UNIVERSITY OF WASHINGTON

Techniques for Fault Detection and Visualization of Telemetry Dependence Relationships for Root Cause Fault Analysis in Complex Systems

by

Nathaniel Guy

A thesis submitted in partial fulfillment for the
degree of Master of Science in Aeronautics & Astronautics

in the

William E. Boeing Department of Aeronautics & Astronautics
University of Washington College of Engineering

February 2016

Declaration of Authorship

I, NATHANIEL GUY, declare that this thesis titled, 'TECHNIQUES FOR FAULT DETECTION AND VISUALIZATION OF TELEMETRY DEPENDENCE RELATIONSHIPS FOR ROOT CAUSE FAULT ANALYSIS IN COMPLEX SYSTEMS' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Okay, 13. We've got lots and lots of people working on this; we'll give you some dope as soon as we have it, and you'll be the first one to know.'

Jack Lousma, Apollo 13 CapCom [30]

UNIVERSITY OF WASHINGTON

Abstract

William E. Boeing Department of Aeronautics & Astronautics
University of Washington College of Engineering

Master of Science in Aeronautics & Astronautics

by Nathaniel Guy

This thesis explores new ways of looking at telemetry data, from a time-correlative perspective, in order to see patterns within the data that may suggest root causes of system faults. It was thought initially that visualizing an animated Pearson Correlation Coefficient (PCC) matrix for telemetry channels would be sufficient to give new understanding; however, testing showed that the high dimensionality and inability to easily look at change over time in this approach impeded understanding. Different correlative techniques, combined with the time curve visualization proposed by Bach et al (2015), were adapted to visualize both raw telemetry and telemetry data correlations. Review revealed insights into understanding and an intuitive grasp of data families that suggests the viability of this approach to enhance system understanding root cause analysis for actual aerospace systems.

Acknowledgements

I would like to thank my advisor, Dr. Mehran Mesbahi, for his academic guidance, his patience, and his constant encouragement as I struggled with the process of determining my own interests in this field. Deep thanks go to Dr. Jeff Heer for his insights and suggestions into the best practices for visualizing the hidden patterns within correlation data. My sincere thanks to Dr. Chris Lum for his opinions and suggestions about aerospace fault detection systems. I'd also like thank my various internship teams for their guidance and encouragement: the JPL OpsLab team, specifically Scott Davidoff, Jeff Norris, and Garrett Johnson, for introducing me to teleoperation interface design and testing techniques with the Unity 3D game engine; the SpaceX Flight Software team, specifically Mike Soares, Dan Gelband and Derek Bronish, for introducing me to Fault Detection, Isolation and Recovery systems and teaching me much about aerospace ground software systems; and the HAKUTO Lunar XPRIZE team, specifically Dr. Nathan Britton, Louis Burtz, Kurai Shimizu, Toshiro Shimizu, Toshiki Tanaka, Dr. John Walker, and Dr. Kazuya Yoshida for letting me use their lunar rover as a testbed for new ground software design methodologies and fault diagnosis techniques. Special thanks go to Daisuke Kikuchi for countless bits of design advice and for helping to increase the aesthetic appeal of the HAKUTO rover ground station UI. My thanks to the team at Unity, for developing an excellent and adaptive game engine. Finally, I'd like to thank Steve Rabin at Nintendo of America for his advice about fault detection and valuable comparisons to runtime analysis and profiling of executable code. Without the aforementioned people and many others, I would be lost!

Contents

Declaration of Authorship i

Abstract iii

Acknowledgements iv

List of Figures viii

List of Tables x

Abbreviations xi

Symbols xii

1 Introduction 1

2 Background in Fault Detection, Isolation and Recovery 4
- 2.1 Goals and Definitions . . . 4
- 2.2 Model-Based FDIR . . . 5
- 2.3 Signal Processing-Based FDIR . . . 6
- 2.4 Redundancy-Based Systems . . . 6
- 2.5 Advanced Techniques for Fault Detection . . . 7
  - 2.5.1 Rule Combinations, Fault Syndromes, and Machine Learning . . . 7
  - 2.5.2 Machine Learning and Classification . . . 7
  - 2.5.3 Common FDIR Issues and Motivations . . . 7

3 Correlative Analysis 9
- 3.1 Correlation for Fault Analysis: Motivation . . . 9
  - 3.1.1 Covariance Matrices . . . 9
  - 3.1.2 Pearson Correlation Coefficient . . . 10
  - 3.1.3 Spearman Rank Correlation Coefficient (ρ) . . . 11
  - 3.1.4 Kendall Rank Correlation Coefficient (τ) . . . 11
- 3.2 Limitations of Traditional Correlative Techniques . . . 12
  - 3.2.1 Linearity Assumptions . . . 12
  - 3.2.2 Implied Causation . . . 12

4 Visualization of Correlative Relationships **14**
4.1 Challenges . . . 14
4.2 Corrgrams . . . 15
4.2.1 Limitations . . . 15
4.2.2 Corrgram Example . . . 16
4.3 Animated Corrgrams . . . 17

5 Case Study: Hakuto "Moonraker" Lunar Rover **18**
5.1 HAKUTO Lunar XPRIZE Team . . . 18
5.2 Ground Station Interface . . . 19
5.3 Rover Data Path . . . 20
5.3.1 Non-FDIR-Related Components . . . 20
5.3.2 FDIR-Related Components . . . 21
5.3.2.1 Threshold-based fault detection . . . 21
5.3.2.2 Fault alert panel . . . 22
5.3.2.3 Expert fault information . . . 22
5.3.2.4 Correlative Functionality . . . 23
5.4 Testing . . . 24
5.4.1 Field Testing . . . 24
5.4.2 Usability Testing . . . 26
5.4.2.1 Test Tasks . . . 28

6 Intermediate Results and Reassessment **29**
6.1 Field Test Results . . . 29
6.2 Usability Test Results . . . 30
6.3 Improvements and Additions . . . 31

7 Analytical and Visualization Improvements in Response to Intermediate Test Data **33**
7.1 Dimensional Reduction . . . 33
7.1.1 Singular Value Decomposition and PCA . . . 33
7.1.2 Applying PCA to Raw Telemetry and Correlative Telemetry Models 34
7.1.3 Ad Hoc Dimensionality Reduction Techniques . . . 35
7.2 Two-Dimensional Graph Embeddings . . . 36
7.2.1 Undirected Dependency Graphs . . . 36
7.3 Time Curves . . . 40
7.4 Applying Time Curves to System Telemetry Data . . . 42
7.5 Discussion . . . 43

8 Conclusion and Future Work **49**
8.1 . . . 49
8.2 Future Work . . . 49

A Relevant Code Samples **50**
A.1 Generating Example Corrgrams for PCC, Kendall's Tau, and Spearman's Rho Correlations (MATLAB) . . . 50
A.2 Generating Undirected Graph from Filtered Correlation Data (Python) . . . 51

A.3 Generating Time Curve Distance Matrix from Raw Telemetry (Python) . 52
A.4 Generating Correlated Distance Matrices from Bucketed Telemetry (MATLAB) 57

Bibliography **59**

List of Figures

2.1 A typical fault protection flow. From [13]. 5
2.2 A simple residual generation model for a dynamic system. Adapted from [12]. 6

3.1 A comical observation on the nature of correlation and causation. From [18]. 13

4.1 Three common correlation coefficient algorithms are compared on a sample data set. Note that items 1 and 6 have a strong rank-based correlation, but the relationship is non-linear, resulting in a visibly lower correlation score within the PCC visualization. Also note the strong negative correlation between items 4 and 5. 16

5.1 Hakuto's Moonraker "Pre-Flight Model 2." Photo by George Thomas Mendel. 19
5.2 Moonraker's communication flow, from rover subsystem to ground station data storage, is shown. 20
5.3 Various telemetry values are shown for the mobility subsystem. Colors indicate current fault detection levels, with green representing a nominal state. 22
5.4 An alert panel monitors potential faults on various different subsystems, indicating any faults and their severity by color. 22
5.5 Information for monitored faults on a given subsystem. 23
5.6 Correlation map visualization showing various different data channel pairs and their correlations. Bright orange signifies a high positive correlation, and bright blue is a high negative correlation. 25
5.7 Lead software engineer Toshiro Shimizu and the author work on radio testing during a Pittsburgh field test at the Lafarge rock quarry. 26
5.8 Engineering members of Team Hakuto discuss terrain challenges during a nighttime field test. 27
5.9 A usability test participant focuses on the incoming telemetry to try to ascertain patterns behind a faulted system. 27

7.1 Singular values are shown for the Principal Component Analysis of non-correlative, sample mission telemetry data. 35
7.2 A simple example of a traditional dependency graph is shown. Here, node A's value depends on the value of node B, and node B's value depends on the value of node C. 37

7.3 A snapshot of the correlation map display from a simulated run. Note the strong correlations illustrated by opaque orange (positive) and blue (negative) cells. 38
7.4 A snapshot of an undirected dependency graph display from a simulated run. Correlated components have been isolated, with edges drawn for all correlation relationships exceeding a certain value ($r^2_{PCC} > 0.8$). Both positive and negative correlation connections are shown. Self-correlations are not shown. 38
7.5 A snapshot of an undirected dependency graph display from a simulated run. Correlated components have been isolated, with edges drawn for all correlation relationships exceeding a certain value ($r^2_{PCC} > 0.8$). Only positive correlations are shown. Self-correlations are not shown. 39
7.6 A snapshot of an undirected dependency graph display from a simulated run. Correlated components have been isolated, with edges drawn for all correlation relationships exceeding a certain value ($r^2_{PCC} > 0.8$). Only negative correlations are shown. Self-correlations are not shown. 40
7.7 A snapshot of an undirected dependency graph display from a simulated run. Correlated components have been isolated, with edges drawn for all correlation relationships exceeding a certain value ($r^2_{PCC} > 0.8$). Positive correlations, and negatively correlated subgraphs, are shown. Self-correlations are not shown. 41
7.8 A time curve is "folded" from an initial linear timeline to bring similar data points close together in the 2D embedding. From [5]. 42
7.9 A time curve embedding visualizing mission events using raw telemetry state. Event annotations are described in Tbl. 7.1. 45
7.10 A time curve embedding visualizing mission events using PCC correlation state. Event annotations are described in Tbl. 7.1. 46
7.11 A time curve embedding visualizing mission events using Spearman's Rho correlation state. Event annotations are described in Tbl. 7.1. 47
7.12 A time curve embedding visualizing mission events using Kendall's Tau correlation state. Event annotations are described in Tbl. 7.1. 48

List of Tables

7.1 Events during a visualized user simulation are shown. Cross-reference "Event Numbers" with labels in Figs. 7.10, 7.12, 7.11 and 7.9 to see correspondence. 45

Abbreviations

FD	**F**ault **D**etection
FDIR	**F**ault **D**etection, **I**solation, and **R**ecovery
GSN	**G**round **S**statio**n**
PCA	**P**rincipal **C**omponent **A**nalysis
PCC	**P**earson **C**orrelation **C**oefficient
SVD	**S**ingular **V**alue **D**ecomposition

Symbols

σ_X standard deviation of vector X

For Carl, who showed me the way to space.

Chapter 1

Introduction

Complex, remote-operated systems present many problems to engineers and designers who are conscious of mission safety. Complex systems often have detailed and comprehensive rules for the detection of anomalous conditions, or "faults," but even the most complex cannot capture the full range of possible anomalies that can occur on a system, especially if false positives are a concern and if the user has not thought of all possible conditions. Visualization of fault conditions can be an even greater problem, owing to issues such as the impracticality of simultaneously displaying data from thousands of telemetry channels, organizing data in a logical and discoverable way, maintaining system reliability in the presence of performance constraints, and leaving screen space for other non-fault-related visualization components and control affordances.

Because of these difficulties, root cause analysis can be a very long and difficult task. There are several historical cases of major system anomalies that have required very long periods of concentrated, manual telemetry data analysis (and, in some of the most catastrophic cases, post-disassembly hardware analysis) in order to piece together the root cause for anomalies. Some examples include:

- On October 28th, 2014, the Orbital Sciences "Antares" rocket suffered a catastrophic failure 6 seconds after launch, experiencing a large explosion which destroyed its cargo, which had been bound for the International Space Station. Orbital Sciences immediately launched an investigation to determine the cause, but preliminary root cause data loosely linking the mishap to a failure of one of the AJ26 engines was not publicly indicated until November 5th, and a final root cause assessment has still not, to this date, been released [2]. An independent review team within NASA evaluated telemetry data, historical data and hardware samples, beginning in November 2014, and roughly a year later, issued a report that

was still unable to provide a clear root cause for the mishap, instead linking it to three likely causes, all involving the AJ26 engine which initially exploded [28].

- On July 28th, 2015, SpaceX's "Falcon 9" rocket, carrying the "CRS-7" payload delivery up to the International Space Station, experienced an overpressure event in the second stage liquid oxygen tank, causing rapid unscheduled disassembly and failure of the mission. SpaceX engineers, working with NASA and the Air Force, began intensively examining system telemetry from the event. Despite SpaceX's well-known history of transparency about anomalous events and engineering challenges, the root cause of flawed second-stage helium system strut was not publicly identified until nearly a month later, on July 20th [25].

- On December 4rd, 2015, the PROCYON interplanetary cubesat, developed by the University of Tokyo and JAXA, went completely "dark," ceasing to provide any telemetry data at all. As of February of 2016, attempts to analyze previously received telemetry data in order to gain insight into the cause of the anomaly, and possible ideas for recovery, continue, but no root cause has been able to be determined [20].

As is shown by the cases above, the root cause diagnosis process is very difficult, and can take weeks to months to complete. It involves intense scrutiny of potentially thousands of data channels, and often the only comprehensive understanding of how these data channels relate to each other is encoded in human "tribal knowledge." Although the examples above are extreme ones, root cause diagnosis can be extremely valuable even with trivial anomalies for gaining a better understanding of system properties and subsystem connections, and the tools to do so that are currently in use are inadequate for the task. Having seen this problem first-hand in the space industry, we set out to examine the space of possible tools that could begin to tackle this problem.

In this paper, we will examine some possible solutions to the long-standing problem of root cause analysis for complex, remotely monitored systems. The paper starts with a review of schemes for identification of irregular telemetry via a typical fault detection models, and describes the necessity to characterize anomalous conditions in a more in-depth way than traditional fault detection algorithms allow, in order to identify root causes for anomalies, or to predict the occurrence of anomalous conditions ahead of time. We will point out some of the issues that commonly occur with time series data on multiple channels which can make it difficult to both analyze and visualize.

Next, we will examine traditional methods of analyzing connections between sets of time series data. We will see how solutions to some of the aforementioned issues are

provided by these analytical techniques. We will also examine a number of visualization techniques that have been used in the past to show connections within correlation data.

We will propose a data visualization technique, based on an animated adaptation of statistical correlation matrices. To assess the efficacy of this visualization, we will apply it to simulated data from a sample system, and will show test results when users are given an implementation of this visualization and asked to use it to gain insight into events during a simulated scenario. We will discuss some of the downsides of our techniques that were uncovered by testing.

Next, we will iterate and propose adaptations to address some of the issues in the first round of testing. We will look at analytical improvements as well as new visualizations, borrowing from recent research in the field of temporal data visualization, and will evalutate them for their efficacy.

Finally, we will present our conclusions about the employed analysis and visualization techniques, and will propose avenues for further research.

Chapter 2

Background in Fault Detection, Isolation and Recovery

In this chapter, we will discuss the theoretic fundamentals of Fault Detection, Isolation and Recovery (hereafter, "FDIR"). We will look at how behavior of complex systems can be modeled in such a way that undesirable states can be clearly defined and detected. We will look at more modern, advanced techniques for this analysis. We'll also spend some time talking about analytical as well as human-centered issues with current techniques, in order to motivate the subsequent work within this paper.

2.1 Goals and Definitions

First, we will define a handful terms in order to precisely discuss FDIR problems.

