Machine Learning for Reaction Wheel Fault Detection Using Simulated Telemetry Data

J. Vaz Carneiro*, H. Schaub† and M. Lahijanian‡
University of Colorado Boulder, Boulder, CO

K. Borozdin§ and K. Lang¶
Verus Research, Albuquerque, NM

Autonomous fault detection in satellites is an enticing proposition that will be essential to prevent catastrophic failures and guarantee the success of future missions. The current approach relies on highly qualified human technicians to search for anomalous behaviour in large amounts of telemetry data. This method raises some challenges; as the complexity of missions increases, the ability for those technicians to detect faults declines. Furthermore, the growing number of missions with thousands of satellites in complex constellations makes traditional fault detection techniques unfeasible. Machine learning algorithms can be used to solve these issues. They require less human interaction and can be used in large operations with multiple active satellites.

This paper proposes supervised and unsupervised learning algorithms to detect whether a fault has occurred and when it happened. The focus is to study faults related to reaction wheels and its subsystems, such as wheel speed encoders. A high-fidelity simulation is developed using the Basilisk software tool and the produced synthetic telemetry stream is fed into the machine learning algorithms for fault detection. The approaches used proved successful in detecting faults and underlined how some data parameters are more useful for fault detection than others.

I. Introduction

Fault detection through monitoring telemetry data is a crucial operational task. It allows for the identification of failures that can compromise the mission and, in some cases, can prevent catastrophic mission failure through the use of "safe modes" that limit the spacecraft's operations [1]. In most cases, this process requires humans-in-the-loop for analyzing potential false positives and negatives, and for making decisions on the spot [2]. While this has always been the industry standard, having humans analyze large amounts of data brings its own set of drawbacks. Detecting satellite faults requires highly qualified technicians and, even then, some faults have such a small impact on telemetry that they cannot be detected (false negatives). Moreover, as missions become more complex with constellations of hundreds or thousands of satellites, it becomes difficult to monitor every single satellite's telemetry data.

The use of machine learning aims to solve the drawbacks described above. It removes the dependency of humans on detecting faults and, by training on nominal data, not only better detects small faults that are unnoticeable to technicians but also disregards false fault flags, improving both false positives and false negatives. It is also easily scalable, addressing the challenge of having a large number of spacecraft in orbit. The use of machine learning tools to detect changes in expected behaviors has shown promising results in recent years [3].

The effectiveness of any machine learning algorithm heavily relies on the training process. By simulating telemetry data, the algorithms can be fed virtually unlimited amounts of data to train on, improving their performance in detecting faults. Another major benefit compared to using real-world data is that the exact type of the fault is known *a priori*, along with the time it happened. As the truth is know, debugging is facilitated and an easy metric for how the algorithm

*Graduate Research Assistant, Ann and H.J. Smead Department of Aerospace Engineering Sciences, University of Colorado, Boulder, 3775 Discovery Drive, Boulder, CO, 80303

†Professor, Glenn L. Murphy Chair in Engineering, Ann and H.J. Smead Department of Aerospace Engineering Sciences, University of Colorado, Boulder, 429 UCB, Colorado Center for Astrodynamics Research, Boulder, CO, 80303. AAS Fellow, AIAA Fellow.

‡Assistant Professor, Ann and H.J. Smead Department of Aerospace Engineering Sciences, University of Colorado, Boulder, 429 UCB, Research & Engineering Center for Unmanned Vehicles, Boulder, CO, 80303.

§Senior Machine Learning Engineer, Verus Research, Albuquerque, NM

¶Technical Director, Space and Autonomy, Verus Research, Albuquerque, NM

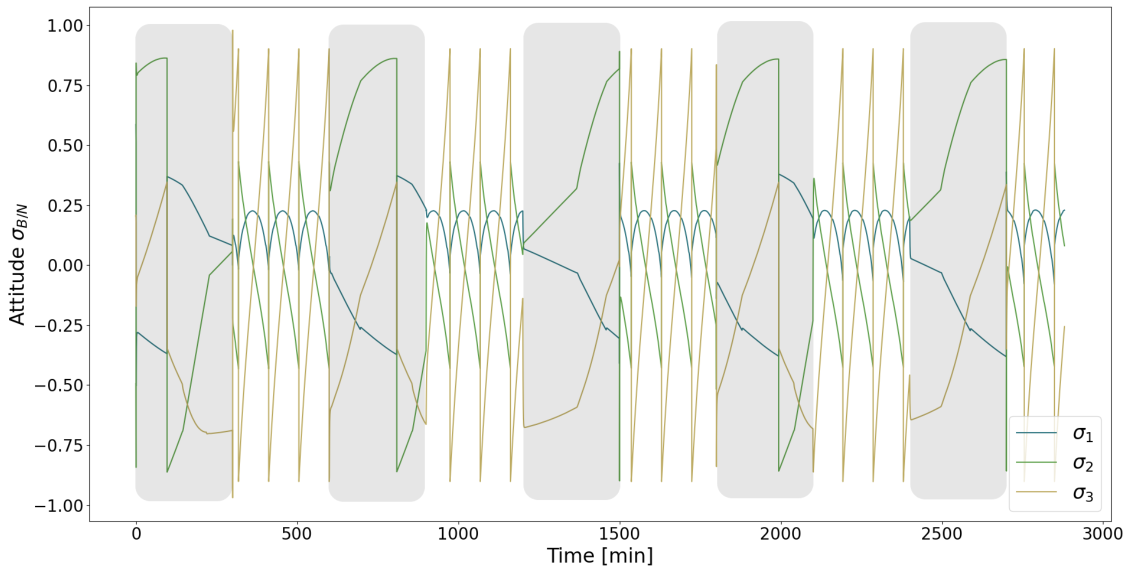

Fig. 1 Attitude parameters for a 2-day nominal run. The Sun-pointing mode is active in the shaded regions, and the Earth-pointing is active everywhere else. The mode changes are noticeable every 5 hours.

is performing exists. Finally, with simulated telemetry data, machine learning algorithms can be pre-trained before the satellites go into orbit. Without this, learning would have to be done during the first phase of the mission while the spacecraft is already in orbit, which means no faults could be detected at the beginning of the mission.

This paper aims to study unsupervised and supervised learning techniques being applied to simulated telemetry data, with the aim of detecting faults in a satellite's reaction wheels. No effort is made towards identifying which fault has occurred. Moreover, decision making given a fault detection is also outside the scope of this paper. This work is divided into two parts. First, the entire simulation is run using the Basilisk [4] software tool.* The corresponding data is processed and organized in a suitable format to then be fed into the machine learning tools. Then, the algorithms evaluate each data set and yield a diagnosis, detecting when the fault is likely to have happened. Inertia matrix dispersion is also tested, to see how the algorithms fare against over-fitting.

II. Problem Statement

This work focuses on building a software framework that generates telemetry data, which is then fed into a machine learning algorithms for fault detection. To that end, a simulation framework is created that acts as a digital twin to a real spacecraft. The objective is to mimic a real spacecraft as close as possible, and so the most critical aspects of a spacecraft, such as the attitude control system, power and battery modules are simulated to see how these variables react to faults.

The simulated CubeSat is inspired by the VPM satellite [5]. From its publicly-available TLE description, VPM's orbit is described as a quasi-circular low Earth orbit with an inclination of 51.6°, argument of periapsis 137.7° and longitude of the ascending node of 357.7°. The initial true anomaly is randomized for each run.

