University of Padova

Department of Information Engineering

Master Thesis in Control System Engineering

Analysis of Artificial Intelligence Based Diagnostic Methods for Satellites

Supervisor
Prof. Giulia Michieletto
University of Padova

Co-Supervisors
Ing. Chiara Brighenti
Ing. Mattia Ricatto
S.A.T.E. Srl

Master Candidate
Paolo Arvotti

Student ID
2021129

Academic Year
2022-2023

"Artificial Intelligence is the power to transform our perception into knowledge, our knowledge into insight, and our insight into action for the betterment of our world."

— Fei-Fei Li

Abstract

The growing utilization of small satellites in various applications has emphasized the need for reliable diagnostic methods to ensure their optimal performance and longevity. This master thesis focuses on the analysis of artificial intelligence-based diagnostic methods for these particular space assets.

This work firstly explores the main characteristics and applications of small satellites, highlighting the critical subsystems and components that play a vital role in their proper functioning.

The key components of this study revolve around Diagnosis, Prognosis, and Health Monitoring (DPHM) systems and techniques for small satellites. The DPHM systems aim at monitoring the health status of the satellite, detecting anomalies and predicting future system behavior. The reason why advanced DPHM systems are of interest for the space operators is the fact that they mitigate the risk of satellites catastrophic failures that may lead to service interruptions or mission abort. To achieve these objectives, a hybrid architecture combining Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks is proposed. This architecture leverages the strengths of CNNs in feature extraction and LSTM networks in capturing temporal dependencies. The integration of these two neural network architectures enhances the diagnostic capabilities and enables accurate predictions for small satellite systems.

Real data collected from an operational satellite is utilized to validate and test the proposed CNN-LSTM hybrid architecture. Based on the experimental results obtained, advantages and drawbacks of the exploitation of this architecture are discussed.

Contents

Listing of figures

Listing of tables

Listings

Listing of acronyms

AI ..... Artificial Intelligence

AOCS ..... Attitude & Orbit Control Subsystem

Bi-LSTM ..... bi-direction LSTM

BOL ..... Beginning-Of-Life

BPTT ..... Backpropagation Through Time

CBM ..... Conditional-Based Maintenance

CHD ..... Command & Data Handling Subsystem

CMI ..... Conditional Mutual Information

CNN ..... Convolutional Neural Network

CN-FA-LSTM .. Causal Network and Feature-Attention-based LSTM

CRNN ..... Convolutional Recurrent Neural Network

DCNN ..... Deep Convolutional Neural Network

DPHM ..... Diagnosis, Prognosis, and Health Monitoring

ELU ..... Exponential Linear Unit

EO ..... Earth Observation

EOL ..... End-Of-Life

EPDS ..... Electrical Power & Distribution Subsystem

ETTF ..... Estimated Time To Failure

FDI ..... Fault Detection and Identification

FPS ..... False Positive Rate

FTC ..... Fundamental Theorem of Calculus

GNN . . . . . Growing Neural Network

GRU . . . . . Gated Recurrent Unit

HI . . . . . Health Index

KPCA . . . . . Kernel Principal Component Analysis

LSTM . . . . . Long Short-Term Memory Network

ML . . . . . Machine Learning

MLS . . . . . Microwave Limb Sounder

MSE . . . . . Mean Square Error

NMCTE . . . . . Normalized Modified Conditional Transfer Entropy

NPL . . . . . Natural Language Processing

NRMSE . . . . . Normalized Root Mean Square Error

OBC . . . . . On-Board Computer

PCA . . . . . Principal Component Analysis

PL . . . . . Payload Subsystem

PROP . . . . . Propulsion Subsystem

RMSE . . . . . Root Mean Square Error

RNN . . . . . Recurrent Neural Network

RUL . . . . . Remaining Useful Life

SMAP . . . . . Soil Moisture Active & Passive

SMS . . . . . Structures & Mechanism Subsystem

TBM . . . . . Time-Based Maintenance

TCNN . . . . . Temporal Convolutional Neural Network

TCS . . . . . Thermal Control Subsystem

TTC . . . . . Telemetry, Tracking & Command Subsystem

Var-LSTM . . . . . variable sequence LSTM

1 Introduction

1.1 Motivation

Satellites have always been playing a crucial role in various applications, ranging from communication and navigation to Earth observation and scientific research. The proliferation of small satellites over the last decade has been remarkable, driven by factors such as technological advancements, cost reduction, and increased accessibility. These compact spacecraft offer unique advantages, including flexibility in design, faster development cycles, and the ability to address specific mission requirements.

The deployment of small satellites brings numerous opportunities and increases reliance on the services offered by these flying systems. For this reason, ensuring their reliable and efficient operation becomes paramount. To reach this goal, there is the need for diagnostic and prognostic capabilities that aim at identifying and resolving issues that may arise during the lifetime of a satellite. However, traditional diagnostic methods, which often rely on predefined rules or human expertise, can struggle to keep up with the scale and complexity of satellites currently in flight. Furthermore, the limited resources and constraints associated with small satellites necessitate efficient and automated diagnostic techniques to ensure their continued operation in a cost-effective manner.

This is where artificial intelligence (AI) and machine learning (ML) come into play. AI and ML techniques offer the potential to revolutionize satellite diagnostics by enabling auto-

mated analysis of vast amounts of data, detection of complex patterns, and accurate prediction of faults or anomalies. By leveraging these advanced technologies, it becomes possible to develop diagnostic methods that are adaptable, scalable, and capable of handling the intricate subsystems and components in small satellites. The benefits of AI and ML-based diagnostic techniques are multifaceted. Firstly, they can enhance the reliability and performance of small satellites by enabling early fault detection and diagnosis. By promptly identifying and resolving issues, the downtime and potential mission failures can be minimized, ensuring the continuous operation and success of satellite missions. Secondly, these techniques offer the potential for real-time or near real-time monitoring of satellite status, enabling proactive maintenance and predictive maintenance strategies that can improve overall system reliability. By harnessing the power of AI and ML, satellite operators and engineers can unlock valuable insights from the vast amount of data generated by these complex systems. Through automated data analysis and intelligent algorithms, patterns and correlations that may not be apparent to human operators can be detected, enabling more accurate and efficient diagnostic processes.

In this thesis, the aim is to explore and analyze AI and ML-based diagnostic methods for small satellites, with a focus on improving the reliability and efficiency of satellite operations. By investigating the various techniques and approaches available in the field, the thesis seeks to identify the most suitable approach to address the diagnostic challenges specific to small satellites.

In the next section, a deeper exploration of the specific problem statement will be provided, along with an outline of the thesis structure. This will outline the key areas of focus and the subsequent chapters.

1.2 Problem Statement

This thesis aims to support the definition of automated and reliable diagnostic systems for small satellites. This thesis aims to address this pressing demand and contribute to the development of automated and reliable diagnostic systems for small satellites.

This work has been performed during a 6-month internship at S.A.T.E. (Systems and Advanced Technologies Engineering), an engineering company that provides advanced and forefront customised software products and services for simulation, diagnostics and data analysis in the automotive, space and energy fields. S.A.T.E. has more than twenty years of experience in developing diagnostics solutions and during the last decade these have been tailored also for space applications. For this reason, the company recognises the importance of having robust di-

agnostic and prognostic systems that can ensure reliable operations for these complex systems.

To effectively tackle this problem, it is essential to first focus on the failures and anomalies that occur in small satellites. Therefore, a comprehensive analysis of the critical subsystems and components is required to identify the areas that require diagnostic attention. Through the examination of historical data and extensive research, the aim is to determine the most critical subsystems and identify the key components within those subsystems that have a significant impact on the overall performance and reliability of small satellites.

The current state of the art in satellite diagnostics offers various techniques and methodologies; however, there are still gaps and limitations that need to be addressed. For instance, conventional diagnostic methods often depend on predefined rules or manual analysis, which can encounter difficulties when dealing with the intricate nature of small satellite systems. Additionally, the ever-increasing number of small satellites being launched poses significant challenges in terms of scalability and efficiency of diagnostic methods. This thesis aims to bridge these gaps by leveraging AI and ML techniques to develop automated and data-driven diagnostic solutions that can adapt to the unique characteristics of small satellites.

Automation assumes a pivotal role in satellite diagnostics due to its numerous advantages over manual processes. By embracing AI and ML techniques to automate the diagnostic process, higher levels of efficiency, accuracy, and reliability can be achieved. The ability to analyze vast amounts of data in real-time enables the early detection and diagnosis of faults, leading to reduced downtime and increased operational reliability. Furthermore, the integration of prognostic capabilities can enable predictive maintenance strategies, further enhancing the longevity and efficiency of small satellites.

1.3 Thesis Objectives

The primary objective of this thesis is to develop and demonstrate the effectiveness of an AI and ML-based diagnostic technique for small satellites to develop an AI based solution for the detection of anomalous behaviour in satellites components, by leveraging real telemetry data made available by SATE. To accomplish this objective, the following workflow was implemented:

1. **Alalysis of the Subsystems and Critical Components of Small Satellites:**

    - Investigation of small satellite missions deployed over the last decades, in order to understand the main goals and characteristics of the so-called New Space Economy.

- Identification of all subsystems that are part of a satellite, with particular attention to those that significant impact the overall performance and reliability of the system.
- Determination of the most critical components within these subsystems and understand their role in satellite failures and anomalies.

2. **Evaluation of Various Diagnostic and Prognostic Techniques Based on AI and ML:**
    - Review and Evaluation of the state-of-the-art diagnostic and prognostic techniques that utilize AI and ML in the context of satellite engineering.
    - Analysis of the advantages, limitations, and applicability of these techniques to small satellite diagnostics.
    - Comparison of the different approaches, considering factors such as accuracy, computational complexity, data requirements, and scalability.

3. **Selection and Testing of a Specific Diagnostic Technique on Real Data:**
    - Selection of the most suitable AI and ML-based diagnostic technique for small satellite diagnostics based on the findings from the literature review and analysis.
    - Development of the chosen diagnostic technique and customize it to suit the specific requirements of small satellites.
    - Acquisition of real data from small satellite missions and apply the selected diagnostic technique to evaluate its effectiveness in detecting and diagnosing faults.

4. **Demonstrate the Effectiveness of the Chosen Diagnostic Technique:**
    - Analysis and interpretation of the results obtained from the application of the diagnostic technique to real data.
    - Validation of the accuracy, efficiency, and reliability of the technique in identifying and diagnosing faults in small satellite systems.
    - Comparison of the performance of the chosen diagnostic technique with existing methods and highlight its advantages and contributions.

Achieving these goals, this thesis aims to contribute to the satellite engineering field by providing a comprehensive analysis of small satellite subsystems and critical components, evaluating AI and ML-based diagnostic techniques and demonstrating the effectiveness of the selected diagnostic technique on real data.

1.4 Thesis Outline

This thesis is organized into several chapters, each focusing on a specific steps reported in the previous section. The following is an overview of the main sections and their content:

1. **Chapter 1:** *Introduction*

    - Provides an introduction to the field of satellite diagnostics, highlighting the importance of reliable diagnostic techniques for small satellites.
    - Presents the motivation behind the thesis and outlines the problem statements and objectives.
    - Introduces the structure and organization of the thesis.

2. **Chapter 2:** *Small Satellites: Main Characteristic and Applications*

    - Provides a detailed overview of the concept of satellites, with a focus on small satellites, and their applications in various missions.
    - Analyzes the composition and subsystems of small satellites, emphasizing their criticality by investigating failures and anomalies.
    - Concludes with a summary of the research findings.

3. **Chapter 3:** *Diagnosis, Prognosis, and Health Monitoring (DPHM)*

    - Introduces the concepts of diagnostics, prognostics, and health monitoring in the space environment.
    - Discusses various diagnostic and prognostic methods, including model-based, signal-based, data-driven, and hybrid approaches.
    - Explores the existing literature and research in the field of AI and ML-based diagnostic techniques for satellites.
    - Proposes and expands the problem statement, highlighting similar examples and their state-of-the-art solutions.
    - Concludes with the selection of the most appropriate diagnostic technique.

4. **Chapter 4:** *Long Short-Term Memory (LSTM) Network*

- Introduction to LSTM and its architecture.
- Exploration of the functionality and advantages of LSTM networks.
- Discussion of training and learning in LSTM networks.
- Exploration of the application of LSTM in various domains, including its relevance in the space environment.
- Presentation of examples showcasing the utilization of LSTM in the space domain.
- Addressing limitations and considerations of using LSTM.

5. **Chapter 5:** *Workflow and Results*

   - Provides an overview of the environment and data used in the study.
   - Describes the dataset split into train and test periods and explains the data preprocessing techniques applied.
   - Discusses the network architecture, specifically the CNN-LSTM hybrid network.
   - Presents the results obtained from the training process.
   - Details the testing process and evaluation metrics and conducts an overall evaluation and discussion of the findings.

6. **Chapter 6:** *Conclusions*

   - Summarizes the main findings and contributions of the thesis.
   - Reflects on the achievements and limitations of the research.
   - Provides final thoughts and recommendations based on the research outcomes.

2

Small Satellites: Main Characteristic and Applications

2.1 Introduction to Satellites

In this section, a deep study of satellites will be conducted, aiming to provide a comprehensive understanding of their characteristics, operational environment, and the challenges they commonly face.

A satellite can be defined as a deliberately placed object in orbit around a celestial body, such as the Earth, Moon, or another planet. It relies on the principles of orbital mechanics to maintain its trajectory and spatial position. Satellites can be categorized based on their purpose, size, and orbital characteristics. Notably, the deployment of small satellites, also known as CubeSats or nanosatellites, has witnessed a remarkable surge in recent years. These satellites, typically weighing from a few kilograms to a few hundred kilograms, have gained popularity due to their compact size, cost-effectiveness, and versatility in executing a wide range of missions.

The space environment presents numerous challenges for satellites. In fact, they must withstand extreme temperatures, vacuum conditions, radiation exposure, micro-meteoroids, and other harsh elements. These factors can significantly impact the performance, reliability, and longevity of satellite systems. Moreover, the intricate nature of satellite operations, encompassing deployment, orbit control, data transmission, manoeuvres, and power management,

demands meticulous engineering and continuous monitoring to ensure optimal functionality.

In this particular scenario, the accurate identification and diagnosis of failures or anomalies in satellite subsystems and components are essential for maintaining optimal performance, prolonging operational lifespan, and minimizing the risk of mission failure. Traditional diagnostic approaches have inherent limitations in terms of efficiency and accuracy, underscoring the need to explore advanced techniques that exploits the power of AI and ML to automate and enhance the diagnostic process.

In the subsequent sections, this study delves deeper into the analysis of small satellites, exploring their proliferating launch rate, diverse mission profiles, the involvement of funding agencies and developers, as well as the composition of their subsystems. Furthermore, the study investigates the failures and anomalies that occur within these satellites, aiming to identify the most critical subsystems and components.

2.2 Types of Satellites

Satellites can be categorized based on their physical characteristics, specifically their size and mass. This section focus on the various types of satellites classified according to their mass, providing comprehensive insights into their distinct characteristics and differentiating factors.

1. **Large Satellites:** Large satellites typically weigh more than 500 kilograms and feature complex designs, multiple payloads, and their own propulsion systems. These satellites are capable of accommodating a wide range of advanced instruments and technologies, making them suitable for various missions requiring extensive capabilities.

2. **Small Satellites:** Small satellites, on the other hand, are designed to be smaller and more cost-effective compared to traditional satellites. They generally have a mass between 1 and 500 kilograms and offer simpler designs with fewer payloads. Some small satellites may not have their own propulsion systems and rely on alternative methods for maneuvering in space.

Small satellites can be further categorized based on their mass, which helps distinguish their specific capabilities and applications. Are listed in the following the common categories of small satellites:

- *Nano satellites:* Nano satellites have a mass between 1 and 10 kilograms. They often adhere to standardized designs, such as CubeSats (10 cm × 10 cm × 10 cm) or PocketQubes, which employ modular components for easy assembly and launch. Nano satellites are

popular for various purposes and are well-suited for missions that require compact and cost-efficient solutions.

- *Micro satellites:* Micro satellites weigh between 10 and 100 kilograms. They are larger and more complex compared to nano satellites and find applications in specialized areas where additional capabilities and payloads are required.

- *Mini satellites:* Mini satellites have a mass ranging from 100 to 500 kilograms. They represent the largest and most sophisticated category among small satellites. Due to their increased size and complexity, mini satellites are capable of supporting demanding missions that require advanced instruments and systems.

In addition to these primary categories, there are also smaller types of small satellites, including *pico satellites* (0.1-1 kg), *femto satellites* (10-100 g), *atto satellites* (1-10 g), and *zepto satellites* (0.1-1 g). These miniaturized satellites meet to specific applications and offer further flexibility in terms of design and deployment.

2.3 Growth of Small Satellites

The emergence and growth of small satellites have significantly transformed the landscape of the satellite industry. Referring to Figure 2.1, the growth of nano satellites can be divided into three distinct phases, each characterized by different levels of expansion and advancements:

1. From 1999 to 2013, there was a relatively slow growth observed in the deployment of small satellites. During this period, the industry was still exploring and developing the potential of small satellites, with initial experiments and technology demonstrations taking place.

2. From 2013 to 2016, there was a significant increase in the number of small satellite launches. This surge can be attributed to the development of projects that began in the late 2000s and early 2010s, including the first constellation demonstrations. Advances in miniaturization, standardized CubeSat platforms, and commercialization opportunities played a key role in this growth spurt.

3. From 2017 onwards, the launch of small satellites has continued to show a remarkable upward trend, despite a temporary downward phase. It is important to note that while there was a decline in launches for three consecutive years after the record of 297 spacecraft launched in 2017, this decrease was influenced by various factors, including launch delays of new small launchers and large rideshare missions. However, it is essential to

highlight that many satellite projects are built according to predetermined launch schedules, which means that the scaling-up of constellations and mission deployments often occurs gradually.

Despite the temporary dip in launches, 2022 witnessed a new record in the number of nano satellite launches, with 338 spacecraft successfully reaching orbit. This trend indicates the continuing growth and interest in small satellites, with numerous missions and projects in the pipeline.

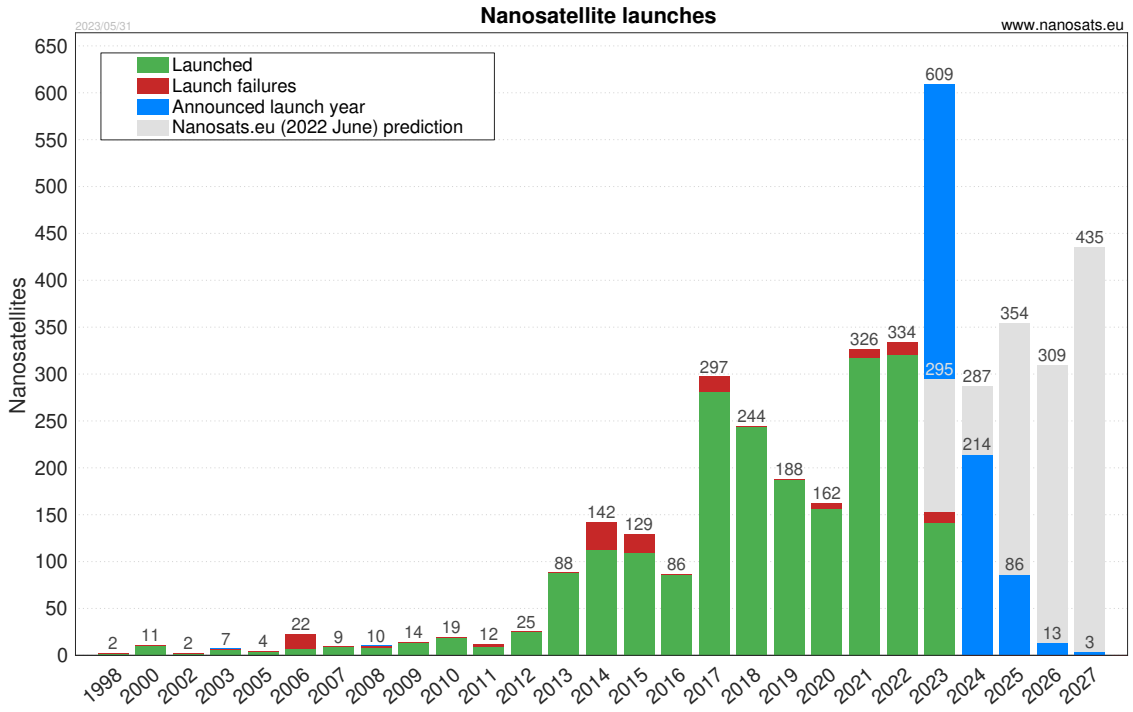


Figure 2.1: Nano Satellite Launches (Nanosats Database) [1]

Advantages and Disadvantages of Small Satellites

The increase of small satellite launches can be attributed to several to the following factors:

- Technological advancements facilitated the development of sophisticated satellite systems in compact form factors. Through the miniaturization of components, improved power efficiency, and enhanced computing capabilities, it has become increasingly feasible to create small satellites with scalable and accessible functionalities.
- Cost reduction has emerged as a driving force behind the growth of small satellites. Their reduced size and weight contribute to lower manufacturing, launch, and operational expenses compared to larger satellites. This enhanced affordability has attracted a wide

range of stakeholders, including academic institutions, research organizations, commercial entities, and even individual enthusiasts, fostering active participation in satellite missions and the utilization of space-based data.

- The expanding market and increased opportunities for commercialization have further facilitated the growth of small satellites. The demand for Earth observation data, communication services, and scientific research has stimulated the development of small satellite constellations and the establishment of new companies dedicated to small satellite operations. This competitive environment has fueled innovation in advanced small satellite technologies.

However, in order to better understand it is worth mentioning that the exploitation of small satellites also present the following limitations.

- Limited payload capacity: The small size of these satellites limits their payload capacity, which can restrict the types and sizes of instruments and sensors that can be carried onboard. This limitation may impact the scope and complexity of missions that can be undertaken.
- Reduced power and communication capabilities: Small satellites typically have limited power generation capacity and communication bandwidth, which can impact data acquisition, transmission, and real-time communication. This constraint requires careful optimization of power management and communication protocols.
- Shorter operational lifetimes: Due to their smaller size and resource limitations, small satellites often have shorter operational lifetimes compared to larger satellites. Factors such as orbital decay, limited fuel reserves, and technological obsolescence may necessitate shorter mission durations or satellite replacement.
- Reliance on commercial launch opportunities: The availability of suitable launch opportunities for small satellites can be influenced by the priorities and schedules of commercial launch providers. This reliance on commercial providers may introduce potential delays or uncertainties in accessing space.

2.4 Mission Applications and Stakeholders

The wide range of mission applications for small satellites encompasses a multitude of purposes and addresses to the needs of various stakeholders. The mission types, illustrated in Figure 2.2, have been systematically classified into the following categories:

- **Earth Observation (EO):** Earth Observation missions provide imagery coverage and data products relating to terrestrial activity. EO missions focus on capturing and analyzing data related to human activity on Earth's surface. They play a crucial role in applications such as environmental monitoring, urban planning, agriculture, and disaster management.

- **Technology/Test:** Technology/Test missions are designed to demonstrate new payloads, components, or subsystems that lack space flight heritage. These missions serve as testbeds for innovative technologies and advancements in satellite systems. Examples include the demonstration of new reaction wheels, propulsion systems, or mission operations paradigms like proximity operations or cluster flight. Technology/Test missions contribute to the advancement of small satellite capabilities and enable the validation of new concepts before their integration into operational missions.

- **Communications:** Communications missions focus on providing reliable and efficient communication services from space. These missions facilitate real-time connectivity, data storage and forwarding, radio frequency communications, and system identification. Small satellites deployed for communications purposes enhance global connectivity, enable internet access in remote areas, support disaster response efforts, and provide critical communication infrastructure for various stakeholders.

- **Science:** Science missions gather data about the Earth's surface, weather, the atmosphere, or free space outside the atmosphere. While some Science missions may use visual imagery or data products, EO missions primarily focus on studying human activity, while Science missions look at natural phenomena. Science missions contribute to scientific research, enabling a better understanding of our planet, weather patterns, climate change, and space environment.

- **Other:** There exist also some new and emerging missions that do not fit directly into the above categories. This includes missions such as early warning systems, on-orbit servicing missions, signals intelligence, or cargo missions. These diverse missions reflect the ever-evolving nature of the small satellite industry and the continuous exploration of new applications and capabilities.

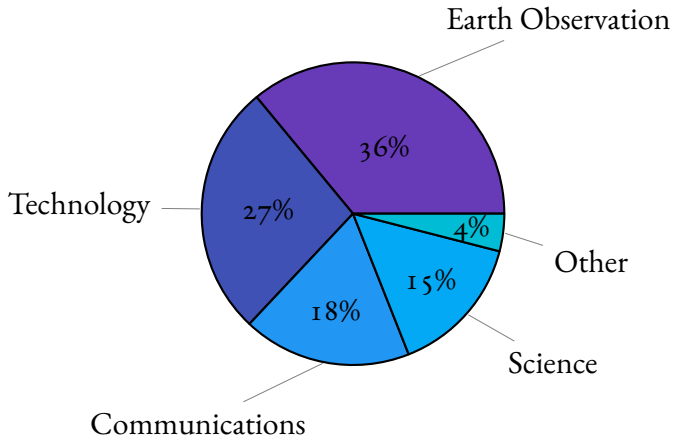


Figure 2.2: Mission Type in Aerospace Corporation Database (2009 – 2018) [2]

Within the development and deployment of small satellites, multiple public and private sector stakeholders play a crucial role. These stakeholders can be broadly classified as funding agencies and developers, which include a number of entities involved in the process. Specifically, the main stakeholders identified are as follows, as depicted in Figure 2.3:

1. **Commercial:** In this sector there are for-profit entities such as Planet, ComDev, and Orbcomm actively participating in small satellite missions. These companies leverage small satellites for a wide range of applications, including Earth observation, communication services, and scientific research.

2. **Academic:** In the academic sector, universities and an increasing number of high school and organizations engaging in small satellite launches. These educational institutions use small satellites as platforms to promote hands-on learning and research in space-related fields.

3. **Civil:** In the civil sector, Prominent funding agencies and developers in the civil sector, such as NASA and NSF in the United States, ESA, DLR, and the Norwegian Space Centre in Europe, and the Indian Space Research Organization (ISRO) and JAXA, play a significant role in small satellite missions. These civil organizations contribute to small satellite missions, supporting scientific research, technology development, and national space initiatives.

4. **Military:** In this sector, organizations such as the United States Army, Air Force, Navy, and various military branches worldwide actively participate in small satellite missions. These military-based stakeholders utilize small satellites for both civil and military purposes, including communication, reconnaissance, and surveillance.

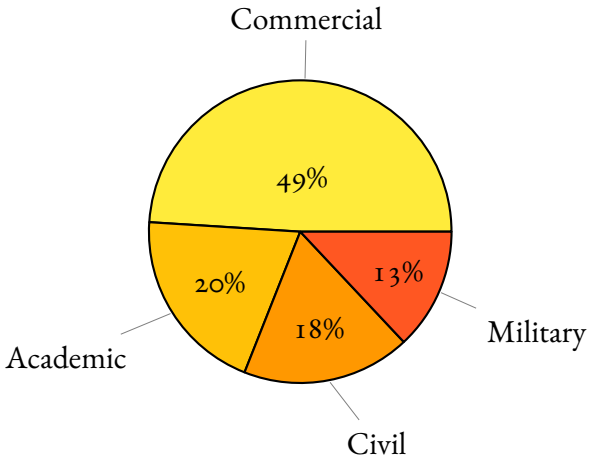


Figure 2.3: Funding Agency and Developers in Aerospace Corporation Database (2009 – 2018) [2]

The diverse range of funding agencies and developers in small satellite missions highlights the broad impact and collaborative nature of this evolving industry.

2.5 Subsystem and Composition

Small satellites are composed of various subsystems that work together to ensure the proper functioning and operation of the satellite in space. Each subsystem has specific functionalities, as reported in this section

1. **Command & Data Handling Subsystem (CDH):** The Command & Data Handling Subsystem has two main functions:

   - First, it receives, validates, decodes, and distributes commands to other spacecraft systems.
   - Second, it collects, processes, and formats spacecraft housekeeping and mission data for downlink or use by an onboard computer.

   This subsystem includes additional functions like spacecraft timekeeping, computer health monitoring, and security interfaces. The CDH subsystem acts as the "brain" of the satellite, and it consists of an Onboard Computer (OBC) that controls the operation of the satellite. The OBC software manages the programs written to handle various tasks, such as creating a telemetry stream about the status of the payload and then encoding the stream. The objective of the CDH subsystem is to provide the spacecraft with operational sequences for various subsystems. Due to size restrictions of small satellites, the

CDH subsystem needs to be efficient, small, lightweight, and easy to integrate with other subsystems.

2. **Attitude & Orbit Control Subsystem (AOCS):** The Attitude and Orbit Control Subsystem is an essential component of small satellite systems that is responsible for maintaining the orientation and stability of the spacecraft during its mission. The AOCS is used to determine and adjust the spacecraft's attitude, using sensors to monitor external disturbances such as solar radiation, gravity, magnetic fields, or atmospheric drag. If left unaddressed, these disturbances can cause the spacecraft to lose its orientation and fail to achieve its mission objectives. The AOCS system typically includes actuators that counteract these disturbances, either passively by exploiting the spacecraft's inherent characteristics or actively by measuring and applying corrective torques. The subsystem is also responsible for performing manoeuvres to reorient the spacecraft to a desired direction, such as pointing to a new target or communicate to the ground stations. By providing this crucial functionality, the AOCS subsystem ensures that the spacecraft remains stable and oriented correctly throughout its mission.

3. **Communications Subsystem or Telemetry, Tracking & Command Subsystem (TTC):** The communications subsystem of a satellite is responsible for ensuring telecommunication between the satellite and other systems, such as ground stations or other satellites. This subsystem utilizes electromagnetic pulses, which are transmitted by the satellite's transmitter and received by the ground station's receiver, to exchange data. The communication subsystem serves as the interface between the spacecraft and ground systems. It passes payload mission data and spacecraft housekeeping data from the spacecraft to operators and users at the operations center, while also receiving operator commands to control the spacecraft and payload operations. The communication subsystem itself consists of a set of antennae and transceivers, which allow for communication with monitoring stations. The instructions received from the ground station are processed by the satellite's control system, which can be the main computer or certain components of the CDH Subsystem. The communication subsystem also receives and demodulates up-link signals and modulates and transmits down-link signals.

4. **Payload Subsystem (PL):** The equipment that a spacecraft carries and that interacts with the subject to perform a specific mission is called the Payload. This equipment is unique to each mission and is the primary reason why the spacecraft is launched. The rest of the spacecraft is designed to support the payload and keep it in optimal conditions. Space missions encompass a wide range of objectives, including detection, communication, and interaction. The subject of a mission refers to the specific entity or target that the spacecraft will engage with in order to accomplish its goals. These missions can be broadly classified into categories such as communications, remote sensing, navigation, in situ science, and others. In scientific missions, remote sensing payloads are the most

common type. Remote sensing refers to any observation that a spacecraft makes without direct contact with the object in question. Measurements in the electromagnetic spectrum are used to determine the nature, state, or features of a physical object or phenomenon. These signals can either be produced by the subject or reflected, providing valuable information about a certain feature of the subject.

5. **Electrical Power & Distribution Subsystem (EPDS):** The Electrical Power and Distribution Subsystem is responsible for providing, storing, distributing, and controlling the electrical power needed by a spacecraft. To properly size each component of this subsystem, it is important to identify the electrical power loads required for mission operations at the beginning-of-life (BOL) and end-of-life (EOL). For most missions, the EOL power demands must be reduced to compensate for solar array performance degradation, so a detailed power budget must be done at different stages of the mission. There are three main elements to the EPDS:

    - *Power Source:* This component generates electrical power within the spacecraft. In most missions, photovoltaic solar cells are the most common power source used. They convert incident solar radiation directly to electrical energy and are well-known for their reliability.
    - *Energy Storage:* Any spacecraft that uses photovoltaic or solar thermal dynamics as a power source requires a system to store energy for peak-power demands and eclipse periods. Typically, energy storage occurs in a battery, although other systems such as flywheels and fuel cells have been considered in some cases.
    - *Power Distribution:* This component consists of cabling, fault protection, and switching gear to turn power on and off to the spacecraft loads. The design of power distribution systems for various power sources depends on source characteristics, load requirements, and subsystem functions. When selecting a type of power distribution, the focus is on keeping power losses and mass at a minimum while attending to survivability, cost, reliability, and power quality.

6. **Propulsion Subsystem (PROP):** The propulsion subsystem serves three main purposes. Firstly, it is responsible for lifting the launch vehicle from the surface and placing it into low-Earth orbit (LEO). Secondly, it transfers payloads from LEO to higher orbits or interplanetary trajectories. Finally, it provides crucial attitude control and orbit corrections throughout the mission.

7. **Structures & Mechanism Subsystem (SMS):** The Structures & Mechanism Subsystem is responsible for providing a framework to support and safeguard the spacecraft and its payload during launch and throughout its operational lifetime. The primary structure forms the backbone of the spacecraft and bears the main load, while the secondary

structure, such as brackets, closeout panels, and deployable components, supports the primary structure.

8. **Thermal Control Subsystem (TCS):** The Thermal Control Subsystem is responsible for ensuring that all components and subsystems of the spacecraft and payload remain within their required temperature limits during each mission phase. These limits include both a cold temperature and a hot temperature, with operational limits defining the range of temperatures that the component can tolerate while it is in use, and survival limits defining the range of temperatures that it must remain within at all times, even when it is not powered. Exceeding survival temperature limits can cause permanent damage to equipment, whereas exceeding operational limits may result in out-of-tolerance performance. There are two broad categories of thermal control techniques: passive and active.

    - Passive thermal control utilizes materials, coatings, or surface finishes like blankets or second surface mirrors to regulate temperatures. These measures help maintain temperature limits by reflecting or absorbing thermal radiation. Passive techniques are often employed in areas where temperature stability is crucial but active control is not necessary.
    - Active thermal control employs more complex means to actively regulate temperatures. This may include heaters or thermo-electric coolers that actively heat or cool specific components or subsystems. Active techniques are used when precise temperature control is required, especially for sensitive payloads or electronics that may be affected by temperature variations.

Interdependencies between subsystems play a remarkable role in diagnosing failures and anomalies. For instance, a failure in the power system can impact the operations of other subsystems, leading to the malfunctioning of the entire satellite. Similarly, if the attitude control subsystem encounters some issues, it can affect the pointing accuracy of the payload, impacting data acquisition or communication capabilities.

Understanding the interdependencies between subsystems is vital for troubleshooting and anomaly resolution. By analyzing telemetry data from various subsystems, engineers can identify the root cause of failures or anomalies and develop appropriate corrective measures. Additionally, a comprehensive understanding of the subsystems' functionalities and their significance in the overall satellite operation enables efficient system design, integration, and mission planning.

Figure 2.4 visually represents the intricate interconnections between the CDH subsystem and all other subsystems, showcasing the comprehensive network within the satellite architecture.

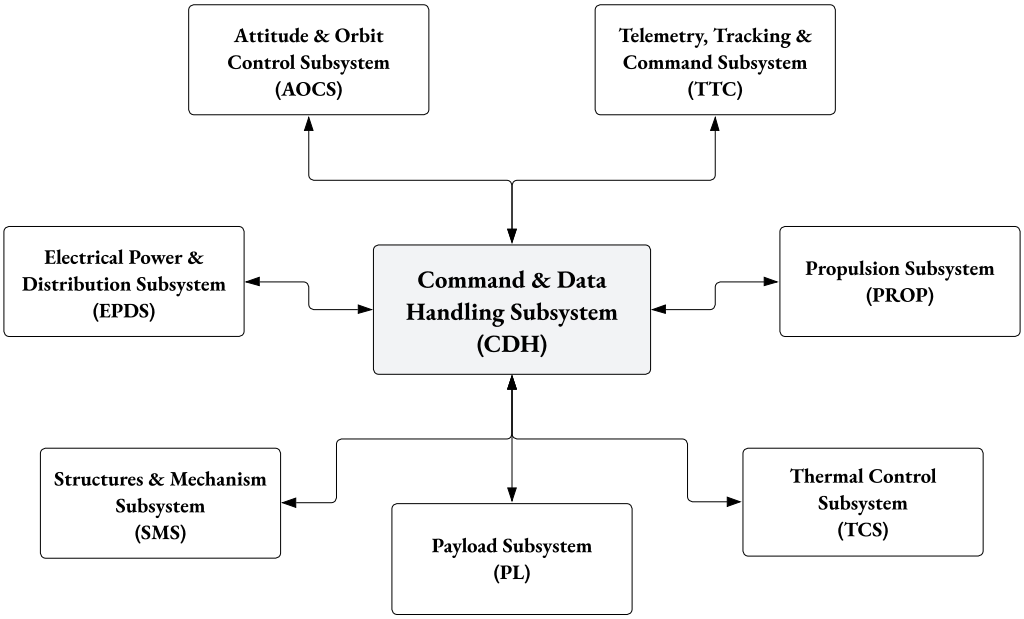


Figure 2.4: Satellite Subsystem Interconnections

2.6 Failures and Anomalies

2.6.1 Levels of Criticality in Satellites

When assessing failures and anomalies in small satellites, it is essential to consider the levels of criticality associated with these issues. These levels help to classify the severity of failures based on their impact on the overall mission. The four levels of criticality commonly used are:

- **Negligible:** Failures or anomalies with a negligible level of criticality have minimal impact on the satellite's operation and mission objectives. These issues may be minor and easily rectified without significant consequences. Examples of negligible failures could include minor fluctuations in power consumption or temporary communication signal interruptions that do not hinder the satellite's primary functions.

