Virtual Ground Station for Automated Spacecraft Operations

by

Varsha Parthasarathy

A thesis
presented to the University of Manitoba
in fulfilment of the
thesis requirement for the degree of
Master of Science
in
Mechanical Engineering

Winnipeg, Manitoba, Canada 2021

Abstract

Low-earth orbit satellite constellations provide important services over very large areas in a uniform way. Traditional spacecraft communication typically requires manually-staffed ground stations with space systems experts controlling and monitoring bus and payload systems during each pass. These onerous requirements limit the ability to access valuable space assets, more so in the case of large constellations of satellites.

This thesis presents a virtual ground station (VGS) with a real-time virtual satellite model (VSM) and a fault-management system based on industrial statistical process control (SPC) techniques and time-domain feature extraction. The VSM is in continuous view of the VGS at all times and allows the operators to send and receive data as required without waiting for a pass. The operators always interact with the VSM through a graphical user interface (GUI) terminal as opposed to the spacecraft itself such that the VSM mimics the actual satellite as much as possible. The VGS streamlines spacecraft operations by managing every real pass, uploads stored commands when a pass occurs, automatically downloads telemetry and maintains the VSM. This eliminates trivial housekeeping activities and lets the experts focus on complex problems. The VGS also contains a real-time orbit propagator that provides the real-time position and velocity of the satellite and lets the operators visualize the mission in 3D. In this thesis, the VSM uses the power subsystem as an example and takes the form of a real-time power subsystem simulator of the spacecraft. The fault-management system employs custom algorithms to monitor telemetry from the spacecraft and compare it to the predicted telemetry from the VSM to perform early fault diagnosis. The specific faults considered are the loss of a solar string(s), increase in the battery's internal resistance and excessive power consumption onboard the spacecraft.

A unique testbed consisting of a simulation engine with an actual satellite model (ASM) and a serial communication protocol is presented. It is used to demonstrate the functions of the VGS through various scenarios during a typical interaction between the VGS and the satellite. Some of these scenarios include initiation of communication with the spacecraft, automatic telemetry downloading, anomaly detection, real-time data requests and storing and uploading commands to the spacecraft.

Acknowledgements

I would like to express the deepest appreciation to my advisor and committee chair Professor Philip Ferguson for all his invaluable supervision, support and patience throughout this journey. He always guided me positively and made me feel confident in my abilities. His insight and knowledge into the space industry steered me through my research. His feedback pushed me to sharpen my thinking and learn more than I would have imagined.

I would like to thank Macdonald, Dettwiler And Associates Corporation and NSERC for funding this research through their Collaborative Research and Development Grant. Thank you to the Canadian Space Agency for funding the Iris (ManitobaSat-1) project and for providing me with expert opinions and exciting opportunities to grow.

A special thanks to Dr. Ally Ferguson for providing guidance in the application of Statistical Process Control techniques. It has provided me with the groundwork for this thesis. I also thank my committee members, Dr. Xihui Liang and Dr. Faouzi Bellili, for their precious time to review and provide feedback. Thank you to Matlab's online support staff and forum members who have patiently helped me debug my code and have given me solutions that I could not have thought of. They made my life easier.

I would like to gratefully acknowledge the support of my dearest friends who have cheered me up during difficult times and have celebrated with me even the smallest of my accomplishments. Thank you to my lab members who journeyed with me as I worked on this thesis. Their constant encouragement and feedback helped me improve.

I have been very lucky to have my parents stand by me throughout my life and particularly during this time, loving me unconditionally and giving me unceasing

support. I am forever indebted to them. Lastly, I would like to dedicate my thesis to my grandfather whom I lost recently and loved immensely. He never lost faith in me and always encouraged me to follow my dreams.

Contents

List of Tables

List of Figures

Nomenclature

Acronyms / Abbreviations

ADCS	Attitude Determination and Control System
ASM	Actual Satellite Model / Actual Spacecraft Model
C	Capacity
C&DH	Command and Data Handling
DET	Direct Energy Transfer
DOD	Depth of Discharge
EMF	Electromotive Force
EUVE	Extreme Ultraviolet Explorer
FDIR	Fault Detection, Identification and Recovery
GEO	Geostationary Equatorial Orbit
GMSEC MPI	Goddard Mission Services Evolution Center Message Passing Interface
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
HL	High Increase Long Duration
HS	High Increase Short Duration
I	Current
ICS	Industrial Control Systems

ISS	International Space Station
JPL	Jet Propulsion Laboratory
LCL	Lower Control Limit
LEO	Low Earth Orbit
LFP	Lithium Iron Phosphate
LL	Low Increase Long Duration
LS	Low Increase Short Duration
MDA	Macdonald, Dettwiler and Associates
ML	Moderate Increase Long Duration
MS	Moderate Increase Short Duration
NASA	National Aeronautics and Space Administration
PC	Personal Computer
PCA	Principal Component Analysis
PCU	Power Control Unit
RMS	Root Mean Square
SOC	State of Charge
SPC	Statistical Process Control
STK	Systems Tool Kit
TLE	Two Line Element
UCL	Upper Control Limit
V	Voltage
VGS	Virtual Ground Station
VGST	Virtual Ground Station Testbed
VSM	Virtual Satellite Model / Virtual Spacecraft Model

Greek Letters

Σ	Standard Deviation
Ω	Ohms

Miscellaneous

Satellite:	An object that has been sent to space to obtain scientific information or is used for communication. In this thesis, the words satellite and spacecraft are used synonymously
Two-Line Element Set:	The mean keplerian orbital element at a given point in time for a space object

Chapter 1

Introduction

The space industry has come a long way since the launch of "Sputnik-I" in 1957. Ground control facilities and operations have played a significant role in this. The space industry has experienced a shift recently from large geostationary spacecraft to groups (or "constellations") of smaller, low-cost spacecraft in lower orbits [1, 2]. Some recent constellations include Canada's RADARSat Constellation Mission (RCM) [3] currently in orbit, ESA's Swarm Constellation Mission launched in 2013 [4] and NASA's A-train Satellite Constellation for weather observation [5]. These constellations have several advantages over a unitary spacecraft in terms of science return [6]:

- Higher reliability for data return
- Lower cost for larger science data
- Repeated ground passes
- Allow simultaneous coordinated measurements promoting redundancy

1.1 Motivation

There is a growing market for constellations of satellites clearly driven by the advantages they provide as mentioned above. Naturally, this has led to an emerging need to augment the accessibility to the services offered by them. Operating a ground station for even one satellite is a costly and time-intensive undertaking. Traditional satellite operations (Figure 1.1) require that all commands be rehearsed and tested prior to execution, and each pass must be staffed by expert teams of highly-trained engineers, scientists and technicians to monitor and command the spacecraft. These onerous staffing requirements not only limit our ability to access valuable space assets but also increases mission costs and errors and is time consuming. Operators and experts inspect the telemetry data to determine the current satellite health using different statistical techniques and the analysis of such a large volume of data by humans is error prone. For large constellations, this amplifies the ground operations challenge with each spacecraft having its own idiosyncrasies that must be managed individually, thus, becoming virtually unmanageable. This is especially difficult when on limited budgets, as is the case for many small satellite missions. An example is the use of a Low Earth Orbit (LEO) constellation requiring approximately 40 to 80 spacecraft to provide global services such as remote sensing or communications data [7]. With each spacecraft making multiple ground station passes daily, staffing the station for these passes, managing the data and monitoring anomalies would be infeasible without a large and dedicated team. Moreover, constant human attention is required since satellite visibility from Earth can be limited which requires fast satellite to satellite handover especially for communication satellites. This puts a tremendous strain on the experts or engineers and operators, potentially leading to costly mistakes and largely influences the costs of mission operations. Also, ground operations currently represent about 20-30% of the total mission costs which is significant [8].

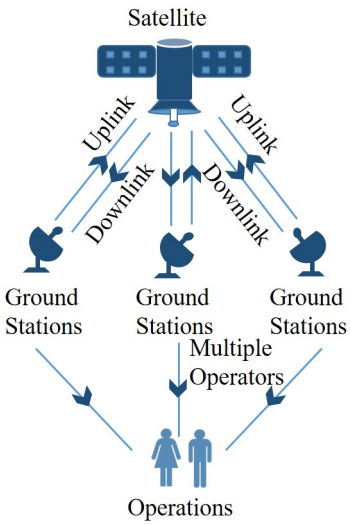

Figure 1.1: Traditional spacecraft communications [9]

The industrial partner for this research is Macdonald, Dettwiler and Associates (MDA), a world leader in spacecraft design and operations. With the recent shift towards constellations of small spacecraft from a single large spacecraft, MDA needs to adopt technologies that facilitate greater efficiency when interacting with space-based assets. Clearly, there is a need to automate most of the space/ground interface including fault detection.

1.2 Hypotheses

The challenges identified above can be overcome by developing an automated ground station capable of performing normal housekeeping activities, thereby, reducing the dependency on experts and mitigating the risk of human error. Such a system would streamline the ground station operations processes and reduce the financial and human resources burden. By reducing manual activity, an automated ground station,

hereafter called the "virtual ground station" or VGS, facilitates interaction between the satellite, operations and science teams. The virtual ground station is made up of two primary components. The first component comprises a mathematical model of the spacecraft systems, along with a communications "pass" manager that handles actual command and data transmissions with the spacecraft. It is called a mathematical model as it can be represented by a set of interdependent equations. The user interface terminal accesses data from the spacecraft model, providing the user with either real data from the past or predicted data for "live" data requests for which real data has not yet been linked to the ground. The second component of the virtual ground station is a Statistical Process Control (SPC) module that monitors the real telemetry streams from the spacecraft and compare them with the model and historical data to identify potentially anomalous operations. This research tests the following specific hypotheses:

- Virtual Satellite Model: A virtual ground station can be developed based on a mathematical model of the spacecraft that abstracts away spacecraft communications logistics, giving ground operators the sense of a spacecraft in continual communication at all times. The model is based on interactions between different components of a spacecraft and is simulated using the mathematical equations governing the concepts behind those interactions and hence, is described as a mathematical model. I hypothesize that the advantages to such a system are manifold as given below:

  - Reduces burden on highly-trained engineers to attend and analyse every ground station pass to ensure safe and efficient operations. Engineers can focus on more complex and intricate problems that need their attention
  - Manages every pass, uploads stored commands when available, downloads telemetry since the last pass and maintains the virtual spacecraft model

reducing the burden on human operators and experts

- Facilitates continuous interaction with the virtual spacecraft model which mimics the real spacecraft and is essentially a crude model of the spacecraft.

- Statistical Process Control: Traditional analysis techniques from the field of automated Statistical Process Control can provide near real-time health and diagnostic evaluations based on spacecraft telemetry trends and how they change over time.

  I hypothesize that SPC tools developed for the virtual ground station system are beneficial in many ways.

  - It will make it possible to monitor a constellation of satellites with ease. Traditionally, trend analysis of telemetry data has been time-consuming, repetitive and labour intensive. Extensive human involvement could be subject to error, leading to catastrophic failures if the operators fail to identify and detect faults in critical safety components [10]. Many failures are subtle and could be precursors to a fatal event. For example, the electronics on a reaction wheel may slowly begin to require more current due to a bearing starting to show premature wear (potentially due to a manufacturing error). Initially, this issue would present itself as merely slightly more current than normal, but still well within the acceptable bounds provided by the reaction wheel manufacturer. If left unchecked, this issue could lead to the reaction wheel drawing down the entire power bus as the motor draws more and more current. Even worse, the bearing could suffer a dramatic failure, resulting in the sudden cessation of the wheel and either a tumbling spacecraft and/or a damaged structure. If ground operators could be made aware of the degradation before it escalates into a failure

using SPC, they could switch to a redundant unit or modify operations to avoid the failure.

- With SPC, key telemetry items (such as bus voltage, current) are analysed through automated control charts to detect issues that require attention. Patterns, trends and out of control conditions aid in continually verifying if the "process" (i.e., spacecraft operations) is in equilibrium.
- It also acts as a visualization tool to facilitate the assessment of various subsystems' performance.
- Most importantly, the use of SPC will reduce the dependence of satellite monitoring on human resources to a large extent, which is especially important when considering the growing popularity of constellations of satellites. Since, SPC is executed in real-time, it helps identify anomalies in the system that might otherwise not be detected by operators before they lead to catastrophic failures. This is preferred to expert systems for the virtual ground station since it is easier to develop and simplistic.

1.3 New Contributions

The work presented in this thesis develops the software framework for supporting a virtual ground station that uses fundamental concepts from industrial SPC to automate spacecraft ground station operations and thereby minimize the burden on human operators, particularly for constellations of spacecraft. In this research, the power subsystem of the spacecraft is used as an example to demonstrate fault-detection for subtle anomalies including a sudden loss of a solar string, abrupt excessive power consumption and sudden increase in the battery's resistance. It also presents a mathematical model of the spacecraft known as the virtual satellite model that would give

operators a sense of continuous communication with the spacecraft. The virtual satellite model is a crude model of the spacecraft that mimics the actual spacecraft as much as possible. Simple and reliable SPC algorithms are developed for fault-detection and diagnoses based on control charts and rules used in the industry. These are tested extensively and can be applied to other spacecraft without many modifications. The fault-management system using the SPC algorithms in no manner depends on the output from the virtual spacecraft model as the fault-detection only occurs by analyzing the telemetry obtained from the actual spacecraft. In this sense, the virtual spacecraft model and the fault-management are independent of each other. Though the virtual spacecraft model is a rudimentary model of the real spacecraft, this research does not necessarily assure or aim to establish the accuracy of the model.

This thesis also presents a user interface terminal as part of the virtual ground station for data access and sending commands to the spacecraft. Using it, the operators can store commands in a queue at any time without waiting for a pass such that these commands are later sent to the spacecraft automatically. This is a valuable tool for evaluating the architecture of mission operations and is also easy to use for amateur operators.

A testbed to verify and demonstrate the functioning of the virtual ground station is presented. It provides a unique set of capabilities and flexibility and works as a reliable setup. The setup consists of serial communication between the simulated spacecraft and the virtual ground station which is a starting point for extending scalability to large spacecraft constellations. Multiple simulated spacecraft can be connected similar to the procedure presented for a single spacecraft for this testbed. Demonstration of complex mission operations can be done using the testbed in a cost effective manner.

This research has advanced management and control of spacecraft from the ground

by empowering spacecraft ground controllers and reducing operation costs significantly. This automated software framework is especially useful when dealing with a constellation of satellites. With the development and application of SPC algorithms to mission operations, highly-trained and qualified engineers would no longer be required to initiate communication with the spacecraft, perform mundane housekeeping activities and troubleshoot basic systems. The virtual ground station is the first system to use SPC combined with a detailed spacecraft mathematical model to streamline satellite operations. The insight from the research presented in this thesis will help aerospace companies such as MDA and amateur spacecraft operators to leverage the power of streamlined ground control. This framework is a beneficial tool for the Canadian space industry.

The conceptual model of the virtual ground station can be seen in Figure 1.2.

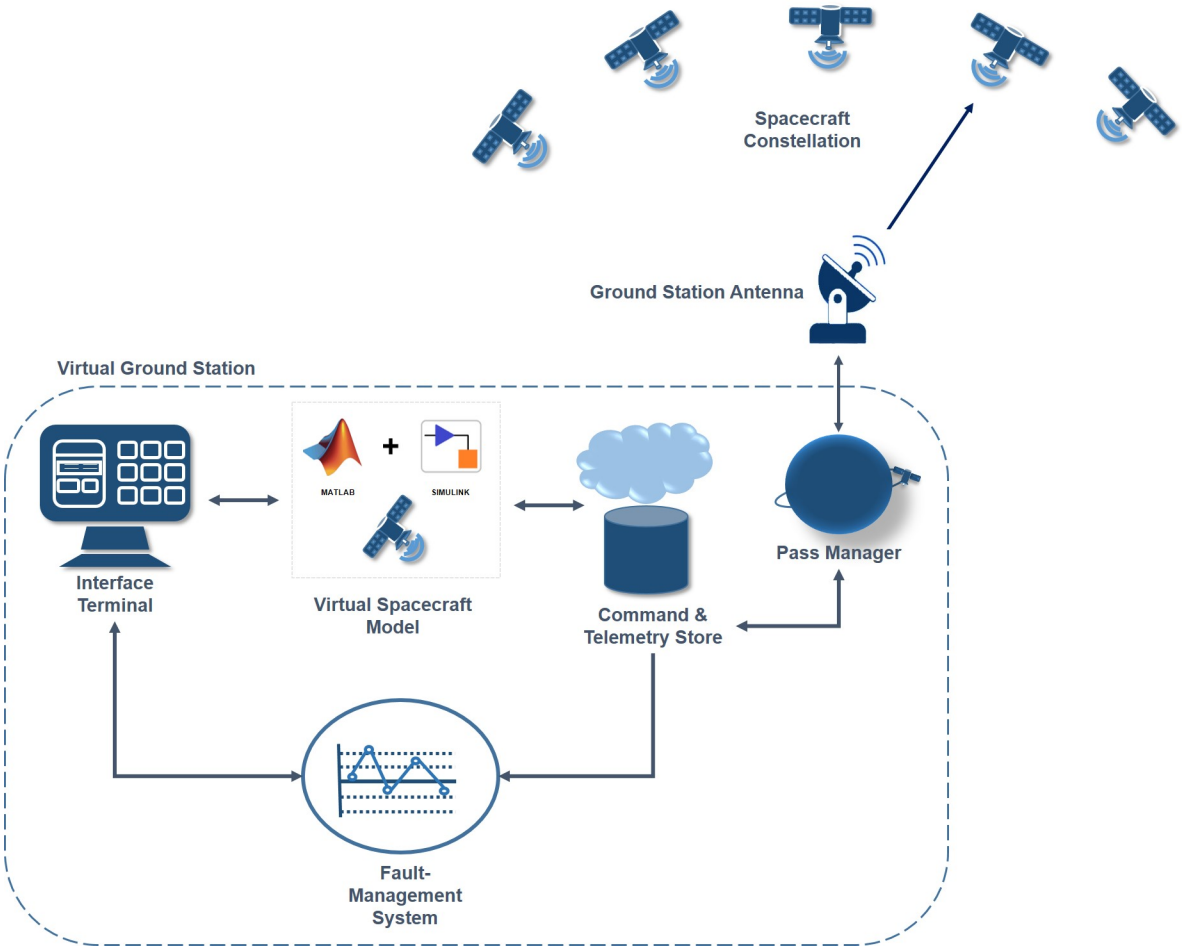

Figure 1.2: Conceptual model of the virtual ground station. Note that the virtual spacecraft model is a crude model of the spacecraft and the fault-management system runs independent of this model.

In summary, the salient contributions of my research are:

- A power subsystem simulator for a satellite to help with the designing of the electrical power system and sizing the electrical components such as the batteries. The power subsystem is taken as an example to demonstrate the concept of the virtual ground station such that the power subsystem simulator takes the form of the virtual spacecraft model for this thesis.
- A user-interface terminal built in Matlab App Designer to allow the operators to communicate with the virtual ground station and facilitate data access and command handling.
- A fault-management system based on statistical process control methodologies and time-domain feature extraction to detect major anomalies in the spacecraft's power subsystem. Fault-detection occurs on the telemetry obtained from the actual spacecraft during passes.
- A testbed with a simulated model of the actual satellite's power subsystem used for demonstrating the functionalities of the virtual ground station. The testbed uses serial communication between the virtual ground station and the actual satellite simulator.

1.4 Thesis Outline

This thesis is separated into seven chapters. Chapter 1 introduces the idea of a virtual ground station with fault-detection capabilities using statistical process control techniques and discusses the significance of such a system. This chapter also includes a description of the hypotheses and the components of the VGS. Chapter 2 provides an

insight into the concept of autonomous ground stations and the challenges in their development. It includes background research into existing autonomous ground stations, applications of SPC and research gaps to be addressed. Chapter 3 describes a power subsystem simulator for a low earth orbit CubeSat which is later applied to a real mission, ManitobaSat-1, in Chapter 4 in which the results from the simulator help design the power subsystem for the mission. Chapter 5 focuses on the development of the fault-management system of the VGS that utilizes SPC techniques and time-domain feature extraction. The methodology behind the development of the fault-detection algorithms is elaborated. Chapter 6 is devoted to the designing of the VGS's interface terminal, its features and the implementation of the fault-management system in it. The chapter also explains the use of the power subsystem simulator as the virtual spacecraft model. The latter part of the chapter presents a testbed used to verify the functionalities of the VGS such as sending commands and receiving telemetry from a simulated satellite model and details a simulated demonstration of the typical communications between the VGS and a satellite. Finally, Chapter 7 summarises the conclusions and the major contributions of this research and ends with the recommendations for future work. This thesis may contain some content from [11] and is used with permission from Springer Nature Journal of Aerospace Systems.

Chapter 2

Literature Review

This chapter provides a comprehensive literature review of all the fundamental concepts required to understand the design, methodology of the virtual ground station and the results of this research. There are four sections in this chapter. Section 2.1 discusses the existing autonomous ground station systems, their features and limitations. This is followed by a list of challenges faced during automation of ground station systems. Section 2.2 discusses existing power subsystem simulators for satellites and their importance. An overview of statistical process control (SPC) applications is provided in Section 2.3. This also covers some major existing systems using time-domain parameters for anomaly detection. Finally, the research gap in the published literature from the previous sections is identified in Section 2.4.

2.1 Autonomous Ground Stations

There has been considerable research done on autonomous and virtual ground stations for satellite communication in the past. One of the earliest autonomous ground stations were the Automatic Picture Transmission stations used for National Oceanic

and Atmospheric Administration satellites in the US. This was introduced when the satellite, TIROS-8 was launched in 1963. These simple stations could support autonomous data acquisition. Though useful, these did not have a lot of features and did not support continuous ground station operations. With the introduction of "systematic spaceborne data collection services" from the ground segment, multiple remote ground stations, each operationally autonomous could form a system together. Interrogation, Recording and Location System was such a system employed by Nimbus-3 launched in 1969 [12]. Standardizing the format for processing and distributing satellite data in 1987 has allowed information from different parts of a satellite to be put into packets to be sent to the ground stations or as commands to the satellites. The European Space Agency started using the Consultative Committee for Space Data Systems for its ERS-1 mission [13]. This provides very reliable uplink with opportunities for automatic retransmission. The above systems largely focus on ground station interfaces and remote operation of ground station hardware. This can be laborious due to the various types of radio and antenna systems available. This can be overcome by maximizing automation in the ground station itself, hence, reducing the required software for remote operation [13].

NASA Goddard Space Flight Center (GSFC) operates many spacecraft with varied objectives in space. NASA has made several attempts to implement automated spacecraft operations over the years. Anderson [14] conducted research on these systems. Several existing systems are analyzed and ranked according to a scoring system. The operations are analyzed for their automation potential using attributes such as their ability to be inspected, predicted or repaired within the existing ground systems architecture. A score based on these (and other) attributes helps identify which operations should be automated. One of the earliest systems reviewed is GENIE (Generic Inferential Executor) which reduced the required labour by 50% in the case of a typical NASA command centre where two people, namely the command controller and

spacecraft analyst, would be staffed per satellite per shift. The command controller decides which commands to send based on input from the analyst who monitors the spacecraft's health and is technically the 'expert'. While useful, GENIE has no statistical monitoring capability (for detecting subtle changes that could suggest an early stage fault) and is often too slow to be executed in real-time. Based on the research conducted on existing systems, Anderson [14] provides a framework for developing artificial intelligences to conduct autonomous satellite operations with a focus on CubeSats. It is built using a rule based system developed using the Java expert system shell. It provides error recovery and scalability but is still, largely dependent on human operators. He suggests that this can be overcome using a Deterministic finite automata which contain agent actions as states and satellite operations as its transitions.

GSFC follows an incremental approach for the smaller, low-risk and low-priority satellites. Newer automation functions are added as needed avoiding a one-time upgrade. This allows satellites to run until their life's end at minimal cost [15]. Similar to the standardizing efforts mentioned in [13], an automated message passing interface called Goddard Mission Services Evolution Center Message Passing Interface (GMSEC MPI) facilitates simple addition of system components. All modules have the same message wrapper. The components know what the messages look like, allowing them to read messages reliably. Though automation has been implemented in multiple applications at GSFC, no anomaly handling is done. When an anomaly is detected, all corrective action is taken by a human. That is when a system parameter goes red or indicates fault, a message is sent to the engineers and the system waits for help. This is purposefully done as no dependable anomaly handling system has been developed yet. Another limitation is the absence of a simulator that can be applied to many satellites with minimal changes. Each simulator at GSFC used for anomaly detection is custom designed uniquely. "Tailored automated scripting" is required for

each satellite [15].

NASA's Extreme Ultraviolet Explorer (EUVE) satellite is part of the Deep Space Network that is designed to operate with the Tracking and Data Relay satellite System. The EUVE Science Operations Center uses an autonomous telemetry monitoring software that has helped reduce console work from three shifts to one. This system is similar to the one used by NASA for Low Earth Orbit (LEO) weather satellites. Orbital elements are obtained from official tracking databases and passes are scheduled and tracked autonomously. It is cost-effective since multiple ground stations share the responsibilities [16].

The European Ground System - Common Core Initiative is proposed to have an architecture similar to GMSEC MPI [15]. Unlike GSFC, the Navel Research Laboratory's Neptune architecture implements scripts to handle basic anomalies. Examples of corrective actions include memory flush and rebooting. They have modes where automation can be chosen over manual operations and vice-versa. This promotes flexibility. Also, Neptune is capable of automating contact with each satellite and connecting with the most suitable hardware tools in real-time to do that on a case by case basis using software-defined switches. This is one of the most advanced automated ground station systems in the world [15]. Nevertheless, this system requires advanced resources to build and is expensive.

Holdaway [8] developed a program for autonomous ground station control of small satellites. He theorized that mission costs could be reduced by making very small compromises in data return and standardizing subsystem procedures for housekeeping. Examples of these compromises include occasionally missed passes and slower return of non-urgent data. He concluded that the personnel cost the most out of all operational elements. Around 5-50 people per shift would be required to monitor the satellites around the clock. The proposed system includes features such as automatic

and eventual autonomous checking of critical data during real-time passes to identify abnormal parameters, automatic reception and storage of downlink telemetry, open-loop tracking of high-altitude satellites, prediction of tracking data for a week, and automatic dialing to expert engineers when in need. There is also a closed-loop feedback of error-signals into the orbit determination to autonomously update the predicted orbit parameters for the future. An advanced version of this system is in use at the Rutherford Appleton Laboratory and Chilbolton Observatory. Nevertheless, Holdaway's system does not provide "continuous" communication with the satellite as is the case with the virtual ground station that does not require the communication to be restricted to the periods when passes occur.

Fault detection in the components of a ground station is a significant aspect of ground station automation. A system called Fault Detection, Identification and Recovery (FDIR) system is described in [13]. The ground station's control server sets the proper configuration for all the components using the FDIR system. When an anomaly occurs in any component's behaviour, the FDIR system performs a fault diagnosis using a list of possible reasons and solves the issues with predefined recovery algorithms. This functions as a tree-like checking sequence where the current state is compared to the desired configuration and known fault cases. When a fault occurs in the form of an undesired state in a software programme, software starts the programme for that interface to recover. In cases of complicated failures, human operators and experts are informed. Freimann et al. [13] suggests that the autonomy of this ground station system is obtained by looking at the ground station as a "gateway" and the whole system as a "black box" on route to the target satellite. More recently, Planet Labs, a private Earth imaging company uses many open source packages for fault detection. These packages were originally designed to monitor websites and servers but are now applied to ground stations as well. This open source network monitoring system runs on the ground station servers and continually monitors the

satellite's health. The cloud-based central server is notified by the local machine when a fault occurs. An alert is issued to an engineer when needed. In the case of minor faults, hardware is adjusted to reboot or recover in events of power failures [17].

Bernier and Barbeau [18] describe a virtual ground station system that effectively improves the accessibility of satcom services when connected to the internet. This allows launching of tracking sessions from different cities. To maximize platform adaptability, it is designed using platforms such as Java and Cobra. It is equipped with virtual equipment such as a transceiver, an antenna and a rotor. It provides features such as antenna steering, transceiver configuration, satellite position computing, satellite tracking and pass prediction. An aesthetic end user application accompanies the system. This system has been successfully tested for satellites from the Iridium constellation.

Bentley et al. [15] opine that non-space domains provide ideal tools for overcoming the challenges of a ground station architecture. They studied the similarities between industrial control systems (ICS) and satellite ground stations. These include flexibility, cost-efficiency, failure intolerance and lifetime efficiency. Based on this, they suggest an architecture using the Model View Controller widely used in ICS. The controller and model are connected in a closed loop making the system to be simulated and tested with operational code. This makes integrated simulation possible.

2.1.1 Challenges in the development

Automation is clearly essential to improving operating efficiencies for satellite operations. There are several difficulties that were encountered while designing the virtual ground station. This sub-section summarizes the main challenges faced by researchers automating ground station operations [15]:

- Most existing ground stations have not been built with automation in considera-

tion. Hence, adding these new software capabilities is costly and time intensive. Adding new design features and requirements after development is difficult than implementing them during the planning phase.

- As mentioned in Chapter 1, constellations of LEO satellites are getting popular. Each satellite is different and this diversity creates more challenges in terms of developing common software packages, simulators and code reuse.

- There is no universal documentation defining what automation means for a ground station. Some systems classified as automatic only handle scripted satellite passes while others are capable of performing most routine operations.

2.2 Power Subsystem Simulators

The virtual ground station consists of the virtual satellite model that mimics "continuous" communication between the operator and actual satellite. The virtual satellite model is essentially a satellite simulator with the focus being the power subsystem simulator in this thesis as mentioned in Chapter 1. The idea of the power subsystem simulator can be extended to the whole satellite as required. The power subsystem simulator can be used to assess system operations, behaviour of the power source and subsystems, and analyse the power consumption and generation cycles throughout the mission. In this thesis, the goal of the developed simulator is to function as closely as possible to the actual spacecraft.

With the demand for advanced efficiency and reliability, smaller size and lighter weight, the ability to design and test sophisticated spacecraft power subsystem simulators becomes relevant. Research for such simulators has been going on for decades and a major issue faced in developing these simulators is the complex interconnectivity between the components and the numerous interactions leading to unpredictability

of the whole system despite individual component behaviour being understood well. Most simulators are either based on generalized system analysis models or custom-made models for specific systems. Some of the earlier systems such as SPICE and SCEPTRE faced problems with dynamic memory management and inflexibility for future modifications [19]. Cho and Lee [19] hypothesize that this infeasibility is due to defining the complete power subsystem completely in terms of circuit elements. They suggest having local subsystems connected using global interconnection laws. Based on these observations, the EASY5 software developed by Boeing Computer Service is used to develop a model for small and large scale power subsystem models. Lee et al. [20] describe an advanced version of this system for application to more complex power systems. Another software based on modularity is the EBLOS which uses a nodal description input. Capel et al. [21] validated this software against the ERS-1 mission. Similarly, many other sophisticated power subsystem simulators especially in the case of CubeSats employ nodal analyses where the circuit is analysed in a complex network using the nodes as inter-connective reference points. Voltage and current values at each node are calculated and the entire circuit is solved using simulatenous equations [22]. A more recent simulator based on nodal analysis developed for the TUSUR University project is given in [23]. In this simulator, modelling time is reduced by using the method of moments and the modified method of nodal potentials.

Melone [24] developed an electrical power subsystem simulator for preliminary design and test of the TINYSCOPE nanosatellite, a three-axis stabilized, low earth orbiting, electro-optical imager. It had high power requirements which posed challenges in terms of power collection, energy storage and power management and distribution. This was overcome with an analytical-numeric approach.

Bauer [22] describes a method to perform an electrical analysis and a transient thermal analysis of a satellite electric power subsystem. The program they developed

runs the power subsystem through one or more complete orbits and plots curves to investigate voltages, currents and temperature changes. But, this program does not consider load variation for various subsystems during the mission. Kim et al. [25] performed statistical analysis of the satellite electrical power subsystem's on-orbit failures and anomalies with a focus on comparison between LEO and Geostationary Equatorial Orbit (GEO) satellites. Partial failures are classified into different classes depending on their severity. Non-parametric estimation is used to study the failure and degradation behaviour.

A power subsystem simulator designed for Mysat-1 at Khalifa University has a robust algorithm that can be used to test several control algorithms over multiple orbits. The dynamic interaction between the components is implemented using a coordinate control with voltage regulation and battery charging and discharging systems [26]. The popularity of the Matlab/Simulink environment for simulators has increased drastically. One such simulator developed for a nanosatellite is described in [27]. However, the system seems complicated in implementation but the results show little variation in the power consumption throughout the simulated time indicating that the mission operations are assumed to be too simplistic.

Traditional programming languages such as C, C++ and FORTRAN typically result in large code making it difficult to understand, modify and troubleshoot. This makes Matlab/Simulink a better alternative due to increased flexibility and easier handling. More established software such as EASY5 mentioned previously are sometimes used alongside Simulink where the controller is built in Simulink while the plant model in EASY5 [28]. Saraf et al. [29] performed a comparison of different simulation tools on the basis of various criteria such as real-time capabilities, design flexibility and user-friendliness. It was concluded that Simulink was better than EASY5 for low-budget spacecraft simulators since the former has many popular toolboxes and wide range of aerospace add-ons. Though a good software, EASY5 is not used widely

and has a limited availability of toolboxes.

2.3 Statistical Process Control

SPC has been used in industrial applications since the 1920s when chart patterns were analysed and interpreted manually [30]. In the 1980s, with technological advancement and the changing needs of the manufacturing industry, manual methods for creating and analysing control charts were no longer sufficient. Quality control practitioners were required to have considerable skill and experience to perform control chart pattern recognition. This led to the idea of expert systems and other computing technologies. This was explored by researchers such as Swift (1987) [31] and Cheng (1989) [32]. Cheng focused his research on the application of SPC in small-batch manufacturing using expert systems concepts. In the 1990s, with the development of artificial neural networks for SPC chart pattern recognition, researchers overcame drawbacks in the previous expert system approaches by improving the handling of non-linear data, fault tolerance and adaptability. Zorriassatine and Tannock [33] provide a review of neural networks for SPC developed during the 20th century. Feature extraction using mathematical models coupled with projection techniques is used to apply SPC to make a better distinction between damaged and undamaged samples. Here, damaged samples refer to the samples with defects or faults and the undamaged samples are the samples that meet the product or design requirements reasonably. Sohn et al. [34] performed vibration-based damage diagnosis using SPC. This is done using data compression for feature extraction through control chart analysis which is very suitable for automated continuous system monitoring. This is applied to specific features to identify damage in structures. The feature extraction is done by developing an auto-regressive model with an undamaged sample. The control limits of the $\bar{X}$ control chart are selected from the coefficients of this model. Several projection

models such as "principal component analysis (PCA) and linear and quadratic discriminant operators with SPC" are used to identify gradual progression of damage in the structure. Methods such as PCA can be applied in conjunction with multi-variate statistics for anomaly detection. Bingqing et al. [35] constructed a PCA model to study the correlations between variables in an undamaged data sample. They suggest that PCA reduces the dimensionality of the data space which makes the process of anomaly detection simpler as that eliminates overlapping information. Based on this, an algorithm for anomaly detection for an attitude control system of a spacecraft is devised. For detection, multivariate statistics and system states are studied. Nevertheless, researchers such as Bouzenad and Ramdani [36] opine that conventional PCA often assumes a Gaussian distribution which is not always practical in industrial applications. They suggest combining it with non-parametric control charts to overcome this. Numerous papers have been published with learning methods and feature extraction for spacecraft [37, 38, 39, 40]. Though PCA is useful, it works best with high-dimensionality data and is an unsupervised learning technique similar to clustering. Univariate SPC is used more than multi-variate SPC with projection models such as PCA when the process in question is relatively stable. Also, the identity of the original variables is lost in PCA which means that it is difficult to identify the variable in which a particular detected fault is in. For processes such as the spacecraft health-monitoring which are expected to be stable most of the time, the operators would like to know exactly where the fault is. Multi-variate SPC with PCA algorithms has a greater complexity in techniques due to the inter-dependence between different variables and often requires a complete understanding of the univariate SPC analysis. So, it usually takes more time to understand, develop and write code for as compared to simplistic univariate SPC algorithms. More information on PCA can be seen in [41]. Sometimes, PCA's principal components are biased towards features with high variance, leading to false results when data is not standardized. More de-

tails can be found in [42]. Simplistic algorithms for performing SPC are required which would not require the operators to improve the process, instead methods that would determine whether the spacecraft or process is in control are needed.

Though typical machine learning techniques are more common for anomaly detection, unsupervised and supervised machine learning algorithms usually involve a complicated model that understands the process or the system it is trying to study to provide insight into the roots of the process. When a parameter in a process is relatively in control most of the time and does not need to be gauged all the time, SPC is cost-effective and accurate and can be implemented using a simple rule-based model [43, 44, 45]. Such is the case for the trivial housekeeping activities of a spacecraft.

SPC techniques are frequently used for maintenance of huge industrial units such as aircraft systems. Beabout [46] developed a SPC visualization tool for the maintenance management of aircraft at an Air Force base in the United States. He used control charts for displaying mission capable rates, flying scheduling effectiveness rates and the aircraft subsystems influencing these parameters. A detailed guide on how to develop the algorithms for different SPC applications with control charts and rules is given in [47]. The Jet Propulsion Laboratory (JPL) uses SPC to monitor NASA's Deep Space Mission Network. The data received on providing telecommunications services and ground based science observations is retrieved for analysis and interpretation by the SPC software. SPC provides a cost-effective and important tool to evaluate data system tracking, functional availability and system configurations. Recent applications include exploration of the mean time between failure for different antennas. Shewhart control charts are the major tools used at JPL for this [48].

In manufacturing applications for quality control, a need for real-time SPC capabilities is often observed. Lee et al. [49] provide a software with algorithms that conduct analysis of real-time sensor data from semiconductor manufacturing equip-

ment. Multi-variate analysis techniques coupled with SPC are used to generate real-time alarms indicating malfunctions. The algorithm allows automatic generation of time-series models for the baseline process and any deviation is detected using SPC control charts.

Though SPC has been part of the space industry in the form of manufacturing quality control and network monitoring, it has not been used in real-time satellite operations and maintenance. This explains the scarcity of published works in the domain of SPC applications to real-time anomaly detection in satellite ground control systems.

2.3.1 Time-domain Features

Data-driven systems require feature extraction to transform raw signals into informative signatures. Data descriptive statistics using time-domain features are one of methods for this. Time-domain features that represent the characteristics of the system are used to devise algorithms that perform classifications to effectively divide the data and detect anomalies. These features are provided as input to the system rather than using the raw data [50]. This is demonstrated by Park et al. [50] in developing a long short-term memory fault detection model for an industrial robot manipulator.

There are many algorithms for characterizing time-series features to help extract useful features from a time series, 9000 of which are analyzed in [51]. Fulcher [52] mentions that quantify patterns in time series across the measurement time uses global features that can simplify complicated temporal patterns from different time intervals into low-dimensional understandable processes. Global features allow us to apply them to various time series of different lengths easily and developing algorithms for analysis much simpler. Some of the major global features include variance, mean, kurtosis, crest factor and autocorrelation. These reasons make global features

lucrative to use for anomaly detection in this thesis.

Nanopoulos et al. [53] uses several statistical first and second order features after evaluating their benefits for classification. These features such as mean, standard deviation, skewness and kurtosis are used to identify synthetic control chart patterns in SPC. The authors theorize that the selected features are simple to implement. While efficiency is certainly higher with more complicated features, automated feature selection methods are much easier to conceive for the selected features. It was also observed that the accuracy of the method decreased with increasing series length. Wang et al. [54] proposes a general framework to perform time-series clustering with high accuracy using a small set of global features such as measures of trend, seasonality, chaos and kurtosis. This approach introduces no additional parameters during the feature extraction process. This method has since been applied to multivariate time-series as well [52]. Rahimi and Saadat [55] used a time-domain feature space to express the difference between the nominal and faulty output from reaction wheels onboard a three-axis controlled in-orbit satellite. This helped extract unique properties between different fault scenarios. These are then utilized in a machine learning algorithm. Similar applications of time-domain features are also given in [56, 57, 58].

It is clear from the existing literature that the choice and application of features to different time-series is subjective and has no prescribed system. This makes it hard to compare the accuracy of the used feature set to another researcher's feature set since different researchers use varying time-series from interdisciplinary literature [51, 59]. One solution is to have a threshold on the standard deviation of the time-series [51]. Wang and Nanda [60] suggest that the high relevance to the objective such as anomaly detection, degradation is essential when selecting features. Also, linear independence leading to low redundancy is desirable among the features.

2.4 Summary of Literature Review and Research Gaps

A brief review of the existing autonomous ground station systems, anomaly detection schemes, SPC methodologies, time-domain feature applications and testing schemes for an engineering system is given in this chapter. It is observed that autonomy of ground stations can be improved by having integrated simulations and layered architectures. This is essential to save time and cost as the same simulator can be used for multiple satellite missions and even constellations by making relatively minor modifications. Layered architectures aid with making changes to the design easily at different stages and times. Though multi-variate statistical analysis, PCA, neural and Bayesian networks have been widely studied for spacecraft components' anomaly detection as discussed, traditional SPC methodologies with algorithms for automated real-time fault detection in spacecraft have not been focused on. SPC algorithms are less complicated and easier to implement than many other techniques which becomes important especially for small-budget student-led CubeSat missions. Many of the models and systems developed in literature make use of complex software that might not be accessible or understood by everyone. There is a need to use a simplistic platform that many have available. The above inferences are kept in mind while designing the virtual ground station.

Chapter 3

Designing a Power Subsystem Simulator

3.1 Introduction

Chapters 1 and 2 discussed the idea of the virtual spacecraft model which is a mathematical model of the spacecraft that would give the ground operators the sense of a spacecraft in continual communication at all times.

A simulated spacecraft model or the virtual spacecraft model continually predicts the system's output, based on the commands uploaded by the operator. When the actual spacecraft passes over the ground station, the pass manager establishes communications, uploads commands that have been issued since the last ground station pass and downloads the latest spacecraft telemetry. Later, these commands are executed by the spacecraft at the specified date and time. Between passes, the ground terminal provides data from the virtual satellite model immediately upon request. In this sense, users can obtain expected spacecraft responses to commands without having to time their requests with physical ground station passes. Once the virtual

ground station receives real data, the historical record of spacecraft operations are automatically updated with the real data. The virtual satellite model provides the predicted values of various parameters that provide insight into the health of the spacecraft. The model provides a baseline of what the telemetry received from the satellite in orbit should look like. The main parameters considered for this research are:

- Battery current (charge and discharge)
- Total solar panel current
- Power consumption
- Battery state of charge
- Bus voltage

In this thesis, the focus is on the power subsystem of the satellite but the idea can be abstracted to the whole satellite to form the virtual satellite model. This chapter presents a novel power subsystem simulator of a satellite, more specifically a low earth orbit CubeSat. The simulator contains a basic battery model, a solar array module, a representative model of direct energy transfer battery charge control and the power control unit (PCU). This is done in the Matlab/Simulink environment. The simulator helps with preliminary sizing and configuration of selected electrical components in an unregulated bus voltage scenario. The solar arrays, batteries and

PCU are modelled using a current-voltage lookup table, a voltage-capacity lookup table and repeating sequences respectively. Later on, the simulator takes the form of the virtual ground station's satellite model. This chapter describes the structure and functionality of the power subsystem simulator. This chapter may contain some content from [11] and is used with permission from the Springer Nature Journal of Aerospace Systems.

3.2 Simulator Model Structure

There are several components in the simulator design, namely, the power consumption timeline, battery charge controller, solar array module, battery module and the eclipse flag module. Each of these is designed and connected in Simulink to develop a power subsystem simulator (Figure 3.1).

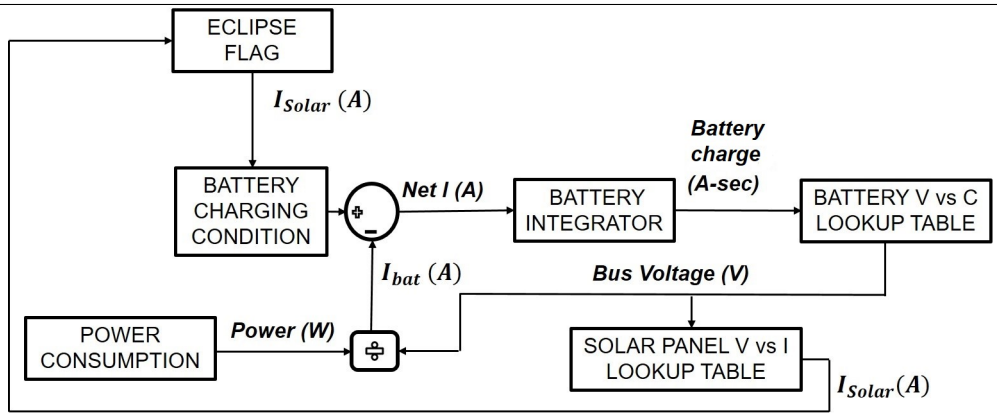

Figure 3.1: The functional model of the power subsystem simulator. Used with permission from the Springer Nature Journal of Aerospace Systems.

Power consumption input The total power consumption with corresponding orbit time throughout the lifetime of the mission or simulation time period are the

parameters required to create a power consumption timeline. Often, the power consumption timeline for missions repeats after certain number of orbits (P). The power consumption timeline for the whole simulation period can be represented as a repeating table. Simulink offers a repeating table component in which I can provide signal inputs. In this component, I enter two signals, the power consumption with the corresponding time for P orbits. These signals are generated as a repeating sequence by the table.

Battery module From the conceptual model in Figure 3.1, an integrator models the charge state of the battery. The corresponding bus voltage is deduced from this charge state by using a Simulink lookup table which uses values from the voltage vs capacity curve for the battery. The points from the lookup table are adjusted for the battery cell configuration that I want to test. The equations for these adjustments are provided in Equations 3.1 and 3.2 where C and V are the capacity and the voltage respectively. The capacity is the product of the capacity per cell and the number of strings where a string is an arrangement of cells in series ($M_{parallel}$). The voltage is the product of the voltage per cell and the number of cells in a string (N_{series}). The interpolated table lookup component in Simulink is used to model the battery module. The voltage values with the corresponding charge states are entered into the lookup table.

$$C_{eq} = C_{cell} * M_{parallel} \tag{3.1}$$

Here, C_{eq} is the total capacity of the battery, C_{cell} is the capacity of one battery cell and $M_{parallel}$ is the number of strings in the battery configuration.

$$V_{eq} = V_{cell} * N_{series} \tag{3.2}$$

Here, V_{eq} is the equivalent voltage of the battery, V_{cell} is the voltage of one battery cell and N_{series} is the number of battery cells in each string of the battery configuration.

Solar array module The solar array module is developed in a similar fashion to the battery module by using a Simulink lookup table which uses values from the I-V curve from the solar cells' datasheet. Generally, datasheets provide the I-V curves in terms of current density (D) rather than current (I). In such cases, the cell surface area (A) is multiplied by the current density to get the current values (Equation 3.3). Using the bus voltage, the solar array lookup table gives the solar array current. I get the I-V curve values for the proposed solar array configuration by multiplying the voltage by the number of cells in a string (M_{solar}) (Equation 3.4). The current is multiplied by the number of strings (N_{solar}) (Equation 3.3).

$$I_{eq} = D_{cell} * A * N_{solar} \tag{3.3}$$

Here, I_{eq} is the total current of the solar array, D_{cell} is the current density of one cell where current density of a cell is the product of its current area and N_{solar} is the number of strings in the solar array configuration.

$$V_{eq} = V_{cell} * M_{solar} \tag{3.4}$$

Here, V_{eq} is the total voltage of the solar array, V_{cell} is the voltage of one solar cell and M_{solar} is the number of cells in each string of the solar array configuration.

Eclipse flag With the passive solar array current, the condition of eclipse is introduced. A signal is created which is zero when an eclipse occurs and one during non-eclipse periods. This signal called eclipse flag (F) usually repeats every orbit and

is entered into a repeating table. So, the ideal solar array current (I_{ideal}) is 0 when there is an eclipse and is equal to I_{eq} when there is no eclipse.

Solar Insolation Angle The pointing accuracy of the Attitude Determination and Control System (ADCS) of a satellite influences the effective solar array current generated. Equation 3.5 defines this relation. The effective solar array current (I_{eff}) is the cosine of the pointing error multiplied by the ideal solar array current (I_{ideal}) from the solar array module. I use the worst-case pointing error (d) here to visualize the worst-case scenarios for verification. F is the eclipse flag.

$$I_{eff} = F * I_{eq} * \cos(d) \tag{3.5}$$

Battery charging controller This component controls the charging of the battery by observing its voltage (V). The battery continues to be charged until its cut-off voltage (C) is reached. After some time, the voltage drops again and charging restarts. This cycle continues throughout the mission or simulation period. Battery charging is only possible during the non-eclipse period as solar array current is available. The battery charging condition component is implemented as a switch in Simulink. The switch logic is depicted in Figure 3.2. The design is based on the concept of Direct Energy Transfer (DET) systems which allow the bus voltage to fluctuate with the state of charge of the battery as the bus voltage is maintained equal to the voltage across the battery [61]. DET systems have been successfully implemented for several University satellite missions [62]. In the DET solar array interface, the solar arrays are directly connected to the battery and the voltage through the battery and solar arrays is maintained at the same level [62]. This is in contrast to a maximum peak power tracking (MPPT) design, where the power control unit actively adjusts the solar array voltage to maintain peak power generation regardless of a fluctuating bus

or battery voltage [63].

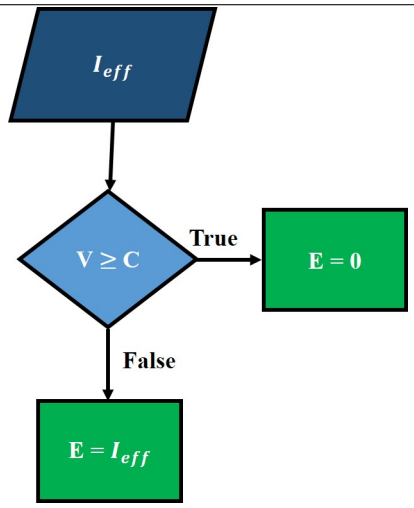

Figure 3.2: Logic for the battery charging condition switch. I_{eff} is the effective solar array current or the net current generated by the solar array, E is the current going into the battery, V stands for the battery voltage and C is the cut-off voltage of the battery. From the figure, zero current goes into the battery if the battery voltage is greater than or equal to the cut-off voltage. Otherwise, current equal to I_{eff} goes into the battery.

Internal resistance of the battery The battery's internal resistance calculated from the battery's datasheet and configuration is modelled as a modifiable variable in Simulink.

$$V_{EMF} = V_{Bus} - I_{Battery}R \tag{3.6}$$

In Equation 3.6, R is the internal resistance of the battery. This equation is used to calculate the electromotive force (V_{EMF}) [64]. V_{EMF} is the battery's internal driving force used to provide energy to a load, but differs from the bus voltage due to the

internal resistance of the battery [65]. Here, the current going into the battery is positive and the current going out of the battery is negative. It is clear from the above equation that while charging the electromotive force is lesser than the bus voltage whereas while discharging the electromotive force is greater than the bus voltage.

Battery charge integrator To regulate the battery's charge state, the battery current is sent into a Simulink integrator which performs continuous-time integration of the current. The initial charge, upper and lower limits for the charge are user-defined and can be changed easily in the integrator block.

All inputs in the simulator are given in SI units. This simulator can be applied to any satellite that uses DET with the solar arrays and batteries by making changes to the design parameters according to the mission being simulated. Changing the configuration and type of battery and solar cells in their blocks and the power consumption timeline based on mission operations is simple. Having basic information about any mission makes it possible for us to simulate its power subsystem with ease and minimal expertise using this simulator.

3.2.1 Representation As a System of Equations

The simulator's design can be represented by a set of equations consisting of a first-order differential equation, a few time-dependent equations and initial conditions. Hence, the design can be seen as a mathematical model. The equations are given below:

$$
\begin{aligned}
I_{Battery}(t) = \frac{dC(t)}{dt} &= (I_{eff}(t) - I_L(t)) \\
C(0) &= C_0 \\
C_L \leq C(t) &\leq C_U \\
I_{eff} &= \begin{cases} F(t) * I_{eq} * \cos(d) & C(t) < C_U \\ 0 & \text{Otherwise} \end{cases} \\
I_L(t) &= \frac{P(t)}{V_{Bus}(t)}
\end{aligned}
\tag{3.7}
$$

Here, $I_{Battery}(t)$ is the current through the battery as a function of time t

C(t) is the charge state of the battery as a function of time t

C_0 is the user-defined initial charge state of the battery

C_L and C_U are the lower and upper limits of the charge state of the battery respectively

$I_{eff}(t)$ is the effective solar array current and $I_L(t)$) is the current drawn by the loads using power

F(t) is the eclipse flag as a function of time and is an input to the system and d is the solar insolation angle

I_{eq} is the total solar array current without considering eclipse or the isolation angle

P(t) is the power consumption as a function of time given as an input to the system based on the mission operations

$V_{Bus}(t)$ is the bus voltage which is implemented using Ohm's Law (Equation 3.6)

3.2.2 Limitations

The simulator presents a realistic model of a CubeSat's power subsystem. Nevertheless, certain assumptions were made in the design. These assumptions result in the design having a few limitations:

- Thermal analysis is not considered. Solar arrays loose efficiency over time when they get hot. This behaviour is not part of the simulator design.
- More sophisticated models for the internal resistance are required to exactly show how it changes over time in reality and results in a decrease in the battery's efficiency.

It should be noted that these limitations are not a challenge for the simulator when used for short and small satellite missions as the effects are minimal.

3.3 Summary

In this chapter, I presented an architectural template for a power subsystem simulator of a low earth orbit CubeSat. It is implemented in Simulink which is a popular software for control engineering and simulations. The modular design of the simulator consists of a battery module, solar array module, battery charging controller, integrator and power consumption/loads module. I also consider the effects of eclipse and the ADCS pointing error on the effective solar array power generation. The simulator provides important parameters such as output voltages, state of charge of the battery and solar array current.

The simulator's design makes it a simple and easily modifiable interface to perform complex analysis. It forms an important tool in research and satellite design. It can be applied with basic electrical knowledge and can be modified through future research.

This simulator is a part of the virtual ground station's spacecraft model which is the basis of my first hypothesis which states that the virtual ground station can be based on a mathematical model of the spacecraft that the operators can continuously be in touch with. This will be elaborated on in later chapters. The next chapter uses

this simulator as a tool for designing the power subsystem for ManitobaSat-1, which forms the basis of the fault detection research that follows.

Chapter 4

Design of the ManitobaSat-1 Power Subsystem

4.1 Introduction

The previous chapter discussed a CubeSat's power subsystem simulator design developed on Simulink and its applications. Details were provided on how the simulator can be used and how the components interact with each other to give measures of important parameters. The required inputs and expected outputs of the different modules were mentioned. The simulator is adapted to simulate the power subsystem of the ManitobaSat-1, a student CubeSat built as part of the Canadian CubeSat Project funded by the Canadian Space Agency.

In this chapter, I provide the background on the ManitobaSat-1 mission which forms the groundwork required to perform the design and sizing of its power subsystem. ManitobaSat-1, being a CubeSat with an average lifetime of 2 years developed by students and its power subsystem based on the DET concept, is a good candidate for this simulator. With the power budget in place, a static power analysis is per-

formed resulting in the solar cell sizing. Dynamic simulations using the simulator are undertaken to finalize the battery cell sizing. The results of these simulations with the static analysis help substantiate the suitability and efficacy of the design. The simulation results and trends are also discussed.