The mission objective is to do science on Earth, and so two attitude guidance modes are simulated, changing between each other every five hours. Optimizing the mode change is beyond the scope of this paper. The first mode corresponds to Earth Nadir pointing, which simulates the spacecraft aiming its instruments directly at Earth. The second is a Sun pointing mode, where the spacecraft points its solar panels at the Sun. The change in attitude can be seen in Figure 1, where Modified Rodrigues Parameters (MRPs) [6] are used to describe the spacecraft's attitude between the body frame $\mathcal{B}$ and the inertial frame $\mathcal{N}$.

*`https://hanspeterschaub.info/basilisk`

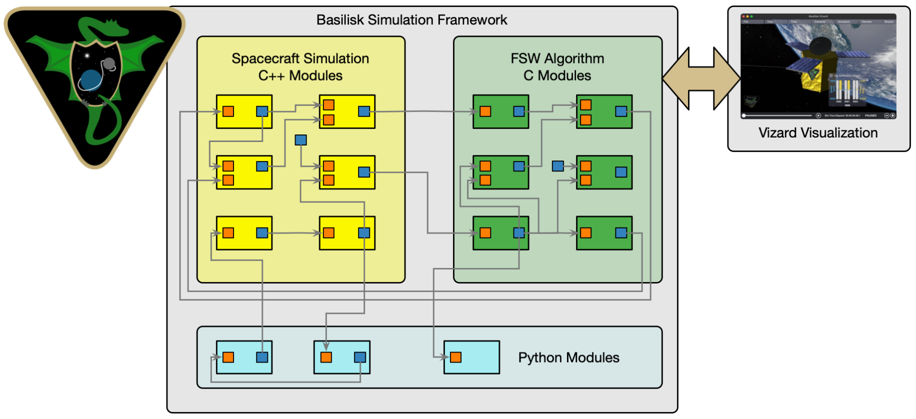
Basilisk Simulation Framework
Spacecraft Simulation C++ Modules
FSW Algorithm C Modules
Python Modules
Vizard Visualization

Fig. 2 Sketch of Basilisk's structure. Different modules are connected to each other through a messaging system. Visualization is done through Vizard.

A. Basilisk Simulation Framework

Basilisk [4] is a flight-proven modular mission simulated framework and it is used to set up the high-fidelity simulation. Its modular nature allows for simple integration of complex simulation tasks, such as power generation and consumption, temperature modelling or orbital perturbations. An example graph of how Basilisk modules connect is displayed in Figure 2.

For this work, a new thermal module is created, as an increase in the temperature of the reaction wheels is expected when a friction fault occurs. The module takes the power required to spin the reaction wheels and converts it to dissipated energy using the efficiency of the motors used. This dissipated energy, along with the energy created by the friction within the reaction wheel's bearing, contribute to an increase in temperature. To counteract this, the environment's temperature is set to be constant and equal to 20°C. When the temperature of the reaction wheel goes above this value, it starts dissipating some energy towards the environment, decreasing the overall temperature. The mechanical power P_{mec} is given by:

$$P_{\text{mec}} = \Omega \cdot u_s$$

where Ω is the motor angular velocity and u_s represents the applied torque. Given the mechanical efficiency η, we can convert mechanical power into thermal power as follows:

$$P_{\text{loss}} = P_{\text{mec}} \frac{1-\eta}{\eta}$$

Similarly to the mechanical power, the friction dissipation is given by:

$$P_f = \Omega \cdot \tau_f$$

where τ_f represents the friction torque. The absolute value of these two terms added together represents the thermal power generation. As for the power dissipation, we assume a temperature gradient between the motor and the surrounding environment. The thermal power dissipation is calculated using the temperature difference between the motor and the environment, scaled by the ambient's thermal resistance R_{ambient}:

$$P_{\text{dissipation}} = (T_k - T_{\text{ambient}})/R_{\text{ambient}}$$

The subscript k in the temperature implies that it is the motor's temperature at time k. For the final calculation, we need to convert power into heat. This is done through a simple Euler integration:

$$Q = P\Delta t$$

where Q represents the heat, P is the power and Δt is the integration time interval, which is the same as the simulation time step. Finally, the new motor temperature can be calculated using the temperature at the previous time step plus the

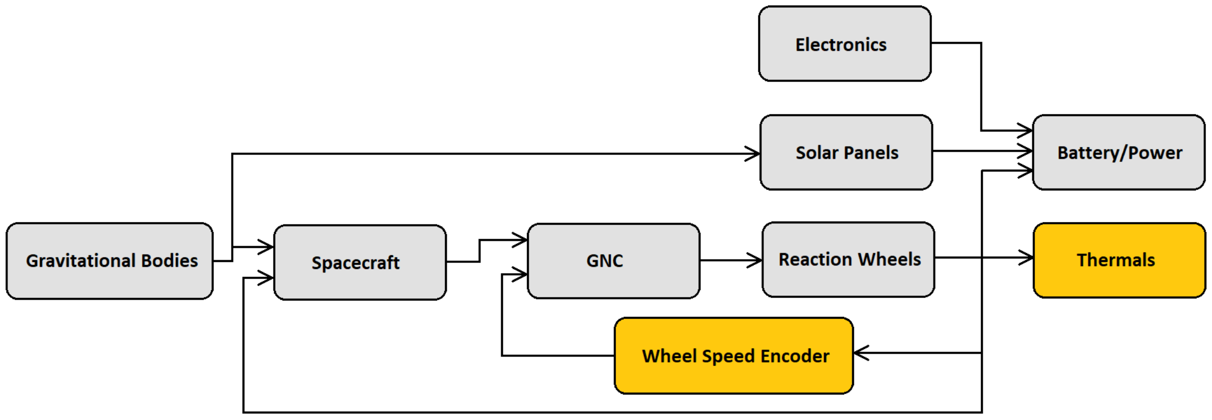

Fig. 3 Simple graph of the modules used in the simulation. The grey boxes represent general subsystems, whereas the yellow boxes highlight the modules added for this work.

net heat scaled by the motor's heat capacity:

$$T_{k+1} = T_k + \Delta t \frac{P_{\text{generation}} - P_{\text{dissipation}}}{C_{\text{motor}}} = T_k + \Delta t \frac{|P_{\text{loss}}| + |P_f| - P_{\text{dissipation}}/R_{\text{ambient}}}{C_{\text{motor}}}$$

where, again, the k+1 subscript denotes the temperature at the next time step and C_{motor} corresponds to the motor's heat capacity.

To simulate reaction wheel speed measurements, a new encoder module is also designed. This module takes the real value of the reaction wheel speeds and outputs a measured set of values. It is in this module that some of the faults are set, such as a wheel speed being a stuck at a specific value, or the sensor not working and displaying a value of zero.

Another change made has to do with the ability to change reaction wheel properties on the fly. To add friction faults at a random point in the simulation, the ability to change reaction wheel properties, such as friction coefficients, is added to the software framework. A simple diagram of the Basilisk simulation showing existing modules (gray) and new modules (yellow) is pictured in Figure 3.

Another important variable is power. Encoder faults make the control system ask for different torques, which affect the reaction wheel's power consumption. A drastic increase in friction also affects power consumption, as the ADCS system has to overcome the increase in friction to provide the same torque to the wheels. To that end, a comprehensive power module is simulated, taking into account the power generated by the solar panels and drawn by the reaction wheels and electronics. The solar panel module takes into account incidence angle and eclipse zones when the Earth is between the spacecraft and the Sun. Therefore, even when the spacecraft is in Sun-pointing mode, the power generated by the solar panels is 0 when the spacecraft is behind the Earth relative to the Sun.