- **Non-Major:** Non-major failures or anomalies have a moderate impact on the satellite's operation but do not result in mission failure. These issues may affect certain subsystems or functionalities, but the satellite can continue to perform its primary tasks. For instance, a temporary glitch in attitude control accuracy or a minor data corruption issue may be classified as non-major failures.
- **Critical:** Critical failures or anomalies significantly impact the satellite's operation, potentially leading to mission failure or severe degradation in performance. These issues affect critical subsystems or functionalities, rendering the satellite unable to achieve its intended objectives. Examples of critical failures could include prolonged power system failures, loss of communication with the ground station, or major attitude control errors.
- **Catastrophic:** Catastrophic failures are severe and result in the complete loss of the satellite or render it entirely inoperable. These failures can have far-reaching consequences, including the loss of the entire mission, financial implications, and potential hazards to other satellites or objects in space. Catastrophic failures could occur due to catastrophic power system failures, catastrophic attitude control errors leading to collision risks, or unrecoverable communication system malfunctions.

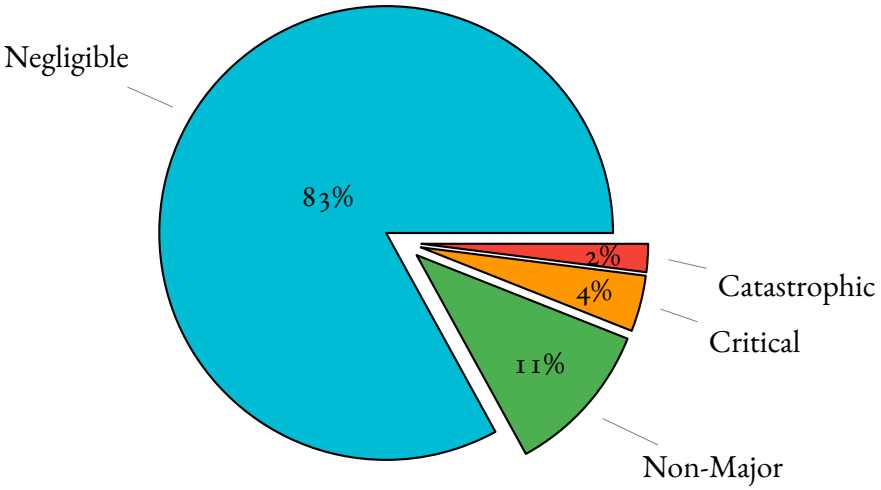


Figure 2.5: Distribution of Failure Criticality in China Spacecraft Database (2000-2017) [3]

From a statistical perspective, Figure 2.5 illustrates the distribution of failures based on their level of criticality.

Understanding these levels of criticality allows engineers and operators to prioritize and address failures and anomalies based on their potential impact. By categorizing failures, appropriate counter measures can be taken to mitigate risks, improve system reliability, and enhance

overall mission success. In the next subsection, this study will analyze common failures and anomalies observed in small satellites. These include power failures, communication issues, thermal management problems, and attitude control errors.

2.6.2 Causes of Failures in Small Satellites

Understanding the causes of failures in small satellites is key to improving their reliability and performance. Several factors contribute to the occurrence of failures and anomalies, including:

1. **Space Environment:** Approximately half of the flight failures in small satellites can be attributed to the harsh space environment. Factors such as radiation, extreme temperatures, micro-meteoroids, and vacuum conditions pose significant challenges to onboard systems and components. The space environment can induce malfunctions and performance degradation, affecting electronic devices, materials, and mechanisms. Radiation can cause electronic component failures or data corruption, while extreme temperatures can lead to thermal stress and material fatigue. Proper shielding, thermal management strategies, and robust design considerations are necessary to mitigate the effects of the space environment.

2. **Design:** Design-related factors contribute significantly to failures in small satellites. Inadequate shielding against radiation, insufficient thermal management, poor structural design, or improper integration of subsystems can lead to performance degradation or complete system failures. Design flaws can exacerbate the impact of environmental conditions and operational stresses, compromising the satellite's overall functionality. Ensuring robust design practices, thorough simulations, and proper consideration of environmental and operational factors are essential to minimize design-related failures.

3. **Parts and Materials:** Failures in small satellites can also be attributed to problems with parts and materials used in their construction. Defective or low-quality components, inadequate testing of materials for space conditions, or mismatches between component specifications and operational requirements can result in malfunctions and reduced system reliability. It is important to ensure the use of reliable and space-qualified parts and materials, conduct thorough testing and verification, and adhere to strict quality control measures during the manufacturing process.

4. **Software:** Although smaller in proportion, software-related failures can have significant consequences in small satellites. Errors in software code, algorithm implementation, or command sequences can lead to unexpected behavior, system crashes, or incorrect data processing. Robust software development practices, rigorous testing, and proper validation procedures are necessary to minimize the occurrence of software failures. Implementing redundancy, error handling mechanisms, and continuous software monitoring can enhance the resilience of the satellite's software systems.

5. **Workmanship:** Failures resulting from workmanship issues, such as manufacturing errors, assembly mistakes, or improper soldering, constitute a smaller portion of the total failures. Poor workmanship can introduce defects or weak points in the satellite's hardware, leading to premature failures or degraded performance. Strict quality control measures, adherence to manufacturing standards, and comprehensive testing and inspection protocols are crucial to reduce workmanship-related failures.

6. **Operations:** Operational failures, although relatively small in number, can still impact satellite performance. Human errors, incorrect procedures, or sub-optimal operational practices can lead to anomalies, communication disruptions, or unintended system configurations. Ensuring proper training, clear operational guidelines, and effective monitoring procedures can help mitigate operational failures and improve the overall reliability of small satellites.

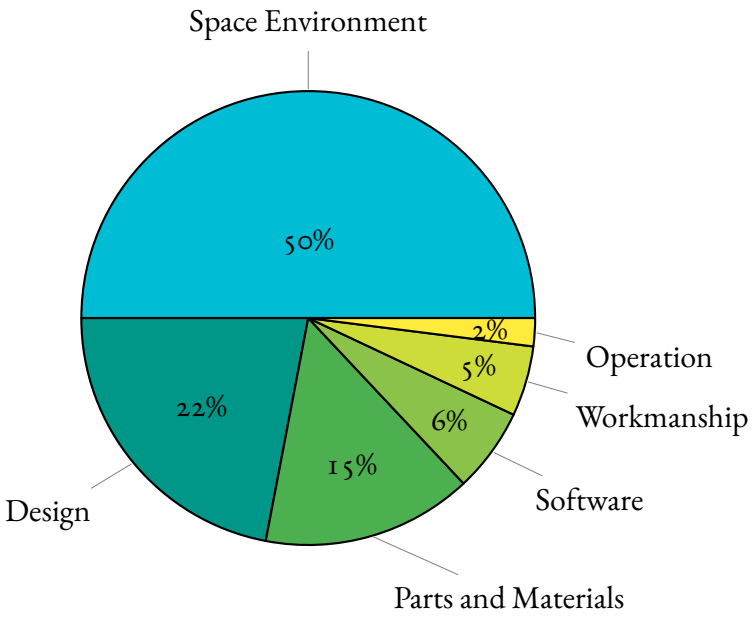


Figure 2.6: Distribution of On-Orbit Failure Causes in China Spacecraft Database (2000-2017) [3]

Figure 2.6 illustrates the distribution of failures in small satellites based on their causes. The chart indicates that approximately 50% of failures are attributed to the space environment, 22% to design-related issues, and 15% to problems with parts and materials. The remaining failures are attributed to software, workmanship, and operational factors.

By mitigating these factors through robust design practices, rigorous testing, proper component selection, and effective operational procedures, developers and operators can enhance the overall reliability of small satellite systems.

2.6.3 Common Failures and Anomalies in Small Satellites

Small satellites are susceptible to various failures and anomalies that can significantly impact their operational effectiveness and mission outcomes. The following are common failures and anomalies observed in small satellites.

- Power Failures: Power failures can occur due to degraded solar panels, battery malfunctions, or problems in power distribution systems. These failures result in insufficient power supply, reduced payload performance, or complete power loss, rendering the satellite inoperable.

- Communication Issues: Failures in transceivers, antennas, or data processing units can disrupt data transmission, command reception, and real-time monitoring of the satellite. Communication issues impair the satellite's ability to receive commands, transmit telemetry data, or retrieve critical information from the spacecraft.

- Thermal Management Problems: Inadequate heat dissipation, malfunctioning thermal control mechanisms, or environmental factors can lead to thermal management problems. Excessive heat or cold can cause component degradation, reduced performance, or permanent equipment damage.

- Attitude Control Errors: Failures in sensors, actuators, or control algorithms within the AOCS can result in attitude control errors, leading to significant deviations from the desired attitude or loss of control altogether. These failures hinder precise pointing, required maneuvers, and accurate mission execution.

- Payload Malfunctions: Failures in sensors, instruments, or data processing units within the payload can compromise data acquisition, analysis, or transmission. This impacts the quality and quantity of scientific or operational results, hindering the achievement of mission objectives.

- Command and Data Handling Issues: Failures within the CDH subsystem can disrupt critical communication and data processing functions. Malfunctions in the onboard computer, data validation systems, or security interfaces hinder command distribution and mission/housekeeping data processing.

- Structures and Mechanism Failures: Failures in the primary or secondary structure, as well as malfunctioning deployment mechanisms, can compromise satellite stability and functionality. Effectively mitigating structural damage or deployment issues ensures overall performance and reliability.

- Propulsion System Malfunctions: Failures in the Propulsion (PROP) subsystem can limit satellite maneuverability and its ability to maintain the desired orbit or adjust attitude. Engine anomalies or fuel system issues require prompt handling and mitigation for mission success.

By recognizing and addressing these common failures and anomalies, the reliability and performance of small satellites can be improved, increasing the chances of successful mission outcomes.

2.6.4 Critical Subsystems and Components in Small Satellites

This section presents the research findings on the identification of the most critical subsystems and components in small satellites. When considering criticalities, the evaluation encompasses both the frequency of failures and the level of impact these failures have on mission outcomes. It is crucial to distinguish between failures that can lead to catastrophic or mission-ending consequences and failures that may only result in the loss of certain satellite features.

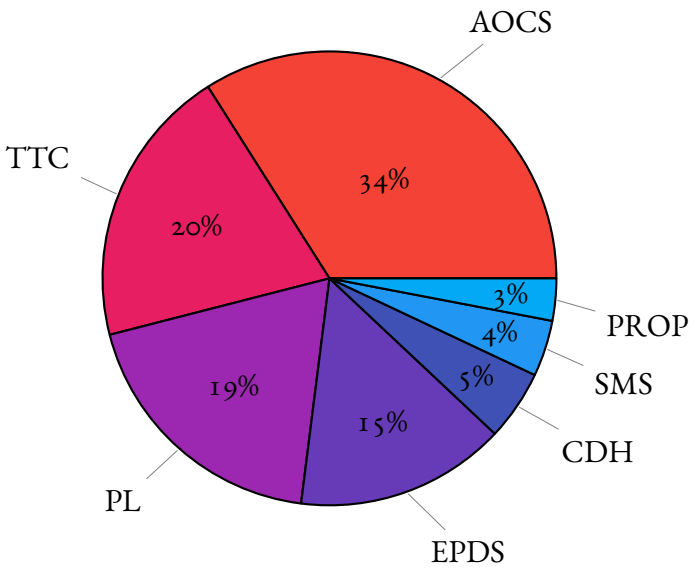


Figure 2.7: Affected Subsystem by Failure in China Spacecraft Database (2000-2017) [3]

Figure 2.7 illustrates the results obtained from the analysis of satellite data from the "China Spacecraft Database", covering launches from 2000 to 2017. The subsystems with the highest number of failures, are AOCS, TTC, PL, and EPDS, while CDH, SMS, and PROP exhibit the fewest failures. This analysis highlights AOCS as the most vulnerable subsystem to failures. It

encompasses critical systems that play a pivotal role in spacecraft operations, and any malfunctions within AOCS can have severe consequences, potentially compromising overall spacecraft functionality. Ensuring the reliability and robustness of the AOCS subsystem is essential for maintaining precise attitude and orbit control.

Although the TTC, EPDS, and PL subsystems experience fewer failures compared to AOCS, it is important to note that failures within these subsystems can still significantly impact core functions and mission objectives. These subsystems are paramount for spacecraft operations, and their smooth functioning is crucial for mission success. Therefore, it is essential to implement measures to mitigate the risk of failures in these subsystems.

On the other hand, the PROP, SMS, and CDH subsystems demonstrate a lower failure rate. This suggests that these subsystems have implemented effective redundancy measures and protection mechanisms to withstand the challenging space environment. The relatively lower incidence of failures in these subsystems indicates that they were designed and implemented with adequate safeguards, resulting in minimal mission degradation.

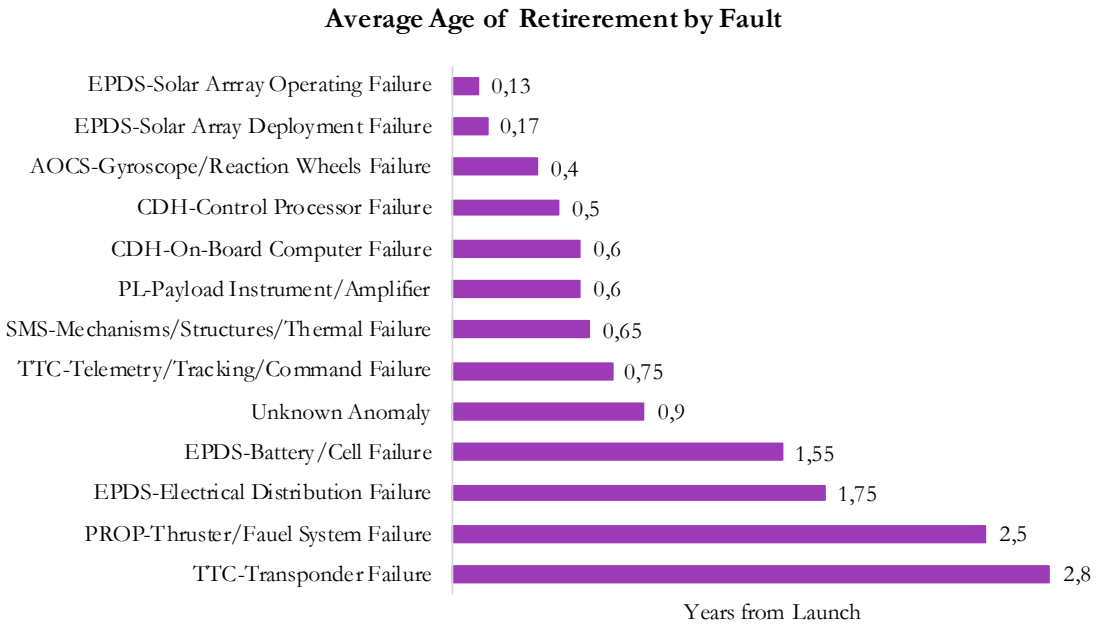


Figure 2.8: Average Age of Retirement by Fault in Aerospace Corporation Database (2009 – 2018) [2]

This research also provides significant insights into the characteristics of component failures within their respective subsystems. Figure 2.8, based on an analysis of data from the ”Aerospace Corporation Database” covering satellite launches from 2009 to 2018, presents the average re-

tirement age of satellites resulting from component failures. Notably, the components with the shortest average lifespan, are in order the solar arrays (part of the EPDS subsystem), followed by the gyroscope and reaction wheels (part of the AOCS subsystem). Subsequent failures include the on-board computer and control processes within the CDH subsystem, as well as payload instruments and components within the SMS and TTC subsystems.

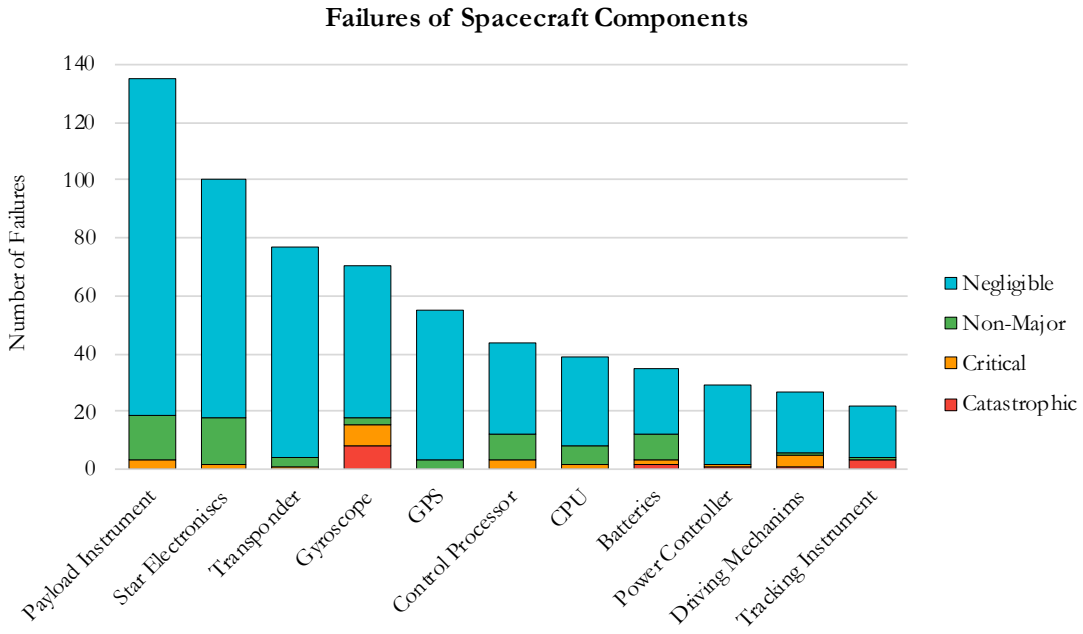


Figure 2.9: Failures of Spacecraft Components in China Spacecraft Database (2000-2017) [3]

To delve deeper into the criticality of these component failures, Figure 2.9, derived from the same analysis of the "China Spacecraft Database", categorizes the failures into catastrophic, critical, non-major, and negligible. Among the components, payload instruments, star electronics, and transponders exhibit the highest number of failures. However, as mentioned earlier, it is crucial to assess the critical implications of these failures. Figure 2.10) demonstrates that the gyroscope emerges as one of the most critical components due to its substantial occurrence of catastrophic and critical failures (12% and 10%, respectively), surpassing other components in terms of criticality.

These findings emphasize the importance of robust design, maintenance strategies, and mitigation measures for critical subsystems and components. Identifying and addressing vulnerabilities and critical aspects of component failures is vital to enhance the reliability and resilience of small satellites. By understanding the failure patterns and their critical implications, satellite

operators can optimize component selection, implement appropriate redundancy measures, and develop effective maintenance protocols, ultimately ensuring the success of space missions.

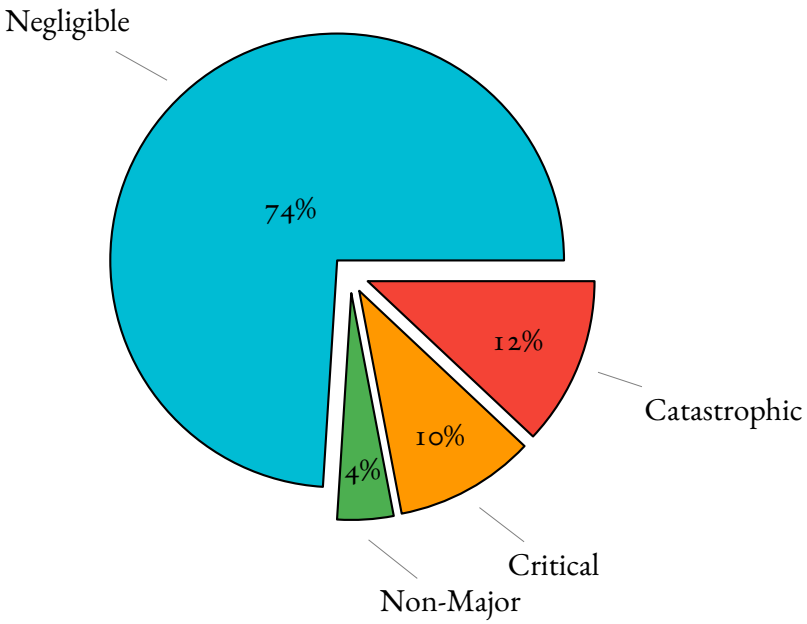


Figure 2.10: Failures of Gyroscope in China Spacecraft Database (2000-2017) [3]

2.7 Summary of Findings

This chapter presents a comprehensive analysis of small satellites, focusing on their subsystems, failures, and critical components. The findings offer valuable insights into the challenges and considerations associated with small satellite missions and provide a foundation for the development of effective diagnostic techniques using AI and ML.

The key findings can be summarized as follows:

1. Small satellites comprise various subsystems that perform crucial functions in space. These include the Command and Data Management (CDH) Subsystem, Attitude and Orbit Control Subsystem (AOCS), Communications Subsystem, Payload (PL) Subsystem, Electrical Power Subsystem and Distribution (EPDS), Propulsion Subsystem (PROP), Structures and Mechanisms Subsystem (SMS), and Thermal Control Subsystem (TCS). Each subsystem plays a vital role in the overall performance and reliability of small satellites.

2. Failures and anomalies can occur in small satellites, affecting their operational capabilities. Examples include power outages, communication disruptions, thermal manage-

ment issues, and attitude control errors. Among the subsystems, the AOCS subsystem is particularly susceptible to failures due to its critical role in spacecraft stabilization and orientation. Failures in the TTC subsystem can impact communication links with ground stations, while failures in the PL subsystem directly affect mission objectives. EPDS failures significantly impact power generation and distribution, and propulsion system failures limit maneuverability and attitude control.

3. The analysis of critical subsystems and components reveals the following insights:

   (a) The subsystems with the highest number of faults are AOCS, TTC, PL, and EPDS.

   (b) The components with the lowest average retirement age are the solar arrays (EPDS components) and the gyroscope and reaction wheels (AOCS components).

   (c) The gyroscope (an AOCS component) exhibits the highest number of critical failures.

   These findings highlight the critical nature of subsystems like AOCS and EPDS, as well as components such as gyroscope and reaction wheels. Ensuring the proper functionality of these critical components and implementing appropriate measures are crucial for accurate attitude control and orientation.

3

Diagnosis, Prognosis, and Health Monitoring (DPHM)

3.1 Introduction to DPHM

Diagnosis, Prognosis, and Health Monitoring (DPHM) are processes exploited for ensuring reliable performance and safety of operating systems or components. DPHM encompasses a range of methods and processes aimed at detecting faults, predicting system behavior, and monitoring the health of that aim systems in real-time. By employing advanced algorithms and AI techniques, DPHM enables proactive maintenance, reduces downtime, and enhances operational efficiency.

However, traditional reactive maintenance approaches are no longer sufficient to address the demands of modern complex systems, included the aerospace systems.

The implementation of DPHM in aerospace systems brings several benefits. It enables early fault detection, allowing maintenance teams to intervene before the fault escalates and causes extensive damage. Moreover, DPHM facilitates improved decision-making regarding maintenance actions, ensuring optimal resource allocation and minimizing operational disruptions. By employing advanced algorithms and AI techniques, DPHM can enable proactive maintenance, reduce downtime, and enhance operational efficiency.

3.2 Fault Diagnosis

Fault diagnosis refers to the process of identifying symptoms of anomalous behaviour of systems or components and determining the root cause of this anomalous behaviour. It consists in analyzing the observed symptoms or deviations from expected behavior to locate the specific error or failure responsible for the observed problems. The objective of fault diagnosis is to accurately and efficiently identify the underlying problem or fault in order to enable timely and effective corrective actions.

Fault diagnosis consists of two main steps: fault detection and isolation.

1. *Fault Detection* involves monitoring and analyzing system parameters, such as sensor readings, control signals, or performance metrics, to detect deviations from normal operation. This is achieved by comparing observed values with expected or reference values obtained from system models or historical data. Statistical techniques, signal processing methods, and ML algorithms can be utilized to identify anomalies and trigger alarms when predefined thresholds or patterns are exceeded.

2. Once a fault is detected, the subsequent step is to isolate the fault to a specific component or subsystem. *Fault Isolation* aims at determining the root cause of the fault by analyzing available data and system behavior. Various diagnostic techniques can be employed to narrow down the potential sources of the fault, such as analyzing system responses, examining fault signatures, or employing logical reasoning based on system models. The objective is to accurately identify the faulty component or subsystem, enabling targeted maintenance actions and minimizing system downtime.

3.2.1 Four Types of Fault Diagnosis Methods

Fault diagnosis can be approached using various methods, which can be classified into four distinct categories.

1. **Model-Based:** Model-based fault diagnosis methods utilize mathematical models of the system to detect and isolate faults. These methods rely on system knowledge, such as physical principles or system dynamics, to develop analytical or numerical models. By comparing the predicted system behavior with the observed data, model-based approaches can identify deviations and attribute them to specific faults. Techniques like fault signature analysis, parameter estimation, and residual generation are commonly employed in model-based fault diagnosis. Model-based methods can be very accurate and efficient, but they require a good understanding of the system and its behavior.

2. **Data-Driven:** Data-Driven fault diagnosis methods rely on ML and data analytics techniques to detect and isolate faults. These methods utilize historical data collected from sensors and other monitoring sources to build models and algorithms that can identify patterns associated with specific fault conditions. Data-Driven approaches include techniques like artificial neural networks, support vector machines, decision trees, and clustering algorithms. They excel in handling complex, non-linear relationships and can adapt to changing system behaviors. Data-Driven methods are often used when there is not enough knowledge about the system to develop an accurate model, or when the system's behavior is too complex to model accurately. Data-driven methods can be very effective in identifying faults, but they require large amounts of data and may be less accurate than model-based methods.

3. **Signal-Based:** Signal-Based fault diagnosis methods focus on analyzing the system's input and output signals to detect and isolate faults. These methods leverage signal processing techniques, statistical analysis, and pattern recognition algorithms to identify abnormal patterns or deviations in the signals. Signal-based approaches often involve frequency analysis, time-frequency analysis, statistical hypothesis testing, and pattern recognition algorithms to detect and isolate faults. Signal-Based methods are often used when the system is complex and difficult to model, or when there is limited data available. Signal-Based methods can be very effective in identifying faults, but they require specialized knowledge of the system and its signals.

4. **Hybrid:** Hybrid fault diagnosis methods combine multiple approaches, such as Model-Based, Signal-Based, and Data-Driven techniques, to achieve more robust and accurate fault detection and isolation. By integrating complementary methodologies, hybrid methods aim to leverage the strengths of each approach and mitigate their limitations. These methods can enhance fault diagnosis performance by considering multiple aspects of the system, combining diverse information sources, and incorporating expert knowledge.

Overall, the choice of fault diagnosis method depends on the specific characteristics of the system analyzed, as well as the available data, expertise and computational resources. Figure 3.1 illustrates the categorization of various fault diagnosis methods.

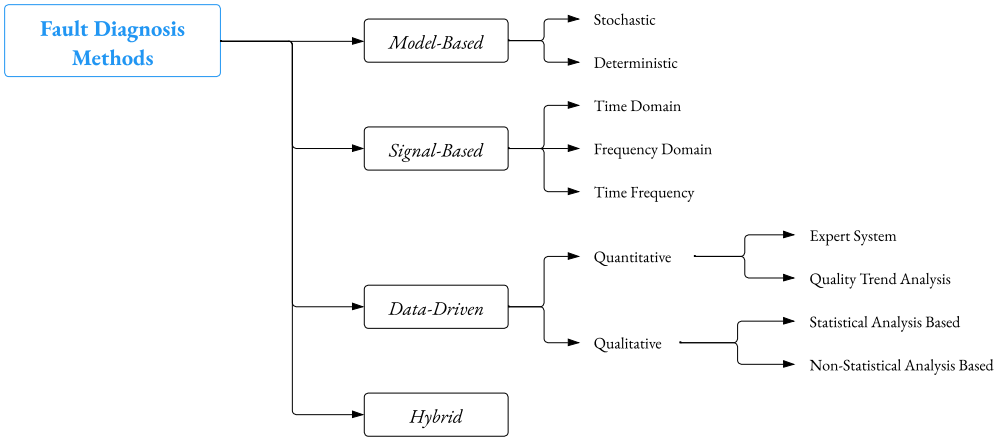


Figure 3.1: Fault Diagnosis Methods Overview

3.3 Fault Prognosis

Fault prognosis refers to the process of estimating the Remaining Useful Life (RUL) or the Estimated Time To Failure (ETTF) of a system or component. It involves predicting the future behaviour, performance, or health condition of the system based on the analysis of historical data and the current state of the system. Fault prognosis aims at providing early warnings or alerts about impending failures, allowing for proactive maintenance actions to be taken to prevent or mitigate the consequences of the failure. The objective of fault prognosis is to provide accurate and timely information about the expected future behaviour of the system, including the time at which a failure is likely to occur. This information enables decision-makers to plan maintenance activities, optimize resource allocation, and minimize downtime and costs associated with unexpected failures.

3.3.1 Four Types of Fault Prognosis Methods

As with fault diagnosis, fault prognostics can also be addressed using various methods, which can be classified into four distinct categories:

1. **Model-Based:** Model-based methods utilize mathematical models that represent the behavior and dynamics of the system under normal and faulty conditions. These models are typically developed based on physical principles, system specifications, or empirical

data. By simulating the system's behavior over time, Model-Based methods can forecast its degradation and predict potential faults. These methods are particularly effective when accurate models are available and when the system's behavior can be well-characterized.

2. **Data-Driven:** Data-Driven methods leverage historical data collected from the system to make predictions about its future behavior. These methods utilize statistical analysis, ML, and pattern recognition techniques to extract patterns, trends, and anomalies from the data. By analyzing the temporal or spatial relationships in the data, Data-Driven methods can forecast potential faults or degradation. Data-Driven methods are particularly advantageous when sufficient historical data is available, allowing for robust and accurate predictions.

3. **Knowledge-Based:** Knowledge-Based methods rely on expert knowledge and domain-specific rules to make predictions about the future behavior of a system. These methods utilize predefined patterns, rules, or heuristics derived from experience or existing knowledge bases. Knowledge-Based methods are beneficial when there is a wealth of domain expertise available and when the system's behavior can be effectively captured through logical rules or expert opinions.

4. **Hybrid:** Hybrid methods combine multiple approaches, such as Model-Based, Knowledge-Based, and Data-Driven methods, to achieve more accurate and comprehensive fault prognosis. By leveraging the strengths of different techniques, Hybrid methods can overcome limitations and enhance the predictive capabilities of fault prognosis systems. These methods often integrate multiple data sources, domain knowledge, and advanced algorithms to provide a holistic and reliable prognosis.

Overall, the choice of fault prognosis method depends on the specific characteristics of the system or being analyzed, as well as the available data, knowledge, and expertise. Figure 3.2 provides representation of the classification of different failure prognosis methods.

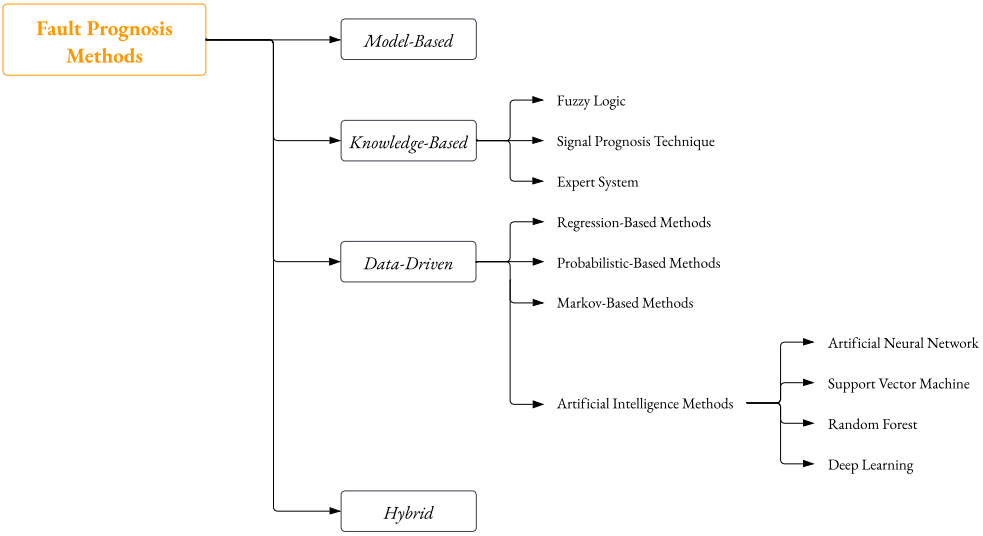


Figure 3.2: Fault Prognosis Methods Overview

3.4 Maintenance Strategies

Maintenance strategies play a crucial role in keeping sufficient level of performance and extending the life of systems and components. Two commonly employed strategies are Time-Based Maintenance and Condition-Based Maintenance. This section delves into the exploration of these strategies and their application within the context of DPHM.

- **Time-Based Maintenance (TBM):** TBM is a preventive maintenance strategy that involves performing maintenance activities at predetermined intervals or fixed time intervals. These intervals are typically determined based on historical data, manufacturer recommendations, or regulatory requirements. TBM aims to prevent failures by replacing components or conducting maintenance tasks before their expected useful life is exceeded. TBM offers several advantages, including simplicity in planning and scheduling maintenance activities, compliance with regulatory requirements, and the ability to address known failure modes. However, it can also result in unnecessary maintenance actions and associated costs when components or systems are replaced before reaching the end of their useful life. Moreover, TBM may not effectively capture the actual degradation and health condition of the system, leading to potential risks or unexpected failures.

- **Condition-Based Maintenance (CBM):** CBM is a proactive maintenance strategy that relies on real-time monitoring and assessment of the system's condition. CBM utilizes

sensor data, diagnostics, and prognostics to detect anomalies, assess the health of components or systems, and determine the optimal timing for maintenance actions. By monitoring the actual condition of the system, CBM aims to minimize unnecessary maintenance and maximize the operational lifespan of components. CBM offers several advantages over TBM. It allows for maintenance actions to be performed only when necessary, based on the actual health condition of the system. This approach reduces maintenance costs, optimizes resource allocation, and minimizes downtime. CBM also enables early detection of faults or degradation, facilitating proactive decision making and minimizing the risk of unexpected failures. However, implementing CBM requires advanced sensor systems, data analysis capabilities, and predictive models to accurately assess the system's condition and predict maintenance needs.

In practice, a combination of TBM and CBM strategies can be employed to optimize maintenance activities and achieve the most effective maintenance outcomes. This integrated approach leverages the strengths of both strategies, taking advantage of the predictability of TBM for certain failure modes while incorporating the real-time condition monitoring and proactive decision making of CBM.

By integrating TBM and CBM, maintenance actions can be tailored for the specific needs and health condition of each component or system. This hybrid approach optimizes maintenance scheduling, reduces costs, minimizes downtime, and enhances the overall reliability and performance of aerospace systems.

In the context of DPHM, the selection of an appropriate maintenance strategy depends on various factors, including the criticality of the components, system complexity, available monitoring capabilities, and operational requirements.

3.5 Recovery and Preventive Actions

In the field of Diagnosis, Prognosis, and Health Monitoring (DPHM) for aerospace systems, the process of fault diagnosis and fault prognosis provides valuable insights into the health condition of the system. However, it is equally important to take appropriate actions based on the diagnostic and prognostic results to ensure the safe and reliable operation of the system. In this section, the focus will be on exploring recovery actions and preventive actions, emphasizing their significance and application in both general and space environments.

Recovery Actions

Recovery actions refer to the corrective measures taken after the fault detection and isolation to restore the system to its normal or desired operating state. These actions aim to rectify the identified faults, eliminate any adverse effects, and bring the system back to its optimal performance. Depending on the nature of the fault, recovery actions can involve various steps such as component replacement, system reconfiguration, software updates, or adjustments in operational parameters.

In the context of space missions, recovery actions are particularly critical due to the harsh and unforgiving environment of space. The remoteness of space systems and the limited ability for human intervention necessitate efficient and autonomous recovery procedures. Space agencies and satellite operators employ specialized protocols and procedures to address faults and anomalies encountered in space systems. These protocols often involve onboard diagnostic systems, redundancy mechanisms, and automated recovery sequences to mitigate the impact of faults and ensure mission continuity.

Preventive Actions

Preventive actions, as the name suggests, are measures taken proactively to minimize the occurrence of faults, failures, or degradation in the system. These actions aim to increase the system's reliability, extend its lifespan, and reduce the likelihood of unexpected failures. Preventive actions can include maintenance activities, system upgrades, design modifications, or changes in operational procedures.

Satellite operators and space agencies employ comprehensive maintenance programs and rigorous quality control measures to prevent faults and ensure the long-term operation of space systems. These programs often include regular inspections, scheduled maintenance tasks, component replacements based on predicted lifetimes, and continuous monitoring of critical parameters to detect early signs of degradation.

The implementation of preventive actions in space systems is often guided by historical data, predictive models, and expert knowledge. By taking proactive measures, space operators can optimize system performance, reduce the risk of mission failure, and enhance the overall reliability and availability of satellites and spacecraft.

