the GPS ranges for all the spacecraft in the cluster and relative ranges between the spacecraft. Including these extra measurements makes the algorithm more sensitive to faults that the RAIM algorithm for individual spacecraft may not be able to detect.

RAIM-RELNG is similar in behavior to RAIM, with the main differences being that the relative ranges are included in the algorithm and that more than one spacecraft's measurements are considered. It operates on the clustered spacecraft simultaneously instead of an individual spacecraft at a time. To begin RAIM-RELNG, the GPS ephemerides must be known and $n$ GPS pseudorange measurements must be obtained for more than four visible satellites for each spacecraft. Then, $r$ relative ranges are obtained between the $m$ spacecraft in the cluster. The current position and clock bias states of the spacecraft are then calculated from this information using a standard, iterative method of solving position from multiple GPS pseudoranges observed at the same time. The system is then linearized around the position estimate so the pseudorange residuals can be calculated using the following equation:

$$\mathbf{y} = \mathbf{Hx} + \mathbf{\epsilon}$$  \hspace{1cm} (1)

where $\mathbf{x}$ is the $4m \times 1$ vector of the perturbations from the position estimate of the three position and one clock states for each spacecraft, $\mathbf{H}$ is the $(n+r) \times (4m)$ linearized output matrix for each of the visible satellites, $\mathbf{\epsilon}$ is the $(4m) \times 1$ noise vector, assumed to be zero mean Gaussian, with variance $\sigma^2$, and $\mathbf{y}$ is the $(n+r) \times 1$ vector of difference between the measured GPS pseudoranges and relative ranges and the values obtained from the above equation. The $(n+r-4m) \times (n+r)$ parity matrix, named $\mathbf{P}$, is obtained by performing QR decomposition on $\mathbf{H}$ and discarding the first four columns of the resulting orthogonal matrix and taking the transpose of that $(n+r) \times (n+r-4m)$ matrix. The parity matrix is used to generate the $(n+r-4m) \times 1$ parity vector $\mathbf{p} = \mathbf{Py}$. The norm of
the parity vector is used in the test statistic as it approximates a chi-squared distribution with \( n + r - 4m \) degrees of freedom, \( \frac{P^2}{n} \sim \chi^2_{n+r-4m} \), during nominal operation without faults. If the test statistic exceeds the threshold corresponding to the selected confidence level, a fault is declared. If the test statistic does not exceed the threshold for the specified confidence level, no fault is declared, and any previous fault is cleared.

**Configuration of Diagnosis**

Diagnosis processes the health state provided by the module level and cluster level fault detection algorithms to create a combined health state for the cluster. It is implemented using two scripts programmed in the Lua language. The first is a library of functions that can be called to perform data fusion on multiple health messages, and this must be provided prior to the start of the simulation. The second script, called the mission script, contains the implementation of Diagnosis for the specific mission, which can be reconfigured at any point during the mission. It can call on the methods provided in the library, or it can even implement new fusion methods.

The library contains a set of methods for performing common health fusion tasks that isolate faults based on the health reports collected from the subsystem, system, and cluster levels. The rules are organized into two groups: voting rules and conjecture rules. Voting rules resolve conflicts among conflicting sources of health. Conjecture rules isolate faults from sources of health information that have intersecting detection domains. Figure 6 briefly demonstrates the configurable application of rules at a high level.

![Figure 6: A high level example of configuring rules in diagnosis. The solid lines represent one application of voting and conjecture that uses majority rules voting. The dotted lines represent an application that uses quorum voting.](image)

The mission script is easily configurable for new missions and module types. Diagnosis for cluster GN&C is configured to accept the health states provided by each monitor on each cluster member to generate an estimate of the health of the entire cluster. The configuration first combines the health data from the different algorithms onboard each spacecraft. If there are conflicts in the health data, then it resolves them by holding a vote between the conflicting data. The resolved health state is then returned so that recovery actions can take place based on the output diagnosis.

