Modular Architecture for Fault Identification

Dalton Anderson, Daniel Greer, Benjamin Hutchinson, Kent Lee, Andrew Levandoski, Andrew Mezich, Samuel O'Donnell, Zach Reynolds, Kristyn Sample, Corwin Sheahan, Pol Sieira, Zack Toelkes
University of Colorado at Boulder, Ann and H.J. Smead Aerospace Engineering Sciences, Boulder, CO, 80309

LLAMAS (sateLLite Adcs fault MAnagement System) is a University of Colorado Boulder Aerospace Engineering senior design project where the team is focused on designing a satellite testbed for the purposes of exhibiting and testing a modular software architecture capable of identifying and mitigating faults. Space vehicles are especially susceptible to faults due to the hostile environment of space, therefore a cost effective and general solution for testing a fault management system is desirable. This project aims to design a testbed which will provide the ability to test fault management software in a controlled environment. Testing of the fault mitigation architecture will occur on a physical satellite mockup (MockSat) with one degree of rotational freedom on a low friction air table (TestTable). The MockSat is controlled by an atitude determination and control system (ADCS) consisting of redundant visual sensors, redundant reaction wheels, and a proportional-integral-derivative (PID) controller. Faults are designed to be injected into the ADCS software to mimic fault scenarios found on modern satellites; this provides the opportunity to test a number of fault management algorithms or procedures. The software architecture allows for simple addition of fault detection techniques to decrease the cost associated with testing numerous mission failure scenarios on separate hardware. With fault injection living in software, an operator can provide his or her fault detection techniques, and simply add their code to the fault management library provided with the TestTable. In regards to the detection strategy, LLAMAS team is focused on designing general fault identification strategies for biased sensors by comparing historical to current moving averages. Using residuals, the two can be compared against a threshold determined by the operator, and once this threshold is exceeded for a modifiable amount of time, a fault is triggered. This paper will discuss the effectiveness of the aforementioned fault injection and identification strategies.

Nomenclature

ADCS	=	attitude determination and control system
B_f	=	viscous damping factor
FM	=	fault management
GSU	=	ground station unit
I	=	moment of inertia of reaction wheel
PID	=	proportional, integral, derivative
α	=	angular acceleration of reaction wheel
θ	=	pointing angle of the MockSat
$\Delta\theta_{fine\ sensor}$	=	angle between reference target and current pointing of MockSat as seen by fine sensor
$\Delta\theta_{coarse\ sensor}$	=	angle between reference target and current pointing of MockSat as seen by coarse sensor
ω	=	reaction wheel speed
τ_c	=	commanded torque
τ_f	=	inherent friction torque in reaction wheel
$\tau_{f,c}$	=	magnitude of coulomb friction
$\tau_{f,current}$	=	current sensed friction torque on reaction wheel
$\tau_{f,expected}$	=	expected sensed friction torque on reaction wheel
$\hat{\tau_f}$	=	induced friction

I. Introduction

Faults are inevitable in any complex system, and, if left unchecked, can lead to costly or mission ending failures. The extreme environment of space extenuates the challenges associated with addressing faults in a system. As a result, reliable means of dealing with faults are extremely important to the aerospace industry for satellites and other space vehicles. The ADCS in modern space vehicles are designed to be single fault tolerant, which is characterized by the ability to continue operation in the presence of any one failure. Thus, these ADCS have the ability to identify, contain, and mitigate faults that occur in the system at large[1]. Since single fault tolerant flight software is generally complex, difficult to test, and vehicle specific, a simple, modular system is desirable for rapid testing of flight software configurations. By separating the ADCS software from fault management software, fault testing becomes increasingly modular, allowing for simpler and less costly design.

This paper lays out and presents results for a basic, modular fault management software architecture tested in a physical system. The system will consist of three primary elements: MockSat, TestTable, and Ground Station Unit as seen in Figure 1. The MockSat will mimic the rotational dynamics of the GOES-16 satellite in restricted planar motion, and will be equipped with an ADCS which uses hardware representative of that on the GOES-16 satellite [2]. The MockSat sensor array will include a coarse attitude optical sensor and redundant fine attitude optical sensors. These three sensors will work in conjunction to determine the attitude of the MockSat relative to a reference target. The MockSat will also have redundant reaction wheels to control its angular dynamics, which will be sized to allow the MockSat to replicate the 0.04 Hz ± 10% bandwidth response of the GOES-16 satellite. Redundant fine sensors and reaction wheels will allow the introduction of bias faults into each, mimicking a sensor failure or increased friction in the primary reaction wheel. By observing the MockSat's ability to regain nominal pointing in the presence of such faults, the user will be able to evaluate the fault response of the flight software.