We will define a **fault** as any system state that is undesirable. When a fault is **detected**, a decision has been made that a fault has occurred. This will often by followed by an **alarm** step, in which the fault is made known to a human operator or to a higher level of a control system hierarchy. The process of fault **isolation** is a narrowing-down step in which the location, whether physical or logical, of a fault is determined [23]. After a fault is isolated, the final step for an autonomous system is to **recover** from the fault, by returning to a non-faulty state or entering a "safe" state in which the fault cannot negatively affect the system's ability to complete objectives. FDIR schemes for ensuring system safety are also referred to in the literature as "fault protection" schemes [13]. A typical flow is shown in 2.1.

System behavior may be modeled as a finite state machine of **operational modes**, and system states are **monitored** or **telemetered** by sets of hardware sensors and software

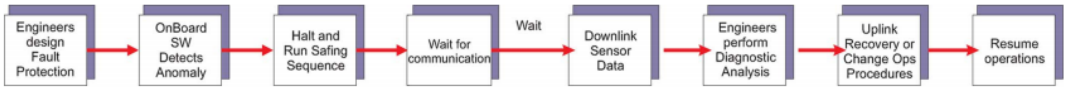

FIGURE 2.1: A typical fault protection flow. From [13].

metrics, which inform the state of the system [8]. A system state or subset of values can be said to be **nominal** when it falls into a user-defined range of normal or safe operation; it can be described as **anomalous** when it does not. The term **out of family** is also used to describe data that is specifically anomalous for a certain operational mode.

System models may include ideal or expected behavior for a given operation mode, and a method for comparing the current state to the expected mode can be expressed as a **residual** [12], which is usually a measurement of a difference from a nominal state. Furthermore, faults can be divided into **fault levels**, often along the lines of a subsystem hierarchy or in terms of severity [31].

2.2 Model-Based FDIR

Control systems for which FDIR is conducted are often represented with a modern state-space model, wherein plant dynamics are modeled as follows [12]

$$x(t+1) = (A + \Delta A)x(t) + (B + \Delta B)u(t) + E_1 n_1(t), \tag{2.1}$$

$$y(t) = (C + \Delta C)x(t) + (D + \Delta D) + u(t) + E_2 n_2(t), \tag{2.2}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^p$ is the plant's input vector, $y \in \mathbb{R}^m$ is the sensor-measured output vector, and n_1 and n_2 are disturbance vectors.

By observing the system via y, along with the known input u, it is possible to assess, via a model of expected system behavior, if the current state is an anomalous one. As discussed above, a measure of our deviation from this nominal state is the residual, and is commonly expressed as a difference between an estimated output $\hat{y}(t)$ and the observed output $y(t)$ [12], as follows:

$$r(t) = y(t) - \hat{y}(t) \tag{2.3}$$

Note that many other residual generation algorithms are possible as well.

A block diagram illustrating how a residual generation system might be structured is shown in Fig. 2.2.

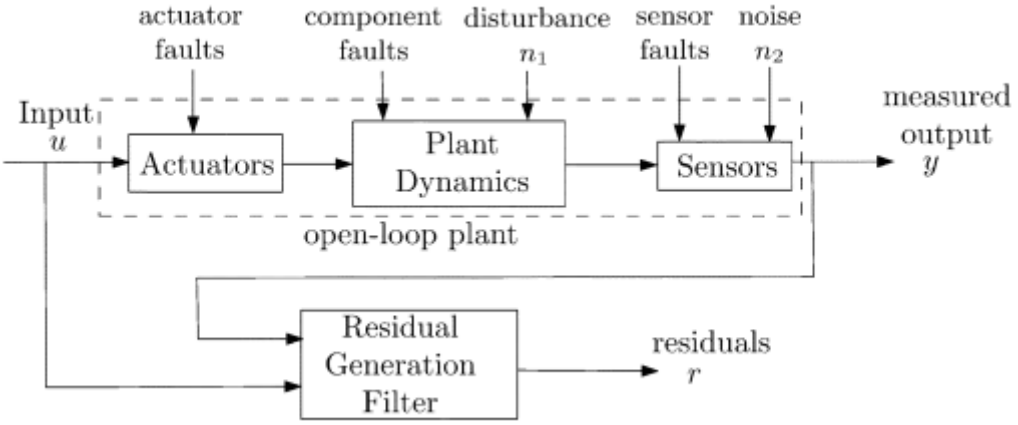

FIGURE 2.2: A simple residual generation model for a dynamic system. Adapted from [12].

Generated residuals can be evaluated via user-defined functions to assess whether a fault ought to be detected or not. The simplest form of fault residual comparison can be found in the setting of upper and lower bounds to describe the range of nominal values for a state. This "threshold-based" system is the easiest variant of fault detection to implement, and is an important part of many modern FDIR systems [33].

2.3 Signal Processing-Based FDIR

Signal analysis can also be used with particular types of dynamic systems in order to identify faults. Time-domain reflectometry, in which electrical line reflection is used to determine wiring integrity, is one such example [16]. This research will not look into applications of this type of FDIR.

2.4 Redundancy-Based Systems

For systems that are difficult to model, such as complex software systems, redundant sensors and processing units may be used, under the suspicion of a fault being likely to occur in a single one of the redundant components. Parity-vector methods such as the Generalized Likelihood Test, or voting schemes such as Random Sample Consensus , may be used to isolate the components which are likely to be in error [11].

2.5 Advanced Techniques for Fault Detection

2.5.1 Rule Combinations, Fault Syndromes, and Machine Learning

Rule-based FDIR is an alternative to model-based FDIR in which a system-dynamic model, and predicted output values $\hat{y}(t)$, are not constructed, but instead, an expert system is used to make diagnostic decisions based on system output [23]. The rule-based FDIR system uses a series of "rules checks" to determine if the system state is an anomalous one for the current mode; often, this may consist of a combination of simultaneous tests, called a "fault syndrome," whose passage or failure indicates the presence of an isolated fault [10]. Additionally, the rules may be structured as a decision tree or Bayesian network allowing for the diagnosis of a complex fault from a combination of simpler rules [11] [21].

2.5.2 Machine Learning and Classification

Recent research has made progress into using machine learning techniques for classifying and identifying faults. An intereesting overview of Support Vector Machines (SVMs) for FDIR is given in [15], and [4] discusses the usage of Hidden Markov Models (HMMs) for a similar purpose.

This paper will not attempt to examine these techniques in detail; however, it seems that they are likely to suffer from those problems typical of machine learning algorithms, including the necessity for a large amount of training data in order to learn nominal states, and the difficulty of human operator insight into the causes of anomalous states.

2.5.3 Common FDIR Issues and Motivations

Many issues exist with traditional FDIR-based schemes which limit their utility.

One major issue is the likelihood of unmodeled errors to occur on any given complex system. FDIR systems designed by humans must rely on human understanding, and as such, unpredicted issues may arise where humans fail to anticipate all of the possible system failure modes. Research suggests that unmodeled faults are incredibly common in complex spacecraft systems, and thus, there is a need for other tools to capture anomalous system states [13].

Another danger, especially relevant with expensive, teleoperated systems is the increased risk imposed by an automated recovery system. If a fault is detected and the system

takes automatic action to recover, this action cannot be potentially harmful to the system. For faults that must be resolved immediately so as to not jeopardize a mission (such a thrust irregularities on a firing rocket), FDIR systems that preempt human decisions may be necessary, on longer timescales, the value of having human experts evaluate fault-suggestive data and manually decide how to proceed may outweigh the risks of autonomous recovery.

A detailed list of issues with model-based FDIR, with multiple references in the space industries and others, is given in [13].

Because complex system faults, especially for teleoperated space systems, require large amounts of human analysis, creativity, and validation in order to understand root causes and recover from them, it is evident that the range of autonomous FDIR schemes is inadequate. We must create human-in-the-loop systems that augment model-based FDIR and analyze the resulting data to optimize the analytical work of root cause analysis and recovery.

Chapter 3

Correlative Analysis

The chapter looks at some of the fundamentals of correlative analysis for dynamic systems, and discusses the merits and demerits of various correlative methods.

3.1 Correlation for Fault Analysis: Motivation

In Chapter 2, we discussed some of the common methods for FDIR on complex systems, using linear system models that represent system state in terms of directly or indirectly telemetered values. However, there is key information that such an analysis can miss: the underlying connections between different telemetry channels. These connections can suggest semantic links, and even causations, between properties of the system, and can thus be used to link identified faults with unidentified changes of significance in other telemetered values. As such, a thorough study of the correlative attributes of a system can provide a framework with which its internal structure may be examined, and may even suggest modes of operation which can be used to understand higher-level system behavior.

3.1.1 Covariance Matrices

Covariance is defined as a measurement of the strength of correlation between two or more sets of random variables; i.e., it shows how much these variables "change together." Statistically, for two variables x and y, the covariance is the expected value of the product of the standard deviations of each of the variables:

$$\text{cov} = \text{E}[(x - \mu_x)(y - \mu_y)] \tag{3.1}$$

Intuitively, it can be seen that a situation in which x and y both are much larger (or both much smaller) than their respective means results in a high covariance; a situation where x is much larger than its mean, but y is around its mean, will result in a small covariance. In this way, the correlation between these two variables is captured by a covariance measurement.

For a vector of random variables $\bar{x} \in \mathbb{R}^n$, there exists a symmetric covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$ such that Σ_{ij} is the covariance between $\bar{x}_i$ and $\bar{x}_j$. This matrix captures valuable correlative data about the system.

When looking at two sets of data, it can often be valuable to reduce their interdependence (i.e., how much they change in sync with each other) into a single number, or "correlation coefficient." This coefficient is often a more generic version of the covariance, such that it can be used as straightforward metric to determine correlation between sets of times series data. It may even be used as input into visualization algorithms to affect shading or even positioning, as we will see in later chapters.

3.1.2 Pearson Correlation Coefficient

The Pearson Correlation Coefficient (PCC), also known as the Pearson Product-Moment Coefficient, is a metric of the linear relationship between two sets of data. It is essentially a scaled covariance, defined as

$$\mathrm{PCC}_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{3.2}$$

where X and Y are two vectors of data (or, in the case of the telemetry data sets we examine in this paper, times series of values over time for two telemetry channels). The PCC gives a quantified measurement of the linear correlation between the two vectors, in the form of a value in the range of $[-1, 1]$, where 1 is a total positive correlation, -1 is a total negative correlation, and 0 is no correlation at all.

The Pearson Correlation Coefficient carries with it a few important assumptions:

- Samples have values that are interval or ratio variables (not ordinal or categorical).
- Sample pairs have a linear relationship (or, at least, these are the type of relationships you wish to see).
- Sample pairs follow a bivariate normal distribution.

If the data doesn't fit the assumptions above, one of the two rank correlation coefficients discussed below may be more appropriate. In many of the spacecraft telemetry datasets we may encounter, the latter two assumptions will not hold, so the rank correlation coefficients will be of greater use.

3.1.3 Spearman Rank Correlation Coefficient (ρ)

The Spearman Rank Correlation Coefficient, or "Spearman's Rho," is a type of correlation coefficient, which, like the PCC, seeks to quantify relationships between vectors of data, but which looks the ranking of variables within an ordering, rather than their linear relationship. This allows the coefficient to express relationship *monotonicity*, in order to be less dependent on linearity of relationships. It actually makes uses of the PCC to do this, by calculating the PCC on the ranked data values.

Spearman's Rho is calculated by ranking the values for each variable, and then performing this calculation on those ranks:

$$\rho_{X,Y} = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}, \tag{3.3}$$

where d_i is the difference in paired ranks, and n is the number of observations. Like the PCC, Spearman's Rho takes the form of a value in the range of $[-1, 1]$, where 1 is a total positive correlation, -1 is a total negative correlation, and 0 is no correlation at all.

Spearman's Rho has assumptions which are far less restrictive than those of the PCC [27]:

- Samples have values that are interval, ratio or ordinal variables (not categorical).
- Sample pairs must have a monotonic relationship (or, at least, these are the type of relationships you wish to see).

These relaxed assumptions allow for its confident application to a wider variety of real system data.

3.1.4 Kendall Rank Correlation Coefficient (τ)

The Kendall Rank Correlation Coefficient, like "Spearman's Rho," seeks to capture nonlinear dependence by using the ordered ranks of the argument variables as input to the algorithm.

Kendall's Tau is defined as follows:

$$\tau_{X,Y} = \frac{n_c - n_d}{n(n^2-1)/2}, \tag{3.4}$$

where n_c is the number of "concordant pairs" (pairs of variables with the same rank order across observations), n_d is the number of "discordant pairs" (pairs of variables with different rank order across observations), and n is the number of observations.

Kendall's Tau carries with it the same assumptions as Spearman's Rho [26].

3.2 Limitations of Traditional Correlative Techniques

There are many limitations to the three traditional correlative techniques above, and it's important to understand them, even if the ultimate decision will be to use one of these techniques. Some of the major limitations are decribed below.

3.2.1 Linearity Assumptions

As discussed above, the PCC algorithm assumes that correlated data has a linear relationship between variables, and cannot adequately model nonlinear relationships. Many dynamic relationships that may be telemetered on a complex system are nonlinear, and will not be properly modeled using PCC. In contrast, the rank correlation methods model in terms of monotonicity, so they can capture many associative relationships that PCC cannot. Certain specially crafted pathological data sets can successfully confound PCC while being intuitively represented by rank correlation methods; see [3] for the classic example of "Anscombe's Quartet."

3.2.2 Implied Causation

One caveat of which a human operator must always be aware, especially when trying to build intuition and make conclusions based on correlative data, is that the formulations of correlation described above do not imply causation, and causation should not be inferred simply due to a high correlation score. However, correlation may *suggest* causation, and can provide valuable next steps for data examination informed by expert knowledge.

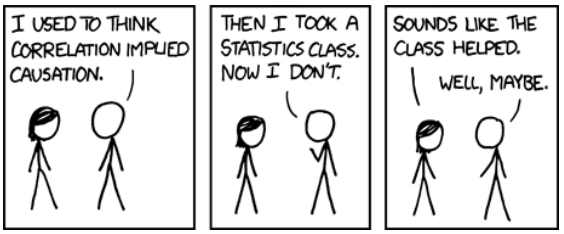

FIGURE 3.1: A comical observation on the nature of correlation and causation. From [18].

Chapter 4

Visualization of Correlative Relationships

As we discussed in the previous chapter, correlative relationships in a system can provide a useful view into the internal dynamics of state variables which a non-correlative approach could not. Because much of the motivation behind these variables is for discovery of new patterns and associations between variables, and as such, intuitive visualization to a human viewer is needed. We will investigate some traditional methods for correlative state visualization in this chapter.

4.1 Challenges

However, correlative data is very high-dimensional; for a system with n state variables, the correlative state looking back within a given time window is of size $\mathbb{R}^{n \times n}$. What's more, the correlative state changes over time as the time window slides across the state history, giving a total correlative state on the order of $\mathbb{R}^{n \times n \times t}$. For mission analysis over a 1,000-sample time window for an aerospace vehicle with 1,000 data channels, the correlative state has on the order of 10^9 elements! Simultaneously visualizing this complex state in such a way that the data displayed to the user is maximized is a major challenge. Some of the difficulties of visualizing time series data, and the benefits of event identification and grouping, are studied in [17].

4.2 Corrgrams

A grid-based matrix representation of a matrix of correlative values, or "corrgram," is a popular visualization used for correlation relationships in the research literature. In these visualizations, a state $\bar{x} \in \mathbb{R}^n$ is represented by a matrix $M \in \mathbb{R}^{n \times n}$, where M_{ij} is shaded with a color hue to indicate the correlation score between $\bar{x}_i$ and $\bar{x}_j$.

[35] suggests methods for visualizing correlations with schematic scatter plots and simple corrgrams. [19] provides a method for more expressively visualizing correlative relationships between time series using embedded ellipse glyphs, but sacrifices dimensionality and screen space. Finally, a thorough survey of corrgram representations is given in [9].