Attitude control is accomplished using a set of four reaction wheels. A thorough description of reaction wheel dynamics can be read in [7]. The control law used is guaranteed to converge [6] and the control torque $\mathbf{L}_r$ is given by

$$\begin{aligned}\mathbf{L}_r = -K\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}} - [P]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{R}} - [I_{\text{RW}}](\dot{\boldsymbol{\omega}}_{\mathcal{R}/\mathcal{N}} + \boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} \times \boldsymbol{\omega}_{\mathcal{R}/\mathcal{N}}) - \mathbf{L} \\ + \boldsymbol{\omega}_{\mathcal{R}/\mathcal{N}} \times ([I_{\text{RW}}]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} + [G_s]\mathbf{h}_s)\end{aligned} \tag{1}$$

where $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$ denotes the attitude between the $\mathcal{B}$ frame and the $\mathcal{R}$ frame (the reference frame), $\boldsymbol{\omega}_{\mathcal{B}/\mathcal{R}}$ denotes the angular velocity between the $\mathcal{B}$ frame and the $\mathcal{R}$ frame, $\dot{\boldsymbol{\omega}}_{\mathcal{R}/\mathcal{N}}$ denotes the inertial derivative of the angular velocity between the $\mathcal{R}$ frame and the $\mathcal{N}$ frame and $\mathbf{L}$ is a know external torque. The spacecraft inertia matrix without the inertia of the reaction wheels about their corresponding spin axis is $[I_{\text{RW}}]$. $[G_s]$ is a matrix that consists of each reaction wheel's spin axis $\hat{\mathbf{g}}_{s_i}$ as its rows. Finally, $\mathbf{h}_s$ is given by

$$h_{s_i} = I_{W_{s_i}}(\hat{\mathbf{g}}_{s_i}^T \boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} + \Omega_i) \tag{2}$$

with $I_{W_{s_i}}$ being the reaction wheel's spin axis inertia and Ω_i being the reaction wheel's angular velocity.

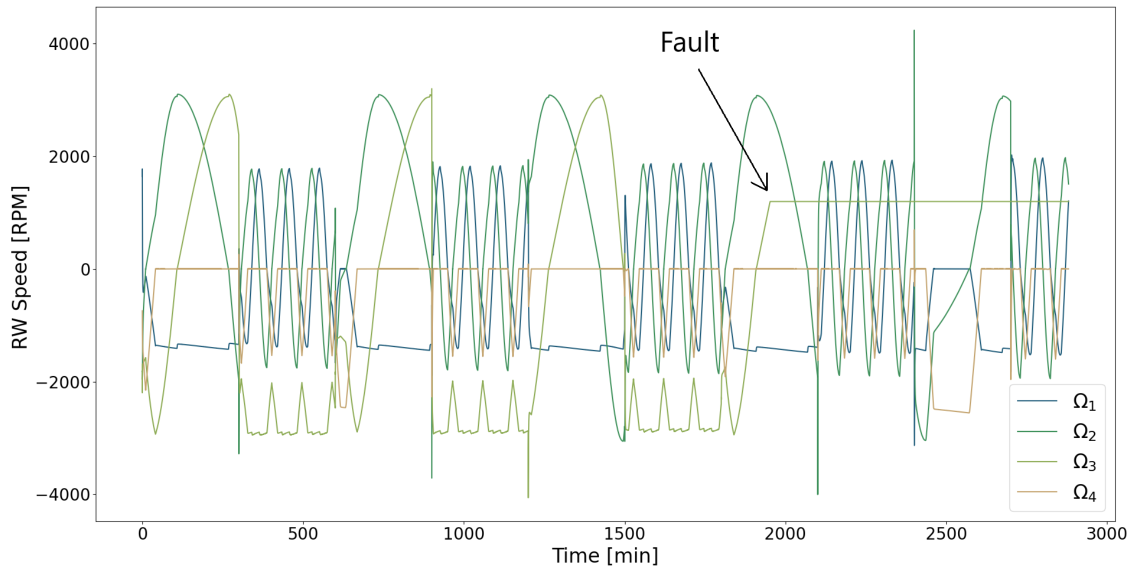

Fig. 4 Effect of a stuck encoder fault on reaction wheel speeds. The fault happens with reaction wheel 3 (light green line) at about 1950 minutes.

B. Fault Injection

Two possible faults are simulated: a reaction wheel friction increase (due to a damaged bearing, for example) and an encoder fault (due to an electrical or software failure).

The friction fault corresponds to a 5 to 20 times increase in the nominal friction of the reaction wheel. This friction acts as a torque that the wheel motor has to overcome, which means the friction increase can be thought of as an unmodeled torque. This external torque is expected to increase the wheel's applied torque and to introduce a steady-state error on the attitude parameters [6].

The wheel speed encoder can fail in one of two ways: the signal can either get stuck and the current input is ignored, or the signal can be turned off, outputting a zero angular velocity. In both cases, the outputted angular velocity of the impacted wheel is constant. As the encoder information is fed into the control loop, it will impact the control motor torque for each reaction wheel. The control law has a feed-forward term that is proportional to the product between the wheel inertia and the wheel speed, as introduced in Equation (1) and explained in Equation (2). Therefore, the control module having the wrong wheel speed information will have an impact on the commanded torque output. An example of an encoder fault is given in Figure 4.

While several days can be simulated, the fault only occurs on the last day. The time of the fault on that last day is randomized, as is the affected reaction wheel. This is done in accordance to how the learning algorithms are set up. The algorithms use the first few days of data, which do not have any fault present, to learn what the nominal behaviour of the spacecraft is. Several parameters such as attitude, attitude rates, reaction wheel speeds and temperatures, power consumption and commanded wheel torques are analyzed and grouped together. The algorithms then use this learned behavior to detect the fault on the last day of the simulation.

C. Model Uncertainty

Another important aspect to consider is the tendency of machine learning algorithms to over-fit towards the training data provided. This happens when the algorithm adapts to the features of the training set too well, and does not extrapolate well on data points outside that set. Different inertia properties mean that closed-loop control response is different, and over-fitting can lead to a greater number of false positives because the algorithms do not expect these slight changes in attitude tracking.

To see how well the algorithms performed under uncertainties, for some of the training runs the inertia matrix entries were dispersed from their nominal values. This dispersion consisted in multiplying each diagonal entry of the inertia matrix by a random value taken from a uniform distribution between 0.9 and 1.1. This means that each entry could take any value between 90% and 110% of its nominal value. These dispersions impact attitude control performance parameters such as settling times and requested torques. The algorithms were then applied to a testing set where the inertia matrix assumed its nominal value.

III. Theoretical Background

Fault detection in satellite telemetry is an example of a broader range of problems known as anomaly detection. Generally speaking, anomaly detection refers to any algorithm that finds the outliers of a data set, which are items that don't belong. Anomaly detection has numerous applications for an extensive variety of practical and important problems. As the amount of data continues to grow exponentially, machine learning techniques play an increasingly more important role for anomaly detection in large data sets.