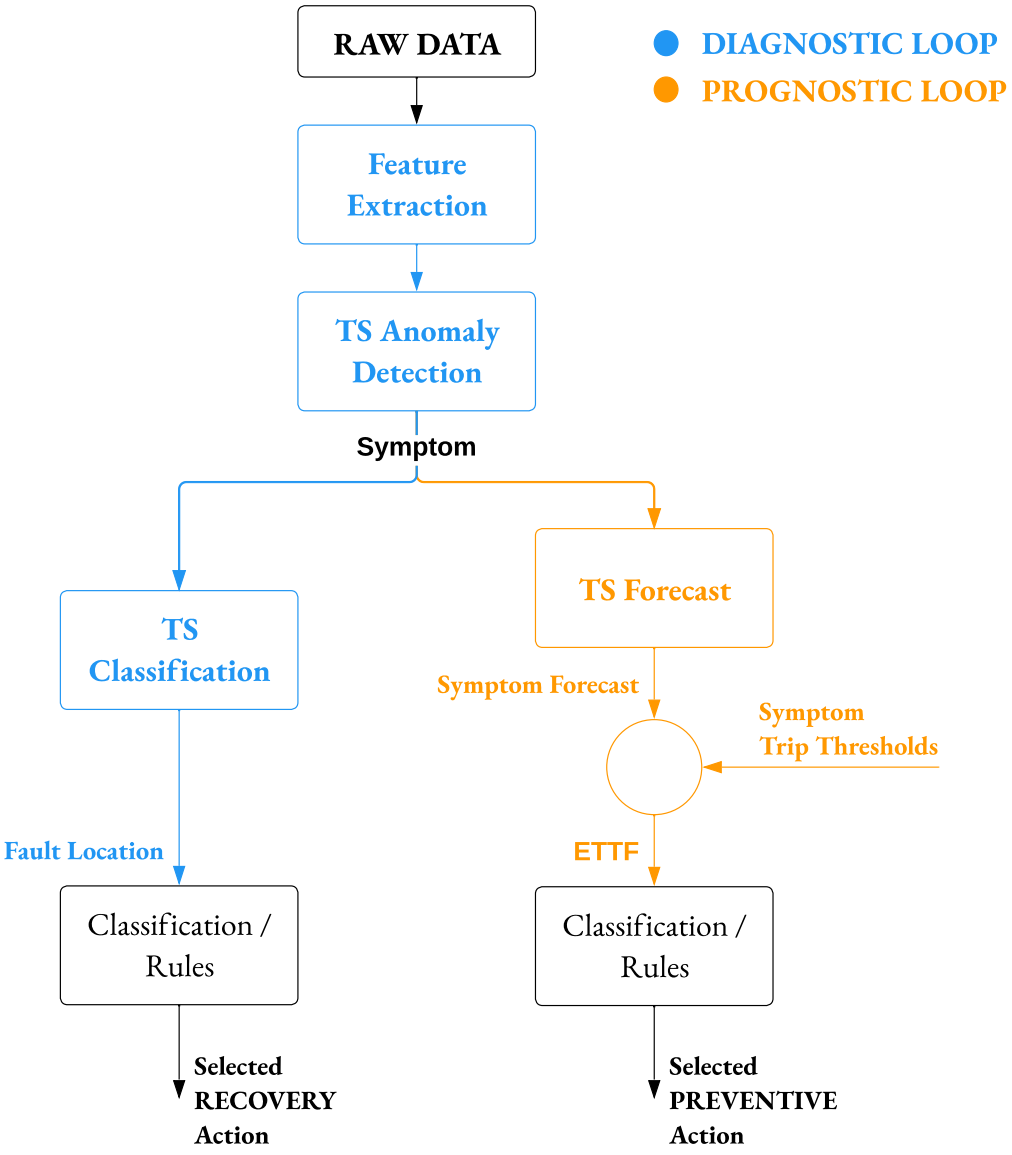


Figure 3.3: Diagnostic and Prognostic Loop

Figure 3.3 combines both diagnostic and prognostic processes. The blue section represents the diagnostic loop, involving a series of sequential stages such as processing raw data, extracting relevant features, detecting anomalies in time series, identifying symptoms, classifying anomalies, locating faults, and selecting appropriate recovery actions. This cycle enables comprehen-

sive analysis and resolution of abnormalities within the system. Within the same figure, the orange section depicts the prognostic loop, which focuses on predicting future symptoms using historical data, intervening when symptom prediction thresholds are reached, estimating the remaining time before failure, and selecting preventive actions. This proactive cycle facilitates the anticipation and prevention of potential failures, ensuring timely interventions and enhancing the overall system's reliability.

3.6 DPHM Methods for Space Systems

In this section, the findings of an extensive literature search on the most widely used DPHM methods in the space environment, with a specific focus on satellite applications, are presented. The research aimed to identify the prevalent techniques and algorithms employed for fault diagnosis, prognosis, and overall DPHM, providing an overview of the field.

The initial focus of the research was on exploring specific methods employed for fault diagnosis. These methods encompass a range of approaches, including data-driven, model-based, and knowledge-based techniques. Notably, data-driven methods emerged as the most widely used approaches in the space environment.

Within the model-based approach, two noteworthy works were identified. In 2011, J. Schumann [4] introduced an integrated software and sensor health management system tailored for small spacecraft, utilizing Bayesian networks. Similarly, in 2013, S. Xie [5] put forward a fault diagnosis technique for the satellite power system, also relying on Bayesian networks. These probabilistic graphical models have proven effective in capturing complex dependencies and relationships among system variables.

Data-driven approaches, on the other hand, have largely replaced model-based methods in the realm of space exploration. Due to the intricate nature of satellite systems and the intricate interactions between their components, model-based techniques are infrequently employed. The benefits of data-driven approaches have proven to be more advantageous, as they offer greater adaptability, capture complex relationships, and provide valuable insights that traditional models cannot achieve. In fact, the data-driven approach encompasses a wide range of works. In 2015, D.Pan [6] introduced an anomaly detection technique for the satellite power subsystem using Kernel Principal Component Analysis (KPCA) and associated rules. In 2017, T.Yairi [7] proposed a data-driven health monitoring method for satellite housekeeping data based on probabilistic clustering and dimensionality reduction. H.Fang [8], in the same year, presented a spacecraft power system fault diagnosis method based on a Deep Neural Network

(DNN). M.Suo [9] also contributed to fault diagnosis in 2017 by utilizing a variable precision fuzzy neighborhood rough set model. In 2019, S.Dheepadharshani [10] proposed a multivariate time-series classification approach for automated fault detection in the satellite power system, combining KPCA and a multilayer perceptron. In 2020, S.K.Ibrahim [11] used machine learning techniques such as K-means clustering, logistical analysis of data, and fault tree analysis for satellite fault diagnosis. B.Xiao and S.Yin [12], in the same year, presented a deep learning-based data-driven approach for thruster fault diagnosis in satellite attitude control systems. Finally, the knowledge-based approach includes one paper from 2021. S. Mengqi [13] introduced a fuzzy-based analysis of thermal effects on component failure for low Earth orbit satellites using fuzzy logic.

While fault diagnosis techniques play a fundamental role in detecting and isolating faults, this investigation extended beyond that to encompass comprehensive DPHM methods commonly utilized in the spatial domain. Aligned with the research objectives, the focus was specifically placed on ML and AI techniques, aiming to harness the unique advantages provided by these advanced approaches.

Under the model-based category, the work of A. Rahimi [14] in 2020 focuses on failure prognosis for satellite reaction wheels. The proposed method combines Kalman and Particle filters for accurate prognostication.

In the data-driven category, several works are listed. J. Wang [15], in 2015, presents a prognosis method for on-orbit satellite momentum wheels using a neural network. C. Zhang [16], in 2017, introduces a multiobjective deep belief networks ensemble approach for remaining useful life estimation in prognostics. X. Li [17], in 2018, proposes a Deep Convolution Neural Network (DCNN) method for estimating remaining useful life in prognostics. K. Hundman [18], also in 2018, suggests the use of LSTMs and non-parametric dynamic thresholding for detecting spacecraft anomalies. J.Dong [19], in 2019, focuses on deep learning-based multiple sensors monitoring and abnormal discovery for the satellite power system using LSTMs. C.-G. Huang, [20], in the same year, presents a bidirectional LSTM prognostics method under multiple operational conditions. D. Pan [21], in 2020, presents a satellite telemetry data anomaly detection approach using a bi-directional LSTM prediction-based model. In 2021, Y. Wang [22] introduced an anomaly detection method specifically designed for spacecraft telemetry data. The proposed approach utilizes a Temporal Convolutional Neural Network (TCNN) to identify and flag any abnormal patterns or behavior in the data. In the same year, J. Murphy [23] conducted a comprehensive review focusing on machine learning algorithms and hardware for space applications. The review particularly highlights LSTM-based methods used for

fault prognosis in space. It provides valuable insights into the use of LSTM models for analyzing and predicting faults in space systems. Also in 2021, M.Sirajul Islam and A.Rahimi [24] presented a fault prognosis technique for satellite reaction wheels. The proposed approach utilizes a Two-Step LSTM Network to forecast and anticipate potential failures in the reaction wheels. M. ElDali and K. D. Kumar [25] introduced a fault diagnosis and prognosis approach for aerospace systems in the same year. The method employs a Growing Recurrent Neural Network (GRNN) and LSTM to diagnose faults and provide prognostic insights, enabling proactive maintenance and efficient system management. In 2021, D. Han [26] proposed a remaining useful life prediction approach specifically for spacecraft bearings. The method utilizes low-frequency current data and combines an autoregression model, backpropagation neural network, and CNN to accurately estimate the remaining useful life of the bearings. Furthermore, C.Wang [27] presented a data-driven degradation prognostic strategy for aero-engines operating under various operational conditions. The approach incorporates a deep forest classifier and LSTM to effectively predict the degradation of aero-engines and provide timely maintenance recommendations. Z. Zeng [28], in 2022, focuses on satellite telemetry data anomaly detection using a Causal Network and Feature-Attention-based LSTM (CN-FA-LSTM). A.-E. R. Abd-Elhay [29], in the same year, presents a reliable deep learning approach for time-varying fault identification, using a one-dimensional CNN with LSTM, with a case study on spacecraft reaction wheels. V. Muthusamy and K. D. Kumar [30], in 2022, discusses failure prognosis and remaining useful life prediction of control moment gyroscopes on-board satellites, employing a general path model and Bayesian updating. Finally, X. Wei [31], in 2022, proposes a fault diagnosis method for spacecraft control systems based on Principal Component Analysis and Residual Network (PCA-ResNet).

It is evident that the identified methods predominantly fall under the data-driven paradigm, with only one example incorporating a model-based approach. Within the realm of data-driven algorithms, LSTM (Long Short-Term Memory) emerges as the most widely adopted algorithm, manifesting in its fundamental formulation as well as various iterations such as Bi-LSTM, Two-Step LSTM, and VarLSTM. The popularity of LSTM can be attributed to its ability to effectively capture temporal dependencies and handle sequential data, making it a favored choice in DPHM applications.

In addition to LSTM, other neural network-based algorithms have been employed in the spatial domain. These include Deep Convolutional Neural Network (DCNN), Temporal Convolution Neural Network (TCNN), Growing Neural Network (GNN), and Convolutional Recurrent Neural Network (CRNN). By harnessing the power of deep learning, these algo-

rithms excel in extracting high-level features and detecting intricate patterns from sensor data or telemetry parameters.

The adoption of these diverse algorithms signifies the growing significance of ML and AI techniques in DPHM applications within the space environment. This trend highlights the recognition of the potential of ML and AI in enhancing fault diagnosis, prognosis, and overall system health management in space-related contexts.

3.7 Summary of Findings

After a thorough analysis of the popular methods and algorithms used in DPHM for the spatial environment, a critical stage in the research was encountered: the decisive selection of an algorithm to serve as the fundamental framework for the DPHM system. Based on the knowledge gained throughout the research, the decision to utilize the LSTM network was made. This selection is motivated by several factors:

- First, as shown in the previous section, LSTM emerges as the most widely used network in the field of DPHM. Its wide adoption, coupled with its growth and use beyond the boundaries of the space industry, attests to its versatility and ability to capture temporal dependencies and effectively handle sequential data.
- Moreover, the sustained popularity of LSTM, both within and outside the space industry, reinforces its position as a reliable choice for DPHM applications. This sparked the interest of S.A.T.E., which expressed a desire to explore and evaluate the capabilities of LSTM. In fact, since S.A.T.E. during the last two decades developed proprietary methods that are part of diagnostics and prognostics commercial solutions, the comparison between the mentioned proprietary methods against state-of-the-art techniques (such as the LSTM) can provide added value to the company. Consequently, the decision to adopt LSTM for the DPHM system aligns with industry trends and presents an opportunity for S.A.T.E. to delve into the practical implementation and evaluation of this network using real satellite data.

The next chapter delves into a detailed exploration of the intricate architecture of LSTM, providing a thorough analysis of its advantages and limitations. Real-world case studies where LSTM has demonstrated successful application are studied to gain valuable insights and knowledge. These findings will serve as a foundation for implementing a tailored DPHM system.

4

Long Short-Term Memory (LSTM) Network

4.1 Introduction to LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that addresses the limitations of traditional RNNs in capturing long-term dependencies in sequential data. Unlike standard RNNs, LSTM networks have the ability to remember information for extended periods, making them suitable for tasks involving sequences with long-range dependencies. They were introduced by Hochreiter and Schmidhuber (1997) [32], and were refined and popularized by many people in following work.

LSTM networks are designed to overcome the vanishing gradient problem, which hampers the training of RNNs by causing the gradients to diminish exponentially over time. By utilizing a specialized memory cell and gating mechanisms, LSTM effectively retains and propagates relevant information through the sequences, allowing it to capture dependencies over longer time horizons.

The importance of LSTM stems from its wide range of applications across various domains. In natural language processing, LSTM has achieved significant success in tasks such as machine translation, sentiment analysis, and speech recognition. The ability of LSTM to model and understand the context and semantics of language has made it a powerful tool in processing

sequential data.

Additionally, LSTM has demonstrated remarkable performance in time series prediction and forecasting. Its ability to capture long-term dependencies enables accurate predictions in domains such as financial market forecasting, weather prediction, and energy load forecasting.

Furthermore, LSTM networks have found applications in image and video processing tasks. They excel at tasks such as object detection, image captioning, video classification, and video summarization. By effectively modeling temporal relationships within image and video sequences, LSTM enhances the understanding and analysis of dynamic visual data.

The versatility and effectiveness of LSTM make it a valuable tool in many domains, including healthcare, finance, robotics, and more. Its ability to handle variable-length input sequences and capture long-term dependencies makes it particularly well-suited for tasks where context and temporal dynamics play a crucial role.

In the subsequent sections, the architecture and functionality of LSTM will be explored, providing a comprehensive understanding of its strengths, weaknesses, and applications. The focus will be on adapting and utilizing LSTM in the specific context of the space domain, taking into account the unique challenges and opportunities that arise in this field.

4.2 LSTM Architecture

LSTM networks are composed of a distinct architecture that enables them to capture and propagate information over long sequences effectively. In this section, the focus will be on exploring the structure and components of an LSTM network, emphasizing the key elements that contribute to its distinctive functionality.

LSTM, like other RNNs, is composed of repeated modules within its network architecture. This structure can be visualized in Figure 4.1, where x_t represents the input and h_t represents the output for the module at time t. What sets LSTM apart from traditional RNNs is the distinctive arrangement and configuration of these cells.

Therefore, the core of an LSTM network is this LSTM cell, which consists of several interconnected components. The LSTM cell is designed to retain and control the flow of information over time, allowing the network to selectively remember or forget information as needed.

The fundamental element of an LSTM cell is the memory cell, also referred to as the *cell state*, the horizontal line running through the top of the diagram. The Figure 4.2 clearly illustrate the concept. The memory cell serves as the ”long-term memory” of the LSTM network.

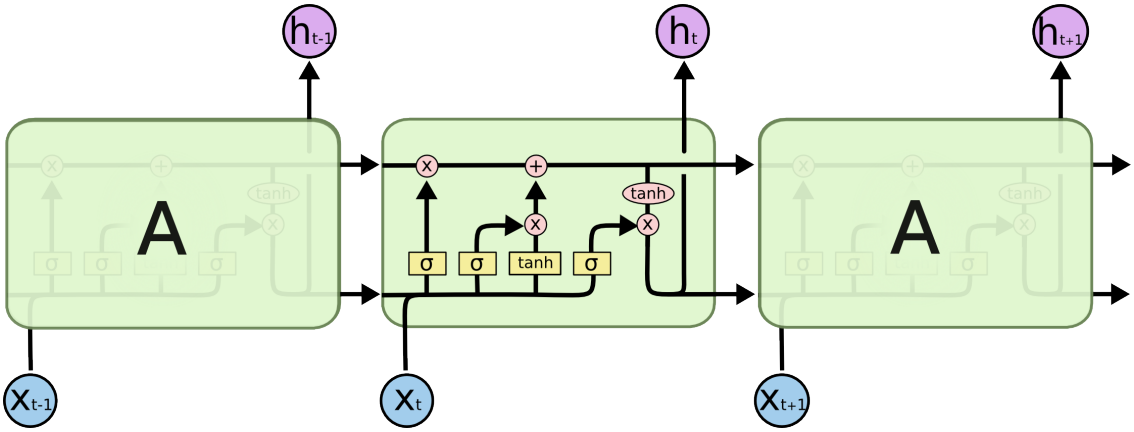


Figure 4.1: LSTM Chain

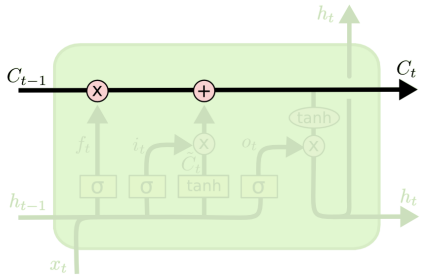


Figure 4.2: LSTM Cell State

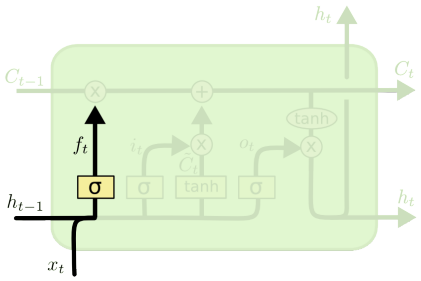
It preserves information from past time steps and carries it forward, allowing the network to capture dependencies over extended sequences.

To control the flow of information into and out of the memory cell, LSTM employs three types of gates: the *input gate*, *forget gate*, and *output gate*. These gates are composed of sigmoid neural network layers and have weights that are adaptively adjusted during the training process.

4.2.1 Step-by-Step LSTM Walk Through

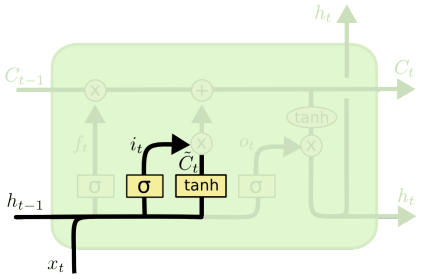
First, the *forget gate* (Figure 4.3) determines what information is retained or discarded from the memory cell. It regulates the flow of information from the previous time step by controlling the amount of information that is forgotten. The *forget gate* considers the current input and the previous hidden state to decide which information is no longer relevant. It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this".

Second, the *input gate* (Figure 4.4) determines how much new information is incorporated



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 4.3: LSTM Forget Gate

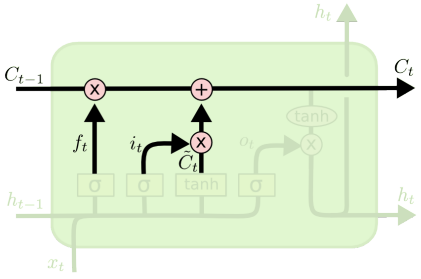


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 4.4: LSTM Input Gate - Part 1

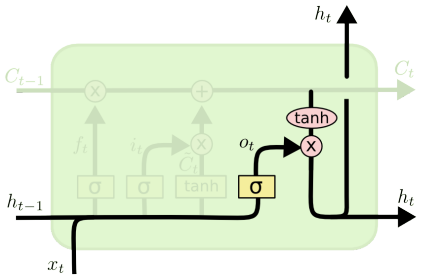
into the memory cell from the current time step. It selectively updates the memory cell based on the relevance and significance of the new input. The *input gate* takes into account the current input and the previous hidden state to make this decision. In details, the tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.

Next, it's now time to update the old cell state, C_{t-1}, into the new cell state C_t (Figure 4.5). The previous steps have determined the necessary actions, and now it is time to implement them. The old state is multiplied by f_t, disregarding the previously discarded elements. Subsequently, $i_t * \tilde{C}_t$ is added. These represent the updated candidate values, adjusted based on the designated update proportions for each state value.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 4.5: LSTM Input Gate - Part 2



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 4.6: LSTM Output Gate

Finally, the *output gate* controls the flow of information from the memory cell to the current time step's output. It selectively exposes the memory cell's contents, considering the current input and the previous hidden state. The *output gate* allows the LSTM network to determine which information is relevant and should be passed on as the output. The cell state is passed through a tanh function to restrict the values within the range of -1 to 1. Following this, it is multiplied by the output of the sigmoid gate, enabling us to selectively generate the desired information (see Figure 4.6).

Flow information within an LSTM cell

1. At each time step, the *input gate* decides how much new information should be stored in the memory cell.

2. The *forget gate* determines which information from the previous time step should be discarded from the memory cell.

3. The memory cell updates its contents based on the combined input from the *input gate* and the *forget gate*.

4. The *output gate* determines which information from the memory cell should be exposed as the output for the current time step.

5. The output of the LSTM cell, along with the updated hidden state, is passed on to the next time step or used for further processing.

By leveraging the memory cell and the gating mechanisms, LSTM networks can effectively capture and propagate information over extended sequences, enabling them to model long-term dependencies and address the vanishing gradient problem.

4.3 Functionality and Advantages of LSTM

LSTM networks possess several key functionalities and advantages that make them highly effective in modeling sequential data. This section focuses on discussing the remarkable capabilities of LSTM networks and highlighting their advantages over other recurrent neural network architectures.

One of the primary strengths of LSTM networks is their ability to capture long-term dependencies in sequential data. Traditional recurrent neural networks (RNNs) struggle with this task due to the vanishing gradient problem, where gradients diminish exponentially over time, making it challenging to propagate information across distant time steps. LSTM networks address this issue by utilizing the memory cell and gating mechanisms, allowing them to retain and propagate information over extended sequences. This enables LSTM networks to capture and leverage long-term dependencies, making them particularly suitable for tasks involving time-series data or natural language processing (NLP).

The vanishing gradient problem is mitigated in LSTM networks through the use of the forget gate and the input gate. The forget gate allows the network to selectively discard irrelevant information from the memory cell, preventing it from being overwhelmed by irrelevant or outdated data. On the other hand, the input gate enables the network to selectively update the memory cell with new and relevant information. By adaptively adjusting the weights of these gates during the training process, LSTM networks can effectively manage the flow of gradients and alleviate the vanishing gradient problem, enabling them to capture meaningful long-term dependencies.

Another notable advantage of LSTM networks is their capability to handle variable-length input sequences. Unlike traditional feed-forward neural networks that require fixed-size inputs, LSTM networks can process sequences of varying lengths. This flexibility makes LSTM networks well-suited for tasks involving sequential data with irregular or unpredictable lengths, such as speech recognition, sentiment analysis, or time-series forecasting. The ability to handle variable-length sequences allows LSTM networks to capture dependencies within different segments of the input, making them robust and adaptable to various real-world applications.

Additionally, LSTM networks offer several advantages over other recurrent neural network architectures. These advantages include:

- Improved memory retention: The memory cell in LSTM networks enables them to store and access information over long sequences, making them particularly effective in modeling temporal dynamics and capturing subtle patterns in sequential data.

- Enhanced training efficiency: By mitigating the vanishing gradient problem, LSTM networks facilitate more efficient training, leading to faster convergence and improved learning outcomes.
- Increased expressiveness: The gating mechanisms of LSTM networks provide additional control and flexibility in modeling complex relationships within sequential data, allowing them to capture intricate dependencies and exhibit a higher level of expressiveness.
- Adaptability to diverse domains: LSTM networks have demonstrated success in various domains, including natural language processing, speech recognition, time-series analysis, and robotics, showcasing their versatility and applicability in diverse problem domains.

In summary, LSTM networks excel in capturing long-term dependencies, mitigating the vanishing gradient problem, and handling variable-length input sequences. Their functionality and advantages make them a powerful tool for modeling sequential data and addressing challenges associated with temporal dependencies.

4.4 Training and Learning in LSTM

Training LSTM networks involves optimizing their parameters to learn and capture meaningful patterns in sequential data. This section presents an overview of the training process in LSTM networks, highlighting key concepts such as backpropagation through time (BPTT), gradient descent, and the significance of forget gates and input gates in the learning process.

The training process in LSTM networks is based on the principles of supervised learning, where the network learns from a labeled dataset to make predictions or generate sequences. The labeled dataset consists of input sequences and their corresponding target outputs. During training, the LSTM network is exposed to the input sequences one timestep at a time, and its predictions are compared to the corresponding target outputs. The discrepancy between the predicted outputs and the targets is quantified using a loss function, such as mean squared error or cross-entropy loss.

To optimize the LSTM network's parameters, the backpropagation through time (BPTT) algorithm is employed. BPTT extends the traditional backpropagation algorithm to recurrent neural networks by unfolding the network over time and computing gradients at each timestep. This allows the gradients to flow backward through time, enabling the network to learn from the entire sequence.

Gradient descent is then used to update the LSTM network's parameters based on the computed gradients. The gradients indicate the direction and magnitude of parameter adjustments

required to minimize the loss function. By iteratively updating the parameters in the opposite direction of the gradients, the network gradually converges towards an optimal set of parameters that minimize the overall prediction error.

The forget gates and input gates play crucial roles in the learning process of LSTM networks. During training, these gates are adjusted to control the flow of information within the network. The forget gate determines which information should be discarded from the memory cell, while the input gate decides which new information should be added. By adapting the weights of these gates during training, the LSTM network can selectively retain and update information based on its relevance and importance for accurate predictions.

The forget gate and input gate are updated using the gradients computed during BPTT. The gradients provide information about the contribution of each gate to the overall loss function. By adjusting the gate weights based on these gradients, the LSTM network can learn to effectively control the flow of information and enhance its predictive capabilities. This mechanism allows the network to focus on relevant features and suppress irrelevant or noisy information during training.

Through the iterative training process, LSTM networks learn to capture complex temporal dependencies and make accurate predictions on sequential data. The combination of BPTT, gradient descent, and the adaptability of forget gates and input gates enables LSTM networks to effectively learn from sequential data and optimize their performance over time.

4.5 Applications of LSTM in Various Domains

The versatility and effectiveness of LSTM networks have led to their successful application in various domains. In this section, notable applications of LSTM will be reviewed in the domains of natural language processing, time series prediction and forecasting, and image and video processing tasks.

LSTM networks have demonstrated remarkable success in natural language processing (NLP) tasks, such as language modeling, machine translation, sentiment analysis, and text generation. Due to their ability to capture long-term dependencies, LSTM networks excel in understanding and generating coherent and contextually relevant sequences of words. They have proven particularly effective in tasks where the context of previous words or phrases is crucial for accurate prediction and generation.

In time series prediction and forecasting, LSTM networks have shown significant improvements over traditional methods. By considering the temporal dependencies in the data, LSTM

models can capture complex patterns and make accurate predictions. This makes them well-suited for tasks such as stock market prediction, weather forecasting, energy load forecasting, and anomaly detection in time series data. The ability of LSTM networks to handle variable-length input sequences makes them particularly valuable in scenarios where the length of the historical data varies.

LSTM networks have also found applications in image and video processing tasks. In image recognition and classification, LSTM networks can be combined with Convolutional Neural Networks (CNNs) to process sequential image data effectively. This enables the analysis of videos, action recognition, and generating textual descriptions of visual content. Additionally, LSTM networks have been used in video analysis tasks, including video captioning, video summarization, and video prediction. The sequential nature of video data makes LSTM networks valuable for capturing temporal dependencies and modeling motion dynamics.

The applications of LSTM networks extend beyond these domains, with successful utilization in speech recognition, music generation, medical diagnosis, and more. Their ability to capture long-term dependencies, handle variable-length sequences, and mitigate the vanishing gradient problem makes them a powerful tool in many complex and sequential data analysis tasks.

However, it is worth noting that while LSTM networks have achieved impressive results in these applications, they are not without limitations. LSTM models can be computationally expensive, especially for large-scale datasets, which may pose challenges in real-time applications or resource-constrained environments. Additionally, training LSTM networks requires a substantial amount of labeled data to effectively capture the underlying patterns in the data.

In the following section, the considerations and contrasts of employing LSTM networks in the space domain will be discussed. This domain involves the analysis and prediction of sequential data, which holds significant importance in ensuring the success of satellite operations and missions.

4.6 LSTM in the Space Domain

The application of LSTM networks in the space domain brings both challenges and opportunities. This section analyzes the specific considerations of utilizing LSTM in space-related applications, examines the adaptation of LSTM to accommodate space-specific data and constraints, and explores the potential enhancements that LSTM can offer in satellite operations, data analysis, and anomaly detection within the space domain.

Space-related applications often involve dealing with complex and dynamic datasets, such as telemetry data, sensor readings, satellite imagery, and orbital information. LSTM networks offer a promising approach to address the challenges associated with analyzing and predicting sequential space data. By leveraging the ability to capture long-term dependencies, LSTM models can effectively handle the temporal characteristics of space data, allowing for improved understanding and prediction of various space phenomena.

One significant challenge in the space domain is the limited availability of labeled data, particularly for rare events or anomalies. LSTM networks require an adequate amount of labeled data to learn and generalize patterns effectively. However, anomalies or critical events in space missions are relatively rare, making it challenging to collect sufficient labeled examples. Therefore, careful data collection, preprocessing, and augmentation techniques are necessary to maximize the potential of LSTM models in space anomaly detection and prediction tasks.

Another crucial aspect is the adaptation of LSTM to handle space-specific data and constraints. For example, space telemetry data often exhibits irregular sampling rates, missing values, and noisy measurements. LSTM networks can be modified to accommodate irregular time intervals and handle missing data effectively, ensuring robust performance even in the presence of data irregularities. Additionally, incorporating domain knowledge into the LSTM architecture, such as including physical constraints or contextual information, can further enhance the model's performance in space-related applications.

LSTM networks have the potential to significantly enhance satellite operations in various ways. By leveraging the sequential nature of space data, LSTM models can assist in satellite orbit prediction, aiding in precise maneuver planning and optimizing satellite operations. Moreover, LSTM networks can improve the analysis of satellite sensor data, enabling real-time anomaly detection, fault diagnosis, and health monitoring of satellite subsystems. Early detection of anomalies or deviations from expected behavior can facilitate proactive maintenance and enhance the overall reliability and longevity of space systems.

Furthermore, LSTM networks can be utilized in data analysis tasks to extract meaningful patterns and trends from large volumes of satellite imagery and remote sensing data. By incorporating LSTM with convolutional neural networks (CNNs), complex spatio-temporal patterns can be captured, enabling improved land cover classification, change detection, and environmental monitoring.

In summary, LSTM networks offer significant potential in the space domain, enabling enhanced satellite operations, data analysis, and anomaly detection. By addressing the challenges associated with space-specific data and constraints, LSTM models can provide valuable insights

into space phenomena and contribute to the success of space missions. However, it is crucial to carefully design and adapt LSTM architectures to suit the specific requirements and characteristics of space-related applications.

4.6.1 Examples of LSTM in Space Domain

In this subsection, notable examples are explored to demonstrate the successful utilization of LSTM networks within the field of space exploration and related applications. These examples shed light on the remarkable effectiveness of LSTM technology, particularly in the field of fault diagnosis and prognosis, with a special emphasis on the AOCS. By focusing on the diagnosis and prediction of faults in crucial components like reaction wheels and gyroscopes, these examples provide insights into the practical applications of LSTM and its potential to revolutionize decision making and operational efficiency in space missions.

Example 1 | A Reliable Deep Learning Approach for Time-Varying Identification: Spacecraft Reaction Wheels Case Study [29]

This study presents a fast and accurate end-to-end architecture for detecting and identifying anomalies occurring in spacecraft reaction wheels. The proposed approach utilizes a combination of One-Dimensional Convolutional Neural Network (1D-CNN) and LSTM network architecture. The 1D-CNN is employed to extract informative features from the raw residual signals, while the LSTM layer effectively handles the time series data and learns long-term dependencies. Experimental results demonstrate the reliability and robustness of the proposed algorithm, offering a compact system architecture for anomaly detection and identification in spacecraft reaction wheels. The use of convolutional neural networks (CNNs) reduces the number of trainable parameters through parameter sharing and local connectivity. CNNs have been widely employed in pattern recognition applications, primarily due to their optimal configuration and ability to extract discriminative features from raw signals. Additionally, the integration of LSTM overcomes the vanishing gradient problem and enables the analysis of patterns in long sequences of data. By combining the 1D-CNN and LSTM networks in a single architecture for reaction wheel fault diagnosis, the contributions of this work [29] can be summarized as follows:

- The proposed deep learning architecture offers a superior approach for spacecraft reaction wheel fault diagnosis, seamlessly integrating the feature-extraction and anomaly

classification phases into a single process, eliminating the need for time-consuming feature extraction methods.

- The 1D-CNN with LSTM fault diagnosis structure effectively captures long-term dependencies from the reaction wheel residual time-series signals, enabling the diagnosis of time-varying faults with a simple architecture and superior performance.
- The compact nature of the proposed deep learning architecture, with a small memory footprint, makes it highly suitable for safety-critical real-time applications, both in general and specifically for onboard computer fault diagnosis tasks in spacecraft.
- By directly learning discriminative features from the raw signals, the proposed fault diagnosis approach can be adapted for other spacecraft subsystems and different industrial systems as well.

The considered faults include bus voltage (V_{bus}) faults, faults caused by variations in motor current (I_m), and faults due to an increase in bearing friction. These faults commonly occur due to the internal non-linearity and complex design of reaction wheels. Figure 4.7 illustrates the proposed Fault Detection and Identification (FDI) scheme for spacecraft reaction wheels, consisting of two phases:

1. Residual generation phase: The residual signal is generated by calculating the difference between the measured torque and the estimated torque.
2. CNN-LSTM phase: The residual time-series signals are processed to extract distinctive features representing different reaction wheel states.

Tthe model architecture is depicted in Figure 4.8, comprising four main layers: the convolutional layer, the pooling layer, the LSTM layer, and the fully connected layer.

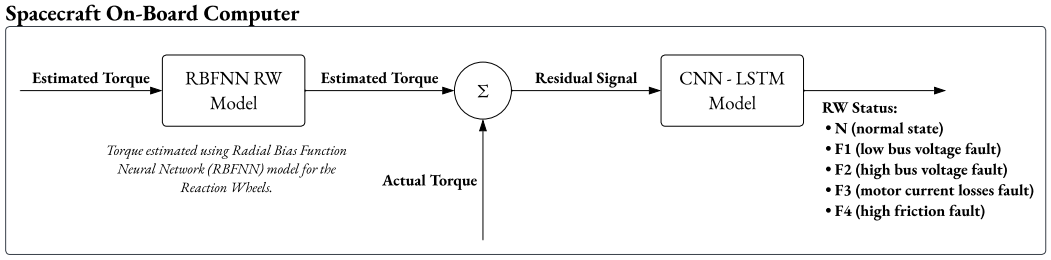


Figure 4.7: FDI Scheme for the Spacecraft RW (Example 1)

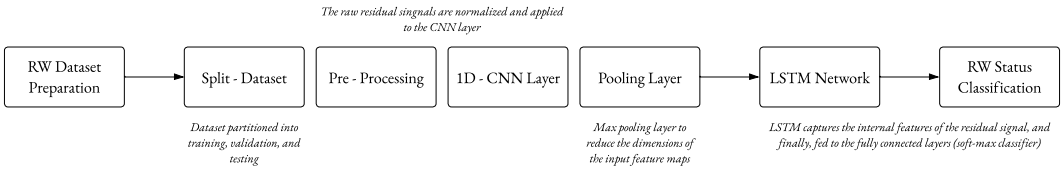


Figure 4.8: Architecture for RW Fault Diagnosis (Example 1)

Example 2 | Fault Prognosis of Satellite Reaction Wheels Using A Two-Step LSTM Network [24]

Reaction wheels are extensively used actuators in the AOCS, but they are prone to premature failure. Failure in reaction wheels often arises from technical issues such as inadequate bearing lubrication and uneven distribution of frictional torque, resulting in motor torque variations. However, measuring motor torque directly from onboard sensors is not feasible. To address this challenge, the study [24] proposes the utilization of a LSTM network to estimate motor torque using available measurable data.

The main objective of this work is to predict the future performance degradation of a reaction wheel, ultimately determining its Remaining Useful Life (RUL) and ensuring smooth system performance. To achieve this, a framework is developed to forecast future system states and parameters based on historical and available sensor data.

The proposed prognostic approach consists of three steps, as illustrated in Figure 4.9.

1. Employing an LSTM network to forecast the system state, which includes Reaction Wheels speed (w_m) and motor current (I_m), based on available and historical measurements.

2. Utilizing the forecasted state data and knowledge of the correlation between system state and system parameters, it is possible to predict future values of system parameters (bus voltage V_{bus}, and motor torque K_t) that are directly linked to the health of the system components. These predicted states are fed into a multivariate LSTM network to predict critical system parameters.

3. Using the results obtained from the first two steps, the RUL of the system components can be determined by applying a thresholding technique to the operating ranges of critical componentnts, thereby generating a system Health Index.