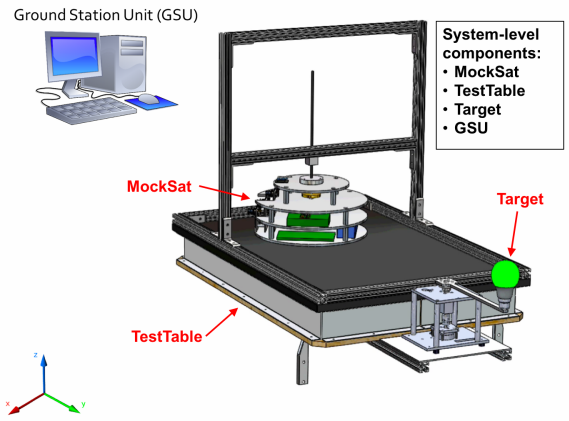

Fig. 1 System Hardware Overview

The software elements onboard involve 3 main modules: the ADCS flight software, the fault injection software, and the fault management software. The ADCS flight software governs the movements of the satellite during nominal operation. Since this software is not the focus of this paper, we will not examine this further.

The fault injection software provides the user with an interface to introduce faults into the MockSat. There will be two types of faults provided: reaction wheel friction faults, and optical sensor offset faults. It's important to note that the software does not physically cause these faults, but instead mimics the effects of these faults on data which are used for fault detection and identification.

The fault management software will provide two things to the system. First, the fault management software onboard the MockSat will be modular in the sense that a user can provide his or her own fault detection algorithm to the system. This is important as it allows cheaper and easier integration of various detection methods to be added to the fault identification software. Second, the fault management system will provide a detection method for use in detecting faults specified by the fault injection system.

This project will be specifically focused on identifying bias faults using a moving average method, which compares current data to a moving average of previous data and compares it to some threshold. This is only one possible detection method, and the modular aspect of the software will allow easy addition of other methods that could be used in conjunction. Along with detection methods, the other modular aspect of this is that it can be used for multiple different sensors.

One example of this bias fault can be seen in reaction wheels. Reaction wheels are used on many space vehicles to control attitude. Any time a component has moving parts there is a greater risk of failure, and the same is true for reaction wheels. These components are driven by motors and if the mechanical bearings are not properly lubricated there are concerns with an increase in friction. This can be modeled in testing by simulating a change in friction that is similar to what would be seen if the reaction wheel was failing and experiencing an increase in friction.

The objective of applying the modular fault management system will be presented in more detail at a high level, after which it will be applied in context for the LLAMAS project. Again, the goal is to develop a modular software architecture that will make testing for faults more financially acceptable while decreasing the time it takes to develop and implement. Results will then be presented to show how this approach was applied to the capstone project.

II. Objectives

As mentioned previously, the goal of the LLAMAS project is to create a fault management testbed. The motivation for the project is to increase the robustness and reduce the financial burden of development and testing of fault management algorithms. In order to create a fault management testbed, the team first needed to create a system for the injection of faults into the testbed. It was desired for this system to replicate real satellite faults as much as possible, and not cause permanent damage to the testbed.

When approaching the task of creating a modular fault management testbed, the team identified two major aspects that could be made modular. The first is the development of a fault detection scheme that works on multiple pieces of hardware, for multiple faults. The idea behind this is to have a common algorithm applicable to a variety of components to detect a fault. Since there are many potential schemes that could be applied to multiple pieces of hardware, this project will focus on biased faults. A biased fault is a fault that produces a consistent offset from nominal operation. As such, the scheme will have to identify each component by its operating conditions, whether it detects a rate, position, etc., and what would be determined as off-nominal operations. These conditions and deviations from optimal are modifiable in the testbed described herein. The second is the ability to easily change between different fault detection schemes. This is desirable because not all faults can be properly detected with a single scheme. The biased faults that are being analyzed by the scheme above is one such scheme. Other examples are schemes that detect faults that produce an effect on the rate of change of data, faults that produce a repeating error on a set interval, or faults that cause the data stream to become unusable. By designing the structure of the software to be able to easily switch between fault detection schemes it allows for a more versatile final system to be deployed. This modularity is also desirable from a testbed perspective as it allows for the rapid testing of multiple fault detection schemes that are being designed in parallel. Figure 2 illustrates these two modular aspects.