**Configuration of Recovery**

The recovery algorithm consists of a recovery action table and configurable state machines for command tracking. The action table is a list of actions that shall be taken when specific changes to the health diagnosis occur. These actions may trigger state transitions in the series of state machines. Some transitions have a recovery command associated with them, but it is not required. Figure 7 and Table 2 compose an abstract example of an action table and the corresponding state machine, respectively.

![Figure 7: Pictorial diagram of the state machine recovery algorithm implemented based on the example actions in Table 2.](image)

**Table 2: Example of a Recovery Action Table**

<table>
<thead>
<tr>
<th>ID</th>
<th>Health Change Trigger</th>
<th>Transition</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subsystem fault appears</td>
<td>Fault</td>
<td>“Recovery command”</td>
</tr>
<tr>
<td>2</td>
<td>Subsystem fault resolved</td>
<td>Repair</td>
<td>No command</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

A table of actions triggers state transitions in one of the configurable state machines. When the fault occurs, a recovery command is executed to modify the cluster’s behavior. The state machine prevents multiple recovery commands until the faulty subsystem is observed as repaired in the health. In the example, a change in cluster behavior is not necessary after the repair is observed. If no transition in the recovery table is met, or if more than one transition in the recovery table is met, then a request is sent to the ground so that the ground can further handle the situation.

A survey of possible recovery actions and a study including fault tree analysis determined the recovery actions for various faults. The chosen recovery actions reduce the probability of mission failure using the
distributed architecture of SAFIR. Furthermore, they demonstrate both a navigation recovery action and a guidance recovery action. Simple recovery actions for faults with a high probability of occurrence were preferable. The recovery actions in Table 3 were selected for demonstration.

Table 3: Recovery Actions for Implementation and the Fault Event to Results in the Action

<table>
<thead>
<tr>
<th>Fault</th>
<th>Recovery Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thruster total LOE for one spacecraft</td>
<td>Configure CFA for leader-follower guidance using the failed spacecraft as the leader</td>
</tr>
<tr>
<td>CFA changes to range measurements for one spacecraft</td>
<td>Configure CFA navigation filters to support permanent use of range measurements only without GPS measurements</td>
</tr>
<tr>
<td>Failure of measurement source (GPS or ranging)</td>
<td>Disable CFA navigation filters using the measurements that are isolated as faulty</td>
</tr>
<tr>
<td>Stuck thruster</td>
<td>Reset CFA navigation filters after the thruster expends all of its fuel (because the navigation filters consider acceleration from commanded burns)</td>
</tr>
<tr>
<td>Multiple spacecraft with Thruster total LOE</td>
<td>Allow natural egress of failed spacecraft, or force egress of the failed spacecraft by maneuvering healthy spacecraft to a safe orbit</td>
</tr>
</tbody>
</table>

PLATFORM FOR VALIDATION AND TESTING

Demonstrations of SAFIR running on four single board computers (SBCs) recently concluded. The demonstration platform is diagramed in Figure 8 and pictured in Figure 9. SAFIR was validated on three HummingBoard i2’s and one Raspberry Pi 2. All four boards run Debian Linux Jessie. Each board has WiFi, which represents the network bus for inter-spacecraft communication. Ethernet connects the boards to a simulation of the spacecraft bus using a consumer network switch. NASA’s Trick simulation environment is set up with NASA’s JEOD (JSC Engineering Orbital Dynamics) for dynamics and EDGE (Engineering Dynamic On-board Ubiquitous Graphics (DOUG) for Exploration) for live visualizations. The cluster of four spacecraft is simulated on a single laptop using high fidelity dynamic models. The SAFIR components are deployed as cFS applications for the demonstration, and the CFA developed for System F6 represents the guidance, navigation, and control flight software.

The CFA is deployed to represent flight software for spacecraft GN&C. SAFIR integrates with the CFA to realistically demonstrate the effectiveness of recovery actions and demonstrate the response of SAFIR after the recovery action is taken. However, SAFIR only uses CFA public interfaces. CFA be interchanged to any GN&C software that uses similar interfaces.

cFS is a software environment that has been used on previous space flights. It contains a message-oriented middleware, but it needs an additional application to extend the message bus over a network of distributed devices. cFS includes the Operating System Abstraction Layer (OSAL) for the host system, which interfaces cFS to the underlying operating system, such as Linux. This flexibility allows us to prototype and demonstrate on a well-known platform (Linux) that still has software interfaces that are similar to a true spacecraft.