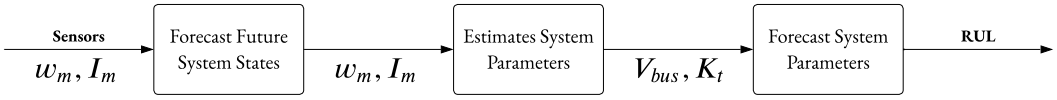


Figure 4.9: Stages of Data-Driven Fault Prognosis (Example 2)

Example 3 | Fault Diagnosis and Prognosis of Aerospace System Using Growing Recurrent Neural Network and LSTM [25]

This work introduces an AI approach utilizing two innovative neural network models, the Growing Neural Network (GNN) and variable sequence LSTM (VarLSTM), for automating the DPHM process in aerospace systems. The proposed model estimates a Health Index (HI) value by comparing measured telemetry data with predictions generated using the GNN algorithm. Subsequently, the HI value is extrapolated for prognostic purposes. For datasets with multiple units, the model directly maps the RUL of training units to their corresponding measured features at each instant to make RUL predictions. The work [25] optimizes the architecture of a recurrent neural network and evaluates its performance in making RUL predictions for aircraft engines and detecting failures in satellite attitude actuators, specifically Reaction Wheels. The model's effectiveness is tested on the CMAPSS and PHM08 aircraft engine datasets (multiple-unit datasets) simulated by NASA, as well as on single-unit datasets from the Kepler spacecraft's reaction wheels.

While the true RUL of a system generally exhibits a linear relationship with time, the system's degradation is typically nonlinear, initially operating nominally and then gradually deteriorating until failure. To ensure accuracy, the learning process employs piece-wise RUL instead of the true RUL. The piece-wise RUL remains constant until signs of degradation appear, at which point the RUL starts decreasing linearly.

The available dataset for the Kepler Mission includes features such as Speed, Torque Friction, Torque Command, Temperature, and Attitude Errors in X, Y, and Z. Test residuals are calculated by comparing the predictions with the actual speed values. The mean (μ^*), standard deviation (σ^{2*}), and 90th percentile (δ^*) values are then computed for the test residual data. Utilizing the GNN model, predictions are generated, and prediction residuals are calculated. Similarly, the mean (μ), standard deviation (σ^2), and 90th percentile (δ) values are derived from the prediction residuals data. Figure 4.10 illustrates the framework of the model.

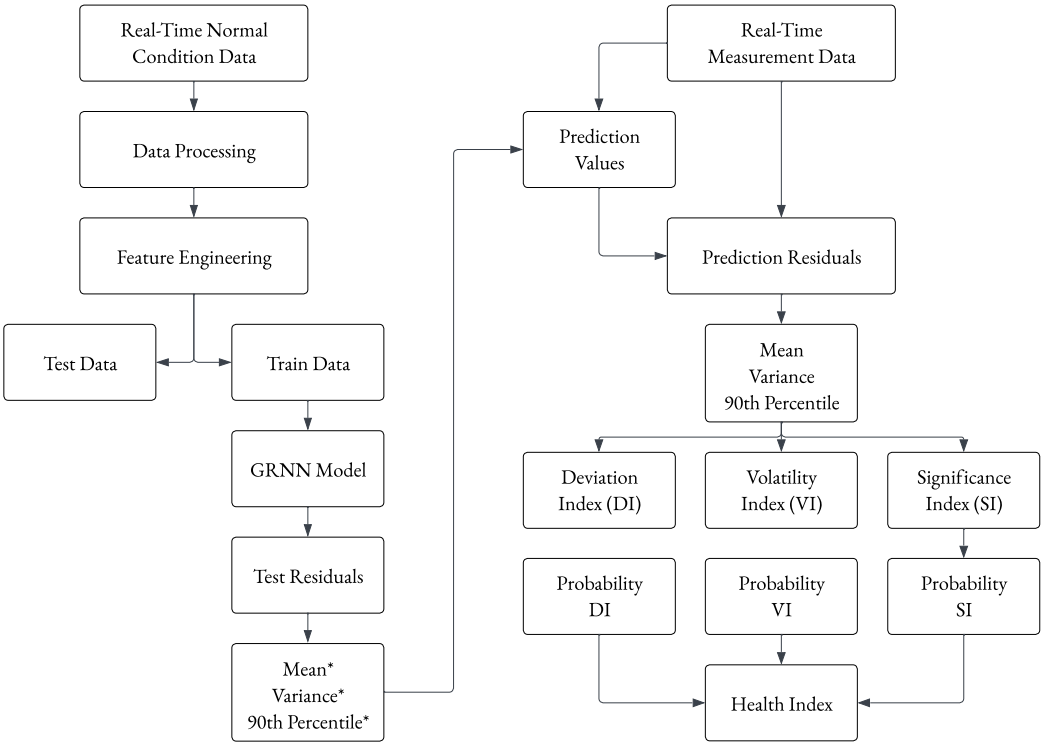


Figure 4.10: Training and Fault Detection Workflow: Flowchart of the Model (Example 3)

The Health Index is a performance index that ranges in [0,1] and it indicates the probability of failure. In this specific case it is the product of three indices: Deviation Index (DI), Volatility Index (VI) and Significance Index (SI).

Instead, the RUL of the reaction wheels is estimated by first determining a degradation model for the HI, from which the HI can be extrapolated to predict when it will exceed the failure threshold. The time from which the extrapolation starts until the HI exceeds the threshold will be the predicted RUL.

Example 4 | Satellite Telemetry Data Anomaly Detection Using Bi-LSTM Prediction Based Model [21]

In work [21], a novel approach for anomaly detection in telemetry time series data is proposed, utilizing a bi-directional Long Short-Term Memory Neural Network (Bi-LSTM). The method leverages the strong temporal feature extraction capabilities of Bi-LSTM to model and regress satellite data, enabling point data anomaly detection by comparing predicted values with real

values. To enhance the suitability of the prediction-based model, a dynamic threshold optimization method is incorporated into the framework (Figure 4.11). Experimental results on three satellite telemetry datasets demonstrate the effectiveness of the proposed method, with performance outperforming RNN and basic LSTM models.

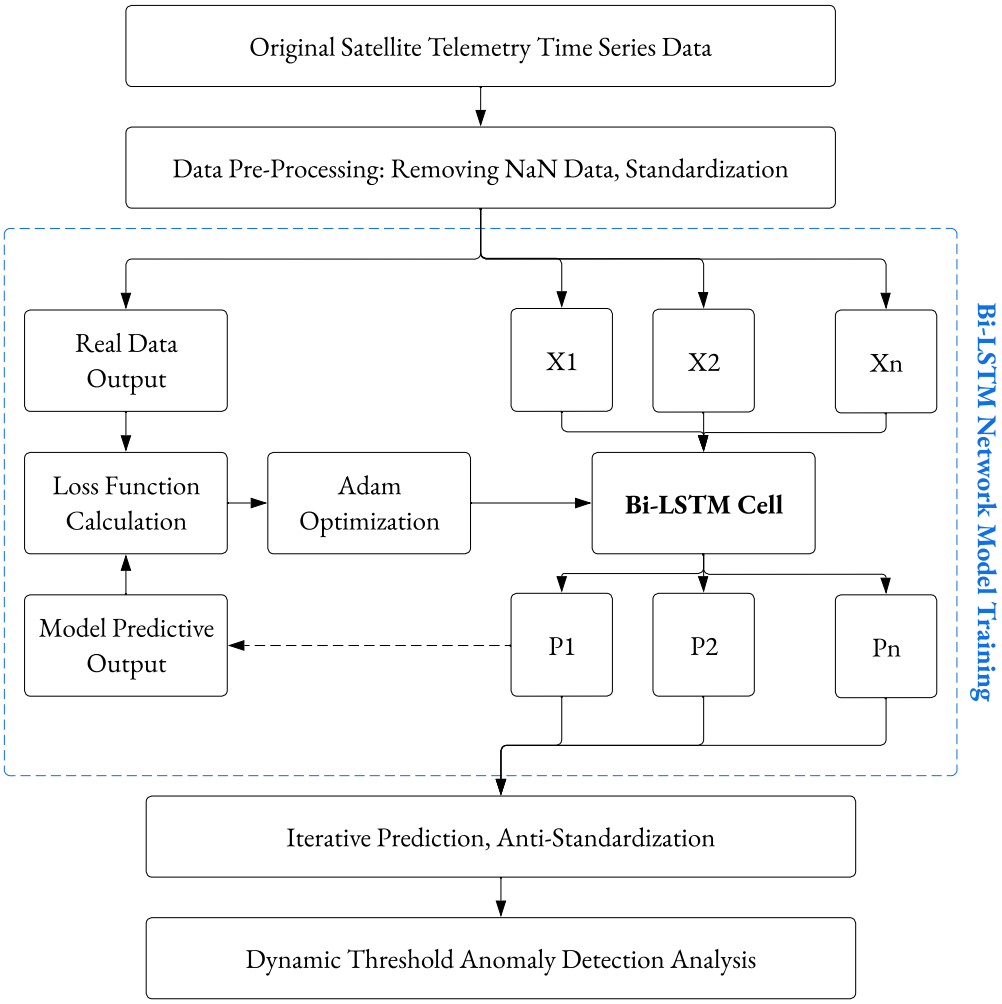


Figure 4.11: Framework of Bi-LSTM Network Model (Example 4)

Traditional LSTM models only consider sequence features in the forward time series order and overlook the impact of previous time data on the current data. However, in the anomaly detection of satellite telemetry data, adjacent telemetry data within the sequence can significantly

influence each other, serving as crucial indicators for evaluating the current data. Bi-LSTM addresses this limitation by processing data in both forward and backward timing directions, capturing the long-term memory of time series data, and providing past and future context for each moment. Consequently, Bi-LSTM effectively captures data characteristics and achieves more accurate anomaly detection results. Despite the high predictive capabilities of Bi-LSTM and its support by powerful devices, its application in anomaly detection has been limited. To validate the performance of the proposed method, experiments were conducted using Soil Moisture Active and Passive (SMAP) satellite telemetry data.

The real satellite telemetry data provided in the official NASA database was utilized for algorithm verification, including detector temperature in SMAP, angular velocity of SMAP windsurfing board, and active power of SMAP power system.

The Bi-LSTM algorithm swiftly adjusts the relationship between the current time data and the data learned at previous and subsequent times, effectively capturing contextual information and simplifying complex problems. The experimental results demonstrate that the proposed Bi-LSTM model improves monitoring efficiency and provides effective decision support for anomaly detection and adjustment tasks.

Example 5 | Satellite Telemetry Data Anomaly Detection Using Causal Network and Feature-Attention-Based LSTM [28]

Traditional data-driven methods for satellite telemetry data anomaly detection often suffer from high false positive rates (FPR) and lack interpretability. To address these challenges, the work [28] proposes an anomaly detection framework called Causal Network and Feature Attention based Long Short-Term Memory (CN-FA-LSTM). The method involves constructing a causal network of telemetry parameters using normalized modified conditional transfer entropy (NMCTE) and optimizing it through conditional independence tests based on conditional mutual information (CMI). A CN-FA-LSTM model is then established to predict telemetry data, and a non-parametric dynamic k-sigma threshold updating method is proposed for setting thresholds. Furthermore, CN-FA-LSTM offers improved interpretability compared to other commonly used prediction models. The effectiveness and universality of the proposed method are verified through experiments on two public datasets.

Causality captures the cause-and-effect relationship between factors, reflecting the irreversible relationship at the physical mechanism level. On the other hand, correlation only describes a statistical relationship, indicating synchronization or similarity between factors. Therefore, causality better captures the internal correlation and mechanism of a system compared to cor-

relation.

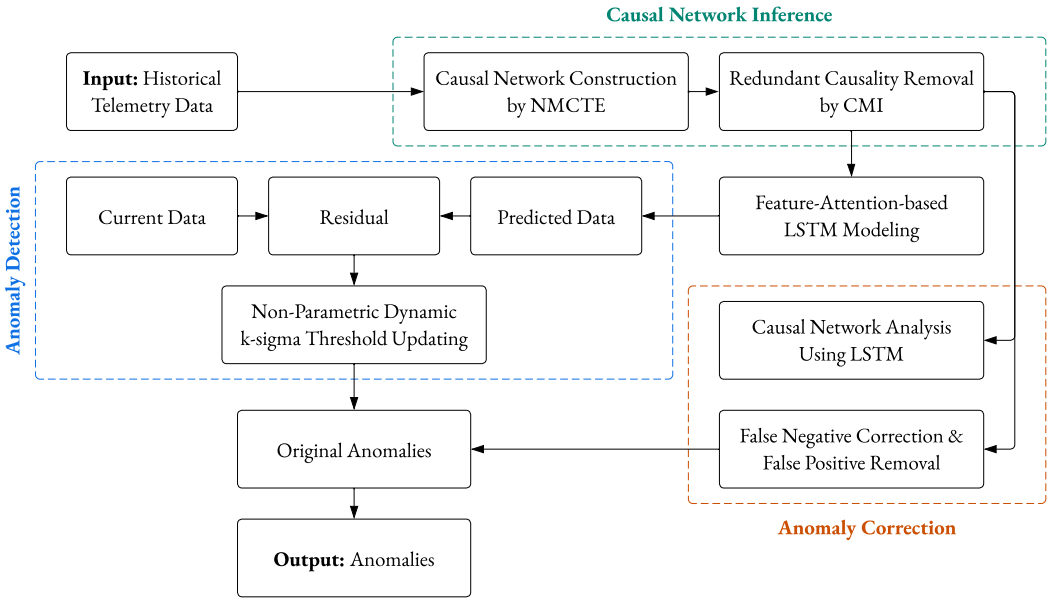


Figure 4.12: Framework of Satellite Telemetry Data Anomaly Detection Using CN-FA-LASTM (Example 5)

Therefore, to improve the interpretability and accuracy of data-driven anomaly detection by discovering causal relationships between telemetry parameters, this new framework called CN-FA-LSTM was introduced. Figure 4.12 illustrates the framework, and the work [28] presents the following key contributions:

- Utilizing normalized modified conditional transfer entropy (NMCTE) as a metric to identify and measure the strength of causality in time series data, constructing the causal network.

- Introducing a conditional independence test method based on conditional mutual information (CMI-CIT) to optimize the causal network by removing redundant causality.

- Developing CN-FA-LSTM, which utilizes the causal network, source parameters, and target parameter (parameter to be predicted) as inputs, and models and predicts the telemetry data using a multivariate LSTM with feature attention. CN-FA-LSTM effectively leverages the causality between telemetry parameters, leading to competitive performance in anomaly detection.

- Introducing a nonparametric dynamic k-sigma threshold updating method that continuously updates the tradeoff coefficient. This method enables more reasonable threshold setting in unsupervised anomaly detection.
- Employing an interpretative structure model (ISM) to analyze the causal network and correct anomalies, effectively reducing false negatives and false positives (FPs).

To validate the universal applicability of the method, experiments were conducted on the SMAP and MLS datasets published by NASA. The experimental results on these datasets demonstrate that the precision, F1-score, and FPR of the proposed method surpass those of other state-of-the-art methods. Moreover, the time complexity of the method is also satisfactory.

This method enables the study of causality in multivariate and large-scale telemetry data, improves anomaly sensitivity in model predictions, and provides more reasonable threshold settings. Additionally, it offers valuable insights for satellite operation and maintenance personnel to explore the internal mechanisms of satellites.

4.7 Limitations and Considerations

While LSTM networks offer several advantages and have proven effective in various domains, it is important to acknowledge their limitations and consider potential drawbacks when utilizing them in practical applications. This section aims to highlight the limitations of LSTM networks, explore computational requirements and memory constraints, and examine alternative architectures and potential enhancements to LSTM.

One limitation of LSTM networks is their vulnerability to overfitting, especially when dealing with limited training data. Overfitting occurs when the model becomes too complex and starts to memorize the training examples instead of learning general patterns. Regularization techniques such as dropout and weight decay can be employed to mitigate overfitting and improve the generalization capability of LSTM networks.

Another consideration is the computational requirements of LSTM networks, particularly when dealing with large-scale datasets or complex models. Training LSTM networks can be computationally expensive and time-consuming, requiring significant computational resources. Additionally, the memory requirements of LSTM networks can pose challenges, especially when working with long sequences or high-dimensional data. Efficient memory management techniques and parallel computing architectures can be explored to mitigate these challenges and improve the scalability of LSTM models.

Furthermore, while LSTM networks excel in capturing long-term dependencies, they may face difficulties in capturing certain types of patterns, such as hierarchical structures or explicit symbolic rules. Alternative architectures, such as Transformer networks, have emerged as promising alternatives to LSTM in natural language processing tasks by utilizing attention mechanisms and self-attention layers. These architectures can offer improved performance in capturing complex dependencies and long-range interactions, which may be beneficial in specific space-related applications.

It is also important to consider the interpretability and explainability of LSTM networks. Deep neural networks, including LSTMs, are often considered black boxes due to their complex and non-linear nature. Understanding the decision-making process of LSTM models and providing explanations for their predictions can be challenging. Interpretability techniques, such as attention mechanisms and gradient-based methods, can be employed to shed light on the important features and patterns learned by LSTM networks, enhancing their transparency and trustworthiness in critical space applications.

Moreover, advancements in LSTM variants and improvements in network architectures continue to expand the capabilities of sequential modeling. Variants such as Gated Recurrent Units (GRUs) and bi-directional LSTMs offer alternative options for capturing temporal dependencies and have shown promising results in various applications. Additionally, incorporating external knowledge, domain-specific constraints, or hybrid models that combine LSTM with other ML techniques can further enhance the performance and address the limitations of LSTM networks.

In conclusion, while LSTM networks offer valuable capabilities in capturing long-term dependencies and handling sequential data, it is important to be aware of their limitations and consider the specific requirements of space-related applications. Mitigating overfitting, managing computational and memory constraints, exploring alternative architectures, and improving interpretability are key considerations when utilizing LSTM networks. By addressing these limitations and leveraging advancements in the field, LSTM networks can continue to play a vital role in space applications and contribute to advancements in satellite operations, data analysis, and anomaly detection.

In the final chapter, the discussion will focus on practical implementation guidelines, offering recommendations for model optimization and deployment. Furthermore, potential future directions in the utilization of LSTM networks in the space domain will be highlighted.

4.8 Summary of Findings

In this chapter, a comprehensive exploration of LSTM networks has been conducted, encompassing their architecture, functionality, advantages, training process, and applications in various domains. Additionally, specific adaptations and considerations required for LSTM utilization in the space domain were examined. To conclude this chapter, the key points discussed will be summarized, strengths and weaknesses of LSTM in the context of space applications will be reflected upon, and final thoughts on future prospects and potential advancements in LSTM research will be provided.

Throughout this chapter, a thorough understanding of LSTM networks has been obtained. The introduction highlighted the significance of LSTM and its wide-ranging applications in natural language processing, time series prediction, and image and video processing. The LSTM architecture was then explored, elucidating its fundamental components such as the memory cell, input gate, forget gate, and output gate. The capacity of LSTM to capture and retain long-term dependencies in sequential data was illustrated through the information flow within an LSTM cell.

The functionality and advantages of LSTM were discussed, emphasizing its ability to handle long-term dependencies, address the vanishing gradient problem, and process input sequences of varying lengths. Remarkable performance and state-of-the-art results achieved by LSTM in domains like natural language processing, time series prediction, and image and video processing were explored.

The training and learning process of LSTM networks were explained, with a focus on back-propagation through time (BPTT) and gradient descent. The significance of forget gates and input gates in the learning process, enabling effective adaptation to sequential patterns by LSTM networks, was highlighted.

Shifting attention to the applications of LSTM in the space domain, the challenges and opportunities associated with its utilization were discussed. The enhancement of satellite operations, data analysis, and anomaly detection in space through LSTM were explored, considering the unique constraints and characteristics of space-related data.

It is crucial to acknowledge the limitations and considerations inherent in LSTM networks. Potential limitations, including overfitting, computational requirements, and memory constraints, were identified. Alternative architectures and improvements to LSTM, such as Transformer networks, interpretability techniques, and LSTM variants like GRUs and Bidirectional LSTMs, were also discussed.

In conclusion, LSTM networks offer valuable capabilities in capturing long-term dependencies and handling sequential data, making them well-suited for various applications in the space domain. Their ability to process and learn from complex and dynamic data can significantly enhance satellite operations, data analysis, and anomaly detection in space.

However, it is crucial to address the limitations and consider the specific requirements of space applications when utilizing LSTM networks. This involves mitigating overfitting, managing computational and memory constraints, exploring alternative architectures, and improving interpretability. By doing so, LSTM networks can continue to advance space-related research and contribute to the optimization of satellite missions, data processing, and decision-making.

Looking ahead, the future prospects for LSTM research are promising. Further advancements in network architectures, optimization techniques, and interpretability methods will continue to enhance the capabilities of LSTM networks. Additionally, the integration of LSTM with other ML techniques and the incorporation of domain-specific knowledge can unlock new possibilities and enable more sophisticated applications in the space domain.

As the exploration of LSTM networks progresses, it becomes imperative to further investigate their potential in space-related tasks, expanding the scope of their application. By harnessing the capabilities of LSTM networks and addressing their limitations, new opportunities can be unlocked, leading to enhanced operational efficiency in space and facilitating groundbreaking discoveries.

In the concluding chapter, practical implementation guidelines will be presented, along with recommendations for optimizing and deploying the models. Additionally, potential future directions for LSTM networks in the space domain will be discussed, paving the way for further advancements in this field.

5 Workflow and Results

5.1 MATLAB Overview

Implementing the network architecture presented in Chapter 4 and conducting the necessary experiments requires a suitable programming environment. In this study, MATLAB (MATrix LABoratory) was chosen as the preferred software package due to its powerful capabilities in scientific computing, data analysis, and algorithm development. MATLAB offers a comprehensive range of tools and functions that facilitate the implementation and evaluation of the network architecture used in the study. It provides an intuitive and interactive environment for exploring, analyzing and visualizing data, making it an ideal choice for developing complex neural network architectures like the hybrid CNN-LSTM.

The decision to utilize MATLAB for creating and training the network architecture was driven by several key factors:

- **Extensive Deep Learning Toolbox:** MATLAB provides a dedicated Deep Learning Toolbox that offers a wide range of pre-build functions, models, and layers for constructing, training, and evaluating neural networks. This toolbox simplifies the implementation process and allows for rapid prototyping of deep learning architectures.
- **Integration with Other Toolboxes:** MATLAB seamlessly integrates with various toolboxes for data preprocessing, signal processing, and statistical analysis. This integration facilitates efficient data manipulation and preprocessing steps required before training the hybrid network.

- **Visualization Capabilities:** MATLAB provides advanced visualization tools, including interactive plots, graphs, and heatmaps, which are invaluable for analyzing and interpreting the performance of the hybrid CNN-LSTM network. These visualization aid in identifying patterns, trends, and potential areas for improvement in the training process.

By utilizing the extensive capabilities and resources provided by MATLAB, the development and training of the hybrid CNN-LSTM network can be efficiently. The following sections will provide comprehensive insights into the dataset, data preprocessing techniques, network architecture, and the process of training and evaluating the network in this study.

In order to ensure the transparency and reproducibility of the research, accompanying MATLAB scripts will be provided for each section of this chapter. These scripts will showcase the implementation of the network architecture, including model configuration, training procedures, and evaluation metrics. The sharing of these scripts aims to facilitate a deeper comprehension of the methodology employed and enables readers to replicate the experiments conducted in this study.

There are several other popular alternatives for developing and training neural networks, as for instance:

- *TensorFlow:* an open-source deep learning framework developed by Google. It provides a flexible and efficient platform for building and training various types of neural networks. TensorFlow uses a data flow graph model, where nodes represent mathematical operations and edges represent data tensor. It offers a Python interface, along with support for other programming languages like C++, Java, and JavaScript.

- *PyTorch:* an open-source deep learning framework developed by Facebook's AI Research lab. It provides a dynamic computational graph approach, making it highly flexible for building and training neural networks. PyTorch offers a Python interface and is known for its simplicity, ease of use, and strong support for GPU acceleration.

- *Keras:* a high-level deep learning library written in Python. It is build on top of TensorFlow and provides a user-friendly interface for designing and training neural networks. Keras offers a wide range of predefined layers and models, making it accessible for beginners and efficient for rapid prototyping.

- *Caffe:* a deep learning framework developed by Berkeley AI Research (BAIR). It is widely used for computer vision tasks and provides a C++ and Python interface. Caffe's architecture emphasizes speed and efficiency, making it suitable for large-scale deep learning applications.

5.2 Dataset Description and Preprocessing

The dataset used for training and testing the DPHM system consists of real satellite data provided by S.A.T.E. for testing purposes. The dataset includes multiple .csv files containing various parameters from different subsystems. This study specifically focuses on five parameters from the EPDS subsystem: *current*_1, *curren*_2, *current*_3, *voltage*_1, and *voltage*_2. For detailed descriptions of these parameters, please refer to Table 5.1. These parameters have been chosen because the ground truth was available and therefore the health status to be detected was known a priori.

Parameter	**Description**
current_1	Measured output current (mA) of the EPDS component that distributes the power. Value referred to specific output channel.
current_2	Measured output current (mA) of the EPDS component that distributes the power. Value referred to specific output channel.
current_3	Measured output current (mA) of the EPDS component that distributes the power. Value referred to specific output channel.
voltage_1	Measured VBAT voltage (mV). Value referred to specific output channel.
voltage_2	Measured VBAT voltage (mV). Value referred to specific output channel.

Table 5.1: Satellite Parameters and Descriptions

The parameters are stored in multiple .csv files, and the process of loading them into MATLAB can now be explored. Each .csv file contains a timestamp column in Unix Time format, followed by various parameters in subsequent columns. To extract the desired parameter's time series data and necessary information for further processing, a MATLAB script has been developed (see Listing A.1). This script allows you to select a specific .csv file from a designated folder and choose the number of parameters you wish to select.

After selecting the file and number of parameters, the script parses the contents and displays the available telemetries along with their identification numbers in the command window. Simply enter the corresponding integer to load the desired telemetry data.

Once the data loading phase is complete, a preprocessing of the timeseries may be performed, by synchronizing and interpolating the timeseries value (following a sample-hold interpolation

method). This preprocessing function allow to pass multiple parameters as input to the network. In this work, only the results of the univariate approach is considered, therefore this preprocessing function was not exploited for the presented results and for this reason is not reported here. However, this preprocessing function has been implemented and tested and it can be used for a multivariate analysis.

5.2.1 Dataset Split: Train and Test Periods

To assess the performance of the DPHM system, it is essential to divide the dataset into distinct training and test periods. In this section, the criteria used for partitioning the dataset will be described, along with the rationale behind selecting three separate test periods. Furthermore, a MATLAB script in Listing A.2 demonstrates the process of dataset division for reference and clarity.

The data parameters were collected over a period spanning from June 1, 2019, to November 10, 2019. The training period was selected from June 1, 2019, to June 23, 2019, based on careful consideration: this period demonstrated no anomalies and exhibited a consistent nominal trend in the parameters.

To thoroughly assess the robustness of the DPHM system, three separate testing periods were established:

1. The first testing period: from June 24, 2019, to July 4, 2019.
2. The second testing period: from September 16, 2019, to October 9, 2019.
3. The third testing period: from October 1, 2019, to November 10, 2019.

These testing periods were specifically chosen to encompass instances where anomalies are expected to occur. By incorporating such scenarios, the system's ability to accurately detect and predict anomalous behavior can be assessed.

To ensure the accurate selection of the training period and the three test periods, two functions have been developed: *selection_of_training_data* and *selection_of_testing_data*. These functions, as demonstrated in Listing A.2, enable us to select the desired training period from the complete dataset using the first function, and select the three predefined test periods using the second function. These functions make use of additional simple functions, namely *datetime2unixtimestemp* and *find_closest_indexes*, which are also available in Listing A.2. These functions facilitate the conversion of datetime values to Unix Time and help in finding the closest indexes within the dataset, respectively.

5.2.2 Data Preprocessing

In this section, the concept of data preprocessing will be explored, with a specific focus on two fundamental techniques: data standardization and data normalization. Furthermore, the process of preparing input sequences for the RNN model will be outlined.

Data preprocessing plays a crucial role in machine learning and neural network applications as it enhances the effectiveness and reliability of models. Two commonly used techniques in data preprocessing are data standardization and data normalization.

- **Data Standardization** also known as feature scaling or z-score normalization, involves transforming the values of different variables to have a common scale. This is achieved by subtracting the mean of the variable from each data point and dividing it by the standard deviation. The resulting standardized values have a mean of zero and a standard deviation of one. Standardization is particularly useful when dealing with variables that have different scales or units of measurement. It eliminates the impact of variable scale, enabling fair comparisons and preventing variables with larger values from dominating the analysis or modeling process.

- On the other hand, **Data Normalization**, also known as min-max scaling, rescales the values of a variable to a specific range, typically between 0 and 1. The process involves subtracting the minimum value from each data point and dividing it by the range (maximum value minus minimum value). The normalized values are then constrained within the range of 0 to 1. Normalization is valuable when the absolute values of the variables are less important than their relative values. It ensures that all variables are on a similar scale, facilitating comparison and combination.

The choice between standardization and normalization depends on the specific requirements of the analysis and the characteristics of the data. In this case, data standardization was implemented, as indicated in Listing A.3. The MATLAB script calculates the mean (μ) and standard deviation (σ) of the data, allowing for the standardization of both the training and test datasets. It is crucial to record the standardization parameters for future denormalization or conversion of the data back to its original values.

Another crucial step in the data preprocessing phase is preparing the input sequences for the RNN. The input sequence size is lagged by n-timesteps, which means the RNN expects an input size of n-timesteps to predict the next timestep, known as "one-step ahead" prediction. Listing A.3 demonstrates this process by creating lagged input sequences for both the training and test data.

In this process, the independent variables (*XTrain* and *XTest*) are shifted by n-timesteps, while the dependent variables (*YTrain* and *YTest*) remain unchanged. This formatting ensures that the data is properly structured for training and evaluating the RNN model.

By implementing sliding windows or look-backs, the neural network can learn temporal patterns within the data. The code achieves this by shifting the data back in time by a specified number of timestamps, defined by the variable *vars.Lag*. Subsequently, the data is organized into a suitable format for training the neural network, using the variables *XrTrain*, *YrTrain*, *XrTest*, and *YrTest*.

Choosing the appropriate value for *vars.Lag* depends on various factors related to the data and the specific problem at hand. This value determines the length of the sliding windows and has a significant impact on the temporal aspect of the information considered by the neural network. Some key considerations for determining this value are:

- **Data Rate:** If the data has a low frequency, such as monthly or yearly data, a higher value may be required to capture significant time patterns. Conversely, for high-frequency data, such as daily or hourly data, a lower value may be adequate.
- **Periodicity:** When dealing with data that displays seasonal or cyclical patterns, it is advantageous to choose a value that corresponds to the length of the period. For example, if you are working with monthly data that exhibits yearly seasonal trends, selecting a value of 12 can effectively capture these seasonal patterns.
- **Dataset Size:** The size of the dataset can also influence the choice of this value. With a larger dataset, there is greater flexibility to experiment with different values and determine the one that yields the best results for the given problem. Conversely, when dealing with a small dataset, it is advisable to make more cautious choices to avoid overfitting.

It is recommended to experiment with different values of *vars.Lag* and evaluate the model's performance on a validation dataset. By testing different values, one can determine the optimal value that yields the best results in terms of accuracy, error, or other relevant evaluation metrics specific to the given problem.

Determining the value of *vars.Lag* involved considering several solutions specific to this case. The time series data consisted of approximately 1800/2600 values collected over a two-week period, with an average sampling rate of approximately 129/186 samples per day.

Initially, the possibility of using a sliding two-hour window was explored, taking into account the periodicity observed in the data set. Since the satellite completes one orbit every two hours, this window size was found to be relevant. The next step was to experiment with a

longer period, first with 12 hours, and then with a 24-hour window, to evaluate the network's performance and results.

In all approaches, the *vars.Lag* value was determined based on the number of samples collected within the specified time interval, whether it was two, 12, or 24 hours. You can refer to Listing A.3 to see the detailed process for determining this value. By changing the *numHours* variable in the script, you can easily adjust the sliding window length to the desired number of hours for further experimentation.

By undergoing data standardization and the preparation of input sequences, the data is effectively scaled and organized, thereby optimizing the performance of the DPHM system. This crucial step ensures that the data is in a suitable format for further analysis and modeling, facilitating accurate predictions and enhanced system functionality.

5.3 Network Architecture: CNN-LSTM Hybrid Network

This section presents the network architecture employed in the DPHM system, which utilizes a hybrid model combining CNN and LSTM components to leverage the respective strengths of each model in feature extraction and sequence modeling.

CNNs are particularly effective in extracting meaningful features from complex and high-dimensional data, such as satellite telemetry and sensor readings. With their hierarchical layers of convolution and pooling operations, CNNs excel at capturing spatial and temporal patterns. By incorporating CNNs into the architecture, the goal is to leverage their ability to automatically extract relevant features, enabling efficient learning from satellite data. Figure 5.1 provides a visual representation of the CNN architecture employed for time series forecasting.

To capture the temporal dynamics and long-term dependencies inherent in satellite data, the network architecture integrates LSTM. As demonstrated in Chapter 4, LSTM is a specialized type of RNN that excels in modeling sequential and time-series data. With its unique architecture comprising memory cells and gating mechanisms, LSTM can retain and propagate information over extended periods, effectively capturing crucial temporal patterns within the dataset. Therefore, the RNN component learns to predict the next value at the subsequent timestep.

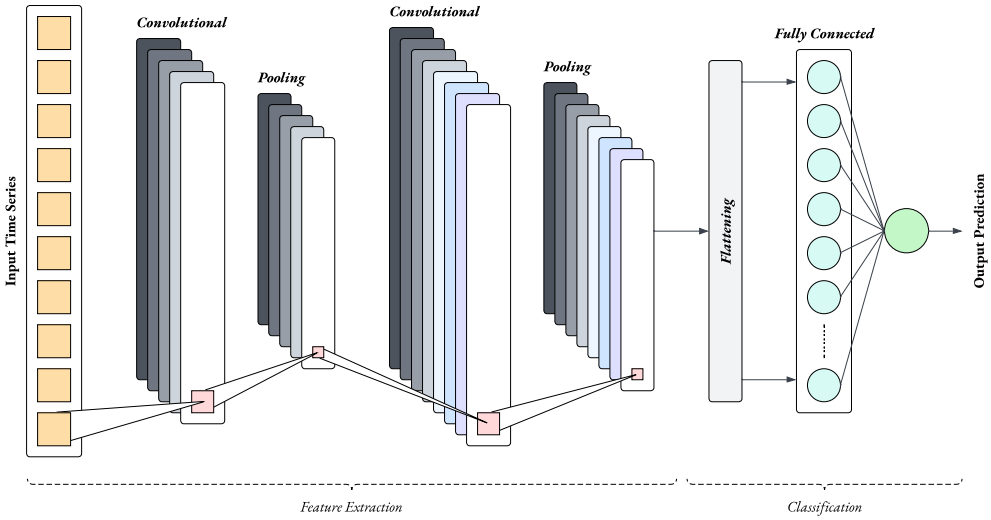


Figure 5.1: Convolutional Neural Network (CNN) Model for Time Series Data Forecasting

The network architecture comprises two main components, as illustrated in the script provided in Listing A.4. Firstly, the architecture incorporates CNN layers, which consist of multiple convolutional layers followed by ELU activation functions. These layers capture hierarchical representations of the data, and max pooling is applied to downsample the feature maps. Once the CNN processing is complete, the output is unfolded and flattened.

The second component of the architecture involves LSTM layers. Both GRU and LSTM layers are utilized for sequence modeling, and dropout layers are incorporated for regularization to mitigate overfitting. The final LSTM layer generates predictions for the next time step. To predict the output for a single time step ahead, a fully connected layer with a single neuron is employed. Given the regression task at hand, a regression layer serves as the final output layer.

By combining the strengths of CNNs for feature extraction and LSTMs for sequence modeling, the CNN-LSTM hybrid network architecture provides a powerful framework for the DPHM system. It enables effective diagnostic and prognostic capabilities, enhancing the system's performance. The inspiration for this architecture can be attributed to an example in MathWorks, developed by H. Sanchez [33].