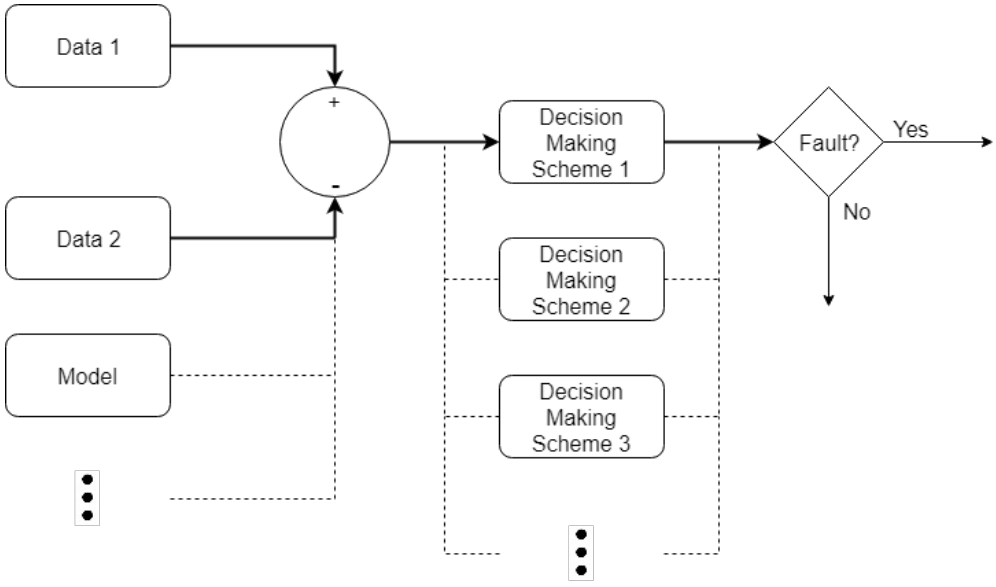

Fig. 2 Decision Making Flow Chart.

On the left, arbitrary data streams (i.e. positional data, historical averages, a predictive model, etc.) are compared, then fed to a decision making algorithm that determines if a fault is present. The decision making scheme can be configured by the user to test different fault management algorithms. The highlighted arrows denote the current state of the fault management software and the dotted arrows represent future additions to the architecture.

III. Methodology

A. Fault Injection

The LLAMAS team desired the fault injection system to inject two types of faults that resembled realistic faults. One of the most common on-orbit faults for space vehicles is an increase in reaction wheel friction, causing off-nominal operation. Therefore, the team designed the fault injection system to simulate an increase in reaction wheel friction. Another possible fault could be sensor bias, due to misaligned sensors or a collision, for example. Therefore, the second type of fault will provide a static offset to the fine attitude sensor onboard the MockSat.

1. Reaction Wheel Fault Injection [3]

During design of this software subsystem, it was desired to make realistic reaction wheel faults. Research revealed that there are three types of friction present in a reaction wheel: coulomb, viscous, and stribeck[4]. Figure 3 shows each of these types of friction graphically. Coulomb friction is analogus to static friction. Viscous friction grows linearly with the angular speed of the reaction wheel, and stribeck friction is the extra friction present when the reaction wheel is being started from rest.

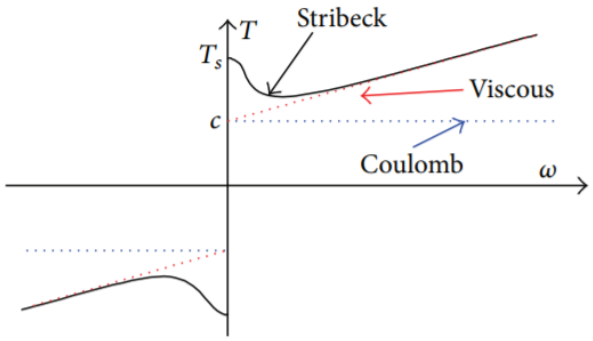

Fig. 3 Different types of reaction wheel friction

More research demonstrated that the primary cause of failure due to friction with on-orbit reaction wheels is due to an increase in coulomb friction. This manifests in a static increase of friction, as demonstrated in Figure 4 [5]. Empirical data on reaction wheel failures is difficult to find, as most of the information is proprietary and thus, not publicly available. However, the team found a report on GlobalStar satellites that revealed that they typically see catastrophic failure from their reaction wheels when the reaction wheel sees five to six times the nominal coulomb friction [5]. As such, it is desired to create an injection system which will mimic the presence of an increase in reaction wheel friction on the same order of magnitude.