4.2 ManitobaSat-1 Power Subsystem

ManitobaSat-1 (Figure 4.1) is a 3U CubeSat weighing approximately 3 kg with deployable solar panels and aims to study space weathering effects on geological samples by exposing them to direct solar radiation [66]. The CubeSat consists of an attitude control and determination system (ADCS), communication subsystem including radio RX (uplink receiver) and radio TX (downlink transmitter), command and data handling subsystem (C&DH), power subsystem and the payload science experiment. The payload camera takes images of the samples at regular intervals. The only source of power on the CubeSat are the solar arrays. For energy storage, Lithium Iron Phosphate battery cells (LFP-18650HT) are connected to the bus providing an unregulated voltage to all subsystems [67]. The concept of operations helps prepare a power budget with power requirements of different subsystems. From the concept of operations, it is observed that the same CubeSat operations take place after every three orbits. Thus, the power consumption timeline repeats every three orbits. A static power analysis is done and results in a design with two arrays, one on the front and one on the back, of Spectrolab XTJ Prime solar cells [68]. The front array has five strings of three cells and the back array (for redundancy and detumbling) consists of two strings with three cells each. Only the front array is sun-facing during regular operations.

Figure 4.1: An image of ManitobaSat-1

4.2.1 Static Power Analysis

The estimated three-orbit average power (different from the peak power) budget for ManitobaSat-1 is given in Table 4.1. I provide the three-orbit budget rather than one orbit since the budget is constant for a three-orbit period. The budget considers the worst-case three-orbit average power consumption values by each subsystem. With static power budgeting, it is found that the three-orbit average solar array power generation is 8.960 W (Table 4.2) and the three-orbit average power consumption is 8.025 W (Table 4.1). As the generation is greater than the consumption, the solar array sizing is suitable.

Component	Three-orbit average power consumption (W)
Thermal	1.000
C&DH	1.730
Radio RX	0.202
Radio TX	0.547
Power control unit and Battery controller	1.52
Payload camera	0.010
Payload controller	0.015
ADCS	3.000
Total	8.025

Table 4.1: Power budget for ManitobaSat-1

Parameter	Value	Explanation
Number of sun-facing arrays N_{solar}	5.00	Design
Cell width W (cm)	6.91	Solar cell datasheet
Cell height H (cm)	3.97	Solar cell datasheet
Cell area A (cm^2)	27.43	W * H
Power/cell P_{cell} (W)	1.17	The power generated by a cell assuming worst-case bus voltage
Cells per string M_{solar}	3.00	Design
Power per array P_{array} (W)	3.35	P_{cell} * M_{solar}
Total power for all arrays P_{eq} (W)	16.75	P_{array} * N_{solar}
Worst-case pointing error d (°)	24.00	ADCS design
Apparent width at pointing error $W_{apparent}$ (cm)	6.31	W * cos(d)
Apparent height at pointing error $H_{apparent}$ (cm)	3.63	H * cos(d)
Power at pointing error P_{eff} (W)	14.65	(P_{array}/A) * $W_{apparent}$ * $H_{apparent}$ * N_{solar}
Orbit period R (s)	5560.00	ISS orbit time period ([69])
% of orbit in the sun (SP)	61.16	STK simulations (Figure 4.2)
Sun-facing time in one orbit (s)	3400.50	R * SP
Worst-case orbit average power generation P_{orbit} (W)	8.96	P_{eff} * SP
Worst-case three orbit average power generation $P_{three-orbit}$ (W)	8.96	P_{orbit}

Table 4.2: Power generation calculations [68]

The worst-case power timeline is formulated on the basis of the power budget and the concept of operations. Important factors influencing the power consumption timeline are:

- The CubeSat remains in sun for a minimum of 61.15 % of the orbit time. This is obtained from Systems Tool Kit (STK) simulations (Figure 4.2). From the plot, the maximum eclipse period is around 38.85 % of the orbit period during the lifetime of 2 years. Hence, there is a minimum of 61.15 % sunlight time in any orbit. STK is used to obtain the eclipse times assuming an International Space Station (ISS) orbit for the satellite. The parameters for the simulations can be found in [69].

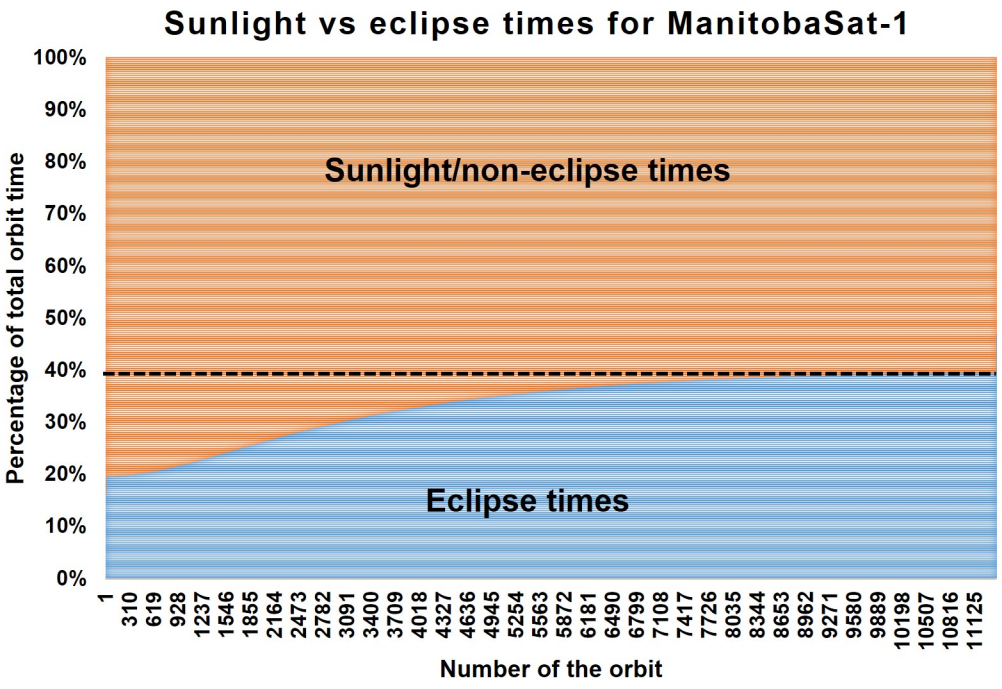

Figure 4.2: Estimation of the sunlight time available during the maximum eclipse period for the mission lifetime (2 years)

- No payload images are taken during eclipse.
- Each orbit is 5560 seconds. In three orbits or 16680 seconds, worst-case eclipse occurs between 3401-5560 seconds, 8961-11120 seconds and 14521-16680 seconds.
- Payload images are taken once approximately every two hours during the non-eclipse period.
- Communication between the CubeSat and the University of Manitoba ground station takes place six times per day on average. The longest contact duration is 400 seconds. Though, in reality not every pass is 400 seconds long, for simulation purposes every pass duration is taken to be 400 seconds. This is because I want to consider the worst- power consumption scenarios.

4.3 Dynamic Power Subsystem Simulations

Dynamic power system simulations verify that the batteries are sized appropriately to ensure safe operations throughout all phases of the mission. The simulation results discussed below only consider the normal operations, that is, the CubeSat is assumed to have completed the commissioning process and started regular operations. From the analysis, factors which influence the size of the battery, such as charging cycles, discharging cycles and depth of the discharge are determined. Since, the power subsystem should be able to support worst-case and peak power requirements at the worst-case eclipse, the simulator uses the worst power consumption timeline. Now, I discuss how the simulator components are adapted to the ManitobaSat-1 power subsystem.

4.3.1 Configuring the Simulator Components

The simulator model structure formulated in Chapter 3 is configured below.

Power consumption input The values for the power consumption with corresponding orbit time for the first three orbits is fed as a repeating table for the entire simulation time (Figure 4.3). The maximum power consumption occurs between 7201 seconds (1.29 orbits) and 7320 seconds (1.32 orbits) in the timeline when the communication and camera are on. Since, the camera is on approximately every two hours, it would be on at 14521 seconds. Because eclipse occurs during that period, imaging can't happen during that time. Hence, I consider a slightly shorter period than the 2 hour gap for imaging in the timeline. The second round of imaging occurs between 14001 seconds (2.52 orbits) and 14120 seconds (2.54 orbits). I enter the power consumption and time vectors in the repeating table in Simulink.

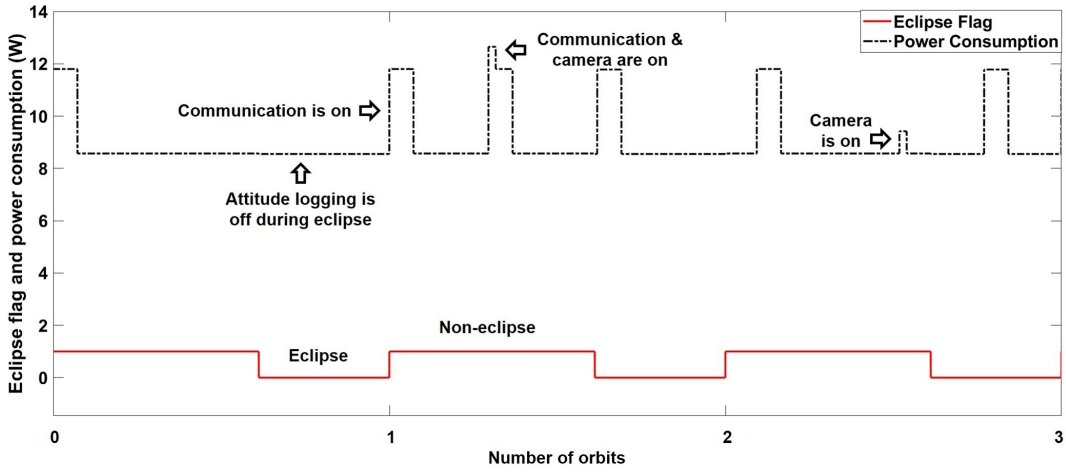

Figure 4.3: Output Power consumption and eclipse flag waveforms of the CubeSat over three orbits

Battery module To develop the interpolation lookup table for the battery module, the V vs C curve from the battery cell datasheet is used [67]. I propose a preliminary battery cell configuration of 2S3P or three strings (parallel) with two cells each (series). The values from the V vs C plot can be modified for the configuration by multiplying the capacity and voltage by 3 and 2 respectively. Note that the values for the capacity in the datasheet aren't in SI units and are converted from mAh to A-sec. The lookup table plot for the battery is shown in Figure 4.4.

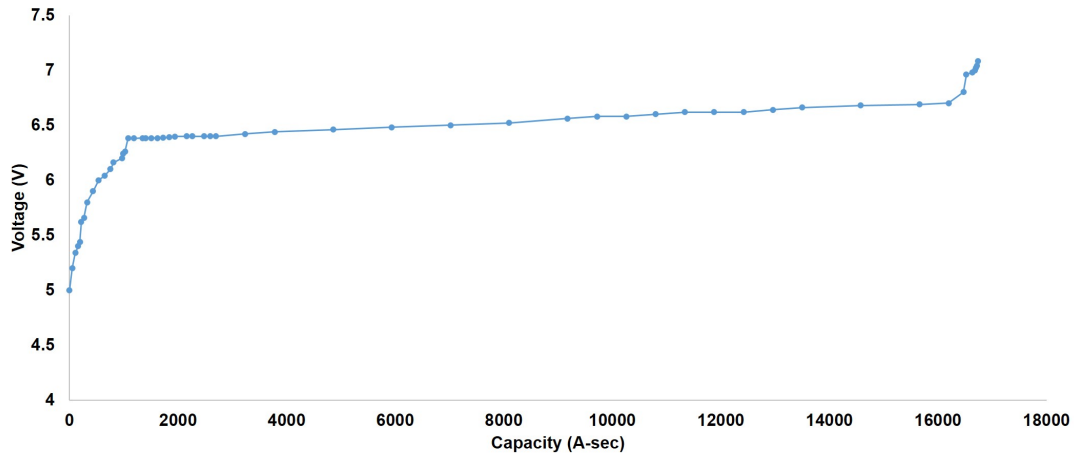

Figure 4.4: The battery module lookup table [67]

Solar array module The solar cell configuration is three cells per string with five strings facing the sun. To develop the interpolation lookup table for the solar array module, the I vs V curve is obtained from the D vs V curve in the solar cell's datasheet by multiplying the currents by the cell area which is 27.43 cm^2 [68] and in SI units by multiplying with 0.001 to have the currents in A. The values from the I vs V plot can be modified for the configuration by multiplying the voltage and current by 3 and 5 respectively. The lookup table plot for the solar array module is shown in Figure 4.5.

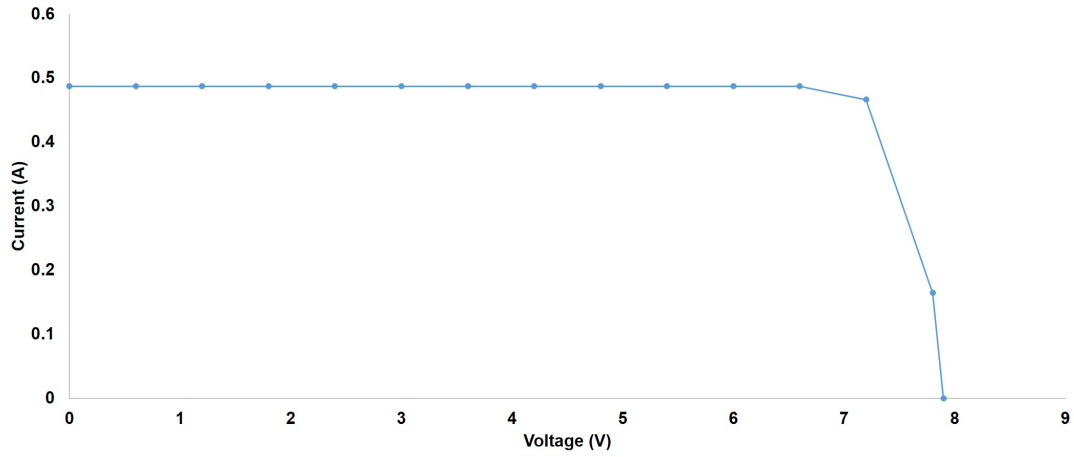

Figure 4.5: The solar module lookup table [68]

Eclipse flag The signal stored in the repeating table for the eclipse flag is graphically shown in Figure 4.3. In the figure, the power timeline is synchronized with the eclipse flag such that the worst-case power consumption occurs right after the eclipse time. This is the worst-case power consumption timeline because the battery charges during the non-eclipse period as solar array current is available and discharges during eclipse. Therefore, the battery charge is the lowest at the end of an eclipse period. Because the communication with the ground is the most power intensive task for the satellite, it is synchronized with exiting an eclipse to create the worst-case power scenario.

Solar isolation angle The worst-case pointing error of the ADCS is $\pm$ 24.00 $^\circ$ and the cosine of its value in radians (0.42 rad) is multiplied with the eclipse flag and the passive solar array current to obtain the effective solar array current.

Battery charging controller The cell data sheet specifies a cut-off voltage of 3.54 V [67]. For the 2S3P arrangement of the battery, the cut-off voltage is 7.08 V (3.54

* 2 V).

Internal resistance of the battery From the data sheet, the average internal resistance of the cell is 20.00 mΩ [67]. Hence, for a 2S3P configuration, the equivalent resistance is approximately 13.00 mΩ.

Initial conditions of the integrator The nominal capacity for the cell is 1500.00 mAhr which isn't at 100% state of charge (SOC) [67]. The maximum capacity is 1550.00 mAhr and I choose a reasonable lowest state of charge of around 30 % [67]. With a configuration of 2S3P, according to the equations in Section 3.2, I need to multiply these values by 3. The initial capacity of the battery is set at 4500.00 mAhr, the lower saturation limit to 1450.00 mAhr and the upper saturation limit to 4650.00 mAhr. Converted to SI units, the values are 16200.00, 5220.00 and 16740.00 A-sec respectively.

4.3.2 Simulator Results and Discussion

The simulator is run with the components configured as described above. The results of the simulator provided here are for the time period after the system has achieved steadiness, that is, I allow the simulation parameters to stabilize. Consequently, Figure 4.6 demonstrates that stability has been reached by the time the simulation has reached the 100th orbit. This also shows that the power subsystem is able to meet the requirements of the subsystems for a substantial amount of time. From the figure, the state of charge is stable for a long time. This is because the battery is consistently charging upto 100% and its state of charge doesn't drop below 82.3% for over 100 orbits. No unexpected fluctuations in the state of charge are occurring and steadiness is seen in the system.

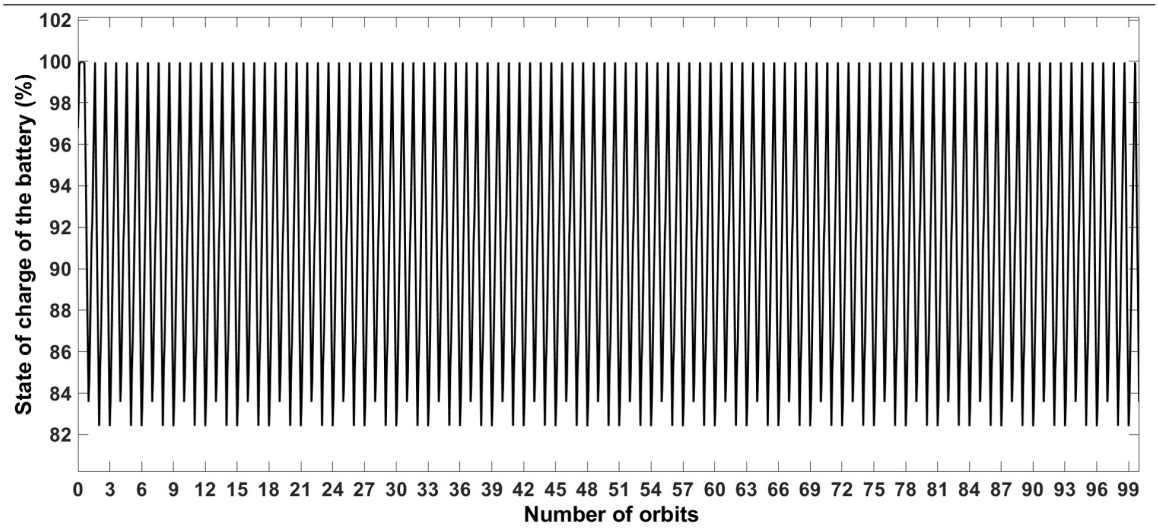

Figure 4.6: State of charge of the battery over the first 100 orbits demonstrating that stability has been reached during this period

Only a period of three orbits is shown in all plots for better visualization. I discuss below the trends and observations made from the results.

a) From Figure 4.7, the solar array current isn't completely constant throughout the sun-facing time of an orbit. This can be explained by looking at the corresponding changes in the battery voltage and the I -V curve of the solar array (Figure 4.5) where there is a decline in the current when the voltage reaches near the cut-off voltage. The drop in current can be observed midway through an orbit. At the corresponding times in Figure 4.7, the solar array current is 0 during eclipse as expected.

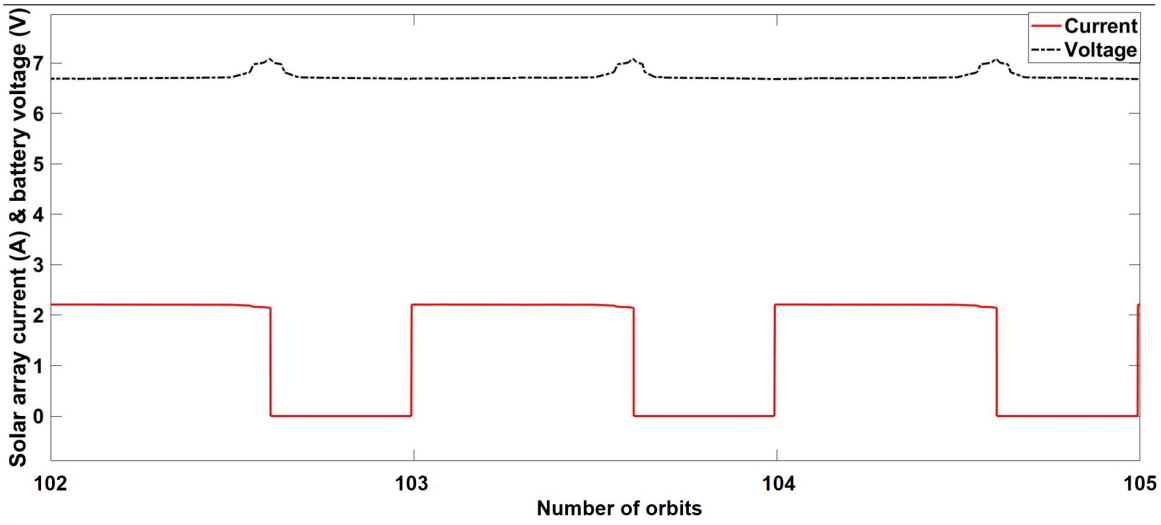

Figure 4.7: Output waveforms of the solar array current and battery voltage for three orbits

b) Figure 4.8 shows the battery voltage and the power consumption as a function of time. As expected, the voltage keeps increasing before an eclipse and when an eclipse occurs, the voltage gradually decreases. The voltage reaches its peak points right before an eclipse begins. The power consumption dips slightly when an eclipse starts.

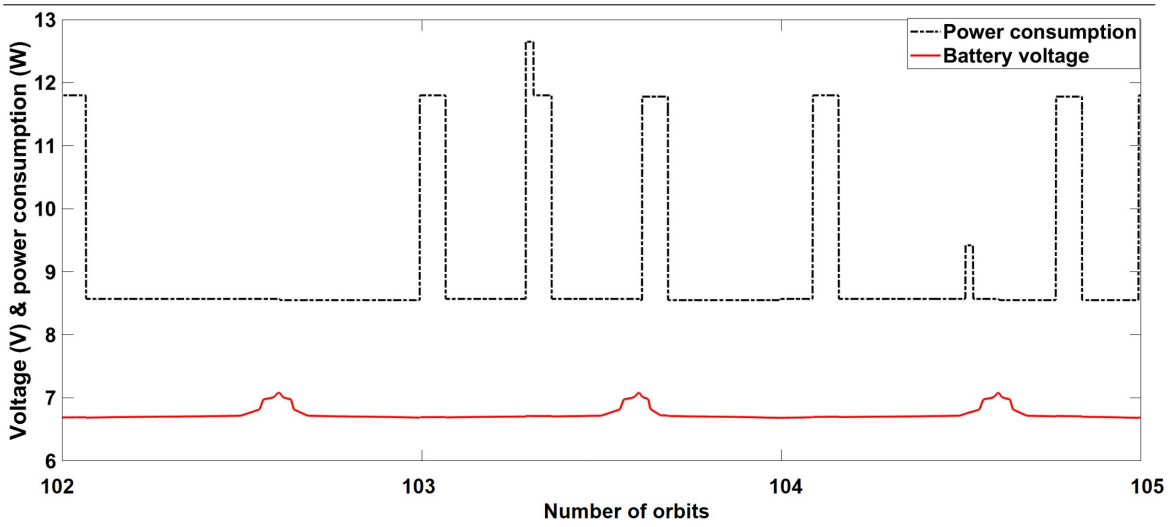

Figure 4.8: Output waveforms of the battery voltage and power consumption for three orbits

c) In Figure 4.6, the initial state of charge of the battery is about 97%. The battery is charged during non-eclipse time to a 100% SOC and discharges during eclipse leading to a lower SOC. This is understandable as solar current is not available during eclipse. A similar trend is seen in Figure 4.9.

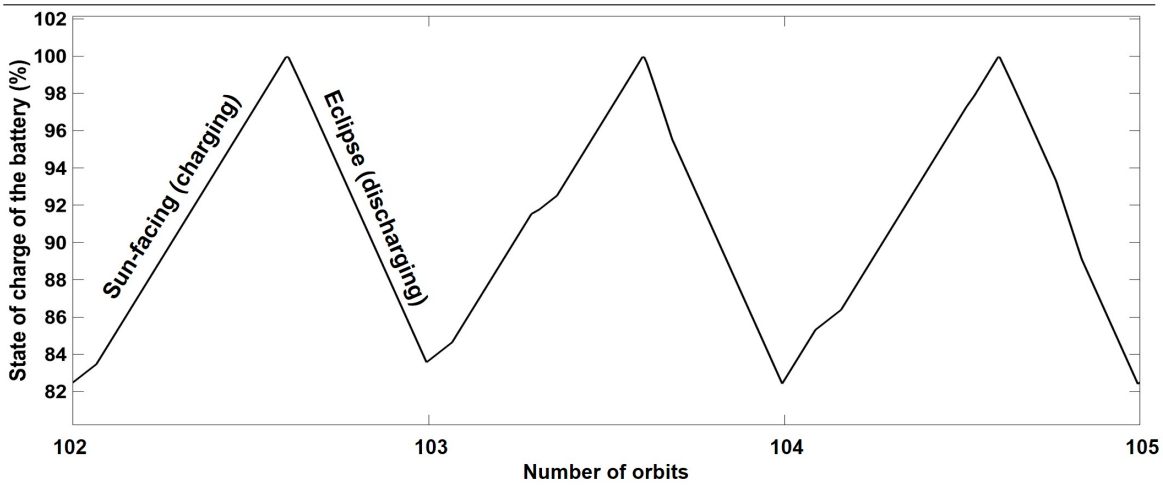

Figure 4.9: State of charge of battery for three orbits

d) Looking at Figure 4.10, the current into the battery is positive (during sun-facing time) and the current going out of the battery is negative (during eclipse time). This is consistent with the sign convention adopted for Equation 3.6 in Chapter 3.

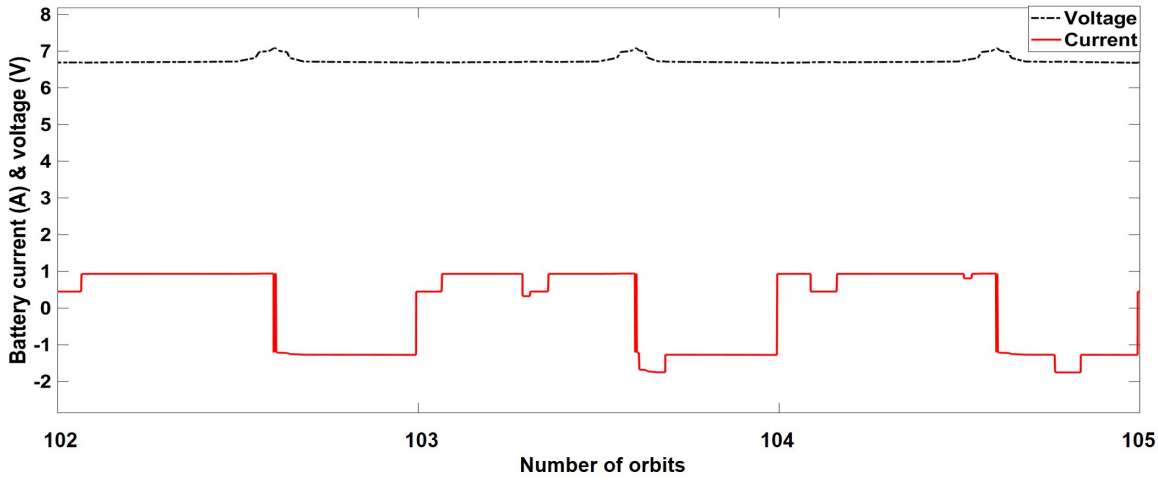

Figure 4.10: Battery current and voltage for three orbits

e) Power consumption is the highest when there is communication and the payload camera is on. It is the lowest when eclipse takes place without communication and with attitude logging being off. Worst-case communication occurs between 1-400 seconds, 5560-5960 seconds, 7200-7600 seconds, 9000-9400 seconds, 11641-12041 and 15402-15802 seconds. Between 9000-9400 seconds, eclipse and communication take place simultaneously and between 7200-7320 seconds, the payload camera takes an image and communication takes place. The camera is on again from 14001 to 14120 seconds. These help analyse worst-case power consumption scenarios.

f) In Figure 4.3, an eclipse causes a very minor decrease in power consumption as attitude logging doesn't take place. The eclipse flag toggles between zero and one. Zero denotes eclipse period and one denotes no eclipse.

g) Earlier, the plot of the state of charge during the later orbits showed that the battery is stable even after a long period of time. Also, the state of charge (Figure 4.9) drops during an eclipse.

Other Inferences From the Results

Depth of discharge (DOD) The DOD is the maximum decrease in the state of charge of the battery from its maximum charge. This maximum decrease occurs during an eclipse. It can be estimated from the SOC vs time plot and is shown in Figure 4.11. The depth of discharge during eclipse is estimated to be 17.70% (Equation 4.1 where SOC_{max} and SOC_{min} refer to the maximum SOC possible and the minimum SOC during eclipse respectively) which is reasonable. The DOD can also be calculated from the power draw during an eclipse and the battery capacity.

$$DOD = SOC_{max} - SOC_{min} \tag{4.1}$$

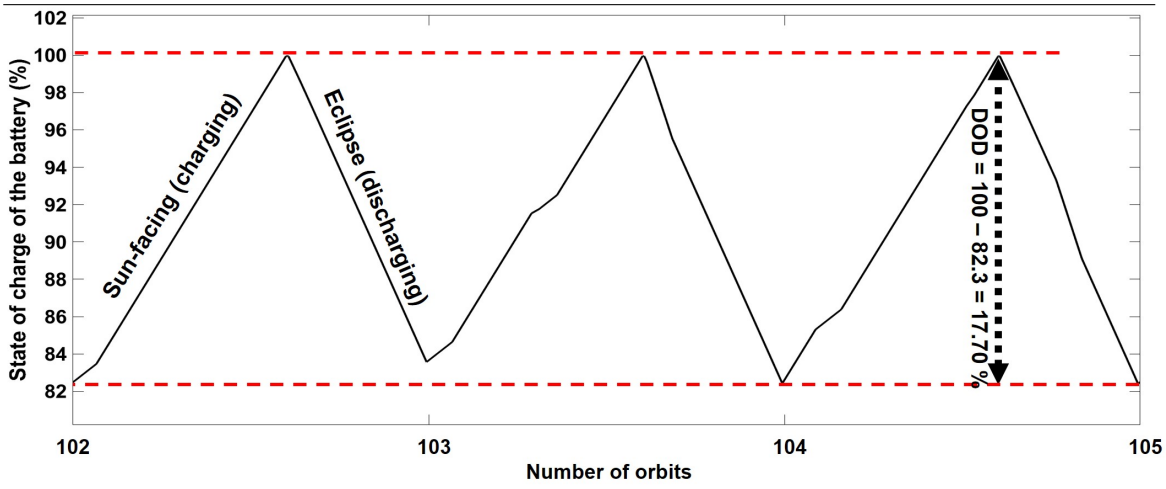

Figure 4.11: Estimation of DOD from the state of charge plot for the battery

Power draw during an eclipse The orbit average power consumption from Figure 4.3 is 9.04 W and the orbit eclipse period is 2160.06 seconds. Therefore, the power draw during an eclipse (Equation 4.2 where P_{orbit} is the orbit average power

consumption) is 9.04 * 2160.06 / 60 = 325.44 W-min. This gives the DOD (Equation 4.3) = 325.44 / 1733.40 = 18.77 % which is slightly higher than the estimated 17.70 %. The discrepancy is most likely the result of a different power consumption calculation in the two cases. While estimating the orbit average power consumption in the static power budget, I assume that the camera is on for 120 seconds in every orbit, which isn't always the case since the camera takes an image every two hours in the simulations. Also, communication occurs two times in an orbit sometimes but the static power budget only takes into account the cases where there is one communication window per orbit. However, the results are very close to each other and are reasonable.

$$Draw_{eclipse} = \frac{P_{orbit} * Period_{eclipse}}{60_{seconds-to-minute}} \tag{4.2}$$

Here, $Draw_{eclipse}$ is the power draw during eclipse, P_{orbit} is the orbit average power consumption and $60_{seconds-to-minute}$ is the conversion factor from seconds to minute.

$$DOD = \frac{Draw_{eclipse}}{C_{storage}} \tag{4.3}$$

Here, DOD is the depth of discharge of the battery, $Draw_{eclipse}$ is the power draw during eclipse and $C_{storage}$ is the storage capacity of the battery.

Maximum battery current The peak battery current is close to 1.00 A which is less than the maximum charge current (3.00 A for a 2S3P configuration) specified for the battery in its datasheet [67].

The results confirm that the selected battery sizing is suitable. The above conclusions from the simulations are logical and give confidence in the simulator design. The static and dynamic analyses prove that the power subsystem design is suitable and sufficient.

4.4 Summary

This chapter describes the power subsystem of a student CubeSat, ManitobaSat-1, funded by the Canadian Space Agency and the application of the simulator described in Chapter 3 to this mission to assess the design of the power subsystem and to size the batteries. I start with the primary goal of ManitobaSat-1, its operations and how these affect the power consumption of the CubeSat. Based on the concept of operations, a power budget and power consumption timeline are devised. STK simulations are used to identify the worst-case eclipse periods for the mission with a 2 year lifetime. It is observed that the timeline repeats every three orbits. With the solar cells selected, I performed a static power analysis to conclude that the power generation is greater than the consumption and the chosen solar array sizing is appropriate.

In the power timeline, maximum power consumption occurs during communication with the ground, and minimum power consumption occurs during eclipses and no communication. To satisfy the objective of the mission, the camera takes images every two hours but doesn't utilize a significant amount of power. During the 120 s of the camera and communication being on simultaneously, the consumption peaks at 12.65 W. I identified the worst-case scenarios for consumption in the timeline such as when communication takes place right after eclipse and when the camera and communication are on simultaneously at the same time. This ensures that the power subsystem is able to perform well even in the worst of situations.

The detailed procedure to configure all components of the power subsystem simulator with ManitobaSat-1's parameters is provided. This gives other researchers a case study when considering applying the simulator to other missions. The simulations are run through several orbits and results and trends are analyzed. In addition to studying basic parameters such as battery voltage, current, solar array current,

SOC and power consumption at anytime, I demonstrated the estimation of other important variables such as the depth of discharge, capacity and the power draw during eclipse. With the conclusions drawn from analyzing the simulation results, I surmise that the proposed battery sizing (2S3P) is appropriate and the described design is sufficient for the mission.

In this thesis, the virtual ground station's virtual satellite model was developed and tested for ManitobaSat-1 as one example, though it is simple to apply it to any other satellite using DET. The power subsystem simulator for ManitobaSat-1 described in this chapter forms the virtual satellite model.

Chapter 5

The Fault-Management System

5.1 Introduction

Chapters 3 and 4 focused on the design and development of the virtual spacecraft model which partially verifies the first hypothesis of my thesis that describes a virtual ground station based on a mathematical model of the spacecraft or the virtual spacecraft model. For the second hypothesis, traditional analysis techniques from Statistical Process Control (SPC) to provide near real-time health and diagnostic evaluations, hence forming a fault-management system, are deemed useful by reducing human involvement in the analysis of telemetry data. To do this, the system monitors telemetry and compares it to the results from the virtual spacecraft model for early fault diagnosis. This chapter details the development of this system (Figure 5.1) and demonstrates the detection of major power subsystem faults in a satellite. To my knowledge, this is the first time that SPC is being used in satellite operations and maintenance making it an important contribution to the field of satellite engineering.

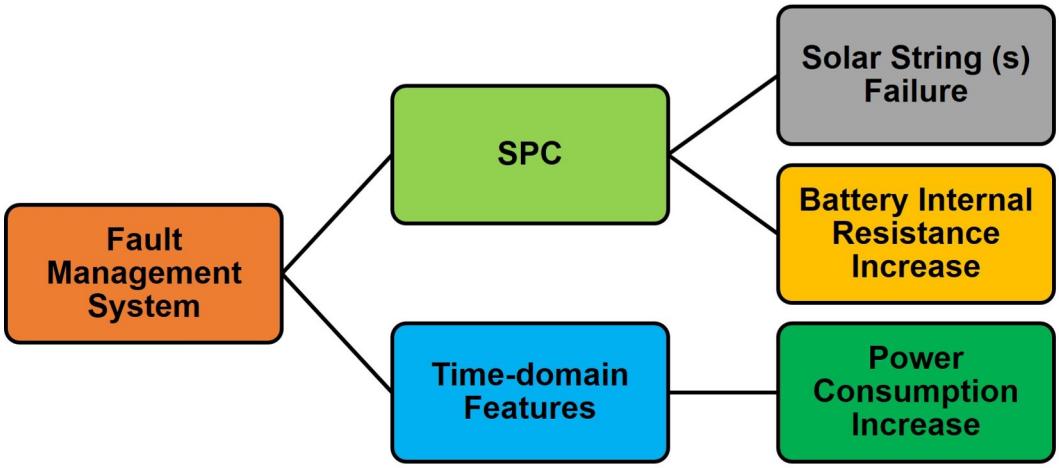

Figure 5.1: The broad structure of the fault-management system

The SPC methodologies such as control rules and charts form a key aspect of the space systems monitoring from the ground station are briefly discussed prior to beginning with the fault-management system's development. This is followed by a tailored pseudocode for each control rule irrespective of the fault. A set of fault-detection rules (specific for each fault) is identified using observations from running the code on parameter datasets (such as internal resistance and solar string current). The results are discussed and inferences are drawn for each fault.

Different failures affect a process in unique ways and in some cases control charts are not an effective tool to conduct fault detection. To detect the subtle changes in complex waveforms such as the power consumption, time-domain features are used to represent the system. Algorithms are devised based on how different faults in the power consumption affect the features individually and combinatorially to perform classification to identify any anomalies. This algorithm is also applied to a hypothetical mission to signify that it works irrespective of the operational plan of a mission.

5.2 Statistical Process Control Module

SPC is a traditional, charting, monitoring and diagnostic tool that the manufacturing industry has been using for decades to improve and maintain product quality [70]. SPC works on the theory that a process behaves predictably to manufacture products which conform to engineering requirements causing the least possible waste (i.e., scrapped product). The process should operate with little variability around the product requirements. When a process is not behaving properly, key metrics using SPC can be identified that can indicate early signs of a fault. In doing so, SPC provides a number of powerful tools aimed at achieving process stability and reduction in variability [71]. SPC plays a critical role in the virtual ground station as it is used to automatically monitor satellite telemetry streams for early signs of anomalies. To develop the SPC module, the virtual satellite model is modified to introduce (simulate) different kinds of failures. SPC algorithms are developed and tested using the virtual spacecraft model.

5.2.1 Components of SPC: Control Charts and Control Rules

Control charts developed in the 1920s by Walter A. Shewhart [72] are regarded as the most technically sophisticated SPC tool. It is deployed using Shewhart's theory of variability that stipulates that a process displaying natural variability is said to be in statistical control whereas a process showing assignable causes of variation which are not part of chance cause pattern and natural variability are out of control. While chance causes, also known as common causes, are embedded in the process or system, assignable causes are the result of an external source [71]. In my thesis, I apply SPC control charts to spacecraft health monitoring.

In SPC, control charts function as process behaviour charts or on-line process

monitoring techniques to estimate parameters of a process to identify if the process is in equilibrium or not [71]. A typical control chart can be seen in Figure 5.2. It has a centre line, upper control limit (UCL) and lower control limit (LCL). These control limits are chosen such that all of the sample points lie between these limits when the process is in control. Using just the control limits does not allow detection of very small shifts. The Western Electric rules also known as the SPC control rules (Table 5.1) are used alongside the control charts to detect anomalies [47][71]. In Table 5.1, zone A refers to the region between a control limit and $\pm 2\ \sigma$, B is the region between $\pm 2\ \sigma$ and $\pm 1\ \sigma$ and C lies between the $\pm 1\ \sigma$ and the mean where σ is the standard deviation of the test data set. The control rules and charts enable identification of upward and downward trends, mixtures with no points in certain zones, stratification and over-control to indicate that the process is not in equilibrium. Each of these conditions or patterns typically points to a specific type of defect, which are identified by studying real and simulated data.

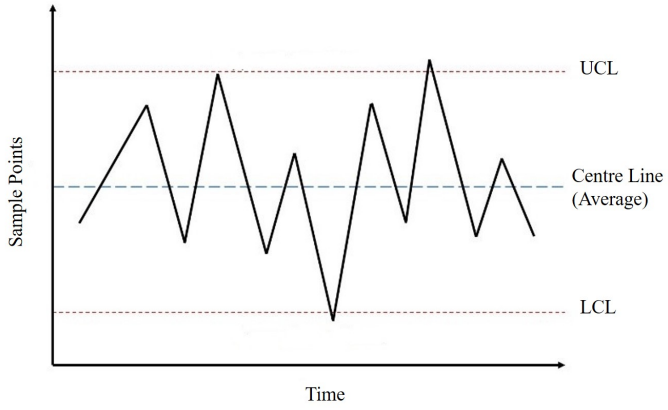

Figure 5.2: A typical control chart

Rule Number	Name/Pattern	Rule Description
1	Beyond limits	One or more points beyond the control limits
2	Zone C (small shifts from the mean)	Eight or more points in a row on the same side of the mean
3	Trends (up and down)	Seven points or more points in a row continually increasing or decreasing
4	Over-control	Fourteen or more points in a row alternating up and down
5	Zone A (large shifts from the mean)	Two out of three points in a row are more than 2 σ from the mean in the same direction (zone A or beyond)
6	Zone B (small shifts from the mean)	Four out of five points in a row are more than 1 σ from the mean in the same direction (zone B or beyond)
7	Stratification	Fifteen points in a row are all within 1 σ of the mean on either side of the mean (zone C)
8	Mixture	Eight points in a row exist, but none within 1 σ of the mean, and the points are in both directions from the mean (no points in zone C)

Table 5.1: SPC Control Rules [73]

There are different types of control charts based on the characteristics being tested and their applications. One of the main types is an individual chart which is used in this thesis. It is generally applied to detect deviations from the mean level of a process when it is not possible to have data subgroups (data points that are collected simultaneously when more than one machine is giving data). The spacecraft is the only system from which I am collecting data, and I will only have one data point at any particular time. An individual chart shows a variety of properties about the process such as trends and center stability in the distribution. This type of chart is typically used with a moving range chart in industrial applications. Moving range charts are used to monitor the process variation by tracking the absolute differences of each measurement from its predecessor. To create these charts for any process, an initial set of observations is used to estimate the mean and the standard deviation of the process.

But, health-monitoring of a satellite is not a regular industrial process. Typical industrial processes involve procedures with chemical, electrical, physical and mechanical steps to help in the manufacturing of an item (s), generally on a large scale. Industrial processes usually follow batch processes while for satellite health-monitoring, there are no manufacturing batches involved. Unlike an industrial process, I do not have a large amount of sample data available for a satellite health-monitoring system. Most of the data I have available is also based on simulators. Since, I can calculate the standard deviation of my data that I obtained from a simulator, that is, σ is a non-estimated or definite value, I do not require any estimation of σ. A moving range chart is generally used when the standard deviation has to be estimated [74]. Therefore, I only use the individual chart. The common eight SPC control chart patterns (Figure 5.3) corresponding to the SPC rules in Table 5.1 include: beyond control limits, zones A, B, C defined above, upward and downward trends, mixtures with no points in certain zones, stratification and over-control curbing the natural

process variability. Each of these conditions could point to a specific type of defect, which need to be identified during the investigation of the process [31].

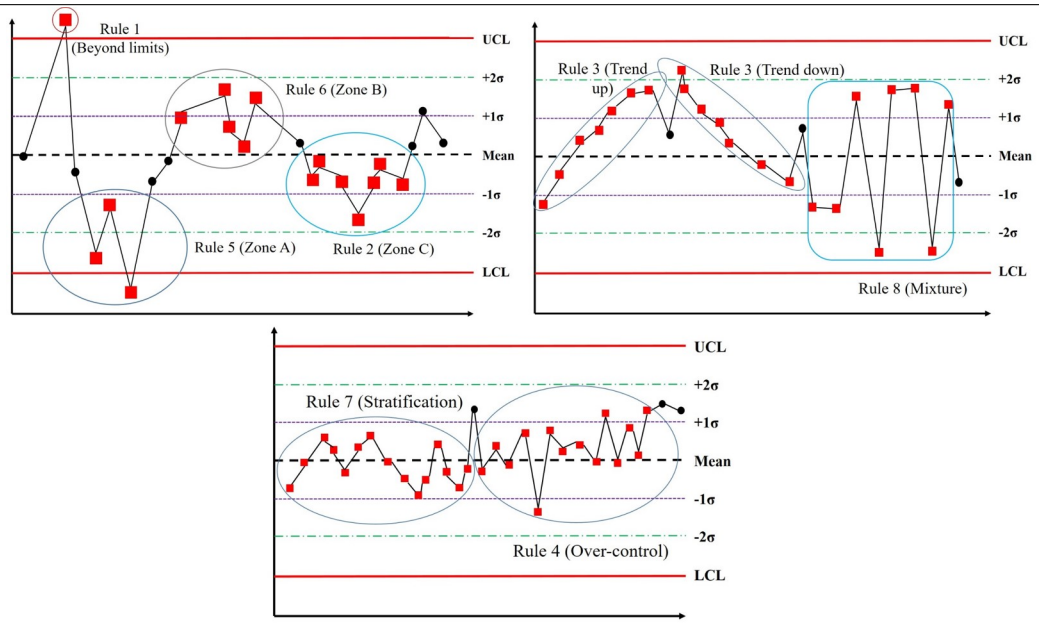

Figure 5.3: Control chart patterns [71]

5.3 Developing the SPC Component of the Fault-Management System

5.3.1 Application of the Control Rules

SPC is usually applied to industrial and machine data. Spacecraft data is not abundant as most space industry missions are different and require unique interpretations of the control rules. Hence, applying these rules to spacecraft data requires a slightly different approach and the control rules have to be analyzed for their applicability. In

fact, there are numerous interpretations of the control rules and multiple versions of them being used in various sectors that are not typical industrial and manufacturing processes. In such cases, it is common to change the number of points mentioned in the standard Nelson and Western Electric control rules to be suitable to the data in question.

A paper from the Agency for Healthcare Research and Quality [75] discusses the importance of smart application of traditional SPC rules. It is necessary to recognize the need to choose the datasets in a sophisticated manner and understand the practical implications of the control charts. SPC is driven by "correct and smart application" against blindly applying the rules [75]. However, extensive testing is required to substantiate the accuracy of the rules. One should note that the violation of a control rule does not necessarily indicate a fault and that one should be aware of false alarms. Many SPC software tools in the market such as QI Macros [76] provide the user with the option to edit the rules according to their needs. They also provide a variety of standard rule sets to choose from apart from Nelson and Western Electric rules including Westgard, AIAG, Montogomery and Healthcare [76]. Unlike the healthcare industry, spacecraft data does not have a set of rules suggested by experts probably due to lack of extensive SPC application in the industry.

5.3.2 Algorithms for the Control Rules

Before developing the algorithms for the rules, I write a script to differentiate the points in the dataset (any telemetry data) on the basis of the zones they lie in, if they lie above or below the mean and scores which are defined by the zone they belong to. The script written for this is called zonescript and is given in Appendix D.1. Any point lying above the UCL or below the LCL has a zone value of X meaning beyond control limits and any point lying in zones A, B and C are given zone values of A,

B and C respectively. Points lying above the mean and below the mean have sides value of P and N respectively. Lastly, points in zones X, A, B and C are assigned scores value of 8, 4, 2 and 1 respectively. The variables scores, zones and sides for each point are used while developing the algorithms for each control rule. Condensed pseudocode for each of the control rule algorithms denoted by crAlg N where N is the control rule (Table 5.1) number that I developed are given in Appendix D.2. The steps of development discussed so far are depicted in Figure 5.4. As seen in the figure, zonescript gives out the zone of each point and computes the side of the mean the point is on along with its score. I use these results to develop the control rule algorithms crAlg N.

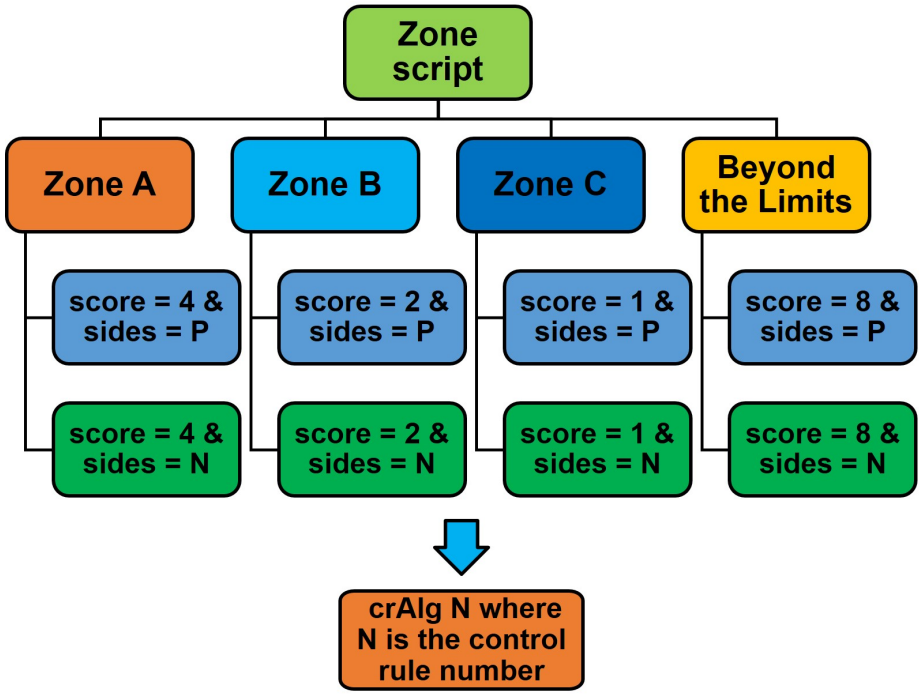

Figure 5.4: Summary of the steps in developing the algorithms for the control rules

In traditional SPC for industrial and manufacturing applications, the control lim-

its and zones are defined by the sample's standard deviation and mean. If the control limits are calculated based on the data received during each pass separately, the limits would be dynamic and would depend on the data received only during a certain time period. This might result in incorrect limits since faults might be present in them. For example, if the satellite has a pass at 2000 seconds and 4000 seconds since the current time and a fault is injected only after 2100 seconds, the limits calculated for the data points from 1 -2000 seconds and 2001 - 4000 seconds would be different. This will increase the probability of false alarms. Therefore, I calculate the control limits based on the ideal data collected from the initial telemetry when there are no faults. The limits and zones remain constant. It is assumed that during the first few orbits after achieving steady state, there are no faults in the satellite.

The crAlg N algorithms can be applied to any process. Before applying these to detect the faults in a spacecraft's power subsystem, the type of faults considered in this thesis for SPC analysis are discussed. Eventually, I focus on each of the selected faults and how these algorithms can be used to detect them in the virtual ground station (VGS).

5.3.3 Type of Faults

Three frequently occurring anomalies in a satellite's power subsystem chosen for implementation are loss of solar cell string(s), increase in battery's internal resistance and excessive power consumption. These can be caused by a number of reasons, some of which are listed in Figure 5.5. Only the first two faults are handled using SPC and the time-domain features are used for the third fault. Each of the faults is discussed in the next few sections.

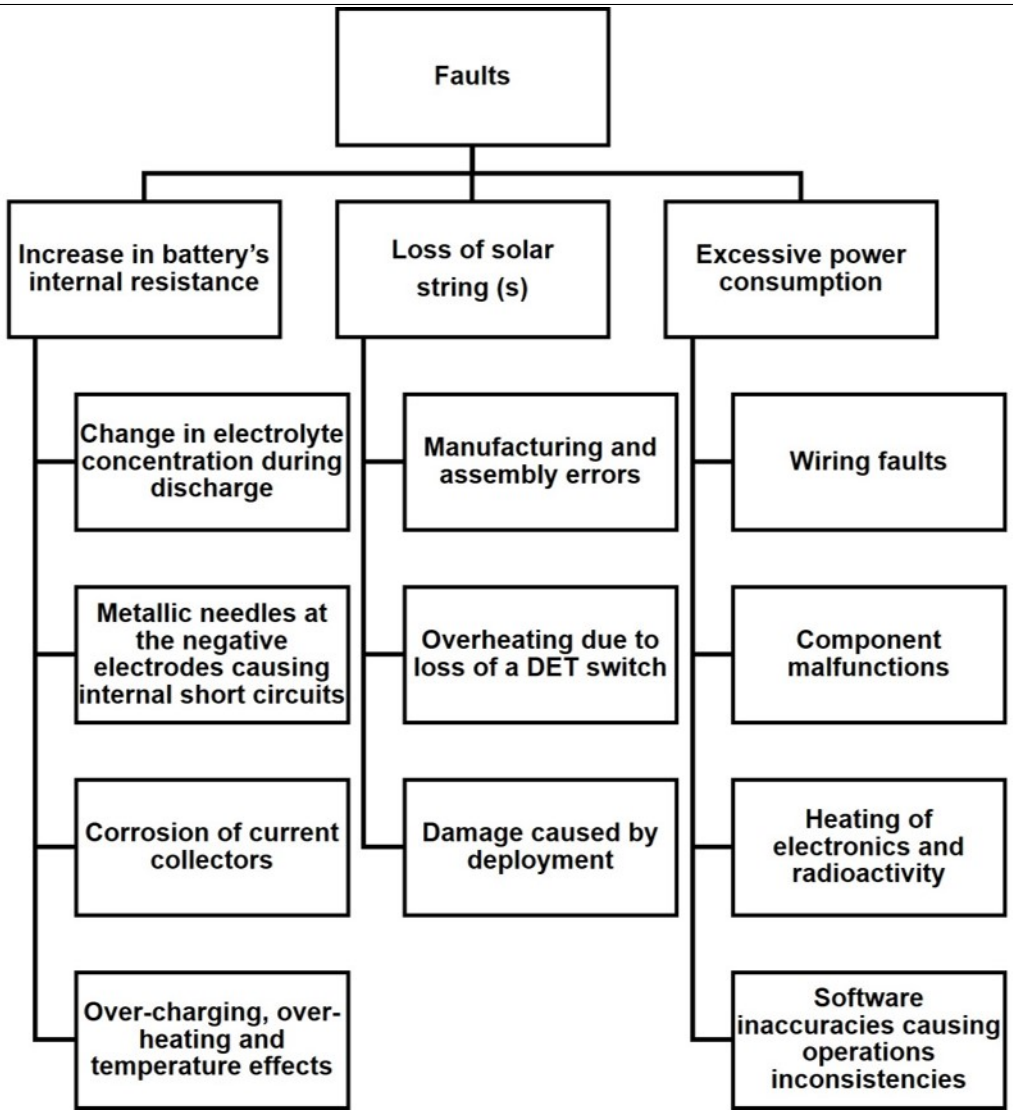

Figure 5.5: Examples of typical failures considered for the power subsystem's fault-management system. Used with permission from the Springer Nature Journal of Aerospace Systems. [77]

5.3.4 Loss of solar cell string(s)

Solar arrays are the primary source of power in many CubeSats. A fault in a solar cell string can result in a total mission failure. So, it is important to detect it as soon as possible to take corrective actions such as reducing the power consumption in a mission.

The solar array current is a function of one orbit (repeats every orbit) and its waveform depends on the eclipse period. In ideal conditions or no malfunctions in the solar arrays, the current generated is expected to be relatively constant during a non-eclipse period. During eclipse, no current is generated. As previously mentioned, the data from the first few orbits is assumed to be ideal. This gives me the ideal current datasets as a baseline of expected current values in the telemetry. I need to calculate the control limits and zone boundaries for the SPC analysis. To do this, the power subsystem simulator is run for a period of 3 orbits (after achieving stability) and the solar array current generated (called TSAI which stands for the total solar array current (I)) is collected. Likewise, the individual currents (called ISAI-N which stands for the individual string of an array current (I)) from each of the five strings is collected where N (1, 2 .. or 5) is the number of the solar string. The individual currents from all strings are expected to be equal as they are identical in configuration. Because parameters such as the power consumption, battery current and voltage follow a 3 orbit cycle, I use 3 orbits instead of 1. This allows sharing of scripts across multiple parameters in the SPC module to maintain consistency. Each of the steps (Figure 5.6) in creating a rule-based SPC system to detect a solar string failure is discussed below.