A typical example of an anomaly is a point that lies outside of a "normal" region in a variable space. Detecting this type of anomaly is rather trivial as long as one is able to select appropriate variables and define well the boundaries of the "normal" region. For the goals of this study, we are also interested in another type of anomaly, where the anomalous point may lie within the "normal" region, but this point or a sequence of points break a pattern established by previous observations. Detecting such anomalies is a more difficult task, and requires application of different class of algorithms as discussed in greater detail below.

As is the case for all machine learning techniques, algorithmic anomaly detection can be based on supervised or unsupervised learning. Supervised learning is based on the use of preliminary labeled data. For supervised anomaly detection we need to have a large sample of anomalies that we want to detect. In practice, this large sample may not be available. Developing methods for fault detection in satellite telemetry has the drawback of few examples of faults, and each fault tends to be unique. Having realistic simulations is one way to overcome this problem. Otherwise, unsupervised learning methods are needed to analyze unlabeled data sets. For anomaly detection, it is assumed that unlabeled data points are normal, and the algorithms learn their probabilistic distribution and identify anomalies as significant deviations from this distribution. This approach works very well as long as the distribution of the nominal points does not change, and as long as anomalies lie outside the normal regions in the space of variables.

Particular algorithms used for fault detection are discussed below. Each method leverages existing machine learning algorithms, using both supervised and unsupervised learning techniques. Unsupervised learning, which does not require labeled data, is better suited to real satellite telemetry, whereas labeled data and supervised learning may produce more accurate detection methods and fully leverage all information available in the simulated data.

A. Gaussian Mixture Model (GMM)

The Gaussian Mixture Model [8] is a type of unsupervised clustering algorithm. Unlike the popular k-means algorithm, which provides hard assignments, GMM provides soft assignments, where each data point is generated by a parametric probability density function represented as a weighted sum of Gaussian component densities. For anomaly detection, we use this set of Gaussian distributions to represent the probabilistic distribution of simulated data points that are assumed to be nominal. We then identify points located outside of the modeled distributions (i.e. having low probability assuming the model is correct), and label these points as anomalies.

The GMM is a weighted sum of M component Gaussian densities given by the following equation:

$$\mathrm{p}(\mathbf{x}|\lambda) = \sum_{i=1}^{M} \mathrm{w}_i g(\mathbf{x}|\boldsymbol{\lambda}_i, \boldsymbol{\Sigma}_i), \tag{3}$$

where $\mathbf{x}$ is an N-dimensional data vector, w_i are the mixture wights that must sum to 1 ($\sum_{i=1}^{M} \mathrm{w}_i = 1$), and $g(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ are the component Gaussian densities. Each function g is an N-dimensional Gaussian distribution with mean vector $\boldsymbol{\mu}_i$ and covariance matrix $\boldsymbol{\Sigma}_i$ given by:

$$g(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{N/2}|\boldsymbol{\Sigma}_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right)$$

The GMM is parameterized by the mean vectors, covariance matrices and mixture weights of each of the component densities. These variables are grouped into the λ variable we see in equation (3), and so we can write $\lambda_i = \{\mathrm{w}_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$ for $i = 1, ..., M$.

The parameters λ are usually estimated using the maximum likelihood estimation principle. The aim is to find the set of parameters λ that maximize the likelihood of the GMM given the training data. The parameter estimates can be obtained iteratively using the expectation-maximization (EM) algorithm.

Beginning with an initial model λ, the goal is to estimate a new model $\bar{\lambda}$ such that $p(X|\bar{\lambda}) > p(X|\lambda)$, where $X = \{\mathbf{x}_1, ..., \mathbf{x}_T\}$ represents the sequence of T training vectors. The new model then becomes the initial model for the next iteration until convergence is reached. On each EM iteration, the following re-iteration formulas are used which guarantee a monotonic increase in the model's likelihood value:

$$\bar{\mathrm{w}}_i = \frac{1}{T}\sum_{t=1}^{T} \Pr(i|\mathbf{x}_t, \lambda)$$

$$\bar{\boldsymbol{\mu}}_i = \frac{\sum_{t=1}^{T} \Pr(i|\mathbf{x}_t, \lambda)\mathbf{x}_t}{\sum_{t=1}^{T} \Pr(i|\mathbf{x}_t, \lambda)}$$

$$\bar{\sigma}_{ij}^2 = \frac{\sum_{t=1}^{T} \Pr(i|\mathbf{x}_t, \lambda)\mathbf{x}_t^2}{\sum_{t=1}^{T} \Pr(i|\mathbf{x}_t, \lambda)} - \bar{\mu}_{ij}^2 \qquad (4)$$

In the last equation, it is assumed that each component's covariance matrix is diagonal, and the j-th entry of the i-th component is given by σ_{ij}^2. Therefore, we can represent the covariance matrix $\boldsymbol{\Sigma}_i$ by a vector of its diagonal components $\boldsymbol{\sigma}_i^2$. Given this, in equation (4), σ_{ij}^2 and μ_{ij} represent the an arbitrary entry (the j-th one) of the $\boldsymbol{\sigma}_i^2$ and $\boldsymbol{\mu}_i$ vectors. Finally, the *a posteriori* probability for component i is given by:

$$\Pr(i|\mathbf{x}_t, \lambda) = \frac{\mathrm{w}_i g(\mathbf{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{k=1}^{M} \mathrm{w}_k g(\mathbf{x}_t|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$

B. Support Vector Machines

A Support Vector Machine (SVM) is a popular type of machine learning algorithm widely used for classification. Generally speaking, the algorithm defines a linear decision surface (hyperplane) in a multi-dimensional feature space. Special properties of the decision surface ensure high generalization ability of the learning machine. The idea behind the support-vector network was initially implemented for the restricted case where the training data can be separated and then extended to non-separable training data. With separable data, the algorithm is maximizing the distance from the closest points to the hyperplane. This distance is called margin. For non-separable data, SVM tolerates some misclassifications (points on the wrong side of the decision hyperplane). The amount of tolerance is represented by a hyper-parameter of SVM algorithm. Additional flexibility is provided to SVM by a kernel that can transform existing features, and is equivalent to having a non-linear decision boundary. Some popular kernels include 'polynomial', 'radial basis', and 'sigmoid'. By tolerating some misclassifications (this concept is called 'soft margin'), and also using kernels SVM is able to define wiggly decision boundaries that can efficiently handle linearly non-separable data sets.

SVM is a classical supervised learning algorithm and its standard implementation requires labeled data. A variation of SVM used for anomaly detection is called one-class SVM [9] and is based on the assumption that all data used for training are 'normal' and do not contain any anomalies. The anomalies are then identified as the points lying outside of the 'normal' region identified during the training. This approach avoids the requirement of having labels for the data, as all data points are assumed belonging to the same class and therefore having the same label. While one-class SVM is conceptually similar to a standard SVM, instead of a hyperplane it uses a hypersphere to encompass all the data points used for training. The concept of margin is then replaced by the concept of the hypersphere radius, i.e. instead of looking for the largest possible margin, the algorithm is looking for the smallest possible radius of the hypersphere. As in practice even the training data set can be contaminated with the anomalies, it often makes sense to allow for a few margin violations – this is similar to the earlier discussed concept of 'soft margin'. Also similar to the standard SVM classifier, one-class SVM can use kernels to change the boundary shape as needed to deal efficiently with less-trivially distributed data sets.