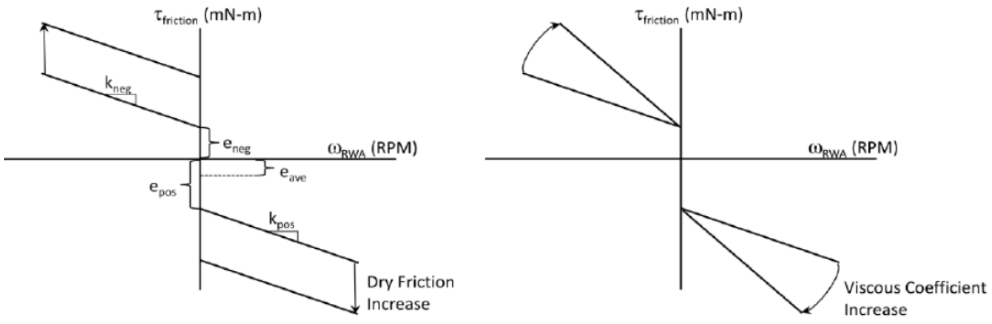

Fig. 4 Examples of increases in coulomb and viscous friction in reaction wheels

Modeling the dynamics of a controlled reaction wheel is necessary to characterize the manner in which an increased

reaction wheel friction will be mimicked, and is presented here. Figure 5 displays the dynamics for a reaction wheel with friction present.

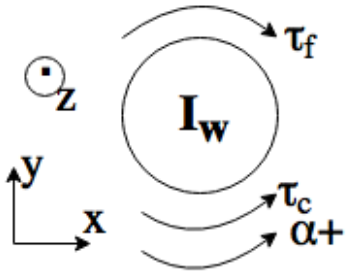

Fig. 5 Reaction wheel dynamics

Newton's second law gives eq. (1):

$$\sum \tau = I\alpha \tag{1}$$

where τ are the torques present, I is the moment of inertia (MOI) of the reaction wheel, and α is the angular acceleration of the reaction wheel. From this relation, the dynamics of the system are derived, as seen in eq. (2):

$$\tau_c - \tau_f = I\alpha \tag{2}$$

where τ_c is the commanded torque and τ_f is the inherent friction in the reaction wheel. It can then be modeled τ_f as a function of reaction wheel speed, as given by eq. (3):

$$\tau_f = f(\omega) = B_f\omega + \tau_{f,c} \tag{3}$$

where B_f is the viscous damping factor and $\tau_{f,c}$ is the magnitude of coulomb friction. An increased friction function can now be defined in terms of the elements of the nominal friction. Equation (4) displays this induced friction function:

$$\hat{\tau}_f = \hat{f}(\omega) = B_f\omega + 6\tau_{f,c} \tag{4}$$

where $\hat{\tau}_f$ is the induced friction created by the fault injection system. Note that this function is simply the nominal friction, but with an increase in static friction of six times, which corresponds to an imminent reaction wheel failure, according to research. Now that the ideal behavior of the desired induced friction is defined, determining the design for how this friction will be injected is next.

In order to make it appear as if there is an increase in friction, the control torques will be modified after they have been calculated by the control law, but before they are sent to the reaction wheel's motor controller. Figure 6 shows this design. Inside the yellow box labeled 'Reaction Wheel' are the reaction wheel's natural dynamics.

