TRIAKIS CORPORATION

Empirical Assurance of Embedded Software Using Realistic Simulated Failure Modes

Phase I Final Report

for the

NASA Office of Safety and Mission Assurance Software Assurance Research Program

Center Initiative 2004-77

Ted Bennett, Principal Investigator
Triakis Corporation
Ted.Bennett(at)Triakis.com

Paul Wennberg, Co-Investigator
Triakis Corporation
Paul.Wennberg(at)Triakis.com

Ken Chen, Co-Investigator
Johnson Space Center
Safety & Mission Assurance
ken.k.chen(at)nasa.gov

Table of Contents

Table of Tables

Table of Figures

Abstract

Reliance on analysis alone to evaluate the effect of failure modes untestable in an integration lab setup will not fully ameliorate the added risks due to ever-increasing system complexities and the use of intelligent behavior in software designs. Few would argue that evaluating system behavior through dynamic testing in its native target environment is the best method of verification, however, this is almost always infeasible for NASA space hardware. In this study we simulated the NASA Mini-AERCam nanosatellite, running its uninstrumented flight software, in its orbital environment. We then demonstrated how a high-fidelity virtual environment simulation can be used for empirical assessment of system and software behavior in response to enhanced system and component failure conditions. Failure-mode testing of the Mini-AERCam controlled by its flight software revealed dynamics and behaviors not discovered in prior failure modes and effects analysis efforts. These results suggest improved mitigation strategies for recovery of the Mini-AERCam experiencing such a failure, and design modifications with the potential for improving controllability in the presence of such a failure.

1.0 INTRODUCTION

Failure Modes and Effects Analysis (FMEA) is a primary tool used by NASA Safety & Mission Assurance (SMA) teams to evaluate the behavior of complex systems in response to operational failures, and to develop operational strategies for mitigating the risk of undesirable consequences of failure conditions.

This document has been developed in support of a Center Initiative research project funded by the NASA Software Assurance Research Program (SARP) beginning in fiscal year 2004. We have created a virtual system integration laboratory (VSIL) simulation of the Johnson Space Center Mini-AERCam (MAC) nanosatellite from project documentation to explore the concept of failure mode effects (FME) testing.

The Mini-AERCam VSIL supports execution and test of the flight software without the need for external test equipment or other hardware. As such, it provides an ideal tool for software quality assurance (SQA) groups to improve their effectiveness by conducting software testing rather than relying exclusively on analysis tools and process monitoring. This domain has historically been inaccessible to them due to the high cost of procuring their own system integration lab (SIL), and limited availability of project-level SILs for their use.

With the opening of the flight software (FSW) testing domain to SQA teams using a VSIL, some of the unique capabilities of this virtual environment may be applied to ease, complement, or supplant laborious analysis activities like failure modes & effects analysis (FMEA). Failure modes may be modeled as intrinsic characteristics of parts within the virtual environment that can be manifested as desired during the execution of test scenarios. We refer to these as “enhanced” failure modes because they are generally difficult or cost-prohibitive to produce in a hardware SIL, but readily achievable in a virtual environment simulator.

Invoking failures in simulated parts during test most closely mimics how the FSW would encounter such an event in an operational setting. Observing/measuring the behavior of the FSW and system in response to a failure condition is the surest way to verify conformance to a requirement. This ability confers less need for reliance on analysis, and more flexibility and realism in the evaluation of the effects of failure modes. For this effort, our selection of failure modes was drawn from the list of those analyzed as part of the Safety and Mission Assurance (SMA) team's Mini-AERCam FMEA activity. Despite advances in analysis methodologies, reliance on analysis alone to evaluate the effect of failure modes untestable in an integration lab setup will not fully ameliorate the risk added due to increased complexity. Supplementing analysis with empirical verification of the system’s response to anomalous conditions (initiated by ionizing radiation, component failures, system failures, et al) may be the best approach to realizing a high level of assurance for this project.

The research effort described in this report was conducted under a grant from the NASA Office of Safety and Mission Assurance – Software Assurance Research Program. Co-Investigator Paul Wennberg developed the simulation tool used in this investigation.

2.0 BACKGROUND

2.1 Failure Modes & Effects Analysis

Failure Modes & Effects Analysis has a long, successful history dating back to the development of United States Military Procedure MIL-P-1629, a few years following the end of World War II. The first major NASA use of FMEA was on the Apollo program in the early 1960s and following publication by several professional societies of formal procedures for its use, FMEA began to see wide-spread adoption throughout the commercial sector in the 1970s.

FMEA was originally developed to help improve the reliability of hardware systems and soon evolved to incorporate prioritization of failure modes according to criticality (FMECA). With the increasing reliance on computers to control hardware systems, it became apparent that this technique could be advantageously applied to the analysis of the effects of software failure modes (SFMEA). Haapanen & Helminen provide a recent overview of FMEA application to software systems in their 2002 paper for the Finnish Radiation and Nuclear Safety Authority [1].

2.2 Defect and Hazard Detection

New complex embedded systems are quick to take advantage of the unrelenting pace of advancement in computer hardware performance and capacity. This increase in hardware capability comes with a considerably greater increase in the functionality and complexity of the software in control. The development of methods and tools used to create, verify, and validate controlling software have been hard pressed to keep up with the unremitting advances.

This software complexity growth produces an exponential increase in the software state space. As a result, we have long since passed the point where the total software state space can be economically verified, and rely on methods of prioritizing states and failure modes to investigate according to criticality, probability, and risk. One of the promising methodologies developed at JPL by Feather et al is called Defect Detection and Prevention (DDP).

DDP is a tool that helps guide the decision-making process early in the development process. In the words of Feather et al., “This is an important but challenging time of the life cycle. It is important because these early decisions have the most leverage to influence the development to follow. It is challenging because information on which to base those decisions is incomplete and uncertain, and in the case of advanced technologies and systems there is little past experience from which to extrapolate” [2]. Establishing the correctness of the system design early in a project not only improves software, safety, and mission assurance, but also can have a substantial impact on development and life-cycle costs (see: *Cost of Defect Propagation*).

Use of a tool such as DDP early in a project can be a highly effective means of establishing a prioritized list of failure modes to be manifested for testing a system's response in support of an FMEA exercise. Testing the system in a VSIL produces data that informs the development of mitigation strategies to deal with operational failures.

3.0 METHODOLOGY

Our research was dependent upon the use of a simulation of the MAC in its orbital environment with sufficient fidelity to run the FSW object code, with part models closely mirroring those of MAC and having failure modes that may be invoked under test control. Since no simulator with these capabilities existed, a major part of our effort was spent developing one using Triakis' IcoSim simulator application. Experience has shown that the most efficient method of creating an embedded system simulation capable of running the executable software object is to first develop one based on a pseudo-processor running the source code (if available), compiled at the time the simulator is built. This has proven an effective way to develop and verify the simulator to a reasonable degree of confidence for several reasons:

1. The software has been designed to run on the target hardware and as such, it contains all of the information required to model the software/hardware interface (e. g. registers, timers, interrupts, memory map, etc.).
2. Compiling the source code along with the simulator allows the use of standard software development tools like source-level debugging, that facilitate the investigation of simulator problems due to incompatibility with the software.
3. Working with the source code promotes a good understanding of the software and system design.

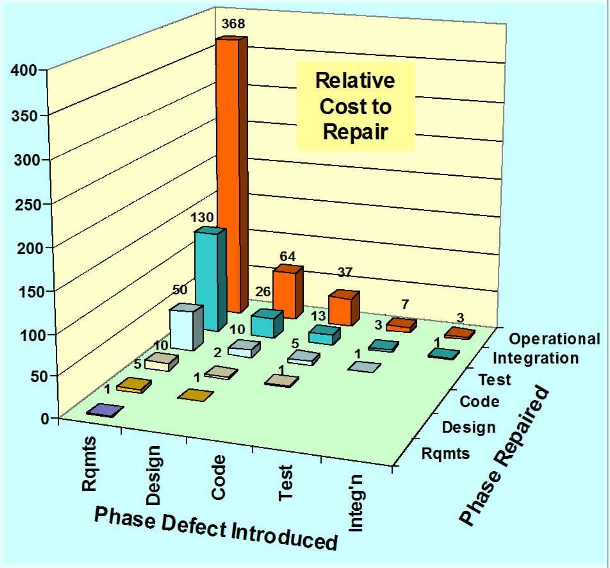
Cost of Defect Propagation

Estimates of the cost to find and correct software faults at each of the principal stages of a project have been publicized and widely referenced since 1976 when Boehm first published his study [3] on the subject. Cost numbers vary depending on the type of application for which the software is being developed but the common thread they all exhibit is the substantial increase in project costs caused by carrying problems from one development stage to the next.

A report released in May 2002 by the National Institute of Standards & Technology (NIST) [4] contains a thorough analysis concluding that inadequate software testing costs the United States an estimated *$59.5 billion annually*. The 309-page NIST report is a well-considered treatise on the economic impact of inadequate software testing.

While these numbers are extrapolated from software developed for the financial services and transportation applications (CAD, CAM, etc.) sectors, the message applies even more significantly to industries engaged in developing software for safety and mission critical applications such as aerospace, medical, defense, automotive, etc. Failures of safety/mission-critical software may result in harm to, or loss of human life and/or mission objectives such as in the case of the Therac-25 radiation overdose accidents [5] and the Ariane-5 maiden launch failure [6]. The Therac-25 software caused severe radiation burns in numerous cancer patients before it was implicated. The cost of allowing the Ariane-5 software defect to pass into the operational phase has been estimated to be as high as $5 billion alone.

NASA recently sponsored a study to evaluate the economic benefit of conducting Independent Validation & Verification (IV&V) during the development of safety-critical embedded systems [7]. This study presented cost-to-repair figures focused specifically on embedded systems projects. The chart at the right shows the relative cost to repair factors – considered to be conservative estimates for embedded systems – used in this study.



This graph tells us that an error introduced in the requirements phase will cost five times more to correct in the design phase than in the phase in which it was introduced. Correspondingly, it will cost ten times more to repair in the code phase, 50 times more in the test phase, 130 times more in the integration phase, and 368 times more when repaired during the operational phase. The graph also gives the cost multipliers for problems introduced in the design, code, test, and integration phases of the development cycle.

Having developed the MAC simulator, we then extended it to include a shuttle orbiter virtual environment (VE) simulation within which the virtual Mini-AERCam operates during software testing. This VE includes GPS satellites, the shuttle orbiting the Earth orbiting the Sun, orbital dynamics, inertial effects, etc. We then added failure modes to selected parts, and developed tests based on selected failure modes analyzed in the MAC FMEA document.