For high-detail matrix structures that push the limits of on-screen display, [34] shows a dense, 2D visualization of a subset of telemetry series data in which a measurement of "association" is found through sorting, although the details of this correlative analysis are not given, and the applications were unclear to the authors as of the publication. Finally, [6] shows a colored, compact grid structure for maximizing telemetry display, although correlation is to be inferred by user inspection (i.e., noticing if two values happen to change similarly), rather than analyzed and displayed directly.

4.2.1 Limitations

Corrgrams can only be used to display correlative state up to a certain level of dimensionality. After a certain point, the number of correlations which can be displayed on the screen is limited by screen space. The theoretical maximum, neglecting human perceptive elements, would be a correlation matrix where every pixel of a screen display is used to show a different pairwise correlation. If corrgram symmetry is exploited (i.e., if we only visualize the upper diagonal of the symmetric correlation matrix), we can reduce the number of visualized elements to $\frac{n^2-n}{2}$; however, on a generous modern screen resolution of 1920x1080, this reduces us to the capability of displaying correlations for only roughly 2,000 data channels. If we increase the pixel size to a much more reason 6x6 square for every data channel, we are reduced to roughly 300 displayable data channels. It is clear that screen space will be a major constraint, especially when precisely interaction with the data is necessary for detailed examination.

And it does seem that interaction will be a necessity. Even with a small number of data channels, horizontal and vertical labels to show data channel identifiers will not easily fit on the screen; therefore, an interaction system for showing the channel names for data pairs of interest is necessary. This interaction load slows down usage, though, and is likely to reduce utility of the visualization.

One final, major limitation of the corrgram visualization is that it only shows correlative state of the system at a single point in time; however, systems are dynamic, and their correlative states change as the systems transition between operational modes (including fault modes). If we are to see changes between modes, and to be able to identify these events as possible faults, we need a method to see correlative data change over time.

4.2.2 Corrgram Example

A monochrome corrgram comparison of the Pearson Correlation Coefficient, Spearman Rank Correlation Coefficient, and Kendall Rank Correlation Coefficient is shown in Fig. 4.1. Note that the strong negative linear correlation between data channels 4 and 5 is captured well by all three correlative measurement methods; however, the strong, nonlinear association between data channels 1 and 6 is not captured as strongly by the PCC due to its inability to model nonlinear relationships.

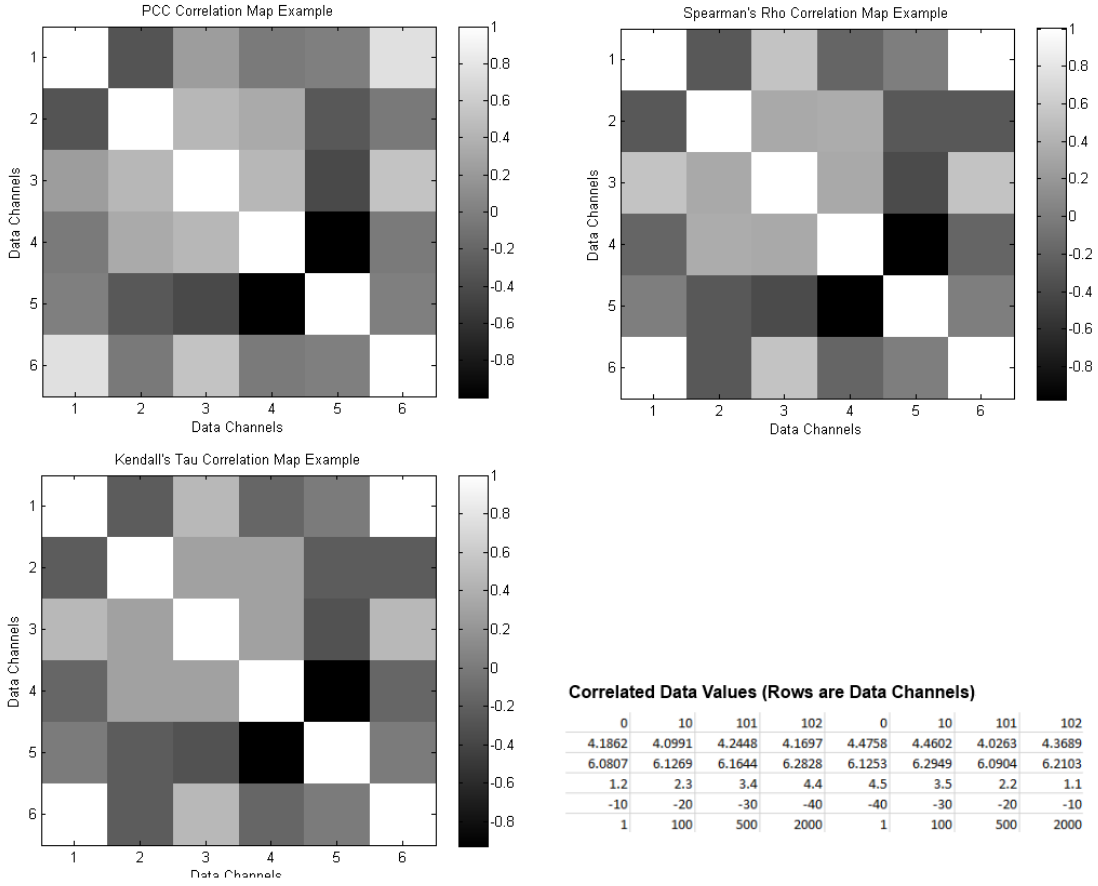

FIGURE 4.1: Three common correlation coefficient algorithms are compared on a sample data set. Note that items 1 and 6 have a strong rank-based correlation, but the relationship is non-linear, resulting in a visibly lower correlation score within the PCC visualization. Also note the strong negative correlation between items 4 and 5.

4.3 Animated Corrgrams

Animated corrgrams are a potential solution to address the traditional corrgram's inability to capture changing correlative state over time. In an animated corrgram, the correlative state of a system is calculated over a retrospective time window, and every time new data is received, the correlative state is recalculated and redisplayed. The next chapter will walk through the creation of an animated corrgram with real system data. We have been unable to find any examples of animated corrgrams used in previous research for displaying changing system state over time.

Chapter 5

Case Study: Hakuto "Moonraker" Lunar Rover

In this section, we will discuss a motivating problem and platform on which to test some of the algorithms we've developed and assess their practicality.

5.1 HAKUTO Lunar XPRIZE Team

In the summer and autumn of 2015, research was performed at Tohoku University's Space Robotics Lab (hereafter "SRL"), under the guidance of Professor Kazuya Yoshida, among others. This laboratory focuses on the research and development of robotic systems for space exploration and science missions.

One of the major sub-groups within SRL is the Hakuto Lunar XPRIZE team, a group of engineers who, working together with their promotional counterparts in Tokyo, have been working for several years on the core mission of sending a lunar rover to the Moon, traveling at least 500 meters, and sending back high-resolution and photos. Completing this mission would satisfy the requirements of the Google Lunar XPRIZE, an international lunar rover competition with a combined purse of $30M USD [1].

As a secondary mission, Hakuto hopes to explore the interior of caves on the Moon, as precursor exploration to assess their feasibility as future human habitats. Recent high-resolution photography from JAXA's Kaguya spacecraft, and from NASA's Lunar Reconnaissance Orbiter, has confirmed that large "skylights" exist on the lunar surface leading into these caves [7], and Hakuto aims to land near enough to one of these skylights to make its exploration a possibility.

We decided that Hakuto's four-wheeled "Moonraker" rover would be an excellent testbed against which to develop advanced fault analysis algorithms and visualization. Moonraker is a state-of-the-art micro-rover, developed over the past 5 years by the Hakuto team. During normal operation, Moonraker sends back status reports on 100 to 150 channels of telemetry data to its ground station on Earth, at a rate of once per second. This telemetry covers everything from IMU attitude data and temperature sensors readings to motor rotations, solar charge voltage, and communication metadata such as packets errors detected and radio signal strength.

See Figure 5.1 for a photo of Moonraker.

FIGURE 5.1: Hakuto's Moonraker "Pre-Flight Model 2." Photo by George Thomas Mendel.

5.2 Ground Station Interface

In early July 2015, Team Hakuto began designing a new ground station software suite for the most recent version of the rover. The rover had been updated in several ways since the prior Pre-Flight Model, and many of its avionics, including its software, were updated and redesigned, breaking compatibility with the previous version of the ground

station software. We worked together to evaluate the current and future needs of the rover, and to redesign and reimplement software to optimally fit these needs.

Our discussions focused on optimizing performance, reliability, and maintainability of the software. The latter factor was of particular concern, given that the software was to be used and maintained in a university laboratory environment, with many students—some of them inexperienced in software engineering—potentially responsible for updating the software and adding new features. After evaluating a list of disparate choices, including C++ on Linux with the Qt framework and multi-platform JavaScript running in HTML5, we ultimately decided that the ideal choice would be the Unity Game Engine, for its high frequency of software updates, its active developer community, and its aerospace legacy within the NASA Jet Propulsion Laboratory [29].

5.3 Rover Data Path

In Hakuto's network configuration, Moonraker sends data packets over radio from the Moon, and they are intercepted on Earth and relayed to the ground station over the Internet. Subsequently, these packets are decoded and processed in order to be visualized by the ground station. The data is stored locally in memory by the ground station software for subsequent display. A flowchart of this data path is shown in Fig. 5.2.

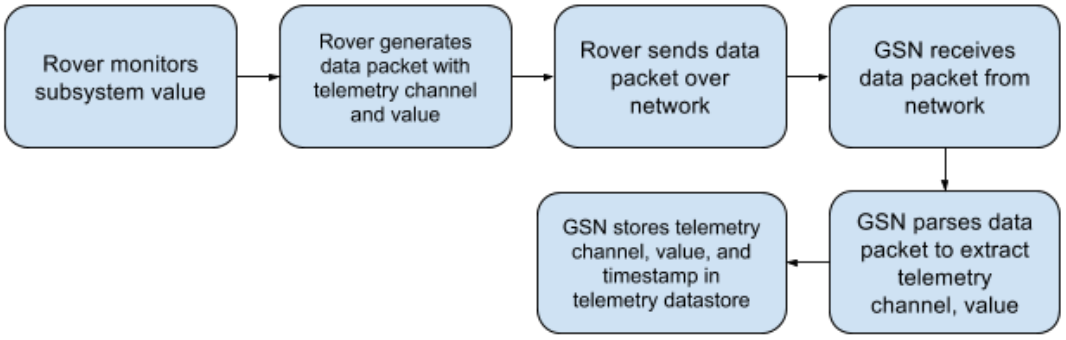

FIGURE 5.2: Moonraker's communication flow, from rover subsystem to ground station data storage, is shown.

5.3.1 Non-FDIR-Related Components

The focus of this paper is on the fault detective and correlative visualizations implemented within the ground station, so we will spend the bulk of the time focusing on this component. However, there were several non-FDIR-related components as well, briefly described below to provide context:

- *Numerical telemetry display*, to visually show the state of various sensors on subsystem boards (such as IMU accelerations, motor voltages, and board temperatures), as well as internal software metrics. Whenever possible, telemetry data was placed in semantically meaningful positions and groups, to improve discoverability.
- *Attitude display and visual tachometer*, to provide more intuitive visualizations of pitch, roll, and current wheel rotation rate (which mostly corresponded to vehicle speed).
- *Telemetry change indicators*, to point out data channels that have strong downward or upward trends over time.
- *Quad-camera display*, to display the most recent images and streaming video from the rover cameras.
- *Connectivity map*, to show the state of connectivity to various subsystems based on the elapsed time since packets from those subsystems had been received.
- *Immersive viewing*, allowing users to navigate the camera data in an embedded 3D mapping.
- *Map display*, showing the position of the rover with respect to the surrounding selenography, based on mobility subsystem telemetry and SLAM telemetry.
- *Audio alert cues*, to draw the user's attention to the UI in the event of faults.
- *Telemetry saving, loading, and playback*, to facilitate the review of mission events after the fact.

5.3.2 FDIR-Related Components

In comparison with traditional aerospace ground station software, particular attention was given in this implementation to FDIR-related components. The components below were implemented and used extensively.

5.3.2.1 Threshold-based fault detection

In order to build a threshold-based fault detection system for Moonraker, we worked with engineers on our team to define the "danger" thresholds that indicated points of severe jeopardy, as well as the "warning" thresholds that indicated points of concern. We implemented these thresholds in my ground station software as a general-purpose fault detection engine. Faults are detected constantly, and are displayed to the user via

color and detailed information (see the following sections for more details). All fault occurrence details and times are logged for future review as well. See Figure 5.3.

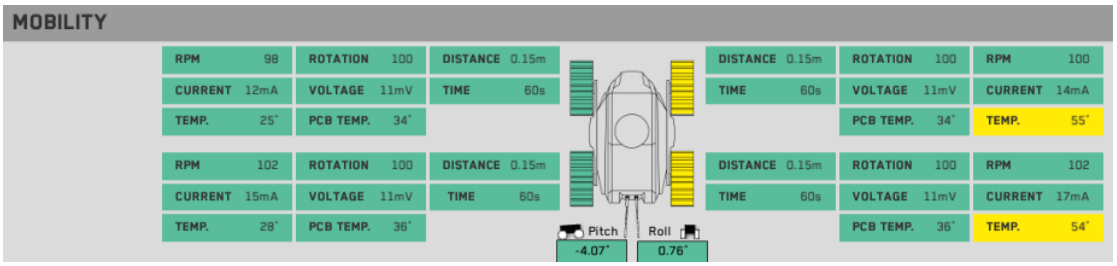

FIGURE 5.3: Various telemetry values are shown for the mobility subsystem. Colors indicate current fault detection levels, with green representing a nominal state.

5.3.2.2 Fault alert panel

For safety, faults that have occurred need to be easily visible and understood by human operators. Towards this end, a highly visible, brightly colored alert panel was placed at the top of the screen seen by human operators. Each panel cell corresponds to a subsystem or other type of data grouping, and any issues with that grouping (i.e., faults that occur on monitored channels) will trigger a color change on that cell. Interacting with the cell can give the user more information on the fault (see the next section for details). See Figure 5.4.

FIGURE 5.4: An alert panel monitors potential faults on various different subsystems, indicating any faults and their severity by color.

5.3.2.3 Expert fault information

Ensuring human understanding of fault data is an essential part of the fault diagnosis and recovery process. As such, it's important to design a system where detailed information can be provided about individual faults that have occurred, while maintaining a high-level understanding of which systems are behaving anomalously. The system designed allows for this hierarchical organization of information. When high-level fault information is indicated in the "fault alert panel," more concise information is provided in the "fault information panel," including which anomalous data channels are contributing to the problem. The fault information is highly extensible, allowing for any other additional fault-related notes that system designers or operators would like to include for

reference. This additional information, uncommon in traditional fault monitoring systems, accelerates the fault diagnosis problem by immediately pointing towards possible root causes. See Figure 5.5.

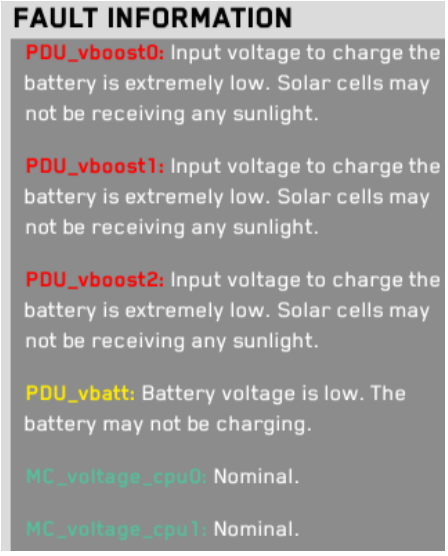

FIGURE 5.5: Information for monitored faults on a given subsystem.