Figure 8: Three HummingBoards and one Raspberry Pi 2 are connected by Ethernet to a laptop running dynamics, sensor, and actuator simulations in Trick.
Each SAFIR component is implemented as a cFS app. The apps are run across five deployments of two types across the four SBCs. The first type is deployed four times, one instance on each SBC, because it contains apps that need an instance deployed on every spacecraft. The second type is deployed once on a single SBC. It contains apps that only have a single instance across the cluster. The CFA apps for cFS are also included in these deployments.

RESULTS FOR SPACECRAFT GN&C

Validation testing culminated in a final demonstration of a design reference mission scenario that demonstrates SAFIR’s capabilities. Shorter, individual test cases verify fault detection and recovery for scenarios that could not be included in the reference mission. Two such test cases are discussed herein.

Validation of Design Reference Mission

The design reference mission demonstrates and tests FDIR actions for SAFIR for multi-spacecraft missions. The mission events and the planned fault simulations are selected to provide a range of cluster actions and demonstrate specific FDIR operations performed by the spacecraft and the cluster. The mission is condensed to 18.5 hours for demonstration purposes, but the periods of cluster maintenance could be expanded to accommodate a longer mission timeline. The shortened timeline facilitates demonstration and shortens testing time for the agile development process.

Figure 10 summarizes the cluster formation actions and the number of elapsed orbits when each action is taken. The scenario depicts ingress of four modules to form a more tightly packed cluster. During the scenario, multiple faults occur. The time periods with faults are highlighted red in Figure 10, and actions taken by the cluster timed at the black markers. The timeline of the mission is designed so the faults that occur have significant influence on the system so that both fault and recovery actions will be more dramatic. (Note that for a non-simulated mission in space, it would be advisable to simulate possible faults at safer points in the timeline.)

There is a communications ranging error while the spacecraft are performing the cluster ingress. After the cluster station keeping is established, there is a simulated pseudorange bias which gradually ramps up in magnitude on one GPS SV until the SV is no longer visible by the cluster. The cluster then reconfigures, requiring Module 1 to maneuver to a different set of relative orbital elements. Another reconfigure occurs, requiring Module 4 to maneuver to a different set of relative orbital elements. However, before Module 4 ...
maneuvers, it suffers a simulated loss of effectiveness in its thruster, which lasts for the entire maneuver. Once this fault is detected, the cluster reforms by commanding CFA orbit maintenance to set Module 4 the leader in a leader-follower control scheme. Finally, the GPS receiver on Module 2 fails permanently, which leads to reconfiguration of Navigation.

A video demonstration of this scenario was created, and five frames captures from the video are presented in Figure 11 through Figure 15, which are described chronologically in Table 4. The frame split horizontally to demonstrate two different runs of the scenario. Both runs experience a thruster failure on spacecraft four (cyan) during the scenario. The top run does not have a recovery action commanded by SAFIR, which results in the failed spacecraft drifting away. The bottom run has a recovery action command by SAFIR, which automatically prevents the cluster from drifting apart.

Testing of the design reference mission was successful.

Figure 10: Timeline of mission actions and faults during the design reference mission.

Table 4: Chronological Summary of Video Frames Captured from the Design Reference Mission Described in Figure 10