3.1 Enhanced Failure Modeling

We define "Enhanced Failures" as those that are generally difficult or cost prohibitive to produce in a realistic manner, in a system integration laboratory facility – but are readily created in a virtual system integration laboratory (VSIL) environment simulator. The following types of verification are typically very difficult to realistically evaluate in a physical laboratory environment, but relatively easy in a high-fidelity virtual environment:

1. Verifying the response of controlling software to the effects of single or burst event ionizing radiation on one or more hardware components (e.g. FPGA, RAM, CPU internal bit/state changes, latch-up, etc.).
2. Verifying the software response to single or combined hardware component failures/degradations during critical software task execution (e.g. FPGA internal failures, D/A & A/D bit failures, telemetry & control link noise interference or component failures, sensor failures, propulsion system component failures, loss of inertial reference axis, etc.).
3. Verifying the software response to single or multiple system-level component failures, degradations, or unexpected behaviors during critical software task execution (e.g. loss/degradation of single/multiple GPS satellite signal(s), loss of command link during manual control, etc.).

The failures we modeled in the MAC were of the type identified in item 2 above, and selected among the failures analyzed in the MAC FMEA report (filename: "FMEA Mini AERCam Demonstrator.doc"). We selected the following failure mode categories to implement for our project:

1. Internal propulsion leak venting through seam in Mini-AERCam case
2. Thruster stuck open or closed
3. MEMS rate gyro total failure
4. MEMS rate gyro axis gain & bias offset failure
5. Isolation valve unresponsive

In the VSIL simulator, we implemented failure modes at the part level as functions within the part that may be called under test control to alter part behavior (e.g. manifest failures) when desired. Parameters are typically passed to the function to control activation/deactivation of the failure, to select different characteristics of the failure, to control the severity of degraded mode failures, etc. For example the following 'C++' function within the MEMS rate gyro part is used to manifest rate gyro failures:

```
void MEMS::SetFailureMode(int axis, double offset, double gain)
{
  error[axis].offset  = offset;
  error[axis].gain    = gain;

  if((gain == 0) && (offset == 0)) // This forces the MEMS to be unresponsive
    MEMS_INOP = TRUE;
}
```

This is the code that implements the gain and offset bias errors for the rate gyro Roll Axis (the value of parameter 'ffrate[ROLL]' is driven by the simulator inertial model):

```
rg[ROLL].rate = ffrate[ROLL] * error[ROLL].gain + error[ROLL].offset;
```

When the MEMS_INOP flag is set, the rate gyro outputs to the FPGA that reads its values are disabled. The complementary function, "MEMS::ClearFailureModes()," is called to reset the gain bias to 1 and offset bias to 0 for all axis', and to reset the MEMS_INOP flag to FALSE, thereby restoring nominal operation to the rate gyro.

Similarly, "SetFailureMode()" functions have been programmed into the parts that implement the behavior of the thrusters, and the isolation valve.

3.2 Failure Mode Testing

Our selection of failure modes to test was based on the failure modes that were evaluated in the MAC FMEA report produced by the Johnson Space Center SMA team in 2002 (filename: "FMEA Mini AERCam Demonstrator.DOC"). Of those evaluated in the FMEA report, we chose failure modes whose analysis had been conducted at a high abstraction level, that we felt might produce interesting results through evaluation in more detail. For example, there are 12 micro thrusters on the MAC that are independently controlled by the FSW. The FMEA analysis assesses them as each thruster within the group controlling the axis having the same effect. We felt that testing individual thruster failures might allow a better assessment of MAC behavior than the analysis could provide.

Failure mode testing is initiated from a program that is executed when the MAC simulator starts. To maintain consistency, we ran each test sequence independently beginning after the MAC initial conditions were set up during simulator start-up. Each test begins by commanding the MAC to move from its starting position, to a destination waypoint and attitude, through the use of the MAC auto-move function. Failure modes are manifested while the MAC attempts to fly the commanded profile, and position and attitude data are logged to a file at one second intervals. To establish a basis for comparison, each test profile is repeated without manifesting failure modes so that data from a nominal MAC flight may be collected.

4.0 RESULTS

Meeting our objectives required that we first develop a MAC VSIL simulator capable of running the unmodified executable MAC FSW object code. As the development of the simulator is not the focus of this effort, discussion of the details of this aspect of our effort will not be included herein. The authors, however, will be pleased to discuss this subject in some detail with those interested.

Having developed a VSIL simulator of the MAC, we employed it to address our following three primary objectives:

A) To be able to independently test (rather than rely solely on FMEA) software behavior in response to failure modes.
B) To create & test failure mode scenarios in the virtual environment that are not practical in a HW integration test lab .
C) To evaluate if project SQA can be more effective by complementing FMEA with flight SW testing.

The results of our effort are presented in the following paragraphs, ordered by objective. Access to the VSIL, documentation, and data produced during this project effort may be obtained with permission from the NASA IV&V Facility. Tests and test results are documented in the Flight Software Test Document for the JSC Mini-AERCam Nanosatellite (SARP-CI04-77-001b).

4.1 Objective A

To be able to independently test (rather than rely solely on FMEA) software behavior in response to failure modes.

Having developed the MAC VSIL simulator, the next step was to program the MAC components involved, with failure modes we selected based on the FMEA report. Table 1 lists the failure modes we implemented in the simulated MAC. The Reference (Ref #), Failure Mode (FM #), and Page Number (Page #) columns are provided as cross references into the FMEA report to help correlate the listed failure mode tests with those failure modes analyzed.

Table 1: Failure Modes Implemented

Test	*Ref #*	*FM #*	*Page #*	*Failure Mode Description*
1.	00010	00010	1	External Leakage (Propellant leak causing thrust)
2.	00001	00080	17	Stuck thruster: Thruster 1 fails OPEN
3.	00001	00080	17	Stuck thruster: Thruster 1 fails CLOSED
4.	00001	00080	17	Stuck thruster: Thruster 2 fails OPEN
5.	00001	00080	17	Stuck thruster: Thruster 2 fails CLOSED
6.	00001	00080	17	Stuck thruster: Thruster 3 fails OPEN
7.	00001	00080	17	Stuck thruster: Thruster 3 fails CLOSED
8.	00001	00080	17	Stuck thruster: Thruster 4 fails OPEN
9.	00001	00080	17	Stuck thruster: Thruster 4 fails CLOSED
10.	00001	00080	17	Stuck thruster: Thruster 5 fails OPEN
11.	00001	00080	17	Stuck thruster: Thruster 5 fails CLOSED
12.	00001	00080	17	Stuck thruster: Thruster 6 fails OPEN
13.	00001	00080	17	Stuck thruster: Thruster 6 fails CLOSED
14.	00001	00080	17	Stuck thruster: Thruster 7 fails OPEN
15.	00001	00080	17	Stuck thruster: Thruster 7 fails CLOSED
16.	00001	00080	17	Stuck thruster: Thruster 8 fails OPEN
17.	00001	00080	17	Stuck thruster: Thruster 8 fails CLOSED
18.	00001	00080	17	Stuck thruster: Thruster 9 fails OPEN
19.	00001	00080	17	Stuck thruster: Thruster 9 fails CLOSED
20.	00001	00080	17	Stuck thruster: Thruster 10 fails OPEN
21.	00001	00080	17	Stuck thruster: Thruster 10 fails CLOSED
22.	00001	00080	17	Stuck thruster: Thruster 11 fails OPEN
23.	00001	00080	17	Stuck thruster: Thruster 11 fails CLOSED
24.	00001	00080	17	Stuck thruster: Thruster 12 fails OPEN
25.	00001	00080	17	Stuck thruster: Thruster 12 fails CLOSED
26.	30	30	5	Isolation Valve failure OPEN: Not responsive to CLOSE command
27.	40	40	7	Isolation Valve failure CLOSED: Not responsive to OPEN command
28.	00900	00900	67	Faulty rotational gyro: Incorrect Roll Rate: Low offset; Low gain
29.	00900	00900	67	Faulty rotational gyro: Incorrect Roll Rate: Low offset; High gain
30.	00900	00900	67	Faulty rotational gyro: Incorrect Roll Rate: High offset; Low gain

Test	*Ref #*	*FM #*	*Page #*	*Failure Mode Description*
31.	00900	00900	67	Faulty rotational gyro: Incorrect Roll Rate: High offset; High gain
32.	00900	00900	67	Faulty rotational gyro: Incorrect Yaw Rate: Low offset; Low gain
33.	00900	00900	67	Faulty rotational gyro: Incorrect Yaw Rate: Low offset; High gain
34.	00900	00900	67	Faulty rotational gyro: Incorrect Yaw Rate: High offset; Low gain
35.	00900	00900	67	Faulty rotational gyro: Incorrect Yaw Rate: High offset; High gain
36.	00900	00900	67	Faulty rotational gyro: Incorrect Pitch Rate: Low offset; Low gain
37.	00900	00900	67	Faulty rotational gyro: Incorrect Pitch Rate: Low offset; High gain
38.	00900	00900	67	Faulty rotational gyro: Incorrect Pitch Rate: High offset; Low gain
39.	00900	00900	67	Faulty rotational gyro: Incorrect Pitch Rate: High offset; High gain
40.	00900	00900	67	Faulty rotational gyro: NO Signal

We met this objective by successfully implementing the failure modes listed in table 1, and demonstrating that they may be manifested during test of the FSW.

4.2 Objective B

To create & test failure mode scenarios in the virtual environment that are not practical in a HW integration test lab .

4.2.1 Project-level Test Environments

Essentially all of the failure modes in table 1 may prove difficult and/or economically challenging to create and test realistically in a hardware integration test laboratory. The MAC is designed to operate in a space environment under micro-gravity conditions, with its 12 micro-thrusters oriented to provide 6 degrees of freedom (DOF) (3 attitudinal, 3 directional) of control over its movement. Two separate integration/test environments have been created by the development project to test the MAC:

1. The first MAC test environment comprises a massive, granite air-bearing table with a cradle to hold the MAC prototype. The cradle supplies the MAC with propellant and provides the air cushion required to float the assemblage during operation. This has proven helpful for validating much of the Guidance, Navigation, & Control algorithms but is limited by its inability to provide more than 3 DOF (2 translational + 1 yaw).
2. The second test environment is a virtual orbital environment simulator providing 6 DOF movement with a hardware-in-the-loop (HITL) connection to the MAC as well as a GPS signal generator. This supports the validation of algorithms developed for 6-DOF control, inertial & relative automatic altitude hold, automatic point-to-point maneuvering, relative GPS navigation, and GPS-based translation hold.

Clearly, the goal of these two test set-ups was not principally that of failure mode testing, but of MAC hardware and algorithm development. Of these two test environments, the second should support more realism in test scenario creation with its ability to provide 6 DOF mobility. However, test set-ups incorporating HITL generally come with a high price tag and other limitations that serve to restrict the number that can be produced and, correspondingly, access by non-development personnel such as SMA and IV&V teams.

4.2.2 VSIL Environment

The VSIL environment is purely virtual, i.e. it does not require the use of HITL to run the FSW, to provide sensor stimulus to the MAC, or for any other purpose. This VSIL models the MAC hardware and its orbital environment at the component level, and with sufficient fidelity to run the uninstrumented MAC FSW object code. Fidelity of each component model has been developed according to what is required for the purposes of our investigation. For example, the PowerPC750 CPU part is modeled with a relatively high level of fidelity (as required to run the FSW object) while components such as the rate gyro, FPGAs, etc. are modeled at a somewhat higher level of abstraction.