5.3.2.4 Correlative Functionality

The Pearson Correlation Coefficient was chosen as an algorithm for calculating correlationbetween sets of data channels, in order to give a metric of their mutual "connectedness." This algorithm was chosen for its simple and efficient calculations, under the (in retrospect, nave) assumption of linear relationships.

To visualize this data, a two-dimensional corrgram was used, visualizing relationships between data channels as a matrix with cell shadings representative of the PCC score of each relationship. Hue of each cell shows positive/negative correlation, and intensity shows the strength of that correlation. This visualization allows an operator can see changing channel correlations, which may suggest possible interconnectedness or causation and aid with troubleshooting. See Figure 5.6.

To show the flow of this data, pseudocode for the algorithm used is shown in Algorithm 1. Assume that a pre-generated $n \times n$ grid of transparent blocks, **Corr**, is displayed on the screen. Also assume that **GetColor** refers to an arbitrary function which maps from a PCC score $\in [-1, 1]$ to an RGB color $(r, g, b) \in \mathbb{R}^3$, where $0 \leq r, g, b \leq 255$. We experimented with several variations of this function, and found that a linear mapping exaggerated the importance of low correlation scores, which led to the development of a hand-tuned exponential color mapping function to produce the correct scores. Certain work has explored the complexities of determining a proper mapping, which relate to the

non-linearity of human perception of color [9]. Other work has suggested the efficacy of pre-squaring the PCC score before display, which instead results in displaying the coefficient of determination (i.e., the shared portion of the variance) for the pair of data items, which intuitively corresponds to a more intuitive human understanding of data connectivity [22].

Algorithm 1 Animated Corrgram Generation Algorithm

procedure CORRGRAMGENERATOR(D) ▷ Takes data point matrix $D \in \mathbb{R}^{n \times t}$.
  $S \leftarrow \{U_{:,j}\}_{j=t-s}^{n}$ ▷ Reduce data to most recent s points.
  $P \leftarrow \mathbf{PCC}_s(D_r)$ ▷ Calculate a symmetric PCC matrix using last s points.
  for row $= 1$ **to** n **do**
    for col $= 1$ **to** n **do**
      color $=$ **GetColor**$(S_{row,col})$ ▷ Convert PCC score to a color.
      $\mathbf{Corr}_{row,col}$**.color** $\leftarrow$ **color** ▷ Assign color to cell in corrgram.
    end for
  end for
end procedure ▷ Algorithm is re-run on every graphical update.

A few additional modifications were applied in order to make this algorithm performant. For instance, the entire corrgram is not updated at once, due to the performance hit from calculating new PCC scores for thousands of corrgram cells on each graphical update. Instead, a subset of the corrgram cells are updated, resulting in a "rolling" effect wherein cells update gradually, with a full update of the corrgram being achieved on the order of once per second.

We also built functionality to adjust the sampling rate, the period of time over which samples are taken, and parameters related to averaging for smoothing out noise. Channel pairs may also be filtered via substring search.

5.4 Testing

Hakuto carried out a number of field tests on the ground station software, including FDIR-related and correlative components. The major tests we conducted are described in detail below.

5.4.1 Field Testing

Over the course of two weeks, we performed daily field tests in the rock quarry, to both assess the physical characteristics of our radio communication and to confirm our mobility in a simulated lunar environment. (See Fig. 5.8 for an image of the quarry terrain.) The ground station interface was in constant use throughout the course of the

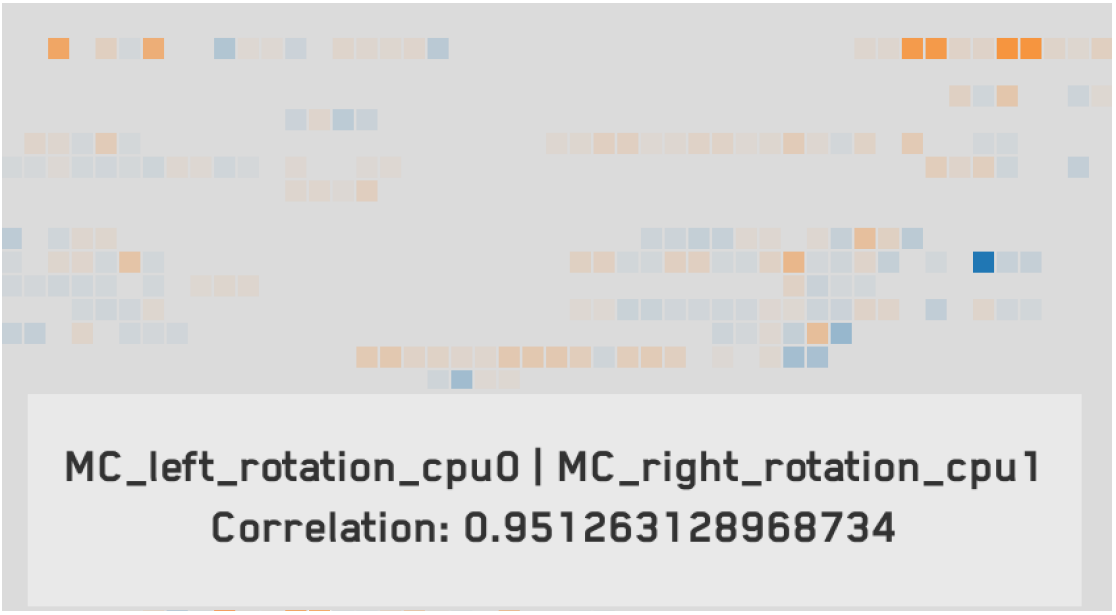

FIGURE 5.6: Correlation map visualization showing various different data channel pairs and their correlations. Bright orange signifies a high positive correlation, and bright blue is a high negative correlation.

tests, with alternating drivers at the helm. The setup usually involved one primary pilot and one copilot, who helped with troubleshooting. See Fig. 5.7 for an image of normal operation.

Our radio testing was multi-pronged. First, we endeavored to established that we could a) communicate from the ground station to Moonraker and vice-versa by going through Astrobotic's radio relay, which they set up to emulate their lunar lander which will function as a communication relay during the actual, planned lunar mission. Second, we tested the communication capabilities of Moonraker's radio antenna at two different operating frequencies (900 MHz and 2.4 GHz), in order to characterize performance over long distance and with line-of-sight blocked by rocky terrain. We looked for communication signal strength and commanding efficacy in the presence of packet loss, and also tested the effect of a signal strength amplifier in a poor connectivity situation. These results were promising and are currently being analyzed by our communications team.

Our mobility testing consisted of driving long distances over rocky, yet generally even, terrain, occasionally using only the near-real-time ("NRT") streaming video telemetry as visual feedback. We set up various challenges, such as large rock obstacles and inclines of increasing steepness, to test the rover's mobility capabilities as well as our operational capabilities using the ground station interface.

Other tasks performed during the field tests included coordinating with Hakuto crew and Astrobotic/Caltech engineers to fulfill mission tasks, scouting for field test locations,

setting up and testing the ground station and radio equipment, and driving the rover during all of the tests to accomplish all of our test goals.

Results of this field test, and other similar ones, will be discussed in the next chapter.

FIGURE 5.7: Lead software engineer Toshiro Shimizu and the author work on radio testing during a Pittsburgh field test at the Lafarge rock quarry.

5.4.2 Usability Testing

In November 2015, we performed a set of usability tests on a slimmed-down version of the Moonraker ground station interface, in order to evaluate the various data analysis affordances and to determine any usability issues in need of attention. Seven users participated in the test, mostly interns and students possessing some familiarity with the rover, but with limited to no experience operating it via the ground station interface. See Fig. 5.9 for an image of a user taking the test.

Participants were presented with a simplified form of the ground station, with the Pilot Screen, Copilot Screen, and the Correlation Map screen. As the correlation map is a relatively novel idea, and has not (to the best of the author's knowledge) ever been used in ground station software, this test also served to see if it could be immediately usable

FIGURE 5.8: Engineering members of Team Hakuto discuss terrain challenges during a nighttime field test.

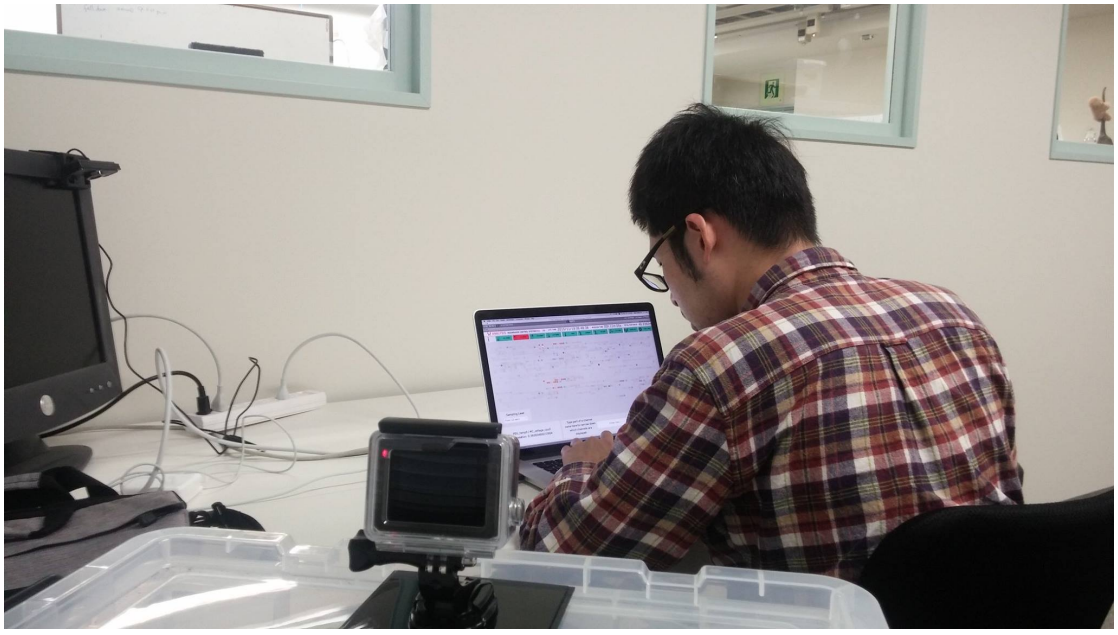

FIGURE 5.9: A usability test participant focuses on the incoming telemetry to try to ascertain patterns behind a faulted system.

in its present state, or if special training or modifications will be necessary before the correlation map is useful for correlation discovery and root cause analysis.

5.4.2.1 Test Tasks

During the test, participants monitored telemetry and looked for patterns in received data during a 10-minute simulation drive on the lunar surface. During the course of the drive, camera images were unavailable, but the rover telemetry was indicative of various events that occurred to the rover during the run. The scenario consisted of the rover descending down into a crater, where it loses direct sunlight, resulting in reduced temperatures and solar charge voltages. It also loses line of sight with the lunar lander, causing packet losses and a reduction in signal strength. The rover then climbs out of the crater, bringing about an all-around improvement in these conditions. It enters some rough terrain, and ultimately stops after a rock becomes stuck in the wheel.

It was the task of the usability test participants to analyze the telemetry as it was received in order to try to understand the details of the story above. After the 10-minute run was complete, users were then given up to an additional 10 minutes to review received telemetry until they were satisfied they had understood the "story" of the rover's trip.

Results of usability testing will be discussed in the next chapter.

Chapter 6

Intermediate Results and Reassessment

This chapter discusses some of the learnings from our intermediate testing, and how we used them to shape our next steps for correlative assessment techniques.

6.1 Field Test Results

Our field tests went swimmingly, and we returned to Japan with excellent results and ideas for improvement, but these tests were not without a small number of issues. We uncovered several minor bugs within the ground station, which were quickly fixed on-the-spot. The tests—especially the act of watching others operate the ground station—yielded useful data about which features were used the most, and which were all but ignored entirely.

We found the "immersive viewing" feature to be immensely helpful for gaining situational awareness, and proved its efficacy in testing where engineers placed the rover in a precarious situation, then asked us over radio to use the telemetry and visual data to give them a detailed description of the current situation. The "connectivity map" also proved extremely useful in quickly informing us of issues with the rover, as did the overall fault detection system. Some features were determined to be of limited utility and removed.

Fault information was shown to be somewhat sparse for encoding expert understanding of problems, and there were several instances of users accidentally ignoring unsafe rover conditions. In response to these observations, space made available from removing other features was used for providing more detailed fault information (this was an enhancement

of the "Expert fault information" feature discussed in the previous chapter). Also, fault detection and display rules were rethought and tweaked, to enhance visibility of anomalous states.

6.2 Usability Test Results

Usability testing yielded many results that were useful for setting subsequent development priorities. Overall, all participants were able to deduce the general course of events undergone by the rover, and at the end were able to narrate a story which resembled the intended one outlined above.

Users remarked that they found the Copilot Screen's telemetry display the most useful, and they spent the majority of their time monitoring telemetry data on this screen. Most participants used the data review feature heavily while on this screen, rewinding and fast-forwarding through time and watching displayed telemetry values change as they did so. Users remarked that this feature was easy to use and excellent for reviewing the flow of telemetry. However, multiple users expressed a desire for time series plots of data channels, to better see the history of data at a glance.

Faults were generally very visible, and users commented that they found the additional fault information very helpful when trying to understand the meanings of various fault states. However, it was observed that tunnel vision was occasionally a problem for users; too much attention on one specific UI component, such as the correlation map or displayed telemetry on the Copilot Screen, seemed to be responsible for delays of up to 30 seconds in reacting to critical fault events. This observation led to the addition of audio cues to redirect user attention, which has already shown to be effective in informal testing.

Multiple users commented on the difficulty of understanding telemetry for channels whose meaning they did not understand. (Many of the subsystems have very specific data channels whose meaning is not well understood to anyone except the designers of those systems.) Results indicate that the expert fault information feature mentioned above was effective in eliminating much of the confusion about specific faults; however, we believe that adding information that leads to a better understanding of individual telemetry channels would result in less user confusion and could improve the effectiveness of human telemetry monitoring. Knowledge capture efforts are currently underway to collect detailed information about data channels from subsystem engineers to incorporate this data into the interface.

The usability test uncovered many issues with the correlation map in its current form. Many of the users commented that they did not understand the proper way to use it to analyze data (although they understood the basic idea of the visualization). Users requested better, more intuitive spatial organization of data channel pairs, and the ability to more easily filter channels of interest and ones pertaining to faults. We received comments that additional training sessions might be beneficial. Nearly all users expressed an interest in using this feature, but the performance of those who endeavored to analyze faults with this tool showed evidence of a need for automated simplification to reduce stimuli, and to come up with better ways to train users and to lead them to useful conclusions.

A few other minor issues came up which we had not foreseen until performing the testing. Some users had difficulties with transition lag between screens (there is a lag of approximately a second between screen transitions due to a need to load resources). We am looking into performance enhancements and/or loading screens to fix this issue. We also received the feedback that a more visible speed/RPM indicator for the rover would be helpful, as speed is one of the most important aspects of the rover as it operates, and this data is easily overlooked in the midst of other types of telemetry. This led to the development of the visual tachometer RPM visualization discussed in the previous chapter.

6.3 Improvements and Additions

Looking at the rests of this intermediate testing, we were able to identify various targets for further research and improvements in the field of analysis of the fault-related, and correlative data.

Even with the small number of data channels available on the Moonraker rover (112), the dimensionality was very high for a full corrgram display, and seemed to stretch the visual and attentive capacities of the users who participated in the test. Even with the addition of data filtering features, without focused training of the use of these features, they didn't seem to provide a better experience for users looking for patterns in the correlative data. Though users were able to, from an integrative viewing of telemetry data as shown in the numerical and color-based fault displays, able to come to understand a timeline of the rover events, mental links between associated channels seemed to emerge due to sheer coincidence; remarks from users were along the lines of "the temperature is increasing... and I see that the voltage is increasing too... maybe they're related." The correlation map, as a way to call out these patterns, did not seem to be as effective as anticipated.