C. Neural Network Classifiers

A very popular class of machine learning algorithms used for data classification are based on neural networks [10]. Artificial neural networks are crudely based on the neural structure of the brain and consist of multiple artificial neurons. Each of these neurons takes inputs, weights them with different weights, sums them up, and then passes the sum through a non-linear function producing the output. These operations can be represented mathematically as a sequence of matrix multiplications (affine transforms) followed by a non-linear activation function applied element-wise:

$$\boldsymbol{x}' = \phi.(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

where $\boldsymbol{x}'$ is the output of the layer, $\boldsymbol{x}$ is the input vector, $\mathbf{W}$ is the weight matrix, $\boldsymbol{b}$ is the bias and ϕ is the activation function. If layers of the neural network can be fully connected, then its corresponding weight matrix is fully populated. One can use a multitude of activation functions; sigmoid, hyperbolic tangent and ReLU are commonly used functions.

The weights are normally randomized at initialization. These matrices are then updated by taking the output of the last layer and back-propagating through the entire network. The core objective is to minimize a cost function, and the weights are updated using the gradient of that function with respect to the weights. Typically, both inputs and outputs for each neuron, as well as for the entire network, are normalized for the range from 0 to 1. This helps with stabilizing regression of the neural network's weights.

Various neural networks architectures are especially proliferated for image classification and analysis, however, many of them could be applied also for non-imaging data like satellite telemetry that we consider here. While network architecture is defined by the code, the network is learning how to weigh individual inputs by comparing results of its performance to known classes (labels). As other supervised learning algorithms, neural networks require labeled data for training.

D. Long Short-Term Memory Neural Networks (LSTM)

Long Short-Term Memory Networks [11] are a type of recurrent neural network architecture that is particularly suited for time series data. These LSTM networks are a type of regression model rather than a classifier as is typically associated with neural networks. Recurrent neural networks (RNNs) have feedback connections beyond the feed-forward connections of typical neural networks, which work as memory gates. LSTM aims to solve one of the common problems with RNNs, which is the vanishing or exploding gradients of these long-term connections when back-propagating the neural network weights.

LSTM is based around memory cells where the back-propagation occurs. Let the j-th memory cell be denoted by c_j. Each memory cell is built around a central linear unit with a fixed self-connection called constant error carousel. In addition to the cell's net input net_{c_j}, c_j also has a input in a form of a multiplicative unit out_j (output gate) and from another multiplicative unit in_j (input gate). Let $y^{\text{in}_j}(t)$ and $y^{\text{out}_j}(t)$ be in_j's and in_j's activation at time t, respectively:

$$y^{\text{in}_j}(t) = f_{\text{in}_j}(\text{net}_{\text{in}_j}(t))$$
$$y^{\text{out}_j}(t) = f_{\text{out}_j}(\text{net}_{\text{out}_j}(t))$$

where f denotes the activation function for the particular net input values. The net values are given as follows:

$$\text{net}_{\text{out}_j}(t) = \sum_u w_{\text{out}_j u} y^u(t-1)$$
$$\text{net}_{\text{in}_j}(t) = \sum_u w_{\text{in}_j u} y^u(t-1)$$
$$\text{net}_{c_j}(t) = \sum_u w_{c_j u} y^u(t-1)$$

where w_{ab} represents the weight on the connection from unit b to a. The summation index u can stand for input units, gate units, memory units or any conventional hidden units, depending on the architecture used. At time t, the memory cell output is given by:

$$y^{c_j}(t) = y^{\text{out}_j}(t) h(s_{c_j}(t))$$

where $s_{c_j}(t)$ is the internal state and is represented by:

$$s_{c_j}(t) = s_{c_j}(t-1) + y^{\text{in}_j}(t) g(\text{net}_{c_j}(t))$$

and is initialized by setting $s_{c_j}(0) = 0$. Intuitively, the differentiable function h squashes net_{c_j}, whereas the differentiable function h scales the memory cell outputs from the internal state s_{c_j}.

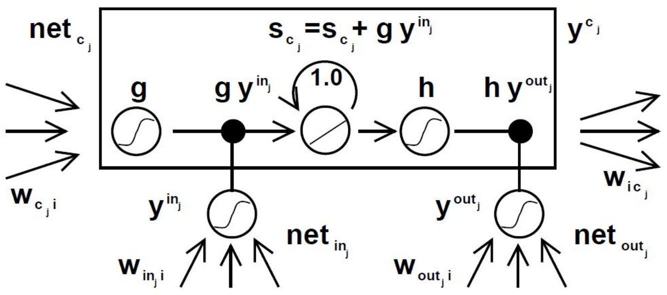

Fig. 5 Memory cell architecture for LSTM. See [11] for a more thorough explanation.

In the case of LSTM, we define anomaly as large deviation of the data point value from the prediction. During training, LSTMs learn the pattern of temporal change that is then used to make predictions for the values of the variables. Large deviations from the predictions means that the pattern learned does not hold, identifying a break in the pattern type anomaly.

IV. Method

The above algorithms were used in combination with other data modifications in an attempt to isolate the two types of faults described above (reaction wheel friction increase and and wheel speed encoder error). All data generated by Basilisk for the specific problem instances were first treated separately, and labels were added to the data for supervised learning. In addition to analyzing each data type in isolation, results were fused at the subsystem and system level as well in order to generate confidence notions in the results. Data was divided into *Attitude*, *Power*, and *Temperature* subsystems. The *Attitude* subsystem encompasses attitude and attitude rates, but also position and velocity data as well. Table 1 shows all of the data used for fault detection, and which subsystem it falls within. The Fault variable was added specially for supervised learning as described below. The remaining variables were used for both supervised and unsupervised learning.

The methods applied for both unsupervised and supervised learning techniques are described next. In addition to the traditional supervised and unsupervised learning techniques described above, we also implemented the so-called *no-change detection* method, which is not a machine learning approach, but looks for variables which freeze over long periods in an unexpected manner.

A. Unsupervised Learning for Fault Detection

The study applies GMM, one-class SVM and LSTM to unlabeled data for unsupervised learning. Each variable is modeled individually. Training data containes no faults (they are taken from the first day or first days of the simulated period), while testing data is drawn from both before and after the fault occurred. Training and testing data sets are completely independent (no training data is used for testing and vise versa). Anomalies are identified for each variable separately, and then the results are combined together through simple addition of identified anomalies. Each anomaly is labeled with the value of 1, while normal points are labeled with 0s. Adding together these labels yields a particular value, which is higher for data points in time where multiple variables are identified as anomalous. This number is normalized with respect to the maximum, which is called anomaly power, and is used for final identification of anomalies, where the anomaly power is high. This concept is illustrated by Figure 6. An anomaly is considered successfully detected for a given day (represented by a single file with simulated data) if at least one time point after the onset of the fault is identified as anomalous, otherwise the anomaly is not detected. If any point before the onset of anomaly is identified as anomalous then this is counted as a false positive. The results are next summarized for each algorithm.

B. Supervised Learning for Fault Detection

For the supervised a simple neural network classifier is implemented. As inputs all variables are used for a particular time row and also added the variable for orientation (see variable "Mode" above), as well as anomaly label (variable "Fault") – to the training data set only. This approach is referred to below as the single-row classifier. Software implementation of the neural net is done in Python using functions from a popular library Keras – a wrapper over

Table 1 List of Simulation Parameters.