4.2.3 Failure Mode Test Results

As discussed in the methodology section, failure mode testing is initiated from a program that is executed when the MAC simulator starts. Each test begins by commanding the MAC to move from its starting position to a destination waypoint and attitude, through the use of the MAC auto-move function. Failure modes are manifested while the MAC attempts to fly the commanded profile, and position and attitude data are logged to a file at one second intervals. To establish a basis for comparison, each test profile is repeated without manifesting failure modes so that data from a nominal MAC flight may be collected. All tests presented here were run on the VSIL executing the FSW object code. All position measurements are in meters. Attitude waypoint numbers are in degrees, and plotted data is given in sine of the axis angle.

4.2.3.1 Leaky Gas Test

The Leaky Gas failure mode is described in the FMEA report as: “External Leakage” which may result in:

- Degraded or loss of GN2 flow to thrusters.
- Possible uncommanded motion due to loss of fuel resulting in inability to arrest motion.
- Avionics software commands Safe Mode if uncommanded motion detected.
- Degraded or loss of ability to control free-flier motion.
- Probable loss of Sprint mission objective.
- Potential loss of free-flier.
- Potential injury/damage to crew member or Orbiter.

We have chosen to manifest this failure as one that causes a small (0.031lbs) net thrust at the edge of the Mini-AERCam shell, tangential to thruster 1 & 2 cluster on the X axis, and parallel to the Y-axis in the -Y direction. In the absence of any corrective thruster action, this leak causes the MAC to rotate about the Z-axis (Yaw). The test may be easily adjusted to change the amount of thrust, the direction of the thrust, and the axis about which it is applied. Enhancements to the fidelity of this failure model would include loss of propellant and drop in pressure commensurate with location and severity of leak.

We ran this test using profile #1 with the Mini-AERCam (MAC) starting at rest at x,y,z location -25, 0, 0. Profile #1 commands the MAC to move to waypoint -20, -10, 5 and attitude 0, -40, 0. The test records 30 seconds of position and attitude data before manifesting the leaky gas failure, and another 7 min 30 sec of data with the failure present. We also ran the test without invoking the failure mode in order to collect 8 minutes of nominal data for comparison.

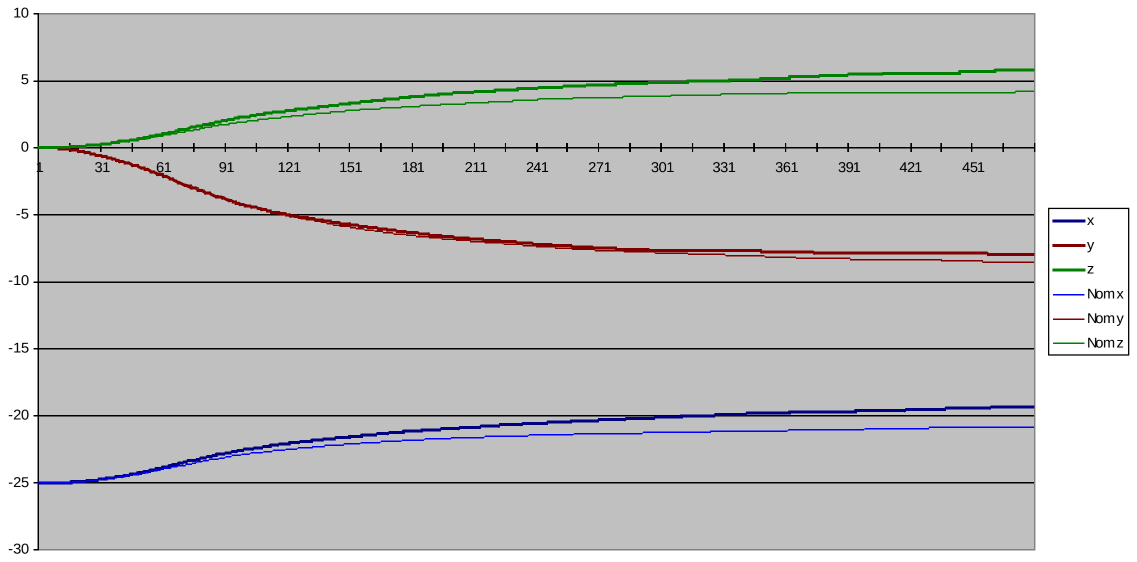


FIGURE 1: LEAKY GAS TEST POSITION PLOT - PROFILE #1

The data in figure 1 show that the MAC maintained a position profile fairly close (within 2.5m) to nominal in the presence of the thrust caused by the leaky gas failure mode while flying profile #1.

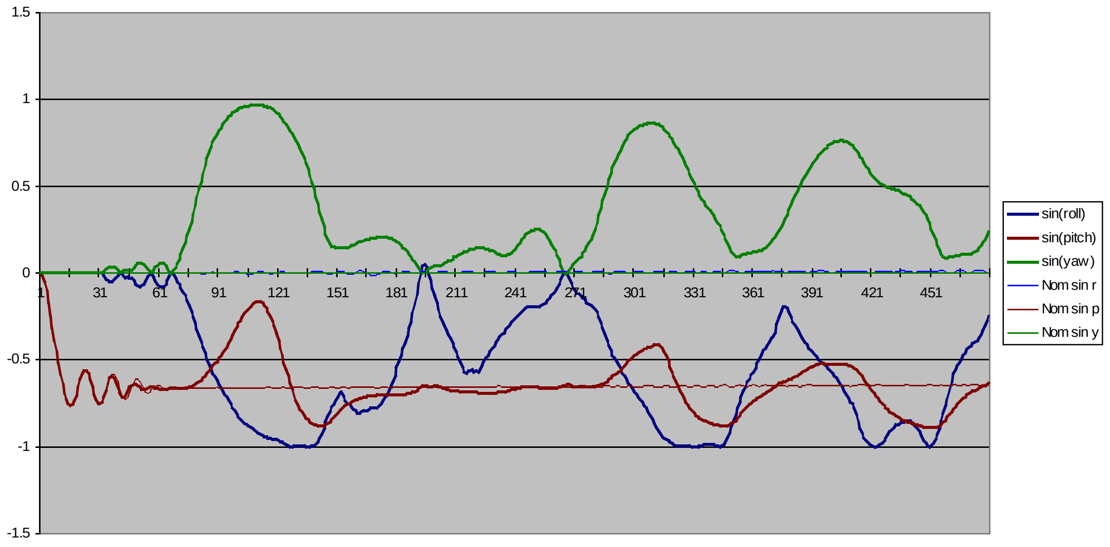


FIGURE 2: LEAKY GAS TEST ATTITUDE PLOT - PROFILE #1

The attitude data in figure 2 show that the MAC FSW was unable to maintain a steady attitude in the presence of the parasitic thrust produced by the leaky gas fault while flying profile #1.

4.2.3.2 Stuck Thruster Tests

The Stuck Thruster failure mode is analyzed by axis and described in the FMEA as “One of the x,y,z axis Thrusters Fails from the Pair Thrusters (x = Jet 1, Jet2), (x = Jet 7, Jet 8 Fails) ” which may result in:

- Messages and audible alarm at control station for Thruster loss.
- Loss of ability to control free-flyer motion about axis.
- Free-flyer in free drift.
- Loss of Mini AERCam Demonstrator mission objective.
- Possible injury/damage to EVA crew member or Orbiter.

The analysis also looked at failure of pairs of jets within each cluster, however, the analysis was not broken down to the failure of individual thrusters, and further to failure state (OPEN/CLOSED). The MAC thrusters are individually controlled by the FSW and the probability of a single thruster failure is likely higher than having a pair fail together. Figure 3 presents a diagram of the thruster configuration for reference. The rotational axis' are mapped as follows: X == ROLL; Y == PITCH; Z == YAW.

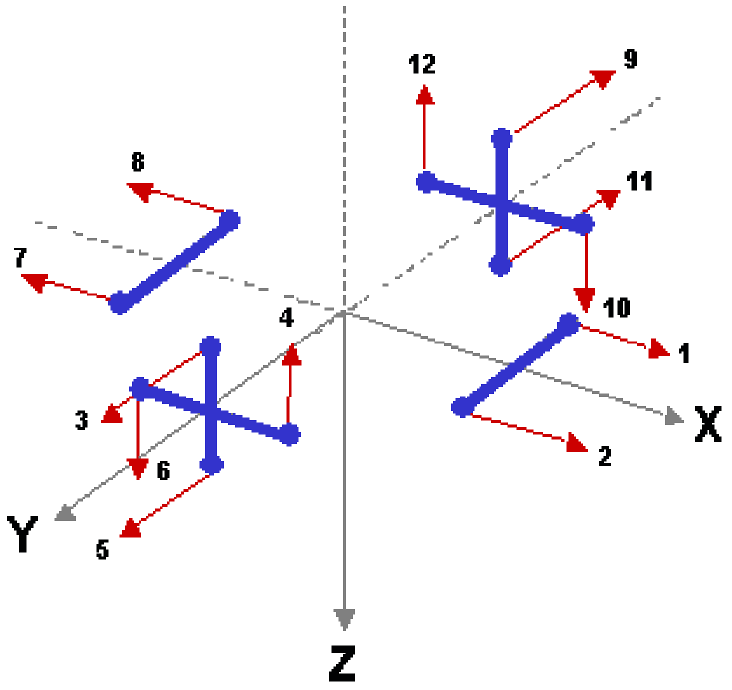


FIGURE 3: MAC THRUSTER CONFIGURATION

We manifested this failure by commanding the thruster to the failed state (OPEN or CLOSED) and making it unresponsive to commands from the FSW.

We ran the first test from rest position -25,0,0 directly to profile #3, collecting 60 secs of nominal data, and then failing each of the 12 thrusters first OPEN and the CLOSED for 60 secs each while collecting data. Each thruster was restored to nominal functioning before proceeding to test the next one and at the end of testing all thrusters, we collected another 60 secs of nominal data. A total of 26 minutes of data were collected and a comparison run of 26 minutes down the same profile with no failures manifested was made. The destination waypoint for profile #3 is Pos: 5,15,-5 and Att: 0,0,0.