While this data was successfully captured by the analysis, we determined that it was a) buried in the noise of lots of unimportant correlations and b) shown on a larger visual display with too much data to easily visual process in a short amount of time. What's more, c) there was no affordance for users to simultaneously display time across different temporal points.

As such, we determined that it would be of value to iterate on these three points, with the main goals of reducing correlative noise and data dimensionality and of providing a simplified visualization capable of displaying data from multiple time points simultaneously. The next chapter will discuss a few approaches towards this end and their results.

Chapter 7

Analytical and Visualization Improvements in Response to Intermediate Test Data

This chapter discusses some of the additional improvements we made in response to problems identified in usability and field testing.

7.1 Dimensional Reduction

In order to reduce complexity of the displayed data for reduced cognitive load, we first looked at methods for reducing the total amount of data to be displayed.

7.1.1 Singular Value Decomposition and PCA

The Singular Value Decomposition, or SVD, is a way to diagonalize a given matrix in a pair of basis vectors [14]. If the matrix to be diagonalized is A, then the SVD may be written as

$$A = \mathrm{U}\Sigma\mathrm{V}^*, \tag{7.1}$$

where $\mathbf{U}$ is a set of basis vectors expressing the range, Σ is the diagonal matrix of singular values, and $\mathbf{V}$ is a set of basis vectors expressing the domain.

The SVD's two bases are orthonormal, and the SVD is guaranteed to exist for any matrix A, which cannot be said about the alternative diagonalization method of the

eigenvalue decomposition [**?**]. The SVD allows us to project a matrix onto a low-dimensional representation in a formal way, a very useful quality for our spring system representations in this project.

Principal component analysis, or PCA, is a technique that makes heavy use of SVD. PCA takes an SVD of the covariance matrix of a series of observations in order to find the basis vectors and singular values that characterize the behavior. By looking at the magnitudes of the singular values, it becomes apparent which are significant, since the singular values are ordered and those of smaller magnitude can be elided in order to achieve a low-dimensional reduction to serve as a representation of the system [14]. This reduction consists of the "principal components" of the system, and thus, PCA can be used to gain a fundamental understanding of how a system behaves, and how many principal modes it contains.

Note that it is important to normalize data before performing PCA, in order to avoid emphasizing data channels with large variances. Refer to [24] for a detailed discussion of PCA.

7.1.2 Applying PCA to Raw Telemetry and Correlative Telemetry Models

As we have seen before, the raw, telemetered state history for system telemetry is a matrix $M \in \mathbb{R}^{n \times t}$, while the correlative state history of a system is even higher order, on the order of $C \in \mathbb{R}^{n \times x \times t}$. Reducing either of these matrices to their principal components could be immensely valuable towards simplifying their dynamics. The post-PCA result would be a new, simplified and reconstructed state vector that captures the majority of the state dynamics. It should be noted that this technique has been shown to be of value in [32].

We can apply PCA, as described above, to first simplify the non-correlative state matrix obtained from raw telemetry. We will use sample data from the usability test simulation discussed in Chapter 5.

After the SVD has been performed on this data, scaled singular values were obtained as shown in Fig. 7.1. Though the state vector of the original system is 112-dimensional, the SVD shows only 29 singular values above 15% contribution, suggesting a system of dimensionality 29. We can use a reduced singular value matrix Σ_r, containing only this subset of the 29 largest singular values, to reconstruct the state dynamics of the system. Though we cannot easily plot such a reconstruction to show its faithfulness visually,

we can calculate a norm difference between the standardized original and reconstructed matrices, and in doing so, find a standardized difference of 11.41.

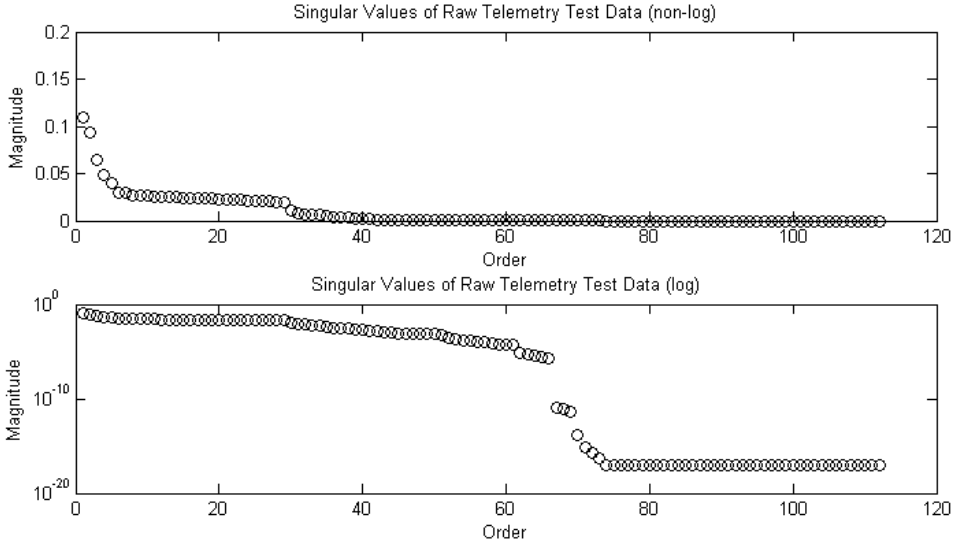

FIGURE 7.1: Singular values are shown for the Principal Component Analysis of non-correlative, sample mission telemetry data.

A similar deconstruction and reconstruction can be applied to the correlative models to reduce dimensionality; however, by calculating new bases for analysis, an intrinsic connection is lost with respect to the nature of the underlying data channels. For a human operator, a simplified system representation is only of value if it represents changing state quantities that have a meaning to the operator. It is the opinion of the author, after examining this possibility, that a weighted basis of 112 different state variables is of questionable value, and should be excluded from any visualizations.

7.1.3 Ad Hoc Dimensionality Reduction Techniques

In addition to identifying principal components of the system and displaying only those modes with significant singular values, we also can look for system-specific optimizations which can reduce dimensionality.

The simplest technique is to remove data channels that are not of interest. Channels that always behave deterministically, such as known functions of other channels or channels that stay constant throughout an entire telemetry recording, are prime candidates for removal. (The latter set of channels could not be covered in correlative analysis anyway, because of the zero standard deviation in the time series.)

Another candidate for dimensionality reduction is reducing the scope of correlative calculation. Instead of correlating all n channels with each other, if channels are divided

into independent subsystem clusters, then these subsystem channels may be correlated internally, without inter-correlations among these clusters. However, this eliminates the possibility of discovering unintended data connections between systems, which could be indicative of unanticipated and important fault-related behavior.

Both of these approaches are highly supervised and require expert knowledge of the system, constraining their general applicability and restricting their capacity to discover unintended patterns in the data. However, they could prove useful in the absence of better techniques.

7.2 Two-Dimensional Graph Embeddings

Since the vast majority of user interfaces in common usage are two-dimensional, and hardware limitations can easily result in 3D user interfaces being infeasible for users, it makes sense to look at ways that n-dimensional system state data can be embedded within a 2D visualization. Towards this purpose, I did a brief survey of state-of-the-art 2D graph embeddings, looking for implementation feasibility and the ability to give a user "insight" into the nature of a system fault. Preferably, a 2D embedding for system understanding will make major state transitions and patterns visibly obvious at a glance, and will spatially separate different system "modes" so that they can easily be mentally grouped by the viewer.

7.2.1 Undirected Dependency Graphs

The first type of two-dimensional graph embedding that we examined as an alternative for animated corrgrams was the "dependency graph." Dependency graphs are a type of 2D embedding in which a complex system is represented as a directed graph, where nodes are system components and edges represent dependencies; for example, if $\mathbf{node}_A \rightarrow \mathbf{node}_B$ and $\mathbf{node}_B \rightarrow \mathbf{node}_C$, this indicates that the value of $\mathbf{node}_A$ depends on the value of $\mathbf{node}_B$, which in turn depends on $\mathbf{node}_C$. A simple example of this type of visualization is shown in Fig. 7.2.

As such, this type of visualization lends itself well to illustrating the effect of causation in a system. Though causation can be difficult to determine in a complex system, as we have already shown, correlation is calculable via the PCC and other metrics, and a correlation-generalized undirected correlation graph could be envisioned, wherein two nodes have an undirected edge if their mutual PCC score exceeds a certain threshold, and have no edge if their mutual PCC score fails to meet that threshold. The steps to generate such a graph are as follows:

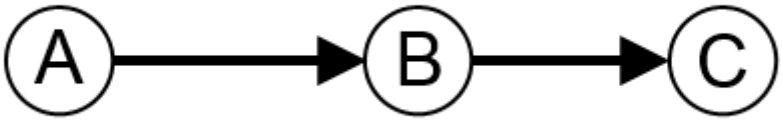

FIGURE 7.2: A simple example of a traditional dependency graph is shown. Here, node A's value depends on the value of node B, and node B's value depends on the value of node C.

1. At a given point in time, generate a PCC matrix for all of the possible data channel pairs.

2. Generate an unconnected graph in which there exists a degree-0 node for each data channel.

3. For node-node pair, add a connecting edge if there exists a PCC score above a reasonably high correlation threshold (e.g., $r^2_{PCC} > 0.8$). This edge can be colored to show correlation sign (e.g., blue for $r_{PCC} < 0$ and red for $r_{PCC} > 0$).

4. Finally, cull all nodes that are still of degree 0.

With the corrgram visualization, we needed to illustrate all possible pairs of channels as a separate cell, and thus ended up needing to visualize $\frac{n!}{2(n-2)!}$ different cells (where n is the number of data channels). However, with a dependency graph visualization, we can reduce the number of colored elements (nodes) to a count of n, by introducing connecting edges. (For systems that are not highly correlated, this will produce far less visual clutter than the corrgram visualization.) Furthermore, we can simplify the undirected dependency graph visualization by eliminating any components of degree 0, if the correlated components are the only ones we wish to see.

We experimented with actually creating this visualization for explorational purposes. First, we ran a system dynamics simulation for our lunar rover described in Chapter 5. We paused the telemetry analysis at a certain time step at which interesting correlated components were present, and examined the data channel correlations at that instant. The correlated components are illustrated in the correlation map visualization in Fig. 7.3. We then isolated the correlated components and illustrated them as an undirected dependency graph, using the steps outlined above. The resulting undirected dependency graph visualization, with positive and negative correlation edges both visible, is shown in Fig. 7.4. In addition, positive and negative correlated components have been isolated into separate graphs for readability and discoverability, as shown in Fig. 7.5 and Fig. 7.6, respectively.

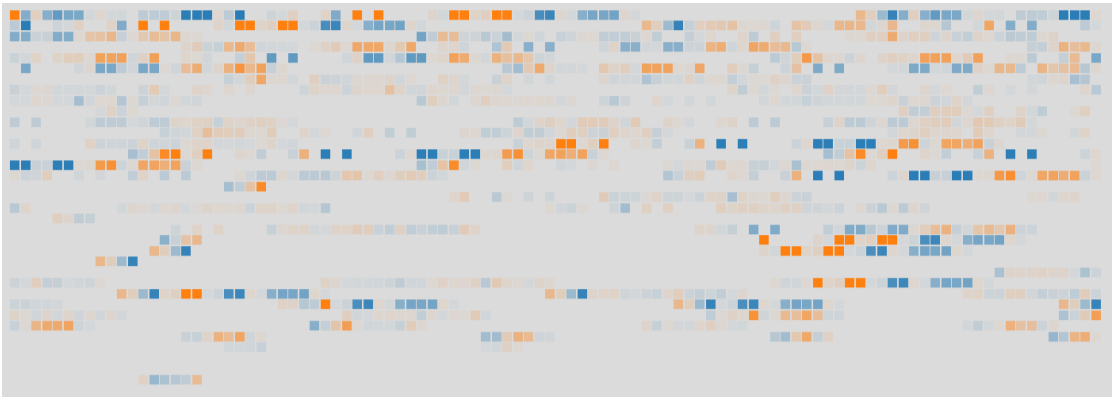

FIGURE 7.3: A snapshot of the correlation map display from a simulated run. Note the strong correlations illustrated by opaque orange (positive) and blue (negative) cells.

FIGURE 7.4: A snapshot of an undirected dependency graph display from a simulated run. Correlated components have been isolated, with edges drawn for all correlation relationships exceeding a certain value ($r^2_{PCC} > 0.8$). Both positive and negative correlation connections are shown. Self-correlations are not shown.

These visualizations present a very different way of viewing the correlative data. While the temporal dimension is still only captured as a snapshot (i.e., the correlative state at only one time point can be shown at a time), correlated components are shown very clearly, and can be understood at a glance. In particular, the intuition behind the positive correlated components in Fig. 7.5 seems clear; the most fully-connected, major clusters are exhibiting behavior which is very similar to each other. (In fact, the lower left cluster channels were all in a state of monotonic decrease, and the upper right cluster channels were in a state of monotonic increase.) The negative correlated components are less obvious, as they don't "cluster" in the same way; however, the negative correlation data can be overlaid onto the positive correlation graphs as a higher-level operation on the clusters. This, perhaps, produces the most informative type of graph; this application

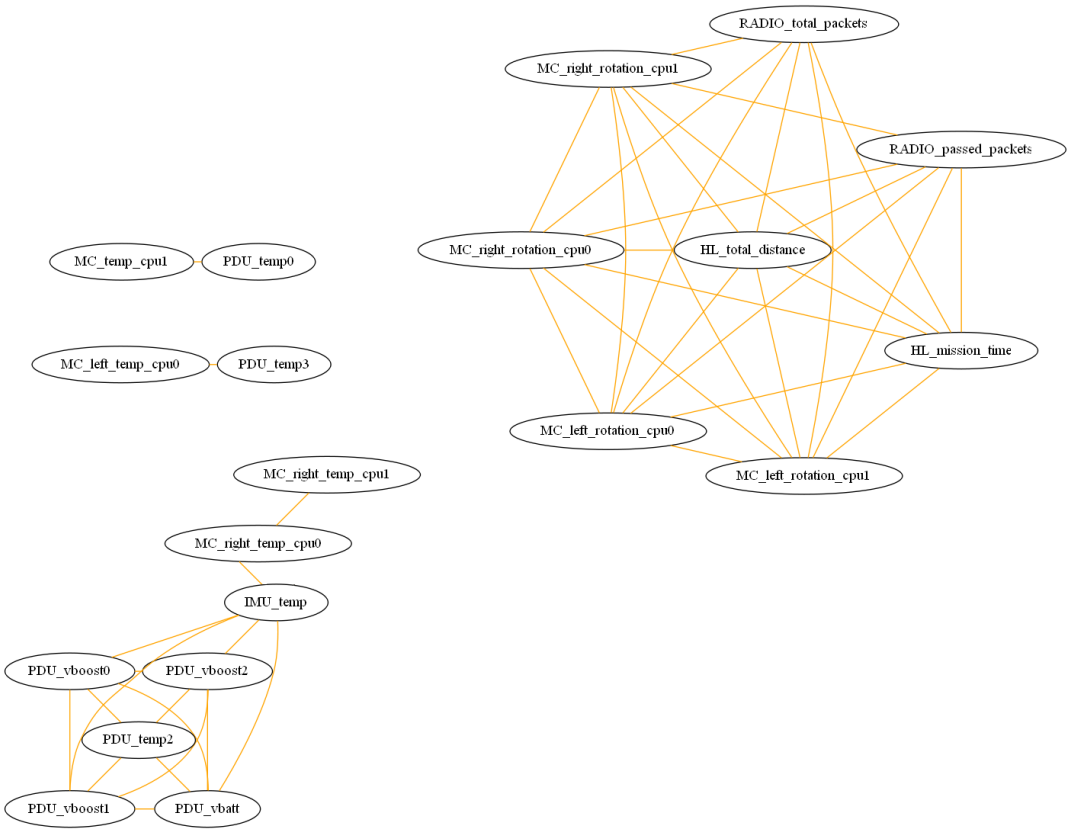

FIGURE 7.5: A snapshot of an undirected dependency graph display from a simulated run. Correlated components have been isolated, with edges drawn for all correlation relationships exceeding a certain value ($r^2_{PCC} > 0.8$). Only positive correlations are shown. Self-correlations are not shown.

is shown in Fig. 7.7.