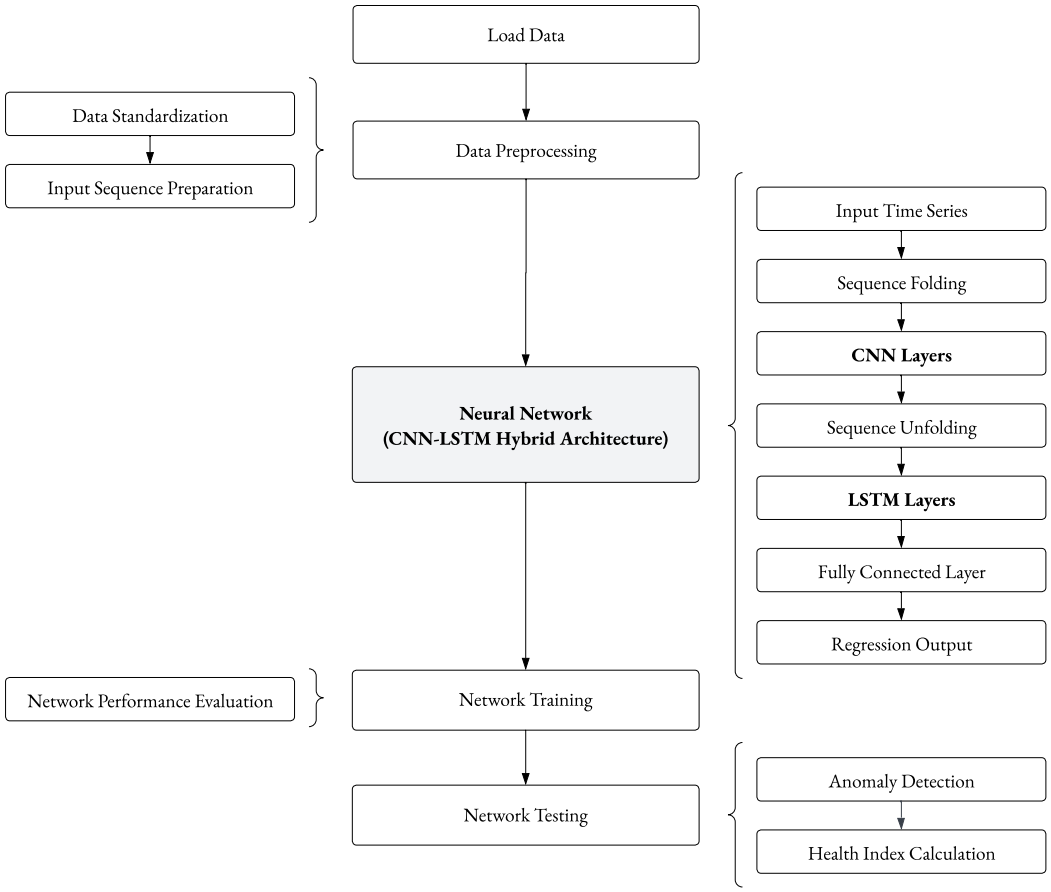


Figure 5.2: Workflow of the Proposed DPHM System (Block Diagram)

To summarize the workflow of the DPHM system, Figure 5.2 illustrates the key steps involved in the proposed system. The process begins with loading the satellite telemetry data and performing pre-processing, including data standardization and preparation of input sequences. The core of the system is the hybrid neural network architecture, which combines CNN and LSTM components for feature extraction and sequence modeling. The network is trained using the prepared data, and its performance is evaluated. Subsequently, the trained network is tested, and anomaly detection techniques are applied to identify deviations between expected and observed values. Finally, a health index is calculated based on the deviation, enabling proactive maintenance and decision-making. The next section will delve into the analysis of the results obtained during the training and testing processes.

5.4 Evaluation Metrics

In this section, the results of the implemented DPHM system are presented, beginning with a discussion on the evaluation metrics employed to assess the network's performance. Key metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Normalized Root Mean Squared Error (NRMSE), are emphasized as important indicators of the accuracy and effectiveness of the network throughout the training and testing phases.

- **Mean Squared Error (MSE):** The MSE measures the average squared difference between the predicted values ($\hat{y}_i$) and the actual values (y_i). It provides an overall assessment of the model's accuracy, with lower values indicating better performance.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error (RMSE):** The RMSE is the square root of the MSE and provides a measure of the average magnitude of the prediction errors. It is a commonly used metric to quantify the model's predictive accuracy, with lower values indicating better performance.

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Normalized Root Mean Squared Error (NRMSE):** The NRMSE is a variation of the RMSE that scales the error by the range of the actual values ($\max(y) - \min(y)$). It provides a normalized measure of the prediction error, allowing for better comparison across different datasets and variables. NRMSE values closer to zero indicate higher prediction accuracy.

$$\text{NRMSE} = \frac{\text{RMSE}}{\max(y) - \min(y)}$$

These evaluation metrics enable the quantitative assessment of the performance of the DPHM system and facilitate a comparison with alternative methods. By incorporating multiple metrics, a comprehensive understanding of the strengths and limitations of the network in capturing underlying patterns and trends in satellite telemetry data is achieved.

The subsequent subsections delve into the results attained during the training and testing processes, providing an analysis of the network's performance based on these evaluation metrics.

5.5 Training Process

The training process begins by configuring the desired training options, which include parameters such as the solver name, maximum epochs, mini-batch size, learning rate, and execution environment. These options play a crucial role in determining the behavior and performance of the training process. Once the training options are set, the hybrid network is trained using the *trainNetwork* function. This function takes the training data, consisting of the input data *XrTrain* and the target data *YrTrain*, along with the defined layers and options. The script used for training is provided in Listing A.5, where all the values of the training options reported in this section have been reported.

During the training process, the hybrid architecture, which combines CNN and LSTM components, is utilized to extract meaningful representations from the input data and capture temporal dependencies. Through iterative adjustments of its parameters, the network strives to minimize the discrepancy between expected and target outputs. As the training progresses, the network learns to extract relevant features from the data and optimizes its weights and biases to make accurate predictions. The resulting trained hybrid network, referred to as *hybrid_network*, encapsulates the acquired knowledge and can be further employed for diagnostic and prognostic tasks.

In the context of this thesis, five different networks have been trained, with each network dedicated to a specific telemetry range. These networks are designed to handle the unique characteristics and patterns present in each telemetry dataset, optimizing the performance and accuracy of the predictions for their respective ranges.

Training Options

The training options used in the script provide control over various aspects of the training process. They are defined as follows:

- **Solver Name:** The solver specifies the optimizer for training the neural network. There are three options:
  - 'sgdm': Stochastic Gradient Descent with Momentum (SGDM) optimizer.
  - 'rmsprop': RMSProp optimizer.
  - 'adam': Adam optimizer.

- **Plots and Display**

  - **Plots:** Determines which plots to display during neural network training. Two options are available:
    * 'none': No plots are displayed during training.
    * 'training-progress': Plots the training progress.

  - **Verbose:** Controls the display of training progress information.
    * '1' (true): Display training progress information.
    * '0' (false): Do not display training progress information.

- **Mini-Batch Options**

  - **MaxEpochs:** Specifies the maximum number of epochs for training. It is a positive integer.
  - **Mini-Batch Size:** Determines the size of the mini-batch used for each training iteration. It is a positive integer. A mini-batch is a subset of the training set used to evaluate the gradient of the loss function and update the weights.
  - **Shuffle:** Specifies the option for shuffling the data during training and validation.
    * 'once': Shuffle the training and validation data once before training.
    * 'never': Do not shuffle the data.
    * 'every-epoch': Shuffle the training data before each training epoch and shuffle the validation data before each neural network validation.

- **Solver Options**

  - **InitialLearnRate:** Sets the initial learning rate used for training. It is a positive scalar. The default values are 0.01 for the 'sgdm' solver and 0.001 for the 'rmsprop' and 'adam' solvers.
  - **LearnRateSchedule:** Controls the dropping of the learning rate during training.
    * 'none': The learning rate remains constant throughout training.

    * 'piecewise': The learning rate is updated every certain number of epochs by multiplying it with a certain factor. The value of this factor is specified using the 'LearnRateDropFactor' training option. The number of epochs between multiplications is set using the 'LearnRateDropPeriod' training option.

  - **LearnRateDropPeriod:** Number of epochs for dropping the learning rate, specified as a positive integer. This option is valid only when the LearnRateSchedule training option is 'piecewise'. The software multiplies the global learning rate with the drop factor every time the specified number of epochs passes. Specify the drop factor using the LearnRateDropFactor training option.
  - **LearnRateDropFactor:** Factor for dropping the learning rate, specified as a scalar from 0 to 1. This option is valid only when the LearnRateSchedule training option is 'piecewise'. LearnRateDropFactor is a multiplicative factor to apply to the learning rate every time a certain number of epochs passes. Specify the number of epochs using the LearnRateDropPeriod training option.

- **Gradient Clipping**

  - **GradientThreshold:** Sets the threshold for gradient clipping. It is specified as 'Inf' or a positive scalar. If the gradient exceeds the value of 'GradientThreshold', it is clipped according to the 'GradientThresholdMethod' training option.

- **Hardware Options**

  - **ExecutionEnvironment:** Determines the hardware resource to be used for training the neural network.
    * 'auto': Use a GPU if available; otherwise, use the CPU.
    * 'cpu': Use the CPU.
    * 'gpu': Use the GPU.
    * 'multi-gpu': Use multiple GPUs on one machine, utilizing a local parallel pool based on the default cluster profile. If no current parallel pool exists, the software starts a parallel pool with a size equal to the number of available GPUs.
    * 'parallel': Use a local or remote parallel pool based on the default cluster profile. If no current parallel pool exists, the software starts one using the default cluster profile. If the pool has access to GPUs, only workers with a unique GPU perform training computation. If the pool does not have GPUs, training takes place on all available CPU workers instead.

Network Performance Evaluation

For evaluating the performance of the network, the following lines of code were executed after the training phase:

Listing 5.1: MATLAB Script for Generating Prediction for the Training Data

```
YPred_Train = predict(hybrid_network, XrTrain, 'ExecutionEnvironment',
    optns.ExecutionEnvironment, 'MiniBatchSize', numFeatures);
YPred_Train = YPred_Train';
YPred_Train = sigTrain. * YPred_Train + muTrain;
YTrain      = sigTrain. * YTrain + muTrain;
```

The provided lines of code demonstrate the process of generating predictions for the training data (*XrTrain*) using the trained hybrid network (*hybrid_network*), using the *predict* function. After obtaining the predictions (*YPred_Train*), the necessary steps are taken to rescale them to their original scale. This involves transposing *YPred_Train*, followed by rescaling it using the factors *sigTrain* and *muTrain*. Similarly, the original target values (*YTrain*) are also rescaled using the same factors. These rescaling steps ensure that the predictions and target values are aligned in terms of their original scale, facilitating a meaningful evaluation of the network's performance.

As stated previously, the approach involved training individual networks for each parameter using three different values of the "vars.Lag" parameter: 2, 12, and 24 hours. Consequently, for each telemetry, three networks were trained. Since there are five parameters in total, a total of 15 networks were trained. By evaluating the performance of each network and calculating relevant metrics, the optimal network can be selected from the available options.

In fact, this section outlines the process of evaluating these metrics and provides an overview of the obtained results.

The MATLAB script in Listing A.6 demonstrates how the performance metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Normalized Root Mean Squared Error (NRMSE), were calculated. These metrics provide valuable insights into the accuracy and reliability of the network predictions by measuring the residual error between the expected results and the observed values.

The network performances were then summarized in three tables: Table 5.2, Table 5.3, and Table 5.4. These tables correspond to the networks trained with sliding windows of 2, 12, and 24 hours respectively. The tables present key parameters related to the error and NRMSE, in-

cluding the mean and standard deviation of the error, as well as the total, mean, and standard deviation of the NRMSE. The optimal values for each telemetry are highlighted in blue. Based on these results, it was determined that the network with a sliding window of 24 hours is the best choice for *current*_1 and *current*_2 telemetry, the network with a 2-hour sliding window is preferred for *current*_3 and *voltage*_2 telemetry, and the network with a 12-hour sliding window is selected for *voltage*_1 telemetry.

The best calculated performance metrics were then visualized from Figure 5.5 to Figure 5.19. These figures provide a comprehensive overview of the network's performance for each telemetry parameter.

For each parameter, the first figure displays a graph comparing the expected results with the observed values, allowing for a visual assessment of the network's predictive capabilities. The second figure presents an histogram of the errors, illustrating the distribution of the residual errors and providing insights into the Error Mean and Error Standard Deviation. Finally, the last figure depicts a histogram of the NRMSE, which provides a normalized measure of the prediction error. The figure includes the Total NRMSE value (NRMSE calculated over the entire training window), NRMSE Mean, and NRMSE Standard Deviation, providing a comprehensive understanding of the network's performance in terms of prediction accuracy.

Parameter	Error		NRMSE		
	Mean	StD	Total	Mean	StD
current_1	0.3830	5.5956	0.0036513	0.026011	0.037904
current_2	0.89478	99.2483	0.31211	0.062909	0.089091
current_3	0.037007	4.2267	0.032256	0.032757	0.038987
voltage_1	0.10599	8.2503	0.0005139	0.0078624	0.008227
voltage_2	0.051303	0.79727	0.0049288	0.0066276	0.010578

Table 5.2: Performance Metrics of Telemetry Errors in the Training Phase with $vars.Lag = 2\ Hours$

Parameter	Error		NRMSE		
	Mean	StD	Total	Mean	StD
current_1	0.25565	1.312	0.0086939	0.0069451	0.0084725
current_2	1.3301	123.1662	0.38702	0.080009	0.10917
current_3	-0.0011145	12.1265	0.092565	0.10522	0.10134
voltage_1	-0.92308	6.12	0.00038548	0.0061601	0.0059091
voltage_2	0.038875	10.2001	0.062934	0.11852	0.10653

Table 5.3: Performance Metrics of Telemetry Errors in the Training Phase with $vars.Lag = 12$ *Hours*

Parameter	Error		NRMSE		
	Mean	StD	Total	Mean	StD
current_1	0.029695	1.2293	0.0080048	0.0075383	0.0066887
current_2	2.1611	22.7218	0.071655	0.0092789	0.023302
current_3	0.0053131	12.167	0.0.092895	0.10553	0.10172
voltage_1	-0.25341	6.3622	0.00039663	0.0068	0.0055
voltage_2	0.0081426	1.9081	0.0.011771	0.0174	0.0242

Table 5.4: Performance Metrics of Telemetry Errors in the Training Phase with $vars.Lag = 24$ *Hours*

Figures 5.3 and 5.4 visually depict the information presented in the aforementioned tables. These figures offer a graphical representation of the mean absolute error and mean NRMSE for each telemetry, considering the different values of vars.Lag. By examining these figures, we can easily compare the performance of the algorithm and network across the various telemetries.

Figure 5.3: Comparison of Absolute Error Mean for Different Sliding Window

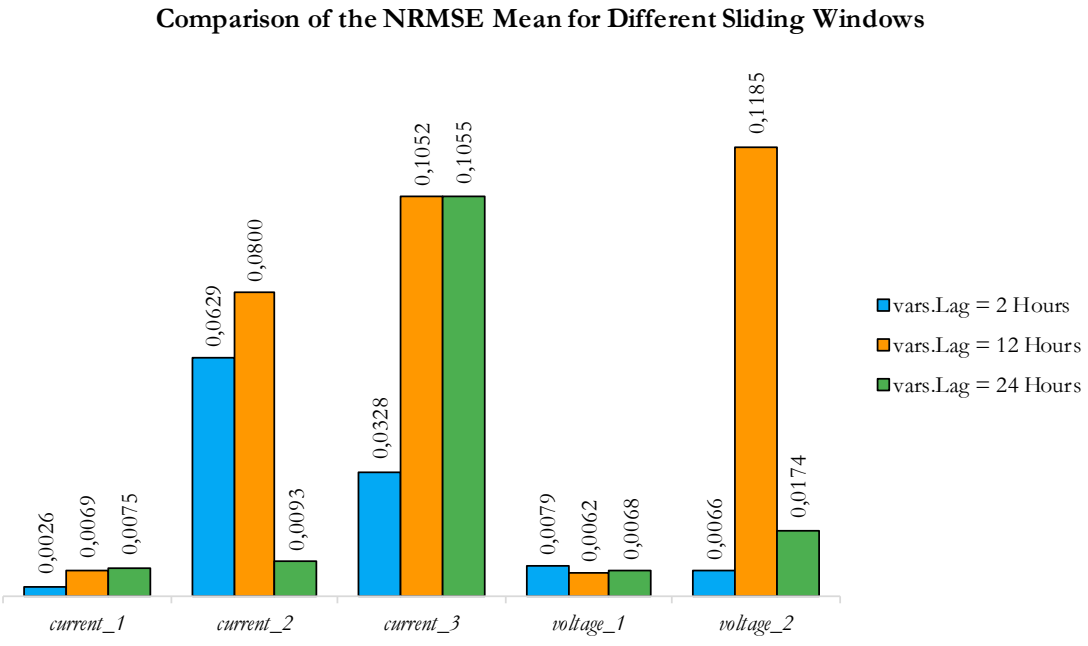


Figure 5.4: Comparison of NRMSE Mean for Different Sliding Window

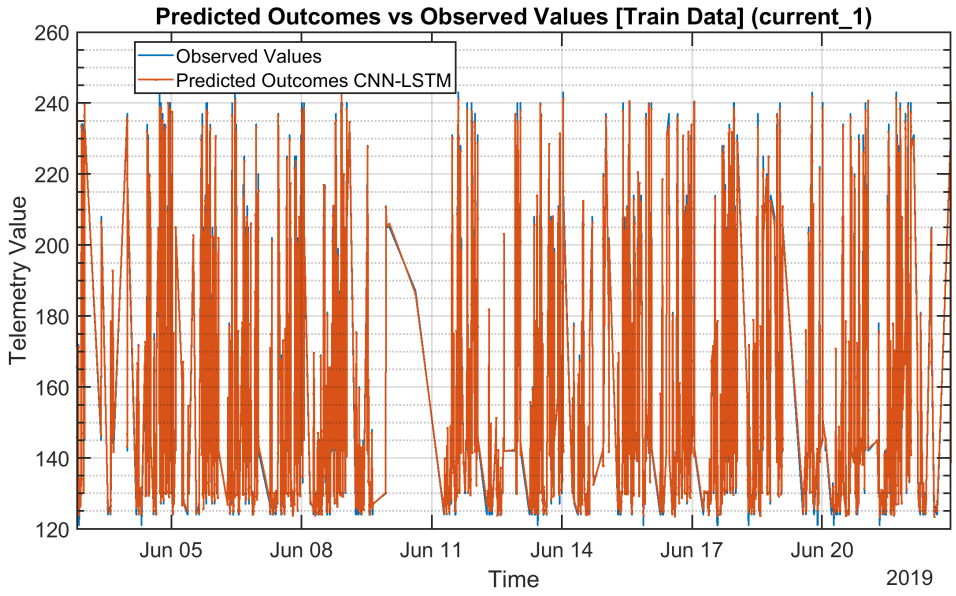


Figure 5.5: Predicted Outcomes v s Observed Values [Train Data] (*current*_1) with $vars.Lag = 24\ Hours$

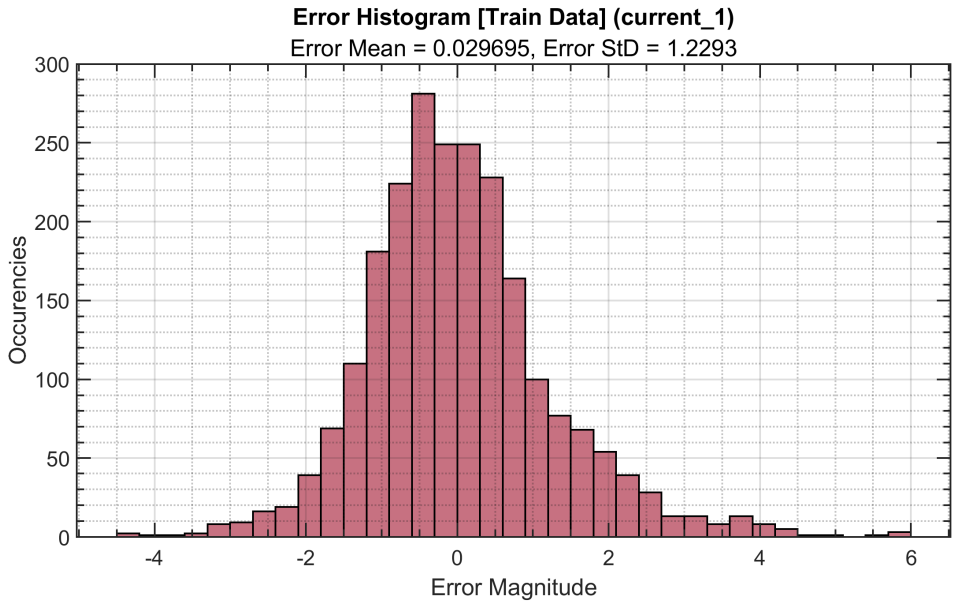


Figure 5.6: Error Histogram [Train Data] (*current*_1) with $vars.Lag = 24\ Hours$

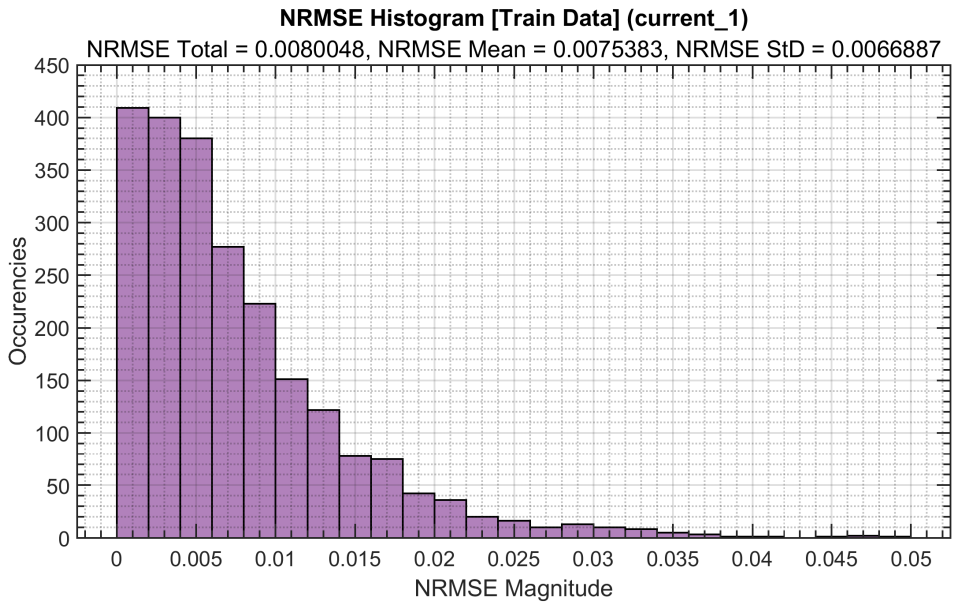


Figure 5.7: NRMSE Histogram [Train Data] (*current*_1) with $vars.Lag = 24\ Hours$

CURRENT_2

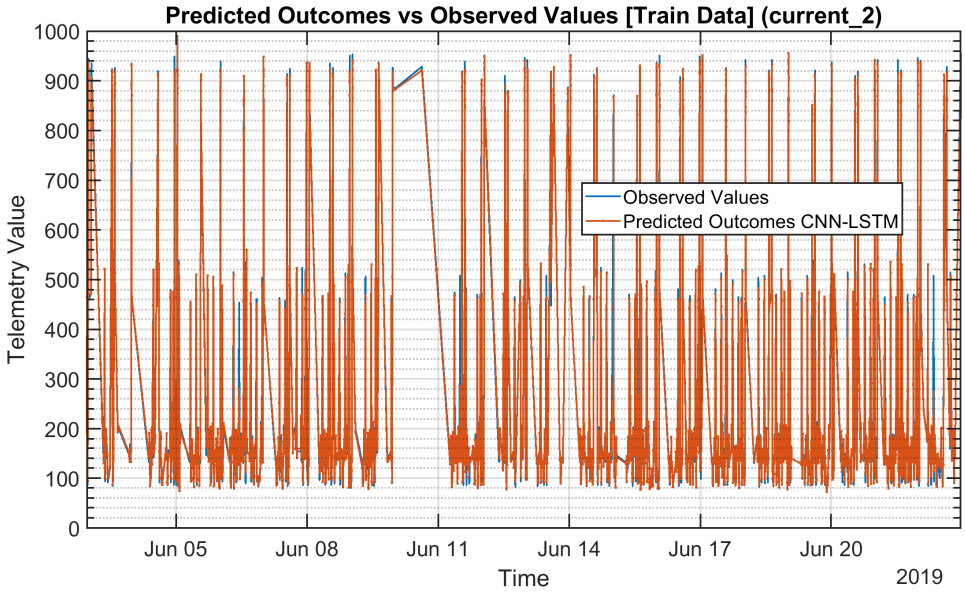


Figure 5.8: Predicted Outcomes v s Observed Values [Train Data] (*current*_2) with $vars.Lag = 24\ Hours$

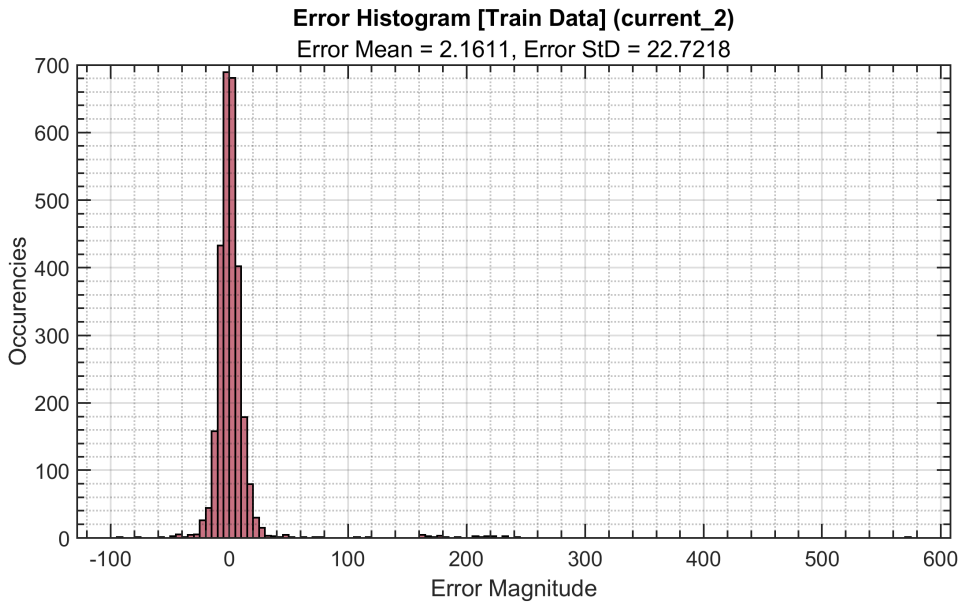


Figure 5.9: Error Histogram [Train Data] (*current_2*) with $vars.Lag = 24\ Hours$

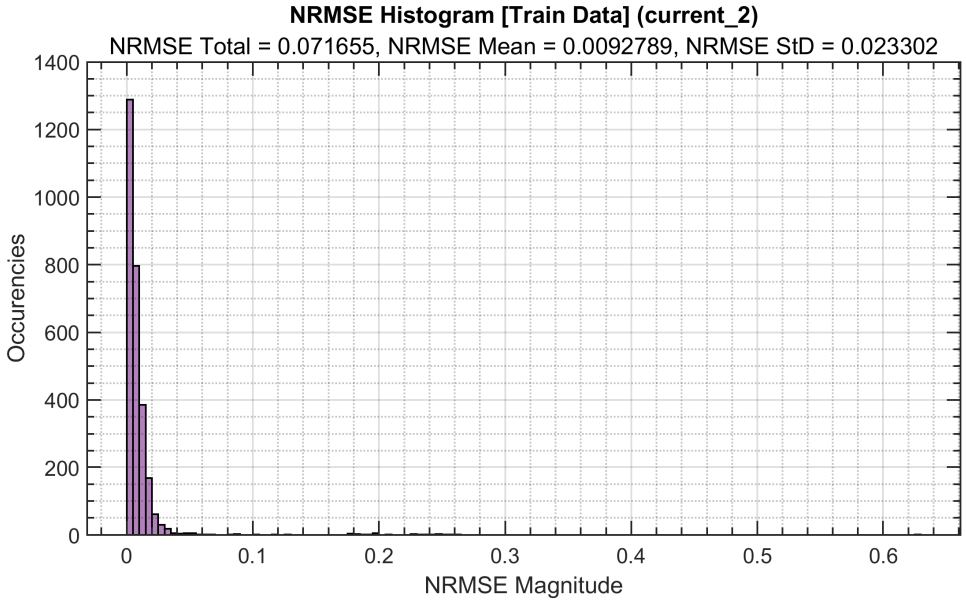


Figure 5.10: NRMSE Histogram [Train Data] (*current_2*) with $vars.Lag = 24\ Hours$

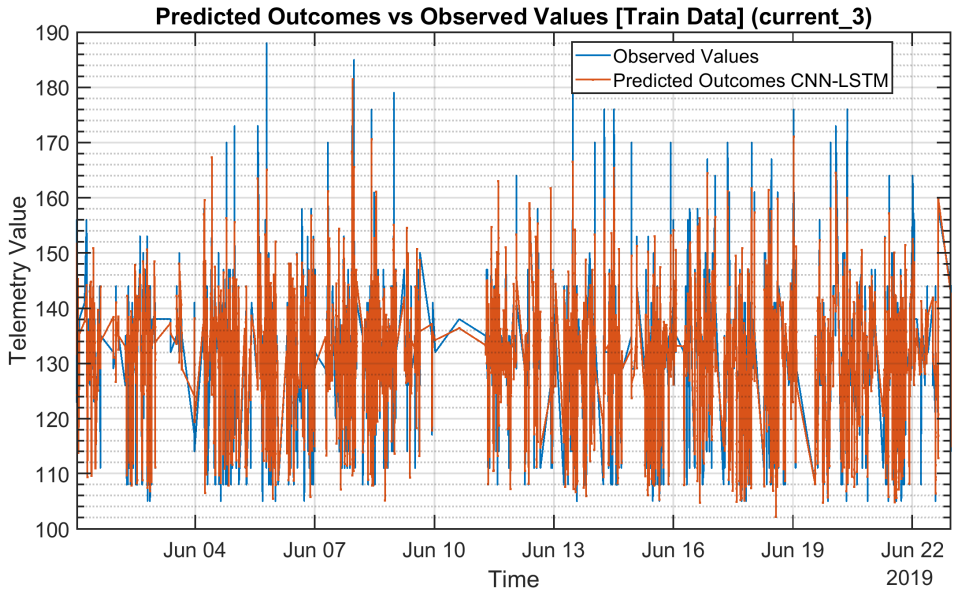


Figure 5.11: Predicted Outcomes v s Observed Values [Train Data] (*current_3*) with $vars.Lag = 2\ Hours$

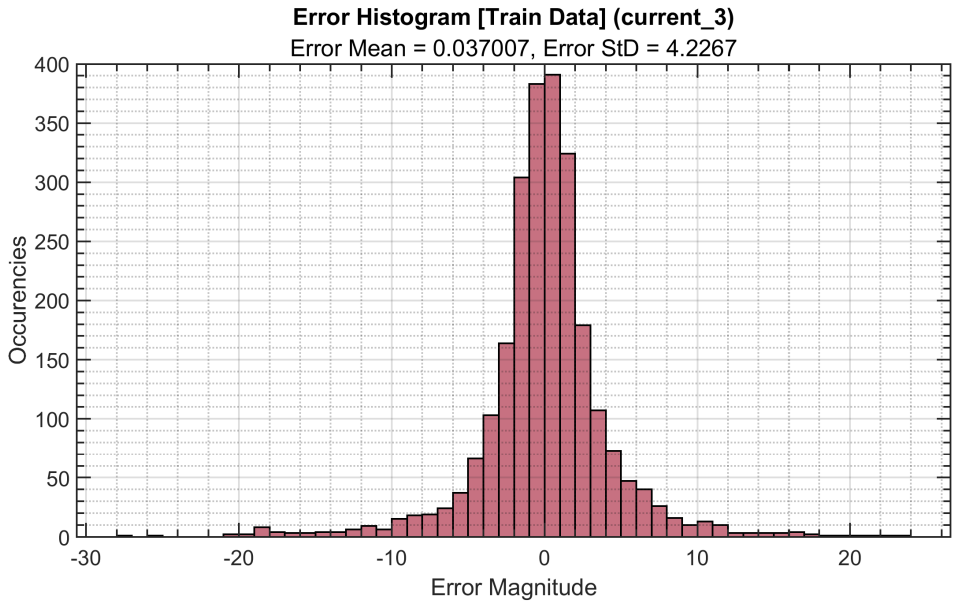


Figure 5.12: Error Histogram [Train Data] (*current_3*) with $vars.Lag = 2\ Hours$

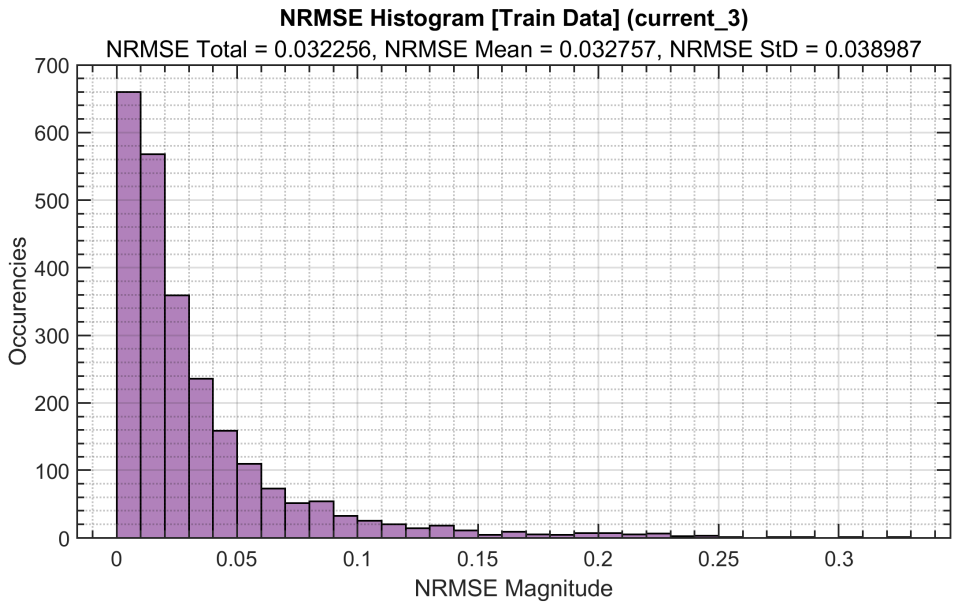


Figure 5.13: NRMSE Histogram [Train Data] (*current_3*) with $vars.Lag = 2\ Hours$

VOLTAGE_I

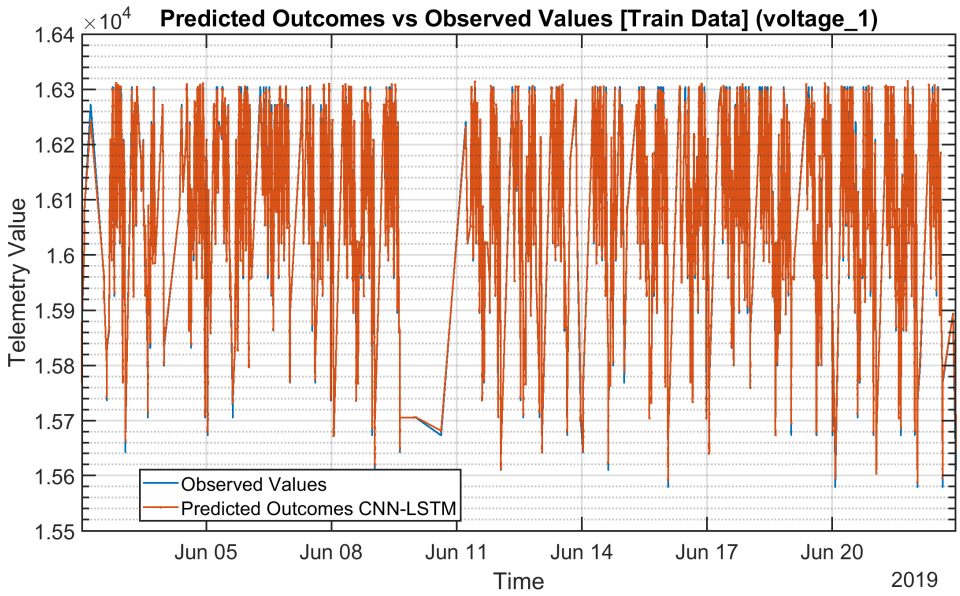


Figure 5.14: Predicted Outcomes v s Observed Values [Train Data] (*voltage_1*) with $vars.Lag = 12\ Hours$

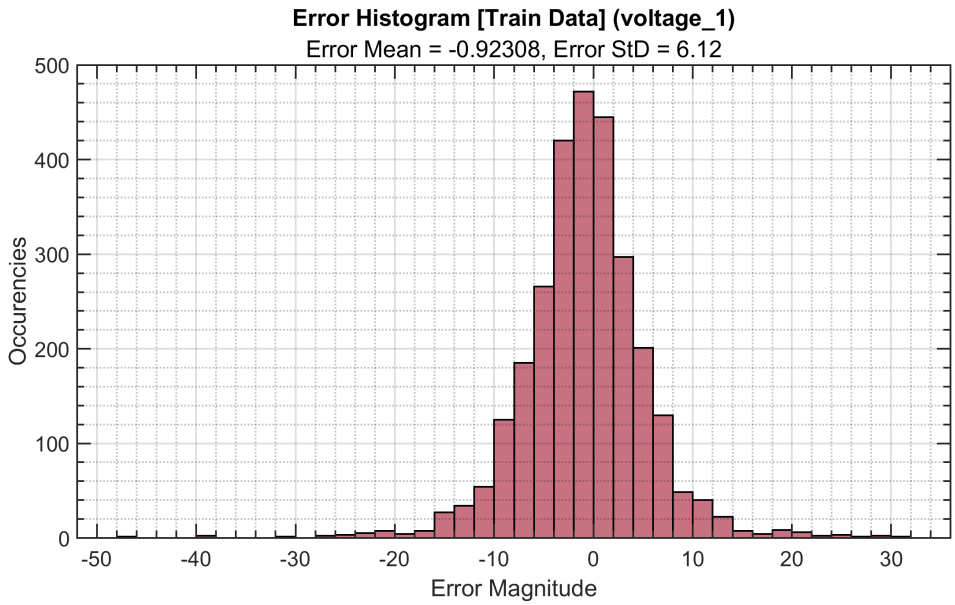


Figure 5.15: Error Histogram [Train Data] (*voltage*_1) with $vars.Lag = 12\ Hours$