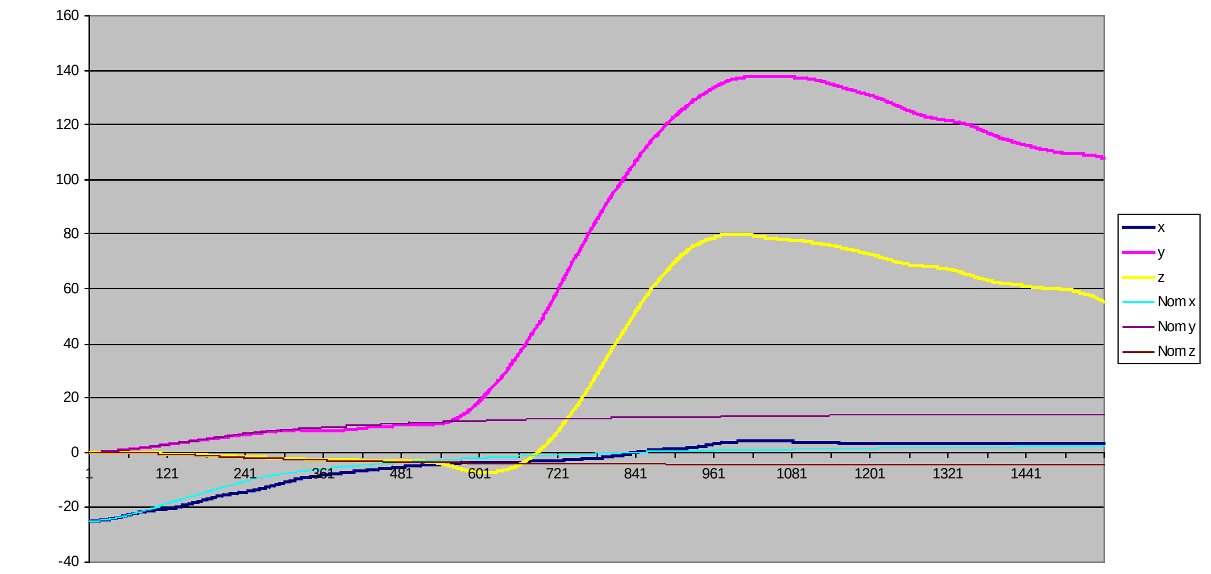


FIGURE 4: STUCK THRUSTER TEST POSITION PLOT - PROFILE #3

As you can see from figure 4, the FSW flew close to the nominal profile for the first 540 seconds despite the thruster failures. At 540 seconds, thruster 5 failure mode tests begin with the thruster failing OPEN for 60 seconds and then CLOSED for 60 seconds. During the thruster 5 failures, the MAC begins to depart the path to its commanded position waypoint – principally in the 'y' and 'z' axis'. Whether this occurs in response to the previous thruster failures or the failure modes of thruster 5 wasn't determined but after the 840 second mark, the MAC begins to slow its departure and appears to head back to the commanded waypoint.

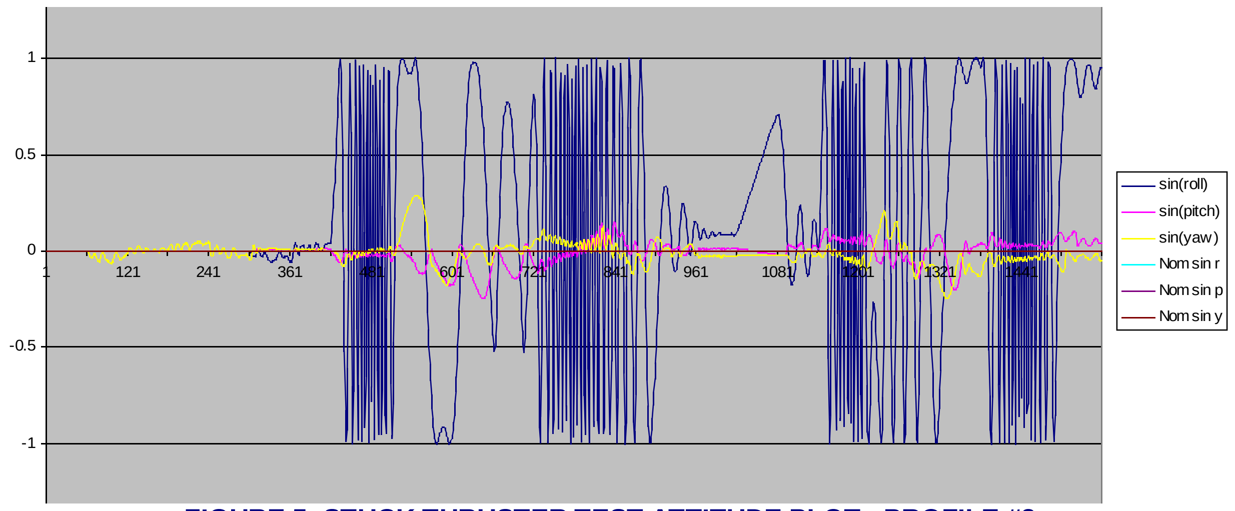


FIGURE 5: STUCK THRUSTER TEST ATTITUDE PLOT - PROFILE #3

Figure 5 shows a plot of the MAC attitude during the course of running profile #3 from the rest position. It's interesting to note that the data show the FSW is able to control the attitude fairly well about the pitch and yaw axis' in the presence of thruster failures, but not the roll axis. Since this is a plot of the sin of the axis angle, oscillation indicates that the MAC is rotating or rocking about the axis.

Expanding the period about the point where the MAC changes course (at 540 seconds), figure 6 shows that the MAC had been rolling during the thruster 4 failures but settled down at a roll angle of about 90 degrees as it entered the thruster 5 failure mode test period.

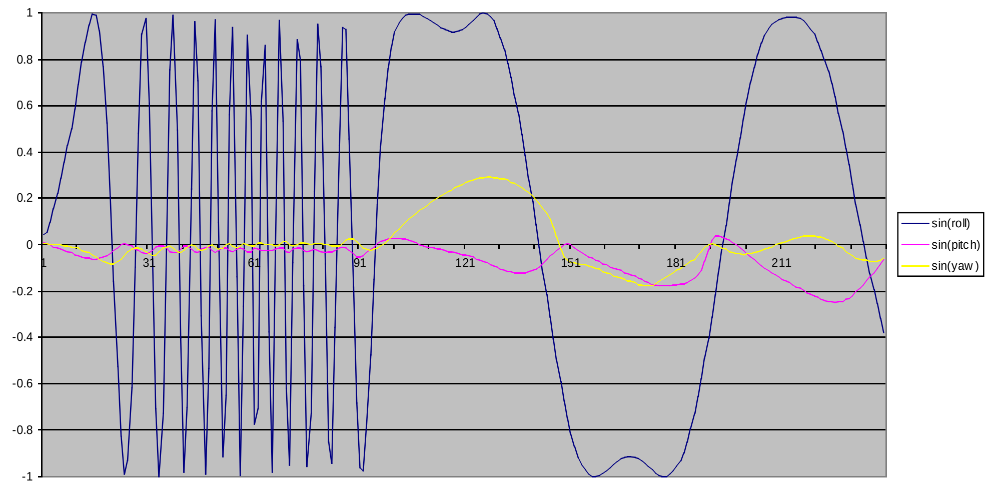


FIGURE 6: STUCK THRUSTERS 4 & 5 TEST ATTITUDE PLOT - PROFILE #3

4.2.3.3 Isolation Valve Tests

The Isolation Valve failure mode is described in the FMEA as “Fails Open” or “Fails Closed” which may result in:

- First failure: Inability to isolate GN2 supply from remainder of propulsion plumbing and components
- Second failure: Thruster fails on or computing upset resulting in uncommanded motion.
- Unable to arrest motion
- Loss of Sprint mission objective
- Potential injury/damage to EVA crew member or Orbiter.

Mitigation of this possible failure mode is handled in the Safe Mode FSW module. Unfortunately, this module was not completed at the time the MAC project was canceled and therefore no commands are sent to the isolation valve by the FSW in response to erratic MAC behavior. While we did implement failure modes in the isolation valve part, there was no useful test to evaluate the FSW behavior in controlling the valve.

Valve failure was implemented as causing the valve to be unresponsive to either the OPEN or CLOSE signal.

4.2.3.4 MEMS Rate Gyro Tests

The Rate Gyro failure mode is described in the FMEA as “Incorrect or no signal” which may result in:

- Avionics software detects discrepancy between commanded and measured rates and commands Safe Mode, closing the iso valve
- Loss of ability to control free flyer motion
- Free flyer in free drift
- Loss of Sprint Mission objective
- Possible injury/damage to EV crew and Orbiter

We implemented three types of failure modes in the simulated MEMS rate gyroscope part. Total failure of the MEMS gyro is accomplished by configuring it to be unresponsive to the multiplexer commands coming from the FPGA controller. Variable gain and offset biases may also be introduced into the output of each axis of the gyro.

We conducted three tests on the FSW object code to assess MAC behavior in the presence of rate gyro failures. The first test commands the MAC to fly profile #3 (Pos: 5,15,-5 and Att: 0,0,0), and begins by collecting 60 secs of nominal data before causing any RG failures. Then each axis is failed in turn with the following sequence: offset = 0.2 , gain = 0.5; offset = .2, gain = 1.5; offset = 0.4, gain = 0.5; offset = 0.4, gain = 1.5. Beginning with the roll axis, each setting is run for 60 secs before restoring the axis to nominal operational parameters and proceeding to the pitch and then yaw axis. The end of the test collects another 60 secs of nominal data. Total duration of the test is 14 minutes. A nominal profile was run for 14 minutes for comparison with the test data.

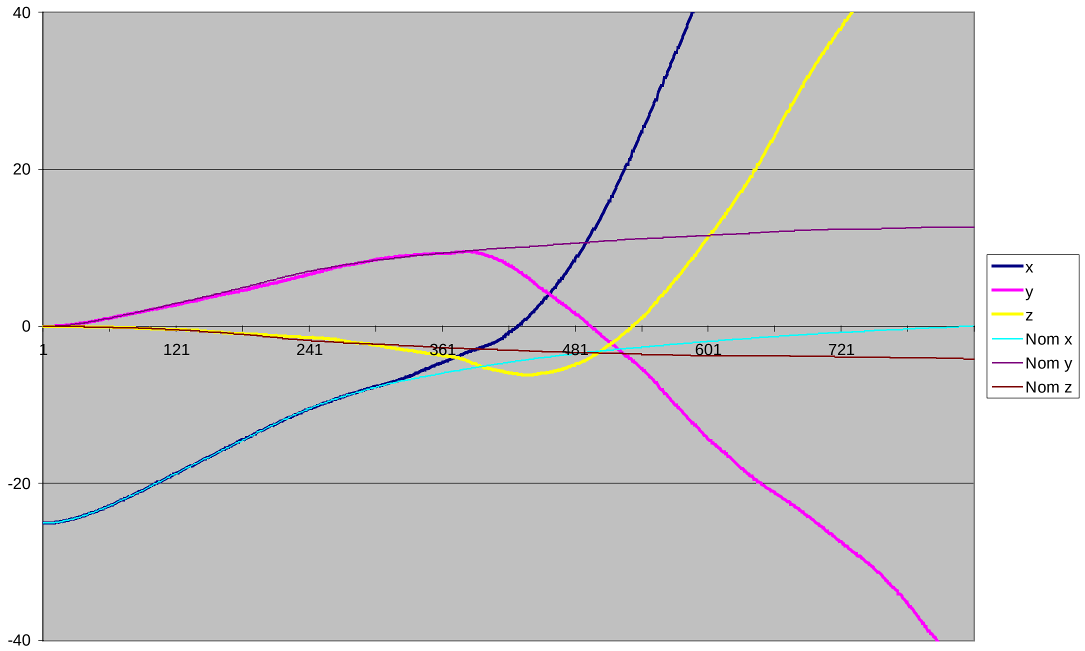


FIGURE 7: RATE GYRO OFFSET & GAIN FAULTS TEST POSITION PLOT - PROFILE #3

Figure 7 shows that after about 6 minutes of bias failures being induced, the MAC appears to have abandoned flying to the commanded waypoint and departed from the nominal path in all three axis'.