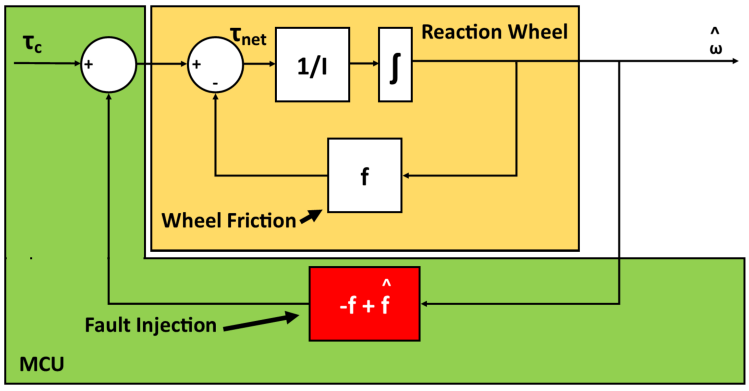

Fig. 6 Reaction wheel fault dynamics

The green box labeled 'MCU' indicates the domain of software aboard the microcontroller unit. The red box labeled 'Fault Injection' indicates the fault injection algorithm. This algorithm will require feedback of the reaction wheel speed. From this reaction wheel speed, the fault injection system can calculate the nominal friction. The nominal friction will be characterized into an expected friction as a function of ω, before the real-time operation of the MockSat, via testing of the reaction wheel over its operational range of angular speeds. Then, the reaction wheel's angular acceleration is approximated by numerically differentiating the output wheel speed. Finally, the nominal friction torque can be determined by finding the difference between the commanded torque and the $I\alpha$ of the reaction wheel. This data will be stored on the MockSat for use in both the fault injection and management systems. More detail on fault management appears later in this report.

The fault injection system will then calculate and subtract this nominal friction and add in the induced friction torque. This net effect will be subtracted from the commanded torque and passed to the motor controller. Equation (5) shows this behavior from the perspective of the output angular acceleration of the reaction wheel:

$$\alpha = \frac{\tau_c - (-f(\omega) + \hat{f}(\omega)) - f(\omega)}{I} = \frac{\tau_c - \hat{f}(\omega)}{I} \tag{5}$$

Thus, it can be seen that the net dynamics of the system, in theory, incorporate only the induced friction and the commanded torque, creating the behavior that is desired for simulating increased reaction wheel friction.

The process of characterizing the reaction wheel's nominal friction can not be completed until after the motor has been chosen and obtained for the final design. The process of characterizing reaction wheel friction has been applied to data from the spring 2017 ASEN 3112 - Introduction to Aerospace Structures reaction wheel lab. From these data, the nominal friction was calculated and an induced friction function was created based off of this nominal data. This induced friction function was of the same form as in eq. (4), with the addition of statistical noise based on a normal distribution with standard deviation equal to the standard deviation of the nominal friction's residual from the nominal friction function.

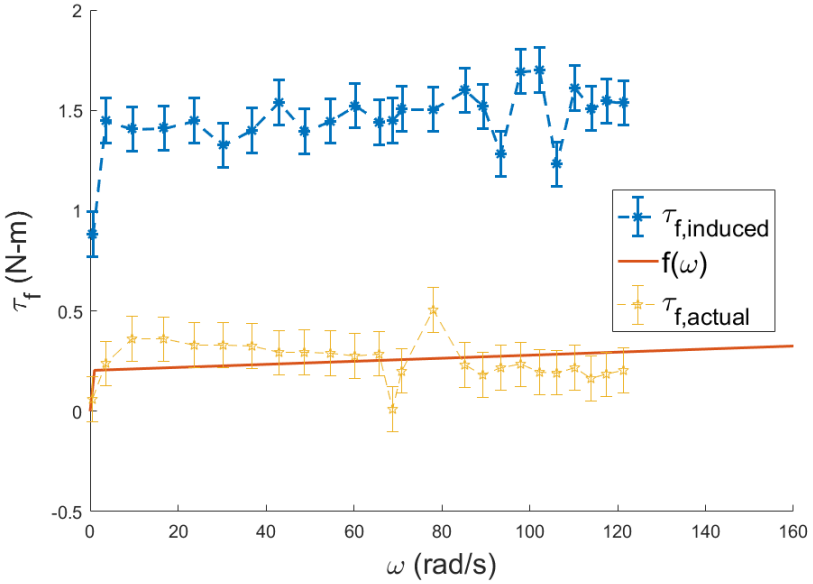

Fig. 7 Reaction wheel friction from ASEN 3200 spin module data and induced friction

2. Sensor Bias Fault Injection

The fine sensor will be faulted by adding a constant $\Delta\theta$ every time the MCU pulls information from the sensors. This will ensure that the satellite constantly senses the reference target position a set angle off of center, whether it be behind or ahead of the reference target. The ADCS will respond by actuating away from the true reference target until completely biased, and continue its nominal motion.