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 11</td>
<td>Checkout of the cluster with large inter-module distances. This and the subsequent four figures contain two runs, top and bottom. Both runs experience a thruster failure during the scenario. The top run does not have a recovery action commanded by SAFIR, which results in the failed module drifting away. The bottom run has a recovery action command by SAFIR, which automatically prevents the cluster from drifting apart.</td>
</tr>
<tr>
<td>Figure 12</td>
<td>A frame captured after all the spacecraft finish ingress to create a pair of two sub-clusters.</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Reconfiguration of the fourth spacecraft (cyan) starts immediately before the thruster fault occurs.</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Immediately after the thruster fault occurs during ingress of the fourth spacecraft (cyan). In the bottom run, the leader change has already been commanded, and the three healthy spacecraft (yellow, red, and green) have already started following the failed spacecraft.</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Several orbits after thruster failure. In the top case, the failed spacecraft (cyan) is drifting away from the cluster. In the bottom case, the healthy spacecraft have followed the failed spacecraft.</td>
</tr>
</tbody>
</table>
Figure 11: Checkout of the cluster with large inter-module distances. This and the subsequent four figures contain two runs, top and bottom. Both runs experience a thruster failure during the scenario. The top run does not have a recovery action commanded by SAFIR, which results in the failed module drifting away. The bottom run has a recovery action command by SAFIR, which automatically prevents the cluster from drifting apart.

Figure 12: A frame captured after all the spacecraft finish ingress to create a pair of two sub-clusters.
Figure 13: Reconfiguration of the fourth spacecraft (cyan) starts immediately before the thruster fault occurs.

Figure 14: Immediately after the thruster fault occurs during ingress of the fourth spacecraft (cyan). In the bottom run, the leader change has already been commanded, and the three healthy spacecraft (yellow, red, and green) have already started following the failed spacecraft.
Recovery from a stuck thruster

CFA navigation filters process maneuver commands in the navigation filters. Therefore, a stuck thruster (or missed thrust) will cause the navigation solution in the filter to deviate from the true state. This is demonstrated in Figure 16 (left) for a stuck thruster. (The blue lines are the true navigation error. The black lines are the estimated navigation covariance.) The navigation filter does correct itself naturally over 0.3 orbits, but Figure 16 (right) has a recovery command that causes a correction almost immediately after the command is sent in Figure 16 (right). The command artificially inflates the estimated covariance in the filter, which allows the filter to converge to another solution quickly.

Recovery for permanent navigation using relative measurements

Relative navigation filters in CFA navigation filters rely on a primary filter to anchor their relative solution in the absolute space. If the primary filter solution becomes invalid, then the relative navigation filter will ignore the solution and lose its anchor in relative space. The difference can be seen between Figure 17 (left) without an anchor and Figure 17 (right) with a recovery command to restore a proper anchor.

CONCLUSIONS AND FUTURE WORK

The SAFIR technology is technically feasible as proven by successful testing, some of which is reported in this paper. Although its architecture is shown to be applicable to any system of systems, this paper proves its application to GN&C FDIR for a cluster of spacecraft by developing fault detection algorithms on the Module Level and Cluster Level, configuring Diagnosis for fault isolation, and configuring Recovery to take action in response to faults. The implementation was validated and demonstrated successfully.

It is important to note that while SAFIR was demonstrated for spacecraft clusters, it is applicable to a wide range of possible missions, even those involving fleets of autonomous land, air, or sea vehicles. In fact, most aspects of GN&C FDIR readily apply to other vehicle types. Such clusters of terrestrial vehicles are currently under development. These missions have the same needs for robust and reliable relative navigation as clusters and formations of spacecraft. Most importantly, our FM architecture is applicable to unmanned aerial systems (UAS), which broadens the potential customer base of this innovation. Fleets of autonomous aquatic vehicles are another potential application of SAFIR. Ultimately, SAFIR technology could adapt to FDIR in any system of systems such as cloud servers and networks of biomechanical devices.

ACKNOWLEDGMENTS

The authors thank Richard Alena for his valuable insight and guidance. This work was funded under NASA Contract No. NNX14CA06C through NASA Ames Research Center. We also thank Dr. Sun Hur-Diaz and Brendan O’Connor for their input throughout this research.
Figure 16: Navigation during a stuck thrust without any recovery action (left) and with a recovery action (right). The blue lines are the true navigation error. The black lines are the estimated navigation covariance.

Figure 17: Relative navigation without being anchored to an absolute navigation solution (left), and relative navigation properly anchored to an absolute navigation solution (right). The true navigation error is in green, and the estimated covariance of the navigation error is in black. Note the different y-axis scales.
REFERENCES