The plot in figure 8 shows the attitude of the MAC during the RG failure tests. Not surprisingly, the MAC looses its ability to maintain attitude stability in the absence of rate gyro information. As offset and gain bias faults are introduced in each axis (beginning at 61 seconds), the ability of the FSW to maintain attitude control is challenged accordingly.

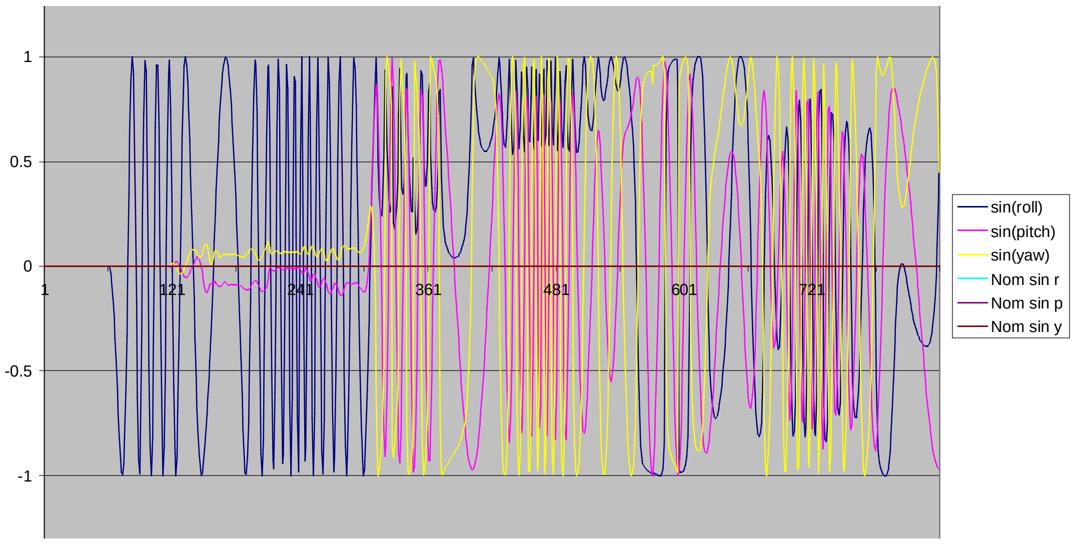


FIGURE 8: RATE GYRO OFFSET & GAIN FAULTS TEST ATTITUDE PLOT - PROFILE #3

The low offset/low gain roll axis failure mode causes loss of roll stability (seconds 61-120) that improves when the gain is boosted (secs 121-180), worsens with the high offset/low gain failure (181-240) and improves again with increasing gain (241-300). The plot in figure 9 below is an expanded view (61-300 seconds) of the attitude behavior during this sequence of roll axis bias failure modes. Note that the time frame of the below chart is relative to the start of the roll axis bias failure mode tests so the first second above corresponds to the 61st second on the previous chart.

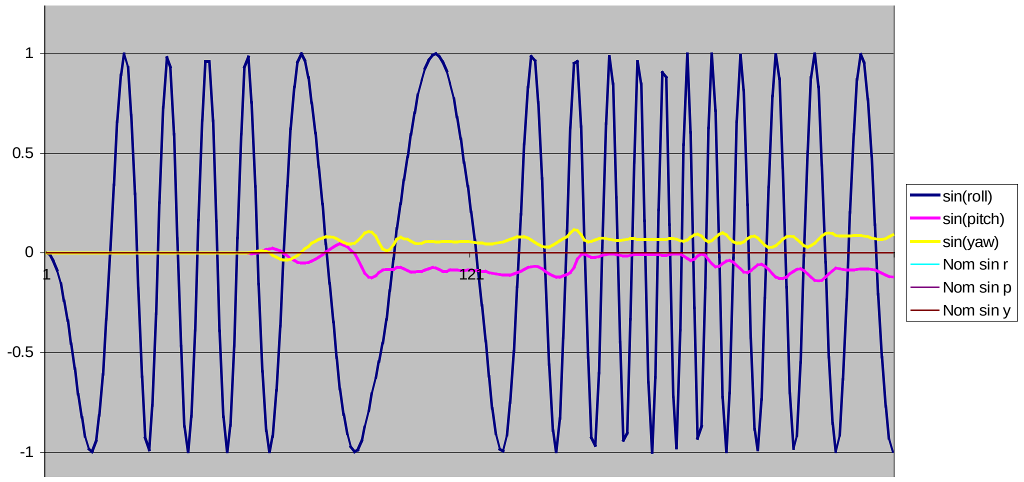


FIGURE 9: RG ROLL AXIS OFFSET & GAIN FAULTS TEST ATTITUDE PLOT - PROFILE #3

The second test was identical to the first except for the insertion of a 10 minute period during which a total RG failure was commanded immediately prior to invoking the sequence of offset and gain bias failures. The total duration of this test then is 24 minutes.

The position data for the second test, plotted in figure 10, show that the MAC followed the nominal course very closely during the total MEMS gyro failure when flying this profile. As with the previous test, however, the individual axis offset and gain bias failures caused the MAC to fly away from the nominal route to the waypoint.

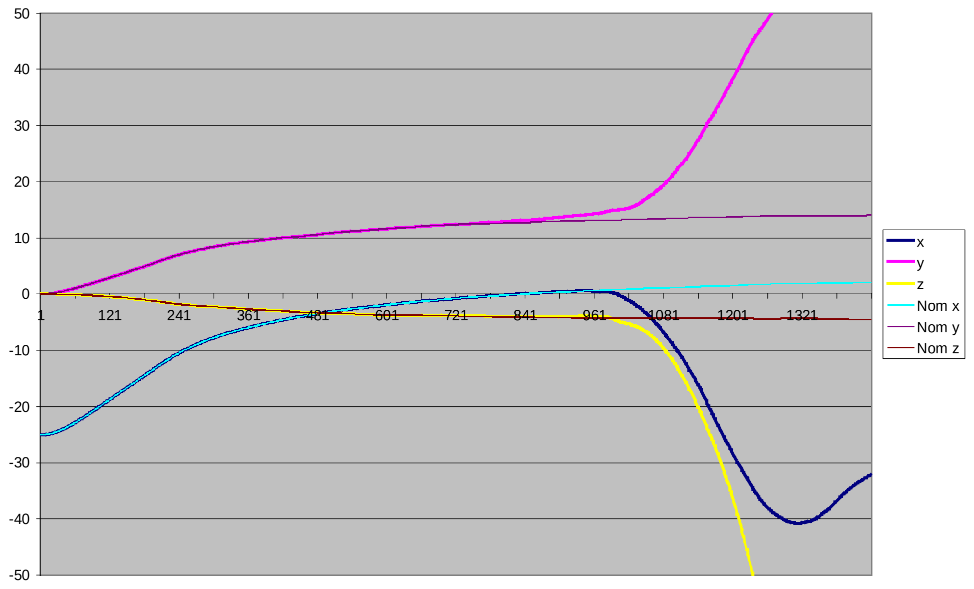


FIGURE 10: RATE GYRO TOTAL FAILURE TEST POSITION PLOT - PROFILE #3

The attitude plot for this test, shown in figure 11 below, reflects the position stability during the period of total RG failure, as well as the loss of attitude control during the offset and gain bias fault period.

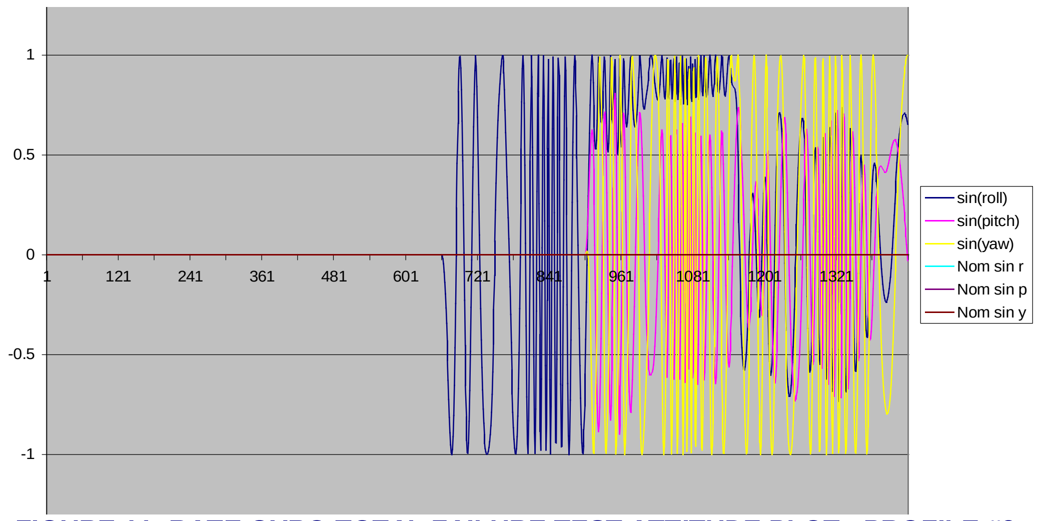


FIGURE 11: RATE GYRO TOTAL FAILURE TEST ATTITUDE PLOT - PROFILE #3

Given the high degree of attitude stability exhibited by the MAC in the presence of a total RG failure while flying profile #3, we thought it might be interesting to run the same test while flying a different profile. Since profile #3 does not incorporate a change of attitude from the 0, 0, 0 position, we hypothesized that with the thrusters firing in balanced pairs, the MAC could fly to the commanded position waypoint without changing its attitude. To test this idea we ran the same test with profile #4, one that commands the MAC to fly to new attitude and position waypoints.