Note that [35] uses a similar approach for graph visualization of correlation matrices relationships; however, they impose a radial structure on all graph layouts, which losing the clustering advantage of the graph layouts we have produced here.

Another advantage of this technique is that it clearly isolates small groups of correlated components; low-degree subgraphs can point towards noisy data, or towards significant links, but if they are persistent, it seems they may suggest interesting correlations that deviate from the patterns exhibiting by the bulk of the data channels, which tend to correlate due to behavior exhibiting positive and negative monotonicity. However, towards the idea of exploring a visualization which can show the evolution of state and correlative data over time, we will look at another technique in the following section.

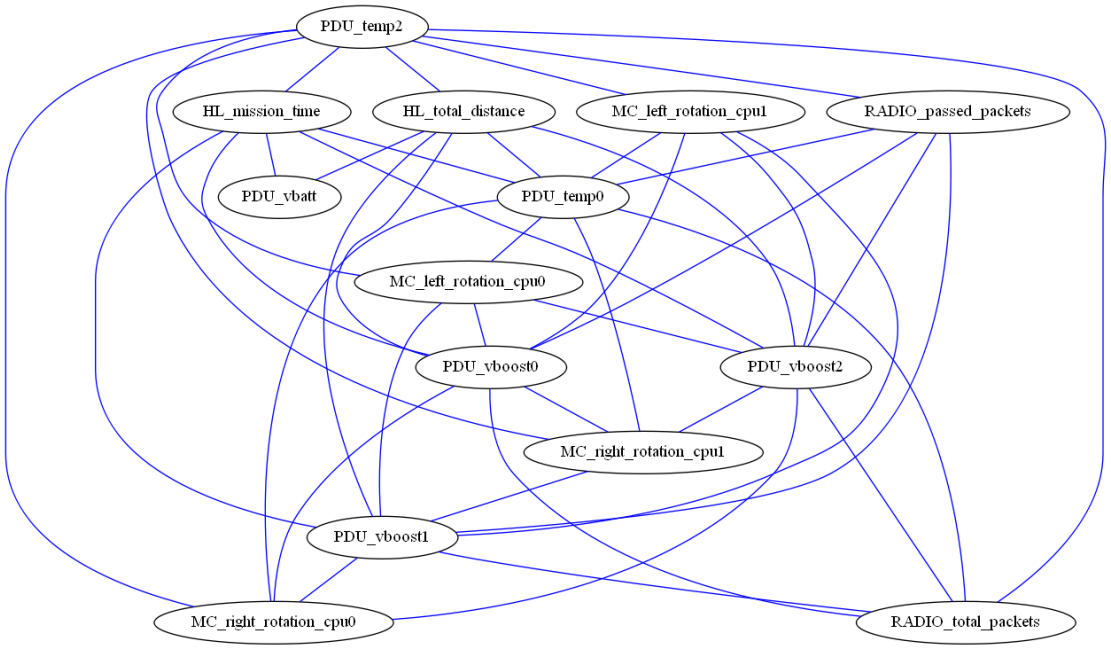

FIGURE 7.6: A snapshot of an undirected dependency graph display from a simulated run. Correlated components have been isolated, with edges drawn for all correlation relationships exceeding a certain value ($r^2_{PCC} > 0.8$). Only negative correlations are shown. Self-correlations are not shown.

7.3 Time Curves

In early 2016, Bach et al presented a powerful new type of visualization tool called "Time Curves" [5]. The time curve is a generic 2D embedding algorithm designed specifically for system state data which changes over time. Time curves visualize system states as a series of points, connected in temporal along curves within the 2D embedding. This allows the viewer to gain an understanding of system behavior by the shape and directions of the curves, and by the grouping of the data points. A visual example of how a time curve is "folded" from an initial linear timeline is shown in Fig. 7.8.

Let's take a closer look at how time curves model system behavior. In the time curves description, states over time are described as a series of "time points," where each time point is a vector $\bar{x} \in \mathbb{R}^n$. For t time points, the state at each time point can be horizontally packed into a matrix $M \in \mathbb{R}^{n \times t}$, which represents the full state history of the system.

The time curves algorithm uses a handful of multidimensional scaling (MDS) techniques which seek to map 2D Euclidean embedding distance to a user-supplied measure of vector distance, such that significantly "similar" states are nearby in the embedding, while significantly "different" states are far away in the embedding. The essential component to generating this embedding is the construction of a symmetric distance matrix $D \in$

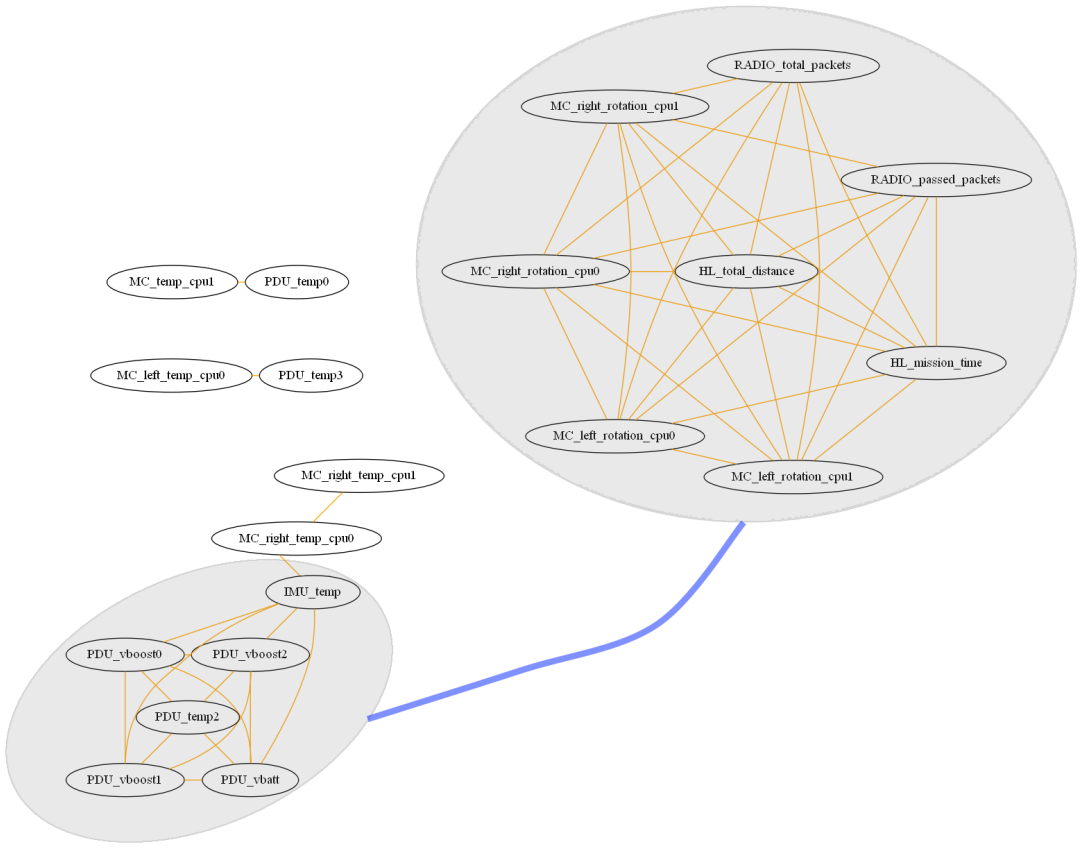

FIGURE 7.7: A snapshot of an undirected dependency graph display from a simulated run. Correlated components have been isolated, with edges drawn for all correlation relationships exceeding a certain value ($r^2_{PCC} > 0.8$). Positive correlations, and negatively correlated subgraphs, are shown. Self-correlations are not shown.

$\mathbb{R}^{t \times t}$, such that $D_{ij} = D_{ji}$ provides a pairwise distance score representing the difference between the states $M_{:,i}$ and $M_{:,j}$. A typical choice for this distance is a Euclidean norm, though many alternatives exist [5]. The time curves algorithm performs an MDS optimization step to map the 2D embedding to the pairwise state distances described in D, producing the final visualization.

The time curves algorithm has many characteristics which make it an appealing candidate for complex system visualization. It exhibits **clustering**, where time points of similar states gather at very close points in the embedding. The viewer's eye naturally processes these clusters as single groups, and they may easily map to system modes. Time curves visualize **cycles**, where a time curve comes back to an earlier point in the embedding after moving elsewhere. This is very useful for modeling cyclic dynamics in complex systems. Time curves also emphasize **transitions** between clusters of points, as the embedding naturally pushes distinct clusters geometrically apart from each other.

Finally, time curves are very extensible, and can be combined with higher-level data which provides further insight into the state at a given time point.

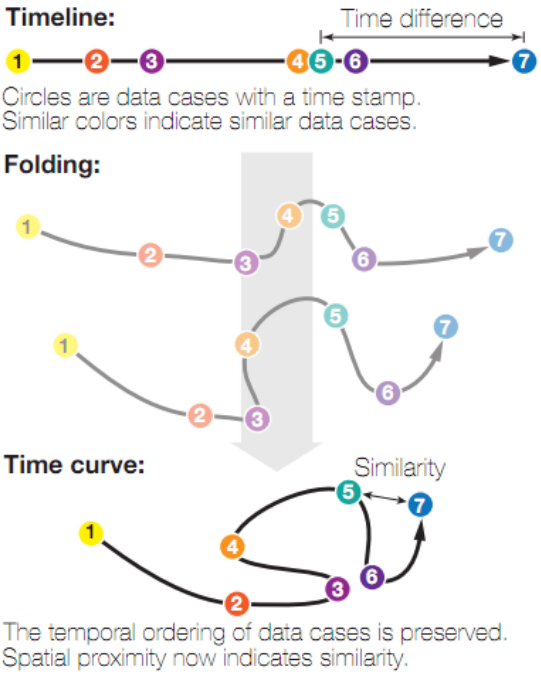

FIGURE 7.8: A time curve is "folded" from an initial linear timeline to bring similar data points close together in the 2D embedding. From [5].

7.4 Applying Time Curves to System Telemetry Data

We set about applying time curves to complex system telemetry data, in order to compare visualization results for raw state telemetry and correlative states, and to see if anomalous system modes could easily be identified in the resulting embeddings. Our data processing pipeline for raw state telemetry visualization was as follows:

1. At regular intervals (e.g., once per second), record the telemetered system state into a vector $\bar{x} \in \mathbb{R}^n$.

2. Once the mission is complete, horizontally concatenate all state vectors to produce a state history matrix $M \in \mathbb{R}^{n \times t}$.

3. Generate a symmetric distance matrix $D \in \mathbb{R}^{t \times t}$, such that $D_{ij} = D_{ji}$ provides a pairwise distance score representing the difference between the states $M_{:,i}$ and $M_{:,j}$.

4. Use the time curves algorithm to generate a 2D embedding of the t states described in D.

In contrast, our pipeline for generating visualizations for correlated telemetry was as follows:

1. Generate a state history matrix M as with the raw state telemetry case.
2. For a given window size w, generate windowed "sample matrices" $W \in \mathbb{R}^{n \times w}$, such that $W_i = M_{:,i:i+15}$.
3. For each windowed matrix W, generate a correlative matrix $R \in \mathbb{R}^{n \times n}$ (via PCC, Spearman's Rho, or Kendall's Tau), such that R_{ij} gives the correlation score of $W_{i,:}$ and $W_{j,:}$. (Note that incalculable correlations, such as those with data channels of zero variance, are set to be 0.)
4. Flatten each windowed matrix R, and concatenate them horizontally into a matrix $C \in \mathbb{R}^{n^2 \times (t-w)}$, such that each column of C represents the total correlative state of the system at a time point.
5. Generate a symmetric distance matrix $D \in \mathbb{R}^{(t-w) \times (t-w)}$, such that $D_{ij} = D_{ji}$ provides a pairwise distance score representing the difference between the states $C_{:,i}$ and $C_{:,j}$.
6. Use the time curves algorithm to generate a 2D embedding of the $(t - w)$ states described in D.

The algorithms described above were applied on the user testing data set used previously during user evaluations. This data set closely simulates telemetry from a lunar mission, in which the events described in Tbl. 7.1 take place at pre-set times. Each of these events triggers numerous differences in simulated environment state, which in turn causes major sensor data differences. High levels of Gaussian white noise have been introduced into the sensor data to approximate real sensor behavior.

The annotated time curve visualization of the raw state telemetry progression is shown in Fig. 7.9. Similarly annotated visualizations of the PCC, Spearman's Rho, and Kendall's Tau correlative state are shown in Figs. 7.10, 7.11, and 7.12, respectively. (These correlative visualizations were generated with a 15-second time window.)

7.5 Discussion

The time curves visualization does a pleasing job of showing the varying modes of system operation, and allows the viewer to follow state progression easily. The major moves between groups of points correspond pretty much exactly to the times during the simulation when the rover started seeing telemetry differences from different environmental conditions.

It's also evident by inspection that the state progression from correlative analysis in Figs. 7.10, 7.11, and 7.12 is more distinct and easy to understand than that visualized using raw telemetry in Fig. 7.9. We believe that there are several reasons for these core differences. First, monotonically increasing data channels (e.g., time or wheel rotations) can make it hard to isolate modes, as state vectors containing this data appear to represent different states at each time step, even when they really only indicate one mode of behavior. These data channels can "smear" the state across a 2D embedding, due to a steady, constant distance introduced through these increasing channels. Unless they are deliberately removed from the data–which could, as a side-effect, remove valuable data crucial to understanding system behavior–these data channels will result in behavior like that seen in Fig. 7.10, where the initial state and final state are actually very similar system modes, but are distint within the 2D embedding.

Additionally, changing environmental dynamics, and thus, the inputs into the system, can change as the environment changes, but if the vehicle continues to handle them in the same way, then the correlative relationships among its data channels may maintain more consistency in spite of these changing dynamics. Correlative analysis, in this way, allows a human operator to more directly investigate internal system relationships (although external dynamics still manifest themselves in the correlative data as correlations of sensor value behavior with time and other monotonically increasing values).

Finally, it appears that the correlative calculations, as they act on a large time window, appear to have had a denoising side-effect on the data, and allow real changes in correlation to clearly "push" the time points around the embedding, making state transitions more clear–while state transitions within the raw telemetry visualization are subtle and mostly buried in the rest of the data.

Another interesting observation about the data produced, in particular visible within the correlative time curve visualizations, is the presence of additional states not foreseen in the event timeline, such as the cluster of time points between events **1** and **2**. Upon close analysis, this cluster of time points appears to be a "hallucinated event," as a side-effect of several fault states being triggered at the same time and cascading effects on telemetered measurements as temperatures and generated voltages drop. The application of the PCC correlative algorithm emphasizes this state, likely because of its limitations with respect to accurately capturing nonlinear associations and the coarse, ordinal nature of a large amount of the discrete telemetry data.

Event Number	Timestamp	Event Description
0	0s	Rover begins traveling forward along smooth terrain.
1	188s	Rover enters crater; begins descending into crater.
2	287s	Rover enters shade, causing temp, comms, and power drops.
3	300s	Rover begins traversing smooth bottom of crater.
4	330s	Rover begins climbing out of crater.
5	343s	Rover exits shade; continues uphill.
6	534s	Rover emerges from crater and enters smooth terrain.
7	594s	Rover enters choppy terrain.
8	643s	Rover wheel has fault; rover stops moving.