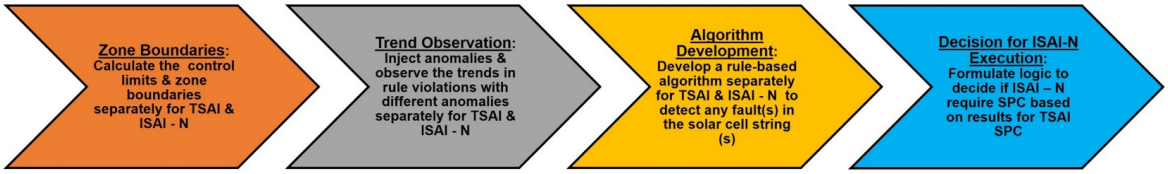

Figure 5.6: Steps in the solar string failure part of the SPC module

Zone Boundaries I write two scripts for control limit and zone boundary calculations for TSAI and ISAI-N. The ideal data is the telemetry I obtain in the first few orbits of the mission. I assume that the data from the initial orbits is fault-free. The current generated in any three orbits after reaching steady state is taken as the dataset for control limit calculations. Since, no current is generated during eclipse, I do not consider those points in this analysis. Consequently, SPC is run on the solar array currents only when no eclipse occurs. The control limits lie three deviations away from the mean. The control limits and zone boundaries for the TSAI and ISAI - N are calculated according to Equations in Table 5.2 which also contains their values. The control charts for the ideal cases of the ISAI - N and TSAI are plotted (Figures 5.7 and 5.8) and the control rules are applied.

Parameter	Individual string current	Total array current	Equation
Mean	0.478	2.174	Arithmetic mean of all elements in the dataset
Upper control limit	0.495	2.259	Mean + 3 * σ
Lower control limit	0.457	2.088	Mean - 3 * σ
+2 σ	0.488	2.231	Mean + 2 * σ
+1 σ	0.482	2.202	Mean + 1 * σ
-2 σ	0.463	2.117	Mean - 2 * σ
-1 σ	0.470	2.145	Mean - 1 * σ
+4 σ	0.501	2.288	Mean + 4 * σ
-4 σ	0.451	2.060	Mean - 4 * σ

Table 5.2: Control limits for individual string and total solar array currents. All values are in A and σ stands for the standard deviation of the dataset. The values +4 σ and -4 σ are shown for reference to visually understand how the points are distributed in the plot. They are referred to as the specification limits.

In Figures 5.7 and 5.8, the shaded regions indicate the points that violate various rules (labels shown). Note that if a point is violating a rule in one orbit, it is violating the same rule in all three orbits. As an example, the shaded region in the first orbit shows points violating rules 6 and 8. The corresponding points in the upper region of the plot, the points lying above the mean in zone B violate rules 6 and 8 in the other two orbits as well. This has been done for visual clarity.

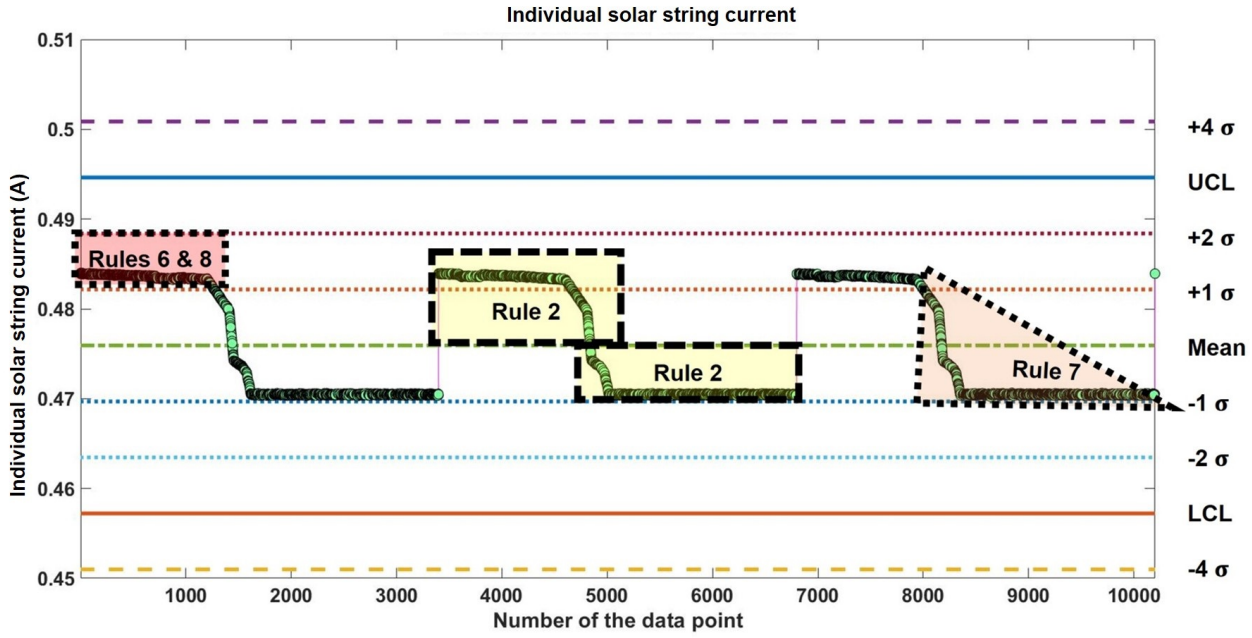

Figure 5.7: Control chart for the ideal individual solar string current over three orbits used to calculate control limits

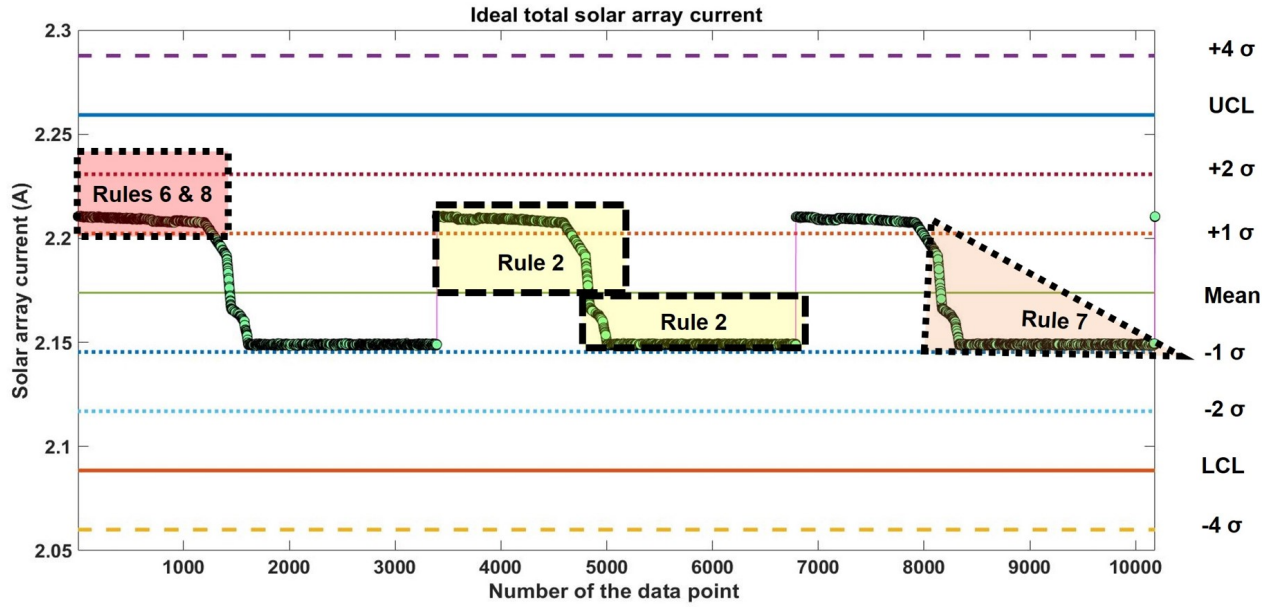

Figure 5.8: Control chart for the ideal total solar array current over three orbits used to calculate control limits

As expected, ISAI - N and TSAI ideal plots (Figures 5.7 and 5.8) follow the same trends since the currents from all 5 strings are equal. Subsequently, the observations regarding the control rules are the same for both the charts. The following observations are made from Figures 5.7 and 5.8:

- Rule 1 is not violated, that is, no points lie beyond the control limits. Most of the points lie within zones B and C.
- There are multiple consecutive points that lie very close to each other and naturally on the same side of the mean. Rule 2 is clearly violated.
- Rules 3 and 4 are violated as well. Upon inspection, it is seen that there are many consecutive points that are equal in value or have an insignificant difference between them. Hence, the tolerance for differences between the consecutive points is taken to be 0.002. This seems appropriate since consecutive points usually seem to change with a magnitude of < 0.002 and reducing false alarms is essential when studying the ideal current datasets. After making this change, no violations of these rules are observed. To conclude, rule 3 is transformed as seven or more consecutive points continuously increasing or decreasing by magnitude of greater than 0.002. Rule 4 is modified as fourteen or more consecutive points alternating up or down with a difference (magnitude) of > 0.002. The changes are made in rules 3 and 4 as they are the only rules that are dictated by the differences in consecutive data points and not by the zones the points lie in. This uniqueness makes them more conducive to modifications.
- Rule 5 is not violated because no point lies in zone A or beyond.
- Rules 6, 7 and 8 are violated. The shaded regions in the figures collectively lie in zone B and zone C on either side of the mean. As many consecutive points

in the upper part are in zone B, there have to be many more than 8 consecutive points none of which lie in zone C. Clearly, the three rules are failed.

From these observations, it is clear that rules 2, 6, 7 and 8 are violated by many points forming large regions in the plots. This is understandable as the data is periodic and these rules are very interconnected. Also, the sudden change in the current between the orbits is due to eclipses (not shown). Only sun-facing time is considered in the plots. I know that there are no faults in the data as of now and thus, these are false alarms. In the next step, I observe how different faults affect these rules.

Trend Observation In this step, I consider the different ways in which solar string failures could occur. The changes in the ISAI - N and TSAI are monitored separately. Let the solar strings be denoted by numbers: 1, 2, 3, 4 and 5. The observations are provided below.

- **A single solar cell string fails**:

  The current generated from solar cell string no. 1 is changed to 0 in the simulator and currents from all the strings are obtained. All other strings are working properly.

  From Table 5.3, rules 1 and 5 are violated only when there is a severe fault. All points lie below the LCL and violate rule 1. Rule 5 requires two out of 3 consecutive points to lie in zone A or beyond. Since, all points violate rule 1, they lie beyond zone A and thereby also fail rule 5 (Figures 5.9 and 5.10).

TSAI	ISAI - 1	ISAI - 2	ISAI - 3	ISAI - 4	ISAI - 5
Rules 1, 2, 5, 6 and 8	Rules 1, 2, 5, 6 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8

Table 5.3: List of the rules that are violated in the case of a single solar cell string failure

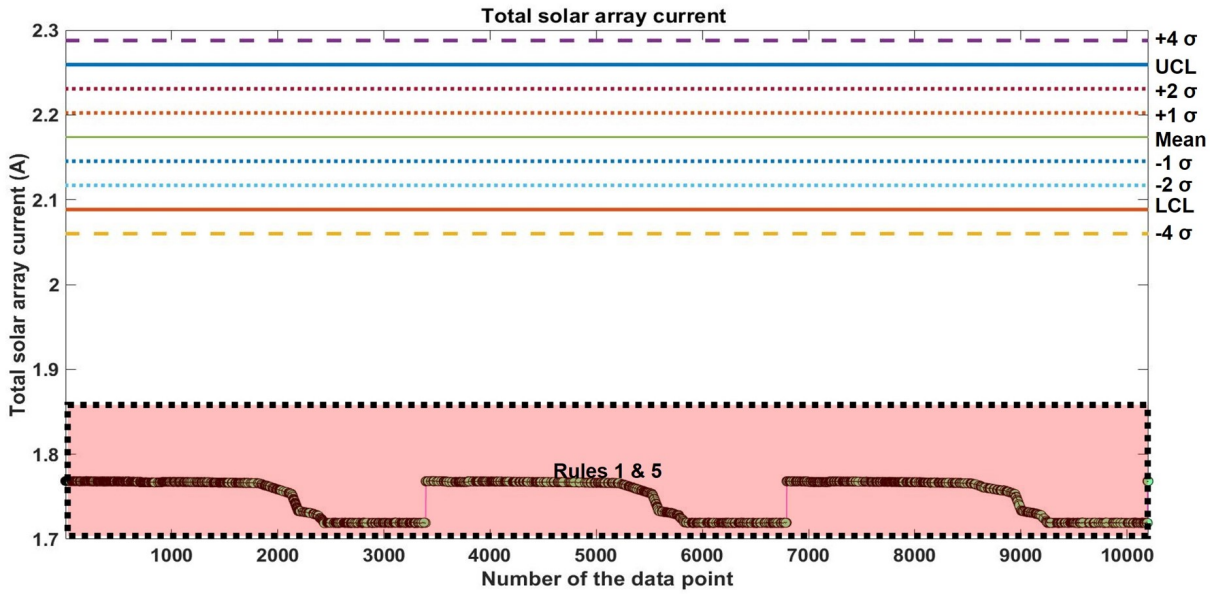

Figure 5.9: Control chart for the total solar array current with one string not generating any current

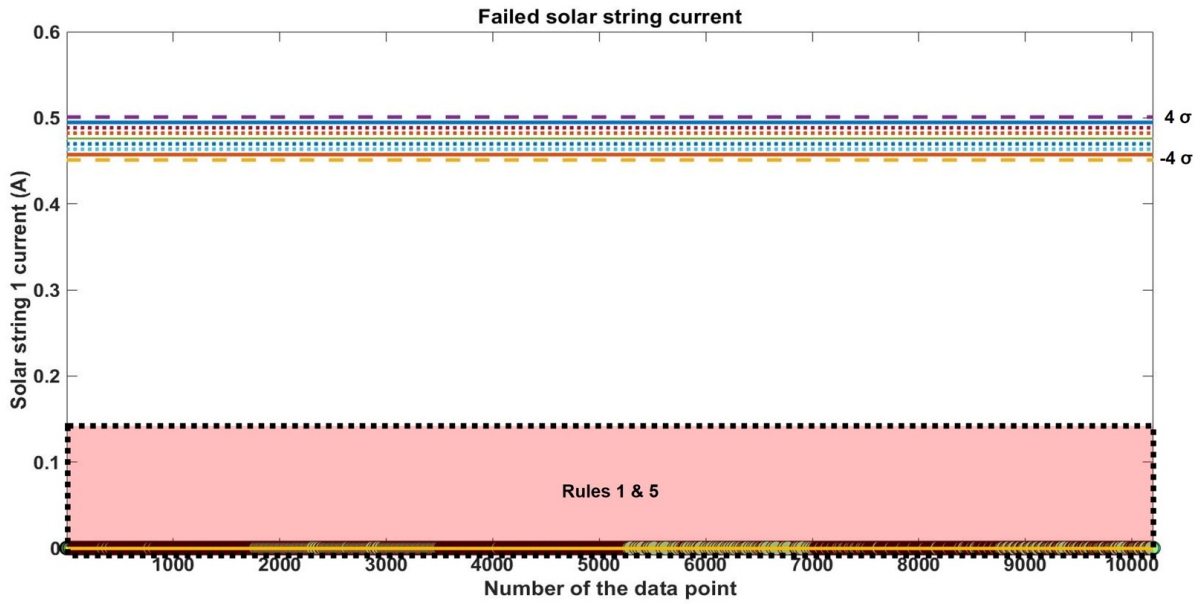

Figure 5.10: Control chart for the solar string 1 current that is generating no current

Unlike in the ideal chart, no points are in zone C or near the mean. For rule 7 to be violated, I require 15 consecutive points to lie within zone C. Hence, rule 7 is passed in the failed solar cell string case. For the other strings that are functioning properly, 2, 6, 7 and 8 are failed similar to the ideal charts. This is expected as there are no faults in the strings and their charts are identical to the ideal ISAI chart. It is interesting to see that a single string failure causes the chart to fall below the LCL completely.

- **Lower efficiency of a solar cell string**:

  For this anomaly, I multiply the current vector for the first string by 90% in the simulator's solar array module's lookup table. All other strings are working properly. In the TSAI control chart (Figure 5.11), a slightly lowered efficiency of a single string does not affect it much. No points go beyond the LCL and

rules 2, 5, 6, 7 and 8 are failed. Rule 5 is violated here unlike the ideal chart as the lower part of the waveforms in each orbit are in zone A (Figures 5.11 and 5.12). This rule requires two out of three consecutive points in zone A or beyond. Rules 6, 7 and 8 are failed as in the ideal chart but here, the points that violate them are different. The results are tabulated in Table 5.4.

TSAI	ISAI - 1	ISAI - 2	ISAI - 3	ISAI - 4	ISAI - 5
Rules 2, 5, 6, 7 and 8	Rules 1, 2, 5, 6 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8

Table 5.4: List of the rules that are violated in the case of a lowered efficiency in a single (first string) solar cell string

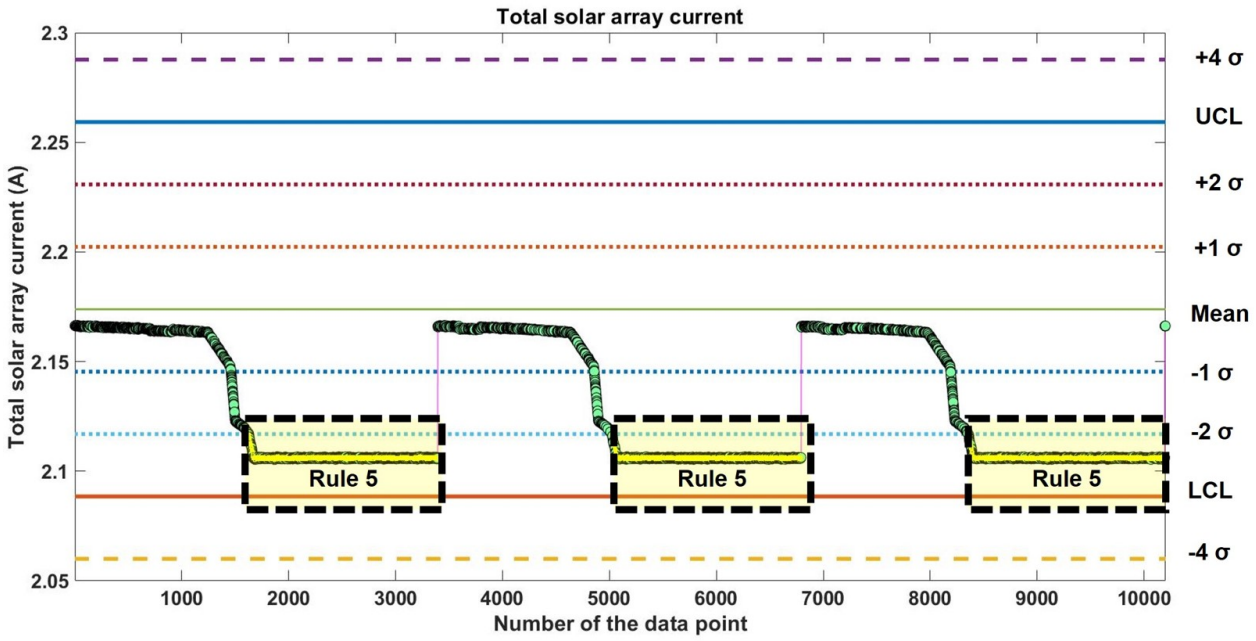

Figure 5.11: Control chart for the total solar array current with string 1 generating reduced current

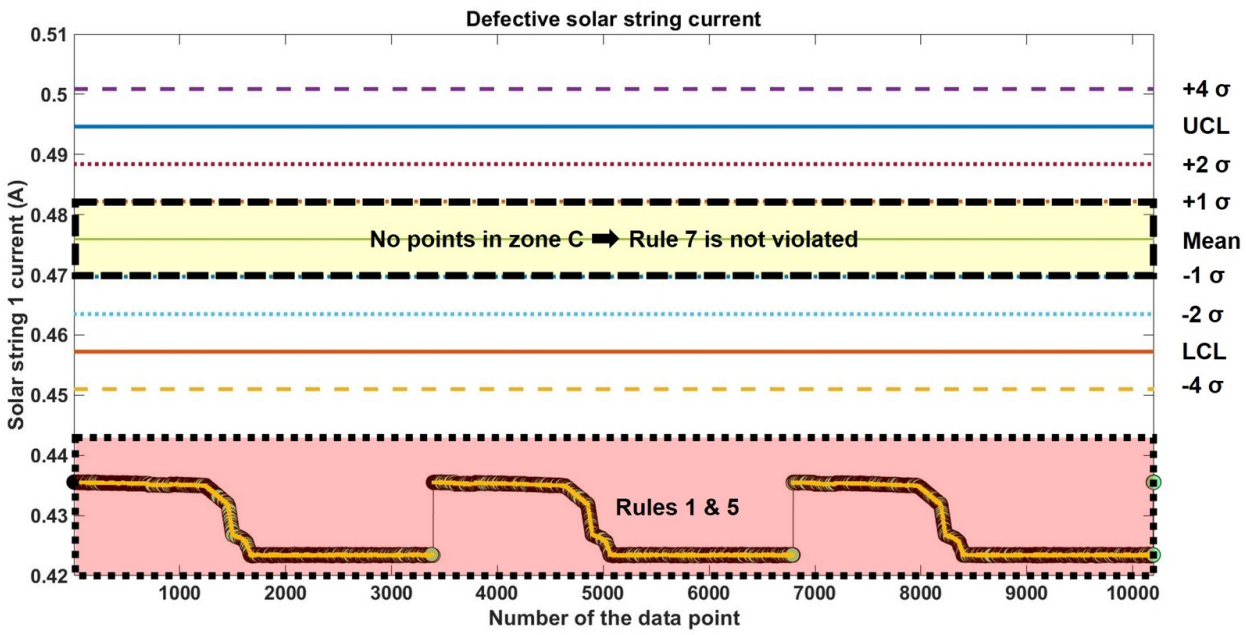

Figure 5.12: Control chart for the solar string 1 current that is generating 90% of its ideal generated current

- **More than one solar cell string fail**:

  I start with having two failed strings. Let them be strings 1 and 2. The current generated from solar cell strings 1 and 2 are changed to 0 in the simulator and currents from all the strings are obtained. All other strings are working properly. From Table 5.5, rules 1, 2, 5, 6 and 8 are violated for the TSAI control chart (Figure 5.13) and both the failed strings (Figure 5.14). This is not different from the results of a single solar string failure. Thus, I can not distinguish a single and multiple string failure(s) with just the TSAI control charts. So, performing SPC on the TSAI to determine if I require SPC analysis for individual solar string currents is essential. If I determine that an individual analysis is necessary, the SPC analysis should show me results similar to the single solar string failure case for the failed strings. The violation of rules 1 and 5 by all points makes it possible to decide if the string is generating any power or not.

TSAI	ISAI - 1	ISAI - 2	ISAI - 3	ISAI - 4	ISAI - 5
Rules 1, 2, 5, 6 and 8	Rules 1, 2, 5, 6 and 8	Rules 1, 2, 5, 6 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8

Table 5.5: List of the rules that are violated in the case of a multiple string failure (string 1 and 2 in this case)

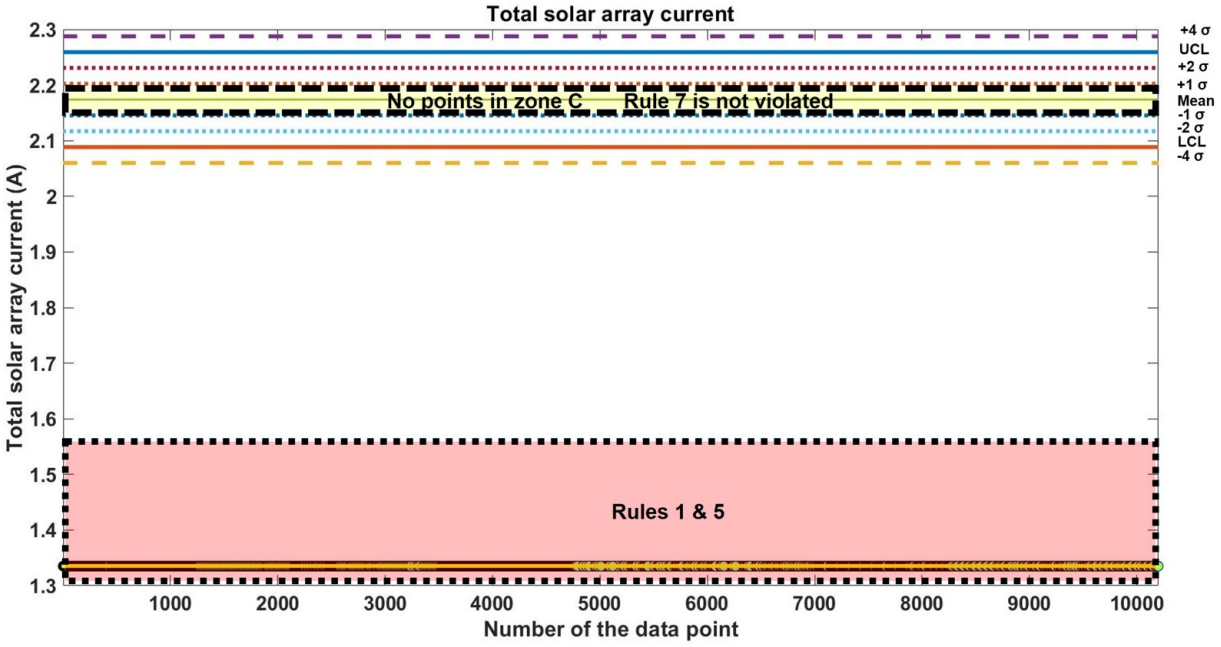

Figure 5.13: Control chart for the total solar array current with two failed strings

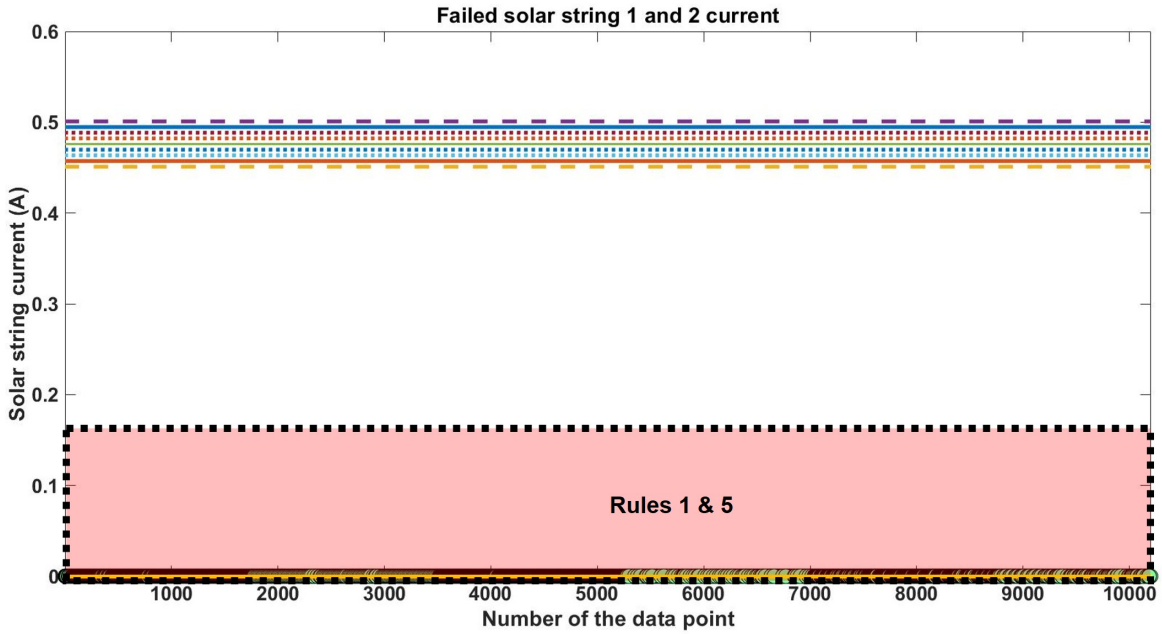

Figure 5.14: Control chart for the solar strings 1 and 2 that are generating no current

- **Sudden failure of a string**:

  The current from string 1 is made equal to 0 in the simulator midway through the simulation time causing it to fail. The observations made are in Table 5.6. Due to a sudden failure in string 1, the points after failure occurs violate rule 1 and 5 but do not violate rule 7 (Figures 5.15 and 5.16). The violation of rule 7 in the ISAI - 1 chart is the key difference between the lower efficiency and sudden failure cases for string 1. The number of out of control points is also lower in a sudden failure scenario as the string is functioning normally in the beginning time period. Again, the TSAI chart (Figure 5.15) does indicate some anomaly in the solar strings but does not tell me the specific details of the strings facing any issues.

TSAI	ISAI - 1	ISAI - 2	ISAI - 3	ISAI - 4	ISAI - 5
Rules 1, 2, 5, 6, 7 and 8	Rules 1, 2, 5, 6, 7 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8	Rules 2, 6, 7 and 8

Table 5.6: List of the rules that are violated in the case of a sudden failure in string 1

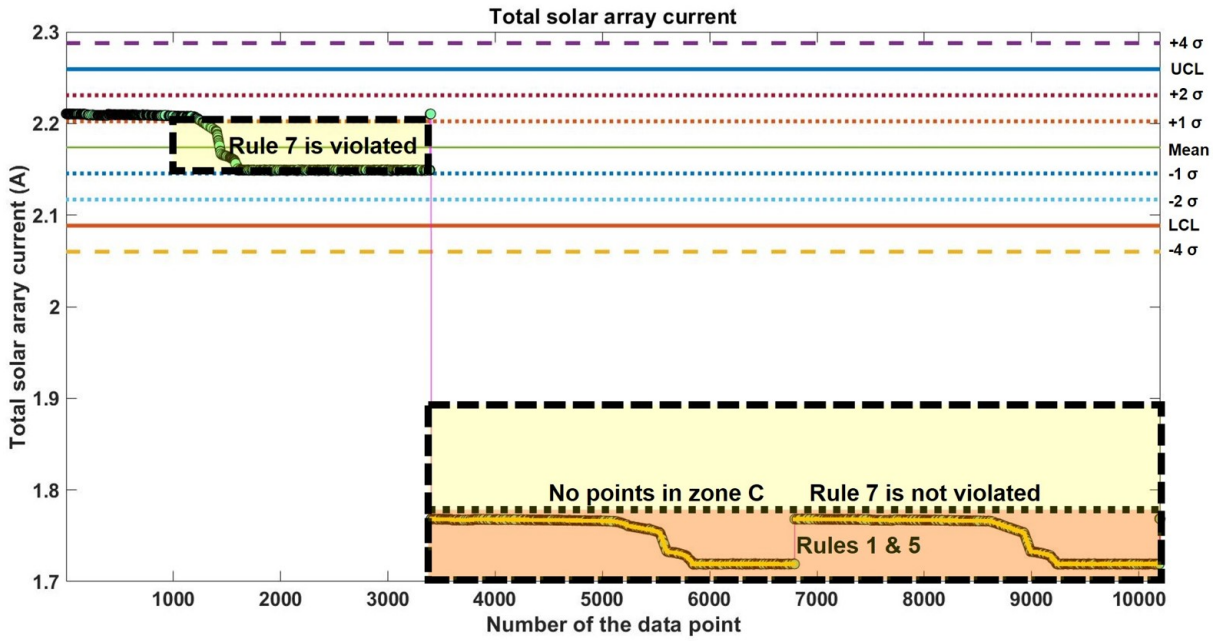

Figure 5.15: Control chart for the total solar array current with a sudden failure in string 1

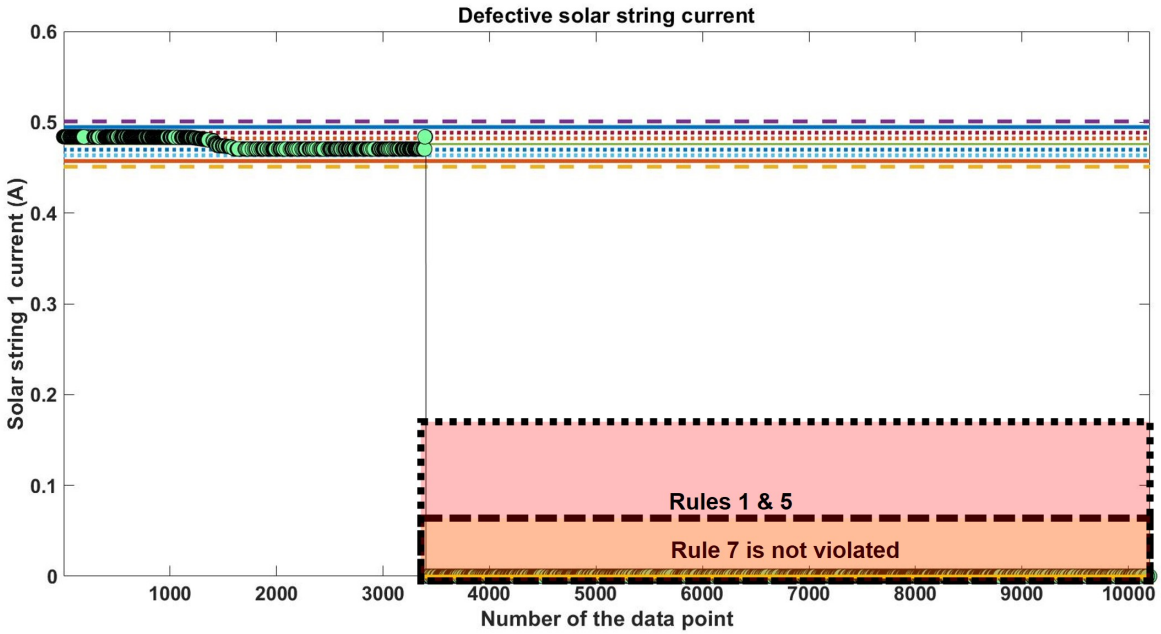

Figure 5.16: Control chart for the solar string 1 with a sudden failure

Algorithm Development The conclusions drawn from the observations made in the previous step are used in developing the algorithms logDetTotal for TSAI and logDetInd for individual string currents. The control charts for all observations were presented for a time period of three orbits. As previously discussed, the solar array current has a period of 1 orbit or 5560 seconds. The algorithms should work for any number of data points provided I have at least one orbit of data. This is because the control limits stay constant for any number of points and the observations have been made over at least one orbit period.

In Figures 5.17 and 5.18, the variable score indicates the severity of the anomaly (in terms of losing solar strings) that is detected by the algorithms logDetTotal and logDetInd. This score becomes important when I design the fault-management system's implementation in the VGS which will be discussed in later chapters. The higher the score, the higher is the severity of the fault.

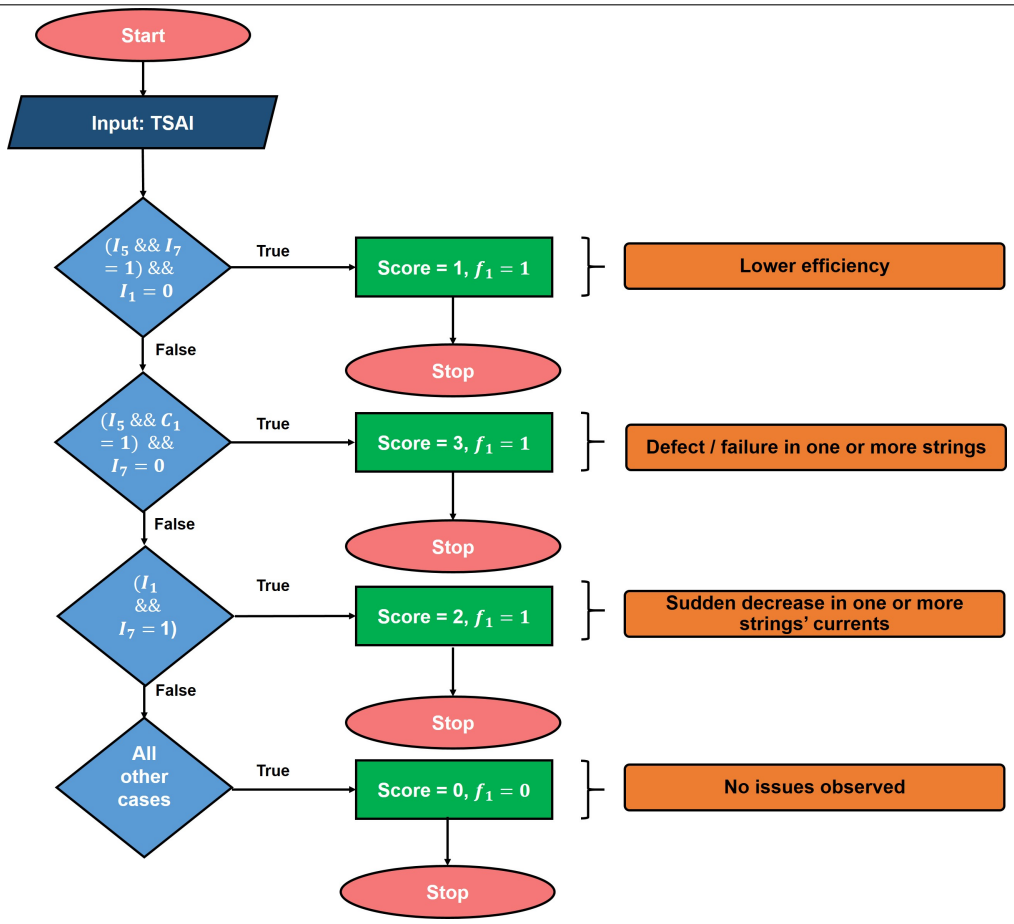

Figure 5.17: The flowchart for the logDetTotal algorithm used to detect faults in the total solar array current. Here, $I_{Rulenumber}$ stands for a particular rule being violated where $I_{Rulenumber} = 1$ means the rule is violated and $I_{Rulenumber} = 0$ indicates no violation; C_1 is the condition that the number of points violating control rule 1 are greater than 75 % of total number of points for TSAI. C_1 takes the value 1 when it is satisfied and 0 when it is false. The symbol && refers to the AND (requires all sub-conditions in a condition to be simultaneously true to be satisfied) operator in a condition. As an example, the first condition would read as $I_5 = 1$ AND $I_7 = 1$ AND $I_1 = 0$. Note that the oval-rectangular shapes towards the right for every condition are not part of the algorithm. They state the type of solar string fault.

Algorithm logDetTotal has three main conditions in which the combinations of different control rules are used. To identify if a control rule is violated, the crAlg N algorithms discussed earlier are used. To begin, I first run all the crAlg N algorithms on the TSAI data. I introduce a variable called $I_{Rulenumber}$ which is 0 when the corresponding rule is passed and is 1 when it is failed. The $I_{Rulenumber}$ values for all rules are computed. A Condition C_1 is defined which is an "if statement" that checks if the number of points in the TSAI dataset violating control rule 1 are greater than 75 % of the total number of points in the TSAI dataset. C_1 has a value of 1 when satisfied and 0 when false. Each of the main conditions is defined using a (AND logic) combination of prescribed values for C_1 and $I_{Rulenumber}$. Based on logDetTotal given in Figure 5.17, the newly introduced variable Score (same as score) is assigned a value. Score gets a value of 3, 2, 1 or 0 for the three main conditions and the last (all other cases) condition respectively. The value of score indicates the severity of the anomaly (in terms of losing solar strings) that is detected by the algorithm logDetTotal. This score becomes important when I design the fault-management system's implementation in the VGS which will be discussed in later chapters. The higher the score, the higher is the severity of the fault.

Also, the value of f_1 is stored where f_1 (discussed in the next and last step) determines if the algorithm logDetInd has to be run or not. If none of the conditions that provide a score of > 0 in logDetTotal are true, there is no need to run logDetInd and the solar array SPC process will end. In other words, any condition/situation not identified by the three main conditions is sent to the last condition, that is, the all other cases condition in which no fault is detected. logDetTotal provides two outputs, f_1 and $M_{1,TSAI}$ that is the number of points in TSAI violating control rule 1. Also, once a condition is satisfied, the algorithm is exited immediately.

If there is a need to run logDetInd based on the value of f_1 from logDetTotal, SPC is run on each and every individual solar string current data. In such a situation

for ManitobaSat-1, the SPC is run for each of the five solar strings.

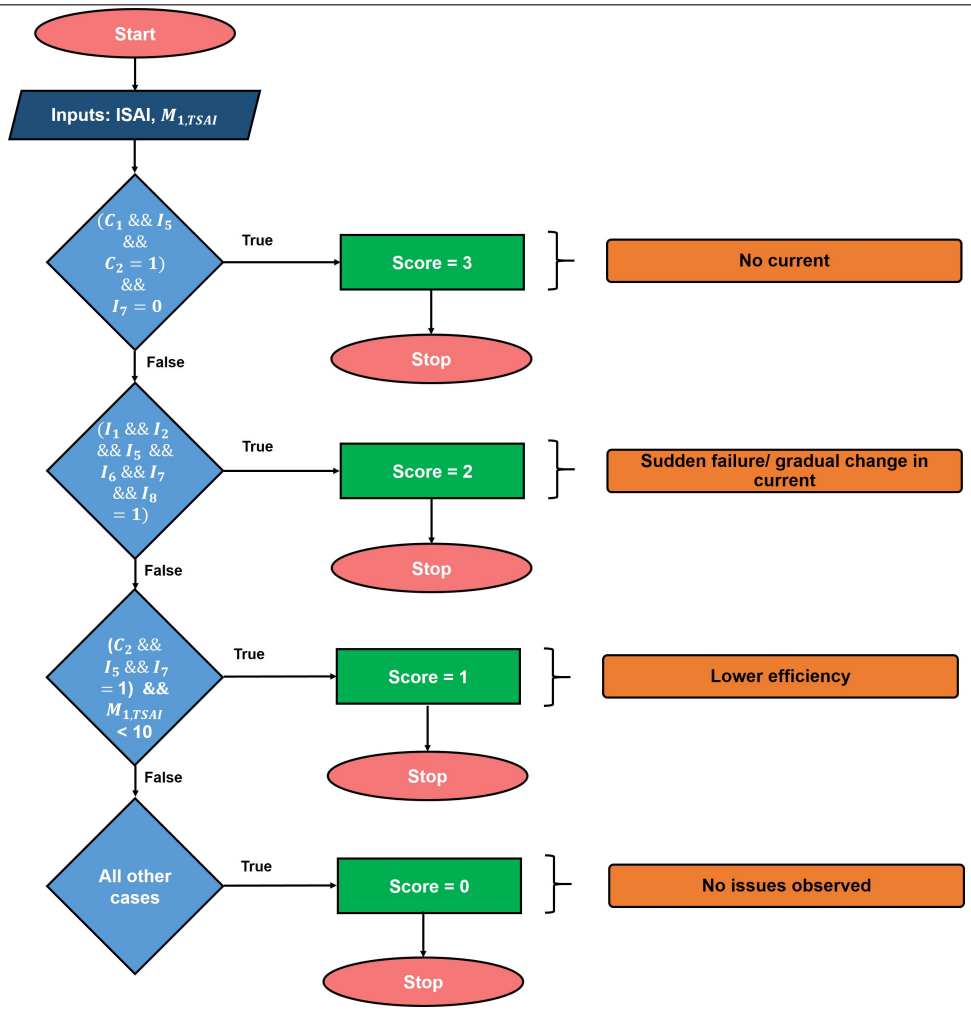

Figure 5.18: The flowchart for the logDetInd algorithm used to detect faults in each solar string current. Here, $I_{Rulenumber}$ stands for a particular rule being violated where $I_{Rulenumber} = 1$ means the rule is violated and $I_{Rulenumber} = 0$ indicates no violation; C_1 is the condition that the number of points violating control rule 1 for TSAI ($M_{1,TSAI}$) are greater than 75 % of total number of points and C_2 has a value of 1 when the number of points violating control rule 1 for ISAI are greater than 75 % of total number of points. C_1 and C_2 take the value 1 they are true and 0 when they are false. The symbol && refers to the AND (requires all sub-conditions in a condition to be simultaneously true to be satisfied) operator in a condition. As an example, the first condition would read as $C_1 = 1$ AND $I_5 = 1$ AND $C_2 = 1$ AND $I_7 = 0$. Note that the oval-rectangular shapes towards the right for every condition are not part of the algorithm. They state the type of solar string fault.

There are 3 main conditions in the logDetInd algorithm. Before the algorithm is checked for any of these conditions, I use the variable $M_{1,TSAI}$ given as an output from logDetTotal as an input to logDetInd. Now, the three main conditions are checked one by one in the order given in Figure 5.18. These conditions contain the variable $I_{Rulenumber}$ and the nested conditions C_1 and C_2. $I_{Rulenumber}$ is a variable that takes the value of 1 when the corresponding rule is failed and 0 otherwise. Conditions C_1 and C_2 are if statements that check if the number of points in the TSAI and ISAI-N datasets violating control rule 1 are greater than 75 % of the total number of points in the corresponding datasets respectively. C_1 and C_2 have a value of 1 when satisfied and 0 when false. Each of the main conditions are defined using a (AND logic) combination of prescribed values for C_1, C_2 and $I_{Rulenumber}$. Based on logDetInd given in Figure 5.18, the newly introduced variable Score (same as score) is assigned a value. Score gets a value of 3, 2, 1 or 0 for the three main conditions and the last (all other cases) condition respectively. The value of Score indicates the severity of the anomaly (in terms of losing the solar string) that is detected by the algorithm logDetInd. This score becomes important when I design the fault-management system's implementation in the VGS which will be discussed in later chapters. The higher the score, the higher is the severity of the fault.

Any condition/situation not identified by the first three main conditions is sent to the last condition, that is, the all other cases condition in which no fault is detected. Also, once a condition is satisfied, the algorithm is exited immediately. Hence, in scenarios where the criteria for multiple cases are satisfied, the algorithm function chooses the condition with the highest score. The corresponding scores for every satisfied main condition are shown in Figure 5.18.

Decision for ISAI-N Execution At this point, I have the algorithms required to detect anomalies in the currents separately for TSAI and ISAI but I am yet to

establish the logic (Figure 5.19) behind the decision to run ISAI based on the outputs from TSAI. When logDetTotal does not detect any issues with the array, that is if the score is 0, it gives a value of 0 to f_1. In other situations, f_1 is 1 and logDetInd has to be triggered for every string. To perform this, I write a script SS where all the previous algorithms are assembled and the solar current SPC framework is executed. Note that the actual fault-detection is related to the score values obtained from the algorithms and will be discussed in later chapters.

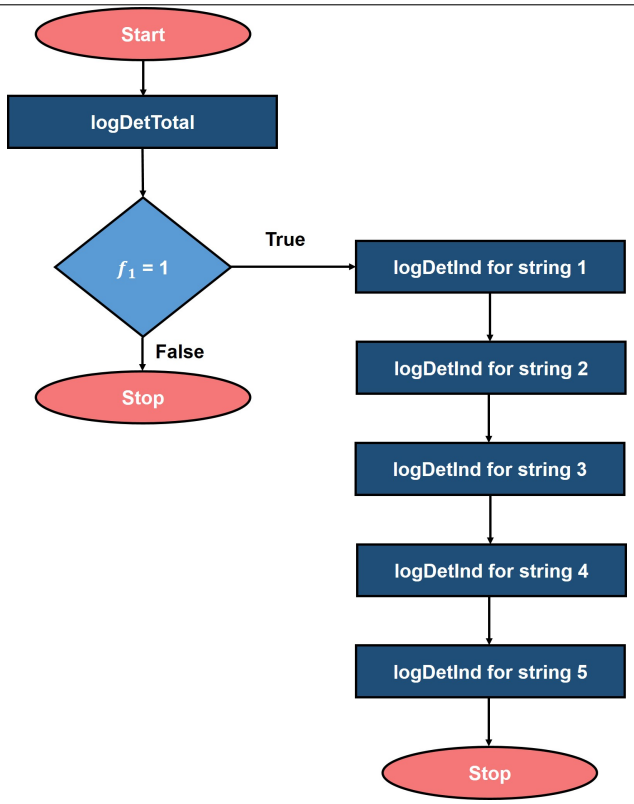

Figure 5.19: The SPC logic for the solar array module used in the script SS

Eclipse handling: As mentioned earlier, only points corresponding to an eclipse flag of value 1 are considered for SPC. SPC does not run if no sun-facing points exist in the dataset.

Limitations

- Though most possible faults are identified by the algorithms, any combination of anomalies other than the ones described above may not be detected by the algorithms.
- The algorithms assume that at least one orbit of data is available during analysis. False alarms might be generated when less than one orbit of data is used at a time.
- Another assumption is that at least one of the strings generates some power during all sun-facing time, that is, at no time do all the strings completely fail together.
- The analysis depends on the eclipse flag for each time point in the telemetry. Any errors in the eclipse flag from telemetry might result in incorrect conclusions.

5.3.5 Increase in the battery's internal resistance

Increase in a battery's internal resistance tends to heat it up often leading to equipment shutdown causing current to be restricted in such cases. The power subsystem could ultimately fail due to this, creating an important need to detect it. To do this using SPC, I first calculate the battery's ideal internal resistance. The specific battery cell (LFP-18650HT) for ManitobaSat-1 with the configuration 2S3P has an average internal resistance of 0.013 Ω [67] as mentioned in Chapter 4.

As with the previous fault of solar string failure, the data from the first few orbits is assumed to be ideal. I need to calculate the control limits and zone boundaries for the SPC analysis. To proceed, it is necessary to formulate the estimation of the internal

resistance (iRes) from the simulator results that will be used for SPC analysis. Once that is decided and the limits are calculated, I start injecting anomalies and making observations from the results of the crAlg N algorithms. Based on these observations, an algorithm is devised to detect an increase in the internal resistance and reduce false alarms. Each of the steps (Figure 5.20) in creating a rule-based SPC system to detect an increase in the battery's internal resistance is discussed below.

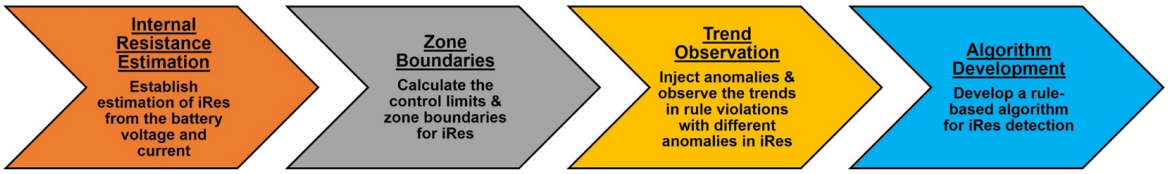

Figure 5.20: Steps in the internal resistance increase detection part of the SPC module

Internal Resistance Estimation This step focuses on estimating iRes from the battery voltage and current data I get from the simulator. From Chapter 3, I have the Equation for the electromotive force (V_{EMF}) of the battery. Assuming that V_{EMF} in Equation 5.1 stays relatively constant, I use the discrete derivative to calculate the k^{th} internal resistance in the sample which is given by the Equation 5.2 where V_{Bus} is the bus voltage, $I_{Battery}$ is the current into the battery or the battery current and R is its internal resistance. The assumption of V_{EMF} being constant is reasonable since in my calculations, the estimation of V_{EMF} is done using the bus voltage and battery current values at consecutive time steps where the time step is one second. That means, I am assuming that V_{EMF} stays relatively constant when moving from one second to the next. As I observed that the bus voltage and the battery current do not change significantly (differences in V_{Bus} and $I_{Battery}$ between two seconds are $<$ 0.005 in magnitude) from one time step to the next, assuming that V_{EMF} is constant between two steps while estimating the internal resistance is reasonable.

$$V_{EMF} = V_{Bus} - I_{Battery}R \tag{5.1}$$

$$R_k = \frac{(V_{Bus,k} - V_{Bus,k+1})}{(I_{Battery,k+1} - I_{Battery,k})} \tag{5.2}$$

Since, the current into the battery does not change significantly between two consecutive time steps most of the time, I get R(t) incorrectly equal to infinity at certain times. This indicates that the assumption of a constant V_{EMF} does not hold true at these points. To get a set of points where the assumption is true, conditions are imposed on the current and voltage values and only these values are considered in the SPC analysis. A minimum difference of 0.01 A and 0.001 V between consecutive measurements for the battery current and voltage respectively are taken. These conditions have been obtained by looking at the battery voltage and current values and their appropriate differences between two time steps to provide a meaningful resistance estimate. Clearly, it takes some time for a certain number of resistance points to be collected for analysis. The pseudocode for the script resistanceEstimation used in the estimation of the internal resistance is given in Algorithm 1.

Algorithm 1: Pseudocode for resistanceEstimation

```
Result: iRes
counter = 1;
for every element in the battery voltage and current datasets of length l from
 i = 1 to l-1 do
    voldiff = (V_Bus(t + 1) - V_Bus(t));
    curdiff = (I_Battery(t + 1) - I_Battery(t));
    if (curdiff > = 0.01 A) and (voldiff > = 0.001 V) then
        iRes(counter) = - voldiff/curdiff;
        timearray(counter) = i;
        counter = counter + 1;
    end
end
if length of iRes > 0 then
    iRes = iRes;
else
    iRes = "N/A";
end
```

SPC for the internal resistance is only run when at least 5 data points are estimated from the telemetry at which the constant V_{EMF} assumption holds true. 5 is the minimum number of points required to detect a significant shift in the data according to the control rules. This is because zone A (rule 5) and zone B (rule 6) tests require at least 5 points for their algorithm (crAlg 5 and crAlg 6) execution. The other tests (except rule 1) focus on smaller shifts which require more points. So, 5 is the minimum number to run a meaningful test. The iRes values obtained during the first 100 orbits is shown in Figure 5.21.

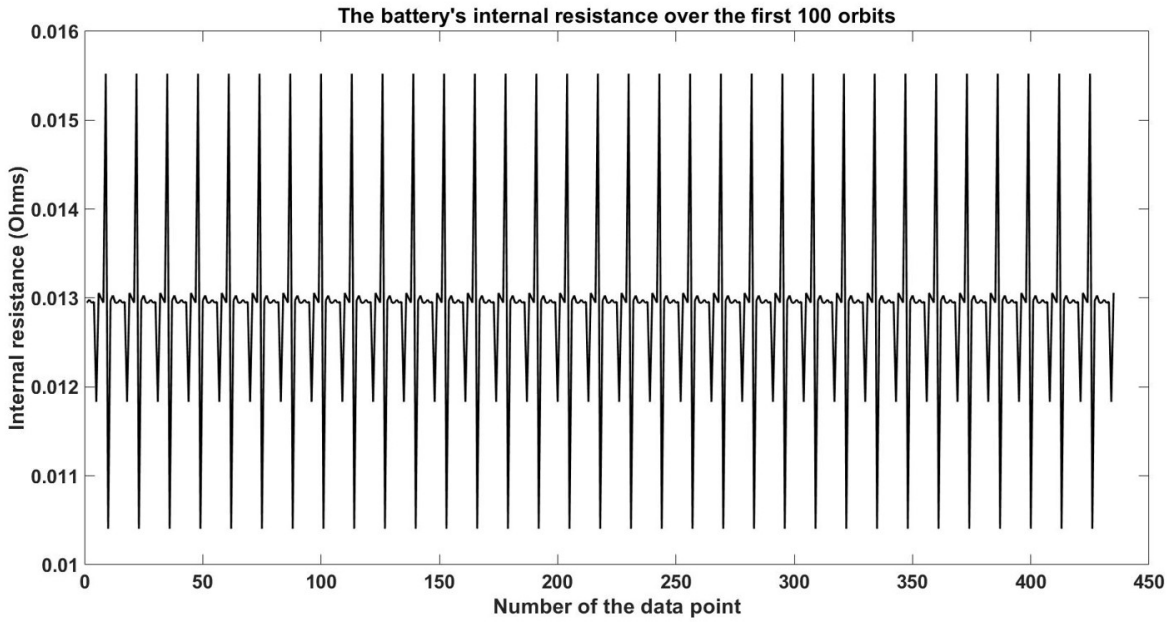

Figure 5.21: The estimated internal resistance of the battery over the first 100 orbits

The resistance values in Figure 5.21 should ideally have formed a straight line but they vary a lot and have a maximum error of approximately 19 %. The error in the estimated values is periodic as can be seen in the figure and seems to be the highest when a major change in the power consumption is taking place which in turn affects the battery voltage and battery current. There are some implications of the internal resistance estimate being potentially wrong by about 19 %. This error increases chances of false alarms and decreases the effectiveness of the SPC techniques for anomaly detection. One way to slightly alleviate this effect would be to consider appropriate margins and wide ranges in the categories of faults when the detection algorithm for the increase in the internal resistance is developed. For example, a moderate increase in the internal resistance which is a category of the fault considered for detection can be between 0.018 Ω and 0.027 Ω. This provides a wide enough range such that when there is an increase to an estimated 0.023 Ω or a maximum resistance value of 0.027 Ω accounting for error. The value 0.027 Ω is still in the range of the same category of the fault. It might seem that 0.023 Ω is a comfortable choice for this example instead of a number closer to the boundary of the category such as 0.019 Ω where the minimum resistance value after accounting for the overestimation error is 0.015 Ω which moves into the previous category of a lower resistance increase. But, this leads to the next step of alleviation I incorporate in the algorithm development, that is, detecting even a change of 0.002 Ω from the ideal value. Doing this ensures that the algorithm detects most cases of change. Nevertheless, this also means that the false alarm rate will be increased. It is a trade-off between wanting to detect a majority of faulty cases including some false positives against not being able to detect some significant increases. Clearly, the former is the better choice. It is better to have some false alarms than having no alarms when significant increases occur.