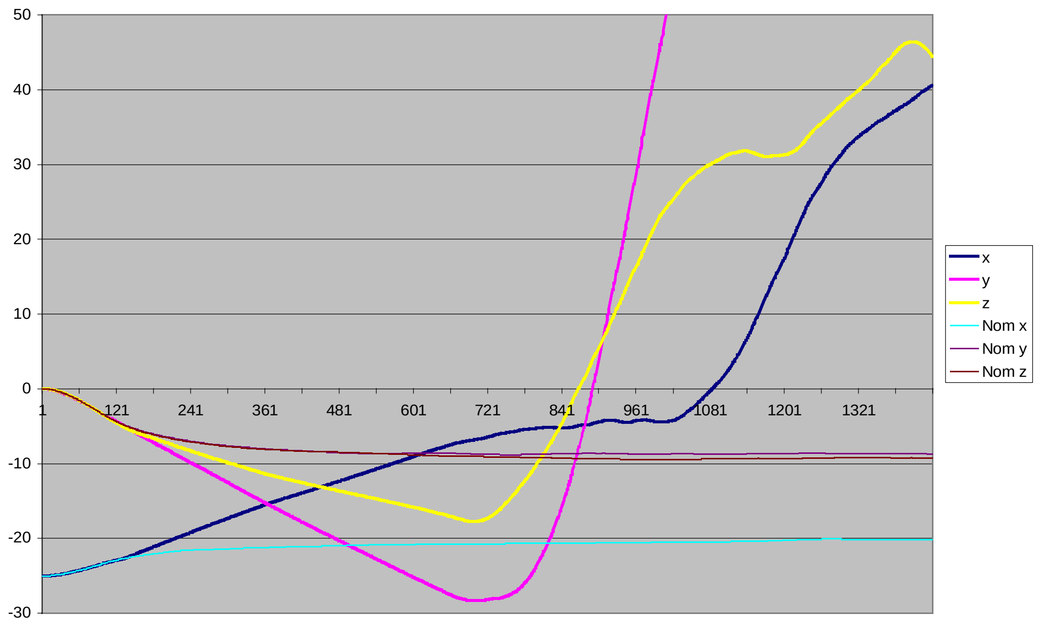


FIGURE 12: RATE GYRO TOTAL FAILURE TEST POSITION PLOT - PROFILE #4

From the data in the plot shown in figure 12 above, it is clear that the MAC was unable to follow a nominal course during the total failure of the RG while flying profile #4.

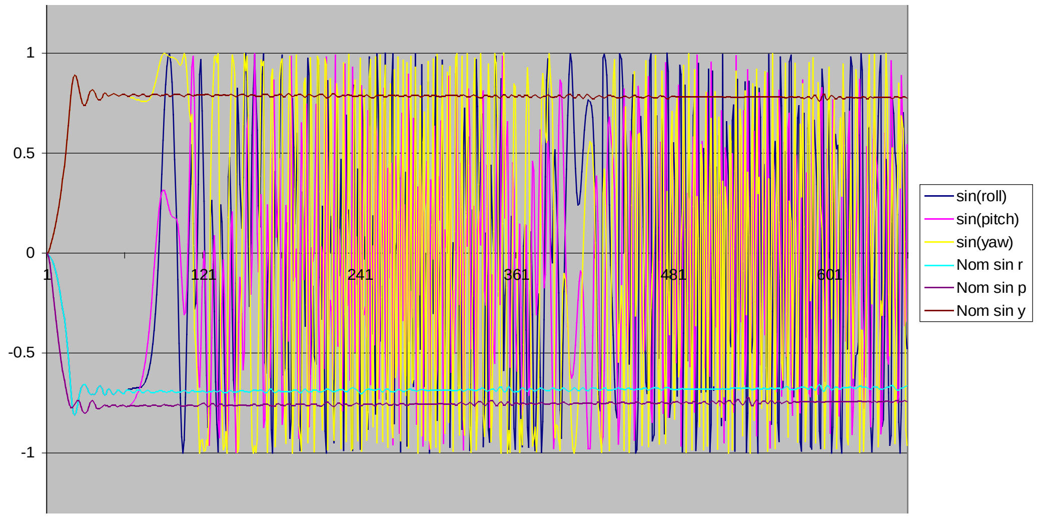


FIGURE 13: RATE GYRO TOTAL FAILURE TEST ATTITUDE PLOT - PROFILE #4

The data in the expanded attitude plot in figure 13 shows that the MAC had already stabilized on its new attitude waypoint before the total gyro failure occurred at 61 seconds. So it is not necessary for the MAC to be in transition attitudinally at the time the RG fails, for it to lose attitude control. Yet it is able to maintain control in the absence of RG data if the attitude is zero in all axis'.

Our selection of the offset and gain bias values were arbitrary (and possibly excessive) and applied in combination. To better quantify how the FSW behaves in response to MEMS rate gyro failure modes, testing of other offset and gain bias values independent of each other could provide useful information about the sensitivity of the FSW to errors of this type on a per-axis basis. It might also be informative to run these tests individually rather than directly following in sequence. This way the test of each axis would begin with the MAC attitude under good control and end by restoring nominal operation of the RG and monitoring how the FSW recovers.

4.3 Objective C

To evaluate if project SQA can be more effective by complementing FMEA with flight SW testing.

Our failure mode testing of the MAC FSW produced results that we believe provide a compelling case for the use of failure mode testing of FSW by the SMA, and/or IV&V teams. Lets examine how testing the FSW with the failure modes we selected might have added to the MAC FMEA report. The Safe Mode application was not implemented in the FSW build we tested so we were unable to evaluate its behavior for this study.

a) **Leaky Gas Test:** The FMEA report lists “Avionics software commands Safe Mode if uncommanded motion detected. Degraded or loss of ability to control free-flyer motion.” among the failure effect End Items. Our test reveals that the MAC is able to fly within 2.5m of a nominal profile to a programmed waypoint with a gas leak fault producing 0.26lbs thrust tangential to thruster 1 & 2 cluster on the X axis, and parallel to the Y-axis in the -Y direction. In the absence of any corrective thruster action, this leak causes the MAC to rotate about the Z-axis (Yaw).

   Had Safe Mode been operational during our test, it should have taken control as a result of the uncommanded motion and the outcome of the flight would likely have differed from what we observed. Additional tests could be run to create a profile of course deviation vs. leak thrust vectors. This could be used to guide decisions on whether or how to retrieve the MAC, dependent upon its distance and number of waypoints to fly to a safe rendezvous, if it should develop such a failure.

   Further, if the magnitude of parasitic thrust does not exceed the corrective capacity of the thrusters, perhaps a safe strategy could be developed for the MAC to auto-move to a retrieval point (or even the hanger), and the remaining propellant vented in a balanced manner (if shutting the iso-valve doesn't stop the leak). While the mission objectives may be unfulfilled, these measures could improve the likelihood of retrieving the MAC.

b) **Stuck Thrusters:** To evaluate the effect of thruster failures, the FMEA report grouped thrusters by those controlling the movement about each axis (4 per axis), and assessed the effect of the failure of a single thruster of the group. The FMEA indicates that the failure will cause the loss of ability to control the MAC in the affected axis – leaving the MAC to drift.

   Failure mode testing reveals that the MAC FSW is able to maintain control of the pitch and yaw attitudes despite each of the thrusters failing open and closed sequentially. However, the FSW is unable to maintain roll axis stability during failures of about half of the thrusters (see figure 5). If pitch and yaw can be controlled during thruster failures, could roll control be improved to perform similarly through a software or thruster configuration change?

   In addition to the attitude control dynamics, we observed that during the failures of thruster 5, the MAC begins to depart the path to its commanded position waypoint – principally in the 'y' and 'z' axis'. During the failure tests of thruster 7, the MAC begins to slow its departure and later turns back to the commanded waypoint. Is this a consequence of the roll instability? Will this behavior occur under more realistic failure scenarios, e.g. one thruster failing rather than all failing sequentially? Additional testing is necessary to better understand this behavior.

Characterizing the behavior of the MAC in response to thruster failures may support the development of effective strategies for responding to failures in a manner that improves the probabilities of recovering the MAC, or avoiding possible injury to EVA crew-member or orbiter.

c) **Isolation Valve:** We implemented the isolation valve part in the MAC VSIL and created a failure mode that put it into an unresponsive state such that if it was open, it would not respond to commands to close it, and visa versa. During our testing of this failure mode, we discovered that commands were not being sent from the FSW during the test, to the FPGA register controlling operation of the iso-valve. It turns out that the Safe Mode application had not been incorporated into this software build. This was not surprising since funding for continued Mini-AERCam development was not assured at the time, and was eventually cut. This finding would certainly be of interest if discovered near the end of the MAC development effort, but it is unlikely that such an omission would occur.

d) **Rate Gyro Offset/Gain Bias Errors:** It is no wonder that the MAC loses attitude stability when faced with rate gyro bias errors, particularly those related to offset of a significant magnitude. The FSW demonstrated an impressive ability to continue flying to its commanded translational waypoint despite bias faults in the roll axis rate gyro signal. When pitch axis bias failures are introduced, however, the MAC departs the path to its commanded position waypoint in all three translational axis' (figure 7).

The FMEA report indicates that the MAC should transition to Safe Mode upon detection of a discrepancy between commanded and measured rates, and close the iso-valve. It recommends attempting to reacquire the MAC, presumably under manual control.

It may be that certain rate gyro bias failures develop over time, in which case the FSW may be able to detect a discrepancy before the point of attitude instability is reached. Failure mode testing can help determine that point empirically, and lead to recommendations about reporting rate discrepancies, and when to abort a mission and return to the hanger while still within safe attitude controllability limits.

Another observation from the test results is that the MAC is less sensitive to gain bias failures than offset bias failures. This may or may not warrant separate strategies for dealing with each failure type. Since these types of failures fall into the realm of degraded performance, it may be possible to compensate for the bias by adding a calibration function to the software that could be exercised when a rate discrepancy has been detected.

Failure management strategies may be further refined based on the observation that the MAC reacts differently to bias failures in different axis'.

e) **Rate Gyro Total Failure:** The total failure of the MEMS rate gyro part was modeled by disabling its response to requests for data from the controller (processor FPGA) part. In the FMEA report, this was classified in the same category as the rate gyro bias failures and, accordingly, the MAC responds by switching to Safe Mode and closing the iso-valve.

Testing with this failure mode revealed that when the MAC attitude was at its zero reference orientation, it was able to auto-move to a translational waypoint provided no attitude change was commanded (see first 11 minutes of figure 10). This may suggest a response strategy that takes the MAC attitude into account at the time of rate gyro failure. Since the MAC behaves differently in the face of a total failure, it makes sense to classify it separately from rate gyro bias failures.

In all cases, testing the behavior of the FSW in response to failures provides quantitative, empirical information that promotes an improved understanding of the MAC dynamics under anomalous conditions. The test scenarios we created were intended to stress the system to elicit behaviors that may be candidates for more detailed investigation.

Evaluate methods of gathering metric data on dynamic aspects of a software program only possible (or at least a lot easier) in a virtual environment.

In addition to running the failure mode tests developed for assessing the MAC FSW, we configured the test programs to gather some SW execution metrics of interest. None of the metrics gathered required instrumentation of the code itself. The metric data files are included as part of the project documentation submitted to the NASA IV&V Facility and may be accessed with permission from the research authority at the facility.

a) **Software Path Coverage:** The software path coverage report shows all conditional jump (or call) assembly code instructions and identifies whether a condition caused a jump or no jump (execution fell through to the following instruction) during execution. Typically, test development would not be considered complete until all code paths are exercised. Table 2 shows an excerpt from the MAC FSW path coverage report.