TABLE 7.1: Events during a visualized user simulation are shown. Cross-reference "Event Numbers" with labels in Figs. 7.10, 7.12, 7.11 and 7.9 to see correspondence.

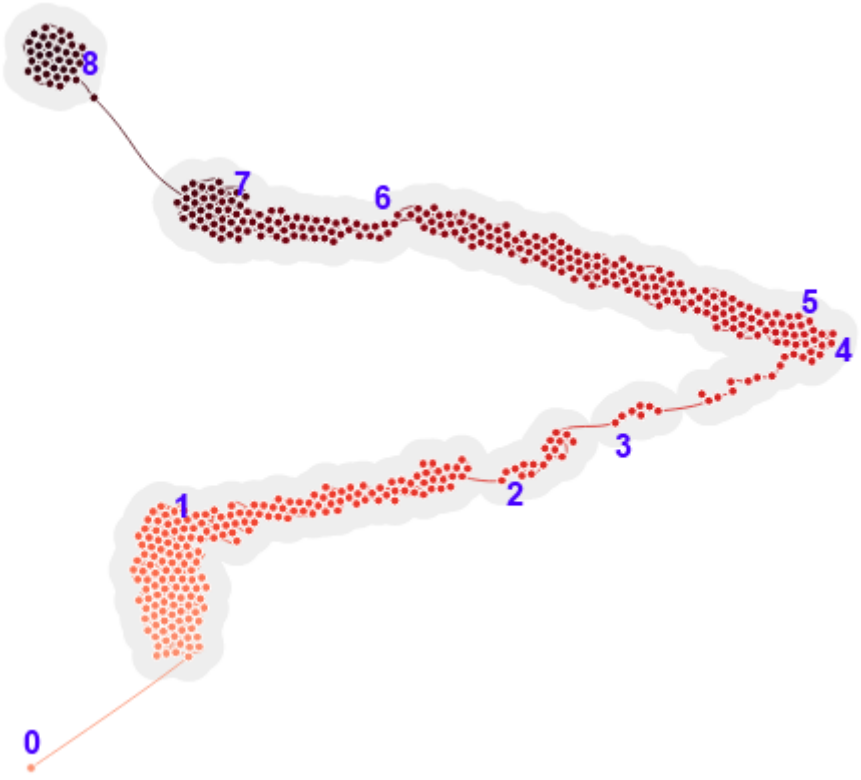

FIGURE 7.9: A time curve embedding visualizing mission events using raw telemetry state. Event annotations are described in Tbl. 7.1.

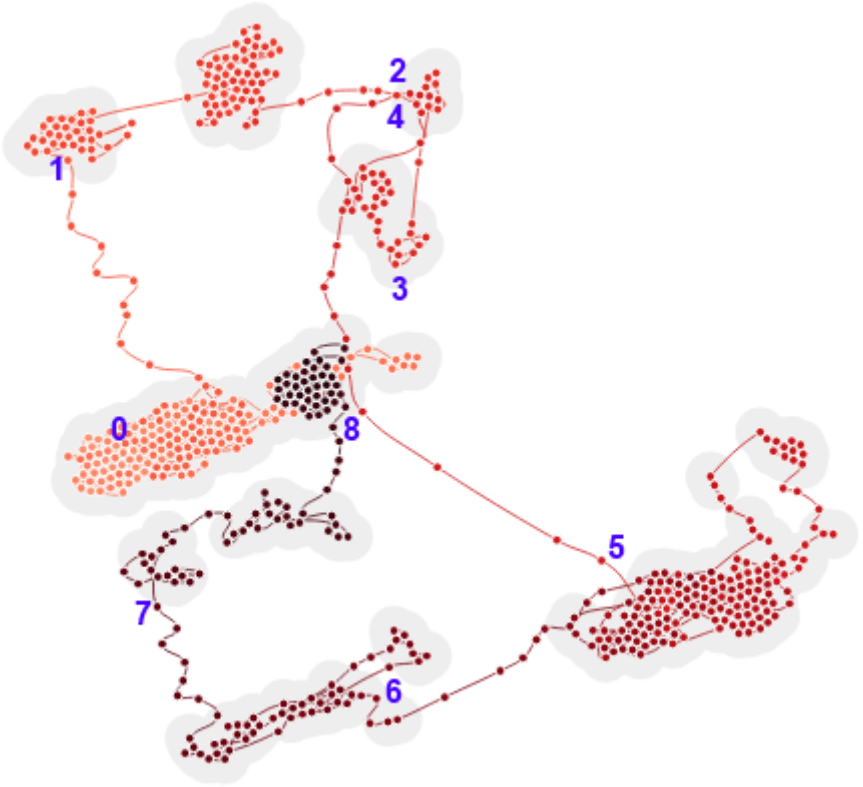

FIGURE 7.10: A time curve embedding visualizing mission events using PCC correlation state. Event annotations are described in Tbl. 7.1.

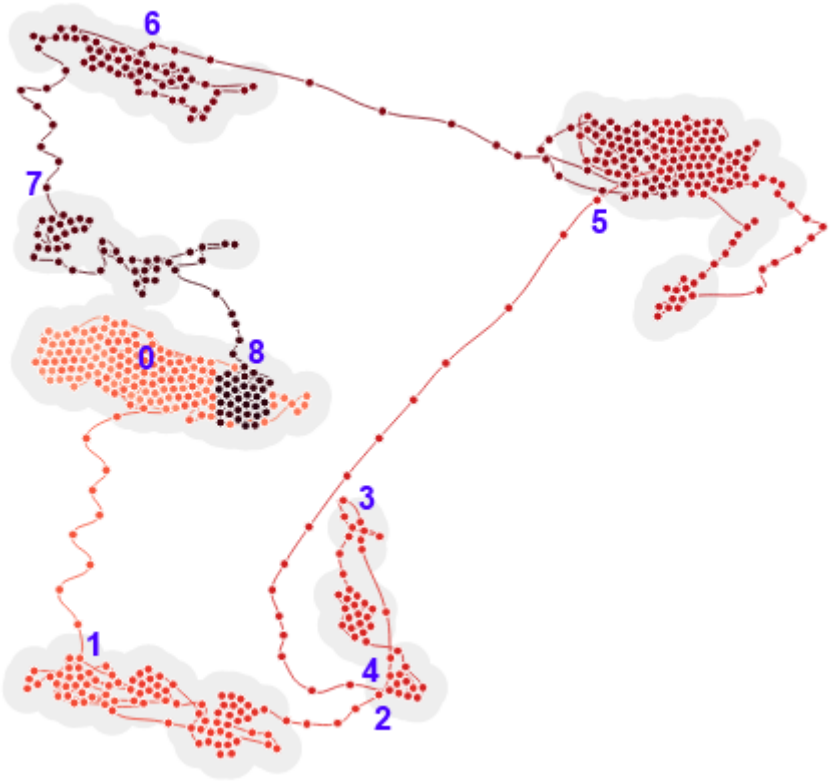

FIGURE 7.11: A time curve embedding visualizing mission events using Spearman's Rho correlation state. Event annotations are described in Tbl. 7.1.

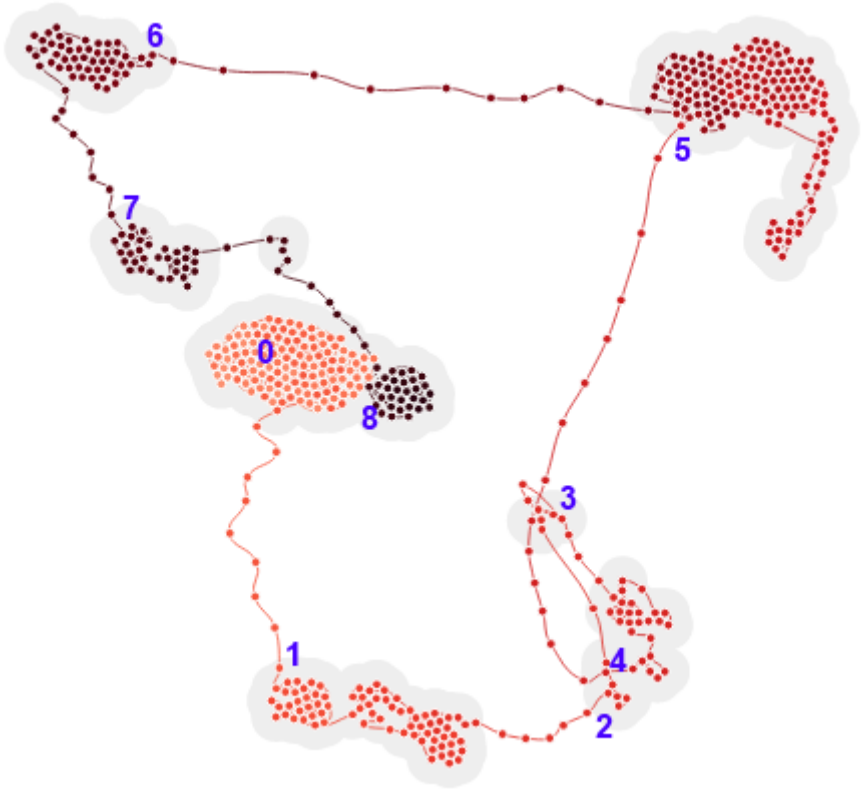

FIGURE 7.12: A time curve embedding visualizing mission events using Kendall's Tau correlation state. Event annotations are described in Tbl. 7.1.

Chapter 8

Conclusion and Future Work

Purpose:

recap of intro:

Purpose:

8.1

8.2 Future Work

-to lay out the basic idea of the paper -explain motivating problem of paper -briefly mention what aerospace faults are -say why they're hard to troubleshoot -explain a little bit about root cause analysis -talk about what I'm trying to do in the paper -which is to look at data connections, correlation, etc. on a real aerospace system to try to figure out how data discovery can best be achieved and how patterns can be found with math methods -these aren't easy to show, so also look into viz -implement for actual system -evaluate -reassess -make modifications to both math and viz -test again -find conclusions

Appendix A

Relevant Code Samples

This appendix contains a sampling of some of the most relevant code used to generate the results shown in this paper. Over 10,000 lines of code were written for this research, but only an illustrative subset of this code is shown here.

A.1 Generating Example Corrgrams for PCC, Kendall's Tau, and Spearman's Rho Correlations (MATLAB)

```
% Initial cleanup
close all;
clear;
clc;

% Make a random matrix of data channel values
channel_count = 6;
time_slots = 8;
channel_data_rand = rand(channel_count, time_slots);

% Make purposefully interesting channel data
channel1_data = [0 10 101 102 0 10 101 102];
channel2_data = 4 * ones(1, 8) + rand(1, 8)*0.5;
channel3_data = 6 * ones(1, 8) + rand(1, 8)*0.3;
channel4_data = [1.2 2.3 3.4 4.4 4.5 3.5 2.2 1.1];
channel5_data = -[10 20 30 40 40 30 20 10];
channel6_data = [1 100 500 2000 1 100 500 2000];

channel_data = [channel1_data; channel2_data; channel3_data; channel4_data;
    channel5_data; channel6_data];

% Generate Pearson correlation coefficient
corr_mx_pcc = corr(channel_data', 'type', 'Pearson');
% Generate Kendall's tau
corr_mx_tau = corr(channel_data', 'type', 'Kendall');
% Generate Spearman's rho
```

```
corr_mx_rho = corr(channel_data', 'type', 'Spearman');

% Plot PCC
figure(1)
imagesc(corr_mx_pcc);
colormap(gray(256));
colorbar;
title('PCC Correlation Map Example');
xlabel('Data Channels');
ylabel('Data Channels');

% Plot tau
figure(2)
imagesc(corr_mx_tau);
colormap(gray(256));
colorbar;
title('Kendall''s Tau Correlation Map Example');
xlabel('Data Channels');
ylabel('Data Channels');

% Plot rho
figure(3)
imagesc(corr_mx_rho);
colormap(gray(256));
colorbar;
title('Spearman''s Rho Correlation Map Example');
xlabel('Data Channels');
ylabel('Data Channels');
```

A.2 Generating Undirected Graph from Filtered Correlation Data (Python)

```
import itertools
import sys
from sets import Set
import csv

csv_filename = 'strong_corrs_for_undirected_graph.csv'
undirected_graph_filename = 'undirected.dot'
telem_filename = 'telem.csv'

channel_names = Set()

positive_pairs = Set()
negative_pairs = Set()
paired_nodes = Set()

with open(csv_filename, 'rb') as csvfile:
    csv_reader = csv.DictReader(csvfile)

    header_skipped = False
```

```
    for row in csv_reader:
        row_name = row['']
        # Go through all of the different columns and create connections if
    necessary
        for key in row.keys():
            if key != '' and row_name != key:
                if row[key] == 'P':
                    print "Positive correlation between %s and %s." % (row_name,
    key)
                    positive_pairs.add(Set([row_name, key]))
                    paired_nodes.add(row_name)
                    paired_nodes.add(key)
                elif row[key] == 'N':
                    print "Negative correlation between %s and %s." % (row_name,
    key)
                    negative_pairs.add(Set([row_name, key]))
                    paired_nodes.add(row_name)
                    paired_nodes.add(key)
                elif row[key] == '':
                    # print "No strong correlation between %s and %s." % (
    row_name, key)
                    pass
                else:
                    print "Undefined correlation between %s and %s!!!" % (
    row_name, key)

# Now, let's print out to the graph file
with open(undirected_graph_filename, 'w') as outfile:
    outfile.write('graph {\n')
    outfile.write('node [shape=ellipse];\n\n')
    outfile.write('// nodes\n')
    for node in paired_nodes:
        outfile.write(node + ' ');
    outfile.write(';\n\n')
    outfile.write('// positive edges\n')
    outfile.write('edge [color=Orange];\n')
    for positive_pair in positive_pairs:
        node1, node2 = list(positive_pair)
        outfile.write(node1 + ' -- ' + node2 + ';\n')
    outfile.write('\n// negative edges\n')
    outfile.write('edge [color=Blue];\n')
    for negative_pair in negative_pairs:
        node1, node2 = list(negative_pair)
        outfile.write(node1 + ' -- ' + node2 + ';\n')


    outfile.write('}')
```