Variable	Values	Subsystem	Description
Guidance Scenario	[1/2]	Attitude	Describes the attitude mode, 1 for Earth Nadir pointing, 2 for Sun pointing.
Matrix Dispersion	True, False	Attitude	True if inertia matrix is randomized between 90% and 110% of original value.
$\mathbf{r}$	$\mathbb{R}^3$ $[m]$	Attitude	Inertial position.
$\mathbf{v}$	$\mathbb{R}^3$ $[m/s]$	Attitude	Inertial velocity vector.
$\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{N}}$	$[-1, 1]$	Attitude	Attitude as modified Rodrigues Parameters.
I_{sc}	$[kgm^2]$	Attitude	Spacecraft inertia matrix in the B frame.
Time	$[s]$	Time	Current simulated time.
$\boldsymbol{\beta}_{\mathcal{B}/\mathcal{N}}$	$\mathbb{R}^4$ $[-1, 1]$	Attitude	Components of the quaternion that describes the attitude between the $\mathcal{B}$ frame and the $\mathcal{N}$ frame.
${}^{\mathcal{B}}\omega_{\mathcal{B}/\mathcal{N}}$	$\mathbb{R}^3$ $[rad/s]$	Attitude	Components of the angular velocity between the $\mathcal{B}$ and $\mathcal{N}$ frames written in $\mathcal{B}$ frame components.
$\boldsymbol{\Omega}_k, k = 1..4$	$\mathbb{R}^4$ $[rad/s]$	Attitude	Reaction wheel angular velocities.
$u_{s_k}, k = 1..4$	$\mathbb{R}^4$ $[Nm]$	Attitude	Torque applied to the reaction wheels.
RW$_k$ Power, $k = 1..4$	$\mathbb{R}^4$ $[W]$	Power	Power drawn by the reaction wheels.
Net Power	$[W]$	Power	Net power change (negative if the battery is losing energy, positive if it's receiving energy).
Panel Power	$[W]$	Power	Power being received from the solar panels.
Stored Energy	$[W]$	Power	Energy, stored in the battery.
$T_k, k = 1..4$	$\mathbb{R}^4$ [°C]	Temperature	Reaction wheel temperatures.
Fault	[0/1]	Training	Binary fault indicator (0 before the fault, 1 after the fault).

lower-level Tensorflow functions. The rectified linear activation function ('relu') is used – the most common choice for machine learning with neural nets – for the input layer. For the binary output of the classifier, a single neuron is used in the output layer with sigmoid as the activation function. The model is compiled with Adam – an adaptive learning rate optimization algorithm designed specifically for neural networks training. The popular choices of binary cross-entropy loss function and accuracy are used as the metrics. To increase the robustness of our results we used a dropout of 25% for an intermediate layer of the network.

The single-row classifier is static, i.e., it analyzes each temporal snapshot by its position in multi-dimensional feature space, but does not take into account changes in the values of the variables. This approach can work for out-of-range anomaly, but not for break in the pattern type of anomaly. To address this deficiency, another supervised learning approach is developed. This time not only current values for each variable are included, but also differences between current values and values recorded earlier and later. The study found empirically that a time difference of 150s is close to optimal value. In this case anomaly detection is done in two stages: first, a classifier is built to identify all transitions, including both mode change and faults, and then eliminated mode changes from the consideration as potential anomalies due to its high rate of false positives. This approach is referred to later as transition classification.

C. No-Change Detection

There is a significant difference between the two types of faults analyzed in this work. Reaction wheel friction faults manifest themselves in multiple variables, sometimes as an out-of-range anomaly and sometimes as break-the-pattern anomaly. Encoder faults have a much more subtle effect on the data. These faults could only be seen in one variable where the value stopped changing, however, it was barely noticeable in other variables. This meant that the algorithms struggled with detecting this type of encoder fault. A dedicated algorithm is implemented to deal with this situation. This algorithm is looking specifically for the situation when the value of a variable does not change for a long time. The resulting algorithm, though admittedly not using machine learning approach, can be easily implemented in realistic

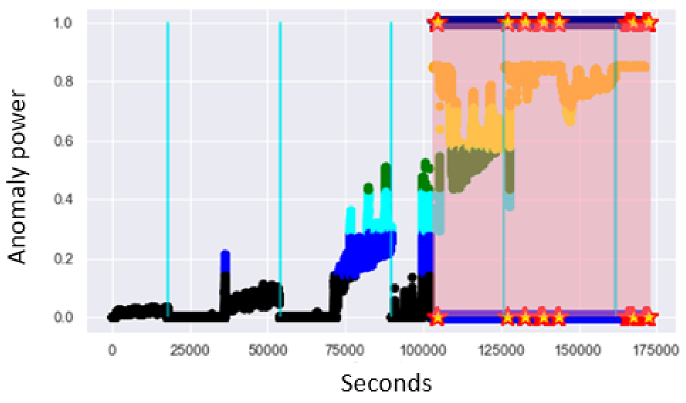

Fig. 6 An example of anomaly power change before and after the fault. Data analyzed here is for one of the encoder faults, tested by the single-row classifier. Higher values mean higher confidence that the point is anomalous. Pink shading represents time after anomaly. Vertical cyan lines correspond to the change of the satellite orientation. Colors show different levels of anomaly. Large stars represent identified anomalies.

situations and serve as a simple addition to more complicated AI algorithms.

V. Results

When the unsupervised algorithms are to the individual variables, the anomalies can be identified in most cases. However, false positives were very common, particularly when analyzing single variables (see Figure 7). Grouping the results according to each subsystem, combining together the results for temperature, power and attitude control, reducing the number of false positives while preserving anomaly detection.

Another important aspect relates to the sensitivity of some variables to the simulated faults. Ideally, the greater the impact a fault has on a particular variable, the easier it will be to detect the anomaly by analyzing the data for that particular variable. Looking at Figure 7, the temperature subsystem is able to detect the fault very clearly. The same cannot be said for the attitude system, where a fault was detected every time the attitude surpassed 180 degrees and the quaternion flipped its sign.

A performance comparison for the different unsupervised learning algorithms can be seen in Figure 8. The LSTM method (at the bottom) does not perform as well as the others, as it not only gives out a false positive result where others do not, but also repeatedly detects multiple anomalies throughout the simulation. Both GMM and one-class SVM do well in detecting the fault close to when it happened, and do not indicate the presence of multiple failures as was the case with LSTM.

However, the methods studied had a particular problem in distinguishing between anomalies and changes of the satellite orientation. Looking at the individual variables, it is evident that the change of the orientation looks in the data as either out-of-range anomaly, or break-of-pattern anomaly. The algorithms are designed to detect both of these types of anomalies, with GMM and one-class SVM especially sensitive to the out-of-range anomalies and LSTM to the break-of-pattern anomalies. Several approaches were attempted to address the issue of false positives during the satellite reorientations.

The first, rather brute force approach was simply to filter out anomalies that happen during the reorientations. This approach worked quite well and was able to eliminate most false positives because they were mostly occurring during the short time windows immediately before and after the reorientation. This is also realistic, because the reorientations are planned, and therefore, their time is known to the operators. This approach eliminated most of false positives if less than 2% of the total test data is ignored. Even though the approach worked well, the goal was to try some less *ad hoc* ways of dealing with the reorientations. To this end, the data was analyzed with supervised learning techniques.

As mentioned before, three approaches are tested for supervised learning: single-row classification, where multiple variables are used for each time step of data; transition classification, where a comparison between data before and after the current time step is also analyzed; no-change detection, which looks for variables that are frozen for long periods of time. Figures 9 and 10 illustrate results of application of supervised learning algorithms described above to individual data sets.