Table 2: SW Path Coverage Report Excerpt

ADDRESS	JUMP	NO JUMP	SYMBOL OFFSET
001bc434	x	x	gnc_collect_rate_data + 1c hex
001bc4c0		x	gnc_collect_rate_data + a8 hex
001bc4dc	x	x	gnc_collect_rate_data + c4 hex
001bc4e4	x		gnc_collect_rate_data + cc hex
001bc560	x	x	gnc_collect_rate_data + 148 hex
001bc6d8	x		gnc_resume_setup + b8 hex
001bc720	x	x	gncexec_cmd_switch + 44 hex
001bc724	x	x	gncexec_cmd_switch + 48 hex
001bc72c	x	x	gncexec_cmd_switch + 50 hex
001bc730	x	x	gncexec_cmd_switch + 54 hex
001bc740		x	gncexec_cmd_switch + 64 hex
001bc74c	x		gncexec_cmd_switch + 70 hex
001bc774		x	gncexec_cmd_switch + 98 hex
001bc778	x	x	gncexec_cmd_switch + 9c hex
001bc780		x	gncexec_cmd_switch + a4 hex
001bc788	x		gncexec_cmd_switch + ac hex
001bc7a8	x	x	gncexec_cmd_switch + cc hex
001bc7ac		x	gncexec_cmd_switch + d0 hex
001bc7b4	x		gncexec_cmd_switch + d8 hex
001bc7dc	x	x	gncexec_cmd_switch + 100 hex
001bc7e4	x	x	gncexec_cmd_switch + 108 hex
001bc7ec		x	gncexec_cmd_switch + 110 hex
001bc7f8	x	x	gncexec_cmd_switch + 11c hex
001bc7fc	x	x	gncexec_cmd_switch + 120 hex
001bc804		x	gncexec_cmd_switch + 128 hex
001bc80c	x		gncexec_cmd_switch + 130 hex
001bc820	x		gncexec_cmd_switch + 144 hex
001bcf18	x	x	gncexec_cmd_switch + 83c hex
001bcf6c	x		gncexec_cmd_switch + 890 hex
001bd0d4	x	x	gncexec_cmd_switch + 9f8 hex
001bd2a8	x	x	gncexec_cmd_switch + bcc hex
001bd2b4		x	gncexec_cmd_switch + bd8 hex
001bd2bc	x		gncexec_cmd_switch + be0 hex
001bd340	x		gnc_check_and_make_fireinput + 30 hex
001bd7e4	x	x	gnc_check_and_make_fireinput + 4d4 hex
001bd860	x	x	gnc_check_and_make_fireinput + 550 hex
001bda10	x		gncexec + 54 hex
001bdbc0	x	x	gncexec + 204 hex
001bdc84		x	gncexec + 2c8 hex
001bdca4	x		gncexec + 2e8 hex
001bdee8	x		gncexec + 52c hex
001bdf4c	x	x	gncexec + 590 hex
001bdf88	x	x	gncexec + 5cc hex
001bdfa0	x	x	gncexec + 5e4 hex

b) **Software Code Coverage:** The software code coverage report calculates the percentage of code executed during the test run. For this report, the MAC was commanded to move from its initialized position to the waypoint and attitude of profile #1. No failure mode tests were run during the collection of this report. Our report shows that 17% (763 lines of code in aercam_gnc()) of the GNC code was executed during this auto-move exercise. Ideally, tests would continue to be written until 100% coverage was achieved.

c) **Software Subroutine Call & Interrupt:** This report logs the time of all subroutine calls, interrupts, and returns made during program execution. These data are useful for analyzing and debugging program control flow, and may find use by SMA as a means of observing task and interrupt timing & nesting behavior. Table 3 presents an excerpt of this report showing an interrupt occurring during the taskDelay() call.

Table 3: SW Subroutine Call & Interrupt Report Excerpt

```
   Time     Address          Subroutine Call / Interrupt
60.211105 001c4d7c bl  -> 002784c0 [taskDelay] <r1 = 00c574c0>
60.211105 00278674   bl  -> 0027c008 [windDelay] <r1 = 00c574a0>
60.211106 0027c0dc     bcctrl  -> 0025d850 [qPriBMapRemove] <r1 = 00c57480>
60.211106 0025d880       bl  -> 0025bb6c [dllRemove] <r1 = 00c57470>
60.211106 0025bb9c       bclr  -> 0025d884
60.211107 0025d8a0       bl  -> 0025d7a8 [qPriBMapClear] <r1 = 00c57470>
60.211107 0025d7e8       bclr  -> 0025d8a4
60.211107 0025d8b8       bl  -> 0025d7ec [qPriBMapHigh] <r1 = 00c57470>
60.211107 0025d808         bl  -> 0023c390 [ffsMsb] <r1 = 00c57460>
60.211107 0023c398         bclr  -> 0025d80c
60.211108 0025d81c         bl  -> 0023c390 [ffsMsb] <r1 = 00c57460>
60.211108 0023c398         bclr  -> 0025d820
60.211108 0025d84c       bclr  -> 0025d8bc
60.211108 0025d908     bclr  -> 0027c0e0
60.211109 0027c144     bcctrl  -> 0025d9d4 [qPriDeltaPut] <r1 = 00c57480>
60.211110 0025da2c       bl  -> 0025bb24 [dllInsert] <r1 = 00c57460>
60.211110 0025bb5c       bclr  -> 0025da30
60.211110 0025da80     bclr  -> 0027c148
60.211110 0027c178   bclr  -> 00278678
60.211110 00278678   bl  -> 002616a0 [windExit] <r1 = 00c574a0>
60.211112 002716e4     bl  -> 001fc460 [_savegpr_21_l] <r1 = 00c58cf0>
60.211113 001fc490     bclr  -> 002716e8
60.211113 0027175c     bl  -> 0027d9c8 [workQDoWork] <r1 = 00c58cf0>
60.211113 0027d9d4       bl  -> 001fc474 [_savegpr_26_l] <r1 = 00c58cd0>
60.211113 001fc490       bclr  -> 0027d9d8
60.211113 0027d9d8       bl  -> 0023fbd8 [__errno] <r1 = 00c58cd0>
60.211113 0023fbe0       bclr  -> 0027d9dc
60.211114 0027da54       bl  -> 0023fbd8 [__errno] <r1 = 00c58cd0>
60.211114 0023fbe0       bclr  -> 0027da58
60.211114 001fc41c     bclr  -> 00271760
60.211114 00271760     bl  -> 001efb30 [intLock] <r1 = 00c58cf0>
60.211114 001efb40     bclr  -> 00271764
60.211115 00271850     bl  -> 001efb44 [intUnlock] <r1 = 00c58cf0>
60.211115 001efb58     bclr  -> 00271854
60.211115 00271854     bl  -> 0027d7f8 [windPwrDown] <r1 = 00c58cf0>
60.211115 0027d824       bl  -> 001efb30 [intLock] <r1 = 00c58cd0>
60.211115 001efb40       bclr  -> 0027d828
60.211115 0027d82c       bl  -> 0027d1e4 [windTickWaitGet] <r1 = 00c58cd0>
60.211116 0027d220         bcctrl  -> 0025dbf0 [qPriDeltaKey] <r1 = 00c58cc0>
60.211116 0025dc14         bclr  -> 0027d224
60.211116 0027d234       bclr  -> 0027d830
60.211116 0027d868       bl  -> 001efb44 [intUnlock] <r1 = 00c58cd0>
60.211116 001efb58       bclr  -> 0027d86c
```

```
  Time     Address          Subroutine Call / Interrupt
60.211117 0027d878       bcctrl  -> 001f01e0 [vxPowerDown] <r1 = 00c58cd0>
60.211117 001f0218       bclr  -> 0027d87c
60.211117 0027d890     bclr  -> 00271858
60.211824 00271858     interrupt  -> 00000500
60.211824 00000508       bl  -> 001eeeec [intEnt] <r1 = 00c58cf0>
60.211827 001ef16c         bcctrl  -> 0027d894 [windPwrUp] <r1 = 00be6828>
60.211827 0027d8c0           bl  -> 001efb30 [intLock] <r1 = 00be6808>
60.211827 001efb40           bclr  -> 0027d8c4
60.211827 0027d928           bl  -> 001efb44 [intUnlock] <r1 = 00be6808>
60.211828 001efb58           bclr  -> 0027d92c
60.211828 0027d944         bclr  -> 001ef170
60.211828 001ef18c       bclr  -> 0000050c
60.211828 00000514       bl  -> 00010604 [sysAerPicIntHandler] <r1 = 00be6838>
60.211828 00010610         bl  -> 001fc464 [_savegpr_22_l] <r1 = 00be6808>
60.211828 001fc490         bclr  -> 00010614
60.211829 00010630         bl  -> 000105b0 [sysAerPicIntLevelSet] <r1 = 00be6808>
60.211829 000105bc           bl  -> 001fc47c [_savegpr_28_l] <r1 = 00be67e8>
60.211829 001fc490           bclr  -> 000105c0
60.211829 001fc43c         bclr  -> 00010634
60.211830 0001063c         bl  -> 001efb44 [intUnlock] <r1 = 00be6808>
60.211830 001efb58         bclr  -> 00010640
60.211830 000106a0         bcctrl  -> 000108f8 [sysClkInt] <r1 = 00be6808>
60.211830 00010938           bcctrl  -> 0001395c [usrClock] <r1 = 00be67f8>
60.211831 00013968             bl  -> 0027ab1c [tickAnnounce] <r1 = 00be67e8>
60.211831 0027abdc               bl  -> 0027c78c [windTickAnnounce] <r1 = 00be67d8>
60.211831 0027c798                 bl  -> 001fc46c [_savegpr_24_l] <r1 = 00be67a8>
60.211831 001fc490                 bclr  -> 0027c79c
60.211832 0027c82c                 bcctrl  -> 0025db88 [qPriDeltaGetExpired] <r1 = 00be67a8>
60.211833 0025dbd4                   bl  -> 0025bb6c [dllRemove] <r1 = 00be6798>
60.211833 0025bba4                   bclr  -> 0025dbd8
60.211833 0025dbec                 bclr  -> 0027c830
60.211834 0027c910                 bl  -> 001efdb4 [taskRtnValueSet] <r1 = 00be67a8>
60.211834 001efdb8                 bclr  -> 0027c914
60.211834 0027ca30                 bcctrl  -> 0025d738 [qPriBMapPut] <r1 = 00be67a8>
60.211835 0025bb5c                 bclr  -> 0027ca34
60.211835 0027ca44                 bcctrl  -> 0025db88 [qPriDeltaGetExpired] <r1 = 00be67a8>
60.211836 0025dbd4                   bl  -> 0025bb6c [dllRemove] <r1 = 00be6798>
60.211836 0025bba4                   bclr  -> 0025dbd8
60.211836 0025dbec                 bclr  -> 0027ca48
60.211836 0027c910                 bl  -> 001efdb4 [taskRtnValueSet] <r1 = 00be67a8>
60.211837 001efdb8                 bclr  -> 0027c914
60.211837 0027ca30                 bcctrl  -> 0025d738 [qPriBMapPut] <r1 = 00be67a8>
60.211838 0025bb5c                 bclr  -> 0027ca34
60.211838 0027ca44                 bcctrl  -> 0025db88 [qPriDeltaGetExpired] <r1 = 00be67a8>
60.211838 0025dbec                 bclr  -> 0027ca48
60.211839 0027cb6c                 bcctrl  -> 0026ed08 [kernelRoundRobinHook] <r1 = 00be67a8>
60.211840 0026ee14                   bcctrl  -> 0025d850 [qPriBMapRemove] <r1 = 00be6798>
60.211840 0025d880                     bl  -> 0025bb6c [dllRemove] <r1 = 00be6788>
60.211840 0025bb9c                     bclr  -> 0025d884
60.211840 0025d8a0                     bl  -> 0025d7a8 [qPriBMapClear] <r1 = 00be6788>
60.211841 0025d7e8                     bclr  -> 0025d8a4
60.211841 0025d908                   bclr  -> 0026ee18
60.211841 0026ee30                   bcctrl  -> 0025d738 [qPriBMapPut] <r1 = 00be6798>
60.211842 0025bb5c                   bclr  -> 0026ee34
60.211842 0026ee48                 bclr  -> 0027cb70
60.211843 001fc41c               bclr  -> 0027abe0
60.211843 0027abe0               bl  -> 002616a0 [windExit] <r1 = 00be67d8>
60.211843 002616cc                 bl  -> 0026152c [emptyWorkQueue] <r1 = 00be67b8>
60.211844 0026169c                 bclr  -> 002616d0
60.211844 002616f8               bclr  -> 0027abe4
60.211844 0027abf4             bclr  -> 0001396c
60.211844 00013978           bclr  -> 0001093c
60.211844 00010948         bclr  -> 000106a4
```

```
  Time     Address          Subroutine Call / Interrupt
60.211844 000106ac         bl  -> 001efb30 [intLock] <r1 = 00be6808>
60.211844 001efb40         bclr  -> 000106b0
60.211845 001fc41c       bclr  -> 00000518
60.211845 0000051c       bl  -> 001ef2fc [intExit] <r1 = 00be6848>
60.211847 002716e4         bl  -> 001fc460 [_savegpr_21_l] <r1 = 00c58ce0>
60.211847 001fc490         bclr  -> 002716e8
60.211847 0027175c         bl  -> 0027d9c8 [workQDoWork] <r1 = 00c58ce0>
60.211848 0027d9d4           bl  -> 001fc474 [_savegpr_26_l] <r1 = 00c58cc0>
60.211848 001fc490           bclr  -> 0027d9d8
60.211848 0027d9d8           bl  -> 0023fbd8 [__errno] <r1 = 00c58cc0>
60.211848 0023fbe0           bclr  -> 0027d9dc
60.211848 0027da54           bl  -> 0023fbd8 [__errno] <r1 = 00c58cc0>
60.211848 0023fbe0           bclr  -> 0027da58
60.211848 001fc41c         bclr  -> 00271760
60.211849 00271760         bl  -> 001efb30 [intLock] <r1 = 00c58ce0>
60.211849 001efb40         bclr  -> 00271764
60.211849 00271888         bl  -> 001efb44 [intUnlock] <r1 = 00c58ce0>
60.211849 001efb58         bclr  -> 0027188c
60.211850 00271958         bcctrl  -> 00246e98 [taskVarSwitchHook] <r1 = 00c58ce0>
60.211850 00246ec4           bl  -> 001f5684 [objVerify] <r1 = 00c58cc0>
60.211850 001f56a8           bclr  -> 00246ec8
60.211851 00246fcc         bclr  -> 0027195c
60.211851 00271978         bl  -> 001efb30 [intLock] <r1 = 00c58ce0>
60.211851 001efb40         bclr  -> 0027197c
60.211851 00271ad4         bl  -> 001fc344 [_restgpr_21] <r1 = 00c58ce0>
60.211852 001fc370         bclr  -> 00271ad8
60.211854 00261a50     interrupt return  -> 0027867c
60.211854 002786b8   bclr  -> 001ec124
```