A.3 Generating Time Curve Distance Matrix from Raw Telemetry (Python)

```
import csv
from sets import Set
import datetime
import json
import sys
import numpy as np

telem_filename = 'telem.csv'
timecurve_json_filename = 'timecurve_full_data.json'
dataset_name = 'PFM2 User Test Data (Full Data)'

# Gets a "distance" between two arrays of values
def get_distance(arr1, arr2):
    # print "diffing:"
    # print "1: %s" % arr1
    # print "2: %s" % arr2

    return np.linalg.norm(arr1 - arr2)

# Take an array, and "standardizes" it by subtracting the mean then
# dividing by the standard deviation
def standardize(arr):
    # print "Working on array: %s" % str(arr)
    mean = np.mean(arr)
    stddev = np.std(arr)

    if stddev == 0:
        return arr
    else:
        # print "Mean: %s" % str(mean)
        # print "Std dev: %s" % str(stddev)
        # print "Returning %s" % str((arr - mean) / stddev)

        return (arr - mean) / stddev

with open(telem_filename, 'rb') as csvfile:
    csv_reader = csv.reader(csvfile)

    channel_names = Set()

    # First, just grab the channel names
    for row in csv_reader:
        # Parse out data
        channel_name = row[0]

        channel_names.add(channel_name)

    # Now, make channel_names a list to ensure the ordering doesn't change
    channel_names = list(channel_names)

    print "Collected names of %s distinct data channels." % len(channel_names)
    # for channel_name in channel_names:
    #     print channel_name
```

```
# Initialize this value so it'll be global
earliest_time_struct = None

with open(telem_filename, 'rb') as csvfile:
    csv_reader = csv.reader(csvfile)

    # Now that we have channel names, let's make data entries for the entire
    state
    # of the system at each second interval. Let's start by ingesting all of
    # the data and then looking it by the timestamps.
    data_points = []
    for row in csv_reader:
        channel_name = row[0]
        value = row[1]
        timestamp = row[2]
        fault_level = row[3]

        # We won't worry about ingesting the fault_level for this
        # (we might use it later...)
        data_points.append((timestamp, channel_name, value))

    print "Processed data for %s data points." % len(data_points)
    print

    # Sort time points
    data_points = sorted(data_points, key=lambda x: x[0])

    print "Extracted and sorted data points by time. See below for a few examples
    :"
    print data_points[0]
    print data_points[1]
    print data_points[2]
    print '...'
    print data_points[-3]
    print data_points[-2]
    print data_points[-1]
    print

    # Now, let's coalesce points into buckets to reduce their size
    earliest_time = data_points[0][0]
    earliest_time_struct = datetime.datetime.strptime(earliest_time, "%m/%d/%Y %H
    :%M:%S")

    latest_time = data_points[-1][0]
    latest_time_struct = datetime.datetime.strptime(latest_time, "%m/%d/%Y %H:%M
    :%S")

    # Calculate difference and make buckets
    timespan = latest_time_struct - earliest_time_struct
    timespan_seconds = timespan.total_seconds()
    print "Total elapsed seconds of data:", timespan_seconds
    print

    # Divide up timespan into interval seconds-sized buckets
    time_bucket_size = 1
```

```
time_bucket_data = {}

for floor in xrange(0, int(timespan_seconds)+1, time_bucket_size):
    # Initialize with an empty dictionary mapping channel names to a list of
data values
    time_bucket_data[floor] = {channel_name: [] for channel_name in
channel_names}

# Now we'll go through all of the data points and dump them into the buckets
for data_point in data_points:
    time_string = data_point[0]
    time_struct = datetime.datetime.strptime(time_string, "%m/%d/%Y %H:%M:%S"
)
    timestamp_seconds = (time_struct - earliest_time_struct).total_seconds()
    bucketed_timestamp = int(timestamp_seconds - timestamp_seconds %
time_bucket_size)

    # Now, add a value for the channel name in this timestamp bucket
    channel_name = data_point[1]
    value = float(data_point[2])

    time_bucket_data[bucketed_timestamp][channel_name].append(value)

# Now, go through and coalesce (average) all data
for time_bucket_floor in xrange(0, int(timespan_seconds)+1, time_bucket_size)
:
    for channel_name in (time_bucket_data[time_bucket_floor].keys()):
        # Replace each list of values with the average of the values
        values = time_bucket_data[time_bucket_floor][channel_name]
        if len(values) > 0:
            # print "setting ", time_bucket_data[time_bucket_floor][
channel_name]
            time_bucket_data[time_bucket_floor][channel_name] = float(sum(
values)) / float(len(values))
        else:
            # No values... take the value for the previous timestep, if there
 is one
            if time_bucket_floor != 0:
                time_bucket_data[time_bucket_floor][channel_name] =
time_bucket_data[time_bucket_floor - time_bucket_size][channel_name]
            else:
                # Otherwise, just zero out the data
                time_bucket_data[time_bucket_floor][channel_name] = 0.0

# Now, time_bucket_data contains all of the averaged, bucketed data. This is
great!!
# Let's start treating this data as matrix data, so we can do some numerical
things
# to it. First, let's make an n x t matrix, where n is the number of data
channels,
# and t is the number of timesteps.

num_channels = len(channel_names)
data_matrix_initialized = False
```

```
# We'll make each timestamp into a column vector
for time_bucket_floor in time_bucket_data.keys():
    data_vector = np.empty((num_channels, 1))

    # We'll iterate using the channel name list so we're always using the
same channel ordering
    for i, channel_name in enumerate(channel_names):
        # print 'i:', i
        # print 'channel_name:', channel_name
        # print 'time_bucket_floor:', time_bucket_floor
        # print 'time_bucket_data[time_bucket_floor][channel_name]:',
time_bucket_data[time_bucket_floor][channel_name]
        data_vector[i, 0] = time_bucket_data[time_bucket_floor][channel_name]

    # Now that we've filled up a vector, let's append it to the data matrix
    if not data_matrix_initialized:
        data_matrix = data_vector
        data_matrix_initialized = True
    else:
        # print "data_vector size: %s" % str(data_vector.shape)
        # print "data_matrix size: %s" % str(data_matrix.shape)
        data_matrix = np.hstack((data_matrix, data_vector))

print "Combined all data into a giant matrix."
print "Size of assembled data matrix: %s" % str(data_matrix.shape)
print "First few values of %s from data matrix:" % channel_names[0]
print str(data_matrix[0,:5])
print

# Finally, we need to normalize using the mean and variance of each row,
# in order to be able to calculate "distances" in an unbiased way
standardized_data_matrix = np.apply_along_axis(standardize, 1, data_matrix)

print "Standardardized data matrix."
print "Size of assembled data matrix: %s" % str(standardized_data_matrix.
shape)
print "First few values of %s from data matrix:" % channel_names[0]
print str(standardized_data_matrix[0,:5])
print

# Now we've standardardized the data matrix! Heck yeah!
# For our last trick, we'll make a symmetric distance matrix (where
# the value at n x m shows the "distance" between column n and column m)
num_timesteps = standardized_data_matrix.shape[1]
distance_matrix = np.empty((num_timesteps, num_timesteps))

for (col_num, row_num), value in np.ndenumerate(distance_matrix):
    # print 'Trying to fill in spot at row %s, col %s' % (row_num, col_num)
    distance_matrix[row_num, col_num] = get_distance(standardized_data_matrix
[:, row_num], standardized_data_matrix[:, col_num])

# Now, we're done generating our distance matrix
print "Distance matrix generated. Here's a snippet:"
print str(distance_matrix[:5, :5])
print
```

```
    # Let's dump this out into a .json file for timecurve conversion
    json_dict = {}
    json_dict['distancematrix'] = distance_matrix.tolist()

    times = [str(earliest_time_struct + datetime.timedelta(seconds=t)) for t in
    sorted(time_bucket_data.keys())]

    json_dict['data'] = [{'name': dataset_name, 'timelabels': times}]

    # Finally, write the .json file
    with open(timecurve_json_filename, 'w') as outfile:
        json.dump(json_dict, outfile)
        print "Outputted time curve data to .json file (%s)" %
    timecurve_json_filename
```

A.4 Generating Correlated Distance Matrices from Bucketed Telemetry (MATLAB)

```
% Initial cleanup
close all;
clear;
clc;

% Load bucketed telemetry data
data_matrix = csvread('one_second_bucketed_telem_output.csv');
channel_count = size(data_matrix,1);
timestep_count = size(data_matrix,2);

% Size of the window in which to calculate the correlation
corr_window = 15;

% Matrices to hold changing PCC, Tau, and Rho correlations over time
pcc_corr_over_time = zeros(channel_count^2, timestep_count - corr_window + 1);
tau_corr_over_time = zeros(channel_count^2, timestep_count - corr_window + 1);
rho_corr_over_time = zeros(channel_count^2, timestep_count - corr_window + 1);

% Let's generate correlation data at each time step. We'll advance at
% the rate of one second per step, and we'll look back to get the
% instantaneous correlation based on the last corr_window measurements. This
    means
% we'll start at corr_window seconds in.
disp('Calculating correlation data for each time step:');
for step = corr_window:timestep_count,
   disp(step);
   % Generate a slice of the data matrix using the last corr_window measurements
   data_matrix_slice = data_matrix(:, step-corr_window+1:step);

   % Generate Pearson correlation coefficient
    corr_mx_pcc = corr(data_matrix_slice', 'type', 'Pearson');
    % Generate Kendall's tau
```

```
    corr_mx_tau = corr(data_matrix_slice', 'type', 'Kendall');
    % Generate Spearman's rho
    corr_mx_rho = corr(data_matrix_slice', 'type', 'Spearman');

    % Unroll each of the correlation matrices and put into a matrix
    pcc_corr_over_time(:, step-corr_window+1) = reshape(corr_mx_pcc, [], 1);
    tau_corr_over_time(:, step-corr_window+1) = reshape(corr_mx_tau, [], 1);
    rho_corr_over_time(:, step-corr_window+1) = reshape(corr_mx_rho, [], 1);
end

% Anything that had a standard deviation of 0 will have a NaN correlation,
% which screws us up. Let's replace NaN correlations with 0.
pcc_corr_over_time(isnan(pcc_corr_over_time)) = 0;
tau_corr_over_time(isnan(tau_corr_over_time)) = 0;
rho_corr_over_time(isnan(rho_corr_over_time)) = 0;

% Square correlation values to make them stand out more (preserve signs)
pcc_corr_over_time = (pcc_corr_over_time.^2) .* sign(pcc_corr_over_time);
tau_corr_over_time = (tau_corr_over_time.^2) .* sign(tau_corr_over_time);
rho_corr_over_time = (rho_corr_over_time.^2) .* sign(rho_corr_over_time);

% Create distance matrices for time points
pcc_distance_matrix = zeros(timestep_count - corr_window + 1);
tau_distance_matrix = zeros(timestep_count - corr_window + 1);
rho_distance_matrix = zeros(timestep_count - corr_window + 1);

disp('Calculating distance matrices:');

% Now, we'll fill in these distance matrices
for i = 1:timestep_count - corr_window + 1,
    for j = 1:timestep_count - corr_window + 1,
        % Fill in the cell of each of the distance matrices at (i, j)
        pcc_distance = norm(pcc_corr_over_time(:, i) - pcc_corr_over_time(:, j));
        tau_distance = norm(tau_corr_over_time(:, i) - tau_corr_over_time(:, j));
        rho_distance = norm(rho_corr_over_time(:, i) - rho_corr_over_time(:, j));

        pcc_distance_matrix(i, j) = pcc_distance;
        tau_distance_matrix(i, j) = tau_distance;
        rho_distance_matrix(i, j) = rho_distance;
    end
    disp(i);
end

% These distance matrices will be used to generate .csv files, which we'll
% load in Python and use to generate our file .csv files.
pcc_distance_matrix_file = 'pcc_distance_matrix.csv';
tau_distance_matrix_file = 'tau_distance_matrix.csv';
rho_distance_matrix_file = 'rho_distance_matrix.csv';

% Write files
csvwrite(pcc_distance_matrix_file, pcc_distance_matrix);
csvwrite(tau_distance_matrix_file, tau_distance_matrix);
csvwrite(rho_distance_matrix_file, rho_distance_matrix);
```

Bibliography

[1] Google sponsors lunar x prize to create a space race for a new generation, 2007.

[2] Spaceflight 101. Nasa report on antares launch failure places blame on aj26 engines, 2015.

[3] Francis J Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, 1973.

[4] Olivier Aycard and Richard Washington. State identification for planetary rovers: Learning and recognition. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 1163–1168. IEEE, 2000.

[5] Benjamin Bach, Conglei Shi, Nicolas Heulot, Tara Madhyastha, Tom Grabowski, and Pierre Dragicevic. Time curves: Folding time to visualize patterns of temporal evolution in data. *Visualization and Computer Graphics, IEEE Transactions on*, 22(1):559–568, 2016.

[6] George Cancro, Russell Turner, Lilian Nguyen, Angela Li, Deane Sibol, John Gersh, Christine Piatko, Jaime Montemayor, and Priscilla McKerracher. An interactive visualization system for analyzing spacecraft telemetry. In *Aerospace Conference, 2007 IEEE*, pages 1–9.

[7] Dauna Coulter. Down the lunar rabbit-hole, 2010.

[8] Richard Dearden, Thomas Willeke, Reid Simmons, Vandi Verma, Frank Hutter, and Sebastian Thrun. Real-time fault detection and situational awareness for rovers: Report on the mars technology program task. In *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, volume 2, pages 826–840. IEEE, 2004.

[9] Michael Friendly. Corrgrams: Exploratory displays for correlation matrices. *The American Statistician*, 56(4):316–324, 2002.

[10] Robert Hammett and Robert Luppold. Application of syndrome pattern-matching approach to fault isolation in avionic systems. In *Digital Avionics Systems Conference, 1991. Proceedings., IEEE/AIAA 10th*, pages 56–61. IEEE, 1991.

[11] Niklas Holsti and Matti Paakko. Towards advanced fdir components. *Data Systems in Aerospace, DASIA 2001*, 2001.

[12] Inseok Hwang, Sungwan Kim, Youdan Kim, and Chze Eng Seah. A survey of fault detection, isolation, and reconfiguration methods. *Control Systems Technology, IEEE Transactions on*, 18(3):636–653, 2010.

[13] James Kurien and María DR Moreno. Intrinsic hurdles in applying automated diagnosis and recovery to spacecraft. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 40(5):945–958, 2010.

[14] J Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. OUP Oxford, 2013.

[15] Jiliang Lin and Jingping Jiang. Fault detection and analysis of control software for a mobile robot. In *Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on*, volume 1, pages 862–866. IEEE, 2006.

[16] Chet Lo and Cynthia Furse. Noise-domain reflectometry for locating wiring faults. *Electromagnetic Compatibility, IEEE Transactions on*, 47(1):97–104, 2005.

[17] Wolfgang Müller and Heidrun Schumann. Visualization methods for time-dependent data-an overview. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 1, pages 737–745. IEEE, 2003.

[18] Randall Munroe. xkcd: Correlation, 2016.

[19] DJ Murdoch and ED Chow. A graphical display of large correlation matrices. *The American Statistician*, 50(2):178–180, 1996.

[20] JAXA Institute of Space and Astronautical Science. On the status of the procyon nanosatellite probe, 2015.

[21] Matti Paakko, Petri Myllymäki, Niklas Holsti, and Henry Tirri. Bayesian networks for advanced fdir. In *Proceedings of the ESA Workshop on On-Board Autonomy*, pages 311–318, 2001.

[22] RJ Rummel. Understanding correlation. 1976.

[23] Mark Schwabacher and Robert Waterman. Pre-launch diagnostics for launch vehicles. In *Aerospace Conference, 2008 IEEE*, pages 1–8. IEEE, 2008.

[24] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[25] SpaceX. Crs-7 investigation update, 2015.

[26] Laird Statistics. Kendall's tau-b using spss statistics, 2016.

[27] Laird Statistics. Spearman's rank-order correlation, 2016.

[28] NASA Independent Review Team. Orb3 accident investigation report. Technical report, National Aeronautics and Space Administration, October 2015.

[29] Unity Technologies. Unity powers nasa virtual mars rover experience, 2012.

[30] Apollo Spacecraft Program Office Test Division. *Apollo 13 Technical Air-to-Ground Voice Transcription*. National Aeronautics and Space Administration Manned Spacecraft Center, 1970.

[31] Massimo Tipaldi and Bernhard Bruenjes. Spacecraft health monitoring and management systems. In *Metrology for Aerospace (MetroAeroSpace), 2014 IEEE*, pages 68–72. IEEE, 2014.

[32] Thamara Villegas, María Jesús Fuente, and Miguel Rodríguez. Principal component analysis for fault detection and diagnosis. experience with a pilot plant. In *Proceedings of the 9th WSEAS international conference on computational intelligence, man-machine systems and cybernetics*, pages 147–152, 2010.

[33] Bruce K Walker and Eliezer Gait. Fault detection threshold determination technique using markov theory. *Journal of Guidance, Control, and Dynamics*, 2(4):313–319, 1979.

[34] Takehisa Yairi, Yoshinobu Kawahara, Ryohei Fujimaki, Yuichi Sato, and Kazuo Machida. Telemetry-mining: a machine learning approach to anomaly detection and fault diagnosis for space systems. In *Space Mission Challenges for Information Technology, 2006. SMC-IT 2006. Second IEEE International Conference on*, pages 8–pp. IEEE, 1992.

[35] Shi-Tao Yeh, GlaxoSmithKline, and PA King of Prussia. Exploratory visualization of correlation matrices. In *NorthEast SAS Users Group (NESUG) conference 2007*, 2007.