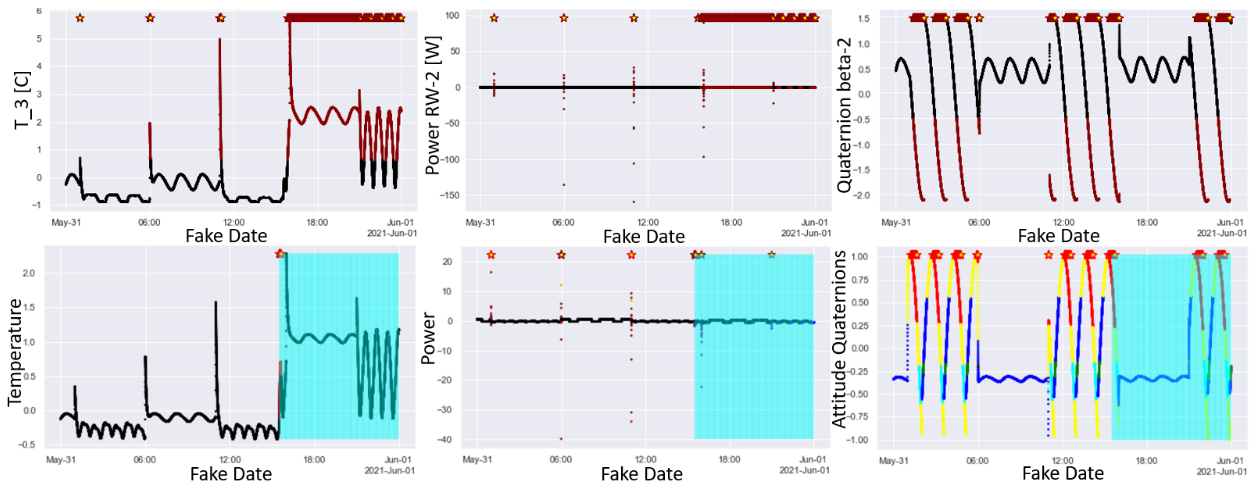

Fig. 7 Examples of anomaly detection with individual variables and subsystem summaries. All data is for reaction wheel fault number 1, analyzed with the GMM algorithm. Top row shows individual variables, while the bottom row shows results for a combination of variables for particular subsystems. These results also hint to the fact that some variables are more susceptible to false positives than others.

Table 2 contains numerical summary of the results for supervised learning approaches. Our supervised learning approaches showed good results on friction faults with inertia matrix uncertainty (no encoder faults were tested with inertia dispersion), especially for single-row classification. This means that the algorithms are robust to slight changes to control performance, such as settling times and requested torques. When we used a larger training set (40 days instead of the usual 1-2), the results were even better, with more true positives and fewer false positives.

Table 2 Summary of the results. Each entry consists of P_d/P_{FA} (probability of detection and probability of false alarm).

Method	Friction Fault (FF)		Encoder Fault (EF)		Inertia Uncertainty (FF)		Inertia Uncertainty 40 days (FF)	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing
Single-row classification	0.8/0.0	1.0/0.6	0.8/0.0	0.6/0.4	0.8/0.0	1.0/0.55	0.95/0.0	0.8/0.0
Transition detection	1.0/0.0	1.0/0.6	0.6/0.0	0.0/0.0	1.0/0.2	0.2/0.0	1.0/0.05	0.4/0.2
No-change detection	0.0/0.0	0.0/0.0	1.0/0.0	1.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0	0.0/0.0

At the same time the same quality of results could not be reproduced for encoder faults. While friction faults are rather obvious in most variables, encoder faults are more subtle and difficult to see in the data, unless you analyze the specific variable representing the faulty encoder. This is clear in Table 2, where the algorithm's performance is worse than for friction faults, with fewer true positives. However, the no-change detection algorithm described above achieved perfect results for encoder faults. Somewhat surprisingly, to avoid false positives long time windows (> 10 min) are required and multiple combinations of values must be considered for the no-change value approach to work. However, this approach only proved useful in detecting encoder faults, not being able to get a single detection for friction faults.

VI. Conclusion

This paper sets out to determine the feasibility of applying machine learning tools to telemetry data in detecting faults in the reaction wheel subsystem. The study proves that both supervised and unsupervised algorithms perform well in detecting reaction wheel faults, despite the small effects that some of the faults produce in telemetry data.

Some variables are quite useful in detecting anomalies, while others could not distinguish between faults and other behavior like attitude changes. Temperature data works well at detecting the fault rapidly, while also minimizing the

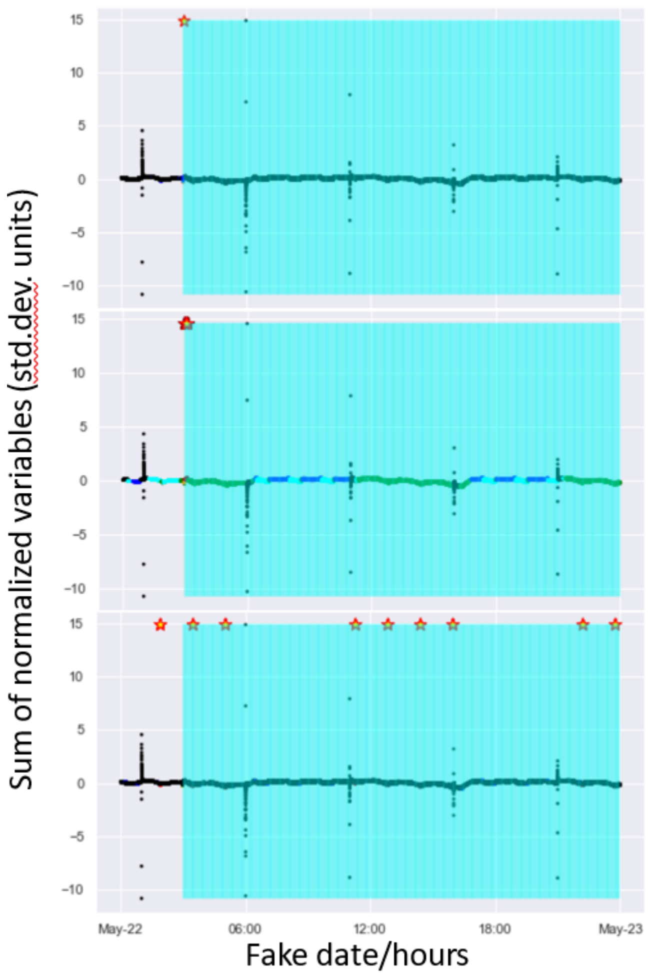

Fig. 8 Comparison of the results from different methods. All data relates to an increase in friction for reaction wheel number 1 (summary of normalized variables is shown). Turquoise region shows the period after the fault, stars indicate anomalies found with GMM (top panel), SVM (middle) and LSTM (bottom panel).