Zone Boundaries Similar to the solar array current fault, the ideal control chart is developed before formulating an algorithm for fault-detection. Since, iRes is estimated from the battery voltage and current, its value cycles with a period of 3 orbits. This is because the voltage and current follow a 3 orbit cycle. As all voltage and current points would not give me a meaningful resistance and to have a large dataset for calculating the control limits and zone boundaries, I use a sample size of 99 orbits. Again, 99 is a multiple of 3 and the calculations should not be affected by expanding the window. This should not be confused with the minimum number of points required to run the SPC algorithm for this fault which is 5. To reiterate, I require only 3 orbits of data with a minimum of 5 points in total in the sample to perform the SPC analysis. 99 orbits of data are taken only for algorithm development and are not necessary when running the SPC analysis once the system is developed.

The ideal data is the telemetry I obtain in the first 99 orbits of the mission (after reaching steady state) with no fault injected. The control limits lie three standard deviations away from the mean. The control limits and zone boundaries for iRes are computed according to Equations in Table 5.7.

Parameter	Internal resistance	Equation
Mean	0.013	Arithmetic mean of all elements in the dataset
Upper control limit	0.016	Mean + 3 * σ
Lower control limit	0.010	Mean - 3 * σ
+2 σ	0.015	Mean + 2 * σ
+1 σ	0.014	Mean + 1 * σ
-2 σ	0.011	Mean - 2 * σ
-1 σ	0.012	Mean - 1 * σ
+4 σ	0.017	Mean + 4 * σ
-4 σ	0.009	Mean - 4 * σ

Table 5.7: Control limits for iRes. All values are in Ω, and σ stands for the standard deviation of the dataset. The values +4 σ and -4 σ are shown for reference to visually understand how the points are distributed in the plot. They are referred to as the specification limits.

The control chart is provided in Figure 5.22. I run all the crAlg N algorithms on the resultant iRes (Figure 5.22). Importantly, it is observed that a few consecutive points in iRes are equal. Similar to the case of the solar string currents, the tolerance for differences between the consecutive points is taken to be 0.002 in rules 3 and 4. Reducing false alarms is essential when studying the ideal dataset. Since a slight modification of a control rule results in removing a false alarm as is the case here for rules 3 and 4, I modify these rules to remove their violations. Recall, I modified rule 3 as seven or more consecutive points continuously increasing or decreasing by more than 0.002. Rule 4 is modified as fourteen or more consecutive points alternating up or down with a difference (magnitude) of > 0.002. The modifications to rules 3 and 4 are used in the both the algorithms for the solar string failure and the internal resistance fault. The changes are made to rules 3 and 4 as they are the only rules that are dictated by the differences in consecutive data points and not by the zones the points lie in. This uniqueness makes them more conducive to modifications.

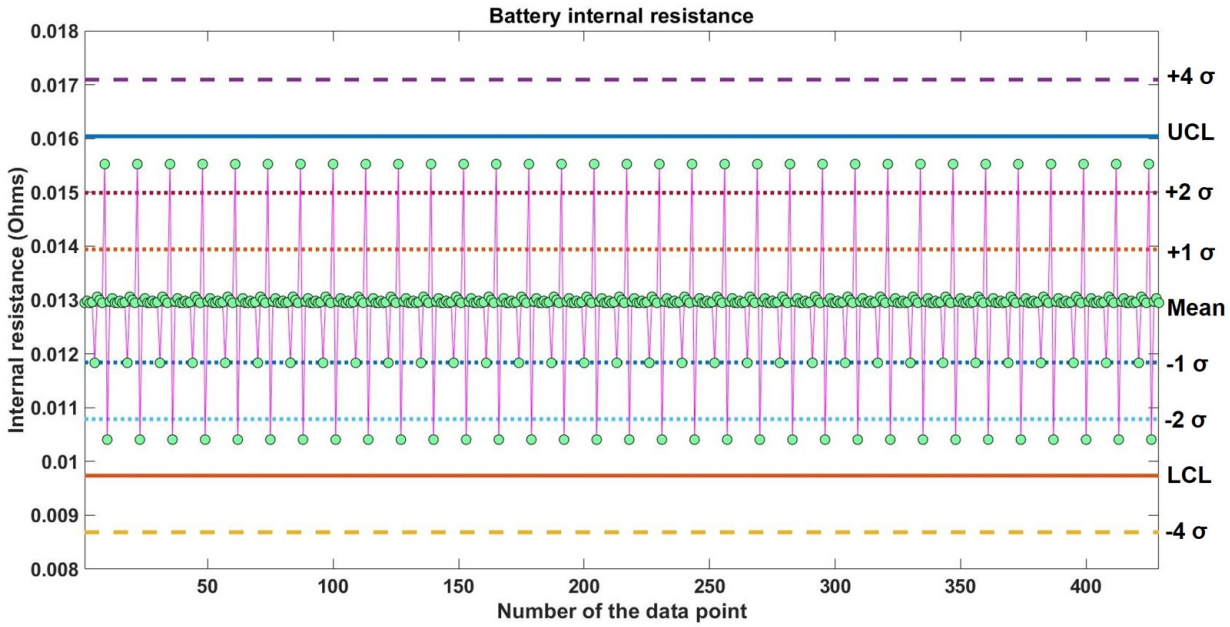

Figure 5.22: Control chart for the internal resistance over 99 orbits used to calculate the control limits and the zone boundaries

This analysis demonstrated that none of the rules are violated by the data which is expected since the data does not contain any faults. Unlike the case of the solar string anomalies, there are no inherent control rule violations in the ideal control chart. The next step is to observe how different types of anomalies in the internal resistance affect the control rules.

Trend Observation In this step, I consider the different ways in which an increase in internal resistance can manifest itself in the data. Each of them is injected and crAlg N algorithms are run on them to make observations. The sample window considered is the same size as the size used for the limits calculations.

The battery cell's datasheet [67] mentions that the maximum expected internal resistance is 0.045 Ω. For a 2S3P configuration for the battery, this is 0.027 Ω. This value is kept in consideration when defining the different categories of resistance increases that are given below.

- **Very high increase in resistance**:

  The resistance is increased to 0.045 Ω in the simulator's internal resistance block, much higher than the 0.027 Ω limit on the datasheet. Though very rare for this increase to occur, this clearly requires detection to protect the battery.

  Rules 1, 2, 5, 6 and 8 are violated by groups of points in the upper portion of the chart as indicated in Figure 5.23. Most of the points are near the 0.045 Ω mark but a few of them lie near the ideal average of 0.013 Ω. Control rule 1 is violated by all points (indicated by small stars in Figure 5.23) that lie near 0.045 Ω as expected. These are indicated by small stars on the points. These points are also on the same side of the mean at several times when the resistance does not steer back to 0.013 Ω within seven points near 0.045 Ω, also violating Rule 2. Rules 5 and 6 are always violated when more than 4 consecutive points lie

beyond the control limits as is the case here. Also, there are many consecutive points that lie far away from the mean violating rule 8. The points that lie below the UCL and are closer in value to the ideal average do not violate any rules. This is easy to detect and is highly deviant from the ideal chart.

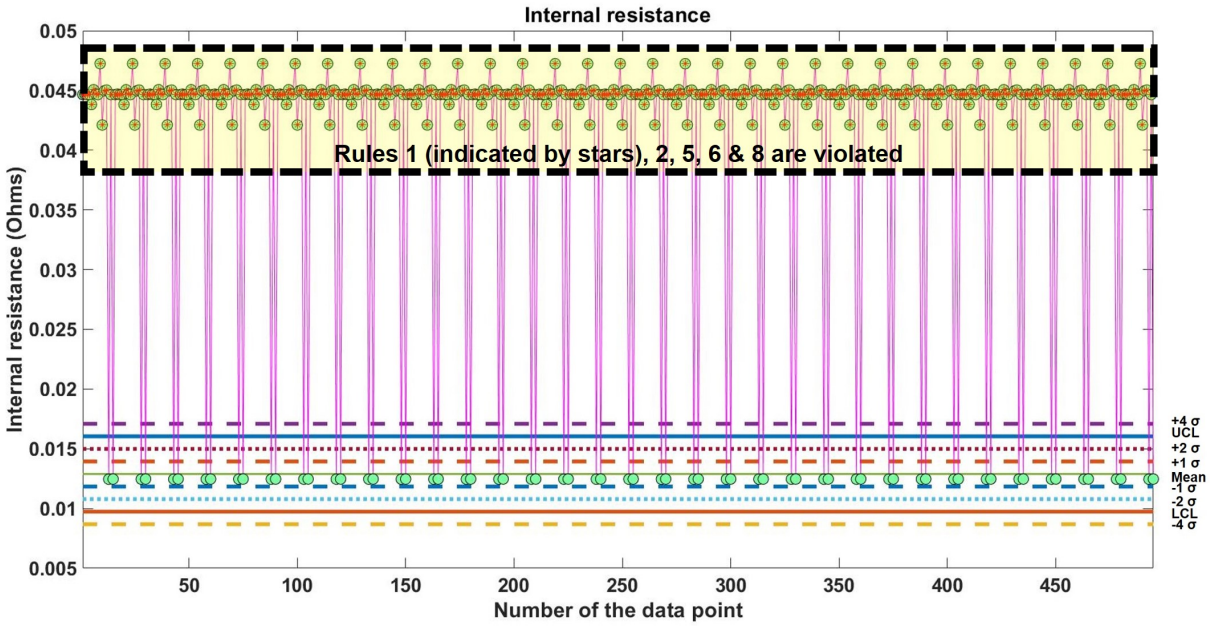

Figure 5.23: Control chart for the internal resistance with an increase to 0.045 Ω from 0.013 Ω

- **Moderate increase in resistance**:

  The resistance is increased to 0.020 Ω in the simulator. Rules 1, 2, 5, 6 and 8 are violated (Figure 5.24). No points go below 0.018 Ω and most lie near 0.020 Ω. Unlike the case of an extreme increase in iRes, all the points violate rule 1. This is a difference between an extreme increase in the first case and the moderate increase to 0.020 Ω.

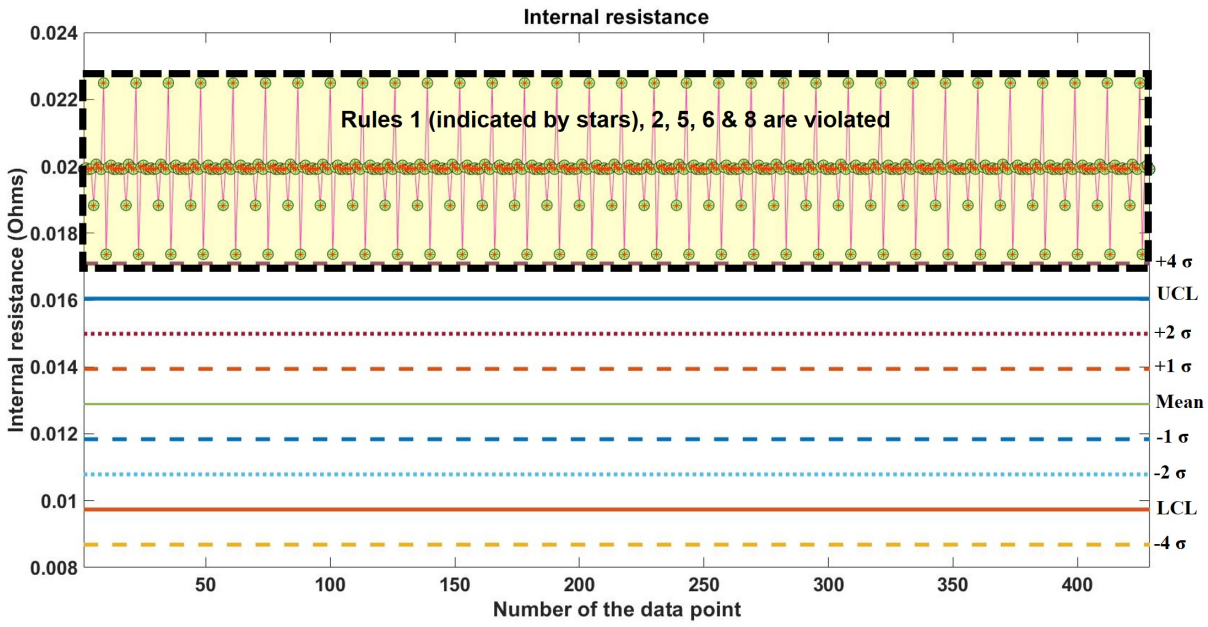

Figure 5.24: Control chart for the internal resistance with an increase to 0.020 Ω from 0.013 Ω

- **A small increase in resistance**:

  The resistance is changed to 0.016 Ω. Again, rules 1, 2, 5, 6 and 8 are violated (Figures 5.25 and 5.26). Note that there are two figures showing the same chart for clarity, the first one indicates the points violating rules 2 and 5 and the second one specifies the violations of rules 1, 6 and 8. Its difficult to differentiate this case from the previous two cases of extreme and moderate increases since the first two cases also showed violations of the same rules, but it is important to observe the control chart and identify differences. In contrast with the previous case, many more points lie near the ideal mean of 0.013 Ω. About 15 % of the points lie above the UCL. Rule 2 is violated because all points are above the mean. Rule 5 is violated by all points violating rule 1 and the remaining points near the UCL. Also, in a row of any 5 consecutive points, at least 4 points lie in zone B or beyond, thus, causing them to fail rule 6. Lastly, there are a few points right above the mean and these do not break rule 8 since they lie in zone C. There are many groups of 8 points in a row that lie beyond zone C. The union (commonality) between the points violating multiple rules becomes an important feature in identifying this type of resistance increase anomaly.

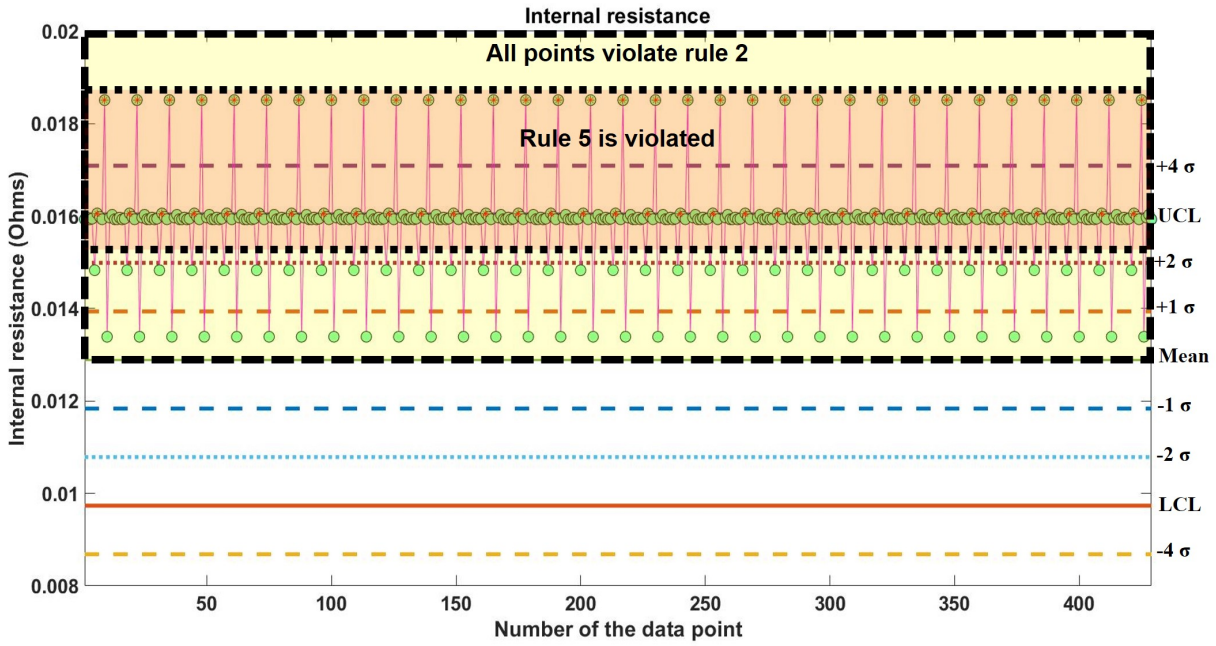

Figure 5.25: Control chart for the internal resistance with an increase to 0.016 Ω from 0.013 Ω. The figure indicates the points violating rules 2 and 5. Also shown are the points violating rule 1 with small stars.

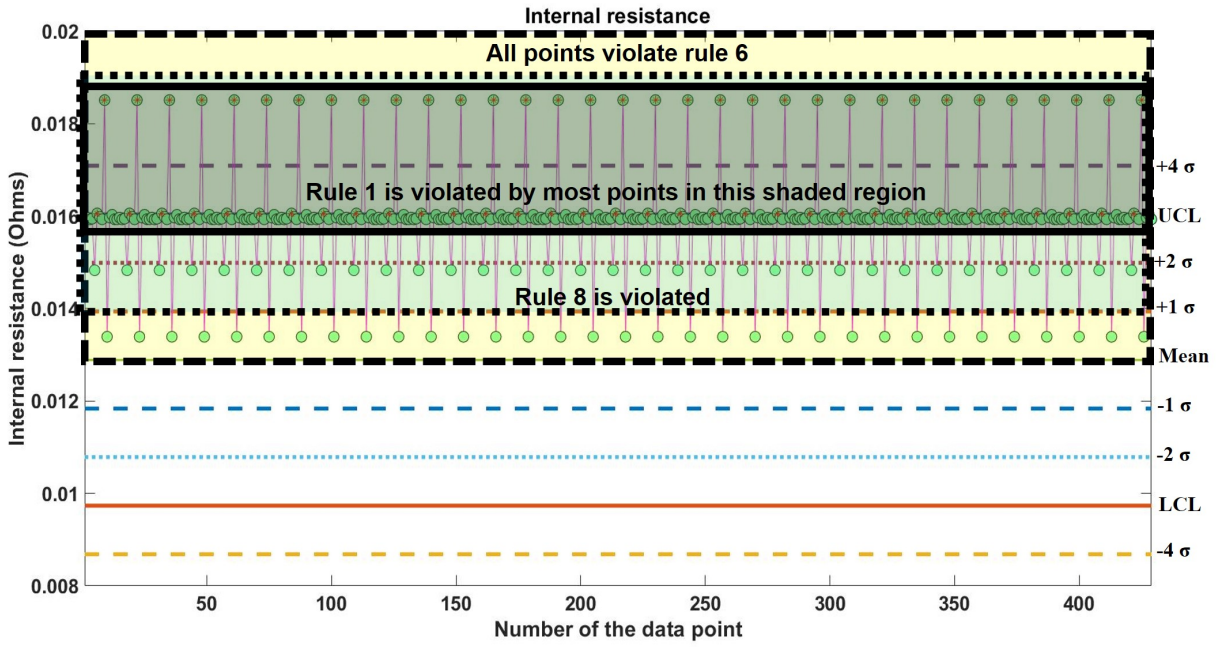

Figure 5.26: Control chart for the internal resistance with an increase to 0.016 Ω from 0.013 Ω. The figure indicates the points violating rules 1 (stars), 6 and 8.

- **Negligible increase in resistance**:

  The resistance is increased to 0.014 Ω which is a 0.001 Ω increase from the ideal mean. Figure 5.27 shows that an unexpected 8% of the points lie beyond the UCL and violate rule 1. Nonetheless, these lie below the upper specification limit indicating their proximity with the UCL. Unlike the previous case of a small increase, only three rules are violated here, namely, rules 1, 6 and 8. Rule 6 is failed by all points even though there are some points lying close (within 1 σ) to the mean. Even then, these points are spaced away from each other such that there exist at least 4 points in every 5 consecutive points that lie in zone B or beyond. Most points except the ones right below the mean break rule 8.

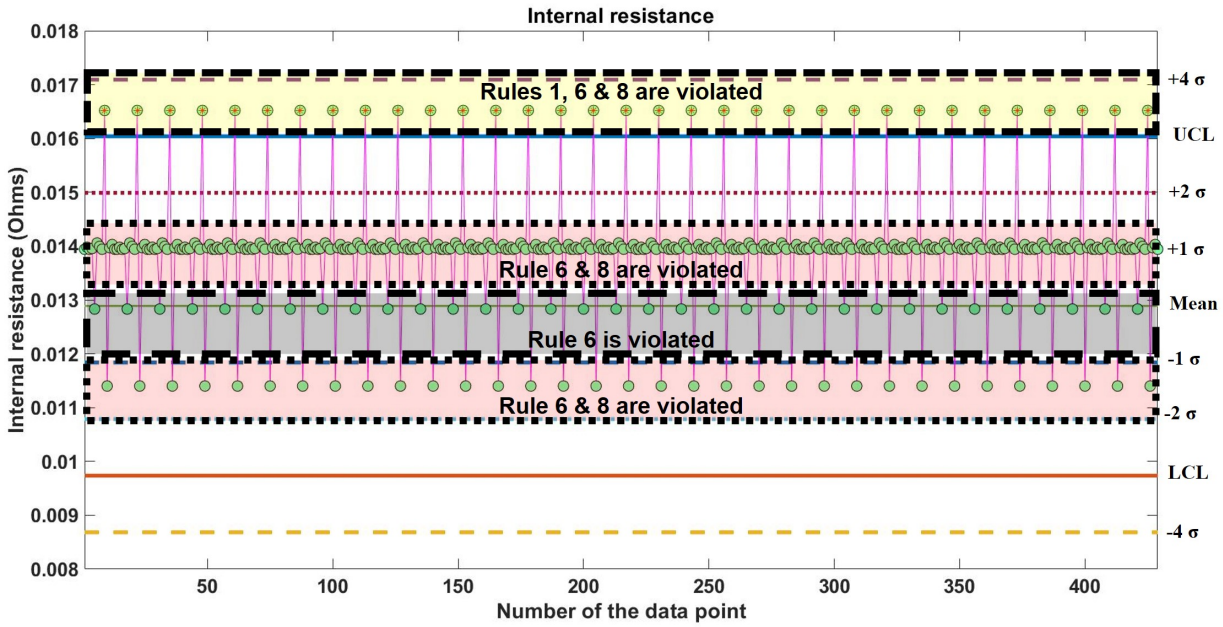

Figure 5.27: Control chart for the internal resistance with an increase to 0.014 Ω from 0.013 Ω

- **Sample size** The sample size of the data has little to no effect on the trends

observed in the control charts for various magnitudes of increase in the internal resistance as long as it satisfies the two conditions of a minimum sample size of 5 points and a sample time of three orbits. This is because the points repeat over a period of time and the resistance points collected from any 3 orbits are the same. To demonstrate that the observations made are the same for a sample time of 99 orbits vs 6 orbits as an example, the plot for the case of a 0.014 Ω resistance over a period of 6 orbits is shown in Figure 5.28. Again, the rules 1, 6 and 8 are breached as I observed in the negligible resistance increase case with a sample time of 99 orbits discussed above. Out of the 26 points, 2 lie above the UCL. This is close to 8 % which is the number I received for a period of 99 orbits. Similarly, by visually inspecting the chart and comparing it to Figure 5.27, it is easy to infer that the distribution proportions of the points over the different zones is almost identical in these figures. Hence, it is clear that the observations made so far do not dramatically change with the sample window.

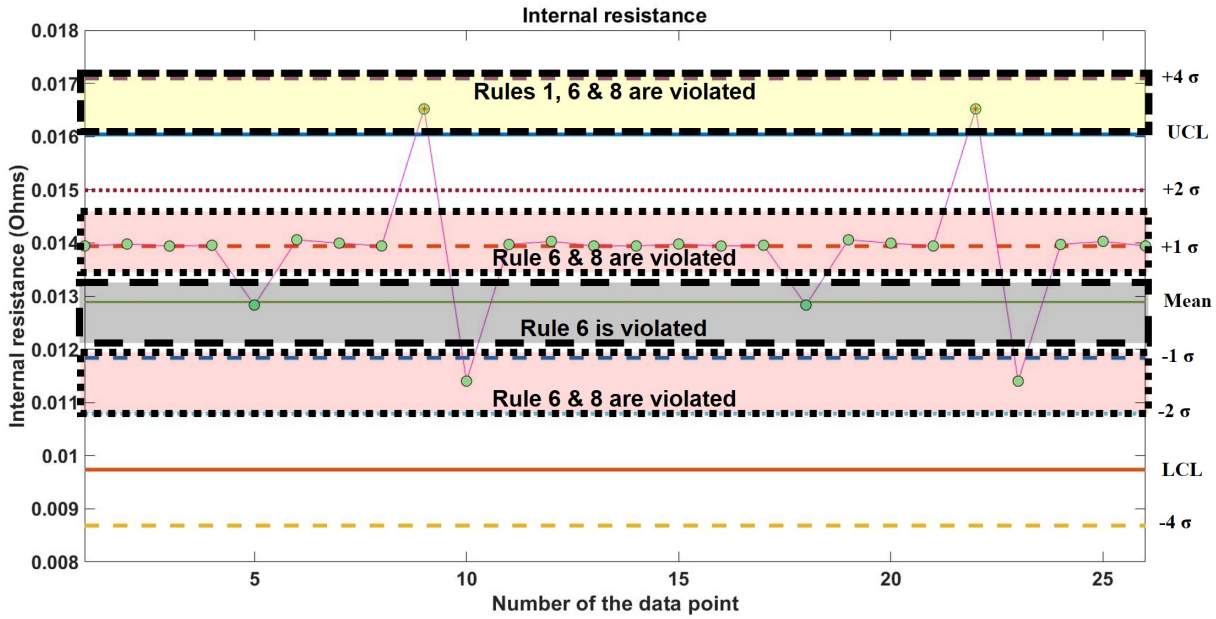

Figure 5.28: Control chart for the internal resistance with an increase to 0.014 Ω from 0.013 Ω over a period of 6 orbits

To summarize, the rules failed by the points in each of the above cases in given in Table 5.8.

Internal resistance (Ω)	Rules violated
0.045 (very high increase)	Rules 1, 2, 5, 6 and 8
0.020 (moderate increase)	Rules 1, 2, 5, 6 and 8
0.016 (small increase)	Rules 1, 2, 5, 6 and 8
0.014 (negligible)	Rules 1, 6 and 8

Table 5.8: List of the rules that are violated for an internal resistance anomaly

Algorithm Development The control charts for various cases of the internal resistance fault were presented in the previous step (Trend Observation). The conclusions drawn from the observations made are used in developing the algorithm logDetRes for iRes anomalies. A flowchart for the algorithm is depicted in Figure 5.29.

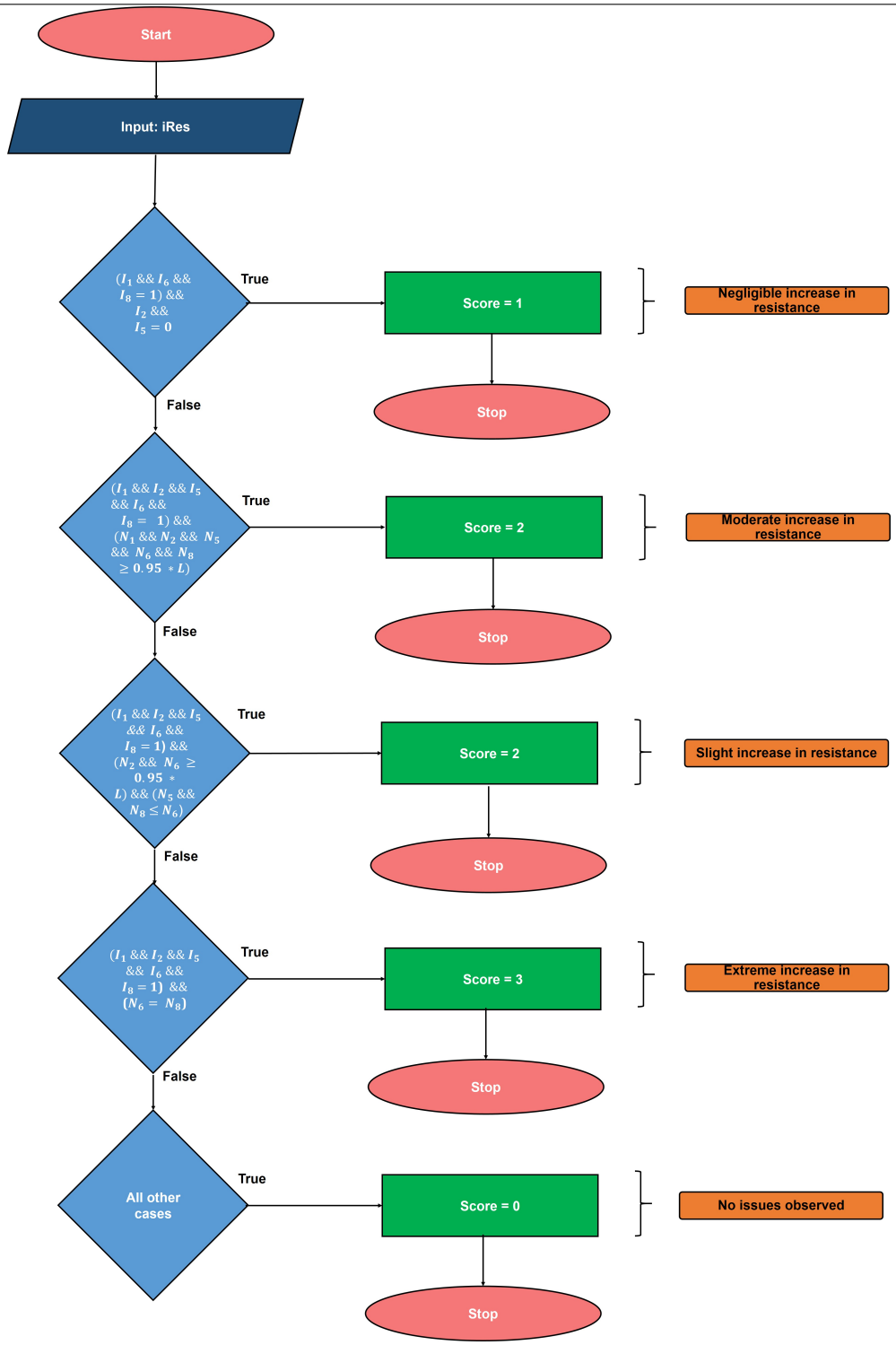

Figure 5.29: The flowchart for the logDetRes algorithm used to detect faults in the internal resistance. Here, $I_{Rulenumber} = 1$ stands for the rule being violated and $I_{Rulenumber} = 0$ indicates no violation; $N_{Rulenumber}$ is the number of points violating the rule and L represents the total number of points in iRes. 0.95 stands for 95 % which is used as a parameter to compare the number of points breaking corresponding rules. The symbol && refers to the AND (requires all sub-conditions in a condition to be simultaneously true to be satisfied) operator in a condition. As an example, the first condition would read as $I_1 = 1$ AND $I_6 = 1$ AND $I_8 = 1$ AND $I_2 = 0$ AND $I_5 = 0$. Note that the oval-rectangular shapes towards the right for every condition are not part of the algorithm. They state the category of the internal resistance fault.

In logDetRes, iRes is given as the input. Here, there are 4 major conditions where combinations of control rules are used to detect the degree of severity of the faults in the internal resistance if at all one exists. To identify if a control rule is violated, the crAlg N algorithms discussed earlier are used. To begin, I need the information of whether the rules are violated and also the number of points violating each rule. The crAlg N algorithms are run on iRes. I introduce the variables $I_{RuleNumber}$, $N_{RuleNumber}$ and L. $I_{RuleNumber}$ defines if a rule is passed or failed and takes the value 1 when a rule is failed and 0 when it is passed. $N_{RuleNumber}$ denotes the number of points violating the corresponding rule and L is the total number of points in the iRes dataset. Each of the main conditions is defined using a (AND logic) combination of prescribed values for $I_{Rulenumber}$, $N_{Rulenumber}$ and L. Based on logDetRes given in Figure 5.29, the newly introduced variable Score (same as score) is assigned a value. Score gets a value of 1, 2, 2, 3 or 0 for the four main conditions and the last (all other cases) condition respectively. The value of score indicates the severity of the anomaly (in terms of battery degradation due to an increase in its internal resistance) that is detected by the algorithm logDetRes. This score becomes important when I design the fault-management system's implementation in the VGS which will be discussed in later chapters. The higher the score, the higher is the severity of the fault.

Limitations

- As with solar array current, though most possible cases of faults are identified by the algorithms, any combination of anomalies other than the ones described above may not be detected by the algorithms.
- A minimum of 5 resistance points and a sample time of three orbits is required for SPC analysis. The collection of the adequate number of points may take time, therefore, SPC on the internal resistance is not always possible.

- The 20 % error in estimating the internal resistance may cause false alarms or no detection of faults in some cases as I discussed previously.

5.4 Development of the Time-Domain Features Extraction Component of the Fault-Management System

5.4.1 Excessive Power Consumption

The faults discussed so far consists of solar array current and battery internal resistance parameters both of which are expected to be relatively constant scalars. Unlike these, the power consumption is defined by the power consumption timeline which has varying values based on the mission operations. Different failures affect a process in unique ways and using control charts might not be applicable to every failure. Subtle changes in complex waveforms such as the power consumption are difficult to interpret for early signs of faults using traditional SPC. As described in chapter 2, time-domain features can be used to represent the characteristics of the system, whereby a customized algorithm can perform classification to identify anomalies.

To select the appropriate time-domain features, the relative influence and trends in the different features (with and without faults) are studied over a period of 99 (multiple of 3) orbits since the power consumption timeline (Figure 5.30) repeats after every three orbits for ManitobaSat-1. Note that the analysis for fault-detection requires only a minimum data of 3 orbits. I am using the data from 99 orbits only for clarity and algorithm development. The overall process is depicted in Figure 5.31.

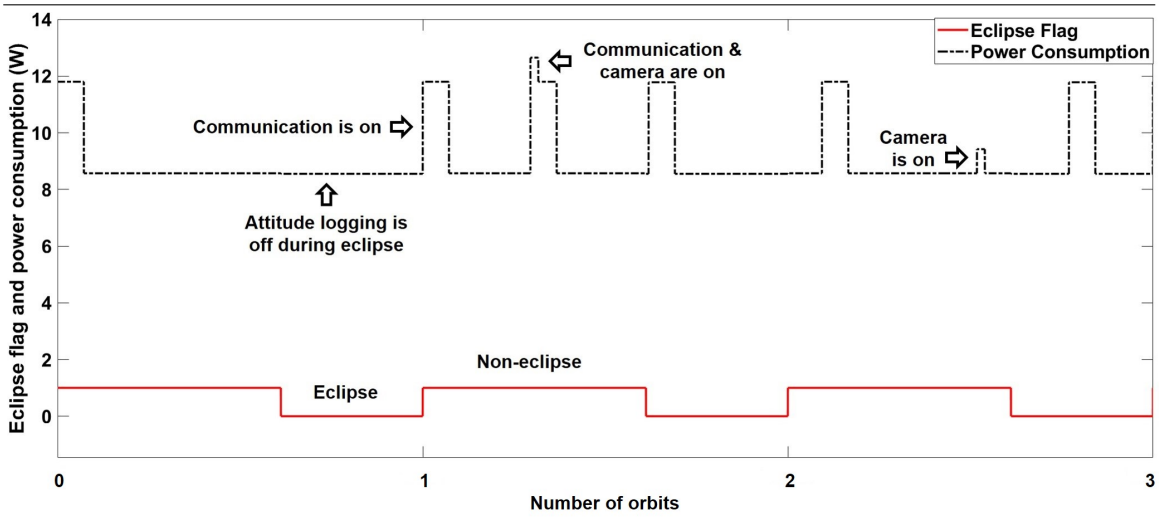

Figure 5.30: Power consumption timeline for ManitobaSat-1 over three orbits

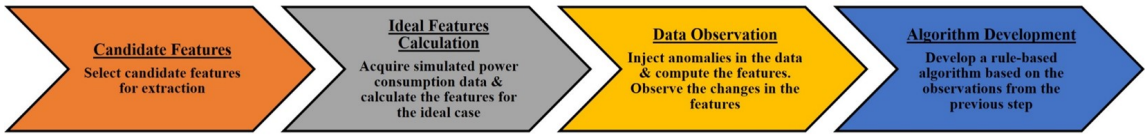

Figure 5.31: Steps in detecting excessive power consumption

Candidate Features I list time-domain features that are candidates in the selection process in Figure 5.32 and provide mathematical descriptions for them in Equations 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11 and 5.12 [50]. In these equations, x_i refers to the i^{th} value in the dataset and n is the number of points in the dataset.

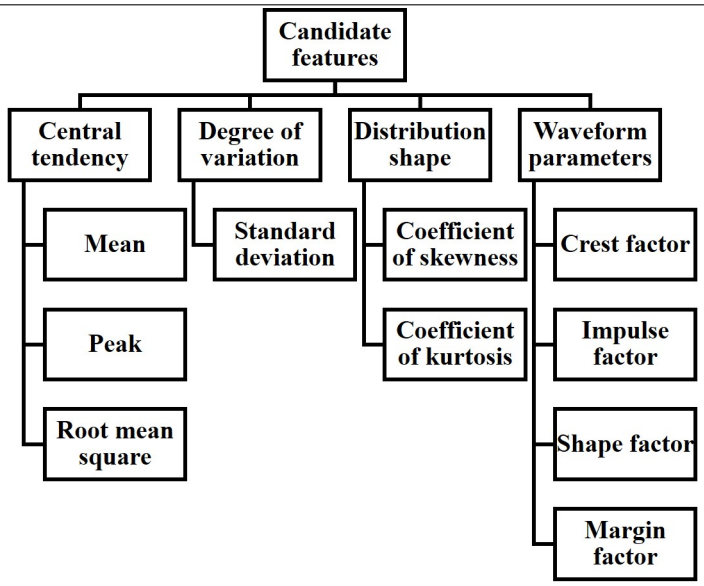

Figure 5.32: List of candidate time-domain features for observation [55] [78]

$$\text{Mean}(\bar{x}) = \frac{\sum_{i=1}^{n} x_i}{n} \tag{5.3}$$

$$\text{Peak}(x_p) = max(x_i) \tag{5.4}$$

$$\text{Root Mean Square}(x_{rms}) = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}} \tag{5.5}$$

$$\text{Standard Deviation}(x_{std}) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{5.6}$$

$$\text{Skewness}(x_{skew}) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{(n-1)x_{std}^3} \tag{5.7}$$

$$\text{Kurtosis}(x_{kur}) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^4}{(n-1)x_{std}^4} \tag{5.8}$$

$$\text{Crest Factor}(CF) = \frac{x_p}{x_{rms}} \tag{5.9}$$

$$\text{Shape Factor}(SF) = \frac{x_{rms}}{\bar{x}} \tag{5.10}$$

$$\text{Impulse Factor}(IF) = \frac{x_p}{\bar{x}} \tag{5.11}$$

$$\text{Margin Factor}(CIF) = \frac{x_p}{(\frac{\sum_{i=1}^{n}\sqrt{x_i}}{n})^2} \tag{5.12}$$

Each of the features is investigated for ideal and faulty scenarios in the later steps. The features are briefly described below as described in [78] and [79]:

- **Central tendency**: This measure attempts to provide the location of a central value for an entire dataset. It defines how close the dataset is to its central or typical value.
  - **Mean**: This is the most common measure of central tendency. Median and mode are also measures of central tendency. They are not considered for this application as when random variables assume values from a vector space as for the power consumption, these measures are not as effective as the mean.
  - **Peak or maximum**: The largest value in a dataset, this occurs when communication with the ground and payload imaging are taking place simultaneously. On the other hand, the minimum value in our dataset is

not a very useful feature for our application since I am trying to detect the increase in the power consumption. Nevertheless, the other selected features are likely to detect major decreases in consumption if they occur.

  - **Root mean square (RMS)**: This metric gives an idea of size of the dataset regardless of positive and negative values. Since, all our values are positive, it is expected that the root mean square is close to the mean. It is essentially the square root of the arithmetic mean of the squares in a dataset.

- **Degree of variation**: This provides the statistical measure of the distribution of the data points around the center values. A greater degree of variation indicates that many measured values deviate from the mean in a dataset.

  - **Standard deviation**: Variance is a statistical property of how far each point in a dataset is from the mean also indicating the relative distances between the points in the dataset. Mathematically, the average of the squared differences between the mean and every data point gives the variance. The standard deviation is the arithmetic square root of the variance.

- **Distribution shape**: It is important to understand how the shape of a dataset changes when faults occur. Together with the central values and variations, it defines the basic characteristics of a dataset.

  - **Kurtosis**: This is a measure of flats and spikes in the data.
  - **Skewness**: Provides a measure of the distortion or asymmetry in the distribution based on the notion of moments of the distribution [80]. Faults cause increases in skewness in a distribution.

- **Waveform parameters**: Crest factor, impulse factor, shape factor and margin factor are key properties for a waveform. Shape factor provides insight into

the shape of the distribution irrespective of the signal dimensions. The crest, impulse and margin factors are categorized as impulsive metrics and relate to the distribution peaks. The crest factor serves an early warning sign for faults in many cases as the manifestation of faults often begins with changes in the peakiness of a signal (with respect to the root mean square) which might not be directly detectable from the peak alone. Margin factor is also called clearance factor and has a high value for ideal (no fault) systems [81].

Ideal Features Calculation As the value of these features are the same for any three orbit period, the features are computed for three orbits as a baseline of ideal values (Table 5.9). The equations (Equations 5.3 to 5.12) given for each feature are used for the calculations.

Feature	Ideal value (units)
Mean	9.034 W
Root mean square	9.107 W
Peak	12.650 W
Standard deviation	1.152 W
Kurtosis	5.231
Skewness	2.046
Shape factor	1.008
Crest factor	1.389
Impulse factor	1.400
Margin factor	9.001

Table 5.9: Time-domain features over three orbits with no faults

Data Observation To introduce different faults in the simulator and investigate the features, each of the three unique orbits in the power consumption timeline are labelled as A, B and C as shown in Figure 5.33. A and C have similar characteristics while B shows the peak with the camera being on for two minutes. It is assumed that faults can take place in different forms: only in one of the orbits, in any two of the orbits and all three orbits. Different cases of faults are considered such as low power increase over a long duration, high power increase over a short duration, etc.

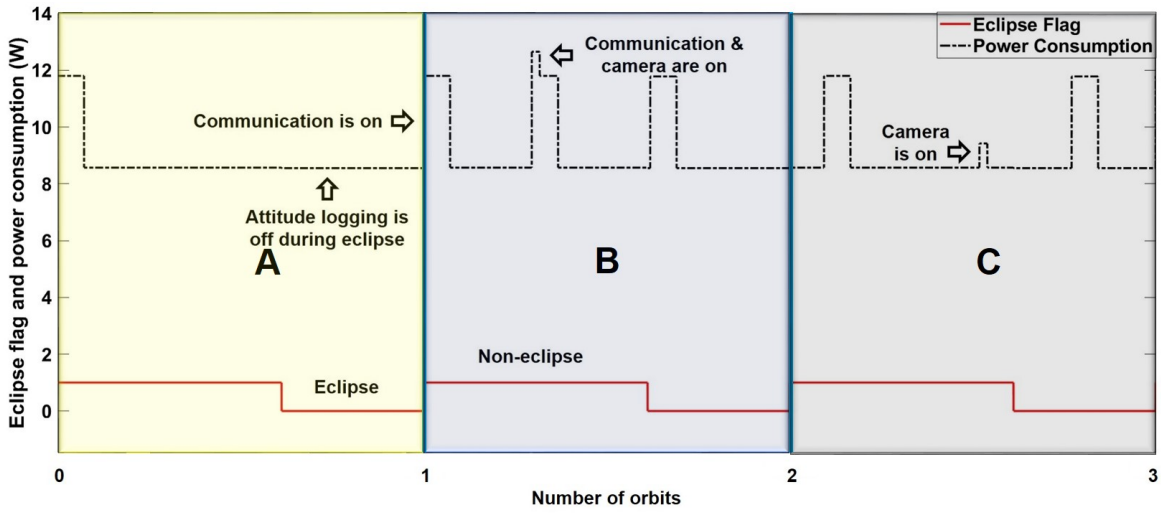

Figure 5.33: The power consumption timeline showing the labels for the three orbits

More than 50 fault test cases are introduced into the data for testing and comparing the trends in the features between the faulty and ideal cases. All computations are made for groups of 3 orbits for ManitobaSat-1. Since the fault-management system should be applicable to any CubeSat and not restricted to ManitobaSat-1, I took measures to ensure that these specific power consumption timeline characteristics do not influence the system I am trying to develop. Therefore, the script (TDFeaturesCompute) written to complete the computations should not always be performed in groups of 3 orbits. This number should be editable based on the period of the power con-

sumption timeline for the specific mission. This idea will be clearer when I discuss the application of the time-domain features part of the fault-management system to a hypothetical mission in a later section (Section 5.4.3).

One early observation found that the peak as a basic statistical parameter might indicate changes to the waveform but does not guarantee a fault. Peaks can remain the same for the ideal and faulty case as the increase in power consumption/fault could occur at points other than the peak of the ideal waveforms and not high enough to cause the peak to change. For example, the maximum power consumption occurs between 7201 and 7320 seconds and is 12.65 W. If there is a spike/fault with an increase in the power consumption during a period other than from 7201 to 7320 seconds and the magnitude of the increase is lesser than the sum of the ideal power consumption and the spike (increase), the peak will remain unchanged. To demonstrate, if the increase is 2 W between 6500 to 7000 seconds where the ideal consumption is 8.57 W, with the fault it changes to 10.57 W which is still lesser than 12.65, the peak of the waveform remains at 12.65 W and does not change from an ideal case despite having a fault in it. Change in peaks can be used to identify huge increases but low and moderate increases can not be conclusively dependent on the peaks.

A script dTDpercent is written to calculate the percentage change in the features in a faulty case from the ideal values. Percent changes in the features for some major fault test cases from ideal cases is given in Figure 5.34. The fault duration and increase in consumption are given in seconds and W respectively and the orbits (labels) where the corresponding faults are injected are indicated.

Case	Orbit	Increase (W)	Duration (s)	Mean (%)	Peak (%)	RMS (%)	STD (%)	Skewness (%)	Kurtosis (%)	CF (%)	IF (%)	SF (%)	CIF(%)
1	A & C	0.5	2000	0.700	0.00	0.700	1.80	-2.50	-1.60	-0.700	-0.600	0.020	0.600
2	B				3.90		4.60	-0.400	2.40	3.20	3.30	0.060	0.640
3	A, B & C	0.5	20	0.007	0.00	0.009	0.140	0.040	0.100	0.009	0.007	0.002	0.007
4	A & C	1.5	2000	2.00	5.14	2.14	10.8	-12.8	-7.40	2.90	3.10	0.140	1.90
5	B				11.8	2.27	18.3	-3.40	8.10	9.40	9.70	0.300	1.90
6	A, B & C	1.5	20	0.020	5.10	0.030	0.500	0.500	1.40	5.10	5.10	0.008	0.020
7	A, B & C	3	2000	4.00	17.0	4.50	34.7	-19.6	-9.17	11.9	12.5	0.540	3.70
8	B				23.7	4.80	47.0	-4.10	14.5	18.0	18.8	0.790	3.60
9	A, B & C	4	120	0.320	24.9	0.480	9.90	22.4	74.0	24.3	24.5	0.160	0.260
10	A, B & C	6	2000	7.90	35.9	8.90	87.0	-12.3	0.400	24.8	26.9	1.60	6.40

Figure 5.34: Percent changes in time-domain features for major test cases in power consumption

The following observations are made from Figure 5.34:

- There are six categories of power consumption increases considered in the figure, namely, low increase long duration (LL) (cases 1 and 2), low increase short duration (LS) (case 3), moderate increase long duration (ML) (cases 4, 5, 7 and 8), moderate increase short duration (MS) (cases 6), high increase short duration (HS) (case 9) and high increase long duration (HL) (case 10). For the short duration test cases, the increase is applied in any one of orbits without preference unlike for the long duration cases where the increase in A and C are studied separately from an increase in B. This is because the increases applied over a small time period do not really affect the overall consumption significantly irrespective of the orbit in which the increase occurs. However, features are heavily impacted if the increases occur over 2000 seconds.

- LS and LL: For the cases 1 and 3, the peak does not change as expected. Orbit B still contains the peak where the camera and communication are on together. Increase of 0.5 W is not big enough to change the ideal peak. The percent changes in the mean and RMS are approximately the same irrespective of the orbit in which the fault occurs. The standard deviation changes more when the fault is in orbit B. This is true for most other features as well. The features hardly change for case 3 which might make it harder to detect.

- MS and ML: Unlike the previous cases, much larger changes are observed in the standard deviation, skewness and kurtosis. The relative signs of the skewness and kurtosis changes and crest and impulse factors changes provide distinctions between the categories of cases. The changes in the crest and impulse factors are close in magnitude. As the magnitude of the increase (fault) is growing, the change in the margin factor (CIF) is drifting away from the changes in the mean and RMS. The CIF seems to be independent of the orbit in which the

fault is occurring. This is important since I am designing a system that is not just applicable to a single mission and should not be specifically designed for particular timelines. While the skewness decreases in both cases 4 and 5, the kurtosis increases only in case 5.

- HS and HL: There is a dramatic increase in the standard deviations when the increase in high and the duration is long. Most of the features increase except for the skewness which decreases in cases 7, 8 and 10. This makes it challenging to find a pattern/trend in the way the features are reshaping with changing faults. Though signs of the changes are useful, it is the combination of the magnitudes and the relative signs that will have to be used for detection.

Algorithm Development With the observations from the previous step (Data Observation), the algorithm logDetPower is developed and boundaries for differentiating between various cases of faults is determined based on the changes in the selected features. For each category of fault, I use the features that are affected the most for detection. The various conditions in logDetPower are described in Table 5.10 where avg, rms, peak, std, skew, kur, cf, if, sf and cif stand for the percentage changes from an ideal to faulty case in the mean, root mean square, peak, standard deviation, skewness coefficient, kurtosis coefficient, crest, impulse and shape factors and the margin factor respectively.

Condition	Description	Score	Type of Fault Identified
AD	magnitudes of rms and avg > 0.01	N/A	N/A
IA	avg > 7, rms > 9, cif > 6.5, peak > 7 and the maximum/peak has increased by at least 2 and sf > 1	5	High increase long duration
IB	avg > 1, rms > 1.2, cif > 1, std > 15, peak > 8 and the maximum/peak has increased by at least 2	3	High increase short duration
IIA	avg, rms and cif are between 4 and 20, std > 10 and skew and kur are of the same sign	3	Moderate increase long duration
IIB	(avg > 1.2, std > 2, magnitude of kur > 15, cf and if <0) or (avg > 1.2, std > 2, magnitude of kur < 15, cf and if >0)	2	Moderate increase short duration
IIIA	avg, rms and cif are between 2 and 6.5 and the magnitude of skew > 2 and kur > 2	2	Low increase long duration
IIIB	(avg, rms and cif are between 0.3 and 3 and skew and kur are negative) or (avg, rms and cif are between 0.3 and 3, skew and kur are positive and sf > 0.05) or (avg, rms and cif are between 0.3 and 3 and skew and kur are of opposite signs)	2	Moderate or low increase
IIIC	avg > 0.5, rms > 0.5, cif > 0.5, std > 4, kur > 2, cf > 3, if > 3 and sf > 0	2	Uncategorised consumption abnormality
IIID	avg < 0.05, rms < 0.05, cif < 0.05, std < 0.2 and peak > 15	1	Low increase short duration
IVA	avg < 1.5, rms < 1.5, cif < 1.5, skew and kur are negative, the magnitudes of if and cf are within $\pm$ 0.1	1	Low increase short duration
IVB	avg < 1.5, rms < 1.5, cif < 1.5	1	Low increase short duration
All other cases	N/A	0	No issues

Table 5.10: Conditions for the Algorithm logDetPower in Figure 5.35

logDetPower is summarized in Figure 5.35. logDetPower is not just based on the major test cases presented. Instead, the conditions in Table 5.10 are the result of extensive testing to estimate the boundaries between the cases and how the features interact with each other. Once a condition is satisfied, the algorithm is exited immediately. More test cases used to characterize the algorithm are discussed in the next subsection while assessing the accuracy of logDetPower.

In Table 5.10, the variable score stands for the severity of the anomaly (in terms of the power consumption compared to the ideal timeline) that is detected by the algorithm logDetPower. Each value of score corresponds to specific categories of faults (LS, LL, MS, ML, HS and HL). logDetPower checks the conditions and finds out the score. The score corresponds to the VGS's fault-management system lamps such that when the score is 5, the lamp denoting the highest risk is lit and when the score is 2, the lamp for lowest risk is lit and so on and so forth. Hence, the higher the score, the higher is the severity of the fault. I will discuss this more in the next chapter. To summarise, the algorithm only provides results in terms of scores and the assigning of specific categories of faults is only part of the background work of the fault-detection process.

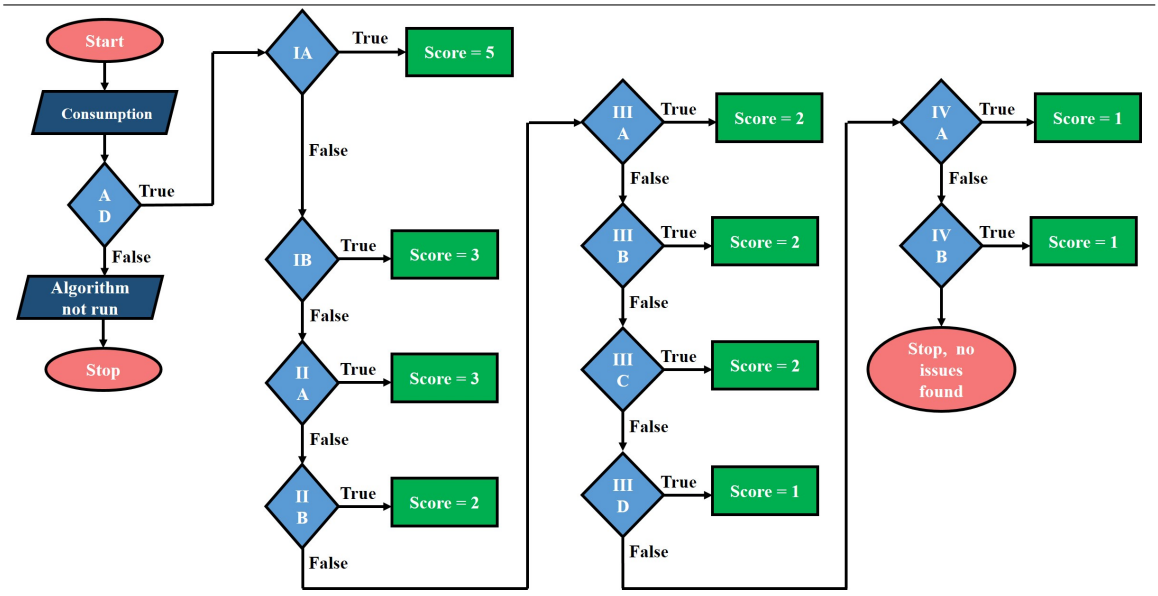

Figure 5.35: The flowchart for the logDetPower algorithm used to detect faults in the power consumption. The conditions used are described in detail in Table 5.10. The faults are detected in terms of the score values as each score corresponds to specific categories of faults as provided in Table 5.10. This figure also shows the preliminary condition AD required to be satisfied to run the algorithm.