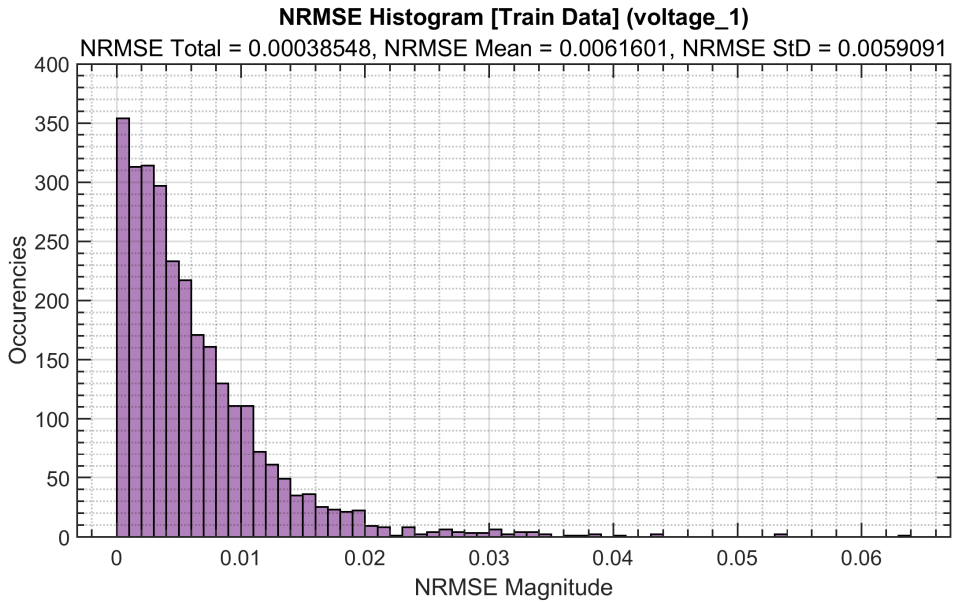


Figure 5.16: NRMSE Histogram [Train Data] (*voltage*_1) with $vars.Lag = 12\ Hours$

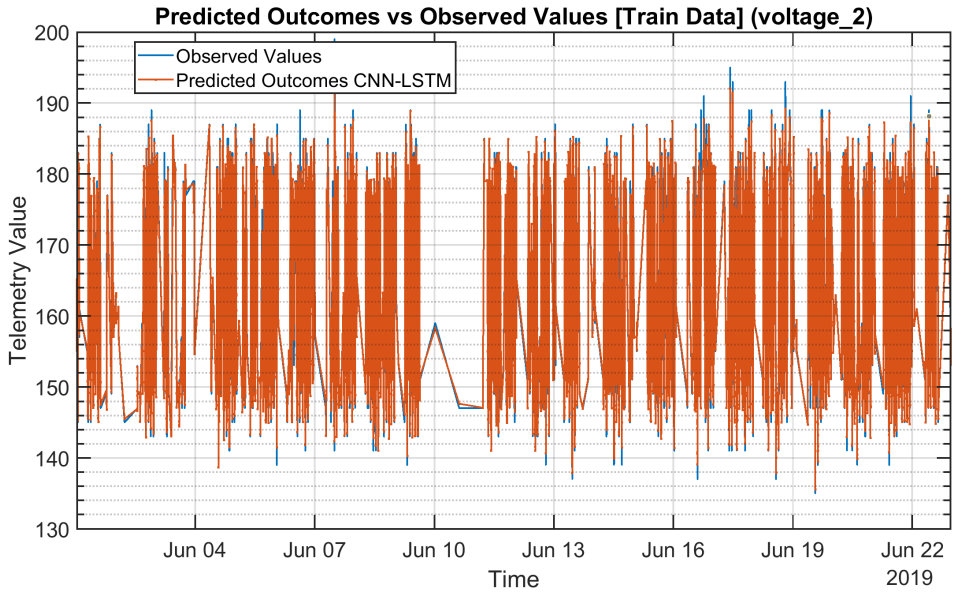


Figure 5.17: Predicted Outcomes v s Observed Values [Train Data] (*voltage_2*) with $vars.Lag = 2\ Hours$

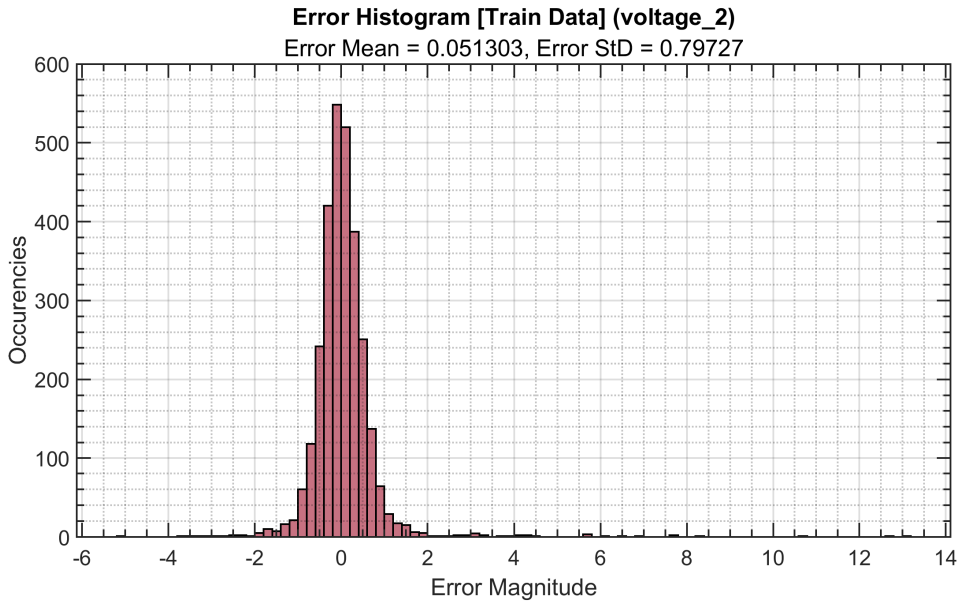


Figure 5.18: Error Histogram [Train Data] (*voltage_2*) with $vars.Lag = 2\ Hours$

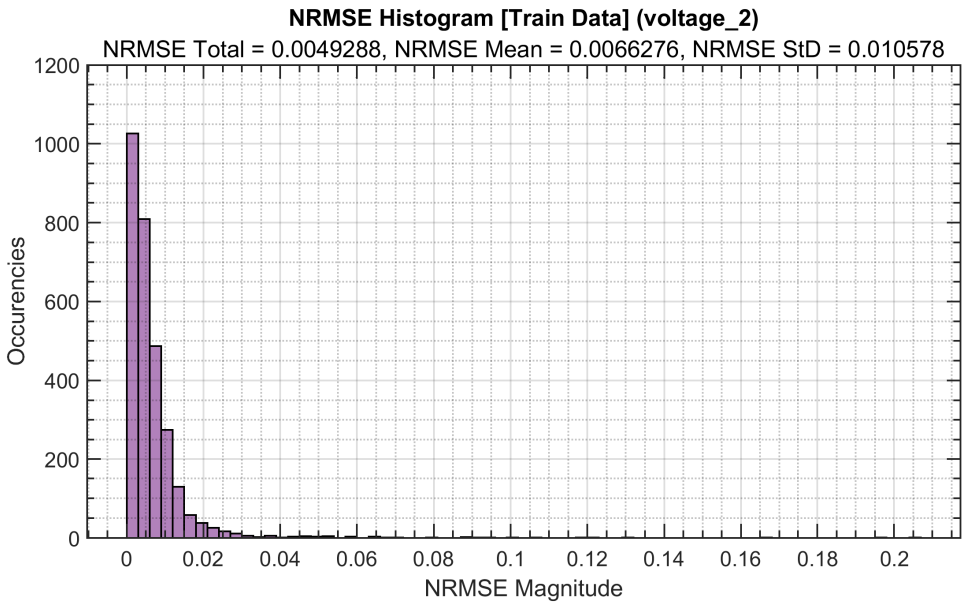


Figure 5.19: NRMSE Histogram [Train Data] (*voltage_2*) with $vars.Lag = 2$ *Hours*

5.6 Testing Process

The testing phase involves applying the trained network to new input sequences and analyzing the predictions and performance metrics. The MATLAB script provided in Listing A.7 demonstrates the testing process of the hybrid network.

In this script, similar to the training process, the trained network is used to predict the output, referred to as *YPred_Test*, for the test input sequences represented by *XrTest*. The predicted values are then transformed back to their original scale using the scaling factors *sigTest* and *muTest*, resulting in *YPred_Test* and *YTest*.

Following this initial step, the testing process delves into the intriguing aspect of the system: anomaly detection and health index calculation. The next lines of code provide insight into the logic behind the anomaly detection phase:

Listing 5.2: MATLAB Script for Anomaly Detection

```
lowerThreshold = prctile(abs(Errors_test), 0);
upperThreshold = prctile(abs(Errors_test), 80);
window_size = maxLag;
anomalies = [];
nrmse_anomaly = [];
```

```
6
7  % Determining the appropriate thresholds and upper limits for the
       specific telemetry
8  if data.telemetry{1} == "current_1"
9      nrmse_threshold = 3.55;
10     hi_upper_th = 100;
11 elseif data.telemetry{1} == "voltage_1"
12     nrmse_threshold = 1.5;
13     hi_upper_th = 100;
14 elseif data.telemetry{1} == "current_2"
15     nrmse_threshold = 0.33;
16     hi_upper_th = 100;
17 elseif data.telemetry{1} == "voltage_2"
18     nrmse_threshold = 1.4;
19     hi_upper_th = 100;
20 elseif data.telemetry{1} == "current_3"
21     nrmse_threshold = 1.4;
22     hi_upper_th = 100;
23 end
24
25 % Looping through the errors to identify anomalies
26 for i = 1:size(Errors_test, 2)
27     if i > window_size && i < size(Errors_test, 2)
28         anomalies(1,i) = abs(Errors_test(1, i)) < lowerThreshold | abs
               (Errors_test(1, i)) > upperThreshold;
29         if anomalies(1,i)
30             % Calculate NRMSE in a window of window_size timestamps
                   before the anomaly
31             windowStart = i - window_size;
32             windowEnd = i;
33             window = Errors_test(1, windowStart:windowEnd);
34             nrmse_anomaly(1, i) = sqrt(mean(window.^2)) / (max(
                   Errors_train) - min(Errors_train)); % PA3.0
35
```

```
            if nrmse_anomaly(1, i) < nrmse_threshold
                anomalies(1, i) = 0;
                nrmse_anomaly(1, i) = 0;
            end
        end
    else
        anomalies(1, i) = 0;
        nrmse_anomaly(1, i) = 0;
    end
end
```

Within this section, the anomaly detection procedure revolves around the establishment of lower and upper thresholds, which are based on the percentiles derived from the absolute errors (*Errors_test*). The size of the detection window, referred to as *window_size*, is determined by the length of the sliding window specified during the preprocessing phase.

Subsequently, the code segment dynamically assigns specific thresholds and upper limits according to the telemetry parameter under examination. For each timestamp within the error array, the script assesses whether it falls within the designated window and verifies if the error value surpasses the threshold boundaries. If an anomaly is identified, the script proceeds to calculate the NRMSE within the window preceding the anomaly, subsequently comparing it against the predefined NRMSE threshold. Should the NRMSE value fall below the specified threshold, the anomaly is disregarded.

Throughout the execution of this process, the arrays *anomalies* and *nrmse_anomaly* serve the purpose of storing binary indications corresponding to the presence of anomalies, alongside their associated NRMSE values, respectively.

The final step of this process involves calculating the Health Index (HI). This index serves as a metric to evaluate the overall system status and the severity of anomalies. It is measured on a scale ranging from 0 to 100, where 0 signifies a completely healthy system, while 100 represents the highest level of criticality. During the testing phase, when anomalies are detected, the health index is determined using the following code:

Listing 5.3: MATLAB Script for Health Index Calculation

```
HI = [];
for i = 1:size(Errors_test, 2)
    if nrmse_anomaly(1, i) < nrmse_threshold
```

```
        HI(1, i) = 0;
    elseif nrmse_anomaly(1, i) >= nrmse_threshold && nrmse_anomaly(1,
        i) < hi_upper_th
        HI(1, i) = nrmse_anomaly(1, i) * 100;
    elseif nrmse_anomaly(1, i) >= hi_upper_th
        HI(1, i) = 100;
    end
end
```

For each detected anomaly, the following conditions are evaluated:

- If the normalized root mean squared error (*nrmse_anomaly*) is less than the defined threshold (*nrmse_threshold*), the HI is assigned a value of 0, indicating no criticality.
- If *nrmse_anomaly* is greater than or equal to *nrmse_threshold* but less than the upper threshold (*hi_upper_th*), the HI is computed by multiplying *nrmse_anomaly* by 100.
- If *nrmse_anomaly* exceeds the *hi_upper_th*, the HI is set to 100, indicating maximum criticality.

Now, let's examine the results of this process for all five telemetries. In the following subsections, we will explore the behavior of the networks in the three different test periods, focusing on one telemetry at a time. For each test period, three figures will be presented:

1. The first figure showcases a comparison between the predicted values and the observed values, providing an assessment of the accuracy of the network's predictions in the absence of anomalies. In the presence of anomalies, this figure offers insights into the deviation between the predicted and observed values, facilitating the evaluation of anomaly detection and characterization.
2. The second figure illustrates the residual error, which represents the discrepancy between the observed and predicted values. This visualization serves as a valuable tool for analyzing the system's behavior and identifying any patterns or trends in the prediction errors.
3. The third figure showcases the HI graph, providing insights into whether the network has successfully identified anomalies.

By examining these figures, we can gain a comprehensive understanding of how the networks perform across different test periods and telemetry parameters.

5.6.1 CURRENT_1

Test Period 1: from 24-Jun-2019 to 07-Jul-2019

The observed period exhibits two distinct anomalous behaviors: one from June 25th to June 30th, and another from July 1st to July 7th.

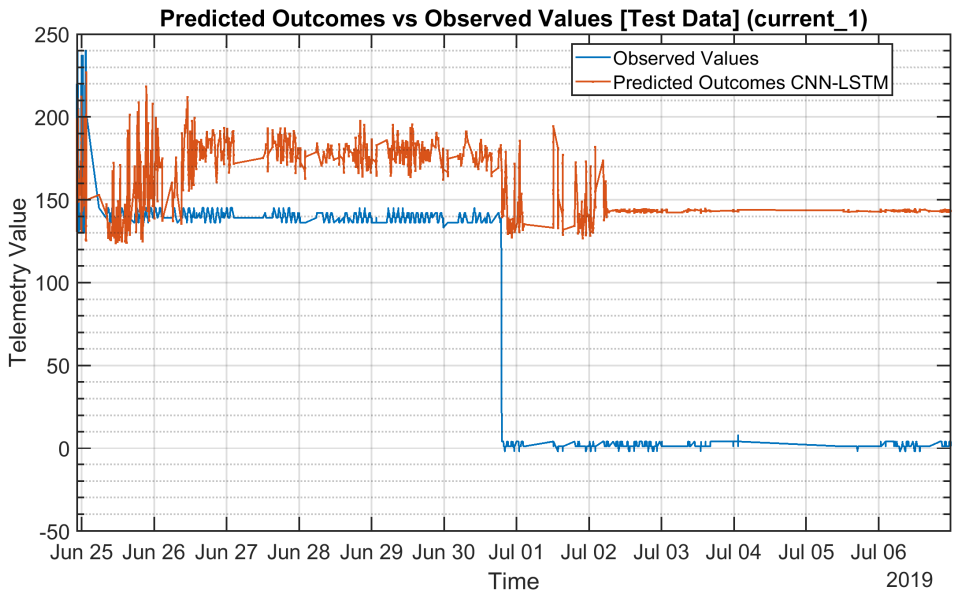


Figure 5.20: Predicted Outcomes v s Observed Values [Test Period 1] (*current_1*) with $vars.Lag = 24\ Hours$

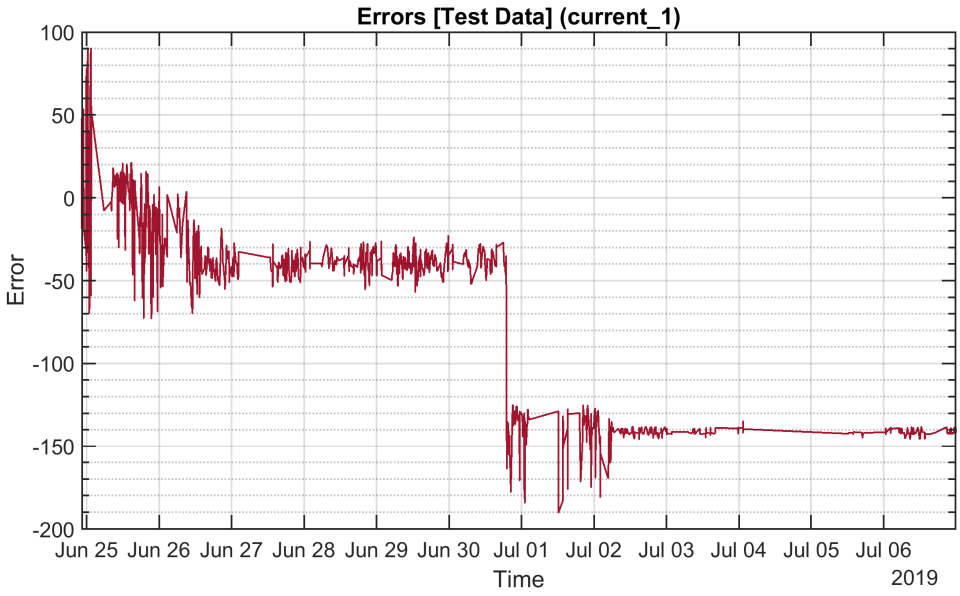


Figure 5.21: Errors [Test Period 1] (*current*_1) with $vars.Lag = 24\ Hours$

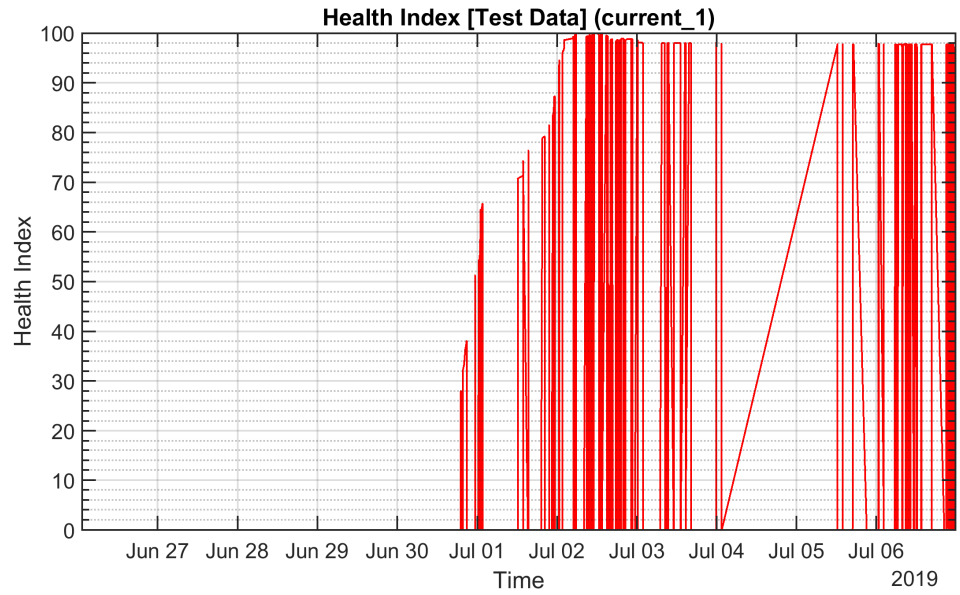


Figure 5.22: Health Index [Test Period 1] (*current*_1) with $vars.Lag = 24\ Hours$

The algorithm fails to identify the initial abnormal period because it disregards the insignificant deviations. This can be characterized as a missed detection, or false negative. On the other hand, the algorithm accurately identifies the second anomaly, resulting in a true positive.

During this timeframe, there is a nominal period until September 19th, followed by two distinct anomalous behaviors: the first occurring between September 19th and September 23rd, and the second between September 23rd and September 24th.

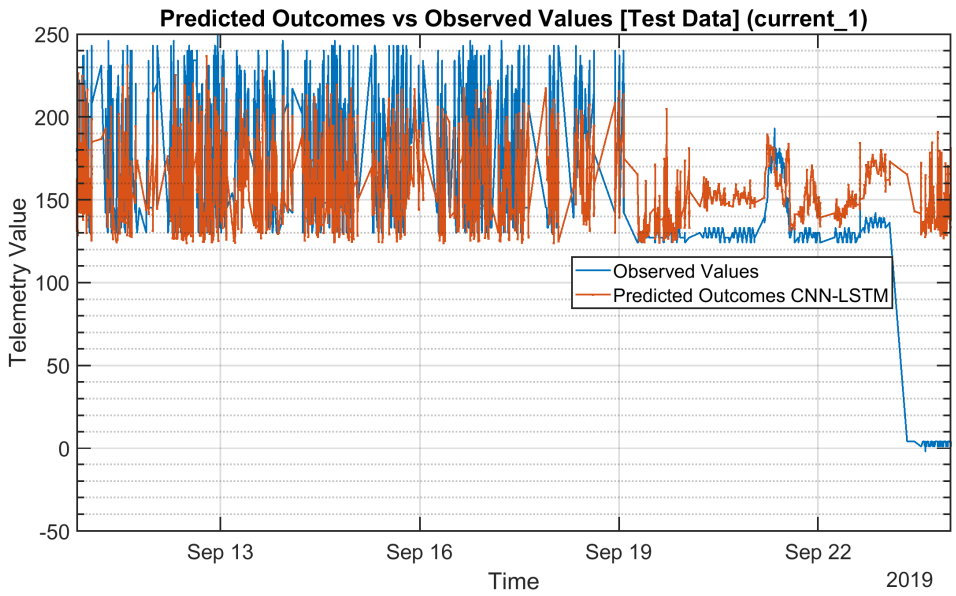


Figure 5.23: Predicted Outcomes v s Observed Values [Test Period 2] (*current_1*) with $vars.Lag = 24\ Hours$

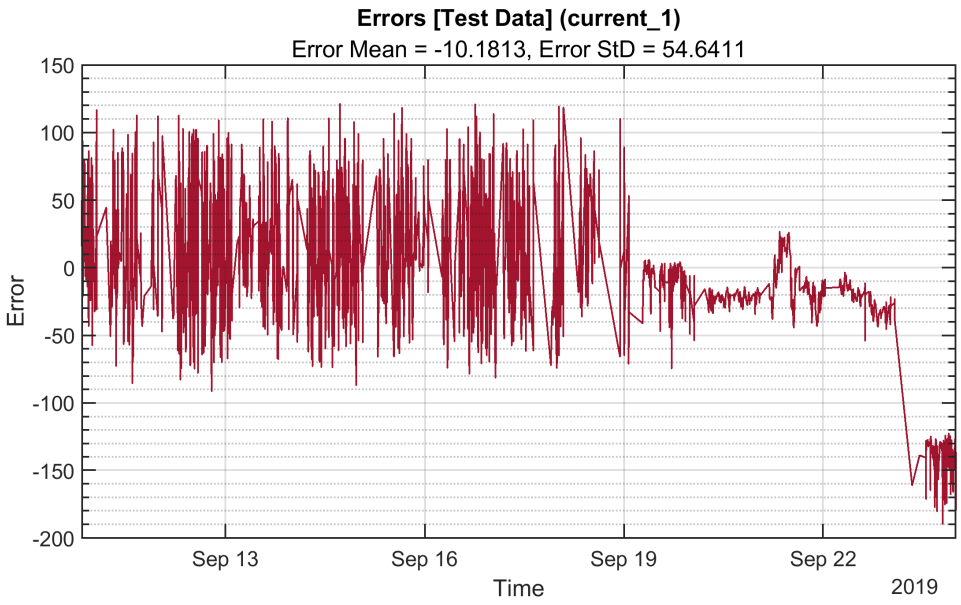


Figure 5.24: Errors [Test Period 2] (*current*_1) with $vars.Lag = 24\ Hours$

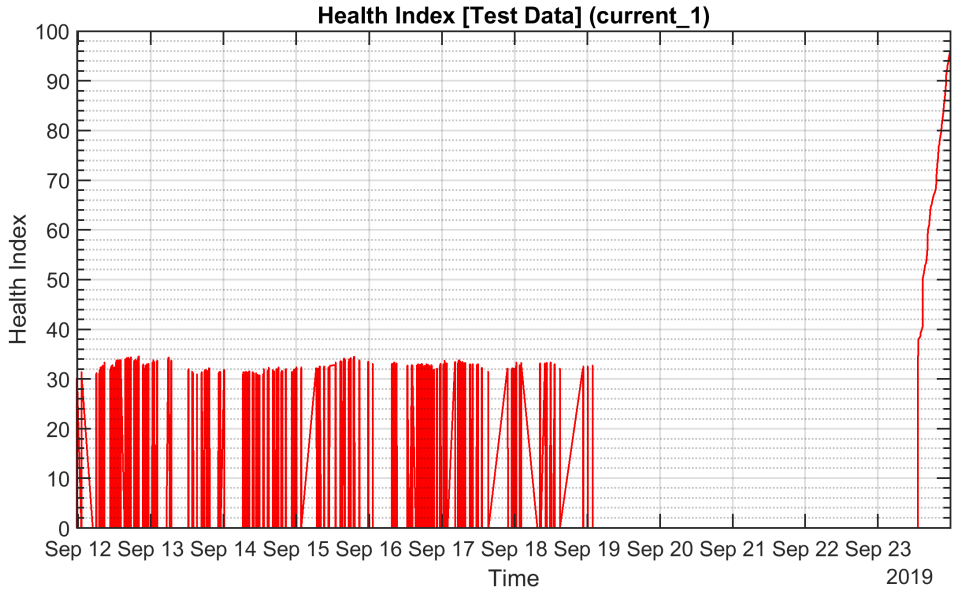


Figure 5.25: Health Index [Test Period 2] (*current*_1) with $vars.Lag = 24\ Hours$

The algorithm's performance in this particular case is not meeting expectations. Surprisingly, it is incorrectly labeling normal behavior as anomalous, resulting in false positives. Additionally, it fails to identify the initial period of anomalies because it disregards insignificant deviations,

leading to a missed detection or false negative. However, the algorithm accurately detects the second anomaly, which can be classified as a true positive.

Test Period 3: from 01-Oct-2019 to 15-Oct-2019

During this period, there are distinct phases: an initial nominal period until October 3rd, followed by two separate anomalous behaviors. The first anomalous behavior occurs from October 4th to October 7th, and the second one from October 8th to October 12th. Subsequently, there is another nominal period from October 12th to October 14th.

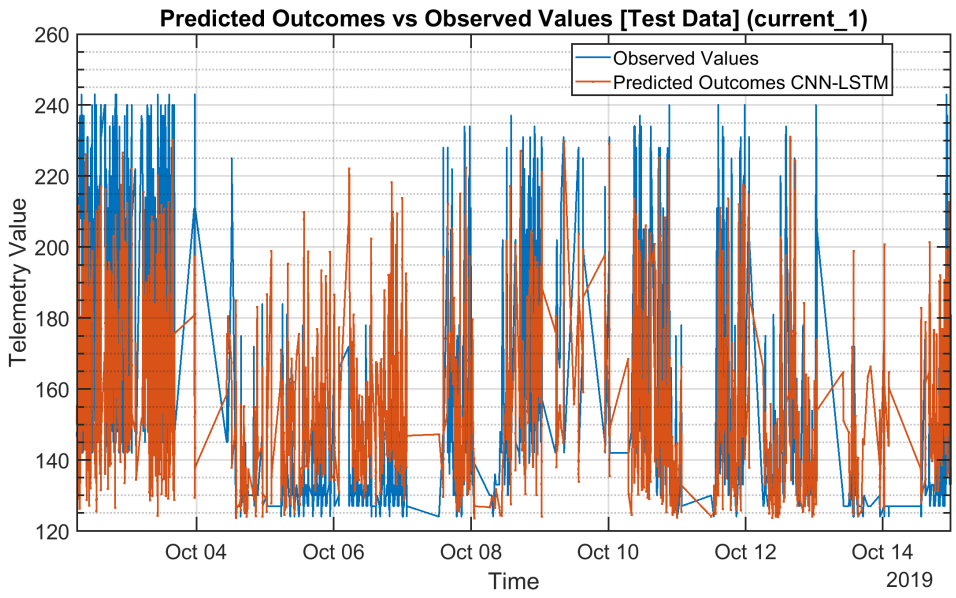


Figure 5.26: Predicted Outcomes v s Observed Values [Test Period 3] (*current_1*) with $vars.Lag = 24\ Hours$

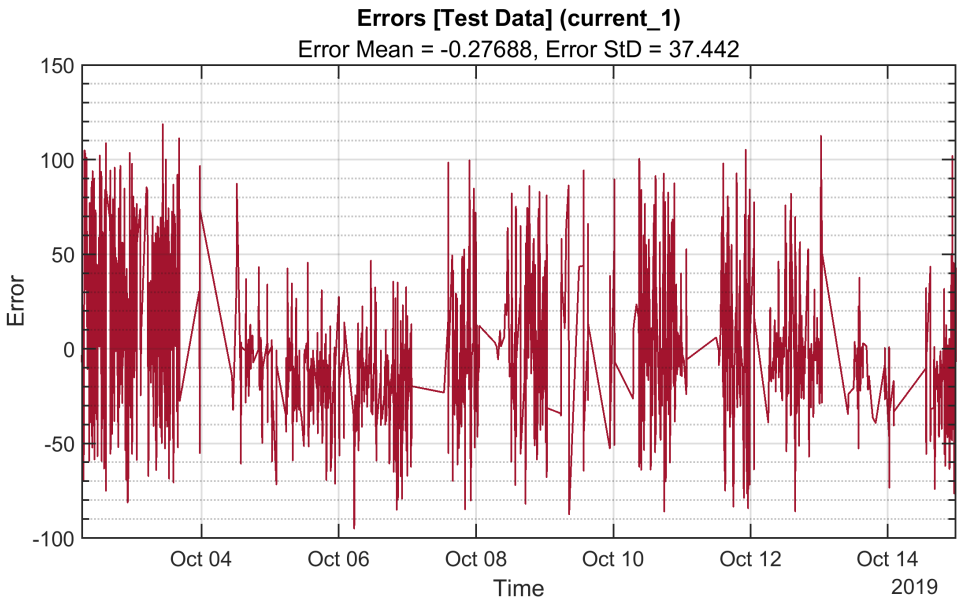


Figure 5.27: Errors [Test Period 3] (*current*_1) with $vars.Lag = 24\ Hours$

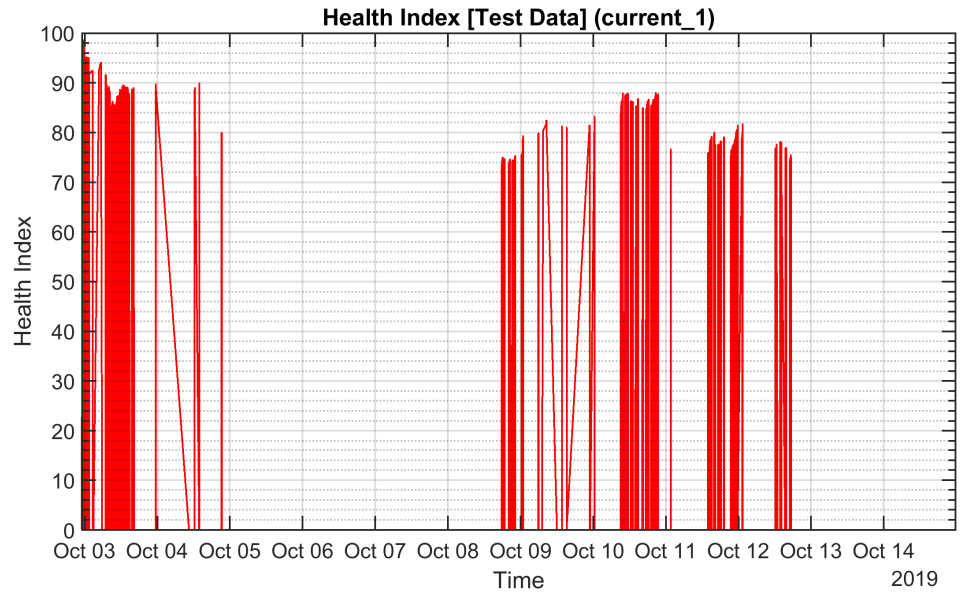


Figure 5.28: Health Index [Test Period 3] (*current*_1) with $vars.Lag = 24\ Hours$

The initial expected behavior is mistakenly identified as abnormal, resulting in a false positive. Additionally, it fails to identify the first period of abnormality, leading to a missed detection or false negative. However, the second anomaly is accurately detected, resulting in a true positive.

Finally, the concluding period of expected behavior is correctly identified as normal, giving a true negative.

5.6.2 CURRENT_2

Test Period 1: from 24-Jun-2019 to 07-Jul-2019

During this period, there is a relatively normal behavior until June 27th. However, two distinct abnormal behaviors occur within the observed period: the first one spans from June 28th to the end of June 30th, while the second one takes place from July 1st to July 7th.

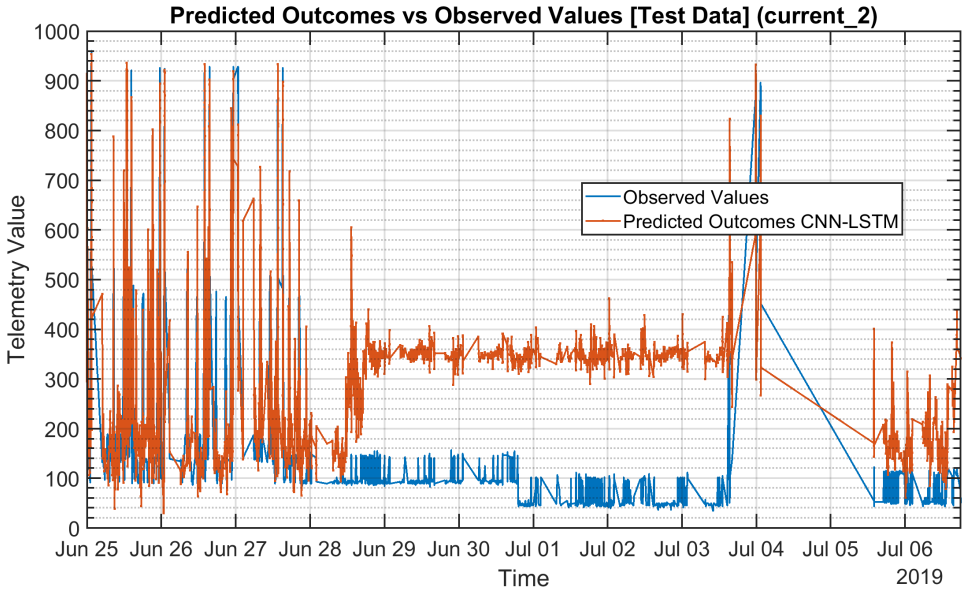


Figure 5.29: Predicted Outcomes v s Observed Values [Test Period 1] (*current_2*) with $vars.Lag = 24\ Hours$

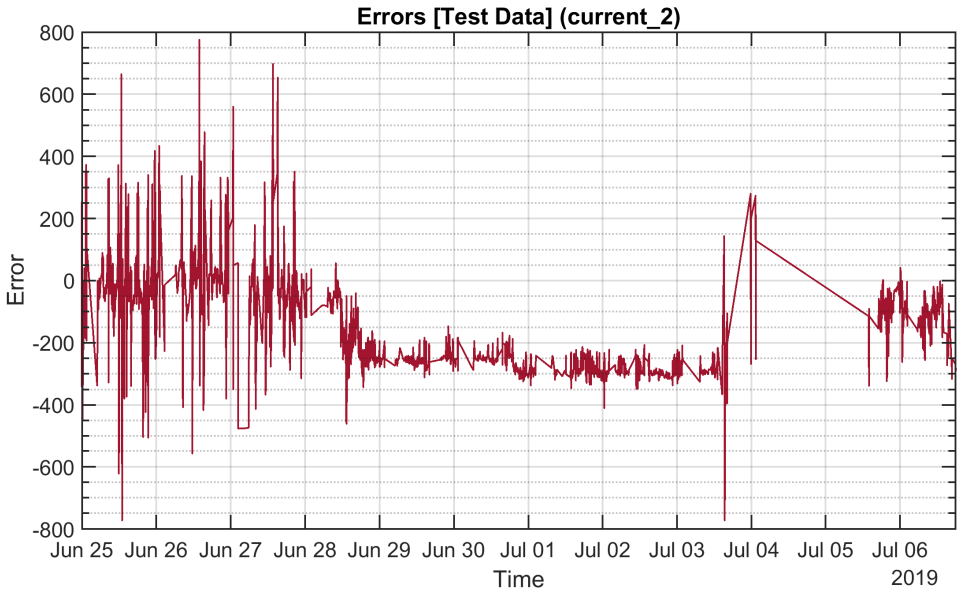


Figure 5.30: Errors [Test Period 1] (*current_2*) with $vars.Lag = 24\ Hours$

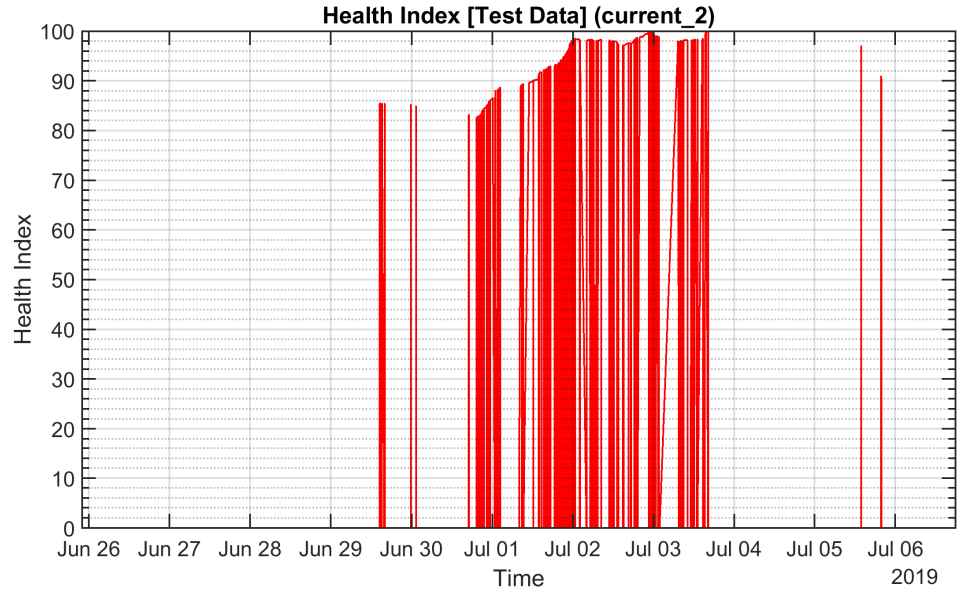


Figure 5.31: Health Index [Test Period 1] (*current_2*) with $vars.Lag = 24\ Hours$

The algorithm accurately identifies normal behavior (true negatives). However, it only detects the first instance of anomalous behavior when the HI rises for the first time. On the other hand, it successfully detects the second anomalous behavior (true positive).