B. Fault Management

Now with faults injected into the system, fault management has the opportunity to detect and classify these faults.

1. Bias Detection Method

Specifically for the MockSat, both the primary fine sensor and primary reaction wheel will be faulted to test the effectiveness of the biased fault detection method. First, the method will be introduced in general then related specifically to the MockSat system.

What exists in the 'Decision Making' box shown in Figure 2, is a fault detection method capable of deciding whether the fine sensor or reaction wheel has been biased. Plainly, current data is compared to a moving average of historical data and continually compared to a configurable threshold value.

$$X_{Threshold} > \left| X_N - \frac{\sum_{i=1}^{N} X_i}{N} \right|$$

X is the data that is being monitored, N is the length of the moving average, and the $X_{Threshold}$ is the value that the difference is being compared to.

X when monitoring the reaction wheel is the difference between $\tau_{f,current}$ and $\tau_{f,expected}$, where $\tau_{f,expected}$ was characterized for all reaction wheel speeds using eq. (3).

$$X = \delta\tau_f = \left| \tau_{f,current} - \tau_{f,expected} \right|$$

Equation (3) will be used to calculate the $\tau_{f,current}$ each iteration through the ADCS loop to then be compared to the characterized data mentioned before.

X, when monitoring the fine sensor is the difference between the $\Delta\theta$ sensed by the fine sensor and coarse sensor.

$$X = \delta\Delta\theta = \left| \Delta\theta_{fine\ sensor} - \Delta\theta_{coarse\ sensor} \right|$$

$\Delta\theta$ in both cases is a measurement of the necessary satellite rotation to directly point at the reference target. With the use of a redundant sensor, the coarse sensor, the bias is compared to detect faults in the fine sensor.

Historical data is stored for both the reaction wheel and the fine sensor to keep a moving average to compare against current data and monitor their difference versus a set threshold. Anything greater than the threshold is indicative of an actual fault and will trigger recovery. To prevent instances such as a blip in data causing the threshold to be crossed, the fault must occur for a set duration of time and checked against a timer.

To proof-of-concept the bias fault detection method, the algorithm was employed in MATLAB using sample data for both the reaction wheel friction and the orientation sensors. First, the reaction wheel's sensed friction is tested using data from a system similar to the MockSat; a reaction wheel commanded by a GSU with one degree of freedom. From this system the nominal friction in the reaction wheel is characterized, then a characteristic friction function and standard deviation are calculated to predict the expected friction at any given reaction wheel speed.

IV. Results

The expected and nominal friction in Figure 8 are both based off the expected friction function, with some random noise according to the magnitude of the standard deviation of the nominal friction from the expected friction function. The wheel speeds at each time step are a random, uniformly distributed set of data ranging from 0 to 10 rad/s. This data is ran through the threshold detection algorithm, which accordingly did not detect any faults, as seen in Figure 8.

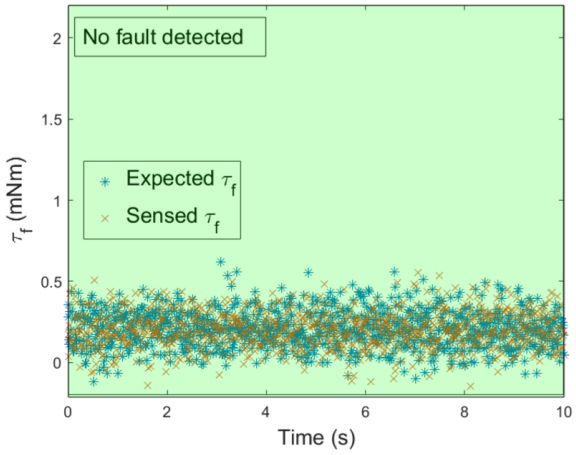

Fig. 8 System showing no fault in the friction of reaction wheel.

The faulty friction data is created by increasing the coulomb friction six times larger, five seconds into the test. The results are shown in Figure 9. Again, the data is ran through the fault detection algorithm, and the results are indicated by the red colored box on the right. Note there is a delay between when the fault is injected and detected. Due to the moving average calculation, there is lag in detection because more thorough historical data is necessary to trigger the fault.