Figure 5.35 also indicates that logDetPower is not always run. There is a preliminary condition named AD (Table 5.10) that is based on the sensitivity of detection. It determines if there is any detectable change in the basic features such as the mean and the root mean square. This is done so that the algorithm is not run unnecessarily every time. This reduces execution time.

The algorithm logDetPower can directly be applied to an orbit size of three. I assume that telemetry for power consumption is sent in multiples of 3 orbits. This will be discussed further in Chapter 6. When I have more than one set of three orbit groups to analyse, the scores of each of the groups of 3 are calculated by running logDetPower on every 3 orbit datasets and the maximum score for all the groups is chosen as the final score since higher the score, the more severe the fault is. The logic (script PL) behind this is given in Figure 5.36. Again, the script only deals with scores and has nothing to do with the categories of faults directly. The categories of faults have already been 'converted' to be represented in the form of scores by logDetPower.

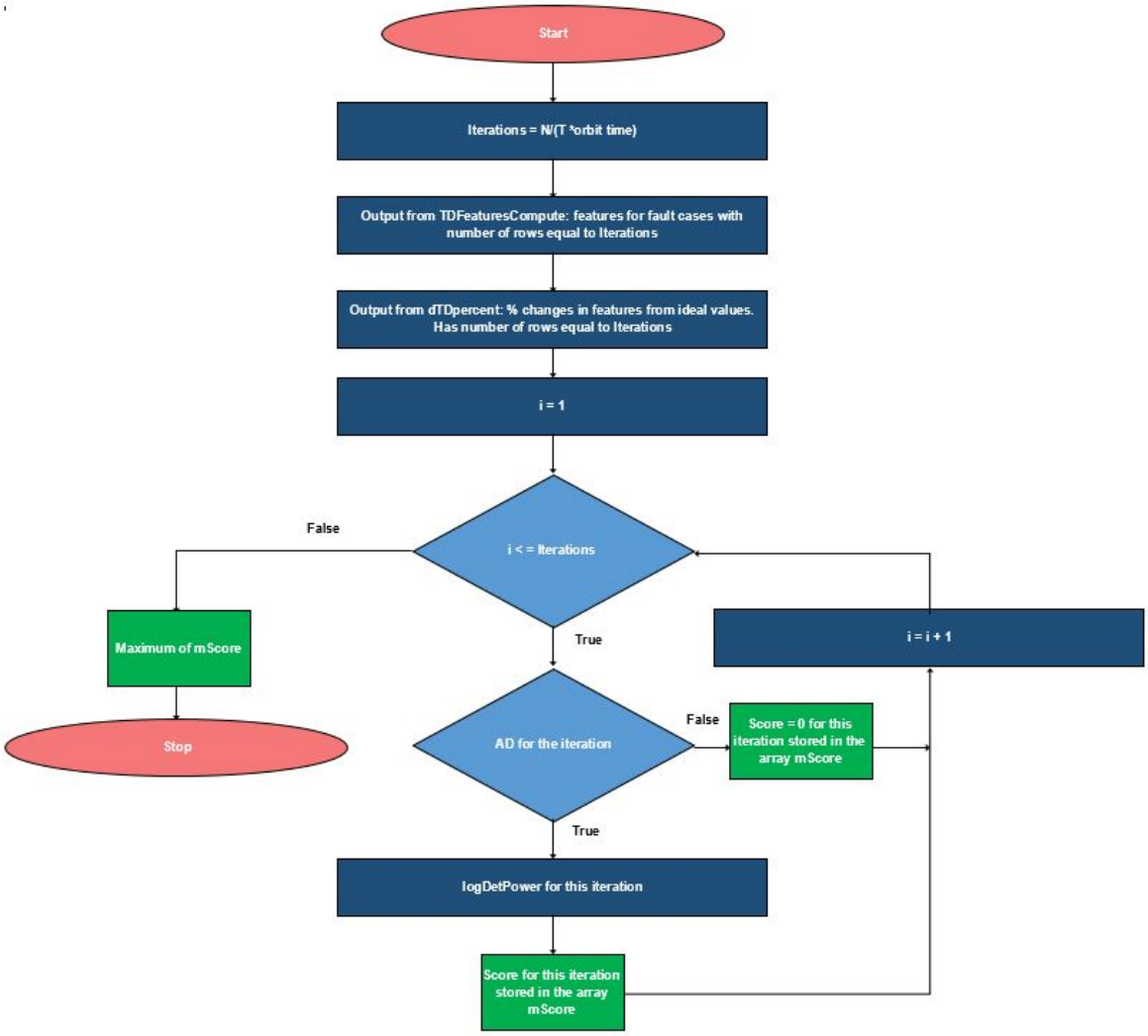

Figure 5.36: The logic for the power consumption fault detection used in script PL

In Figure 5.36, I first compute the number of times (Iterations) the score has to be calculated. This depends on the total number of points in the dataset N, the number of orbits after which the power consumption timeline repeats itself T, which is 3 for ManitobaSat-1 and the orbit time which is 5560 seconds for the mission. I have all the outputs from TDFeaturesCompute and dTDpercent for each Iteration. A for loop is the next step wherein the condition AD is checked and the algorithm logDetPower is executed if necessary. This gives me a score for every group of 3 orbits in the dataset stored in an array mScore with a total number of rows equal to Iterations. Once I am done with all these computations, the maximum of all the elements in mScore is the final output for the whole detection process. This final output is the resultant score for the whole fault-detection process and corresponds to fault-detection lamps in the VGS's fault-management system. I will discuss this further in the next chapter.

As an example, let me consider that the fault-detection logic is being run on 9 orbits of data and a high increase long duration fault exists sometime in the first 3 orbits and a high increase short duration fault exists sometime in the last 3 orbits. The number of iterations is 3 (9/3 = 3). So, logDetPower will be run thrice on the three sets of 3 orbits each, that is, orbits 1 to 3, 3 to 6 and 6 to 9. According to logDetPower (Figure 5.35), the score for orbits 1 to 3 is 5, for orbits 3 to 6 it is 0 and for orbits 6 to 9 it is 3. With the logic used in script PL (Figure 5.36), the resultant score is the maximum of 5, 0 and 3 which is 5. A fault with a score of 5 is considered very severe and will correspond to the high risk lamp in the VGS's fault-management system.

5.4.2 Analysing the Accuracy and Applicability of the Algorithm logDetPower

There are two parts in analysing the accuracy and applicability of the algorithm. Firstly, I assess the accuracy of the algorithm and prove that it meets the requirements with respect to the fault-detection system. The (self-imposed) requirement would be to achieve an accuracy (in terms of detecting different categories) of atleast 80 % and a detection rate of atleast 90 %. Secondly, the designed system is applied to a hypothetical mission to establish that it can in fact be used for any similar CubeSat mission and is not exclusive to ManitobaSat-1.

Accuracy of the algorithm To estimate the accuracy of the algorithm, 60 random test cases are generated (Algorithm 2) and the expected scores and the actual scores from the algorithm are compared. The expected scores are the score values for each test case that I would expect the algorithm to give. For instance, when a test case with a high increase and long duration fault is considered, the algorithm should give a score value of 5 according to the conditions in Table 5.10 and logDetPower. So, 5 is the expected score. The actual score is the real score that the algorithm provides when this test case is tested. If the algorithm fails to give a score of 5 and instead gives a score of 3, the actual score will be 3 and the expected score will be 5 for the test case.

Algorithm 2: Pseudocode for generating the test cases. The test cases are given in Appendix A

Result: Test cases

T: Number of required test cases a: The minimum increase in consumption the user would like to test

b: The maximum increase in consumption the user would like to test

for *i = 1 : T* **do**

inc(i) = randomgenerator(a, b, 1) ;

% inc is the magnitude of the increase of the fault; The function randomgenerator is described after the algorithm.

dur(i) = randi([t1, t2]);

% dur is the duration of the fault and randi is a function in Matlab that generates a random integer between the numbers t1 and t2

startpt(i) = randi([s1, s2]) ;

% startpt is the time at which the user wants to start the occurrence of the fault and s1 and s2 are the time limits for startpt set by the user

endpt(i) = startpt(i) + dur(i) ;

% endpt is the time at which the fault ends

Test case parameters are generated and script PL can be tested on these

end

function [r] = randomgenerator(a, b, number)

% This function generates a random number between a and b and number stands for the number of points I would like to generate. For this research, it is 1

r = (b - a) * rand(number, 1) + a ;

% rand is a matlab function that produces uniformly distributed random numbers

end

The power consumption increase is randomized between 0.2 to 8 W with 20 cases in each group between 0.2 and 2, 2 and 5 and 5 and 8. The number of orbits is set to 3. 30 of the cases occur over 2000 to 5000 seconds while the rest occur for a shorter duration (120 - 1000 seconds). The time at which the fault is introduced is randomized as well. This setup gives a fairly good estimate as it covers a large subset of possible faults. The test case parameters are provided in Appendix A.

From the results in Figure 5.37, it is observed that all cases required the algorithm to be run.

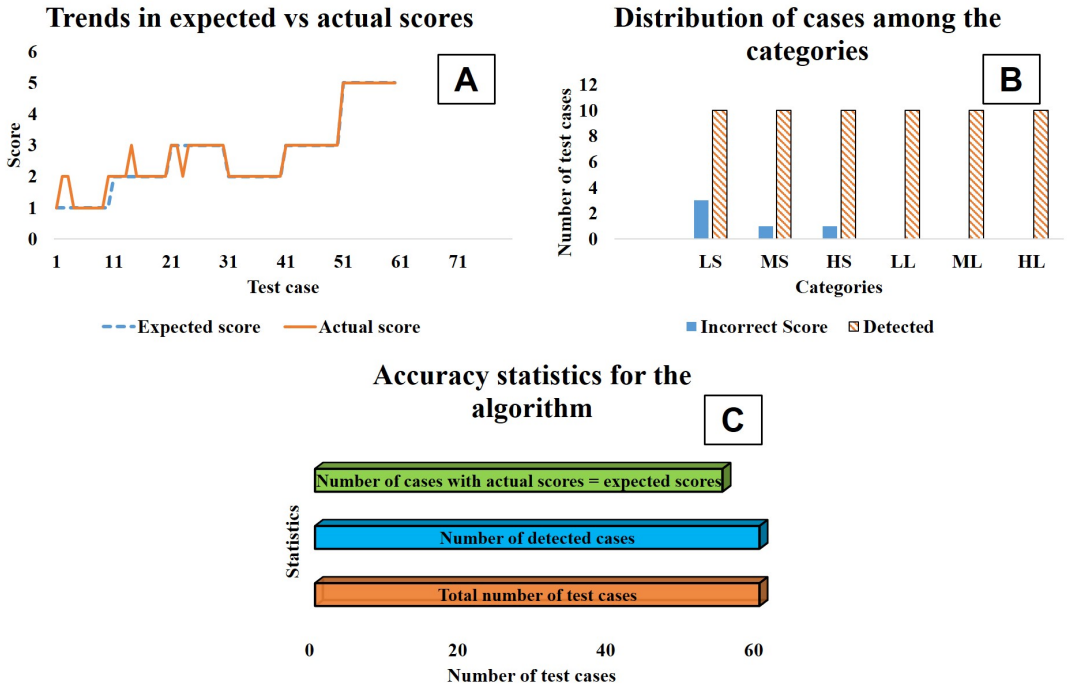

Figure 5.37: Summary of results obtained from the test cases for the power consumption fault for ManitobaSat-1

In plot A of the figure, I observe that the expected and actual scores closely match for most cases except for a few shorter duration cases. This corroborates the

results shown in plot C where the detection rate is 100%. 92% of the cases had the actual score values match their expected scores while the rest slightly underestimate or overestimate the level of urgency/score by 1. Since all the test cases have faults, I expect them all to have a score of greater than 0. Rightly, the scores for all the cases are > 0 and the number of detected cases is equal to the total number of test cases in plot C. In plot B, the various categories chosen for the test cases with the distribution of accurate detections among them is given. Recall, LS, MS, HS, LL, ML and HL stand for low power short duration, moderate power short duration, high power short duration, low power long duration, moderate power long duration and high power long duration respectively. Though detected, low power short duration cases have a higher incidence of overestimation in their scores. This might be due to adjustments made to detect the most minute changes leading to over-sensitivity. The tendency for overestimation can be addressed in future research by improving the algorithm's accuracy and flexibility to work for all cases of power increases. Overall, the accuracy (92%) and detection (100%) rates of the system are good. To conclude, they meet the minimum accuracy and detection rate requirements of 80 % and 90 % respectively.

5.4.3 Application to a hypothetical mission

The fault-detection algorithm for power consumption can be applied to any mission by providing the power consumption telemetry from the satellite. To demonstrate this, a mission loosely based on the 3U CubeSat launched in 2003 called QuakeSat is taken as an example. I assume that its power consumption timeline repeats after every two orbits. Arnold et al. [82] provides the peak power budget for QuakeSat. 3.6 W was continually required to support the C&DH, uplink and board operations with a downlink power consumption of 1.4 W. The peak power consumption was 12.6 W

when all components were operating and consumed 5.7 W during eclipses. Assuming an orbital period of 6000 seconds and sun-facing time of 3000 seconds per orbit, a timeline is formulated. Communication takes place every two orbits between 2000 and 2500 seconds and 5000 and 5500 seconds. The timeline is shown in Figure 5.38.

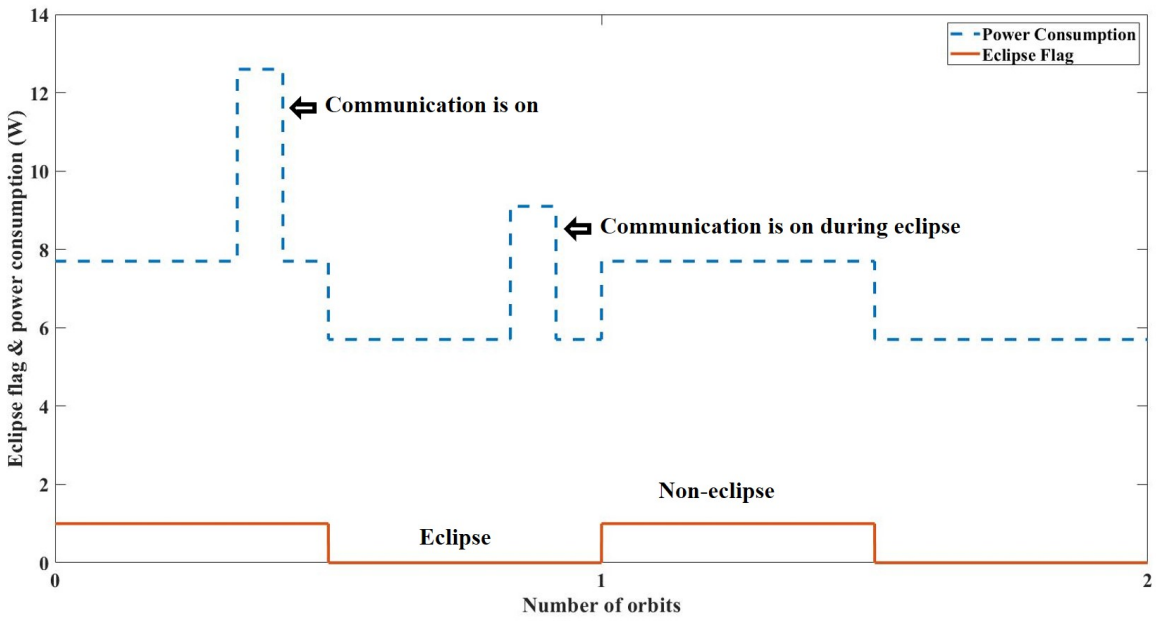

Figure 5.38: Power consumption of the hypothetical mission over every two orbits

I choose ten random test cases from each category as described in the previous section that used Algorithm 2. These cases are tabulated in Appendix A. While the detection rate is 100%, the accuracy rate is 85% which is slightly lower (Figure 5.39) than for ManitobaSat-1 (92%). That means that though all the cases rightly have actual scores > 0, only 85% of them have their actual scores matching their expected scores. Overestimation and underestimation both are seen for the hypothetical mission. The variation in results might also be because the test cases are randomly generated.

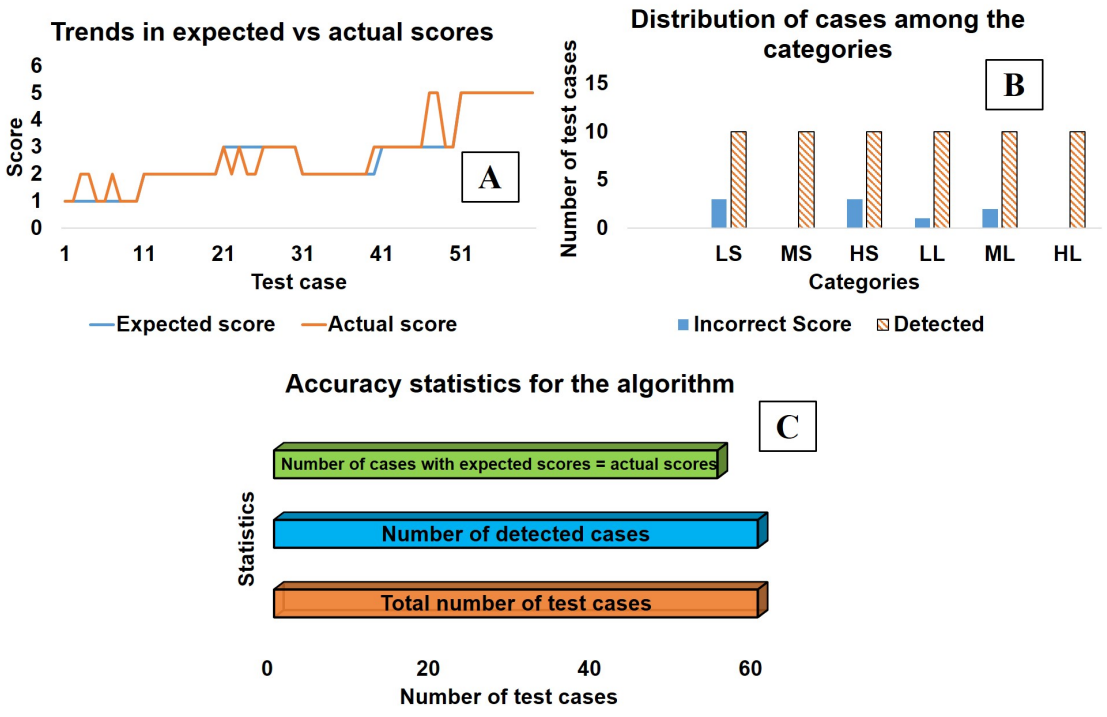

Figure 5.39: Summary of results obtained from the test cases for the power consumption fault for the hypothetical mission

It is interesting to observe the differences in Plot B of Figures 5.37 and 5.39. Though the number of LL cases with incorrect scores is the same for both the missions, the number of HS cases is higher for the hypothetical mission. Also, there are 2 ML cases for the hypothetical mission but zero ML cases that have incorrect actual scores for ManitobaSat-1. The main probable reason for these results is the difference in the power consumption timelines between the missions. The range of the power consumption values for the hypothetical mission is higher than that for ManitobaSat-1. This is primarily due to the vast difference between the power consumption reduction during an eclipse compared to the consumption during the sun-facing time for the hypothetical mission. On the other hand, for ManitobaSat-1 there is a very small decrease in power consumption during an eclipse making the difference between the peak and the minimum power consumptions narrower compared to that for the hypothetical mission. Keeping the definitions of the categories (LS, LL, MS, ML, HS and HL) of the faults the same for both the missions and with a broader range for the hypothetical mission blurs the boundaries between the different categories during detection resulting in inaccurate scores. This is resultantly observed more near the extremes of increases that is the low and high increase cases and further having a shorter duration fault only makes it harder to accurately assess the score. Moreover, the algorithm logDetPower was developed using the observations made with ManitobaSat-1 which makes it obvious for ManitobaSat-1 to have higher accuracy values compared to other missions.

Overall, the results and trends observed are satisfactory and let me conclude that the algorithm developed can be applied to any satellite mission with minimal changes according to the mission operations. The system still meets the minimum accuracy and detection rate requirements mentioned earlier.

5.4.4 Limitations

- Though most possible cases of faults are identified by the system by calculating scores, any combination of anomalies other than the ones described above may not be detected by the algorithms as is observed in the results.
- Detection of low power increase over a few minutes sometimes leads to false alarms.
- Overestimation in the case of moderate and underestimation for high power increases over shorter periods is possible. This can cause a misjudgement of the seriousness of the anomaly in certain cases.

5.5 Summary

In this chapter, I discussed the development of the fault-management system for the virtual ground station. The system has two parts, namely, the SPC module for the detection of solar string failure and an increase in the battery's internal resistance, and the time-domain features module for detecting excessive power consumption. I believe this is one of the most important contributions from this research. The significance of each of the faults and their common causes are detailed. This allows me to understand how the components of the power subsystem interact to cause these faults.

To start with the SPC module development, I discuss the conceptual background of traditional SPC techniques including control rules and charts. The pseudocode for all eight control rules are provided which gives a better understanding of these rules. As I mentioned in this chapter, SPC control rules have to be applied in a smart way

and not mechanically as hard and fast rules. Thus, it is very important to analyze the logic behind these rules, how and why they are applied.

The first fault is the detection of a solar string failure. The steps in the development are broken down systematically into calculation of the control limits and zone boundaries, injecting the faults and synthesizing the algorithms. This provides the user with a road map to thoroughly understand the system. During the first step of understanding the ideal control charts, it was observed that two of the control rules (rules 3 and 4) required minor modifications to accommodate the unique nature of the data I am trying to analyze as opposed to typical industrial data. This was done to reduce false alarms. A comprehensive number of possible anomalies are considered while developing the algorithms. Unlike the other two faults, detecting a solar string failure requires SPC analysis of the array current and then, the analysis of the individual string currents to identify where the fault is exactly. While the SPC on the array current is always run when there are sun-facing current points available, the individual string currents are analyzed only when required and as indicated from the SPC on the total array current. To make this process clear, results in terms of control rule violations and logic flowcharts at every step are laid out. The results are complemented by control charts for every type of considered anomaly. The relative severity of the types of anomalies considered for detection are discussed and are accommodated in the algorithms.

The next fault is an abrupt increase in the battery's internal resistance. Like for the previous fault, the steps in the development are clearly demarcated. Before the calculation of the control limits, the internal resistance is calculated from the battery voltage and current obtained from the simulator. The algorithm for this computation was given. Several categories of faults were considered based on the magnitude of the increase in the resistance. The results for each of them were discussed and it was concluded that the sample size of the datasets did not matter for the SPC analysis.

The control limits are static and as long as there are at least 5 data points and 3 orbits of data, the analysis is independent of the data size. This is in agreement with the conclusion for the solar string failure which is also independent of the dataset size. The main disadvantage of the detection algorithm lies in the estimation of the resistance from the battery voltage and current. This is a time consuming process as I can only consider data points where my assumption of a constant EMF stands true. The algorithm is capable of detecting minute (0.002 Ω) changes as well.

For the detection of excessive power consumption, traditional SPC techniques are not sufficient. These waveforms require unique methods for interpretation and control charts do not satisfy these requirements. Time-domain feature extraction was studied and chosen as the appropriate technique. Similar to the SPC methodologies, the conceptual background for these features was provided and a theoretical understanding of the candidate features was given. The steps for this process were detailed. The power timeline for the ManitobaSat-1 (used as an example) was analyzed to introduce different faults and the different trends in the features for each case. The equations for each of the features used in the calculations were given. The parameters for the fault cases were tabulated and described. These were utilized to develop the algorithms for detection based on the interaction between the features and the relative influence of the different fault cases on the features. Since, extensive characterization was involved in determining the limits in the algorithm's conditions, the accuracy of the algorithm was tested with randomly generated test cases. Bar charts presenting the results and the distribution of the test cases across the fault categories are included. It was observed that the detection rate was 100 % while the accuracy rate was 92 % for ManitobaSat-1. To establish that the algorithm can be conveniently applied to any mission with a few changes in the mission properties, the algorithm was tested with a hypothetical mission. This yielded the same detection rate as for ManitobaSat-1 but the accuracy rate was slightly lower at 85 %. Nevertheless, applications to both

missions satisfied minimum rate requirements.

The conclusions drawn throughout the chapter indicate that this fault-management system is simple and accurate and it is highly beneficial for the VGS. The algorithms were designed by consistently reducing the incidence of false alarms as much as possible. This system is applicable to other missions and the detailed steps described should allow other researchers to implement a similar system for other faults.

The second hypothesis of this research states that the use of traditional SPC in the VGS will help in anomaly detection and reduce the burden on human operators. At this stage, I am able to evaluate this hypothesis to be partially true. The implementation of the developed system in the VGS for real-time management is discussed in the coming chapters. This will allow the complete evaluation of hypothesis two.

Chapter 6

The Virtual Ground Station Interface

6.1 Introduction

The first hypothesis of this thesis states that a virtual ground station can be built based on a mathematical model of the spacecraft that would give ground operators the sense of continual communication with the spacecraft. This system should be capable of managing passes and uploading stored commands, should have a real-time virtual spacecraft model (VSM) and a fault-management system that was developed in Chapter 5. In this chapter, I discuss the development of the virtual ground station (VGS) to accomplish these tasks to minimize the burden on the human operators. Each component of the conceptual model of the VGS in Chapter 1 is described and their significance in the system is established. Recall, the main components of the VGS are:

- Interface terminal through which the operator communicates with the VGS

- Pass manager
- Fault-management system
- Command and telemetry store
- Virtual spacecraft model

The interface terminal is designed in the Matlab App Designer which provides a convenient platform to integrate the visual user interface and an editor to program their behaviour.

This chapter starts with briefly discussing the important features of the graphical user interface (GUI) such as the TLE (Two Line Element) access and latest orbital parameters information and the layout of the GUI alongside the functions of the different tabs. The chapter then goes into the development of the real-time orbit propagator, the pass manager and the command manager. Then, I focus on the embedding of the fault-management system from Chapter 5 into the VGS. Later, I discuss the most important aspect of this thesis, the VSM in the VGS.

Towards the end of the chapter I describe a testbed, designed as part of this research, used to simulate the operations of the ground station with a satellite as realistically as possible. The testbed provides an environment wherein a pass between the ground station and the satellite can be easily executed in real time over real communication links such that the impact of the simulation on the assessment of the VGS performance is minimized. I present results from the VGS with illustrations in the form of a demonstration over a series of events that are expected to take place when the VGS communicates with a satellite.

6.2 Outline of the GUI

The interface terminal's GUI is a software application running on a standard PC with Matlab. The components of the GUI's layout are shown in Figure 6.1. Only the major components are discussed in this chapter. Details regarding the development of the components and different functions written can be found in Appendices B and C. Figure 6.2 shows the GUI with the main screen which hosts the fault alarm lamps and the TLE file finder. The tabs are shown on the right and can be navigated through as required. Each of the subsections that follow are based on the different tabs of the GUI.

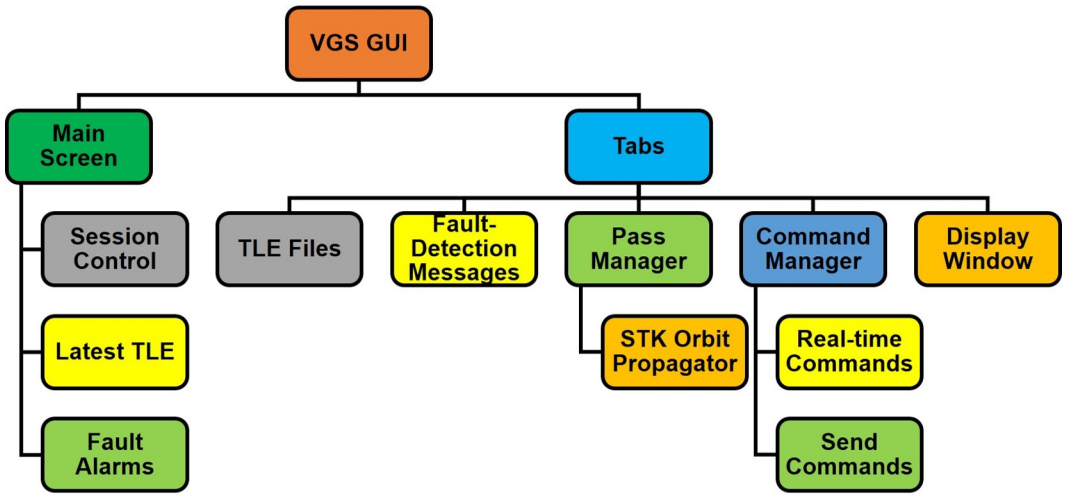

Figure 6.1: GUI outline

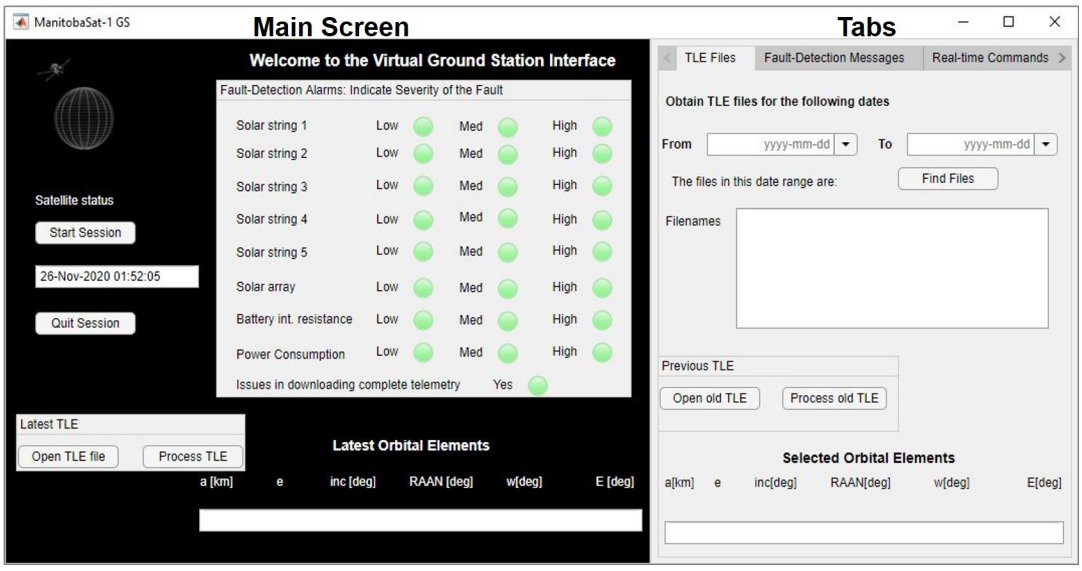

Figure 6.2: The VGS interface terminal (screenshot from Matlab's App Designer)

6.2.1 TLE Files

On the TLE files tab, operators can access the TLE files for the satellite using the button, Open old TLE, which opens the folder with the TLE files with the latest file at the top of the list. To obtain the latest Orbital elements, the Process TLE button extracts the six orbital elements from the selected TLE file. The code for this extraction is adapted from Mathworks website [83] and is provided in Appendix C.1. TLE files are automatically downloaded at regular intervals using the Space Track TLE Retriever software (Figure B.1) [84]. Appendix sections B.1.2 and B.1.3 have more details.

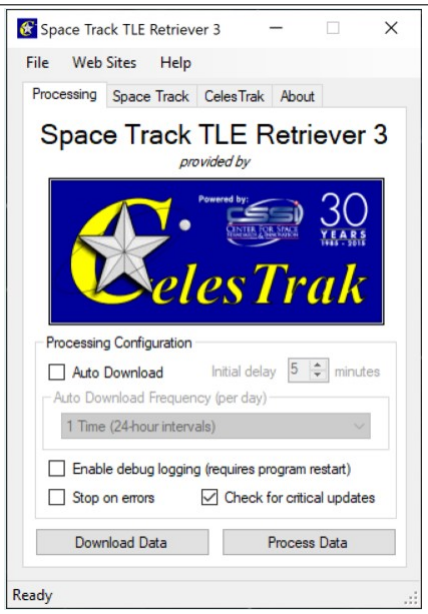

Figure 6.3: Celestrak's Space Track TLE Retriever [84]

6.2.2 Pass Manager with an Orbit Propagator

The VGS should be capable of predicting the passes for any satellite over the physical ground stations I would like to use so that it can initiate contact with the satellite when it approaches the stations. I assume that a satellite has its receiver always on except when transmitting, that is, the satellite is always listening. When a pass occurs, the VGS sends a message to the satellite to initiate communication, which requires prediction of the contact times which, in turn, requires a real-time orbit propagator to indicate the satellite's position at all times. Systems Tool Kit (STK) offers a number of propagators (analytic, semi-analytic, numerical and external) that allow analysing and visualizing a mission. Since the GUI is in Matlab's App Designer, an external real-time propagator is the appropriate choice for my application. Using this type of propagator permits me to integrate STK's propagator with Matlab. More details are in Appendices B.3.1 and C.2.

The real-time propagator lets me import TLE files and generate look ahead ephemeris automatically [85]. I choose a two body propagator in this thesis. The choice of the propagator can easily be changed in the script. The script (STKpropagator) is in Appendix C.2. The pass intervals are predicted by the propagator for a period of 24 hours at a time and are saved in a mat file accessible from the Matlab path. The TLE file is updated every 12 hours when it becomes available from the Space Track Retriever application and the pass intervals are predicted again for the next day. The ground station locations used in the propagator can be found in the script in Appendix C.2. The mission can be visualized in 3D and 2D in the STK window that appears as a separate window when the VGS is running. The views are shown in Figures 6.4 and 6.5. The real-time position and velocity are seen in the STK window.

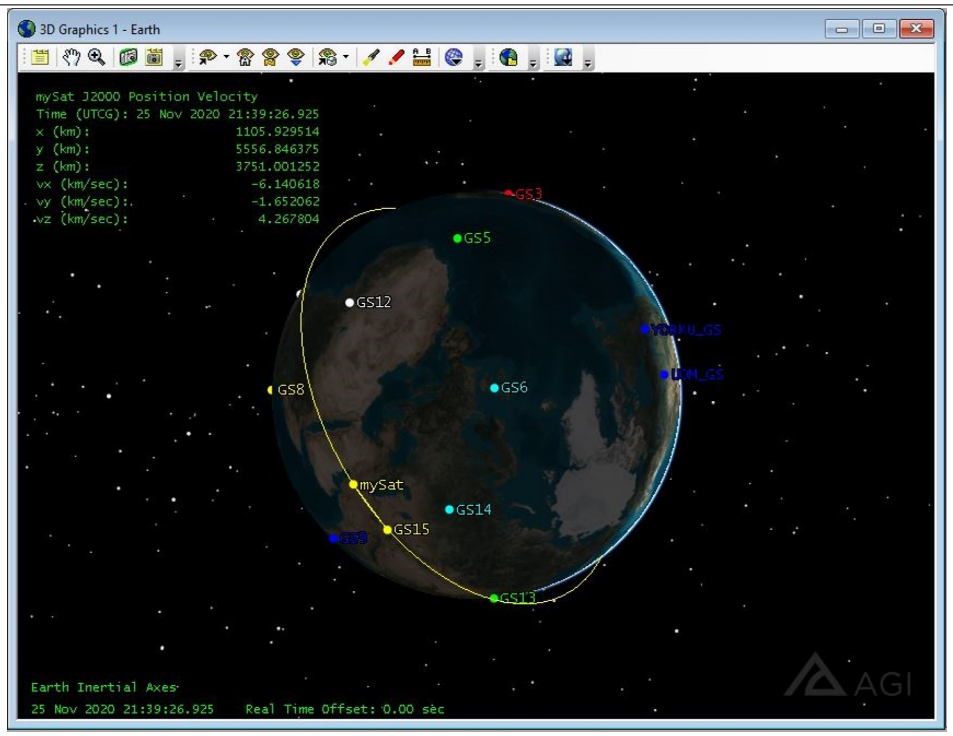

Figure 6.4: The 3D visual from the STK window of the GUI containing the real-time orbit propagator. In the 3D view, observe the satellite 'mySat' and the locations of the various ground stations.

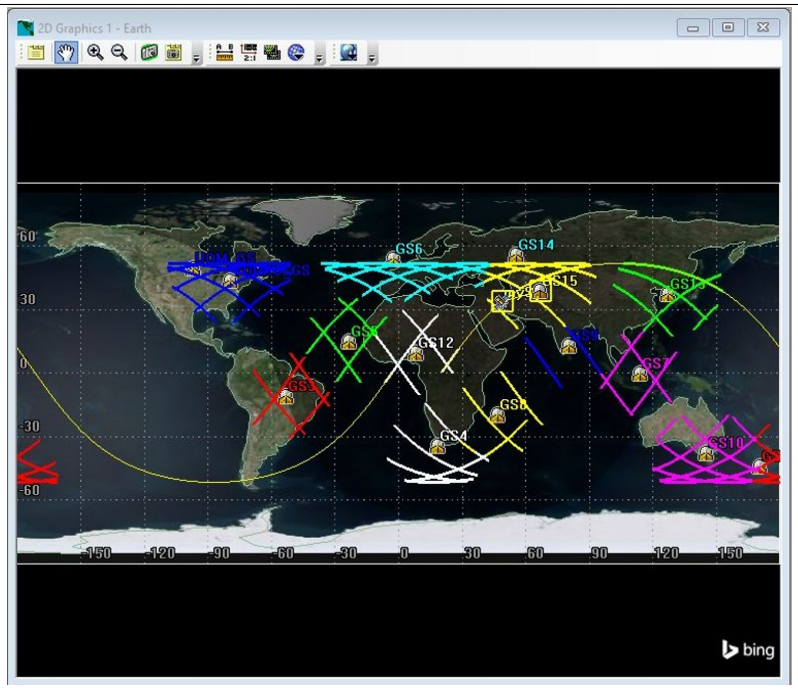

Figure 6.5: The 2D view from the STK window of the GUI containing the real-time orbit propagator. The 2D view shows the path of the satellite and the range or coverage area of each ground station with thick lines forming net like structures. Observe the location of the satellite near the two ground stations, GS14 and GS15 and highlighted by a square around it. The squares around the satellite and GS15's icon appearing simultaneously indicate that the satellite is inside the coverage area of GS15 and a pass is occurring.

The pass manager tab in the GUI displays the access times predicted by the propagator for the next 24 hours. The implementation of the pass manager is discussed in Appendices B.3.1, B.4.1 and C.2.1.

Countdown Timer

To let the operators know that a pass is about to occur, there is a countdown timer (Figure 6.6) of 60 seconds which appears in a separate UI figure prior to every pass. The implementation of the timer is discussed in Appendix B.4.1.

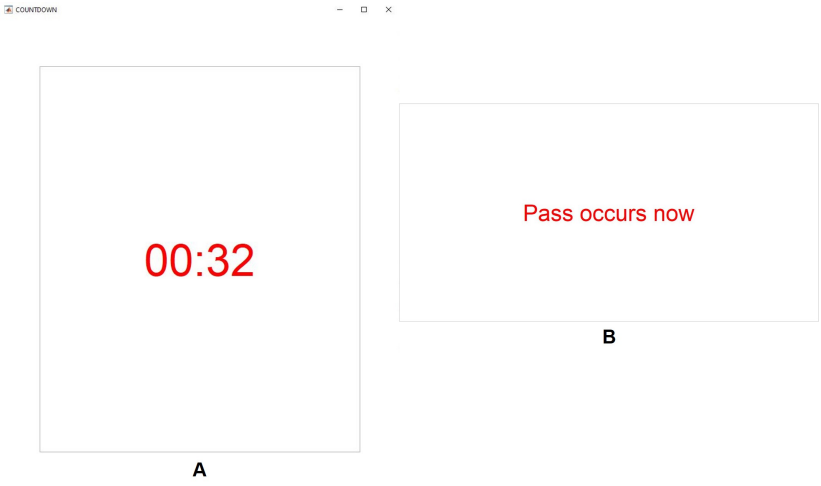

Figure 6.6: The countdown timer is shown in A and it appears 60 seconds before a pass and in B is the message displayed when the pass begins (screenshot from Matlab's App Designer).

Events During a Pass

When a pass begins, the VGS initiates contact with the satellite by sending a predefined 'handshake' message, for example: "Hello ManitobaSat!". In this thesis, I

ignore the technicalities of encrypting messages, packeting, software etc. The satellite is assumed to be always listening for signals from Earth and when it receives the 'handshake' message, it responds with a message saying "Hello Earth!". Once the VGS receives this message, the communication between the VGS and the satellite is opened up. The satellite sends the telemetry, the VGS receives it and saves it in the telemetry store with a timestamp. The cumulative telemetry store which contains all the telemetry data received till date is also updated. All telemetry files are accessible in the Matlab path. The VGS limits the pass duration based on calculations by the orbit propagator using a custom stopwatch in the system. After these events, the fault-management system performs anomaly detection based on the updated cumulative telemetry.

6.2.3 Fault-Management System

The fault-management system for the VGS was developed in Chapter 5 to detect a loss of a solar string(s), increase in the battery's internal resistance and excessive power consumption. The VGS's fault-management system executes the fault-detection algorithm for each of three faults and computes the scores for the telemetry data. The scores (discussed in Chapter 5) value indicates if a failure exists and the level of risk associated with the fault. Scores range from 0 to 3 for the faults detected using statistical process control (SPC) and 0 to 5 for the excessive power consumption fault detected by time-domain extraction. Any score above zero indicates a fault. The values of scores from the four algorithms (logDetTotal (Figure 5.17), logDetInd (Figure 5.18) (if it is run based on the output from logDetTotal), logDetRes (Figure 5.29) and script PL with logDetPower (Figure 5.36)) correspond to the fault-detection alarm lamps (Figure 6.7) on the main screen of the GUI. The labels of the lamps define the severity of the fault, for instance, a high lamp denotes a severe fault. Naturally,

this indicates that the higher the score, the more severe the fault is. Table 6.1 shows the corresponding alarm lamps that light up and their colours based on the score values for the faults in the solar strings and the internal resistance. Likewise, Table 6.2 shows the information for the excessive power consumption fault.

Score	Lamp label	Lamp colour
0	No lamp is lit	N/A
1	Low	Black
2	Medium	Black
3	High	Black

Table 6.1: Fault-detection lamps in the GUI for the faults in the solar string(s) and internal resistance, connected to different score values in the fault-management system

Score	Lamp label	Lamp colour
0	No lamp is lit	N/A
1	Low	Yellow
2	Low	Black
3	Medium	Black
5	High	Black

Table 6.2: Fault-detection lamps in the GUI for the faults in the power consumption, connected to different score values in the fault-management system

In Figure 6.7, as an example, a high risk defect corresponding to a score of 3 is shown in solar string 1. The high label lamp for solar string 1 is lit and is black in colour. Subsequently, the high label lamp for the solar array is lit and black. Whenever there is a defect in one or more solar strings, one of the solar array lamps are turned on. On the right, in the fault-detection messages tab, the corresponding message for the solar array is shown. This message disappears once the operator clicks on the 'issue has been acknowledged and understood' button following which the respective lamps are turned off. The last alarm lamp in the Figure 6.7 labelled 'issues in downloading complete telemetry' corresponds to the fault in connecting with the satellite. When the VGS is unable to communicate with the satellite or has not received the entire telemetry, this lamp lights up (turns black). This is done by checking the number of bytes available to read when telemetry is expected.

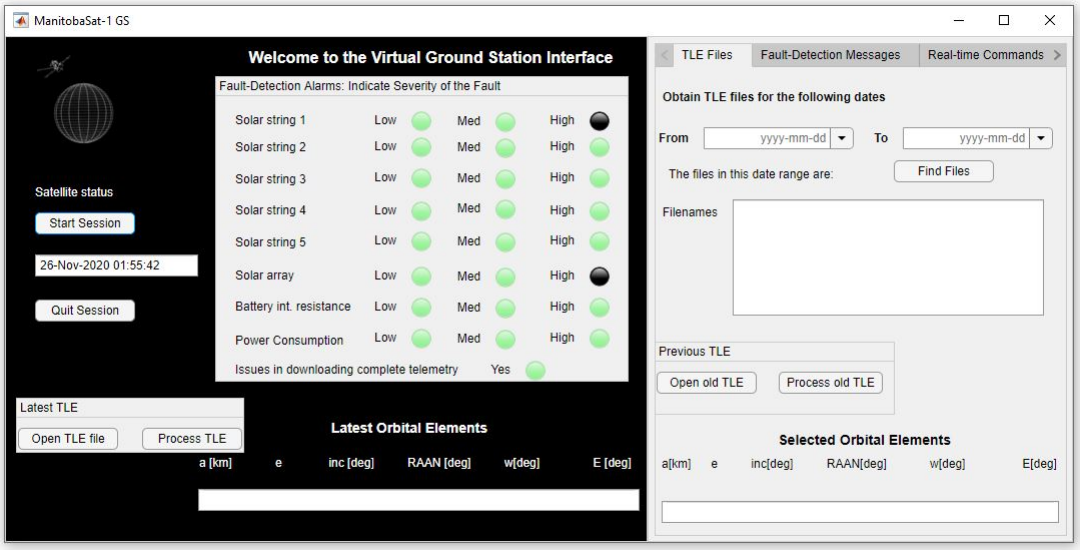

Figure 6.7: The fault-detection lamps on the main screen of the GUI with the fault-detection messages on the right tab (screenshot from Matlab's App Designer)

6.2.4 Command Manager

The command manager that allows operators to send and process commands is split into two tabs (Figures 6.8 and 6.9) depending on the type of command.

- **Real-time** commands

  There are '**real-time**' commands which can be executed immediately by the VGS and the request data is displayed on the tab as shown in Figure 6.8. **Real-time** commands range from requesting for any data from the previous pass of the satellite to analysing how the satellite would be performing currently in real-time. These also allow comparisons between the obtained telemetry and the data predicted by VSM. The operator can send **real-time** commands at any time and can get the data within a few seconds. The implementation is discussed in Appendix B.4.2.

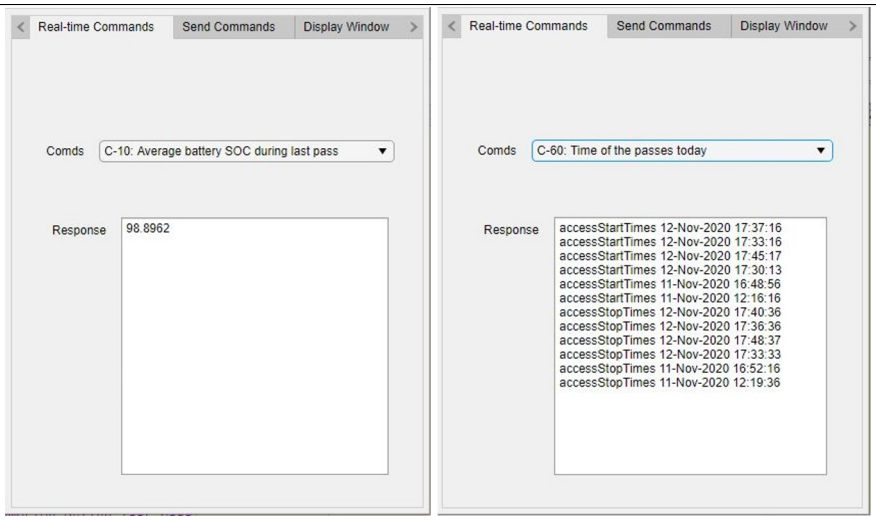

Figure 6.8: Requesting data at any time from the **real-time** command manager through the GUI of the VGS. There are two examples shown: on the left, the battery state of charge is requested while on the right, the pass times for the next 24 hours are provided by the command manager (screenshot of Matlab's App Designer).

- **Send** type commands

  There are also '**send**' type commands that the operator sends to the satellite to perform specific actions. The **send** type commands need to be scheduled for execution on the satellite by providing the VGS with the date and time at which the satellite should take action as seen in Figure 6.9. The datepicker and time spinners are Matlab components that let the operator select the date and time of execution for a command they wish to create. These components do not allow any past and impossible dates and durations to be selected. No command can be created such that no pass occurs before the requested execution time (code in Appendix C.3.4). Again, the operator can store these commands any time through the **Send** Commands tab where they stay in a queue, also known as the command store, which is sent to the satellite when a pass occurs. Once the commands are received by the satellite, they are executed at the specified date and time. If an operator sends a command during a pass, it goes to the command store and from there it is uploaded to the satellite within the pass duration. Details on how to create a command and send it are explained in Appendix B.2. During a pass, commands are uploaded from the queue in the store right after the received telemetry is saved. Implementation of the command manager is given in Appendix B.4.1.

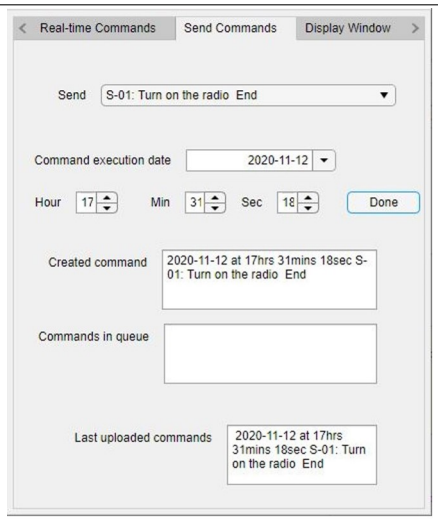

Figure 6.9: Uploading commands from the VGS to the satellite and updating the command queue list in the **send** commands tab of the GUI (screenshot of Matlab's App Designer).

The command manager consists of a set of pre-defined **real-time** and **send** type commands that the operator can choose from. These commands are given in Appendix B.2.

6.2.5 Display Window

The display window shows all the messages displayed on the command window of the main Matlab window or the log file of the VGS sessions. This makes troubleshooting easier and lets the operator get a feel of behind the scene operations. The refresh button when clicked updates DisplayScreen. The transcript from each session displayed on this tab is available to the operators in a text file named 'sessionfile.txt' in the working folder. This is further discussed in Appendix B.1.5.

6.3 Virtual Satellite Model

The power subsystem simulator developed in Chapters 3 and 4 acts as the virtual satellite model (VSM). All the necessary input to configure the Simulink blocks as described in Chapter 4 are automated in the GUI.

The VSM is required to run in real-time so that the VGS can provide real-time data to the operator and behave as closely as possible to the actual satellite. This is done using a real-time pacer embedded in the model. More details on the development of the model are given in Appendix B.4.1.

6.3.1 Obtaining Data and Maintaining the VSM

I obtain real-time data from the VSM every two seconds using listeners and event handles in Matlab and save the data in a file so that it can be accessed by the operator using the real-time command manager or directly by importing the file. Recall, the VGS should be able to provide the operators with any requested data immediately even when a pass is not occurring. This data is the predicted telemetry. The data is saved with timestamps so that the VGS can find the VSM data corresponding to the real telemetry data when the satellite sends it. More details are in Appendix B.4.1 and the script for the relevant functions is in Appendix C.3.2.

6.4 Virtual Ground Station Testbed

The development and implementation of the VGS system involves interaction between a multitude of components and requires a suitable testing environment for verification. The Virtual Ground Station Testbed (VGST) consists of a communication model based on generic interfaces that allow transfer of data and commands between the

VGS and the simulated satellite model. Serial communication is used to interface between the VGS and simulated satellite model. Serial communication is the most common low-level protocol for communication between devices. Many researchers have used the RS232 serial protocol in testing spacecraft communications and systems as they are a reasonable and affordable representation of spacecraft communication modems [86][87][88].

This testbed (Figure 6.10) provides a suitable environment to evaluate the overall architecture of the VGS. More details regarding the VGST are in Appendix B.5.

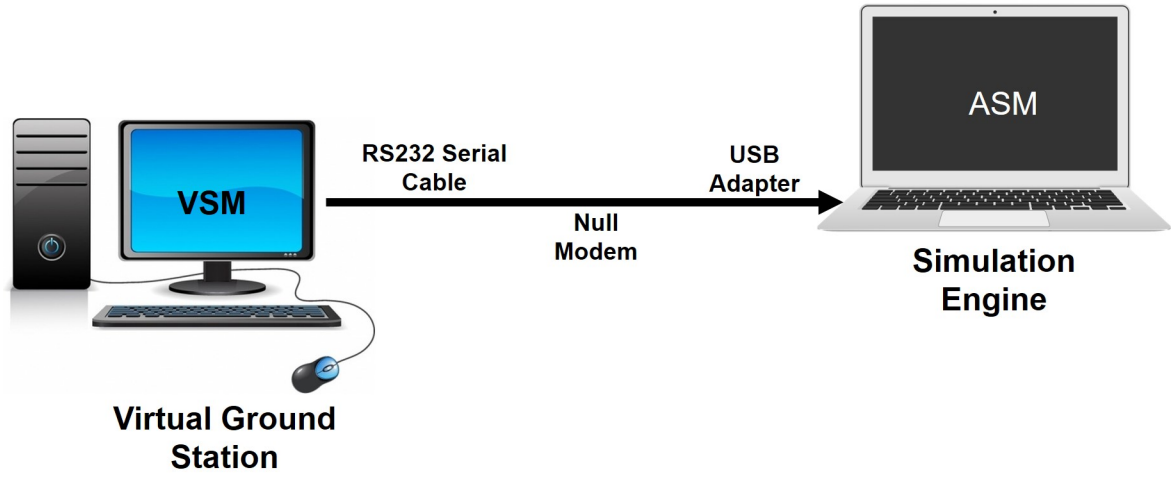

Figure 6.10: Physical architecture of the testbed. ASM stands for the actual satellite model. Image Credits: [89] [90]

The VGST has the four defining features that are the use of Matlab for development, serial communication, the simulation engine and the actual satellite model (ASM). The first three features are explained in Appendix B.5.

6.4.1 Actual Satellite Model (ASM)

The ASM (Figure 6.11) is a Simulink model run on the simulation engine of the VGST. This model represents the actual spacecraft in the testbed. The ASM is executed in soft real-time and sends telemetry generated from it to the VGS during simulated passes. When a command is received at the ASM, the ASM executes it at the specified date and time by applying necessary changes to the parameters of the simulation model. These implementations can be verified from the future telemetry the ASM sends.

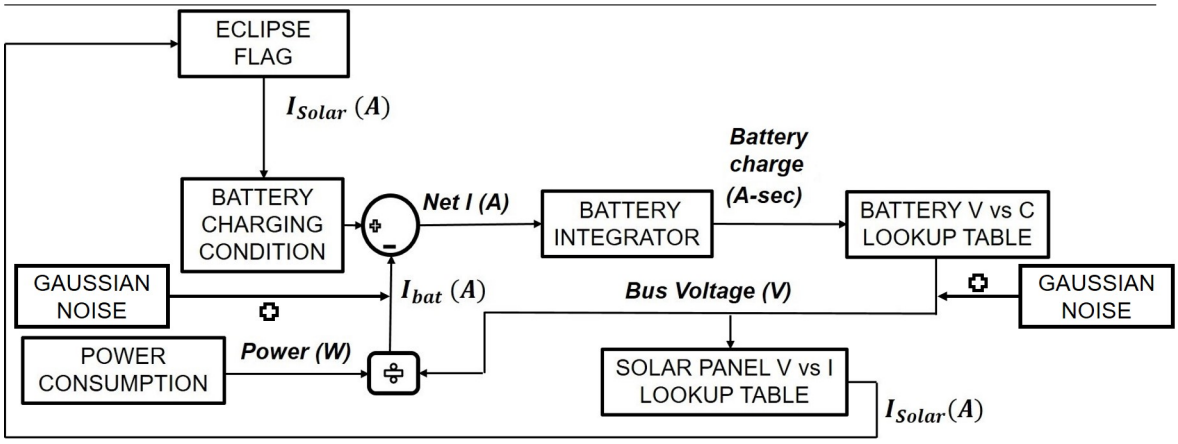

Figure 6.11: The functional model of the ASM

Since the VSM is supposed to be a crude model of the ASM, one would expect them to be implemented in a similar fashion. The same simulator (VSM) is used as the ASM with a difference of noises in the battery / bus voltage and battery current. This is because the ASM represents the actual satellite in space and should be as close as possible to a real satellite. There is always certain amount of noise in the data received from a real spacecraft unlike the data from simulators. Sensor and actuator noise onboard the real spacecraft is the cause of this noise in the data.