Test Period 2: from 10-Sep-2019 to 24-Sep-2019

During the period leading up to September 19th, the behavior remains within expected norms. However, from September 19th until the conclusion of September 24th, an abnormal pattern of behavior is observed.

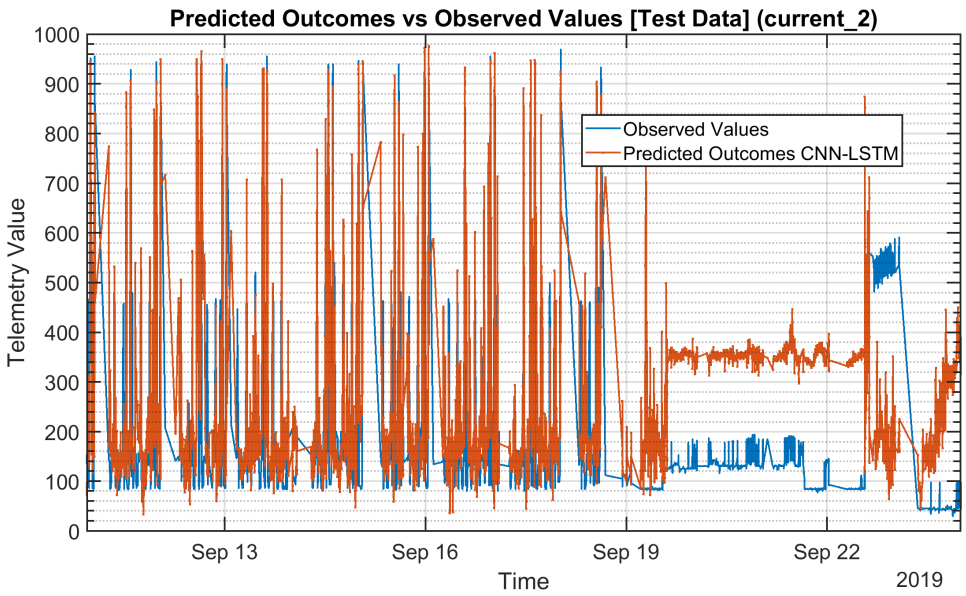


Figure 5.32: Predicted Outcomes v s Observed Values [Test Period 2] (*current_2*) with $vars.Lag = 24\ Hours$

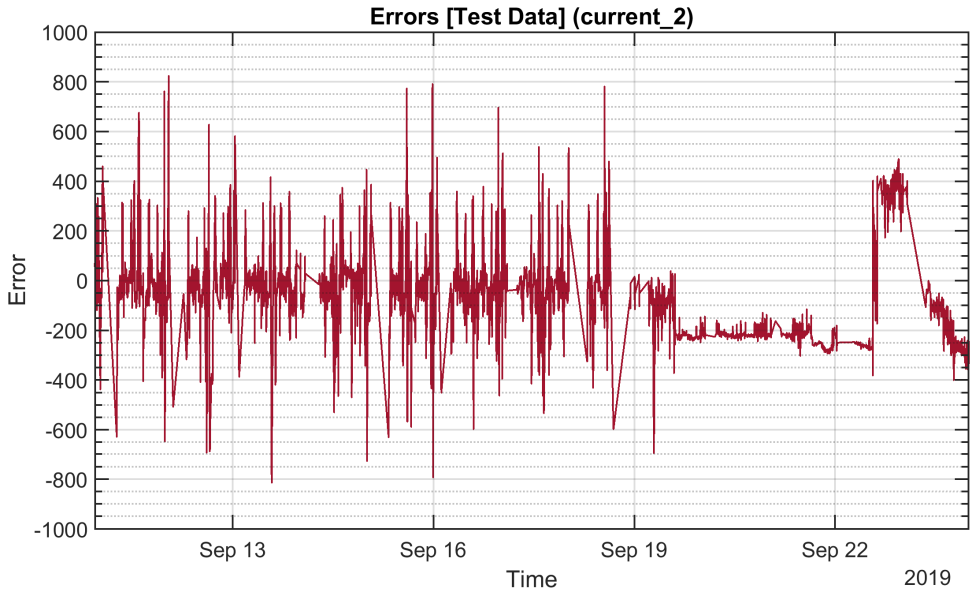


Figure 5.33: Errors [Test Period 2] (*current_2*) with $vars.Lag = 24\ Hours$

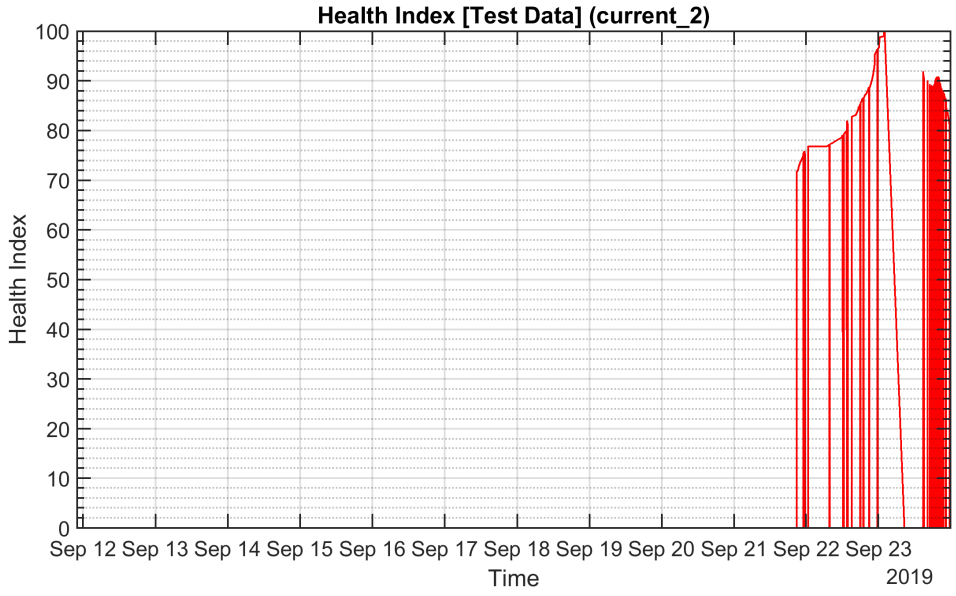


Figure 5.34: Health Index [Test Period 2] (*current_2*) with $vars.Lag = 24\ Hours$

The algorithm effectively identifies normal behavior (true negative), but it detects anomalies three days after they have occurred in the data. There is a delay in raising an alarm until Oc-

tober 22nd, resulting in missed detection. However, from that point onward, the algorithm consistently identifies the events that have occurred (true positive).

Test Period 3: from 01-Oct-2019 to 15-Oct-2019

During the period from October 1st to October 8th, there is an anomaly observed, followed by a return to normal behavior until October 15th.

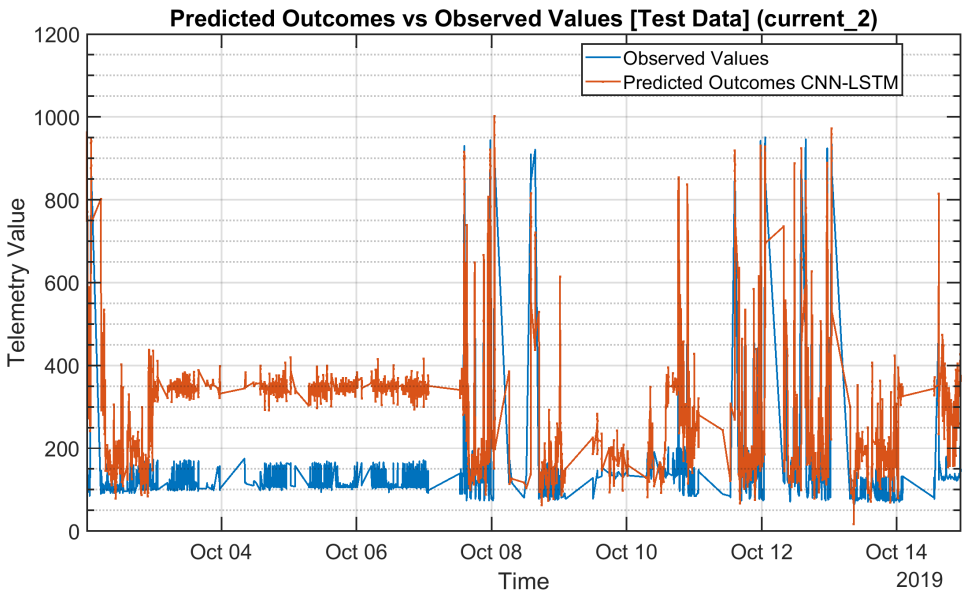


Figure 5.35: Predicted Outcomes v s Observed Values [Test Period 3] (*current_2*) with $vars.Lag = 24\ Hours$

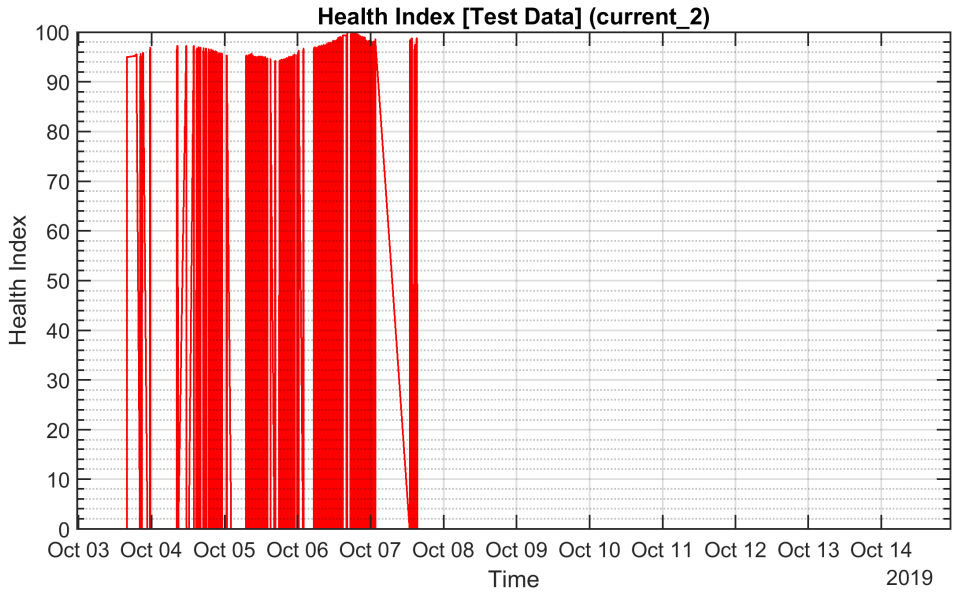


Figure 5.36: Health Index [Test Period 3] (*current_2*) with $vars.Lag = 24\ Hours$

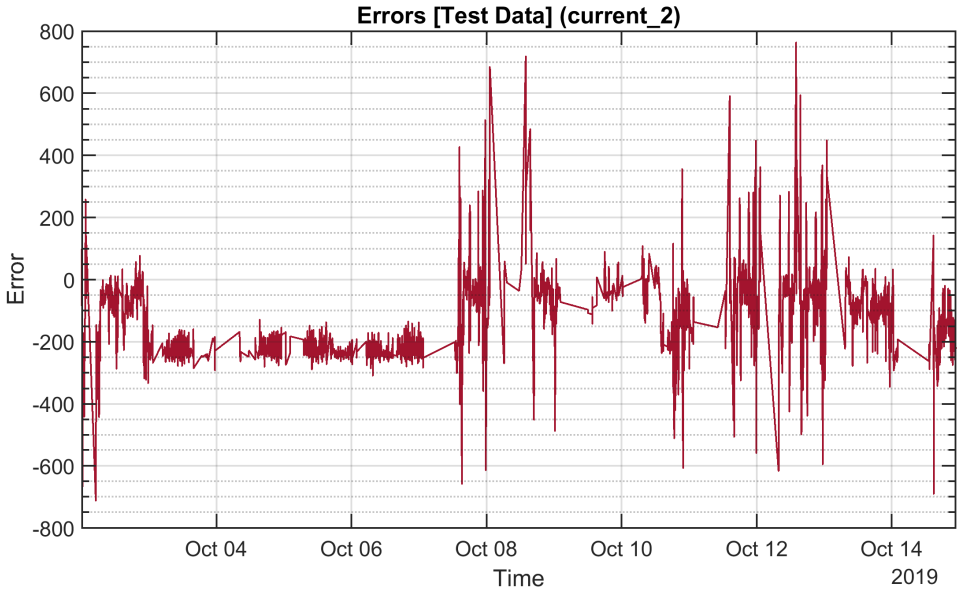


Figure 5.37: Errors [Test Period 3] (*current_2*) with $vars.Lag = 24\ Hours$

The algorithm effectively identifies anomalous behavior (true positives) and accurately detects normal behavior (true negatives). However, there is a noticeable delay in detecting anomalous behavior due to the size of the sliding window.

5.6.3 CURRENT_3

Test Period 1: from 24-Jun-2019 to 07-Jul-2019

During this timeframe, there was nominal behavior observed from June 24th to July 1st, followed by a period of anomalous behavior until July 7th.

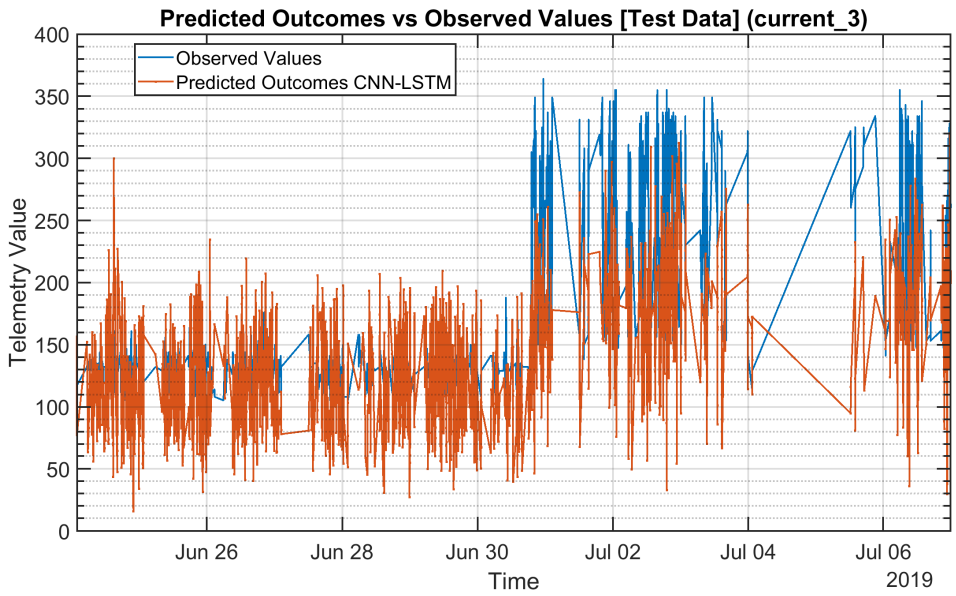


Figure 5.38: Predicted Outcomes v s Observed Values [Test Period 1] (*current_3*) with $vars.Lag = 2\ Hours$

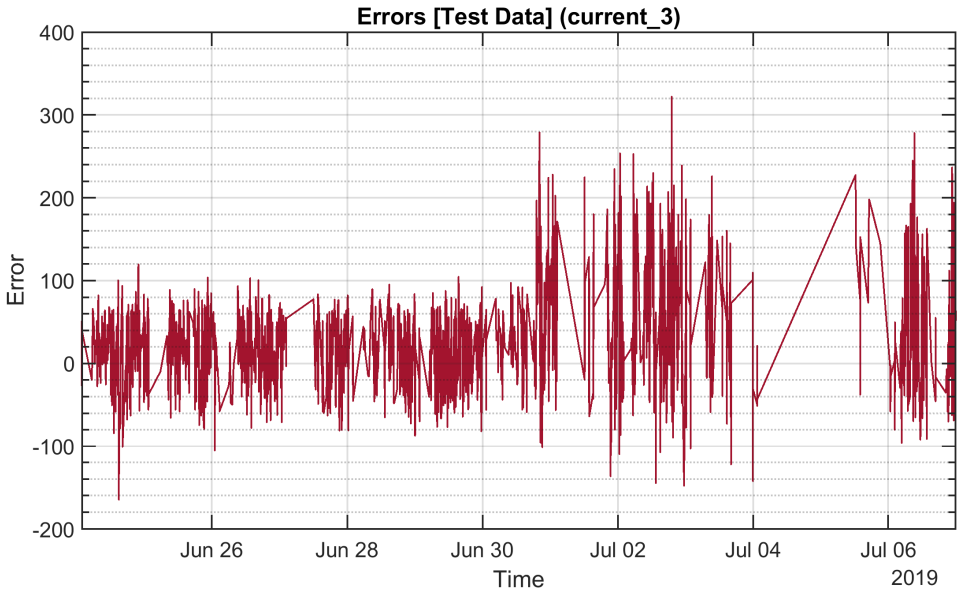


Figure 5.39: Errors [Test Period 1] (*current_3*) with $vars.Lag = 2\ Hours$

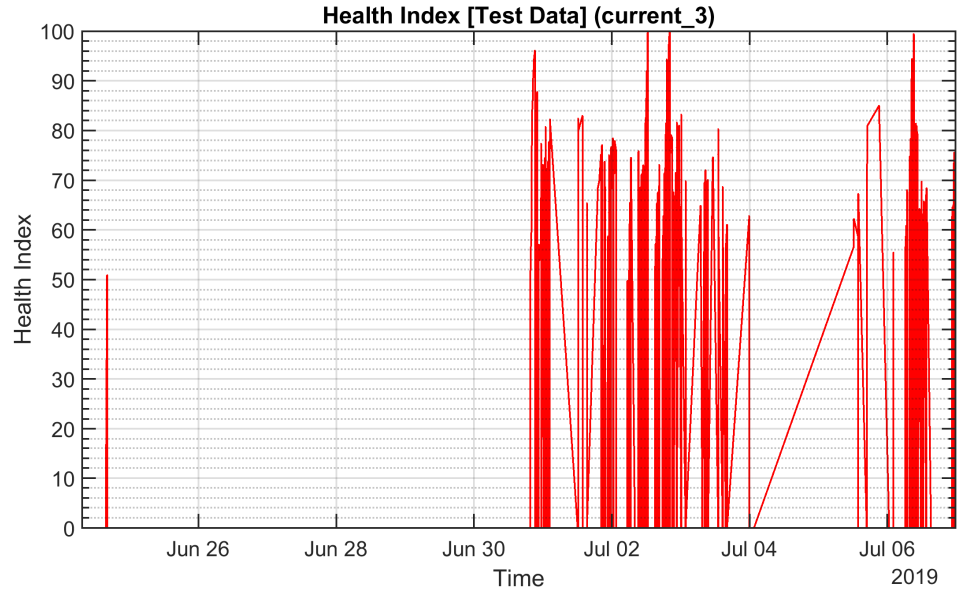


Figure 5.40: Health Index [Test Period 1] (*current_3*) with $vars.Lag = 2\ Hours$

The algorithm excels at accurately identifying normal behavior (true negatives) while also promptly detecting anomalous behavior (true positives).

Test Period 2: from 10-Sep-2019 to 24-Sep-2019

The presented period exclusively encompasses nominal behaviors.

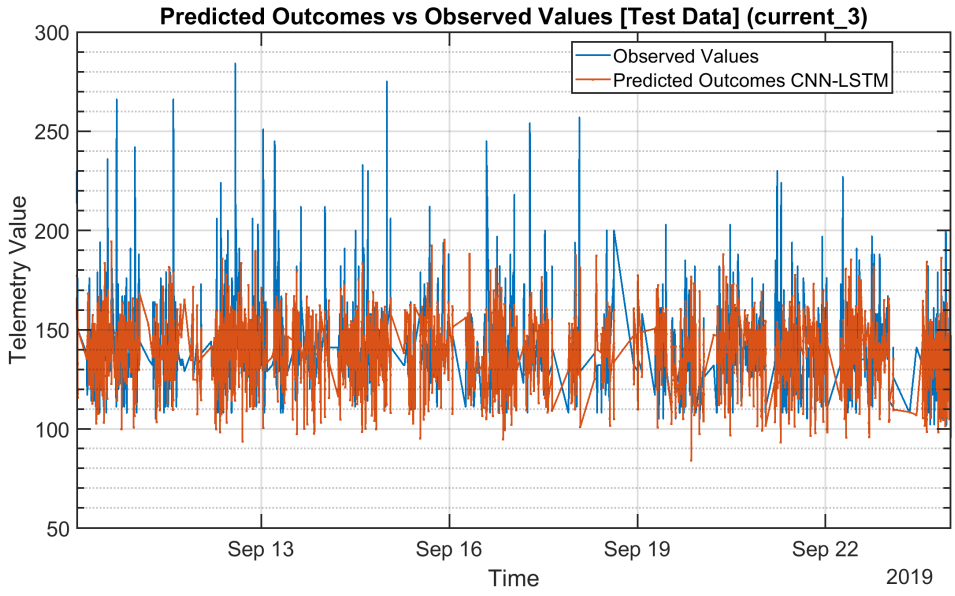


Figure 5.41: Predicted Outcomes v s Observed Values [Test Period 2] (*current_3*) with $vars.Lag = 2\ Hours$

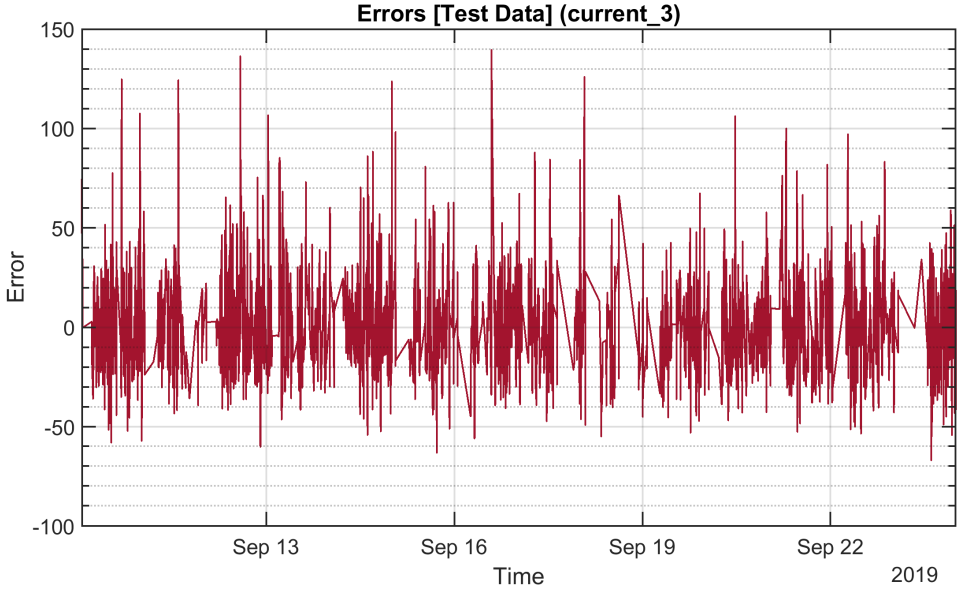


Figure 5.42: Errors [Test Period 2] (*current_3*) with $vars.Lag = 2\ Hours$

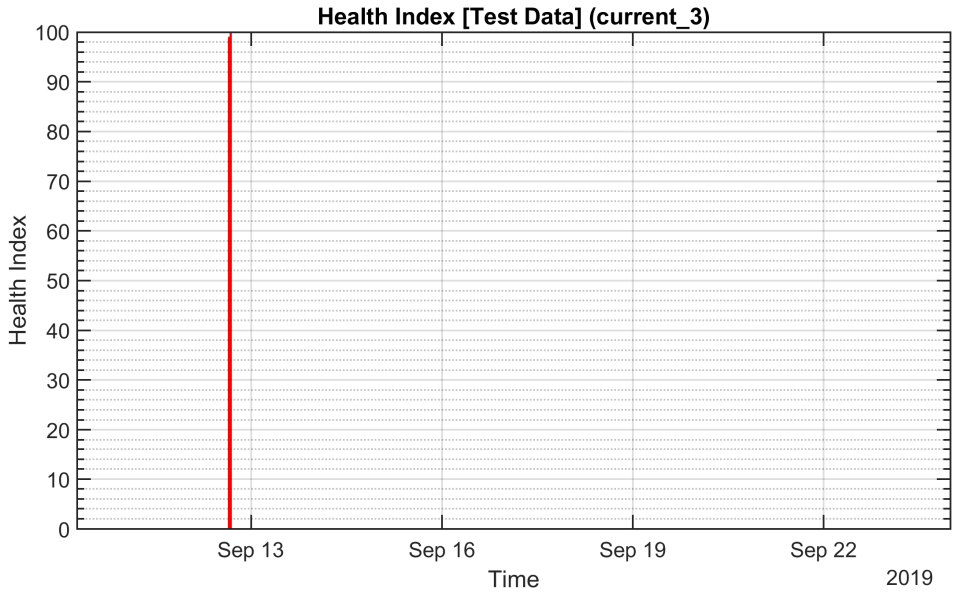


Figure 5.43: Health Index [Test Period 2] (*current_3*) with $vars.Lag = 2\ Hours$

The algorithm successfully identifies all true negatives except for a single momentary peak, but overall its detection accuracy remains intact.

Test Period 3: from 01-Oct-2019 to 15-Oct-2019

The period under consideration primarily comprises nominal behaviors, with the exception of a notable peak around October 13th.

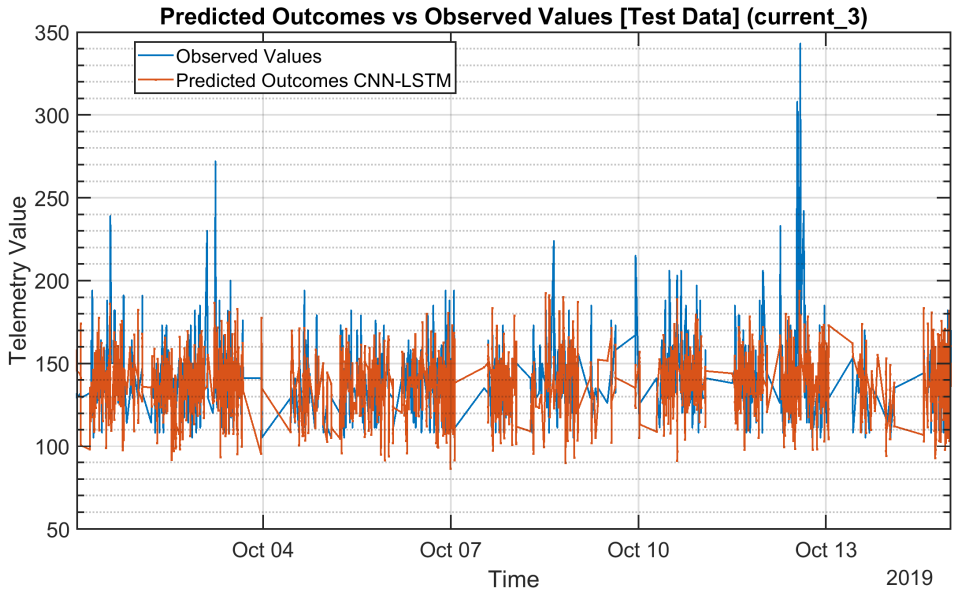


Figure 5.44: Predicted Outcomes v s Observed Values [Test Period 3] (*current_3*) with $vars.Lag = 2\ Hours$

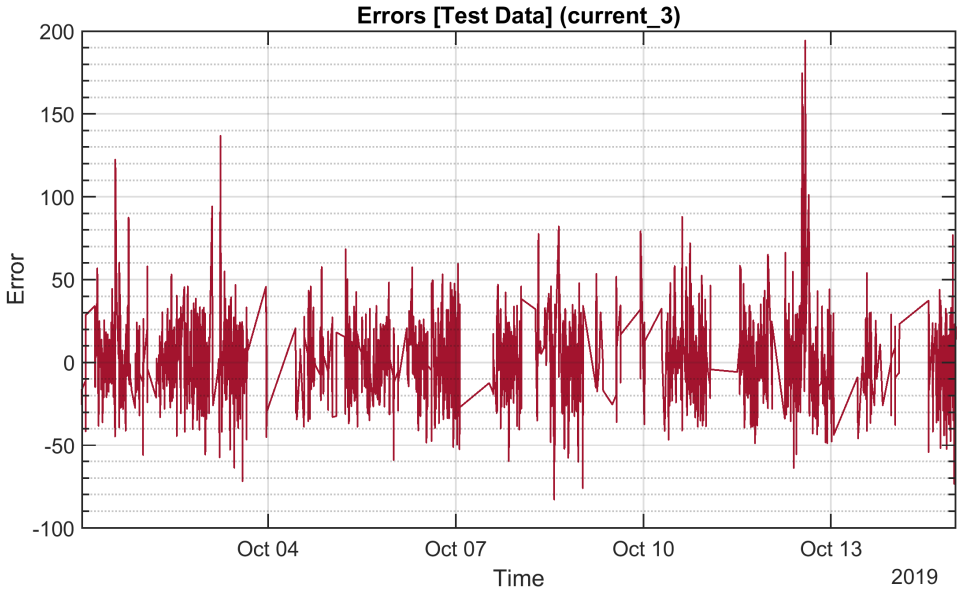


Figure 5.45: Errors [Test Period 3] (*current_3*) with $vars.Lag = 2\ Hours$

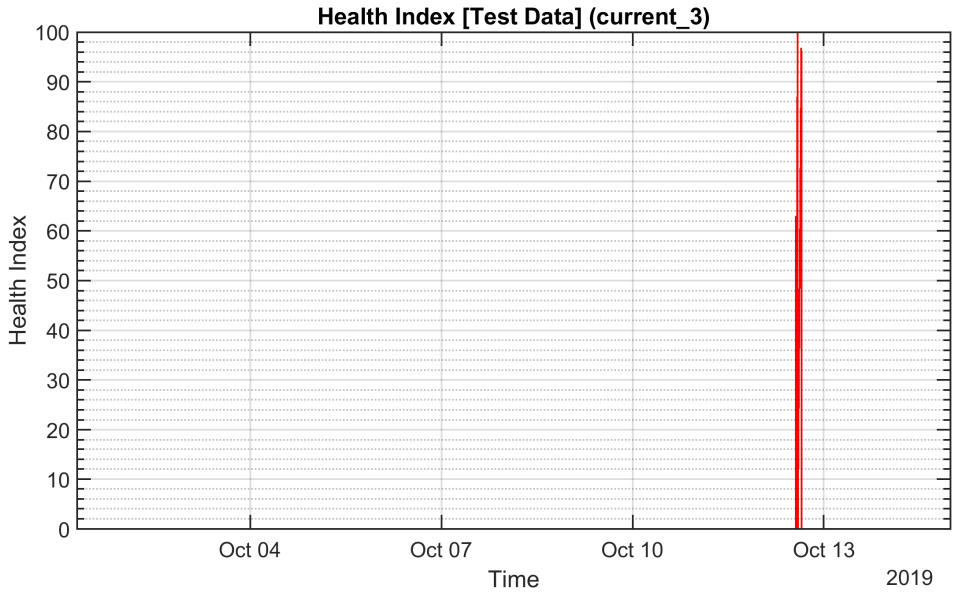


Figure 5.46: Health Index [Test Period 3] (*current_3*) with $vars.Lag = 2\ Hours$

The algorithm effectively identifies all instances of true negatives and even detects an anomaly near the peak on October 13, which qualifies as a true positive.

5.6.4 VOLTAGE_1

Test Period 1: from 24-Jun-2019 to 07-Jul-2019

From June 24th to July 3rd, the period demonstrates typical behavior. However, starting from this point until the conclusion of July 7th, the period deviates from its normal pattern.

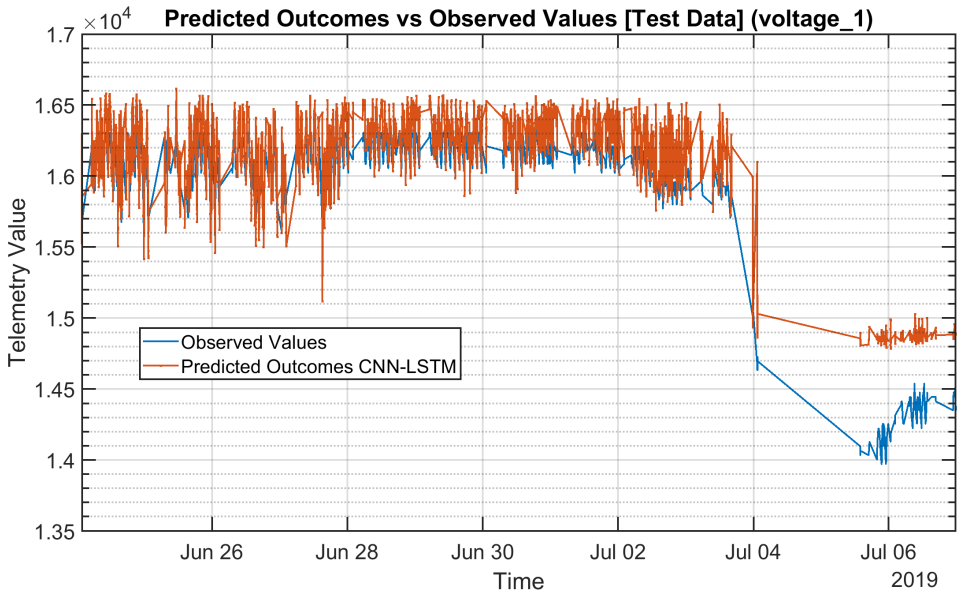


Figure 5.47: Predicted Outcomes v s Observed Values [Test Period 1] ($voltage\_1$) with $vars.Lag = 12\ Hours$

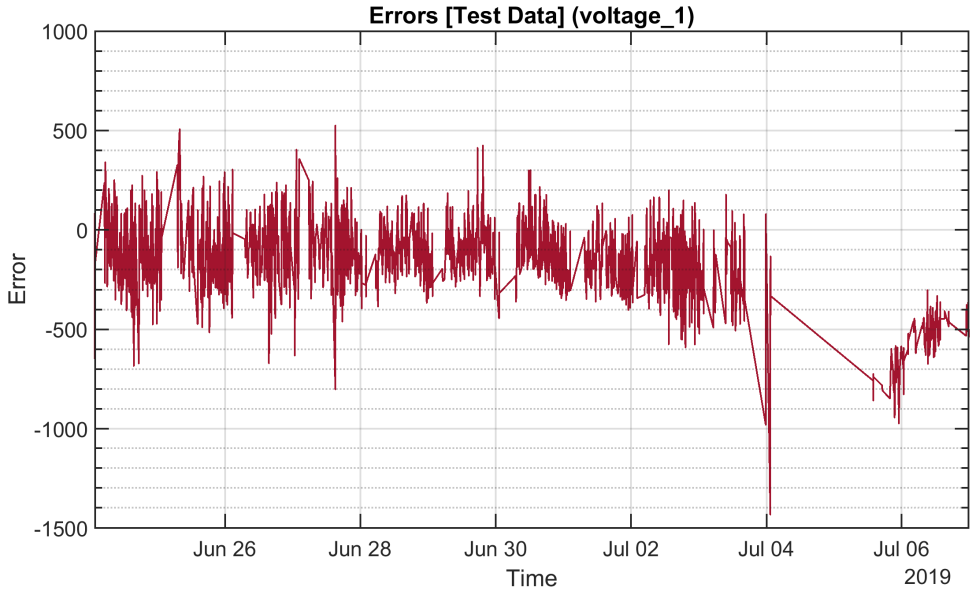


Figure 5.48: Errors [Test Period 1] ($voltage\_1$) with $vars.Lag = 12\ Hours$

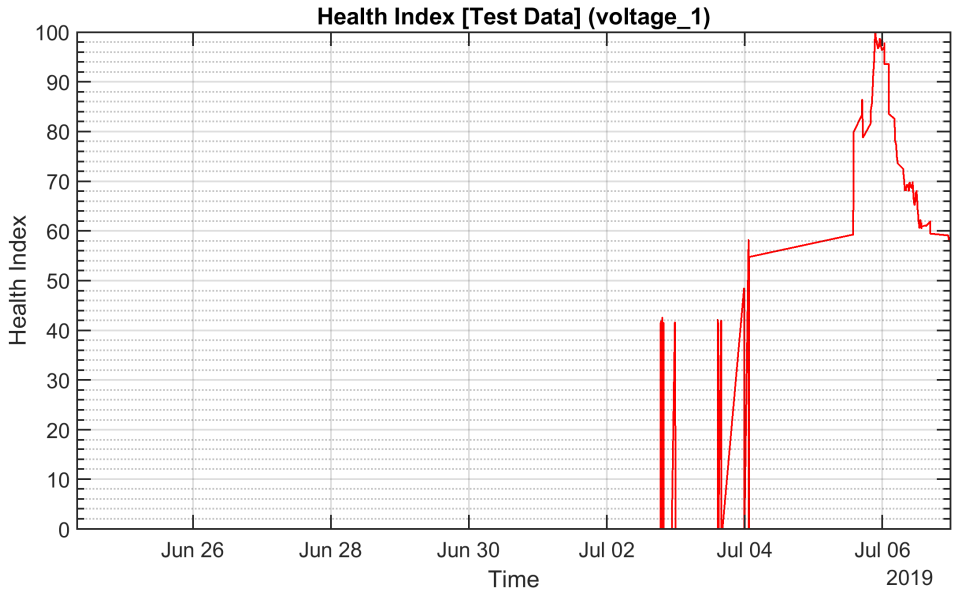


Figure 5.49: Health Index [Test Period 1] (*voltage_1*) with $vars.Lag = 12\ Hours$

The algorithm excels at accurately identifying both the expected behavior (true negatives) and subsequent unusual behavior (true positives).