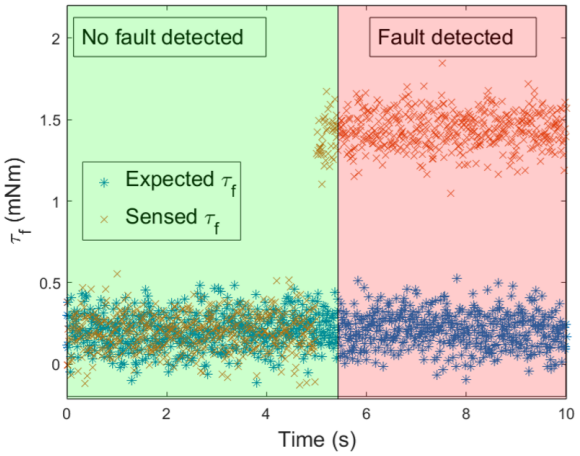

Fig. 9 System showing a fault is identified in the reaction wheel friction.

Similarly to the reaction wheels, sample faulty data were created to test the threshold method on what some anticipated faulty data will look like. The results of this can be seen in Figure 10. To create these data, it was assumed that there was a larger noise for the coarse sensor than the fine sensor, and used the sum of each sensor's standard deviation from zero as a threshold. Again, the delay in detection can be observed, and for the same reason as with the reaction wheel.

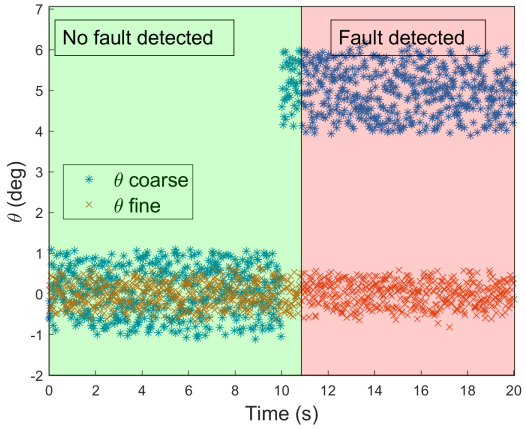

Fig. 10 System showing a fault is identified in the fine orientation sensor.

One can see that Figures 9 and 10 are qualitatively identical despite coming from different simulated sensor/actuator data. In fact, the source of the data is arbitrary and thus illustrates the modularity of this fault identification architecture - specifically the moving average detection scheme for this type of fault. To further generalize, additional data sources or decision making blocks can be easily added as shown in Figure 2.

V. Conclusion

When putting satellites in space, there is added complexity involved to fix something that fails. Therefore, it is desirable for satellites to be single fault tolerant and capable of completing the mission in the presence of any single fault. Testing while the satellite is still on the ground is a crucial phase of mission planning and it would be great if every failure scenario could be thought of, tested against, and mitigated, but that is not reasonable. Testing takes a lot of time and large scale tests are expensive. Scenarios that are testable in software reduce the costs and are more modular to provide rapid testing of several components. Common faults include increased friction in reaction wheels and discrepancies in sensors, and these are great candidates for software testing as these scenarios can be mimicked by data biasing, which has been presented here.

Acknowledgments

The fault management team would like to acknowledge that the reaction wheel data was taken in the ASEN 3112 course given at University of Colorado's Undergraduate Aerospace curriculum.

References

[1] Contributors, W., "Fault Tolerance," , 2017. URL https://en.wikipedia.org/wiki/Fault_tolerance, [Online; accessed 11-March-2018].

[2] Jim Chapel, T. B. S. W. B. C. T. R. D. G. D. F. A. K., Devin Stancliffe, "Guidance, Navigation, and Control Performance for the GOES-R Spacecraft," 2014.

[3] Team, L., "LLAMAS Fall Final Report," , 2017. [Submission Report].

[4] Carrara, V., "Estimating Friction Parameters in Reaction Wheels for Attitude Control," , 2013. URL https://www.researchgate.net/publication/258391329_Estimating_Friction_Parameters_in_Reaction_Wheels_for_Attitude_Control.

[5] Hacker, J. M., "Reaction Wheel Friction Analysis Methodology and On-orbit Experience (Invited)," , 2014. URL https://www.researchgate.net/publication/269163584_Reaction_Wheel_Friction_Analysis_Methodology_and_On-orbit_Experience_Invited.