In this thesis, I model the noise as a random signal with a normal distribution and mean of zero. This noise can be classified as white noise. Then the noise is added to the previously noiseless voltage and current to produce the total voltage and current respectively. The standard deviations in the battery voltage and current are 1 mV and 2 mA respectively [91] [92].

The simulation engine containing a script facilitates the communication between the ASM and the VGS. When the ASM receives a command from the VGS, it is required to execute the command at the specified execution date and time. Before command handling, I discuss configuring the ASM.

Since the ASM acts as the actual satellite in the testbed, it would require to run in real time. Similar to the VSM, the real-time pacer is used for the ASM. Similar to the VSM, real-time data from the ASM is logged every two seconds using listeners and event handles in Simulink. The ASM is always listening for a message from the VGS and requires the VGS to initiate communication. When the ASM receives the 'handshake' message, it sends the telemetry generated since the previous pass to the VGS. The detailed implementation of the ASM is described in Appendix B.5.3.

Though other parameters do not have specifications on the exact number of observations to send in telemetry, the power consumption requires to be sent in multiples of 3 (period for the power consumption timeline for ManitobaSat-1, this number can be changed). The simulation engine operations are modified to accommodate this by sending the power consumption values in sets of three orbits. For example, when there is data from six orbits, two sets of data each with three orbits data are sent. In cases where lesser than three orbits of data is available, a string of zeros is sent. When there are more than three orbits of data available but the number of observations is not a multiple of three, the observations that are possible to be sent in sets of 3 orbits data are sent in the current pass and the remaining are sent during the next pass.

Once the ASM sends the telemetry, it waits for any commands that might be sent by the VGS. The next section provides a detailed discussion on the command handling by the ASM.

Command Handling by the ASM

'Send' type commands in the command queue of the VGS are sent to the ASM during passes. These commands can be placed in queue on the VGS at any time including during a pass. The VGS sends the following information regarding the commands in order: the number of commands from the command store and the commands themselves. This is done right after telemetry is received. Each command contains the date and time of required execution, the command ID (S-01, S-02 and S-03), the command description and the string 'End'.

On the ASM, after receiving the number of commands from the VGS, the ASM starts reading all the commands one by one. Relevant information required to correctly execute the commands is extracted. The command master list is updated every time a command is received.

On the simulation engine, I want to check for commands to execute every second and not only during passes. This is done by comparing the current time with the execution dates and times for the commands. The command ID is used to execute the commands appropriately. Depending on the command ID, the power consumption of the component is increased or decreased based on what the command requires the ASM to do. For example, the VGS sends a command to switch on the camera at the specified data and time. when the camera is switched on, the power consumption increases by 0.85 W. For the commands, I assume that the radio and camera are off when they are required to be on and the radio is on when it is to be switched off. The code used in the command handling on the ASM is given in Appendix C.3.3. For

sophisticated detection by the ASM to check if a component is already on when it is commanded to switch on or it is already off and has been commanded to switch off, flags with embedded logic and listeners can be used in the Simulink model as part of future work.

On the VSM in the VGS, the commands are executed at the same time as on the ASM since the VSM is expected to function similar to the actual spacecraft. This requires the power consumption on the VSM to change accordingly. Note that the 'send' type commands considered for this thesis require the satellite to either turn on or turn off a component. Hence, the VGS does what the ASM does in terms of comparing the execution dates and times of the commands every second with the current time. Accordingly, the VGS implements necessary changes in the ASM's power consumption appropriately.

Faults into the ASM

Faults are injected into the ASM through the script on the simulation engine that controls the ASM. Faults can be injected for specific periods in the ASM by using conditional switches in the model. To inject a failure in the solar string(s), the current to the respective string(s) is set to 0 through the script. Similarly, the internal resistance is changed directly through the script. To introduce a fault in the power consumption, the magnitude of the fault/increase is added to the ideal power consumption in the ASM. A detailed description is in Appendix B.5.3.

With the testbed developed and the VGS all set, the next section presents a number of possible scenarios to demonstrate the functionalities of the VGS.

6.5 Demonstration of VGS Functionalities using the VGST

To illustrate the functioning of the VGS and its features and to demonstrate its communication with a satellite, I use the testbed. A number of possible scenarios before, during and after a typical pass are described below with screenshots of the VGS and the simulation engine to establish that the VGS can detect faults in the spacecraft using its fault-management system and that the VGS behaves as if it is in constant communication with the satellite at all times using the VSM. All the scenarios considered start at a point where the VGS application is already running and the satellite is in orbit. The ASM on the simulation engine is also run simultaneously. It is important to execute VSM and the ASM at the same time since the models need to run in real-time and sync with each other. A timeline of the events is shown in Figure 6.12. I assume that the events take place in the same order as shown. As such, the events such as obtaining the latest orbital elements, accessing the log file of the session and requesting data from the real-time command manager can take place at any time. The corresponding times (Central Standard Time UTC-6, 24 hour format) of occurrence for these events is indicated in the figure. Note that I change the period of the power consumption to 2 and the orbit duration is taken as 60 seconds. For this demonstration, this means that the power consumption telemetry will be sent from the satellite in multiples of 120 points since fault-detection has to be performed. This change in the period and orbit time are made as, for ManitobaSat-1, the period is 3 and the orbit time is 5560 seconds. This would require a very long duration to run and it would be difficult to present results.

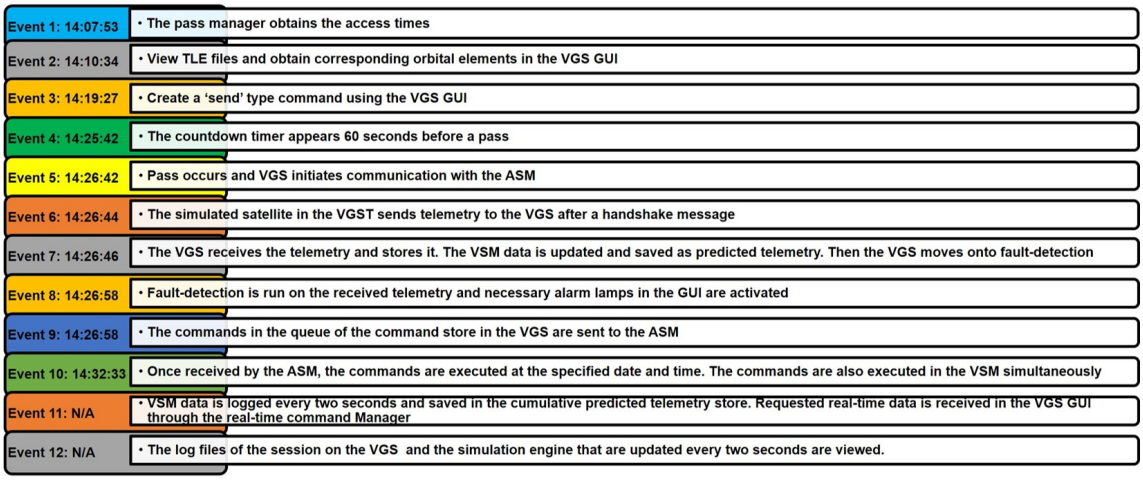

Figure 6.12: A summary of all the events discussed in this section to demonstrate the functioning of the VGS

6.5.1 Event 1: Obtaining Access Times

The session begins at 14:04:27 (Figure 6.13). The VGS loads an updated TLE file (Figure 6.14) downloaded by the Space Track TLE Retreiver and the STK Propagator is run to compute the upcoming access times as seen in Figures 6.15 and 6.16. These figures shows the 2D and 3D views to help visualize the mission. The time shown in the propagator is the Coordinated Universal Time (UTCG).

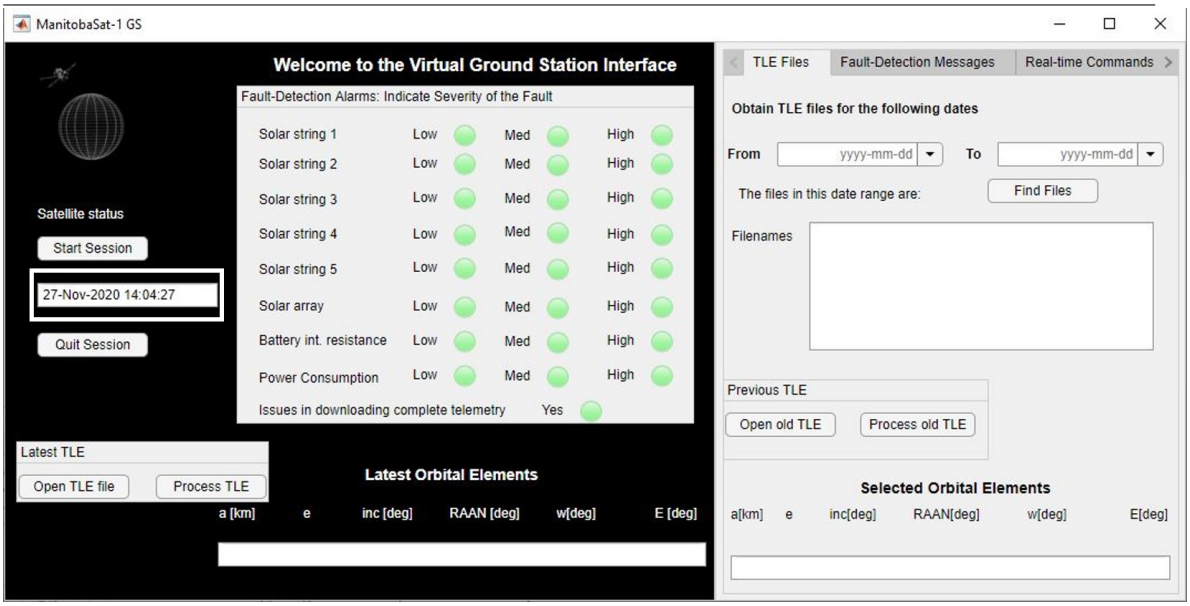

Figure 6.13: The VGS GUI at the beginning of the session. The time indicated is 14:04:27. Screenshot from Matlab's App Designer.

```
1 25544U 98067A   19231.50489250  .00016717  00000-0  10270-3 0  9006
2 25544  51.6395  47.1442 0007365 300.7176  59.3248 15.50378298 25071
```

Figure 6.14: The TLE file given as an input to the real-time orbit propagator. Note that this is a dummy TLE file for a CubeSat and does not necessarily correspond to ManitobaSat-1. This is only used for testing.

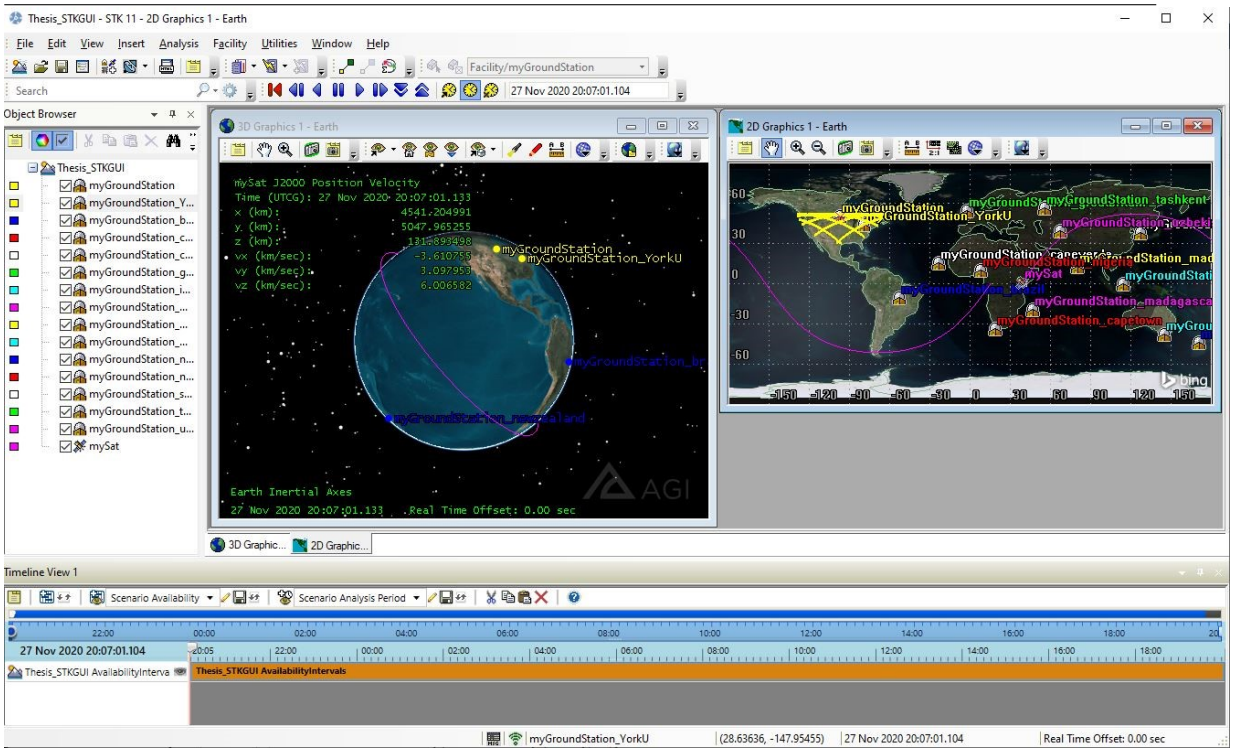

Figure 6.15: The real-time orbit propagator in STK which is a part of the pass manager that computes access times. Screenshot of STK.

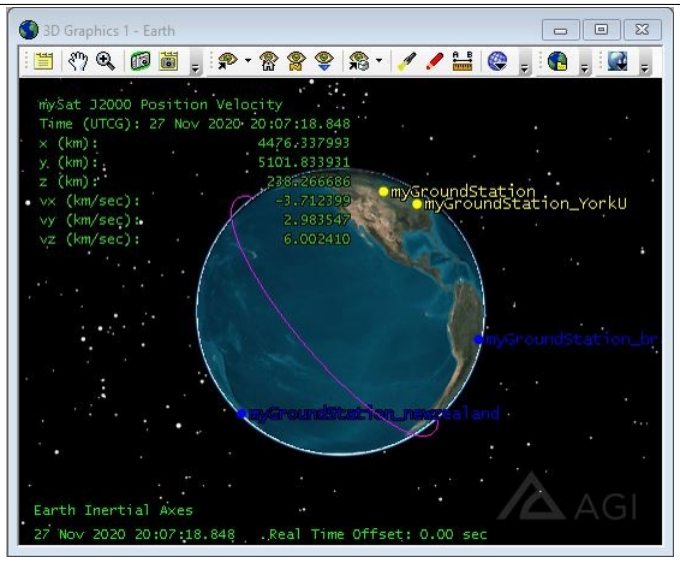

Figure 6.16: A 3D view of the real-time orbit propagator in STK which is part of the pass manager that computes access times. Observe how the propagator indicates the current time (Coordinated Universal Time (UTCG)), current position and velocity vector components of the satellite.

Once the access times are computed, the times are stored in a mat file at 14:07 or 2:07 PM (Figure 6.17 B). At 14:07:53, I use the pass manager to obtain the access time details as shown in Figure 6.17 A. So, there are passes occurring at 14:13 and 14:26. I focus on the pass at 14:26 so that I have enough time to discuss other features in the GUI before a pass occurs.

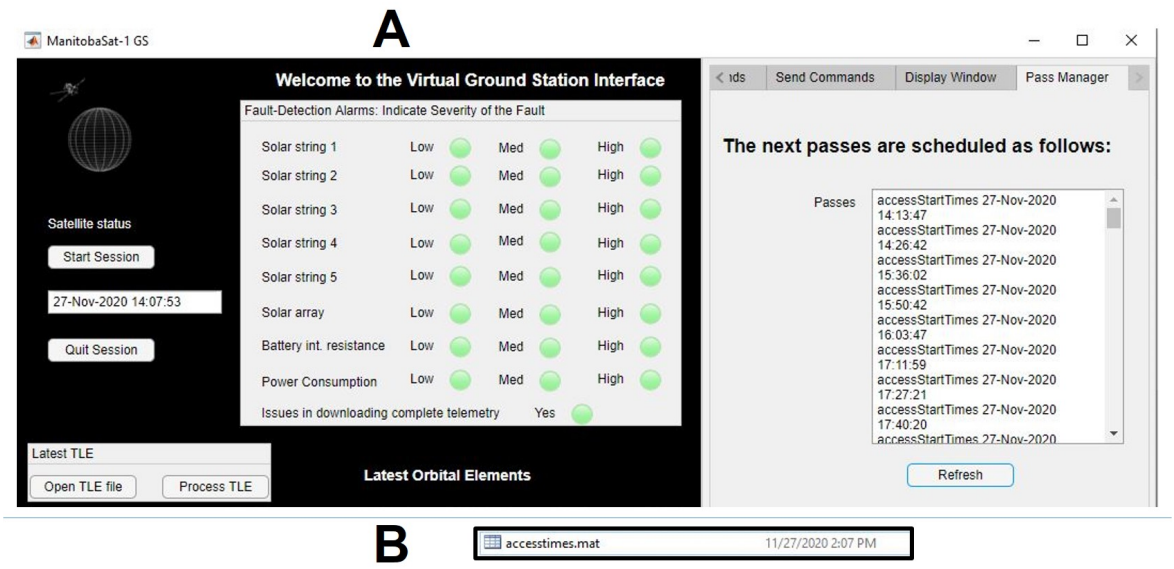

Figure 6.17: A: The mat file with the computed access times, B: The access times viewed using the pass manager in the VGS GUI. Screenshot of Matlab's App Designer.

The display window tab allows me to look at the command window or the log file of the session as shown in Figure 6.18.

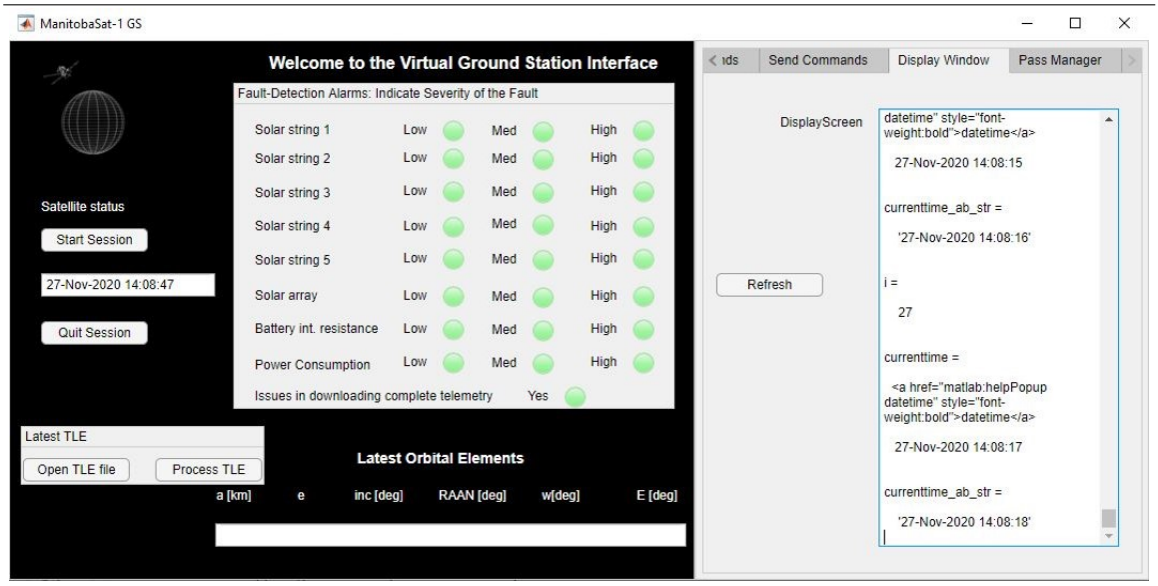

Figure 6.18: Viewing the real-time log file of the session inside the VGS GUI. Screenshot of Matlab's App Designer.

6.5.2 Event 2: Viewing TLE Files and Obtaining Orbital Elements

At 14:09:47, to view the files that were downloaded during a specified date range, I use the TLE Files tab. This is shown in Figure 6.19. A folder appears with all the files when I click on the open old TLE button. TLE files can also be opened on the main screen of the GUI in a similar fashion.

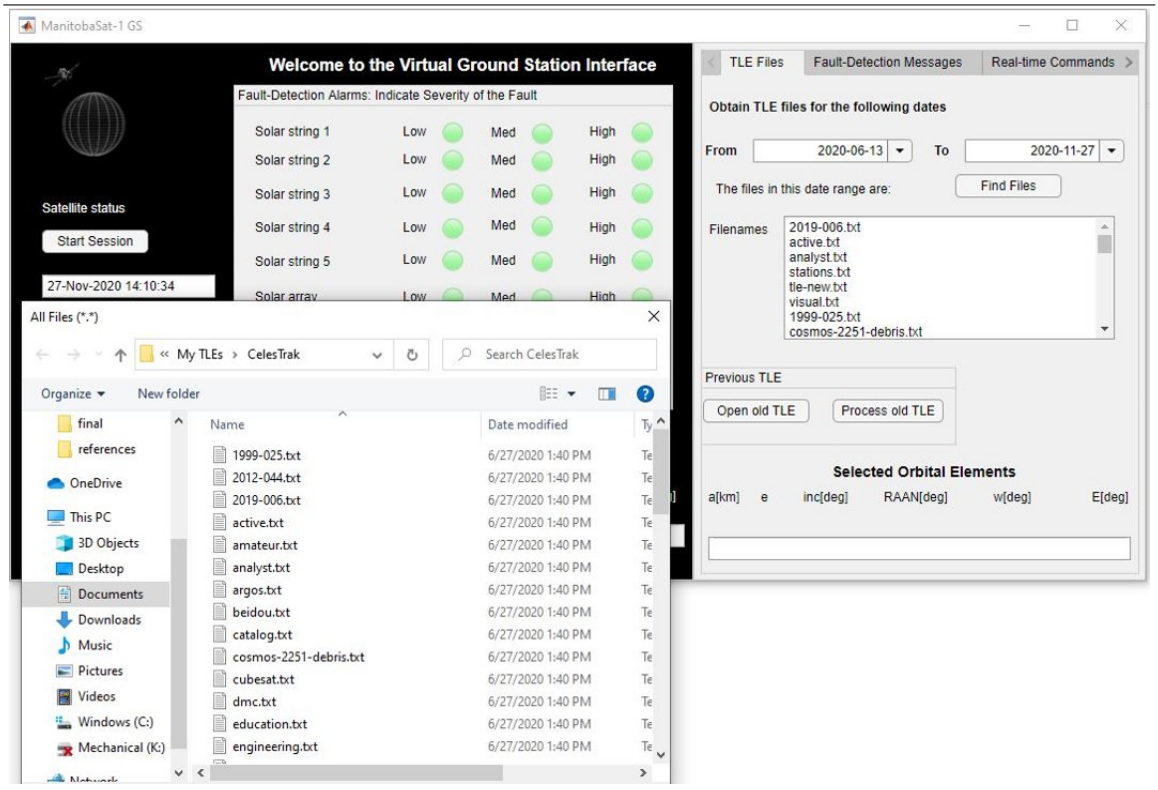

Figure 6.19: Viewing TLE files within a specified date range. Screenshot of Matlab's App Designer.

Now, I select the file called 'tle-new.txt' as an example and click on the process old TLE button to obtain the corresponding orbital elements as seen in Figure 6.20.

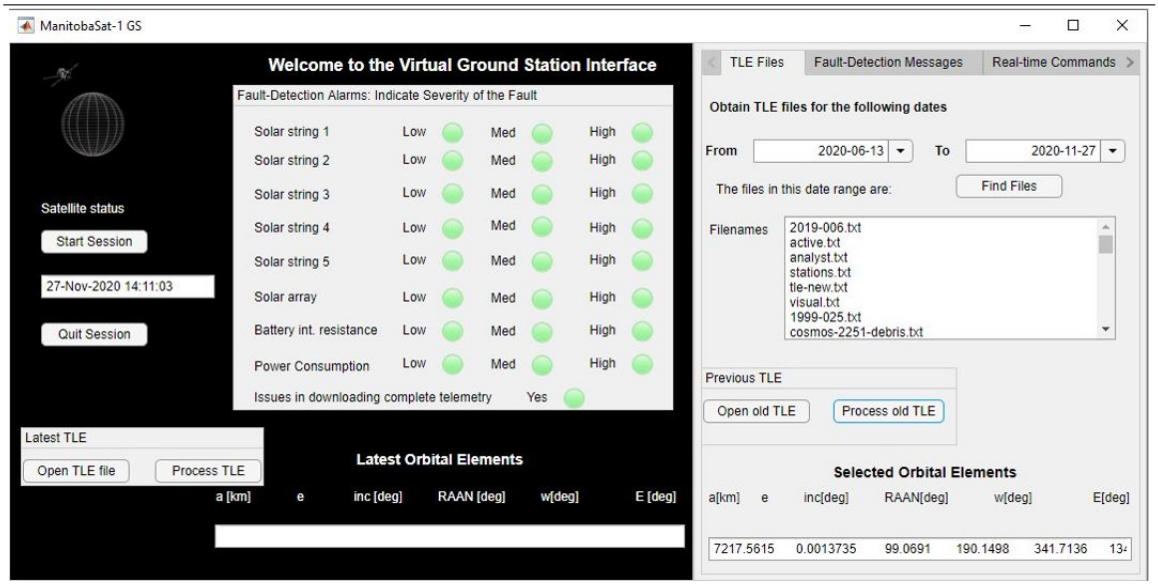

Figure 6.20: Processing TLE files to obtain orbital elements. Screenshot of Matlab's App Designer.

6.5.3 Event 3: Creating a 'Send' Command to Store

At 14:19:27, I use the Send Commands tab to create a command to send to the satellite when the next pass occurs. As can be seen in Figure 6.21, I create a command to turn on the radio on 27th Nov 2020 at 14:32:33. I am able to store the created command in the command queue as a pass is going to occur before the execution time, that is, at 14:26. The command will be sent to the satellite (model) automatically when a pass occurs and will be executed by the simulation engine in the ASM at the specified date and time. Once the command is sent from the VGS to the satellite model, the command queue should be empty and the last uploaded commands will contain the

commands in the queue previously. This will be demonstrated in the coming events.

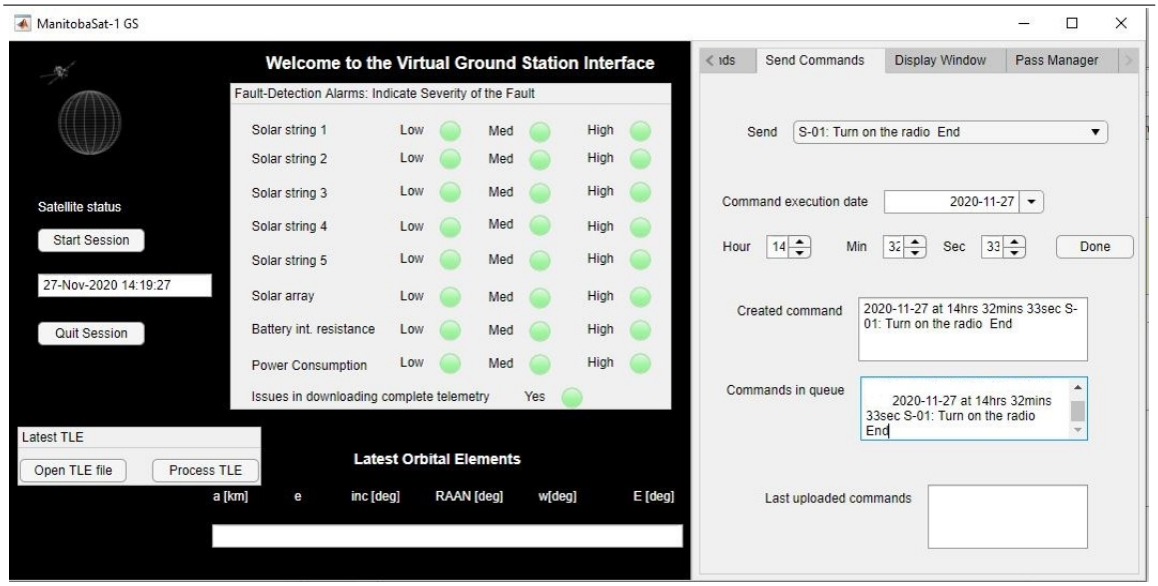

Figure 6.21: Storing a 'send' type command in the queue to send to the satellite during the next pass. Screenshot of Matlab's App Designer.

6.5.4 Event 4: Countdown Timer

A pass is scheduled/predicted at 14:26:42. The countdown timer appears at 14:25:42 (Figure 6.22) since it is 60 seconds before the next pass. When the timer reaches 0 at 14:26:42, it enlarges and displays a message mentioning that the pass is occurring right now.

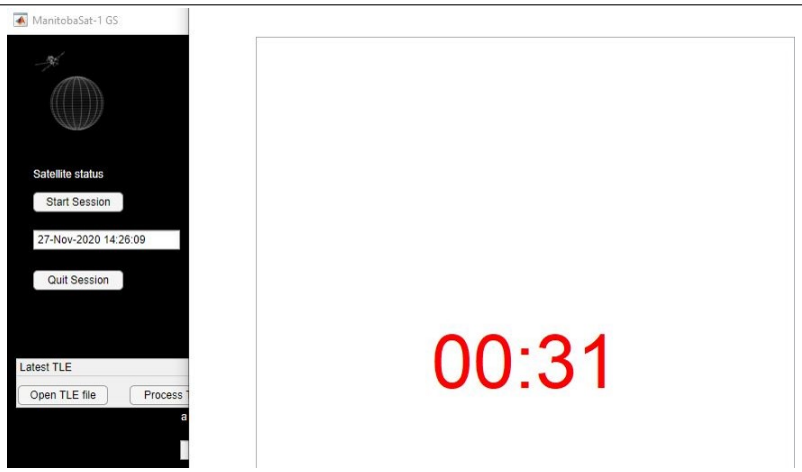

Figure 6.22: The countdown timer in the VGS GUI at 14:25:42 and stops at 14:26:42 when the pass occurs. Screenshot of Matlab's App Designer

6.5.5 Event 5, 6 and 7: Events During a Pass

At 14:26, the VGS initiates the pass and sends a handshake message to the satellite model. The satellite model responds by sending a handshake message as shown in Figure 6.23. After this, the simulation engine sends the telemetry data to the VGS. This communication is shown in Figure 6.23 where A shows a screenshot of the VGS while B shows the simulation engine screen. Note that there is minute difference of less than 45 seconds in the system times of the VGS and the simulation engine. This is not a cause for concern as it is a very small difference and does not affect the test in anyway.

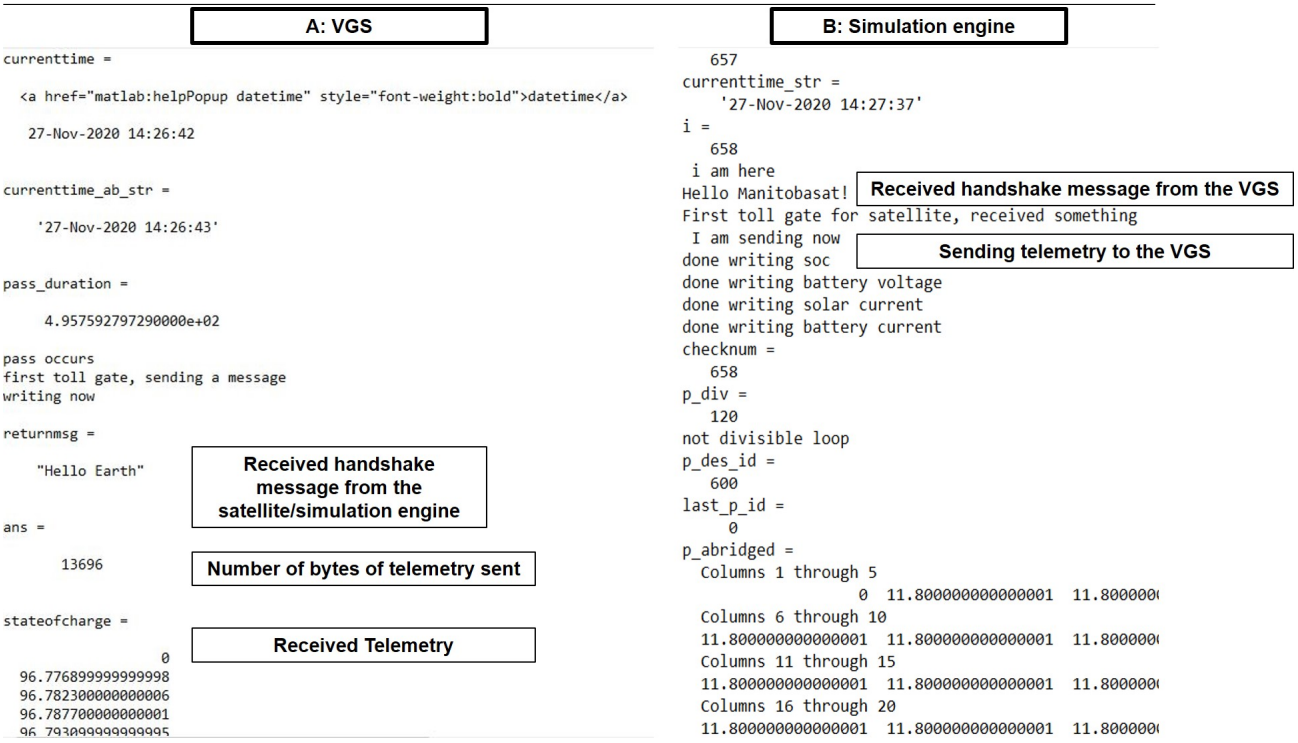

Figure 6.23: A typical handshake between the VGS and the satellite after which telemetry is sent to the VGS from the satellite. Screenshot of Matlab's command window

The VGS reads all the telemetry and converts it to the appropriate format and saves it in a mat file and a text document. Predicted telemetry from the VSM in the VGS is saved as well. These files are saved immediately after receiving the actual telemetry as seen in Figure 6.24. The VGS is now ready to perform fault detection on the telemetry.

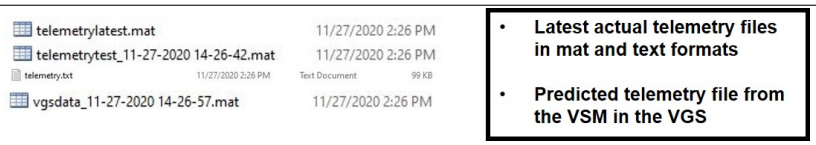

Figure 6.24: The mat files containing the actual telemetry and the predicted telemetry data saved on the VGS computer at 14:26

6.5.6 Event 8: Fault-detection on the Received Telemetry From the Satellite

The VGS performs fault-detection based on the fault-management algorithms developed in Chapter 5 that are run on the solar string currents, the battery's internal resistance and the power consumption (on the VGS). If any faults are detected, the lamps on the Main Screen of the GUI light up. In this case, the low level power consumption lamp lights up at 14:28:52 as shown in Figure 6.25 A. The corresponding alarm messages show up in the fault-detection messages tab (Figure 6.25 A). At 14:29:12, when I click on the message to acknowledge it, the message disappears and the lamp switches off as expected (Figure 6.25 B).

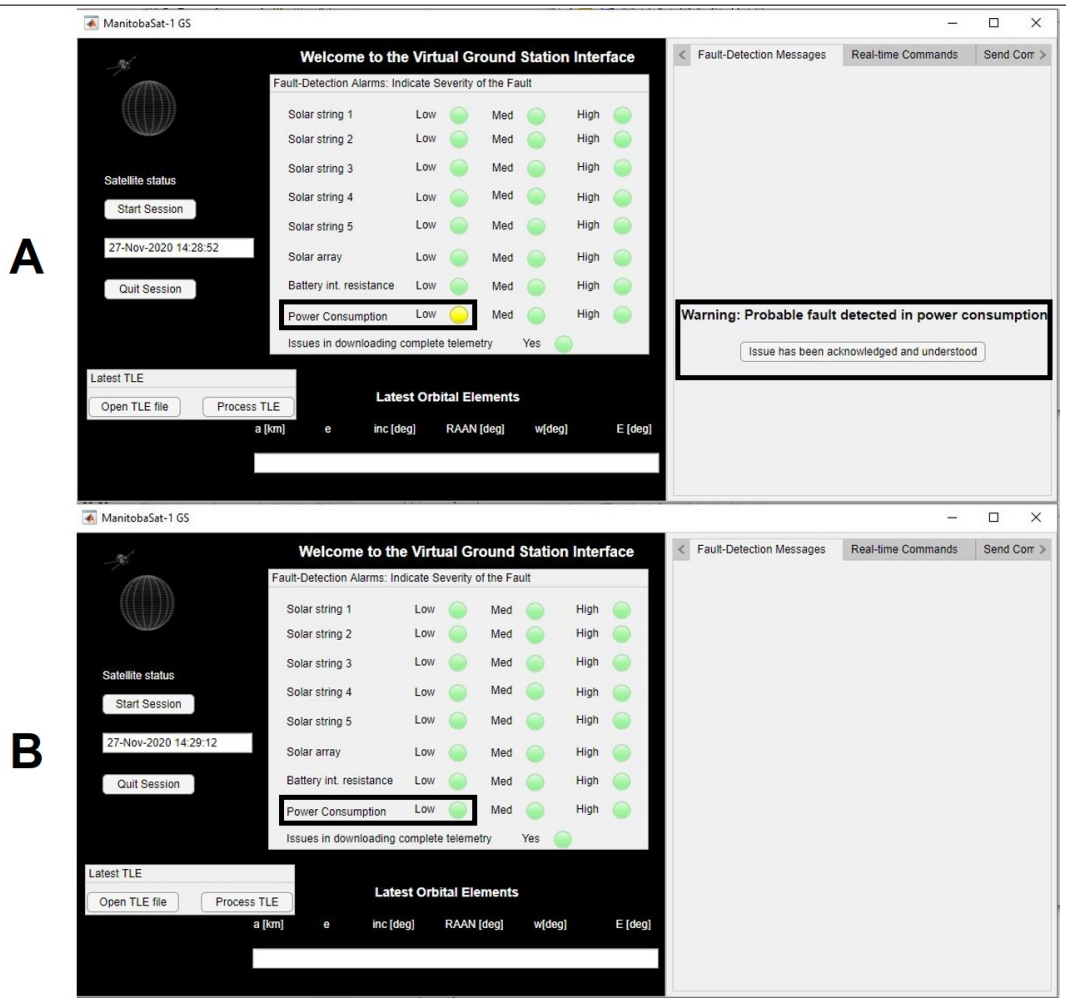

Figure 6.25: The alarm lamps in the VGS GUI light up if required after the fault-detection algorithms are run by the VGS. In Figure A, the indicated low power consumption lamp is turned yellow and a message appears on the fault-detection messages tab. In Figure B, the message disappears and the lamp is turned off when the message is acknowledged. Screenshots of Matlab's App Designer

6.5.7 Event 9: Sending Commands Stored in the Queue to the Satellite From the VGS

I stored a command in the queue in Event 3. The VGS sends this command to the satellite model following the completion of the fault-detection process. This occurs at 14:26:58. Once the commands are sent by the VGS, the commands that were in the queue are deleted and they move to the last uploaded commands as seen in Figure 6.26 B. This confirms that the commands have been sent. There is also a confirmation message on the command window (stored in the log file) as shown in Figure 6.26 A. Compared with Figure 6.21 in Event 3, the commands in queue textbox is empty in Figure 6.26 B.

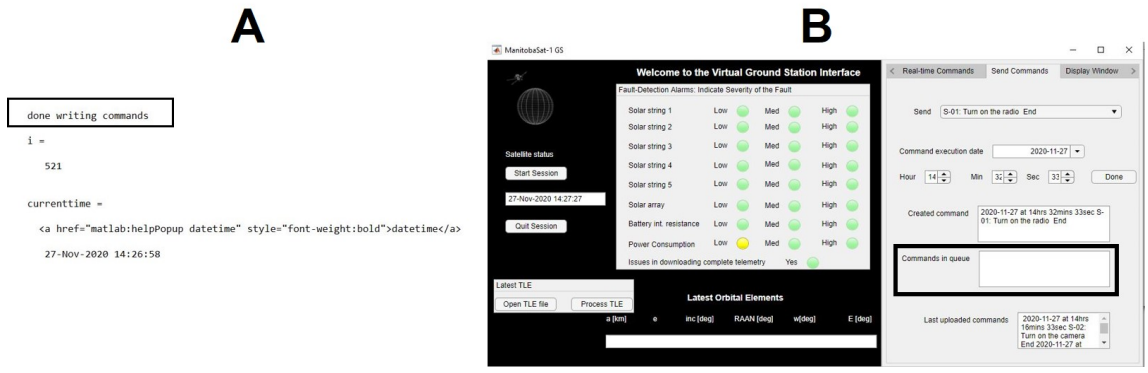

Figure 6.26: Figure A shows the confirmation message on the VGS that verifies that the commands in the queue have been sent to the satellite model (screenshot of Matlab's command window) and Figure B shows the VGS GUI's command manager where there are no commands in the queue after sending the command created in Event 3 to the satellite model (screenshot of Matlab's App Designer).

6.5.8 Event 10: Receiving and Executing Commands in the ASM

On the simulation engine, the commands are read and saved to the master command list. This is confirmed from the log file of the simulation engine session (Figure 6.27).

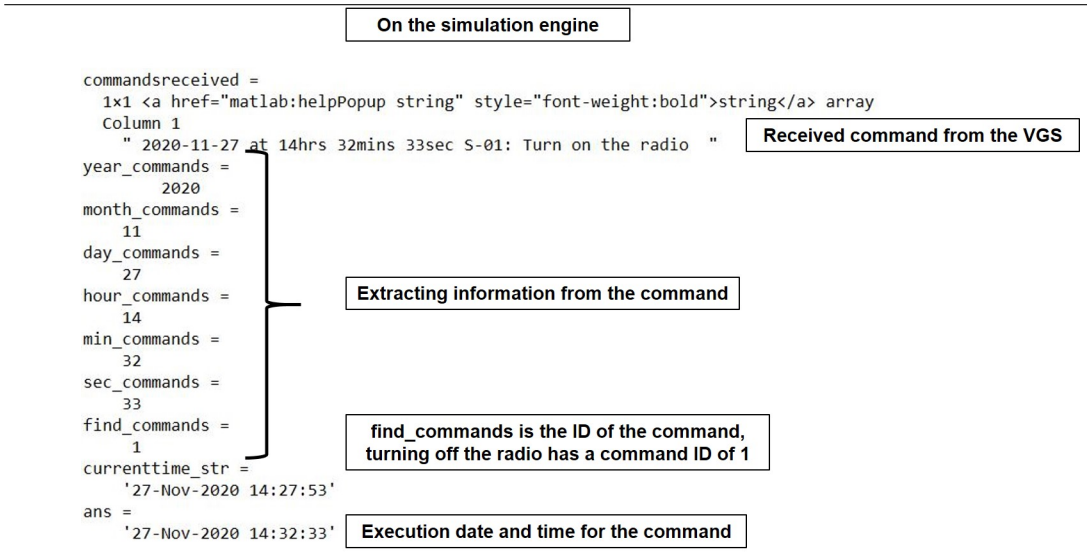

Figure 6.27: Receiving the command sent by the VGS on the ASM. Here, information is extracted from the command and the date and time of execution are noted. Screenshot of Matlab's command window.

At the specified execution date and time, 27th Nov 2020 14:32:33, the simulation engine increases the power consumption by 2.77 W. This is because the command to be executed is 'Turn on the radio' which requires an increase of 2.77 W in the power consumption (Figure 6.28). A confirmation message, 'comm 1' which means that the command with ID 1 has been executed is seen on the screen as shown in Figure 6.29. Similarly, the VGS implements the increase in the power consumption in the VSM.

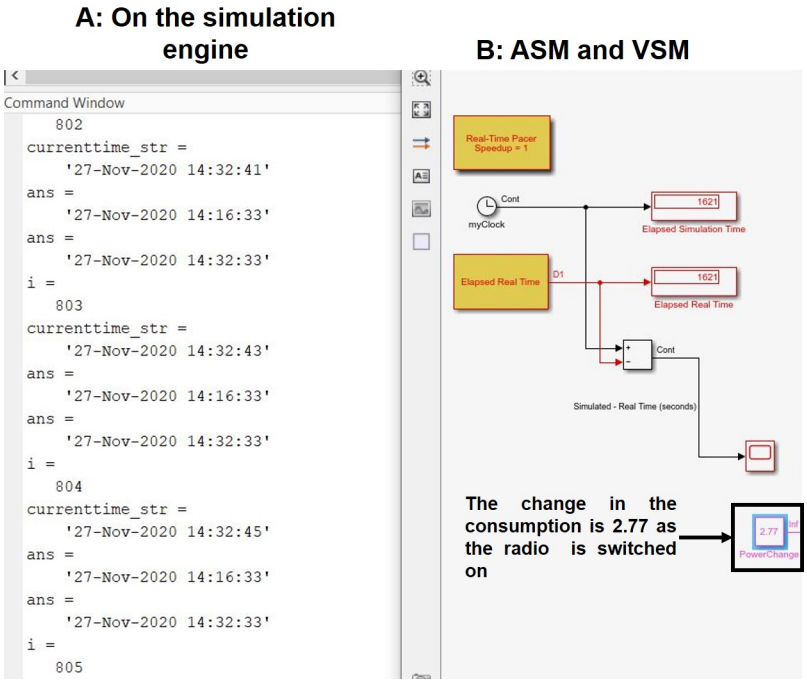

Figure 6.28: Figure A shows receiving the command from the VGS on the simulation engine and Figure B depicts the execution of the command at 14:32:33 by switching on the radio in the ASM and the VSM (Screenshots from Matlab).

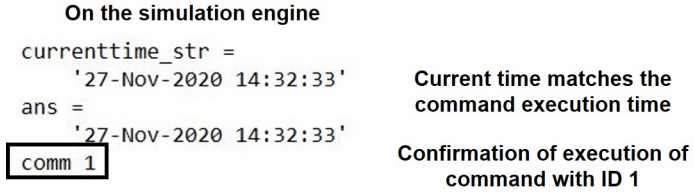

Figure 6.29: Confirmation message on the simulation engine for the execution of command with ID 1, that is, turning on the radio (screenshot from Matlab's command window).

6.5.9 Event 11: Data Logging and Real-Time Commands

Predicted telemetry from the VSM is logged regularly so that real-time data can be obtained from the command manager. It is saved in a mat file that is over-written every two seconds. The VSM and ASM continue to run in real time and the VGS keeps updating the required mat files until the next pass. Now, I request the real-time command manager to provide me with the average power consumption and the average state of charge during the last pass. The results are shown in Figure 6.30.

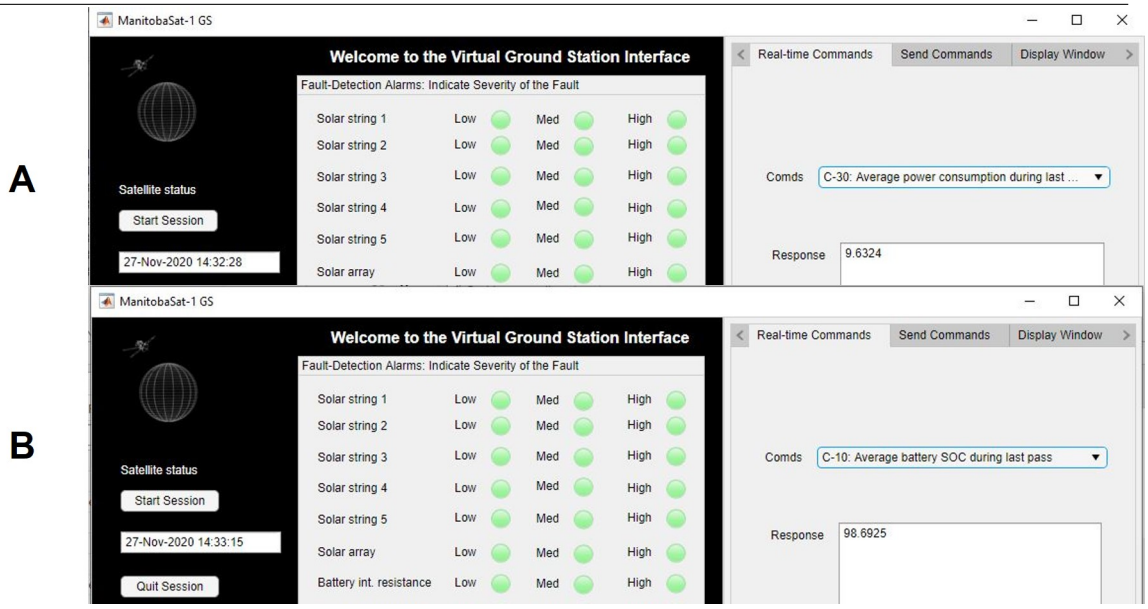

Figure 6.30: Using the real-time command manager in the VGS GUI to obtain data at any time from the VGS. Figures A and B show two examples where the average power consumption and state of charge are obtained (screenshots from Matlab's App Designer).

6.5.10 Event 12: Log Files of the Session

The log files for the session on the VGS and the simulation engine are obtained using the diary function in Matlab.

6.6 Summary

In this chapter, I gave an overview of the VGS's GUI layout and discussed its various features including TLE file retrieval, pass manager, embedded fault-management system, command store and the VSM. As part of the pass manager, the interface between STK and Matlab is utilized to develop a real-time look ahead orbit propagator to predict access times for the satellite from relevant ground stations. The command manager is split into two, namely, the real-time command manager and the 'send' commands. While the former lets the operator request and receive data at any time, the latter provides flexibility to the operator to create and store commands that are to be sent to the satellite at any time without having to wait for a pass. The VSM functions very similar to the actual satellite and gives operators an illusion of continuous communication with the satellite. This is made possible by using a real-time pacer, run-time objects with event listeners and other tools in Matlab.

The fault-management algorithms devised in Chapter 5 are implemented in the GUI to provide real-time health and diagnostic evaluations based on telemetry data and trends. The fault-detection alarm lamps make it easier for the operators to track any faults along with the fault-detection messages that allow the operator to let the VGS know that they have acknowledged a certain fault.

I also presented a unique testbed for testing the VGS interface with the VSM and demonstrating the communication between the VGS and a satellite (ManitobaSat-1). A simulation engine containing the ASM is used to emulate the satellite and the

communication is carried out using a serial data protocol. Through this testbed, the typical pass architecture is detailed with relevant screenshots from the interface.

6.6.1 Hypothesis One Evaluation

With the results from the testbed, the two hypotheses of this research are verified. The development of a VGS based on the VSM that mimics an actual satellite that gives the ground station operators the sense of a spacecraft in continual communication at all times is the first hypothesis. To prove this, I showed how the VGS can be used by the operators to obtain satellite data in real-time as though they were communicating with the actual satellite (Figure 6.30). The VGS also manages passes, uploads commands in queue when a pass occurs and downloads telemetry. I illustrated the autonomous operations of the VGS which reduces the burden on the operators and allows engineers to focus on more complex issues than mundane housekeeping.

6.6.2 Hypothesis Two Evaluation

I also hypothesized that SPC and fault-management tools for the VGS are beneficial. I demonstrated how the fault-management system implemented in the VGS was able to detect anomalies and indicates the findings to the operators in an easy manner by switching on the alarm lamps in the GUI and providing status messages to allow operators to keep track of the faults. Clearly, these tools have reduced the dependence of satellite monitoring on human resources. Together, the hypotheses in this thesis have been validated.

Chapter 7

Conclusions

With the recent shift from large geostationary spacecraft to constellations of smaller satellites, the need for increasingly efficient ground control facilities and operations is becoming evident. Reducing cost and time to operate ground stations require a shift from traditional satellite operations to autonomy. This thesis has described the development of a virtual ground station (VGS) with a real-time spacecraft model that mimics the actual satellite such that operators have a sense of continual communication with the satellite when in touch with this model. The virtual ground station also has a one of a kind fault-management system that uses Statistical Process Control to detect subtle faults.

7.1 Contributions

The primary contributions of this thesis are summarized below:

7.1.1 Power Subsystem Simulator

The power subsystem simulator I designed for a low earth orbit CubeSat using direct energy transfer provides a tool to student design teams and others to perform sizing of their battery, verify the sizing of other electrical components and the overall electrical design with minimal knowledge. The simulator estimates important parameters including the output voltages, state of charge of the battery and the solar array current.

Applying the simulator to a real satellite mission, ManitobaSat-1, provided confidence in the simulator framework as well as helped perform power analysis for the student CubeSat mission. This example illustrating how to use the simulator for a satellite was also discussed so that future researchers or designers could apply this simulator to other missions. A concious effort was made to keep the simulator design modular and less code oriented so that it would be easy to modify components as required and not involve extensive coding increasing complexity.

Results from the simulator for ManitobaSat-1 were discussed. The trends observed were analysed logically and discussed. This simulator forms the virtual ground station's spacecraft model and is also used later in the research project to mimic a spacecraft during testing and validation of the VGS.

7.1.2 Fault-Management System

One of the most important contributions of this research is the use of industrial Statistical Process Control (SPC) tools to develop fault-detection algorithms to reduce extensive human involvement in the trend analysis of telemetry data when compared to traditional satellite communications. The developed SPC techniques ease the time-consuming and labour intensive satellite housekeeping tasks for the operators. On

a large scale, these fault-detection algorithms are beneficial in monitoring satellite constellations. To my knowledge, this is the first fault-management system using SPC techniques for spacecraft health-monitoring.

Two types of faults, namely, the loss of solar string(s) and an increase in the battery's internal resistance are identified using the developed algorithms based on SPC. The process of development involved the custom application of control charts and identifying patterns in the control rule violations by the data. The steps in the development of the algorithms were laid out using logic flowcharts to facilitate understanding. Along with detection, the algorithms inform the operator of the severity of the fault and allow them to track the fault through status messages in the graphical user interface (GUI) of the VGS.

It was observed that not all types of faults can be detected using SPC techniques. Time domain feature extraction was identified as the appropriate method for noticing excessive power consumption. Several candidate features were chosen and the interaction amongst them in the presence of different faults was studied to devise an algorithm to find out when excessive power consumption is taking place. I undertook extensive testing with random test cases which provided a 100% detection rate and 92% accuracy for ManitobaSat-1 and a 100% detection rate and 85% accuracy for a hypothetical mission.

Together, positive results from all three algorithms indicate that the fault-management system is accurate.

7.1.3 Virtual Spacecraft Model (VSM)

One of the major features of the VGS is its mathematically based real-time spacecraft model or the VSM that allows the operators have a sense of continual communication with the actual spacecraft. The operators are able to request data from the VSM at

any time irrespective of a pass occurrence. The VSM provides predicted telemetry to the operators and can look at real-time predicted parameters of the satellite. The power subsystem simulator I developed acts as the VSM in the VGS. It runs in real-time using a real-time pacer and data from the VSM is updated regularly and compared to the actual telemetry when it is received.

7.1.4 VGS Graphical User Interface

I developed a graphical user interface (GUI) terminal for the VGS through which an operator communicates with the VGS to send commands to the satellite, access telemetry data, obtain results of the fault-management system and request real-time satellite data. Apart from these primary utilities of the GUI, it also enables automatic Two Line Element (TLE) file retrieval and has a real-time look ahead orbit propagator created in Systems Tool Kit that lets the users visualize the location of the mission in 2D and 3D. The propagator automatically updates the TLE files every few hours when new TLE files become available through CelesTrak's Space Track TLE Retriever application to maintain the accuracy of the propagation. With real-time propagation, access or pass times are obtained and enable the VGS to initiate contact with a satellite when a pass occurs.

The users can access automatically downloaded telemetry from the actual satellite on the VGS and predicted telemetry from the virtual satellite model (VSM) at any time. The GUI also has a command manager that is split into two, with the first providing real-time data to the operators irrespective of a pass occurring and the second one facilitating sending commands to the satellite. The commands to be sent to the satellite can be created at any time by the operator and stored in a queue which are automatically sent by the VGS when a pass occurs. The operator need not be present during regular operations and housekeeping activities are autonomous in

the VGS.

The fault-management systems analyses the received telemetry to identify any faults and indicates the results using coloured lamps in the GUI. To support the tracking of the detected faults and to reduce the burden on the operators, status messages are displayed letting the operator know of any faults in case they missed the lamps.