Other reports may be generated to aid in both project-level, and SMA analysis activities. For example:

- The **Software Interrupt** report is used to compute the percent of reserve processing time available.
- The **Software Execution Marker** report provides a look at where the processing time is spent. This is done by counting the number of times that individual instructions are executed over a set period of time. This, in effect, forms a histogram of where processing time is spent during program execution.

Access to timing information, memory & register data, interrupt events, actuator & sensor data, and SW parameters, supports the recording of data for creating many other metric reports.

4.4 Discussion

Unfortunately, funding for the Mini-AERCam spacecraft development was discontinued in December 2005 and consequently, development MAC FSW was not completed. Our investigation was conducted with a software build we received early November, 2005 identified as: *Aercam3_V3.14*. This build did not incorporate the Safe Mode software and while the FIRE algorithm was included, it was not running in the simulator during our failure mode testing effort. As such, the results of our failure mode tests do not necessarily represent the behavior of of the MAC under the control of a flight-worthy build of the software.

That said, the failure mode testing we conducted elicited attitudinal and translational control behavior that would have been masked by activation of Safe Mode. For example, determination that the MAC is unable to maintain roll control in the presence of certain thruster failures while it is able to maintain effective pitch and yaw stability regardless of which thruster is failed. This may lead to changes in the Safe Mode design that could improve the ability to recover the MAC should a thruster failure occur. This is illustrative of the value of failure mode testing at least prior to the integration of the Safe Mode function, and perhaps at earlier stages of the spacecraft development as well.

Since Safe Mode is designed to control the MAC in the event of a system failure, its behavior can also be empirically evaluated through failure mode testing. Verification of the final FSW configuration can be independently made by the SMA team through failure testing in the VSIL. This may confirm that:

a) failures are detected and reported as required,
b) transition in and out of Safe Mode occurs as specified, and
c) Safe Mode operates as intended.

Failure mode mitigation strategies may also be evaluated in this environment – particularly those implemented through software control. Modified flight software may be loaded into the VSIL and evaluated empirically using failure mode tests previously developed.

Evaluating operator-in-the-loop (OITL) mitigation strategies using the MAC VSIL would require greater execution speeds than those we obtained while running the FSW object code. At real-time execution speeds, OITL mitigation procedures could be verified in the VSIL and evaluated for their effectiveness. We have identified several promising avenues through which we could speed up execution of the simulator from the present 16% real-time (on an AMD 3700+ Athlon 64 PC) to the 100% real-time necessary for meaningful OITL operation.

5.0 IMPLICATIONS

The results presented in this report establish the potential value that failure mode testing can add to FMEA and consequently, to safety and mission assurance. No longer need strategies for the mitigation of failure modes be developed solely on the basis of system design and control model analysis. Instead, SMA analysts can set up failure modes of interest within a VSIL, evaluate the behavior of the system and software in response to the failures, and develop mitigation procedures based on empirical results. This is a significant step beyond what is typically produced by analysis alone.

Failure mode testing is useful not only as an adjunct to FMEA, but it is also as a tool that allows SQA to confirm that the system (including the software) has correctly implemented the behavior specified in the requirements. With the combined benefits of failure mode testing in support of FMEA, and independent verification of requirements through testing, SMA teams can significantly increase their effectiveness through the use of a VSIL.

6.0 FUTURE WORK

Meeting the objectives set forth for this investigation has prepared the way for use of a VSIL for failure mode testing on an active NASA project. Since funding for the Mini-AERCam was discontinued at the end of 2005, we have proposed to apply the results of this effort to the Crew Exploration Vehicle (CEV) avionics development project.

At the time of this writing, NASA has recently selected a vendor to develop the CEV and related launch vehicle components. This is an ideal time to develop a VSIL for the following reasons:

a) System requirements are available along with essential system design documentation from which the system-level VSIL can be created.

b) Failure modes can be simulated in support of testing the system design.

c) Tests created for evaluating the system design may be reused in the VSIL to verify that the flight software has correctly implemented the requirements.

Establishing a VSIL simulator early in the development of the CEV will ensure that the SMA team is actively involved through testing the system and software from the very beginning of the development effort. Early and increased involvement by the SMA team gives more time for them for a detailed assessment of the design at it is created and evolves.

7.0 ACKNOWLEDGMENTS

We would like to thank the NASA Office of Safety and Mission Assurance for funding our investigation under its Software Assurance Research Program, and the NASA Independent Validation and Verification Facility for providing oversight and guidance throughout the execution of our project. We also thank Dr. Steven Fredrickson, Program Manager for the Mini-AERCam project, for providing us with the Mini-AERCam design documentation and flight software, and for his continued interest and support throughout – and following the completion of – our effort.

8.0 REFERENCES

[1] Haapanen P, Helminen A, "Failure Mode and Effects Analysis of Software-Based Automation Systems," Helsinki, Finland, August 2002, STUCK-YTO-TR 190, ISBN 951-712-585-2, http://www.fmeainfocentre.com/handbooks/softwarefmea.pdf

[2] Feather, M., Cornford, S., Moran, K., "Got Risk? A Risk-Centric Perspective for Spacecraft Technology Decision-Making," Jet Propulsion Laboratory, National Aeronautics and Space Administration, Pasadena, California, February 2004, http://hdl.handle.net/2014/38292

[3] Boehm, B.W. "Software Engineering." IEEE Transactions on Computer. 1976: SE-1(4): 1226-1241.

[4] Tassey, G. The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards & Technology.2002.<http://www.nist.gov/director/prog-ofc/report02-3.pdf> (19 Jan. 2005).

[5] Leveson, N.G. "Safeware - System, Safety and Computers". Addison Wesley. 1995.

[6] Leveson, N.G. "The Role of Software in Spacecraft Accidents." AIAA Journal of Spacecraft and Rockets, Vol. 41, No. 4, July 2004.

[7] Dabney, J.B. "Return on Investment of Independent Verification and Validation Study Preliminary Phase 2B Report." Software Assurance Research Program Results Web Site. NASA IV&V Facility. Fairmont, West Virginia. 2003. <http://sarpresults.ivv.nasa.gov/ViewResearch/289/24.jsp> (19 Jan. 2005).

[8] Bennett, T.L., Wennberg, P.W., *The Use of a Virtual System Simulator and Executable Specifications to Enhance Software Validation, Verification, and Safety Assurance – Final Report*; Software Assurance Research Program Results web site, Fairmont West Virginia, NASA IV&V Facility, June 2004, http://sarpresults.ivv.nasa.gov/ViewResearch/282/32.jsp