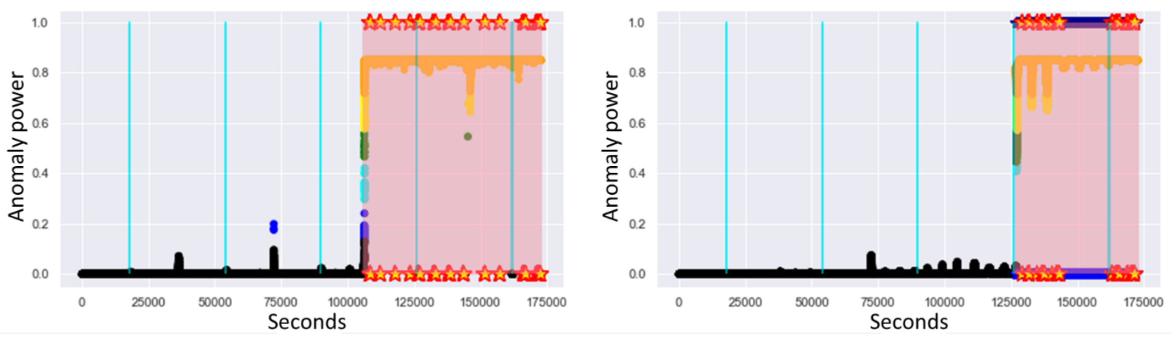

Fig. 9 Single-row classification results (examples). Left panel displays results of anomaly detection for a friction fault on reaction wheel number 2, right panel shows anomaly detection for an encoder fault on reaction wheel number 1. In both cases pink region corresponds to anomaly. Points are colored according to their probability to be anomalous according to the results of the neural nets classifier. Red/yellow stars are anomalies detected by the single-row classifier algorithm; blue/purple stars are anomalies identified by the no-change detection algorithm.

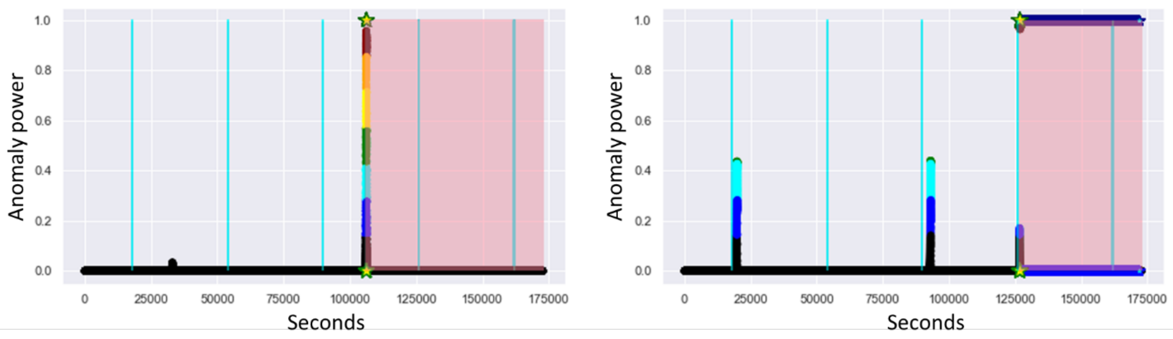

Fig. 10 Transition detection classification results (examples). The data is the same as in the previous figure. Left panel displays results of anomaly detection for friction fault on reaction wheel number 2, right panel shows anomaly detection for an encoder fault on reaction wheel number 1. Pink regions highlight time after the fault. Points are colored according to the results of the neural nets classifier. Green/yellow stars are anomalies detected by the transition detection algorithm; blue/purple stars are anomalies identified by the no-change detection algorithm.

number of false positives. Attitude parameters are not as useful. In fact, the algorithms struggle with parameter-switching when the principal angle is greater than 180 degrees. Combining several variables into one subsystem is also crucial for getting good results from the algorithms. This increases the confidence that the algorithm has actually found an anomaly and dismisses small fluctuations in individual variables.

The goal of testing inertia matrix dispersions is to see how well the algorithms performed with slightly different control performance. The algorithms fared very well, proving that they are robust to these kind of uncertainties. This is particularly important when considering that a digital twin of the spacecraft will never replicate the spacecraft perfectly. This robustness is, therefore, important for real-world applications of these algorithms.

However, there are some aspects to take into account. Attitude mode changes are one of the primary sources of false positives, where a large change in wheel speeds and attitude trigger a fault flag in the algorithms. This can be avoided by dismissing some of the data during the initial phase of the attitude transition. A refractory period immediately after a mode change where any fault flag is ignored is able to solve this issue. By removing less than 2% of the overall data, the false positives are effectively eliminated during attitude maneuvers.

All in all, this approach to fault detection is showing promising results. The ability to train algorithms on synthetic data before a mission could be invaluable to quickly and cost-efficiently detect faults in an autonomous way. This is set to alleviate the burden on operators on the ground and increase mission success while also not substantially increasing the mission's budget.

Acknowledgements

We thank AFWERX for sponsoring this Small Business Technology Transfer (STTR) program between Verus Research and the University of Colorado Boulder.

References

[1] SalarKaleji, F., and Dayyani, A., "A survey on Fault Detection, Isolation and Recovery (FDIR) module in satellite onboard software," *2013 6th International Conference on Recent Advances in Space Technologies (RAST)*, 2013, pp. 545–548. https://doi.org/10.1109/RAST.2013.6581270.

[2] Wander, A., and Förstner, R., "Innovative Fault Detection, Isolation and Recovery Strategies On-Board Spacecraft: State of the Art and Research Challenges," 2013.

[3] Mansell, J. R., and Spencer, D. A., "Deep Learning Fault Diagnosis for Spacecraft Attitude Determination and Control," *Journal of Aerospace Information Systems*, Vol. 18, No. 3, 2021, pp. 102–115. https://doi.org/10.2514/1.I010881, URL https://doi.org/10.2514/1.I010881.

[4] Kenneally, P. W., Piggott, S., and Schaub, H., “Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework,” *Journal of Aerospace Information Systems*, Vol. 17, No. 9, 2020, pp. 496–507.

[5] Willett Gies, T., Meub, J., Smith, D., Starks, M. J., and Voss, D., “The Very Low Frequency Particle Mapper (VPM) Nanosat for Space Weather and VLF Characterization,” *AGU Fall Meeting Abstracts*, Vol. 2014, 2014, pp. SM31A–4168.

[6] Schaub, H., and Junkins, J. L., *Analytical Mechanics of Space Systems*, 4th ed., AIAA Education Series, Reston, VA, 2018. https://doi.org/10.2514/4.105210.

[7] Alcorn, J., Allard, C. J., and Schaub, H., *Fully-Coupled Dynamical Jitter Modeling of a Rigid Spacecraft with Imbalanced Reaction Wheels*, ???? https://doi.org/10.2514/6.2016-5686, URL https://arc.aiaa.org/doi/abs/10.2514/6.2016-5686.

[8] Reynolds, D., *Gaussian Mixture Models*, Springer US, Boston, MA, 2009, pp. 659–663. https://doi.org/10.1007/978-0-387-73003-5_196, URL https://doi.org/10.1007/978-0-387-73003-5_196.

[9] Schölkopf, B., Williamson, R. C., Smola, A., Shawe-Taylor, J., and Platt, J., “Support Vector Method for Novelty Detection,” *Advances in Neural Information Processing Systems*, Vol. 12, edited by S. Solla, T. Leen, and K. Müller, MIT Press, 2000. URL https://proceedings.neurips.cc/paper/1999/file/8725fb777f25776ffa9076e44fcfd776-Paper.pdf.

[10] Kochenderfer, M. J., Wheeler, T. A., and Wray, K. H., *Algorithms for Decision Making*, MIT Press, 2022.

[11] Hochreiter, S., and Schmidhuber, J., “Long short-term memory,” *Neural computation*, Vol. 9, No. 8, 1997, pp. 1735–1780.