7.1.5 Virtual Ground Station Testbed (VGST)

A unique testbed was designed to demonstrate (simulate) the communication between the VGS and a satellite. A slightly modified power simulator model known as the actual satellite model (ASM) acts as the actual satellite. The VGS and the simulation engine with the ASM interact through serial communication which is a good representation of the actual interaction between a CubeSat's radio and a ground station. Results from the testbed were presented in the form of a demonstration of possible scenarios with images from the VGS and the simulation engine throughout the interaction. Pre-pass scenarios included calculating the access times using the orbit propagator and activation of the countdown timer. The VGS initiated communication with the ASM, downloaded telemetry and VSM data followed by fault-detection. Commands in the queue were sent to the simulation engine and were executed on the ASM at the specified date and time. Post-pass scenarios on the VGS included requesting data through real-time commands and TLE file retrieval.

7.2 Future Work

There are a number of improvements that could be made in the design of the VGS and its components that could further its applicability and accuracy. While most

of them focus on furthering the sophistication and abilities of the design, a few are directed towards the verification and validation of the VGS.

7.2.1 Power Subsystem Simulator

Though the power subsystem simulator is a realistic representation of an actual satellite's electrical design, there are a few modifications that could improve its accuracy. The solar array model in the simulator could include the ageing effects on its efficiency due to thermal reasons. Also, the declining trend of a battery's capacity due to use over time could improve the simulator and allow researchers to assess the performance of their battery better over a long period of time. The end of the life battery capacity could be estimated more accurately with such changes. Also, the internal resistance could be represented with a more complex model that would show how it would typically increase over time when factors such as temperature and chemical properties are considered.

Another area of interest could be having switches in the Simulink model for every subsystem and major components to represent their on/off state. Dedicated switches would simplify the model and allow the users to monitor and visualize the power consumption timeline better. Since the power simulator acts as the VSM in the VGS, having dedicated switches for different components would let the operator know when a major component or subsystem is on or off. The operator would be able to make informed decisions about the type of commands to send to the satellite in such situations unlike currently where the operator might send a command to turn off a certain component which is already off as there is no easy way of identifying the on/off states.

Additionally, the idea of this simulator could be extended to other subsystems such as the Attitude Determination and Control Subsystem (ADCS) to increase the

utility of the VGS. Currently, the simulator is restricted to the power subsystem but to simulate the operations of the whole satellite, it is important to include other subsystems and having them interact to accurately model the system.

7.2.2 Fault-Management System

An area of interest could be the application of the SPC algorithms developed for fault-detection to more varieties of faults without increasing the incidences of false alarms. The speed of the algorithms is another aspect that could be explored. Efficiency could be increased by using more sophisticated statistical methods such as process stability metrics and Bayesian probability theories to implement SPC.

Another research topic yet to be explored is the interplay of the different simultaneous faults in the system. Currently, the faults are assumed to be independent of each other and no interaction between them is considered. Studying coexistence and the effects of one or more faults on each other would be an interesting and useful research project for the future.

Integrating learning methods with the already existing statistical methods for fault-management could make the VGS an intelligent system. Fault handling could be more efficient and the accuracy could improve as SPC and learning could both contribute to anomaly detection. Alongside, learning could also introduce the scope for corrective action when faults occur.

7.2.3 Virtual Ground Station Interface Terminal

With learning incorporated and corrective actions defined, an automatic emailing system for the VGS that would send emails to the operators when faults that are beyond the scope of any corrective action taken by the VGS could help improve the

utility of the system. An in-built login system that only allows specific users to operate the system could provide enhanced security and reliability.

Having an embedded weather station in the VGS GUI that would track real-time weather conditions near the physical ground station to restrict operations when pre-defined extreme weather conditions such as thunderstorms and earthquakes exist could improve safety to infrastructure and the operators.

7.2.4 Verification and Validation of the System

The VGST helped simulate and test the communication between the VGS and a satellite model successfully. However, the research I conducted did not assess the usability of the VGS application by the operators. The comfort level of the operators, their suggestions and how they compare the performance of the VSM to the actual satellite would improve the utility of the VGS.

Finally, the primary goal of the development of the VGS was to reduce the burden on the ground station operators especially when there are satellite constellations involved. As such, expanding the testing environment to handle satellite constellations and not just individual satellites could extend the utility of the system.

Chapter 8

Bibliography

[1] N. H. Crisp, K. Smith, and P. Hollingsworth, "Launch and Deployment of Distributed Small Satellite Systems," *Acta Astronautica*, vol. 114, pp. 65–78, 2015.

[2] M. Sweeting, "Modern Small Satellites - Changing the Economics of Space," *Proceedings of the IEEE*, vol. 106, no. 3, pp. 343–361, 2018.

[3] A. A. Thompson, "Overview of the RADARSAT Constellation Mission," *Canadian Journal of Remote Sensing*, vol. 41, no. 5, pp. 401–407, 2015.

[4] N. Olsen, E. Friis-Christensen, R. Floberghagen, P. Alken, C. D. Beggan, A. Chulliat, E. Doornbos, J. T. Da Encarnação, B. Hamilton, G. Hulot, J. Van Den Ijssel, A. Kuvshinov, V. Lesur, H. Lühr, S. Macmillan, S. Maus, M. Noja, P. E. H. Olsen, J. Park, G. Plank, C. Püthe, J. Rauberg, P. Ritter, M. Rother, T. J. Sabaka, R. Schachtschneider, O. Sirol, C. Stolle, E. Thébault, A. W. Thomson, L. Tøffner-Clausen, J. Velímský, P. Vigneron, and P. N. Visser, "The Swarm Satellite Constellation Application and Research Facility (SCARF) and Swarm Data Products," *Earth, Planets and Space*, vol. 65, no. 11, pp. 1189–1200, 2013.

[5] S. Kidder, J. Kankiewicz, and T. Haar, "The A-Train: How Formation Flying Is Transforming Remote Sensing," *NASA and Colorado State University*. [Online]. Available: ftp://ftp.cira.colostate.edu/ftp/Kidder/CGAR/A-Train_paper.pdf

[6] L. A. Singh, W. R. Whittecar, M. D. DiPrinzio, J. D. Herman, M. P. Ferringer, and P. M. Reed, "Low Cost Satellite Constellations for Nearly Continuous Global Coverage," *Nature Communications*, vol. 11, no. 1, pp. 1–7, 2020.

[7] B. I. Edelson and L. Pollack, "Satellite Communications," *Science*, vol. 195, no. 4283, pp. 1125–1133, 1977.

[8] R. Holdaway, "A Program for the Autonomous Ground Station Control of Small Scientific Satellites," in *13th Triennial World Congress IFAC Proceedings*, vol. 29, no. 1. San Francisco, USA: Elsevier, 1996, pp. 7378–7383.

[9] Frost and Sullivan, "Uberisation In The Space Industry," 2018. [Online]. Available: https://www.forbes.com/sites/sarwantsingh/2018/04/18/uberisation-in-the-space-industry/?sh=204edd787efc

[10] V. M. Moreno and A. Pigazo, Eds., *Kalman Filter: Recent Advances and Applications*, 1st ed. Croatia: IntechOpen, 2009.

[11] V. Parthasarathy and P. Ferguson, "Modelling and Simulation of the Power Subsystem of a Low Earth Orbit CubeSat," *Aerospace Systems*, vol. 3, pp. 139–146, 2020.

[12] H. J. Kramer, *Observation of the Earth and its Environment*, 3rd ed. Germany: Springer, 1994.

[13] A. Freimann, A. Kleinschrodt, M. Schmidt, and K. Schilling, "Advanced Autonomy for Low Cost Ground Stations," *19th IFAC Symposium on Automatic*

Control in Aerospace, vol. 19, no. 1, pp. 388–392, 2013. [Online]. Available: http://dx.doi.org/10.3182/20130902-5-DE-2040.00054

[14] J. Anderson, "Autonomous Satellite Operations For CubeSat Satellites," Thesis, California Polytechnic State University, 2010. [Online]. Available: http://medcontent.metapress.com/index/A65RM03P4874243N.pdf%5Cnhttp://users.csc.calp anderson.pdf

[15] M. J. Bentley, A. C. Lin, and D. D. Hodson, "Overcoming Challenges to Air Force Satellite Ground Control Automation," in *IEEE Conference on Cognitive and Computational Aspects of Situation Management*. Savannah, GA: IEEE, 2017, pp. 1–7.

[16] A. Abedini, J. Moriarta, D. Biroscak, L. Losik, and R. F. Malina, "Low-cost, Autonomous, Ground Station Operations Concept and Network Design for EUVE and Other Earth-Orbiting Satellites," *International Telemetering Conference Proceedings*, vol. 31, pp. 304–311, 1995.

[17] K. Colton and B. Klofas, "Supporting the Flock: Building a Ground Station Network for Autonomy and Reliability," in *30th Annual AIAA/USU Conference on Small Satellites*, 2016, pp. 1–7.

[18] S. Bernier and M. Barbeau, "A Virtual Ground Station Based on Distributed Components for Satellite Communications," in *15th Annual Small Satellite Conference*, Logan, Utah, 2001, pp. 1–15.

[19] B. H. Cho and F. C. Y. Lee, "Modeling and Analysis of Spacecraft Power Systems," *IEEE Transactions on Power Electronics*, vol. 3, no. 1, pp. 44–54, 1988.

[20] J. R. Lee, B. H. Cho, S. J. Kim, and F. C. Lee, "Modeling and Simulation of Spacecraft Power Systems," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, no. 3, pp. 295–304, 1988.

[21] A. Capel, J. Ferrante, J. Cornet, and P. Leblanc, "Power System Simulation of Low Orbit Spacecraft: the Eblos Computer Programme," in *IEEE Annual Power Electronics Specialists Conference*. Cambridge, USA: IEEE, 1982, pp. 272–285.

[22] P. Bauer, "Computer Simulation of Satellite Electric Power Systems," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-5, no. 6, pp. 934–942, 1969.

[23] S. P. Kuksenko, "Preliminary results of TUSUR University project for design of spacecraft power distribution network: EMC simulation," *IOP Conference Series: Materials Science and Engineering*, vol. 560, 2019.

[24] C. W. Melone, "Preliminary Design, Simulation, and Test of the Electrical Power Subsystem of the TINYSCOPE Nanosatellite," Thesis, Naval Postgraduate School, 2009.

[25] S. Y. Kim, J. F. Castet, and J. H. Saleh, "Satellite Electrical Power Subsystem: Statistical Analysis of On-Orbit Anomalies and Failures," in *IEEE Aerospace Conference Proceedings*. IEEE Xplore, 2011, pp. 1–12.

[26] O. Khan, M. S. El Moursi, H. H. Zeineldin, V. Khadkikar, and M. A. Hosani, "Benchmark Model for Multi-Orbital Transient Analysis of Satellite Electrical Power Subsystem," *IET Renewable Power Generation*, vol. 14, no. 2, pp. 286–296, 2020.

[27] D. T. M. Krishna, S. S. Paramahamsa, and N. Ashwini, "Modeling and Simulation of Electrical Power Subsystem of Nanosatellite using MATLAB," *Helix*, vol. 8, no. 2, pp. 3031–3035, 2018.

[28] J. Harrison, "Integration of CAE Tools for Complete System Prototyping," in *International Off-Highway & Powerplant Congress & Exposition*. Wisconsin: SAE International, 1998, pp. 1–8.

[29] S. Saraf, P. Melanson, M. Tafazoli, C. S. Agency, S. T. Branch, F. Computers, and S. Section, "Use of an Operations Simulator for Small Satellites," in *15th Annual AIAA/USU Conference on Small Satellites*, 2001, pp. 1–8.

[30] M. Owen, *SPC and Continuous Improvement*, 1st ed. New York: Springer Verlag, 1989.

[31] J. Swift, "Development of Knowledge Based Expert System for Control Chart Pattern Recognition and Analysis," thesis, Oklahoma State University, 1987.

[32] C.-S. Cheng, "Group Technology and Expert Systems Concepts Applied to Statistical Process Control in Small-Batch Manufacturing," Thesis, Arizona State University, 1989. [Online]. Available: https://dl.acm.org/doi/book/10.5555/75919

[33] F. Zorriassatine and J. D. Tannock, "A Review of Neural Networks for Statistical Process Control," *Journal of Intelligent Manufacturing*, vol. 9, no. 3, pp. 209–224, 1998.

[34] H. Sohn, J. A. Czarnecki, and C. R. Farrar, "Structural Health Monitoring Using Statistical Process Control," *Journal of Structural Engineering*, vol. 126, no. 11, pp. 1356–1363, 2000.

[35] F. Bingqing, H. Shaolin, L. Chuan, and M. Yangfan, "Anomaly Detection of Spacecraft Attitude Control System Based on Principal Component Analysis," in *2017 29th Chinese Control And Decision Conference (CCDC)*. IEEE, 2017, pp. 1220–1225.

[36] K. Bouzenad and M. Ramdani, "Multivariate Statistical Process Control Using Enhanced Bottleneck Neural Network," *Algorithms*, vol. 10, no. 49, pp. 1–23, 2017.

[37] G. Biswas, H. Khorasgani, G. Stanje, A. Dubey, S. Deb, S. Ghoshal, and Q. Systems, "An Approach To Mode and Anomaly Detection with Spacecraft Telemetry Data," *International Journal of Prognostics and Health Management*, vol. 7, no. 1, p. 18, 2016.

[38] H. Jiang, K. Zhang, J. Wang, X. Wang, and P. Huang, "Anomaly Detection and Identification in Satellite Telemetry Data Based on Pseudo-Period," *Applied Sciences*, vol. 10, no. 103, pp. 1–20, 2020.

[39] J. Lianxiang, L. Huawang, Y. Genqing, and Y. Qinrong, "A Survey of Spacecraft Autonomous Fault Diagnosis Research," *Journal of Astronautics*, vol. 30, no. 4, pp. 1320–1326, 2009.

[40] W. A. Murtada and E. A. Omran, "Robust Anomaly Identification Algorithm for Noisy Signals: Spacecraft Solar Panels Model," *Neural Computing and Applications*, vol. 32, pp. 12 281–12 294, 2020.

[41] I. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer, 2002.

[42] Christopher Bishop, *Pattern Recognition and Machine Learning*, 1st ed., M. Jordan, J. Kleinberg, and B. Scholkopf, Eds. New York: Springer, 2006.

[43] N. Lai and J.-C. Petkovich, "The Difference Between Machine Learning & SPC (and Why it Matters)," Kitchener. [Online]. Available: https://verhaert.com/difference-machine-learning-deep-learning/

[44] Shewhart and Mark, "Application of Machine Learning and Expert Systems to Statistical Process Control (SPC) Chart Interpretation," in *NASA. Johnson Space Center, Second CLIPS Conference Proceedings*. United States: NASA, 1991, pp. 123–138.

[45] M. Shewhart, "Interpreting Statistical Process Control (SPC) Charts Using Machine Learning and Expert System Techniques," in *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference*. Dayton, OH, USA: IEEE, 1992, pp. 1001–1006.

[46] B. A. Beabout, "Statistical Process Control: An Application in Aircraft Maintenance Management," Thesis, Air Force Institute of Technology, 2003. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a413139.pdf

[47] S. Kassicieh, S. Yourstone, and W. Zimmer, "SPC-Pro: An Expert System Approach for Variables Control Charts," *Quality Engineering*, vol. 7, no. 1, pp. 89–104, 1994.

[48] J. Hodder, "The Use of Statistical Process Control in Deep Space Network Operations," in *SpaceOps Conference*. Houston, TX; United States: Jet Propulsion Lab., California Inst. of Tech., 2002, pp. 1–12.

[49] S. F. Lee, E. D. Boskin, C. J. Spanos, H. C. Liu, and E. H. Wen, "RTSPC: A Software Utility for Real-Time SPC and Tool Data Analysis," *IEEE Transactions on Semiconductor Manufacturing*, vol. 8, no. 1, pp. 17–25, 1995.

[50] D. Park, S. Kim, Y. An, and J. Y. Jung, "Lired: A Light-Weight Real-Time Fault Detection System for Edge Computing Using LSTM Recurrent Neural Networks," *Sensors*, vol. 18, no. 7, 2018.

[51] B. D. Fulcher, M. A. Little, and N. S. Jones, "Highly Comparative Time-Series Analysis : The Empirical Structure of Time Series and their Methods," *Journal of the Royal Society Interface*, vol. 10, no. 83, pp. 1–20, 2013.

[52] B. Fulcher, "Feature-Based Time-Series Analysis, Report," *Cornell University*, no. September Issue, 2017.

[53] A. Nanopoulos, R. O. B. Alcock, and Y. Manolopoulos, "Feature-based Classification of Time-series Data," *International Journal of Computer Research*, vol. 10, no. January, pp. 49–61, 2001.

[54] X. Wang, K. Smith, and R. Hyndman, "Characteristic-Based Clustering for Time Series Data," *Data Mining and Knowledge Discovery*, vol. 13, no. 3, pp. 335–364, 2006.

[55] A. Rahimi and A. Saadat, "Fault Isolation of Reaction Wheels Onboard Three-Axis Controlled In-Orbit Satellite Using Ensemble Machine Learning," *Aerospace Systems*, vol. 3, pp. 119–126, 2020.

[56] I. Bandyopadhyay, P. Purkait, and C. Koley, "Performance of a Classifier Based on Time-domain Features for Incipient Fault Detection in Inverter Drives," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 3–14, 2019.

[57] H. Helmi and A. Forouzantabar, "Rolling Bearing Fault Detection of Electric Motor Using Time Domain and Frequency Domain Features Extraction and ANFIS," *IET Electric Power Applications*, vol. 13, no. 5, pp. 662–669, 2019.

[58] S. Shahabi, "Low Air Pressure Partial Discharge Recognition using Statistical Analysis of Time-domain Pulse Features," Thesis, University of Manitoba, 2019.

[59] B. D. Fulcher and N. S. Jones, "A Computational Framework for Automated Time-Series Phenotyping Using Massive Feature Extraction," *Cell Systems*, vol. 5, no. 5, pp. 527–531, 2017.

[60] T. Wang and S. Nanda, "A Tutorial on Feature Extraction Methods," in *First European PHM conference*. PHM Society, 2012, pp. 1–41. [Online]. Available: http://www.gis.usu.edu/ doug/RS5750/PastProj/FA2002/KelliTaylor.pdf

[61] C. S. Clark and A. L. Mazarias, "Power System Challenges for Small Satellite Missions," in *4S Symposium: Small satellites, Systems and Services*, vol. 625. European Space Agency, 2006.

[62] D. M. Erb, S. a. Rawashdeh, and J. E. Lumpp, "Evaluation of Solar Array Peak Power Tracking Technologies for CubeSats," in *25th Annual Proceedings of the AIAA/USU Conference on Small Satellites*. Logan, Utah: Semantic Scholar, 2011. [Online]. Available: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1141&context=smallsat

[63] J. Gonzalez-Llorente, R. Hurtado, S. Sanchez-Sanjuan, and E. I. Ortiz-rivera, "Evaluation of Techniques for Power Regulation on Nanosatellites," *10th European Space Power Conference - ESPC 2014*, no. April, pp. 1 – 6, 2014.

[64] A. Singal, "The Internal Resistance of a Battery," *Indian Space Research Organization*, pp. 1–3, 2013. [Online]. Available: http://arxiv.org/abs/1308.4913

[65] V. Pop, H. J. Bergveld, D. Danilov, P. P. L. Regtien, and P. H. L. Notten, *Battery Management Systems. Accurate State-of-Charge Indication for Battery-Powered Applications*. Springer, 2008, vol. 9.

[66] A. Yahyaabadi, M. Driedger, N. Turenne, S. Connell, V. Parthasarathy, A. Carvey, V. Platero, J. Campos, M. Rososhansky, and P. Ferguson, "ManitobaSat-1: A Novel Approach for Technology Advancement," *IEEE potentials*, vol. 39, no. July, pp. 17–23, 2020.

[67] AA Portable Power Corp, "LFP-18650HT Battery Product Specification," California, 2006. [Online]. Available: https://www.batteryspace.com/prodspecs/LFP18650.pdf

[68] Spectrolab, "29.5 % NeXt Triple Junction (XTJ) Solar Cell Datasheet," 2020. [Online]. Available: https://www.spectrolab.com/photovoltaics/XTJ-CIC_Data_Sheet.pdf

[69] C. Peat, "ISS - Orbit," 2020. [Online]. Available: https://www.heavens-above.com/orbit.aspx?satid=25544

[70] E. Theroux, Y. Galarneau, and M. Chen, "Control Charts for Short Production Runs in Aerospace Manufacturing," *SAE International Journal of Materials and Manufacturing*, vol. 7, no. 1, pp. 65–72, 2014.

[71] D. C. Montgomery, *Introduction to Statistical Quality Control*, 6th ed. U.S.A: John Wiley & Sons, Inc., 2009.

[72] W. Shewhart, "Quality Control Charts," *The Bell System Technical Journal*, vol. 5, no. 4, pp. 593–603, 1926.

[73] L. Nelson, "The Shewhart Control Chart-Tests for Special Causes," *Journal of Quality Technology*, vol. 16, no. 4, pp. 237–239, 1984.

[74] NCSS Statistical Software, "Individuals and Moving Range Charts," pp. 1–20, 2020. [Online]. Available: http://ncss.wpengine.netdna-cdn.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Individuals_and_Moving_Range_Charts.pdf

[75] J. A. Marsteller, M. M. Huizinga, and L. A. Cooper, "Statistical Process Control:Possible Uses to Monitor and Evaluate Patient-Centered Medical Home Models," *AHRQ Publication*, vol. PCMH Resea, 2013.

[76] KnowWare International Inc., "Changing Control Chart Rules in QI Macros," Colorado, 2020. [Online]. Available: https://www.qimacros.com/free-excel-tips/control-chart-rules/

[77] PowerStream, "Battery impedance and resistance," 2019. [Online]. Available: https://www.powerstream.com/internal-resistance.htm

[78] Z. Ouyang, X. Sun, J. Chen, D. Yue, and T. Zhang, "Multi-View Stacking Ensemble for Power Consumption Anomaly Detection in the Context of Industrial Internet of Things," *IEEE Access*, vol. 6, pp. 9623–9631, 2018.

[79] M. Tahir, A. Hussain, Saeed Badshah, and Q. Javaid, "Rule-Based Identification of Bearing Faults Using Central Tendency of Time Domain Features," *Journal of Engineering and Applied Sciences*, vol. 35, no. 2, pp. 111–121, 2016.

[80] The Concise Encyclopedia of Statistics, *Coefficient of Skewness*. New York: Springer, 2008, pp. 92–95. [Online]. Available: https://doi.org/10.1007/978-0-387-32833-1_64

[81] MathWorks, "Signal Features," 2020. [Online]. Available: https://www.mathworks.com/help/predmaint/ug/signal-features.html#: :text=Impulse Factor — Compare the height,mean level of the signal.&text=Crest Factor — Peak value divided,the signal root mean squared.

[82] S. S. Arnold, R. Nuzzaci, and A. Gordon-Ross, "Energy Budgeting for CubeSats with an Integrated FPGA," in *IEEE Aerospace Conference Proceedings*. Big Sky, Montana: IEEE, 2012, pp. 1–14.

[83] Small Satellites, "Two Line Element," 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/39364-two-line-element

[84] T. Kelso, "Celestrak Space Track TLE Retriever," 2019. [Online]. Available: https://www.celestrak.com/SpaceTrack/TLERetriever3Help.php

[85] Analytics Graphics Inc., "STK Programming Interface," 2020. [Online]. Available: http://help.agi.com/stkdevkit/

[86] P. A. Ferguson, "Distributed Estimation and Control Technologies for Formation Flying Spacecraft," Thesis, Massachusetts Institute of Technology, 2003.

[87] Y. Liu, Q. Zhang, S. Han, and D. Zou, "A GEO Satellite Positioning Testbed Based on Labview and STK," *Lecture Notes in Electrical Engineering*, vol. 571, pp. 1892–1899, 2020.

[88] M. K. Wali, M. H. Baqir, and M. S. Naghmash, "Development of a Serial Communication Protocol for Satellite Attitude Determination and Control System Simulator," *Diyala Journal of Engineering Sciences*, vol. 8, no. 4, pp. 574–588, 2015.

[89] BSGStudio, "Computer Image," 2020. [Online]. Available: https://all-free-download.com/free-vector/download/computer_311799.html

[90] "Laptop Image," 2020. [Online]. Available: https://pngio.com/PNG/a84595-laptop-animation-png.html

[91] Analog Devices Inc., "8-Channel, 1 MSPS, 8-/10-/12-Bit ADCs with Sequencer in 20-Lead TSSOP," 2014. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7908_7918_7928.pdf

[92] Measurement Computing, "Analog to Digital Conversion." [Online]. Available: https://www.mccdaq.com/PDFs/specs/Analog-to-Digital.pdf

[93] B. Shoelson, "Countdown," 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/18165-countdown

[94] MathWorks, "What Is an S-Function ?" 2020. [Online]. Available: https://www.mathworks.com/help/simulink/sfg/what-is-an-s-function.html

[95] G. Vallabha, "Real-Time Pacer for Simulink," 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/29107-real-time-pacer-for-simulink

[96] Systems Tool Kit, "MATLAB: Connect via COM," 2020. [Online]. Available: https://help.agi.com/stk/index.htm#MATLAB_COM.htm%3FTocPath%3DIntegrating%252

[97] M. Mahooti, "SGP4," 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/62013-sgp4

[98] D. Koblick, "Convert Keplerian Orbital Elements to a State Vector," 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/35455-convert-keplerian-orbital-elements-to-a-state-vector?s_tid=FX_rc1_behav

Appendix A

Datasets for Power Consumption Fault Detection

This appendix contains the datasets or the test case parameters for estimating the accuracy of the power consumption increase detection applied to ManitobaSat-1 and the hypothetical mission. This data is relevant to Sections 5.4.2 and 5.4.3 of Chapter 5.

A.1 Datasets for ManitobaSat-1

Case	Increase	Duration of occurrence	Start	End	Expected score	Actual score	Detected
1	0.245270903	502	10240	10742	1	1	Yes
2	1.6064732	462	10429	10891	1	2	Yes
3	1.806080939	356	1815	2171	1	2	Yes
4	0.605122188	447	4020	4467	1	1	Yes
5	1.869478413	193	8298	8491	1	1	Yes
6	0.49321802	863	2346	3209	1	1	Yes
7	1.103961108	1000	4976	5976	1	1	Yes
8	0.284739877	331	5570	5901	1	1	Yes
9	0.800602725	345	13106	13451	1	1	Yes
10	1.429739812	968	6132	7100	1	2	Yes
11	4.52325899	451	6021	6472	2	2	Yes
12	3.716717655	746	10395	11141	2	2	Yes
13	4.273652452	481	6011	6492	2	2	Yes
14	4.869033973	637	11897	12534	2	3	Yes
15	2.829035949	679	8238	8917	2	2	Yes
16	4.890405397	223	7007	7230	2	2	Yes
17	3.564769029	226	12666	12892	2	2	Yes
18	4.653166816	523	10945	11468	2	2	Yes
19	2.445395069	677	3649	4326	2	2	Yes
20	3.336968634	868	2747	3615	2	2	Yes
21	7.814947521	367	7465	7832	3	3	Yes
22	7.864264611	377	3502	3879	3	3	Yes
23	7.783018161	208	4192	4400	3	2	Yes
24	6.774750602	323	8903	9226	3	3	Yes
25	7.395110773	576	9112	9688	3	3	Yes
26	7.387864957	348	8412	8760	3	3	Yes
27	5.337387135	588	11730	12318	3	3	Yes
28	7.762370324	573	3887	4460	3	3	Yes
29	6.957559882	930	7138	8068	3	3	Yes
30	7.922574451	317	1557	1874	3	3	Yes
31	0.797580799	4520	4089	8609	2	2	Yes
32	1.690787486	2529	1425	3954	2	2	Yes
33	1.783790448	2132	7554	9686	2	2	Yes
34	1.520791229	3311	4179	7490	2	2	Yes
35	1.963383392	3197	4843	8040	2	2	Yes
36	0.482254198	2978	3455	6433	2	2	Yes
37	1.810101195	2741	3418	6159	2	2	Yes
38	0.935964034	4124	1581	5705	2	2	Yes
39	1.76837972	2249	5080	7329	2	2	Yes
40	0.254700243	4260	7701	11961	2	2	Yes
41	2.757138533	2600	760	3360	3	3	Yes
42	3.655732205	3211	8252	11463	3	3	Yes
43	3.461550507	3154	676	3830	3	3	Yes
44	2.641074472	3632	4518	8150	3	3	Yes
45	4.702889618	2168	4879	7047	3	3	Yes
46	3.613437582	2402	5951	8353	3	3	Yes
47	4.572104577	2594	1712	4306	3	3	Yes
48	2.184133303	3983	205	4188	3	3	Yes
49	2.87330881	4922	8412	13334	3	3	Yes
50	2.731051791	4047	1517	5564	3	3	Yes
51	7.496263884	4915	2401	7316	5	5	Yes
52	7.118232885	2117	6780	8897	5	5	Yes
53	7.008092512	2111	37	2148	5	5	Yes
54	5.427393062	4588	3037	7625	5	5	Yes
55	6.595074611	3567	6244	9811	5	5	Yes
56	5.999097661	3240	4558	7798	5	5	Yes
57	7.951765353	2173	4362	6535	5	5	Yes
58	7.373961746	3783	3406	7189	5	5	Yes
59	7.705408719	2279	3510	5789	5	5	Yes
60	7.660884844	3972	7530	11502	5	5	Yes

Figure A.1: Test cases for ManitobaSat-1 to estimate the accuracy of the power consumption fault detection algorithm

A.2 Datasets for the Hypothetical Mission

Case	Increase	Duration of occurrence	Start	End	Expected score	Actual score	Detected
1	0.493162245	373	428	801	1	1	Yes
2	1.427716048	173	5500	5673	1	1	Yes
3	1.065250882	387	6496	6883	1	2	Yes
4	0.717348707	932	67	999	1	2	Yes
5	1.216154681	463	4085	4548	1	1	Yes
6	0.238149778	734	6101	6835	1	1	Yes
7	1.577784423	823	2713	3536	1	2	Yes
8	0.559651972	462	7916	8378	1	1	Yes
9	0.425656678	515	180	695	1	1	Yes
10	0.327836285	754	8833	9587	1	1	Yes
11	3.377997099	665	4005	4670	2	2	Yes
12	3.20583521	771	10507	11278	2	2	Yes
13	4.636396422	668	10646	11314	2	2	Yes
14	3.902414443	811	9884	10695	2	2	Yes
15	2.679696771	236	9111	9347	2	2	Yes
16	3.27340996	768	10305	11073	2	2	Yes
17	2.889597279	415	6047	6462	2	2	Yes
18	3.657744884	694	6564	7258	2	2	Yes
19	2.483523915	505	4060	4565	2	2	Yes
20	2.240793883	351	10938	11289	2	2	Yes
21	7.469496232	649	10578	11227	3	3	Yes
22	6.467387961	365	7461	7826	3	2	Yes
23	7.743475346	675	3548	4223	3	3	Yes
24	6.47153579	496	924	1420	3	2	Yes
25	6.775878573	898	6087	6985	3	2	Yes
26	5.898413206	913	8738	9651	3	3	Yes
27	7.327437366	659	6130	6789	3	3	Yes
28	5.899122896	616	10911	11527	3	3	Yes
29	6.450238911	732	5281	6013	3	3	Yes
30	7.239562325	329	2734	3063	3	3	Yes
31	1.311018443	3193	5889	9082	2	2	Yes
32	0.268975473	2288	6637	8925	2	2	Yes
33	0.912828106	3597	6326	9923	2	2	Yes
34	0.812063434	3992	5092	9084	2	2	Yes
35	0.601424481	3666	3187	6853	2	2	Yes
36	1.298476168	3632	3629	7261	2	2	Yes
37	0.714885945	4637	6556	11193	2	2	Yes
38	1.651176708	4908	3670	8578	2	2	Yes
39	0.403164083	3951	1337	5288	2	2	Yes
40	1.742484503	4829	725	5554	2	3	Yes
41	2.733558321	2042	1396	3438	3	3	Yes
42	2.764564637	5649	2206	7855	3	3	Yes
43	2.114102071	4603	4899	9502	3	3	Yes
44	2.251251224	2045	2194	4239	3	3	Yes
45	3.238296327	3664	2781	6445	3	3	Yes
46	3.398019538	4193	3197	7390	3	3	Yes
47	3.523077662	5125	1740	6865	3	5	Yes
48	3.651505644	4299	1224	5523	3	5	Yes
49	3.665710917	2856	3295	6151	3	3	Yes
50	2.20930949	5442	1251	6693	3	3	Yes
51	7.949900997	2572	4967	7539	5	5	Yes
52	7.643398404	2281	2868	5149	5	5	Yes
53	7.216264383	4935	4898	9833	5	5	Yes
54	6.600440183	4243	2080	6323	5	5	Yes
55	6.652266942	2084	4421	6505	5	5	Yes
56	7.501915216	5346	199	5545	5	5	Yes
57	7.736517031	2379	5106	7485	5	5	Yes
58	6.566599458	2065	776	2841	5	5	Yes
59	6.779524116	4887	223	5110	5	5	Yes
60	5.528508021	2929	4291	7220	5	5	Yes

Figure A.2: Test cases for the hypothetical mission to estimate the accuracy of the power consumption fault detection algorithm

Appendix B

Graphical Interface and Testbed

This appendix discusses the different components of the Virtual Ground Station (VGS) Graphical User Interface (GUI) and the VGS testbed (VGST).

B.1 GUI Layout

B.1.1 Main Screen

The main screen is the part of the application that always appears when using the GUI. It contains the session control which allows the user to start and quit a session. Starting a session initiates the orbit propagator window to show up. Realistically, a session will always be ongoing since I want to use the GUI for real time communications. Quitting the session causes the application to reset all the settings and can be used to restart the GUI. A clock displays the time in the time zone (currently Central Daylight Time, GMT-5) where the GUI application is being used. This clock is implemented in the startupFcn of the GUI using a timer that runs at a fixed rate and uses a custom function to display the time every second. The startupFcn (Appendix

C.3.1) executes as soon as the application is started.

The main screen also hosts the alarm lamps for various faults mentioned in Chapter 6. These are connected to the fault-detection messages tab which are also discussed in chapter 6.

B.1.2 TLE and Orbital Elements

CelesTrak allows users with a Space Track account to download TLEs using its Space Track TLE Retriever software (Figure B.1) [84]. The settings on this software are configured to automatically download and archive the TLE files in a specified folder. The button, Open TLE file, opens up the folder with the TLE files with the latest file at the top of the list for the user to choose. To obtain the latest Orbital elements, the Process TLE button extracts the six orbital elements from the selected TLE file. The code for this extraction is adapted from Mathworks website [83] and is provided in Appendix C.1.

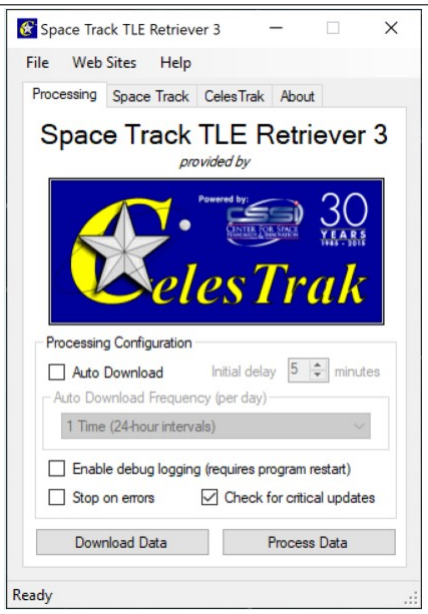

Figure B.1: Celestrak's Space Track TLE Retriever [84]

B.1.3 TLE Files

This tab lets the user access downloaded TLE files and process them to obtain the orbital elements. In Figure B.2, one can choose the from and the to dates such that all TLE files obtained during this time period are displayed in the Filenames text area. This function narrow down the number of files one needs to look into for various purposes. Again, as for the orbital elements on the Main Screen, pushing the Process old TLE gives the corresponding orbital elements. The function (processtles) for this extraction is adapted from Mathworks website [83] and is provided in Appendix C.1.

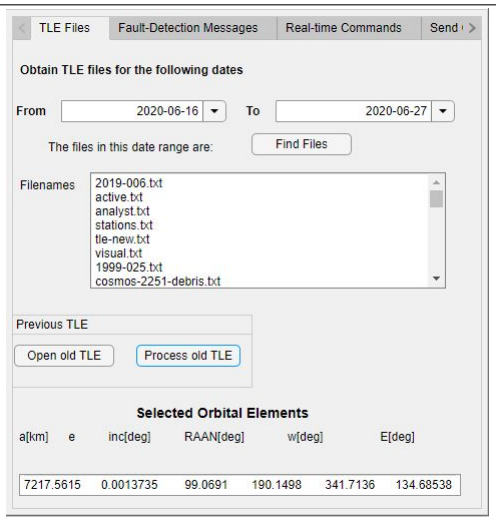

Figure B.2: Opening and processing TLE files in the GUI to obtain orbital elements (Screenshots from Matlab's App Designer)

B.1.4 Fault-detection Messages

This tab displays the alarm messages triggered by the fault alarm lamps on the main screen. Each message pertaining to a fault detected by the fault-management system from Chapter 5 appears as soon a lamp switches on. It describes where the fault is and gives an option to close the warning. Once the operator has addressed the messages, they disappear.

B.1.5 Display Window

This tab (Figure B.3) contains a text area called DisplayScreen that shows all the messages displayed on the command window of the main Matlab window. This is done using the diary command in Matlab.

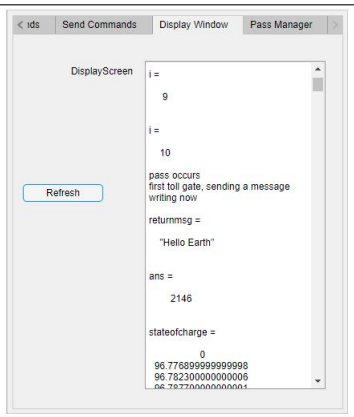

Figure B.3: The Display Window tab (screenshot from Matlab's App Designer)

B.1.6 Countdown Timer

The countdown function that appears 60 seconds before each pass is written by Brett Shoelson and can be found in [93]. When the time is 60 seconds away from a pass, the countdown function is triggered using an if statement. For this, I take the current time and compare it to the accessTimes from the STKOrbitProp (Appendix C.2) script results and when there is a 60 second gap, the timer appears.

B.2 Creating Send Type Commands in the Command Manager

The command manager is split into two tabs depending on the type of command, that is, the real-time commands (Figure B.4) and the send type commands. To create a send type command, the operator needs to select the command, the date and the time (using spinners) at which the command needs to be executed and then click on the Done button. The created command should show up on the Created command

text area (Figure B.5). The commands that are waiting to be uploaded can be seen in the Commands in queue text area (Figure B.5) and once they are uploaded, they appear in the Last uploaded commands text area (Figure B.5).

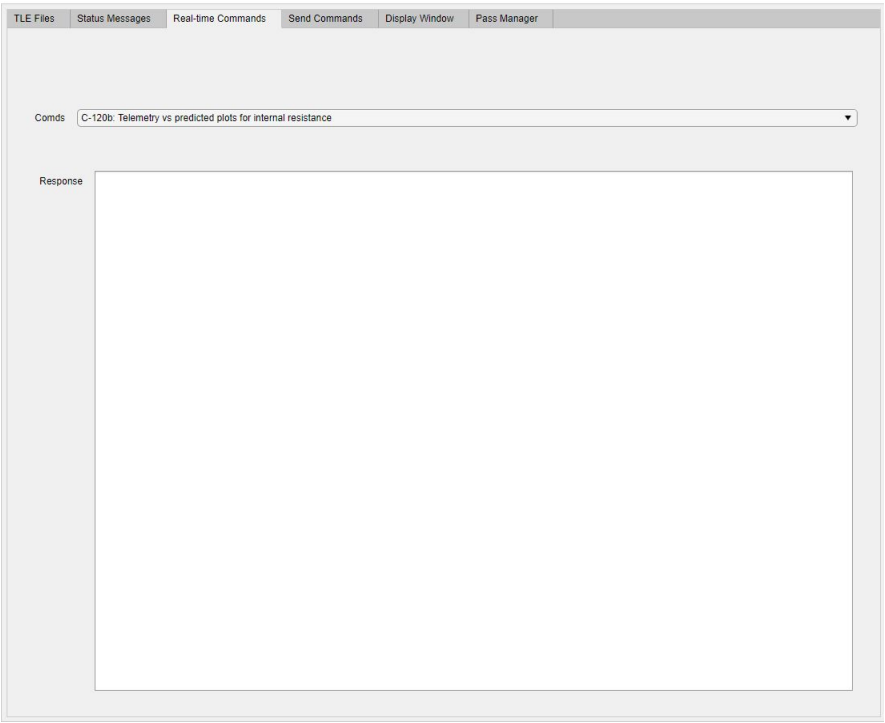

Figure B.4: The Real-time Commands tab (screenshot from Matlab's App Designer)

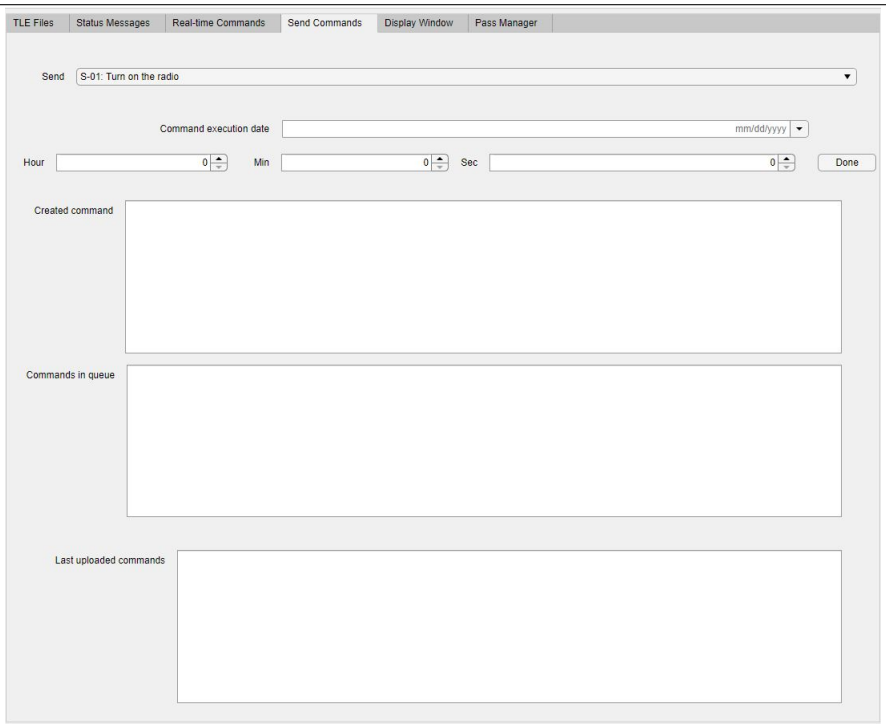

Figure B.5: The Send Commands tab (screenshot from Matlab's App Designer)

The VGS's command store is basically a database that stores any send type commands given during a non-pass duration and uploads them when a pass occurs. The operator can use the real-time type commands at any time to obtain information and data based on downloaded telemetry or historic data and predicted telemetry. So, the operator would not have to delay getting information in real time from the 'satellite'. The command manager consists of a set of pre-defined real-time and send type commands that the operator can choose from. This list can be expanded when needed. For this thesis, the list of the pre-defined commands is given in Tables B.1 and B.2. Telemetry can be used as a verification for the send type commands wherein if a command has been uploaded and appropriate action has been taken by the satellite, the command manager is working as required.

Command ID	Command description	Expected response type
C-10	Average battery state of charge during the previous pass	Number
C-20	Battery depth of discharge during the previous pass	Number
C-30	Average power consumption during the previous pass in W	Number
C-40	Number of passes today	Number
C-50	Pass intervals for today	Date and time
C-60	Plot of the battery voltage vs the state of charge in the previous pass	Graph
C-70	Average internal resistance of the battery during the previous pass in Ω	Number
C-80	Average solar array current generated so far in A	Number
C-90	Minimum SOC so far in real time	Number
C-100	Maximum battery's internal resistance in Ω so far in real time	Number
C-110	Telemetry vs predicted plots for power consumption	Graph
C-120	Telemetry vs predicted plots for the battery's internal resistance	Graph
C-130	Telemetry vs predicted plots for the solar array current	Graph

Table B.1: List of pre-defined real-time commands in the VGS's command manager. Note: commands requiring data during the previous pass include the data obtained from the latest pass's telemetry

Command ID	Command description
S-01	Turn on the radio at the given time
S-02	Turn on the camera at the given time
S-03	Turn off the radio at the given time if its on

Table B.2: List of pre-defined send type commands in the VGS's command manager

B.3 Miscellaneous Images

This section contains descriptions and images of different GUI's components.

B.3.1 Orbit Propagator and The Pass Manager Tab

The Systems Tool Kit (STK) orbit propagator GUI window (Figure B.6) provides 3D and 2D views of the satellite's position and shows the current position and velocity in real time. The pass manager is shown in Figure B.7 where the passes for a 24 hour period are displayed in the Passes text area when the refresh button is pressed.

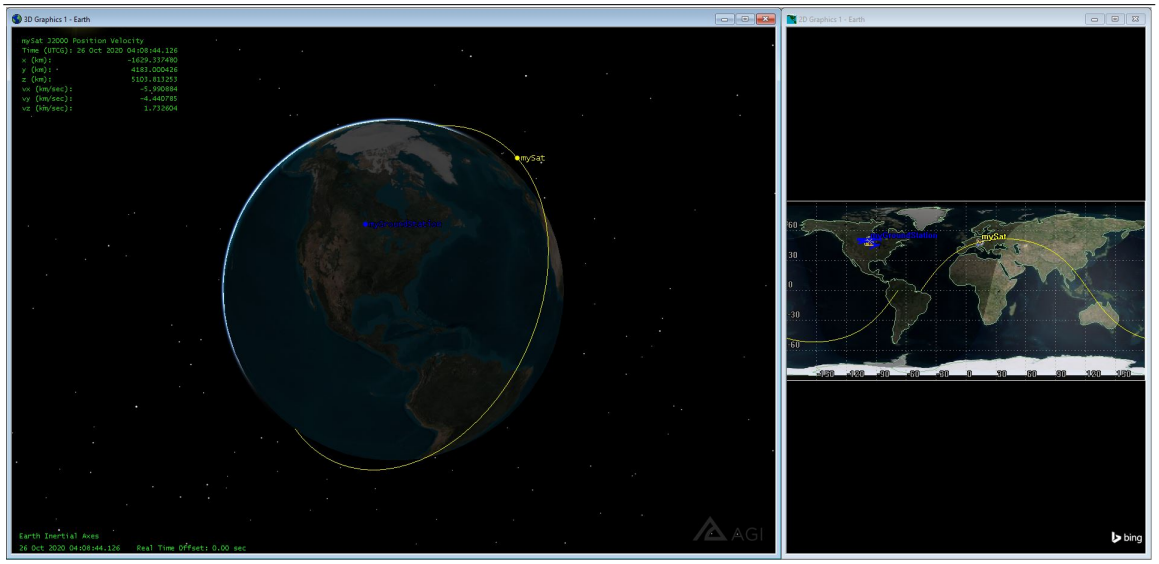

Figure B.6: The STK GUI window showing the real time position and velocity of the satellite

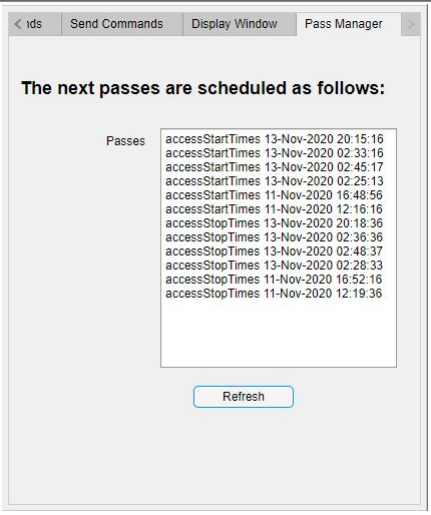

Figure B.7: The pass manager tab (screenshot from Matlab's App Designer)

B.4 Implementation of the GUI

There are many steps in the implementation of the GUI with its several functions and features to perform necessary actions and trigger events. I discuss the steps below.

B.4.1 Master Script of the GUI

The Start Session button on the Main Screen of the GUI triggers the Master Script written outside of the App Designer. This includes the main framework of the VGS system and the implementation of the VSM, the STK propagator, the telemetry store, parts of the command manager and the fault-management system.

Preliminary Steps

Before the implementation of the VSM, two preliminary steps are carried out in the Master Script: creating two text files for the telemetry to be downloaded and the sessionfile used in the Display Window tab (Figure B.3) of the GUI. Then, I create null matrices (for initialization) for each of the telemetry parameters (the solar string currents, the total solar array current, the eclipse flag, the power consumption, the battery current and voltage and the timestamp).

Virtual Satellite Model

The power subsystem simulator developed in Chapters 3 and 4 acts as the virtual satellite model (VSM). All the necessary input to configure the Simulink blocks as described in Chapter 4 are provided through the Master Script to the VSM.

- Real-time Pacer: The VSM is required to run in real-time so that the VGS can provide real-time data to the operator and behave as closely as possible

to the actual satellite. The simulation speed has to be paced in real-time and the stop time is changed to infinity. This is done using a real-time pacer that slows down a simulation according to the speed set by the user. When given a speed of 1, the simulation time becomes approximately equal to the real time. The pacer was created by Gautam Vallabha using Matlab S functions which are user-defined Simulink functions and can be found as a block in Simulink's library browser ([94]). The source code can be obtained from Matlab Central File Exchange.[95]. Once all the files from the downloaded link are saved in the same folder as the VSM Simulink model, the real-time pacer appears in the Simulink library browser. There are two blocks available: Elapsed Real Time and the Real-Time Pacer Speedup. These are added to the Simulink model and configured as shown in Figure B.8. The real-time pacer lies alongside the power simulator model in the same Simulink model and together they form the VSM. These two are not connected to each other in the model.

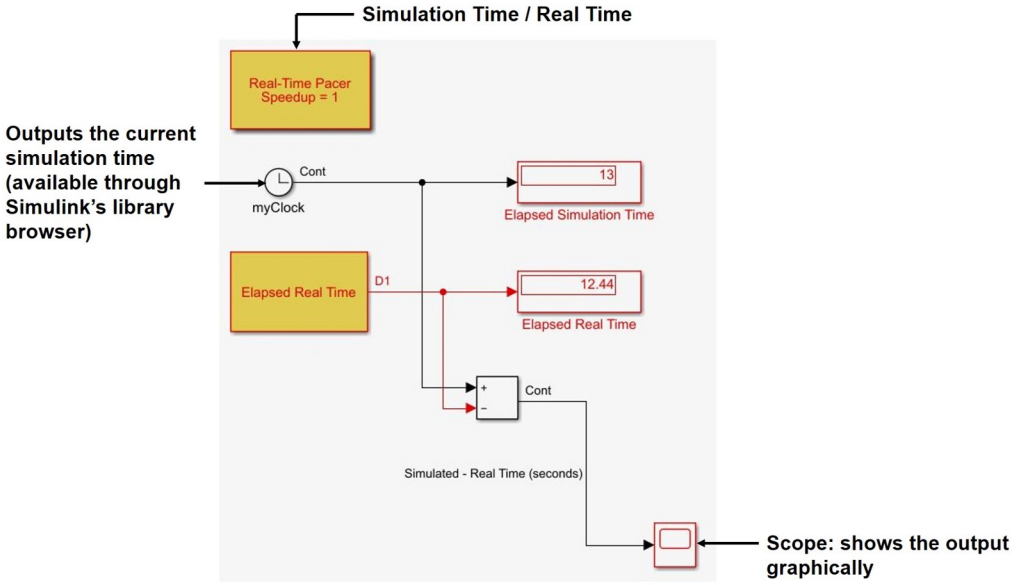

Figure B.8: The real-time pacer used in the VSM [95]

- Run Time Objects: Run the VSM programmatically by using the set_param function that starts the simulation model and within an infinite loop, log all data in real-time for every two seconds. Generally, logged data from a simulation is available only after the simulation time is over but this simulation requires to run in real-time and the simulation period is infinity. To help with real-time logging, I need an interface to access block run-time data from the simulator. I use listeners for specified events in the Simulink model to do this. When a listener is registered, it keeps listening for the event and specifies the block runtime object that it needs to listen to. Only handle classes can define events and listeners. A StartFcn callback (Appendix C.3.2) is added to the model to register the listeners for the different Simulink blocks.

  To access these parameters inside the VSM master script, I need to return the name or value of the specified parameter for the run time object blocks. From the values, I can link the signals to a destination outside the Simulink model and extract them. A snippet to do this for the state of charge is shown below. The same code can be modified for other parameters by simply changing the name of the run time blocks.

```
while(i < inf)  % i starts from 1
r = get_param('virtualsatellitemodel/soc', 'RuntimeObject');  %state of charge
m(1, i) = r.OutputPort(1).Data;
.... %other code
end
```

STK Orbit Propagator

I use the STK orbit propagator script (STKOrbitProp) (in Appendix C.2) to compute the access times with the satellite. The values are saved to a mat file that is editable.

Since I do not want the propagator script to update its TLEs every second and start the propagation again, I set a timer such that the updates and refreshing of the propagator happens every 12 hours. This number can be changed. After the propagator script is executed, a very small pause in the script is introduced to log all data. The pause function temporarily stops the execution of the Simulink models.

```
stk_timer = 12*3600; % 12 hours in seconds
if(rem(i,stk_timer) ==0)  %rem(a,b) function gives the remainder
                          %when a is divided by b
[accessStartTimes,accessStopTimes,accessDurationTimes] = STKOrbitProp();
end
pause(0.05) %seconds
```

Countdown Timer

To let the operators know that a pass is about to occur, I implement a countdown timer of 60 seconds which will appear in a separate UI figure prior to every pass. The countdown function is written by Brett Shoelson and can be found in [93]. When the time is 60 seconds away from a pass, the countdown function is triggered using an if statement. For this, I take the current time and compare it to the accessTimes from the STKOrbitProp script results and when there is a 60 second gap, the timer appears.

Virtual Satellite Model Data Logging

Before I check for a pass, I save the data from the VSM generated since the last pass occurred in a mat file by using indices for the data to be saved being changed only when a pass occurs. By doing this, the system knows that a pass has occurred and

that data has to be overwritten in the mat file again until the next pass takes places. This saved data is required for the real-time commands. The VGS should be able to provide the operators with any requested data immediately even when a pass is not occurring.

Passes

The current time during each iteration or second in the simulation is compared to the access times from the STKOrbitProp results. When equal, the 'pass loop' in the code is entered. The script is given in Appendix C.2.1. For the purposes of this research, I assume that the VGS receives the telemetry data through serial communication. To facilitate serial communication, I create a serialport object in Matlab.

- **Serial Communication Callback**: For the operations during a pass during serial communication, I write a callback function called nestedcallback. The first step in the callback is to initiate contact with the satellite by sending a 'handshake' message, for example: "Hello ManitobaSat!". In this thesis, I ignore the technicalities of encrypting messages, packeting, software etc. This is only for demonstration purposes. Using the function writeline and the serialport object created earlier, this is sent to the satellite.

  A timer object (t) is used to appropriately limit the duration of the pass and to schedule the downloading of telemetry. The execution mode is set to singleShot and there are infinite tasks that the timer has to be perform since the operations are taking place in real time and there is no real end. Before starting the timer, the timer callback function called setup_timer is declared.

  In setup_timer, I use the tic toc function to measure the elapsed time during a pass. A while loop restricts the pass duration such that the tic outside the

while loop starts the stopwatch and the condition for the while loop requires the toc to be lesser than the pass duration. The NumBytesAvailable function helps know when there is data to read from the serialport. A 'handshake' message from the satellite is expected and will be read at this point from the serialport by using the readline function.

After a minuscule pause, all the parameters are read one by one from the serialport. To make sure that there has been contact made with the satellite and that all data has been properly downloaded, the number of elements in the first parameter (state of charge) is compared with the number of elements in the last parameter (timestamps). If not equal, the fault alarm lamp on the Main Screen of the GUI with the label 'issues in downloading complete telemetry' is illuminated indicating an anomaly in the telemetry download. All the data read from the serialport is appended to any existing data from previous passes to form arrays containing parameters from all passes so far. This cumulative data is saved in a mat file. The latest pass data is also separately saved into a latest telemetry mat file. A break is added to the while loop so that the script does not try reading the serialport for telemetry again if the pass duration has not been exhausted. At this point, all operations related to obtaining telemetry are completed and the fault-management algorithms come into play.

- **Fault Management Algorithms** The solar string currents and the total solar array current are sorted with respect to their eclipse flags and the data collected during sun-facing time is extracted to begin statistical process control (SPC) on the currents. Script SS (Figure 5.19) from Chapter 5 is used here. Based on the results from the SPC algorithm, the fault alarm lamps on the Main Screen of the GUI are activated. The variable score described in the SPC algorithms (logDetTotal and logDetInd) from Chapter 5 that have a value between 1 and

3 correspond to the Low, Med and High lamps respectively. Similarly, The algorithm logDetRes is run on the battery internal resistance and triggers the corresponding lamps in the GUI.

Lastly, if there is atleast 3 (power consumption timeline repeats every three orbits for ManitobaSat-1, this variable is adjustable) orbits worth of data available, the time domain feature extraction algorithm (logDetPower) from Chapter 5 for excessive power consumption detection is executed. Again, the appropriate lamps on the GUI are activated based on the results.

- **Upload Commands**: The system uploads all commands in the queue to the satellite through the serialport object. Once uploaded, the Commands in queue text area is empty and these commands are visible in the Last uploaded commands text area of the Send Commands tab of the GUI. All commands created during the pass go to the satellite as well and can be seen in the Last uploaded commands text area. The setup_timer function is exited.

- **End Timer**: The timer t is stopped and then deleted. nestedcallback is exited to return to the end of the 'pass loop' in the virtual master script.

Final Steps

All data from the serialport object is flushed out. Cumulative data from the VSM is saved in a separate mat file. The diary function that triggers the Display Window text area (displays all messages from the command window) in the GUI is turned off. The infinite loop is exited and the set_param function is used again to stop the virtual satellite Simulink model.

B.4.2 Real-Time Commands

When an operator selects a real-time command, the VGS has to provide the requested data immediately. To do this, I write a function called realcommands. In realcommands, all the mat files with the latest and cumulative telemetry, STKOrbitProp results and VSM data are loaded. Each time a command is selected from the command drop down list, the latest versions of these files are loaded giving access to all their data. With switch case statements, the responses for every command are returned and displayed in the response text area of the Real-time Commands tab of the GUI.

B.4.3 Accessing Predicted and Actual Telemetry

Predicted and actual telemetry mat files are accessible in a folder on the VGS computer chosen by the user. The files' names include the date and time they are created at and their data can be imported as variables into Matlab for analysis at any time. This data is used to execute the real-time commands of the command manager. When a real-time command is selected, the VGS retrieves information from the relevant mat files to provide a suitable response.

B.5 Features of the VGST

The Virtual Ground Station Testbed (VGST) consists of a communication model based on generic interfaces that allow transfer of data and commands between the VGS and the actual satellite model (ASM). This testbed provides a suitable environment to evaluate the overall architecture of the VGS. Three of the features of the VGST are discussed below.

B.5.1 Programming Language for the Testbed: Matlab

I discussed the advantages of using Matlab to develop the VGS over traditional languages such as C and Fortran in Chapter 2. The simplicity and ease of modification of the code in Matlab makes it a good choice for the testbed as well. Moreover, connecting the components becomes easier when they all use Matlab. This is because Matlab provides numerous toolboxes and functions for communication between devices. This does not require detailed knowledge of how the protocols work. Since real time Simulink models are used in the VGS and the testbed, these toolboxes also significantly reduce the amount of code to be written and promote rapid development of the testbed. Also, Matlab provides the troubleshooting serial port interface options and the online support forum for external guidance. Clearly, Matlab is a prudent choice for communication between two or more devices, one of which already has the application running on Matlab App Designer and a real-time Simulink model.

B.5.2 Serial Communication

The virtual master script was developed based on the assumption that communication between the VGS and the satellite would follow the serial communication protocol. This retains as much realism as possible. Serial communication is the most common low-level protocol for communication between devices. Many researchers have used the RS232 serial protocol in testing spacecraft communications and systems as they are a reasonable and affordable representation of spacecraft communication modems [86][87][88]. Matlab provides various options for configuring serial port objects and properties such as the output format, baud rate and terminators.

B.5.3 Simulation Engine

A separate computer acts as the satellite and is used to send telemetry during each pass by running the 'actual' satellite model (ASM) in real time. Having a computer different from the VGS computer with GUI application makes the testbed environment more realistic as the satellite and the VGS would not share a host as in real life cases. This also allows me to use the RS232 protocols to depict spacecraft communications. Also, this removes any confusion that might be caused by having the VSM and the ASM on one computer and the complexity of multiple instances of Matlab. Overall, it is easier to visualize and understand the system symbolically as well. Debugging is also made simpler. This computer is expected to implement the commands received from the VGS on the ASM.

On the simulation engine, there is the ASM and a master script called satellitecom written to communicate with the VGS. Each of them is discussed below. Also, the changes required in the VGS to facilitate communication with the ASM are described.

Configuring the ASM

The ASM acts as the actual satellite in the testbed and so, would require to run in real time. Similar to the VSM, the real-time pacer (Figure B.8) in Simulink is used for the ASM. The ASM sends data to the Matlab instance that executes the satellitecom script. Additionally, there are two scripts that dictate the parameters for different blocks in the ASM. Having these scripts minimizes hard coding and makes modifications easier. These scripts are automatically executed when satellitecom is run. The first script, allvariables, has the parameters for all the blocks other than the battery and solar cell lookup tables. The second one, lookuptables, consists of the lookup tables (C - V and I - V). These scripts enable me to inject faults into the ASM for demonstration and testing purposes.

Introducing Faults into the ASM As before, there are three faults that are injected into the ASM, namely, solar string(s) failure, increase in the battery's internal resistance and excessive power consumption. There are three separate Simulink models created for testing for clarity. Each model corresponds to a separate type of fault.

For the solar string(s) failure, the model is called ASMSolar. There are different variables created in lookuptables for the solar string currents depending on what kind of fault I would like to test. The parameter values for the corresponding solar string currents are changed by configuring the solar string blocks and entering the appropriate variable name.

The internal resistance is set by a variable in the allvariables script. This is changed according to the fault to be tested using the ASMRes Simulink model.

The allvariables script also has a variable to adjust the power consumption values. The value of this variable is added to the ideal power consumption timeline and it is 0 when there is no anomaly in the power consumption. The Simulink model in this case is named ASMPower.

If faults are to be injected for specific time periods, switches (Simulink library browser) in the Simulink models with time constraints can be set up. The switches' input would be the simulation time clock.

Master Script: satellitecom

satellitecom has four parts, each part corresponding to one fault with its respective model and the ideal case with no fault and the ASM. The user can switch between each part of the script attached to the specific model (ASM, ASMSolar, ASMRes or ASMPower) with or without a fault by providing an input, that is, the parts

correspond to inputs 0, 1, 2 and 3. The code is essentially the same for each part except for the Simulink model used.

satellitecom starts with executing the allvariables and lookuptables scripts followed by loading other relevant parameters such as the power consumption timeline for three orbits (in the case of ManitobaSat-1) and the eclispe flag. At this point, the corresponding Simulink model is started by using the set_param function just as it was done for the VSM. Since the simulation runs in real time and has no defined simulation stop time, an infinite loop is started inside which I access the values of the several parameters through their Simulink blocks making them run time objects. This was previously discussed while developing the VSM. Similarly, listeners and handles are configured in the Startup Fcn of the Simulink models, the code for which is in Appendix C.3.2.

A serial port object is created and the script checks for any readable data by using the function NumBytesAvailable. If there is something waiting to be read such as the 'handshake' message from the VGS, the ASM starts reading the data. If the message received and read matches the pre-defined 'handshake' message "Hello ManitobaSat!", the ASM starts communicating. This means that the satellite is always listening and the VGS is responsible for initiating contact. Using appropriate indices inside the loop, all parameters are written to the serial port object one by one using the function writeline. Only data that has not been sent yet is written, that is, cumulative data for the parameters is not sent and no data is repeated to reduce unnecessary load on the serial communication. This is also realistic as satellites generally only send data generated since the last pass. Though other parameters do not have specifications on the exact number of observations to send in telemetry, the power consumption requires to be sent in multiples of 3 (period for the power consumption timeline for ManitobaSat-1, this number can be changed in the script). The script is modified to accommodate this by sending the power consumption values in sets of three orbits.

For example, when there is data from six orbits, two sets of data each with three orbits data are sent. In cases where lesser than three orbits of data is available, a string of zeros is sent. When there are more than three orbits of data available but the number of observations is not a multiple of three, the observations that are possible to be sent in sets of 3 orbits data are sent in the current pass and the remaining are sent during the next pass.

Once the data is written to the serial port, the indices in the loop are updated and communication loop that started with verifying the 'handshake message' is closed. Still inside the infinite loop, the serial port object is flushed to prepare the ASM for the next pass. The infinite loop ends here. The last step is to stop the Simulink model using the set_param function once the testing is completed.

The above described code is repeated four times with the only difference of the Simulink models in the set_param function arguments. Depending on the input (0, 1, 2 or 3) given to satellitecom, the corresponding part of the script will run. It is worthwhile to note that before closing the communication loop, certain tasks with respect to the commands sent from the VGS have to be undertaken. The next section provides a detailed discussion on the command handling by the ASM.

Command Handling

Commands are sent from the VGS to the ASM during passes though the commands can be created on the VGS by the user at any time. On the VGS, the data related to the commands that is sent include the number of commands in queue from the GUI's command manager and the commands themselves. This is done right after telemetry is received. Each command contains the date and time of required execution, the command ID (S-01, S-02 and S-03), the command description and the string 'End'.

On the ASM, after receiving the number of commands, a for loop in satellitecom

allows reading all the commands one by one. The components of the commands are extracted, that is, year, month, day, hour, minute, seconds, command ID and the description. These are stored in separate arrays. I create a command master array which contains all commands received so far and not just the commands received from the latest pass. The data from the latest pass' commands are added to this master array. All this takes place inside the pass loop.

Since I want to check for commands to execute every second and not only during passes, the execution of the commands and changes to the ASM are made outside the pass loop but inside the infinite while loop responsible for real-time execution. During each second, I compare the date and times given for the execution of the commands with the current time. If they are equal, the command is executed. The set_param function is again used to change the power consumption of the satellite in the ASM. The command ID of each command triggers the correct set_param statement during its execution. One such statement is shown here where A is the amount of change in the power consumption depending on the command sent. For example, when the camera is switched on, the power consumption increases by 0.85 W and A is 0.85. The code used in the command handling on the ASM are given in Appendix C.3.3.

```
set_param('actualsatellitemodel/PowerChange','Value', A)
```

On the master script of the VSM for the VGS, the commands are expected to be executed at the same time as on the ASM. This requires the power consumption on the VSM to change accordingly. Again, the set_param function is used and the VSM is updated.

Appendix C

Chapter 6 Scripts

This appendix includes the Matlab scripts used in various components of the graphical user interface (GUI) for the virtual ground station and in the development of the testbed and the actual satellite model (ASM). These correspond to Chapter 6.

C.1 Extracting Orbital Elements From a TLE File

```
function [OE] = processtles(fname)
%OE are the orbital elements and fname is the name of the TLE file
%MAKE SURE THAT THE CELESTRAK FOLDER IS IN THE MATLAB PATH

mu = 398600; %  Standard gravitational parameter for the earth

%% Open the TLE file and read TLE elements

fid = fopen(fname, 'rb');
L1 = fscanf(fid,'%24c%*s',1);
L2 = fscanf(fid,'%d%6d%*c%5d%*3c%*2f%f%f%5d%*c%*d%5d%*c%*d%d%5d',[1,9]);
```

```
L3 = fscanf(fid,'%d%6d%f%f%f%f%f%f%f',[1,8]);
fclose(fid);

%% Extract the orbital elements
epoch = L2(1,4)*24*3600;        % Epoch Date and Julian Date Fraction
Db    = L2(1,5);                % Ballistic Coefficient
inc   = L3(1,3);                % Inclination [deg]
RAAN  = L3(1,4);                % Right Ascension of the Ascending Node [deg]
e     = L3(1,5)/1e7;            % Eccentricity
w     = L3(1,6);                % Argument of periapsis [deg]
M     = L3(1,7);                % Mean anomaly [deg]
n     = L3(1,8);                % Mean motion [Revs per day]
a = (mu/(n*2*pi/(24*3600))^2)^(1/3);     % Semi-major axis [km]

% Calculate the eccentric anomaly using Mean anomaly
error = 1e-10;            %Error
E0 = M; t = 1;
iteration = 0;

while(t)
       E =  M + e*sind(E0);
      if ( abs(E - E0) < error)
          t = 0;
      end
      E0 = E;
      iteration = iteration+1;
end

%% Six orbital elements
OE = [a e inc RAAN w E];
end
```

C.2 Developing a Real-Time Orbit Propagator in STK from Matlab

A STK orbit propagator is developed from within Matlab with the STK Integration Plugin which accesses STK's COM interface [96]. By having the script for the propagator in Matlab, a user making changes to the GUI can easily modify any settings in STK as well. Apart from automating the analysis, post processing of the acquired data is possible in Matlab. Finally, visualizing the mission in real-time helps to understand and communicate the data well.

Figure C.1 summarizes the steps in integrating Matlab and STK and creating the propagator.

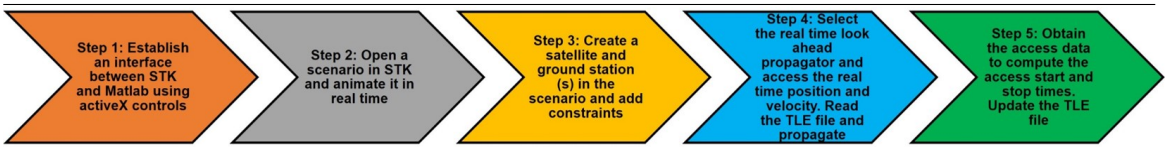

Figure C.1: Major steps in creating a real-time propagator in STK from Matlab

The ground stations used in the orbit propagator are in the STKOrbitProp script.

The script STKOrbitProp.m:

```
function[accessTimes, accessDurations] = STKOrbitProp()
%% Open STK from Matlab
try
    uiapp = actxGetRunningServer('STK11.application'); % ActiveX controls
catch
    uiapp = actxserver('STK11.application');
end
```

```
root = uiapp.Personality2;

uiapp.visible = 1; % show STK GUI


%% Create a scenario
try
    root.CloseScenario();
    root.NewScenario('Thesis_STKRealProp');
catch
    root.NewScenario('Thesis_STKRealProp');
end


scenObj = root.CurrentScenario


root.UnitPreferences.Item('DateFormat').SetCurrentUnit('LCLG');
% all dates in Local Gregorian
root.UnitPreferences.Item('Distance').SetCurrentUnit('m');
% all distances in meters

%% Assign scenario's start and stop times
tomorrow_date = datestr((now+1), 'dd mmm yyyy HH:MM:SS.FFF')
current_date = datestr((now), 'dd mmm yyyy HH:MM:SS.FFF')

scenObj.Epoch = current_date
scenObj.StopTime = tomorrow_date
scenObj.StartTime = current_date

%% Make the scenario run in real time
scAnimation = scenObj.Animation
```

```
scAnimation.AnimStepType = 'eScRealTime'

% Set animation to run in real time
root.Rewind
root.PlayForward

%% Creating the satellite and ground stations
satellite = scenObj.Children.New('eSatellite', 'mySat');

gs = scenObj.Children.New('eFacility', 'myGroundStation');
gs.Position.AssignGeodetic(49.808,-97.134,0); %location of the ground station
gs.HeightAboveGround = 0.237; % in km

gs2 = scenObj.Children.New('eFacility', 'myGroundStation_brazil');
gs2.Position.AssignGeodetic(-10.7524,-53.0812,0);
gs2.HeightAboveGround = 0; % in km

gs3 = scenObj.Children.New('eFacility', 'myGroundStation_capetown');
gs3.Position.AssignGeodetic(-33.9271,18.4201,0);
gs3.HeightAboveGround = 0; % in km

gs4 = scenObj.Children.New('eFacility', 'myGroundStation_capeverde');
gs4.Position.AssignGeodetic(15.0835,-23.6244,0);
gs4.HeightAboveGround = 0; % in km

gs5 = scenObj.Children.New('eFacility', 'myGroundStation_greatbritain');
gs5.Position.AssignGeodetic(54.0006,-2.59319,0);
gs5.HeightAboveGround = 0; % in km

gs6 = scenObj.Children.New('eFacility', 'myGroundStation_indonesia');
gs6.Position.AssignGeodetic(-0.199667,114.01, 0);
gs6.HeightAboveGround = 0; % in km
```

```
gs7 = scenObj.Children.New('eFacility', 'myGroundStation_madagascar');
gs7.Position.AssignGeodetic(-19.3789,46.6947,0);
gs7.HeightAboveGround = 0; % in km

gs8 = scenObj.Children.New('eFacility', 'myGroundStation_madras');
gs8.Position.AssignGeodetic(12.9481,80.1397,0);
gs8.HeightAboveGround = 0; % in km

gs9 = scenObj.Children.New('eFacility', 'myGroundStation_newzealand');
gs9.Position.AssignGeodetic(-43.9858,170.481,0);
gs9.HeightAboveGround = 0; % in km

gs10 = scenObj.Children.New('eFacility', 'myGroundStation_nigeria');
gs10.Position.AssignGeodetic(9.58265,8.09464,0);
gs10.HeightAboveGround = 0; % in km

gs11 = scenObj.Children.New('eFacility', 'myGroundStation_seoul');
gs11.Position.AssignGeodetic(37.5527,126.996,0);
gs11.HeightAboveGround = 0; % in km

gs12 = scenObj.Children.New('eFacility', 'myGroundStation_tashkent');
gs12.Position.AssignGeodetic(56.312,55.4075,0);
gs12.HeightAboveGround = 0; % in km

gs13 = scenObj.Children.New('eFacility', 'myGroundStation_melbourne');
gs13.Position.AssignGeodetic(-37.7992,144.964,0);
gs13.HeightAboveGround = 0; % in km

gs14 = scenObj.Children.New('eFacility', 'myGroundStation_uzbekistan');
gs14.Position.AssignGeodetic(38.9427,66.8506,0);
gs14.HeightAboveGround = 0; % in km

gs15 = scenObj.Children.New('eFacility', 'myGroundStation_YorkU');
```

```
gs15.Position.AssignGeodetic(43.7737,-79.5033,0);
gs15.HeightAboveGround = 0; % in km

%% Select a look ahead propagator
satellite.SetPropagatorType('ePropagatorRealtime');
satellite.Propagator.LookAheadPropagator = 'eLookAheadTwoBody';
satellite.Propagator.Duration.LookAhead = 7200.00;
satellite.Propagator.Duration.LookBehind = 7200.00;
satellite.Propagator.TimeStep = 60.0;
satellite.Propagator.Propagate;

%% Access the real time position and velocity displays
for k = 0:satellite.VO.DataDisplay.Count-1
    if (strcmp(satellite.VO.DataDisplay.Item(k).Name, 'J2000 Position Velocity'))
        posDD = satellite.VO.DataDisplay.Item(k);
        posDD.IsVisible = 1;
        posDD.FontColor = '000255000';
   end
end

primID = 1;
fname =  'tletest.txt'; % the TLE file to feed into the propogator
% Make sure that the latest TLE file to be downloaded each time is named
% 'tletest.txt' to facilitate automatic updating of TLE in the propogator

%% Read position and velocity information from the TLE file
while 1

    [epoch,p,e,i,O,o,nu,truLon,argLat,lonPer,mu] = readTLEtogetpv(fname);
    [epoch,r,v] = orb2rv(epoch,p,e,i,O,o,nu,truLon,argLat,lonPer,mu);
    [epoch, x_pos, y_pos, z_pos, x_vel, y_vel, z_vel] = get_posvel_datafromTLE(r,...
        v,epoch);
```

```
    %use the current system clock time as the timestamp for the data to be
    %passed into STK
    curTime = datestr((now), 'dd mmm yyyy HH:MM:SS.FFF');
    satellite.Propagator.PointBuilder.ECI.Add(curTime,...
         (x_pos), (y_pos), (z_pos),...
         (x_vel), (y_vel), (z_vel));


 %% Add constraints to the ground station(s)
 access = satellite.GetAccessToObject(gs);
 access.ComputeAccess
 accessConstraintsF = gs.AccessConstraints;
 accessConstraintsF.RemoveConstraint('eCstrElevationAngle');
 minmax = accessConstraintsF.AddConstraint('eCstrElevationAngle');

 minmax.EnableMin = true;
 minmax.Min = 20; % Minimum elevation angle in degrees
 RangeMax = 995.94*1000; % Maximum range in m
 accessConstraintsF.RemoveConstraint('eCstrRange');
 range = accessConstraintsF.AddConstraint('eCstrRange');
 range.EnableMax = true;
 range.Max = RangeMax;


%% Compute access times
%% GROUND STATION 1
accessDP1 = access1.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP1.DataSets.Count > 0)
accessStartTimes1 = (accessDP1.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes1 = (accessDP1.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes1 = (accessDP1.DataSets.GetDataSetByName('Duration').GetValues);
```

```
end

%% GROUND STATION 2

access2 = satellite.GetAccessToObject(gs2);
 access2.ComputeAccess

accessDP2 = access1.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP2.DataSets.Count > 0)
accessStartTimes2 = (accessDP2.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes2 = (accessDP2.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes2 = (accessDP2.DataSets.GetDataSetByName('Duration').GetValues);
end

%% GROUND STATION 3
access3 = satellite.GetAccessToObject(gs3);
access3.ComputeAccess

accessDP3 = access3.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP3.DataSets.Count > 0)
accessStartTimes3 = (accessDP3.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes3 = (accessDP3.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes3 = (accessDP3.DataSets.GetDataSetByName('Duration').GetValues);
end

%% GROUND STATION 4

access4 = satellite.GetAccessToObject(gs4);
access4.ComputeAccess
```

```
accessDP4 = access4.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP4.DataSets.Count > 0)
accessStartTimes4 = (accessDP4.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes4 = (accessDP4.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes4 = (accessDP4.DataSets.GetDataSetByName('Duration').GetValues);
end

%% GROUND STATION 5

access5 = satellite.GetAccessToObject(gs5);
access5.ComputeAccess

accessDP5 = access5.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP5.DataSets.Count > 0)
accessStartTimes5 = (accessDP5.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes5 = (accessDP5.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes5 = (accessDP5.DataSets.GetDataSetByName('Duration').GetValues);
end

%% GROUND STATION 6

access6 = satellite.GetAccessToObject(gs6);
access6.ComputeAccess

accessDP6 = access6.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP6.DataSets.Count > 0)
```

```
accessStartTimes6 = (accessDP6.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes6 = (accessDP6.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes6 = (accessDP6.DataSets.GetDataSetByName('Duration').GetValues);
end

%% GROUND STATION 7

access7 = satellite.GetAccessToObject(gs7);
access7.ComputeAccess

accessDP7 = access7.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP7.DataSets.Count > 0)
accessStartTimes7 = (accessDP7.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes7 = (accessDP7.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes7 = (accessDP7.DataSets.GetDataSetByName('Duration').GetValues);
end

%% GROUND STATION 8

access8 = satellite.GetAccessToObject(gs8);
access8.ComputeAccess

accessDP8 = access8.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP8.DataSets.Count > 0)
accessStartTimes8 = (accessDP8.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes8 = (accessDP8.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes8 = (accessDP8.DataSets.GetDataSetByName('Duration').GetValues);
end
```

```
%% GROUND STATION 9

access9 = satellite.GetAccessToObject(gs9);
access9.ComputeAccess

accessDP9 = access9.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP9.DataSets.Count > 0)
accessStartTimes9 = (accessDP9.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes9 = (accessDP9.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes9 = (accessDP9.DataSets.GetDataSetByName('Duration').GetValues);
end

%% GROUND STATION 10

access10 = satellite.GetAccessToObject(gs10);
access10.ComputeAccess

accessDP10 = access10.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP10.DataSets.Count > 0)
accessStartTimes10 = (accessDP10.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes10 =(accessDP10.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes10 = (accessDP10.DataSets.GetDataSetByName('Duration').GetValues)
end

%% GROUND STATION 11

access11 = satellite.GetAccessToObject(gs11);
access11.ComputeAccess
```

```
accessDP11 = access11.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP11.DataSets.Count > 0)
accessStartTimes11 = (accessDP11.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes11 = (accessDP11.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes11 = (accessDP11.DataSets.GetDataSetByName('Duration').GetValues)
end

%% GROUND STATION 12

access12 = satellite.GetAccessToObject(gs12);
access12.ComputeAccess

accessDP12 = access12.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP12.DataSets.Count > 0)
accessStartTimes12 = (accessDP12.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes12 = (accessDP12.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes12 = (accessDP12.DataSets.GetDataSetByName('Duration').GetValues)
end

%% GROUND STATION 13

access13 = satellite.GetAccessToObject(gs13);
access13.ComputeAccess

accessDP13 = access13.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP13.DataSets.Count > 0)
accessStartTimes13 =(accessDP13.DataSets.GetDataSetByName('Start Time').GetValues);
```

```
accessStopTimes13 = (accessDP13.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes13 = (accessDP13.DataSets.GetDataSetByName('Duration').GetValues)
end


%% GROUND STATION 14

access14 = satellite.GetAccessToObject(gs14);
access14.ComputeAccess
%root.UnitPreferences.Item('DateFormat').SetCurrentUnit('EpSec');

accessDP14 = access14.DataProviders.Item('Access Data').Exec(scenObj.StartTime,scen

if(accessDP14.DataSets.Count > 0)
accessStartTimes14 = (accessDP14.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes14 = (accessDP14.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes14 = (accessDP14.DataSets.GetDataSetByName('Duration').GetValues)
end


%% GROUND STATION 15

access15 = satellite.GetAccessToObject(gs15);
access15.ComputeAccess

accessDP15 = access15.DataProviders.Item('Access Data').Exec(scenObj.StartTime,...
    scenObj.StopTime);

if(accessDP15.DataSets.Count > 0)
accessStartTimes15 = (accessDP15.DataSets.GetDataSetByName('Start Time').GetValues);
accessStopTimes15 =(accessDP15.DataSets.GetDataSetByName('Stop Time').GetValues);
accessDurationTimes15 = (accessDP15.DataSets.GetDataSetByName('Duration').GetValues)
end
```

```
%% COMBINE THE ARRAYS

accessStartTimes = [accessStartTimes1 ;accessStartTimes2 ;...
    accessStartTimes3 ;accessStartTimes4 ;accessStartTimes5 ;...
    accessStartTimes6; accessStartTimes7; accessStartTimes8 ;...
    accessStartTimes9; accessStartTimes10; accessStartTimes11; ...
    accessStartTimes12 ;accessStartTimes13; accessStartTimes14 ;accessStartTimes15];
accessStopTimes = [accessStopTimes1 ;accessStopTimes2; ...
    accessStopTimes3 ;accessStopTimes4 ;accessStopTimes5 ;...
    accessStopTimes6 ;accessStopTimes7 ;accessStopTimes8; ...
    accessStopTimes9 ;accessStopTimes10 ;accessStopTimes11; ...
    accessStopTimes12 ;accessStopTimes13; accessStopTimes14; accessStopTimes15];
accessStartTimes = string(accessStartTimes);
accessStopTimes = string(accessStopTimes);
accessStartTimes = datetime(accessStartTimes, 'TimeZone', 'America/Chicago',...
    'InputFormat', 'dd MMM yyyy HH:mm:ss.SS');
accessStopTimes = datetime(accessStopTimes, 'TimeZone', 'America/Chicago',...
    'InputFormat', 'dd MMM yyyy HH:mm:ss.SS');

%% sort the access dates and times
accessTimes = [accessStartTimes accessStopTimes];
no_rows = size(accessTimes,1);

%Find union
accessTimes = sort(accessTimes,2);
[accessTimes, ind] = sort(accessTimes(:));
startstopArray = [ones(1,no_rows) -ones(1,no_rows)]';
startstopArray = startstopArray(ind);
startstopsum = cumsum(startstopArray);

%find ends of intervals
```

```
interval_ends = find(startstopsum == 0);

%find start of intervals
interval_starts = [1; interval_ends + 1];

%start no new interval at the end
len_interval_starts = length(interval_starts);
interval_starts(len_interval_starts) = [];

accessTimes = [accessTimes(interval_starts) accessTimes(interval_ends)];

accessIntervals = accessTimes(:,2) - accessTimes(:, 1);

[Y, M, D, H, MN, S] = datevec(accessIntervals);

accessDurations = H * 3600 + MN * 60 + S; % seconds

end
end
```

In the above code, I use ManitobaSat-1 as the satellite (depends on the TLE file) and the University of Manitoba's ground station as the primary contact facility. There are secondary ground stations added all around the world to increase accessibility. I get the access times for all the ground stations and sort them to find out the intervals during which the satellite would be contact with one of the ground stations.

The function readTLEtogetpv is based on the script test-sgp4 written by Meysam Mahooti on Matlab Central File Exchange [97]. This function obtains orbital elements from a TLE file. These orbital elements are given as an input to the function orb2rv that converts it to position and velocity vectors. orb2rv has been written by Darin Koblick on Matlab Central File Exchange [98]. The code is slightly modified for both the functions to suit this research. The position and velocity vectors from orb2rv are

split into their components by the function get-posvel-datafromTLE.

```
function [ epoch,p,e,i,O,o,nu,truLon,argLat,lonPer,mu] = readTLEtogetpv(fname)

%fname is the TLE file

%% Open the TLE file
fid = fopen(fname, 'r');
mu = 398600.4418 * 10^9; % gravitational constant

%% Read the TLE file
while (1)
    % Read first line
    tline = fgetl(fid);
    if ~ischar(tline)
        break
    end
    Cnum = tline(3:7);                          % Catalog Number (NORAD)
    SC   = tline(8);                            % Security Classification
    ID   = tline(10:17);                        % Identification Number
    year = str2double(tline(19:20));               % Year
    doy  = str2double(tline(21:32));               % Day of year
    epoch = str2double(tline(19:32))  ;            % Epoch
    TD1   = str2double(tline(34:43));              % first time derivative
    TD2   = str2double(tline(45:50));              % 2nd Time Derivative
    ExTD2 = tline(51:52);                       % Exponent of 2nd Time Derivative
    BStar = str2double(tline(54:59));              % Bstar/drag Term
    ExBStar = str2double(tline(60:61));            % Exponent of Bstar/drag Term
    BStar = BStar*1e-5*10^ExBStar;
    Etype = tline(63);                          % Ephemeris Type
    Enum  = str2double(tline(65:end));             % Element Number
```

```
    % Read second line
     tline = fgetl(fid);
     if ~ischar(tline)
         break
     end
     i = str2double(tline(9:16))   ;              % Orbit Inclination (degrees)
     raan = str2double(tline(18:25))  ;           % Right Ascension of
                                                  % Ascending Node (degrees)
     e = str2double(strcat('0.',tline(27:33)));   % Eccentricity
     omega = str2double(tline(35:42)) ;           % Argument of Perigee (degrees)
     M = str2double(tline(44:51));                % Mean Anomaly (degrees)
     no = str2double(tline(53:63));               % Mean Motion
     a = ( mu/(no*2*pi/86400)^2 )^(1/3) ;         % Semi major axis (m)
     rNo = str2double(tline(64:68));              % Revolution Number at Epoch
end
fclose(fid);

%% Orbital elements
p = a ;             % Semi-major axis in m
e = e;              % Eccentricity unitless
i = deg2rad(i);     % Inclination in rad
O = deg2rad(raan);  % Right Ascension of Ascending Node rad
o = deg2rad(omega); % Argument of Perigee rad
nu = deg2rad(M);    % Mean Anomaly (rad)
truLon = nu + O + o;% True longitude in rad
argLat = nu + o;    % Argument of lattitude in rad
lonPer = O + o;     % Longitude of periapsis rad

end


function [epsec,r,v] = orb2rv(epoch,p,e,i,O,o,nu,truLon,argLat,lonPer,mu)
```

```
epsec = epoch;
if ~exist('mu','var');  mu = 398600.4418 *10^9; end
p = p(:); e = e(:); i = i(:); O = O(:); o = o(:); nu = nu(:);
if exist('truLon','var')
truLon = truLon(:); argLat = argLat(:); lonPer = lonPer(:);
end
ietol = 1e-8;
idx = e < ietol & mod(i,pi) < ietol;
if any(idx); o(idx) = 0; O(idx) = 0; nu(idx) = truLon(idx); end
idx = e < ietol & mod(i,pi) > ietol;
if any(idx); o(idx) = 0; nu(idx) = argLat(idx); end
idx = e > ietol & mod(i,pi) < ietol;
if any(idx); O(idx) = 0; o(idx) = lonPer(idx); end
rPQW = cat(2,p.*cos(nu)./(1 +e.*cos(nu)),p.*sin(nu)./(1+e.*cos(nu)),...
    zeros(size(nu)));
vPQW = cat(2,-sqrt(mu./p).*sin(nu),sqrt(mu./p).*(e+cos(nu)),...
    zeros(size(nu)));
%Create Transformation Matrix
PQW2IJK = NaN(3,3,size(p,1));
cO = cos(O); sO = sin(O); co = cos(o); so = sin(o); ci = cos(i); si = sin(i);
PQW2IJK(1,1,:) = cO.*co-sO.*so.*ci;
PQW2IJK(1,2,:) = -cO.*so-sO.*co.*ci;
PQW2IJK(1,3,:) = sO.*si;
PQW2IJK(2,1,:) = sO.*co+cO.*so.*ci;
PQW2IJK(2,2,:) = -sO.*so+cO.*co.*ci;
PQW2IJK(2,3,:) = -cO.*si;
PQW2IJK(3,1,:) = so.*si;
PQW2IJK(3,2,:) = co.*si;
PQW2IJK(3,3,:) = ci;
%Transform rPQW and vPQW to rECI and vECI
r = multiDimMatrixMultiply(PQW2IJK,rPQW)';
v = multiDimMatrixMultiply(PQW2IJK,vPQW)';
end
```

```
function c = multiDimMatrixMultiply(a,b)
c = NaN(size(b));clc

c(:,1) = sum(bsxfun(@times,a(1,:,:),permute(b,[3 2 1])),2);
c(:,2) = sum(bsxfun(@times,a(2,:,:),permute(b,[3 2 1])),2);
c(:,3) = sum(bsxfun(@times,a(3,:,:),permute(b,[3 2 1])),2);
end


function [epsec,x_pos, y_pos, z_pos, x_vel, y_vel, z_vel] = ...
    get_posvel_datafromTLE(r, v,epsec)

%r is the position vector and v is the velocity vector


x_pos = r(1);
y_pos = r(2);
z_pos = r(3);

x_vel = v(1);
y_vel = v(2);
z_vel = v(3);

end
```

C.2.1 Initiating a Pass Based on Access Times From the STK Propagator

This script is part of the Virtual satellite model (VSM) master script and uses the results from the STKOrbitProp script to detect when passes are occurring. The

corresponding pass durations are also identified.

```
[accessTimes,accessDurations]= STKOrbitProp() ;
accessStartTimes = accessTimes(:,1);
accessStopTimes = accessTimes(:,2);

 currenttime = datetime('now', 'TimeZone', 'America/Chicago');
 currenttime_str = datestr(currenttime);

% compare the current time with all start times in the accessTimes array
% column 1 has start times
accessStartTimes_str = datestr(accessStartTimes);

if((ismember(currenttime_str,accessStartTimes_str,'rows'))==1)

[~,dur_index] = ismember(accessStartTimes_str,currenttime_ab_str,'rows');
pass_duration = accessDurations(dur_index == 1) ;

%CODE CONTINUES .....

end
```

C.3 Miscellaneous scripts

C.3.1 Timer Function for the Clock on the Main Screen of the GUI

```
function startupFcn(app)
```

```
t = timer;
t.TimerFcn = @app.refreshtime;
t.Period = 0.02;
t.BusyMode = 'drop';
t.ExecutionMode = 'fixedRate';
start(t)
app.time_the_timer = t;

end

function app = refreshtime(app,~,~)
app.CurrentTimeEditField.Value = datestr(now);
end
% Make sure to delete the timer (in the code) before closing the app
```

C.3.2 Listeners for the Virtual Satellite Simulink Model

```
set(0, 'showHiddenHandles', 'on');
% These are the block run time objects
blk = 'virtualsatellitemodel/soc';  % State of charge
blka = 'virtualsatellitemodel_v2/batteryvoltage';
blkb = 'virtualsatellitemodel/total_solar_current';
blkc = 'virtualsatellitemodel/batterycurrent';
blkd = 'virtualsatellitemodel/powerconsumption';

sc1 = 'virtualsatellitemodel/solar_current1';
sc2 = 'virtualsatellitemodel/solar_current2';
sc3 = 'virtualsatellitemodel/solar_current3';
sc4 = 'virtualsatellitemodel/solar_current4';
sc5 = 'virtualsatellitemodel/solar_current5';
```

```
ef_lis = 'virtualsatellitemodel/eclipseFlag';

event = 'PostOutputs';
listener = @virtualmasterscript;

% Register the listeners

h = add_exec_event_listener(blk, event, listener);
ha = add_exec_event_listener(blka, event, listener);
hb = add_exec_event_listener(blkb, event, listener);
hc = add_exec_event_listener(blkc, event, listener);
hd = add_exec_event_listener(blkd, event, listener);

hsc1 = add_exec_event_listener(sc1, event, listener);
hsc2 = add_exec_event_listener(sc2, event, listener);
hsc3 = add_exec_event_listener(sc3, event, listener);
hsc4 = add_exec_event_listener(sc4, event, listener);
hsc5 = add_exec_event_listener(sc5, event, listener);
```

C.3.3 Command Handling on the Testbed with the ASM

The following code is written as part of the script satellitecom which is the master script for the ASM as part of the testbed.

```
%% Read the commands from the serial port
nocommandsreceived = readline(dObj) %read number of commands
nocommandsreceived = str2double(nocommandsreceived);
counter = counter + 1;  % number of passes during which commands have been received

if(counter < 2)
    for z = 1:nocommandsreceived
```

```
  commandsreceived(z) = readline(dObj)
  % extract date
  year_commands(z) = extractBetween(commandsreceived(z),2,5)
  month_commands(z) = extractBetween(commandsreceived(z),7,8)
  day_commands(z) = extractBetween(commandsreceived(z),10,11)
  hour_commands(z) = extractBetween(commandsreceived(z),16,17)
  min_commands(z) = extractBetween(commandsreceived(z),22,23)
  sec_commands(z) = extractBetween(commandsreceived(z),29,30)
  % detect which command it is
  find_commands(z) = extractBetween(commandsreceived(z),38,38)
    end
elseif(counter > 2)
   commandsreceived(1) = readline(dObj)
  % extract date
  year_commands(1) = extractBetween(commandsreceived(1),3,6)
  month_commands(1) = extractBetween(commandsreceived(1),8,9)
  day_commands(1) = extractBetween(commandsreceived(1),11,12)
  hour_commands(1) = extractBetween(commandsreceived(1),16,17)
  min_commands(1) = extractBetween(commandsreceived(1),23,24)
  sec_commands(1) = extractBetween(commandsreceived(1),29,30)
  % detect which command it is
  find_commands(1) = extractBetween(commandsreceived(1),39,39)

  %if(nocommandsreceived > 1)

for z = 2:nocommandsreceived
  commandsreceived(z) = readline(dObj)
  % extract date
  year_commands(z) = extractBetween(commandsreceived(z),2,5)
  month_commands(z) = extractBetween(commandsreceived(z),7,8)
  day_commands(z) = extractBetween(commandsreceived(z),10,11)
  hour_commands(z) = extractBetween(commandsreceived(z),16,17)
  min_commands(z) = extractBetween(commandsreceived(z),22,23)
```

```
  sec_commands(z) = extractBetween(commandsreceived(z),29,30)
  % detect which command it is
  find_commands(z) = extractBetween(commandsreceived(z),38,38) % Command ID

  % 1: turn on the radio
  % 2: turn on the camera
  % 3: turn off the radio

end
end

%% Add commands to the master command arrays
collective_commands = ([collective_commands commandsreceived]);
collective_year = str2double([collective_year year_commands])
collective_month = str2double([collective_month month_commands])
collective_day = str2double([collective_day day_commands])

collective_hour = str2double([collective_hour hour_commands])
collective_min = str2double([collective_min min_commands])
collective_sec = str2double([collective_sec sec_commands])
collective_id = str2double([collective_id find_commands])

%create date time objects
t = datetime([collective_year' collective_month' collective_day' ...
    collective_hour' collective_min' collective_sec']);
total_commands = length(t);

% The code till here executes during a pass
%% Command Execution: outside the pass, executes every second
currenttime = datetime('now','TimeZone','local','Format','d-MMM-y HH:mm:ss');
currenttime_str = datestr(currenttime);
t_str = datestr(t);
```

```
for y = 1: total_commands
  if((ismember(currenttime_str,t_str(y,:),'rows')) == 1)
  %execute the command
  designated_id = collective_id(y); %command ID
   if(designated_id == 1)
      % 1: turn on the radio
   set_param('aalto1commserialactual_v2/PowerChange','Value','2.77')

   elseif(designated_id == 2)

       % 2: turn on the camera
   set_param('aalto1commserialactual_v2/PowerChange','Value','0.85')

   elseif(designated_id == 3)
       % 3: turn off the radio

     set_param('aalto1commserialactual_v2/PowerChange','Value','-2.77')
   end

 end
end
```

C.3.4 Limiting the Date and Time for Command Execution in the Send Commands Tab of the GUI

This script is implemented a callback for the Done button in the Send Commands tab of the GUI used for creating commands with their execution dates and times. The date and time selection inputs do not allow the operator to enter values of past date and time and execution time before the next pass as a command cannot be executed by satellite unless it is sent to it and this is possible only when a pass occurs.

```
function DoneButtonPushed(app, event)
sendcommand = app.sendcommandslist.Value;
% Selected values in the date picker and the time spinners
            sendcommand_s = char(sendcommand);
            sendcommand_date = app.commanddate.Value;
            sendcommand_date_s = char(sendcommand_date);
            sendcommand_hour = app.Hour.Value;
            sendcommand_hour_s = num2str(sendcommand_hour);
            sendcommand_min = app.Min.Value;
            sendcommand_min_s = num2str(sendcommand_min);
            sendcommand_sec = app.Sec.Value;
            sendcommand_sec_s = num2str(sendcommand_sec);
            zero_char = '0';

            if(sendcommand_hour < 10)

               sendcommand_hour_s =  strcat(zero_char, sendcommand_hour_s) ;
            end
            if(sendcommand_min < 10)

               sendcommand_min_s =  strcat(zero_char, sendcommand_min_s) ;
            end
            if(sendcommand_sec < 10)

               sendcommand_sec_s =  strcat(zero_char, sendcommand_sec_s) ;
            end


             %% done button wont work if selected date and time are before the next

             load('accesstimes.mat', 'accessTimes','accessStartTimes',...
                 'accessStopTimes','accessDurations');
```

```
            d1 = datestr(sendcommand_date);
             space = " ";
            timeString = strcat(d1, space, sendcommand_hour_s,":", ...
                sendcommand_min_s, ":", sendcommand_sec_s);
            timeString_ab = datetime(timeString,'Format',...
                'yyyy-MM-dd HH:mm:ss','TimeZone', 'America/Chicago');
            timeString_ab = datenum(timeString_ab);
            accessStartTimes_check = datenum(accessStartTimes);
            currenttime_check = datenum(datetime('now'));
            check_id = accessStartTimes_check > currenttime_check;
            accessStartTimes_check2 = accessStartTimes_check(check_id);
            log1 =  any(accessStartTimes_check2 <= timeString_ab);

            if(log1 == 1)

          app.Createdcommand.Value = strcat(sendcommand_date_s, space ,...
              "at",space,sendcommand_hour_s,"hrs",space,sendcommand_min_s,...
              "mins",space, sendcommand_sec_s, "sec", space, ...
              sendcommand_s);
          app.Commandsinqueue.Value = strcat(app.Commandsinqueue.Value,...
              " ",app.Createdcommand.Value) ;

            else
            app.Createdcommand.Value  =
      "Please try again, time of execution might be before the next pass" ;

            end
end


        % Value changed function: commanddate
        function commanddateValueChanged(app, event)  %value of ...
        %the date picker
```

```
%
            app.commanddate.Limits = [datetime('today')...
                datetime(9999,12,31)];
        end
```

Appendix D

Fault-Management and Control Rule Algorithms

This appendix contains the scripts and algorithms pertaining to the implementation of the control rules for fault detection in the statistical process control (SPC) module of the virtual ground station's (VGS) fault-management system.

D.1 Script to Identify the Zone of a Point in a Control Chart

The script zonescript (Algorithm 3) categorizes the points in a dataset based on the zones, the side of the mean they lie on and their score in a control chart as discussed in Chapter 5. Here, zones refers to the zones A, B, C or beyond limits (X) for every point and sides stands for the side of the mean on which a point lies on which can be above the mean or P and below the mean or N. A score of 8 refers to a point beyond limits and scores of 4, 2 and 1 stand for the points in zones A, B and C respectively.

Algorithm 3: Pseudocode for zonescript

```
Result: zones, sides, scores
for every data point x do
    if x < LCL then
        scores = 8; zones = "X"; sides = "N";
    else if x > UCL then
        scores = 8; zones = "X"; sides = "P";
    else if x > = LCL & x < -2sd then
        scores = 4; zones = "A"; sides = "N";
    else if x > = -2sd & x < -1sd then
        scores = 2; zones = "B"; sides = "N";
    else if x > = -1sd & x < mean then
        scores = 1; zones = "C"; sides = "N";
    else if x > = mean & x < 1sd then
        scores = 1; zones = "C"; sides = "P";
    else if x > = 1sd & x < 2sd then
        scores = 2; zones = "B"; sides = "P";
    else if x > = 2sd & x < UCL then
        scores = 4; zones = "A"; sides = "P";
    end
end
```

D.2 Algorithms for the SPC Control Rules

The control rule algorithms denoted by crAlg N where N is the control rule number are given below.

Summary of crAlg 1 Control rule 1 is the most important of all the rules. It requires one or more points to be beyond the control limits, that is, greater than UCL or lesser than LCL to be violated.

Algorithm 4: Pseudocode for crAlg 1

```
Result: pass / fail
for every element x with index i in the dataset do
    if x > UCL or x < LCL then
        Control rule 1 has been failed;
        return;
    end
end
Control rule 1 has been passed;
```

Summary of crAlg 2 Control rule 2 requires seven or more points in a row on the same side of the mean [73]. In zonescript, the variable sides indicates which side of the mean is a point on. Two strings called patternn and patternp are initialized representing eight consecutive points below and above the mean respectively. In the algorithm, I create a string by concatenating the sides value for the next seven points to give a string with the sides values for 8 consecutive points. If it matches either patternn or patternp, rule 2 has been violated.

Algorithm 5: Pseudocode for crAlg 2

```
Result: pass / fail
patternn = "NNNNNNNN"; patternp = "PPPPPPPP";
for every possible data point index i do
    stringtest: combine sides for points from index i to i + 7 or the next 7
     points;
    if stringtest matches either patternn or patternp then
        Control rule 2 has been failed;
    else
        Control rule 2 has been passed;
    end
end
```

Summary of crAlg 3 Control rule 3 is violated when seven or more points continuously trend up or down [73]. Let the current point's value in the loop be x_0, then the next six points' values are x_1, x_2, x_3, x_4, x_5 and x_6. The differences between each point and the consecutive point is calculated, for example: x_1 - x_0 .... x_6 - x_5. Note that the differences of the values of the points is taken and not their indices. If all the above differences are positive, that is, if the consecutive points are continuously increasing and every point is lesser than the next point, rule 3 is failed. Likewise, If all the above differences are negative, that is, if the consecutive points are continuously decreasing and every point is greater than the next point, rule 3 is failed.

Algorithm 6: Pseudocode for crAlg 3

Result: pass / fail

for *every possible data point* x_0 *with index* i **do**

calculate differences between two consecutive points from x_0 to x_6, i.e. x_1 - x_0 .... x_6 - x_5 where the indices of x_1, x_2, x_3, x_4, x_5 and x_6 are i + 1, i + 2, i + 3, i + 4, i + 5 and i + 6 respectively

if *all differences* > 0 *or all differences* < 0 **then**

Control rule 3 has been failed;

else

Control rule 3 has been passed;

end

end

Summary of crAlg 4 Control rule 4 is violated when 14 or more consecutive points alternate up and down [73]. Let the current point's value in the loop be x_0, then the next 14 points' values are x_1, x_2, x_3, x_4, x_5 and x_6.... x_{13}. The differences between each point and the consecutive point is calculated, for example: x_1 - x_0 .... x_{13} - x_{12}. Note that the differences of the values of the points is taken and not their indices. If the difference is > 0, that is, if the value of the points has increased, then the variable logicid takes the value 1 whereas if the difference is < 0, that is, if the value of the points has decreased, then logicid is 1. If the difference is 0, logicid is 0 indicating no violation of the rule. pattern1 and pattern2 are vectors of 14 alternating 1's and -1's with the former starting with a 1 and the other starting with -1. The vectors containing the values of all logicid known as the logicid array is compared with pattern1 and pattern2. If pattern1 or pattern2 are subsets of the logicid array, rule 4 is violated.

Algorithm 7: Pseudocode for crAlg 4

```
Result: pass / fail
pattern1 = [1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1];
pattern2 = -1 * pattern1;
for every possible data point x_0 with index i do
    if difference between data points (x_0 and x_1) with index i and i + 1 < 0
     then
        logicid = -1;
    else if difference between data points (x_0 and x_1) with index i and i + 1
     > 0 then
        logicid = 1;
    end
    logicid = 0;
end
for all points in logicid array do
    if any consecutive 14 points match pattern1 or pattern2 then
        Control rule 4 has been failed;
    else
        Control rule 4 has been passed;
    end
end
```

Summary of crAlg 5 Control rule 5 is the zone A test where failing this rule requires two out of three consecutive points lie in zone A or beyond on the same side of the mean [73]. There are two parts to crAlg 5. First, to identify if any two out of three consecutive points lie in zone A or beyond. Second, to check if they are on the same side of the mean. For the first for loop, the sides are not considered. I

use zonescript (Algorithm 3) to get the zones of all the data points and assign the zones value L to any point previously having a value of A or X, that is, zone A or beyond. The condition of two out of three consecutive points in Zone A or beyond can now be seen as two out of three consecutive points in Zone L. The strings of the three concatenated points' zone values need to have at least two characters in L and match one of the patterns: BLL, LBL, LLB, CLL, LCL, LLC and LLL. Their array is called selpatterns. Similar to the previous algorithms, I compare the concatenated strings with zone values for points for x_0, x_1 and x_2 where x_0 is the current point in the loop and their indices are i, i + 1 and i + 2 respectively. If one or more of the strings match any element in selpatterns, crAlg 5 is partially failed. The next step is to check their sides from zonescript. Any identified sets of three consecutive points that have failed the first condition of zones has the zones within each set compared. If the sides values match for all elements in at least one of the sets, rule 5 is failed.

Algorithm 8: Pseudocode for crAlg 5

```
Result: pass / fail
for every element in the zones array of the dataset do
    if zones is A or X then
        zones = "L";
    end
end
selpatterns = [BLL, LBL, LLB, CLL, LCL, LLC, LLL] ;
for every data point x_0 at index i do
    if zones z_0, z_1 and z_2 of points at indices i, i + 1 and i + 2 combined in
     order match any of the patterns in the array selpatterns then
        if two or more of the three consecutive points with zones = "L" lie on
         the same side of the mean then
            Control rule 5 has been failed;
        else
            Control rule 5 has been passed;
        end
    else
        Control rule 5 has been passed;
    end
end
```

Summary of crAlg 6 Control rule 6 is the zone B test which states that failing this rule requires four out of five consecutive points lie in zone B or beyond on the same side of the mean [73]. There are two parts to crAlg 6. First, to identify if any four out of five consecutive points lie in zone B or beyond. Second, to check if they are on the same side of the mean. For the first for loop, I have two if statements. I

use zonescript (Algorithm 3) to get the zones of all the data points and assign the zones value L to any point previously having a value of A, X or B. Also, their scores value from zonescript is given a value of 10. For determining the side of the mean the points lie on, a variable sidescorearray is set to 1 if the sides from zonescript for a particular point is P and -1 if the sides value is N. For the second for loop, the sum of the scores and sidescorearray values is taken for the points from x_0 to x_4 where x_0 is the current point in the loop and their indices are i, i + 1, i + 2, i + 3 and i + 4 respectively. The sum of the scores has to be between 40 (at least 4 have scores 10 each) and 50 (all 5 can have scores 10) and the sum of the sidescorearray should be either -3 (4 out of the 5 points are below the mean), 3 (4 out of 5 points are above the mean), -5 (all 5 points are below the mean) or 5 (all 5 points are above the mean) to violate rule 6.

Algorithm 9: Pseudocode for crAlg 6

Result: pass / fail

for *every element in the zones and sides array of the dataset* **do**

  if *zones is A, X or B* **then**

    zones = "L"; scores = 10;

  end

  if *sides is P* **then**

    sidescorearray = 1;

  else

    sidescorearray = -1;

  end

end

for *every data point* x_0 *at index* i **do**

  if *sum of scores of points* x_0, x_1, x_2, x_3 *and* x_4 *with indices* i, $i + 1$, $i + 2$, $i + 3$ *and* $i + 4$ *respectively is between 40 (inclusive) and 50* **then**

    if *sum of sidescorearray values of points* x_0, x_1, x_2, x_3 *and* x_4 *with indices* i, $i + 1$, $i + 2$, $i + 3$ *and* $i + 4$ *respectively is 3, -3, -5 or 5* **then**

      Control rule 6 has been failed;

    else

      Control rule 6 has been passed;

    end

  else

    Control rule 6 has been passed;

  end

end

Summary of crAlg 7 Control rule 7 is failed when there are 15 consecutive points all lying in zone C [73]. As in the previous algorithms, let x_0 (index i) be the current point in the for loop over all elements in the dataset. The next 14 points will be x_1, x_2..... x_{14} with indices i + 1, i + 2 ..... i + 14. The zones value for each of them is obtained using zonescript (Algorithm 3) and combined into a string zonematch. A string called patternall contains 15 consecutive C's which denotes that the 15 consecutive points' zone values when all lie in zone C. I compare zonematch to patternall and if it matches, rule 7 is violated.

Algorithm 10: Pseudocode for crAlg 7

```
Result: pass / fail
patternall = "CCCCCCCCCCCCCCC";
for all elements in the zones array of the dataset do
    if zones value for a data point and the next 14 points combined (called
     zonematch) match patternall then
        Control rule 7 has been failed;
    else
        Control rule 7 has been passed;
    end
end
```

Summary of crAlg 8 Control rule 8 is failed when there are 8 consecutive points with none of them in zone C [73]. For every element in the dataset, the zones value from zonescript is revised to be L if it was previously A, B or X. As in the previous algorithms, let x_0 (index i) be the current point in the for loop over all elements in the dataset. The next 7 points will be x_1, x_2..... x_7 with indices i + 1, i + 2 ..... i + 7. The zones value for each of them is obtained using zonescript and combined into a string zonematch. A string called patternall contains 8 consecutive L's which

denotes that the 8 consecutive points' zone values when none lie in zone C. I compare zonematch to patternall and if it matches, rule 8 is violated.

Algorithm 11: Pseudocode for crAlg 8

```
Result: pass / fail
for every element in the zones and sides array of the dataset do
    if zones is A, X or B then
        zones = "L";
    end
end
patternall = "LLLLLLLL";
for all elements in the zones array of the dataset do
    if zones value for a data point and the next 7 points combined (called
     zonematch) match patternall then
        Control rule 8 has been failed;
    else
        Control rule 8 has been passed;
    end
end
```