Test Period 2: from 10-Sep-2019 to 24-Sep-2019

From September 10th to September 23rd, the situation remains normal, but there is a noticeable shift in behavior from September 23rd to September 24th, indicating an anomaly.

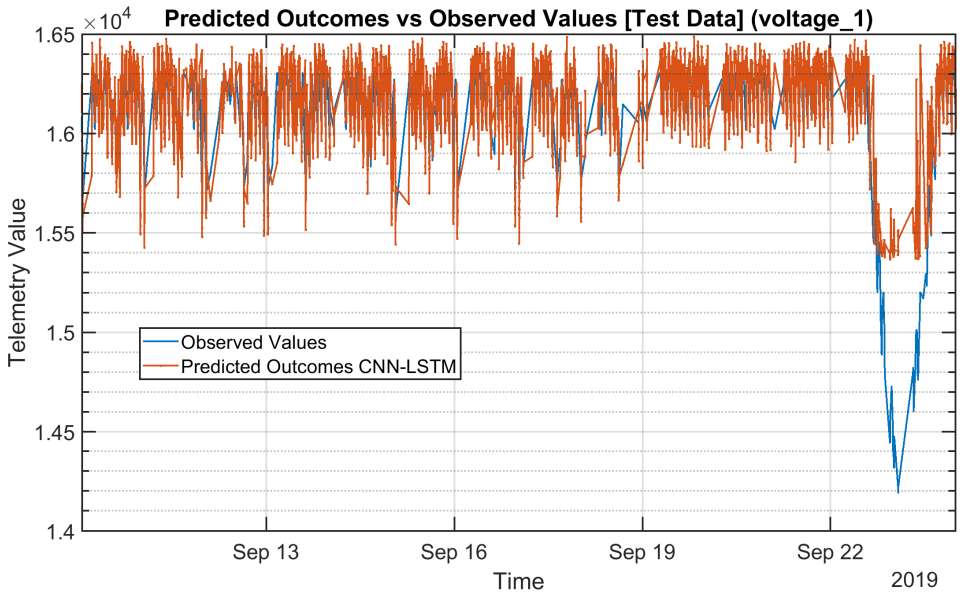


Figure 5.50: Predicted Outcomes v s Observed Values [Test Period 2] (*voltage_1*) with $vars.Lag = 12\ Hours$

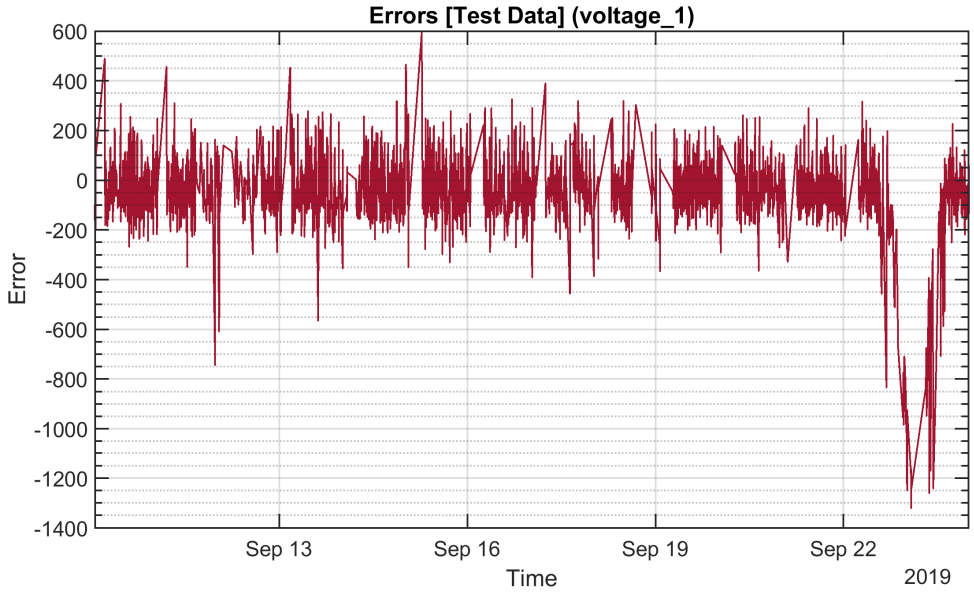


Figure 5.51: Errors [Test Period 2] (*voltage_1*) with $vars.Lag = 12\ Hours$

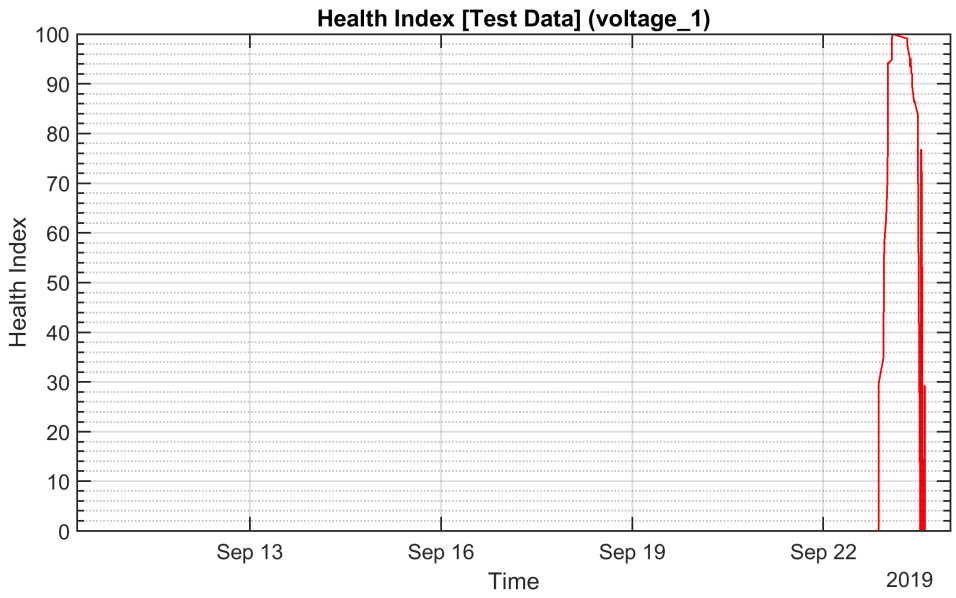


Figure 5.52: Health Index [Test Period 2] (*voltage_1*) with $vars.Lag = 12\ Hours$

The algorithm demonstrates proficiency in accurately detecting both expected normal behavior (true negatives) and subsequent uncommon behavior (true positives).

Test Period 3: from 01-Oct-2019 to 15-Oct-2019

Throughout this period, there is a generally consistent pattern of behavior until October 3rd. However, two noticeable deviations from the norm are observed within this timeframe: the first abnormal behavior lasts from October 3rd until the end of October 8th, and the second abnormal behavior occurs between October 9th and October 12th.

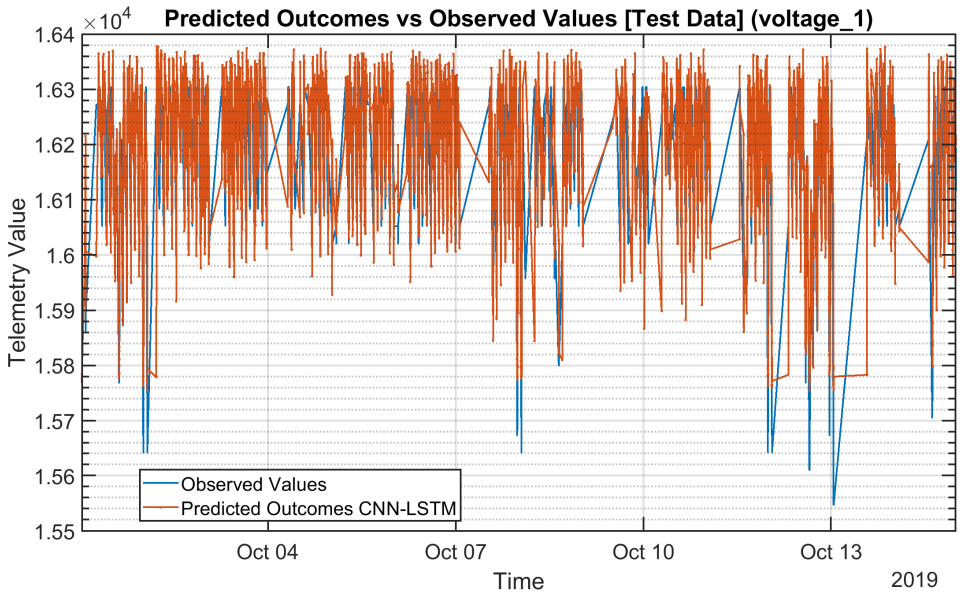


Figure 5.53: Predicted Outcomes v s Observed Values [Test Period 3] ($voltage\_1$) with $vars.Lag = 12\ Hours$

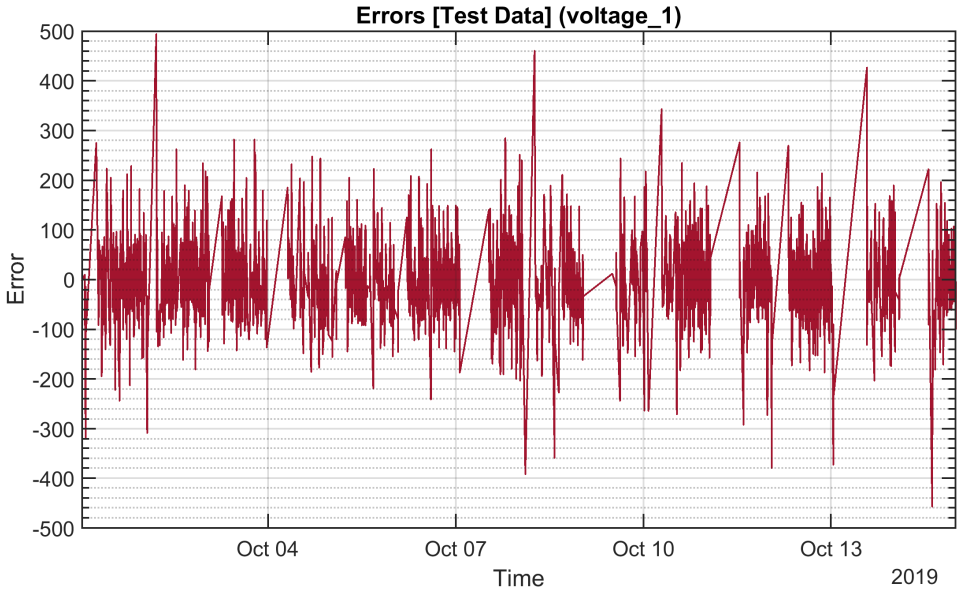


Figure 5.54: Errors [Test Period 3] ($voltage\_1$) with $vars.Lag = 12\ Hours$

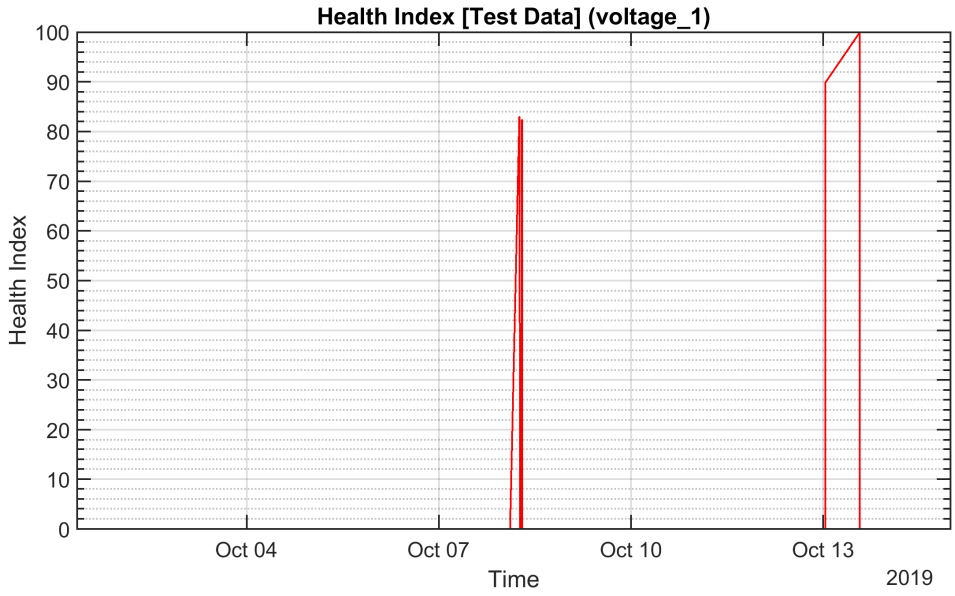


Figure 5.55: Health Index [Test Period 3] (*voltage_1*) with $vars.Lag = 12\ Hours$

The algorithm exhibits shortcomings in detecting anomalies: it initially misses the first anomalous period, resulting in a false negative. Additionally, it mistakenly identifies a false positive shortly before October 9th. Furthermore, for the second anomalous period, the algorithm fails to detect it accurately (true negative) and instead produces false positives towards the conclusion of the anomalous period on October 13th.

5.6.5 VOLTAGE_2

Test Period 1: from 24-Jun-2019 to 07-Jul-2019

The specified period, which extends from June 24th to July 7th, is entirely nominal.

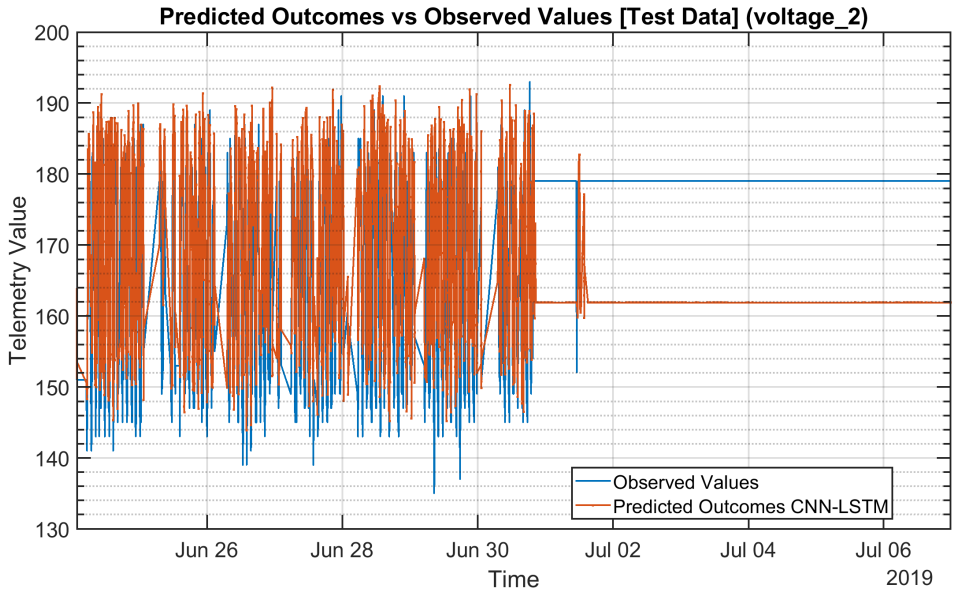


Figure 5.56: Predicted Outcomes v s Observed Values [Test Period 1] ($voltage\_2$) with $vars.Lag = 2\ Hours$

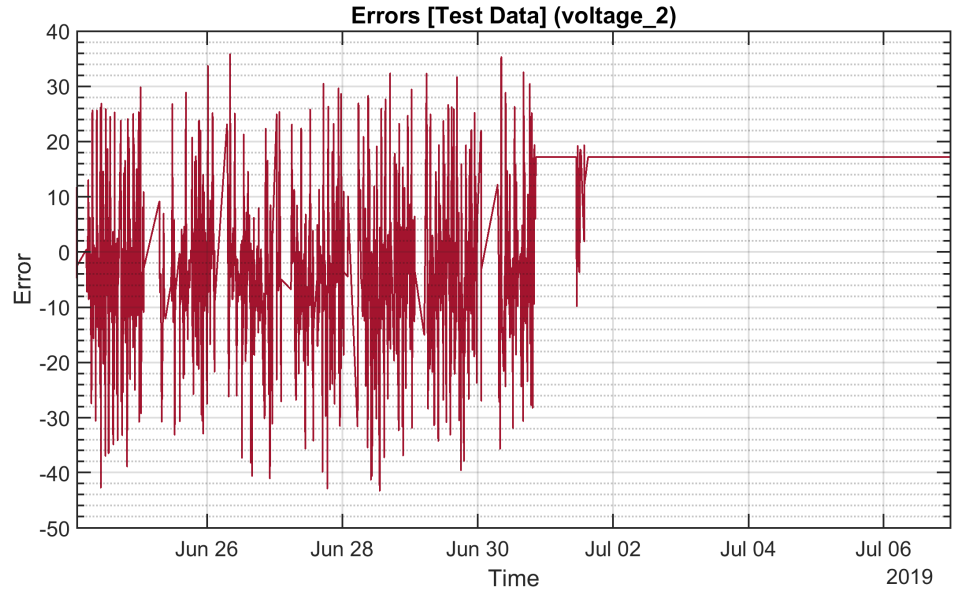


Figure 5.57: Errors [Test Period 1] ($voltage\_2$) with $vars.Lag = 2\ Hours$

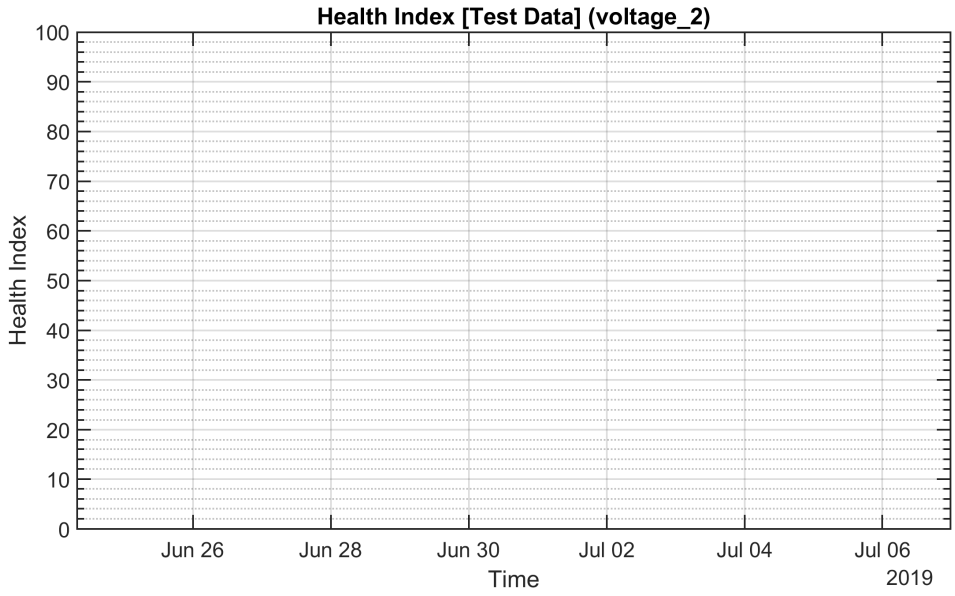


Figure 5.58: Health Index [Test Period 1] (*voltage_2*) with $vars.Lag = 2\ Hours$

The algorithm effectively detects a regular pattern, resulting in a true negative classification.

Test Period 2: from 10-Sep-2019 to 24-Sep-2019

No anomalies are observed during the specified period, which spans from September 10th to September 24th.

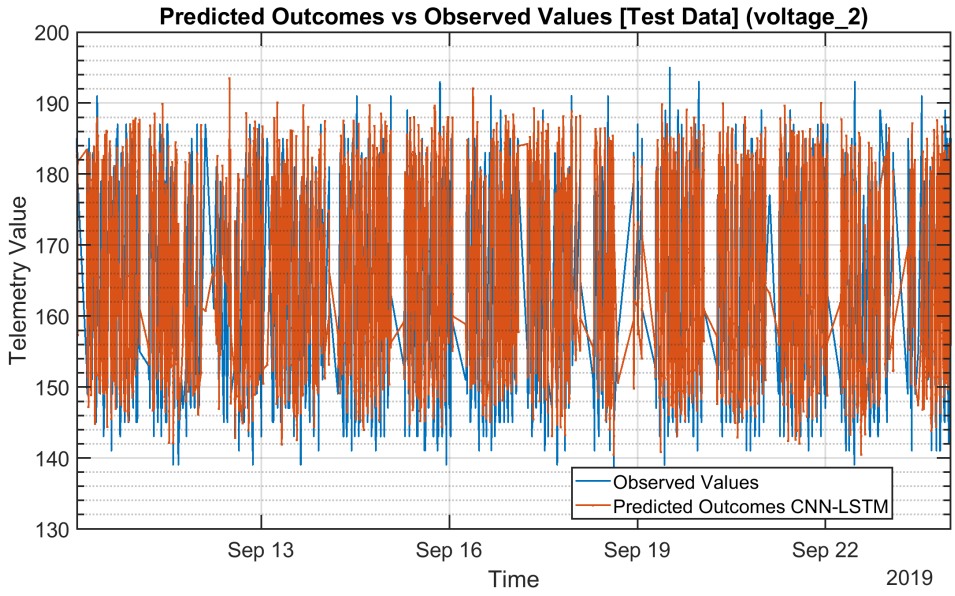


Figure 5.59: Predicted Outcomes v s Observed Values [Test Period 2] ($voltage\_2$) with $vars.Lag = 2\ Hours$

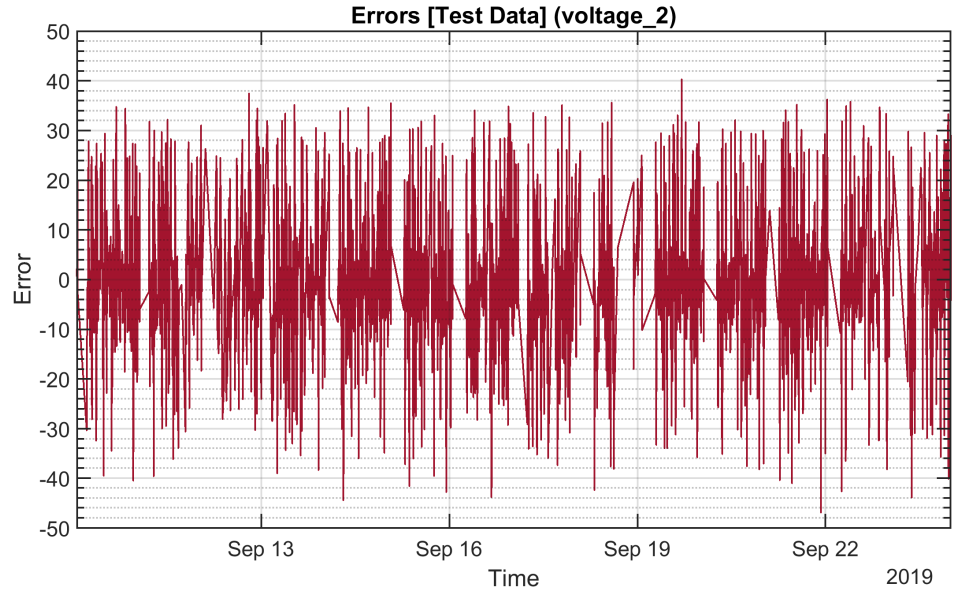


Figure 5.60: Errors [Test Period 2] ($voltage\_2$) with $vars.Lag = 2\ Hours$

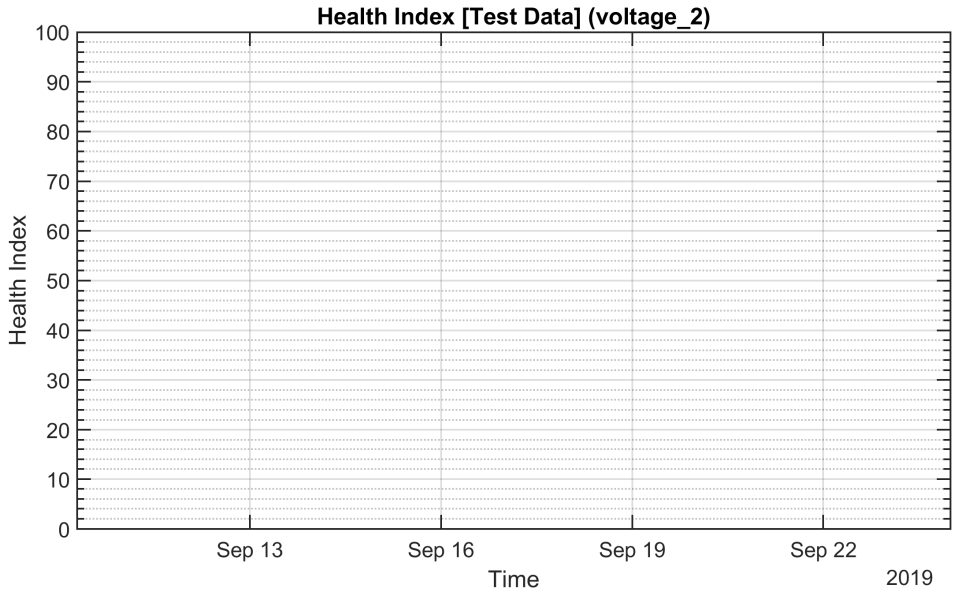


Figure 5.61: Health Index [Test Period 2] (*voltage_2*) with $vars.Lag = 2\ Hours$

The algorithm effectively detects a regular pattern, resulting in a true negative classification.

Test Period 3: from 01-Oct-2019 to 15-Oct-2019

During the designated period of June 1st to October 15th, normal behavior is observed.

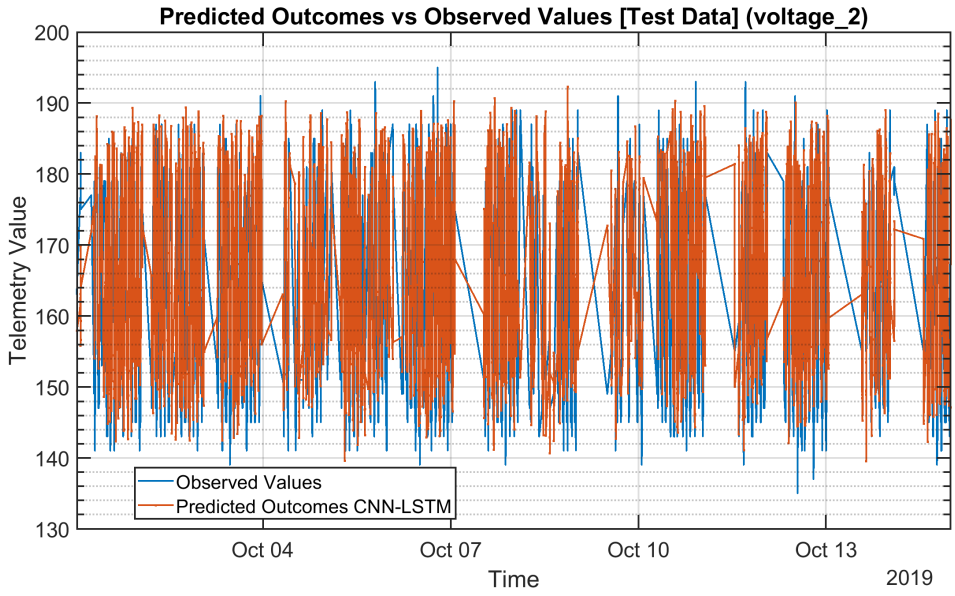


Figure 5.62: Predicted Outcomes v s Observed Values [Test Period 3] ($voltage\_2$) with $vars.Lag = 2\ Hours$

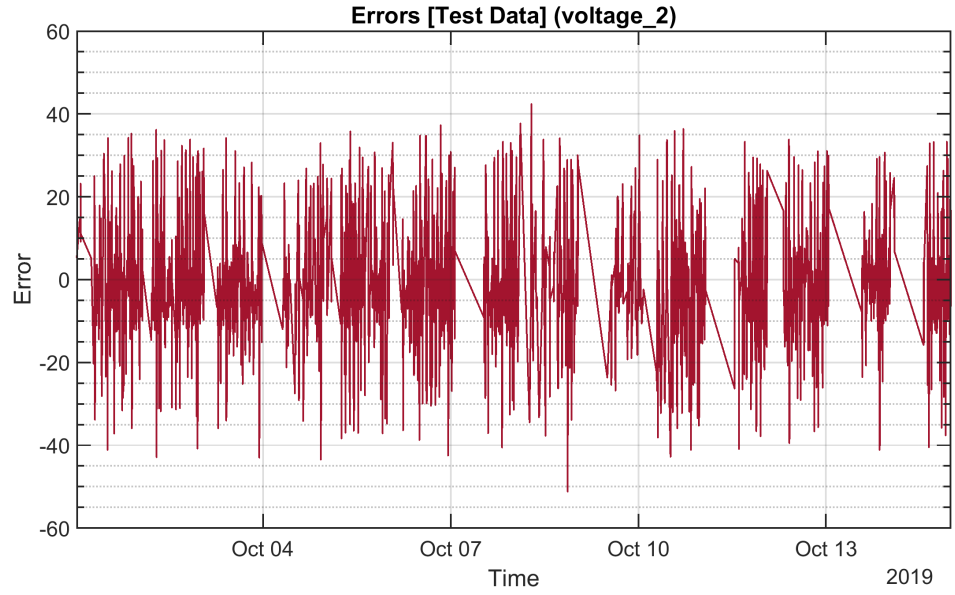


Figure 5.63: Errors [Test Period 3] ($voltage\_2$) with $vars.Lag = 2\ Hours$

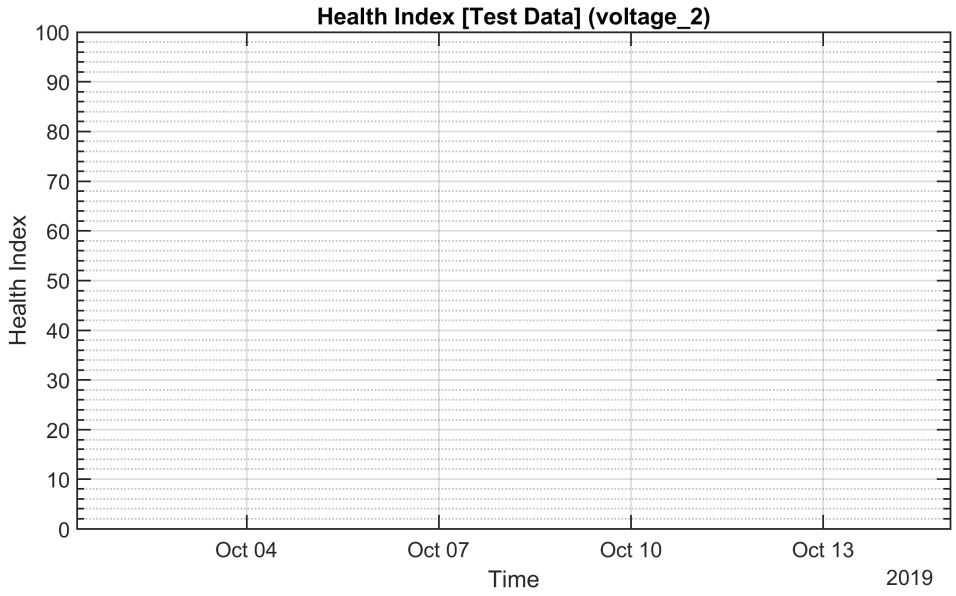


Figure 5.64: Health Index [Test Period 3] (*voltage_2*) with $vars.Lag = 2\ Hours$

The algorithm effectively detects a regular pattern, resulting in a true negative classification.

6 Conclusions

6.1 Research Overview

In this section, an overview of the research conducted in this thesis is presented. The research focuses on exploring and analyzing AI and ML-based diagnostic methods for small satellites, with a specific emphasis on enhancing the reliability and efficiency of satellite operations. The goal is to address the diagnostic challenges specific to small satellites by investigating various techniques and approaches available in the field.

To achieve this, the research involves examining historical data and conducting extensive research to determine the most critical subsystems and key components within those subsystems that significantly affect the overall performance and reliability of small satellites. By identifying these critical elements, the research aims to develop effective diagnostic methods that can detect and mitigate potential failures.

Automation plays a key role in satellite diagnostics, and the research emphasizes the importance of automating the diagnostic process using AI and ML techniques. By leveraging these technologies, higher efficiency, accuracy, and reliability can be achieved compared to manual processes. The research explores the application of AI and ML algorithms in analyzing satellite data, identifying patterns, and making accurate predictions to improve the diagnostic capabilities of small satellites.

Additionally, the research investigates the use of machine learning algorithms for anomaly

detection, fault classification, and health monitoring of small satellites. By training models on historical data and real-time telemetry, the research aims to develop intelligent systems capable of identifying and diagnosing anomalies or faults in the satellite's operation. This could enable proactive maintenance and timely interventions to prevent potential failures and optimize satellite performance.

The research also considers the implications of the findings and their potential impact on the field of small satellite operations. By developing reliable and efficient diagnostic methods, the research aims to contribute to the advancement of small satellite technology, enabling more robust and reliable space missions.

6.2 Summary of Findings

In this section, a summary of the key findings is presented from research conducted on AI and ML-based diagnostic methods for small satellites. The findings provide valuable insights into the effectiveness and applicability of these methods in improving the reliability and efficiency of satellite operations.

1. **Identification of Critical Subsystems:** By examining historical data and conducting extensive research, the identification of the most critical subsystems in small satellites has been accomplished. These subsystems hold significant importance in determining the overall performance and reliability of the satellite. By directing attention to these critical subsystems, diagnostic efforts can be prioritized, and resources can be allocated appropriately for maintenance and troubleshooting purposes.

2. **Key Components Affecting Performance:** In the identified critical subsystems, key components have been identified that play a significant role in the performance and reliability of small satellites. Recognizing the importance of these components, targeted diagnostic methods can be developed to monitor their health and detect potential failures.

3. **Automation for Improved Efficiency:** The research underscores the significance of automation in satellite diagnostics. By implementing AI and ML techniques to automate the diagnostic process, higher efficiency can be achieved in analyzing large volumes of satellite data. This automation facilitates real-time monitoring and enables early anomaly detection.

4. **AI and ML Algorithms for Predictive Analysis:** Extensive research has demonstrated the effectiveness of AI and ML algorithms in analyzing satellite data, identifying patterns, and making accurate predictions. These algorithms have demonstrated signifi-

cant potential in accurately predicting small satellite component behavior, and enabling proactive maintenance strategies. As a result, they contribute to minimizing the risk of failures and enhancing overall system reliability.

5. **Anomaly Detection and Fault Classification:** Machine learning algorithms have proven to be effective in detecting anomalies and classifying faults in small satellites. By training models on historical data and telemetry, deviations from normal behavior can be detected, enabling efficient troubleshooting and corrective actions.

6. **Health Index Calculation:** In addition to anomaly detection and fault classification, machine learning algorithms have been leveraged to calculate the Health Index of small satellite components. By analyzing various performance metrics and telemetry data, the Health Index provides a quantitative measure of the component's condition and overall operational health. This calculation aids in assessing the component's reliability, predicting potential failures, and optimizing maintenance strategies.

These findings lay the foundation for future advancements in the field and provide valuable insights for satellite operators, researchers, and engineers involved in small satellite missions.

6.3 Discussions of Results

For the telemetry *current*_1, the algorithm initially fails to identify the first abnormal period due to disregarding insignificant deviations, resulting in a missed detection (false negative). However, it accurately detects the second anomaly, demonstrating a true positive.

In the case of *current*_2, the algorithm shows mixed performance. It accurately identifies normal behavior (true negatives) but exhibits false positives by incorrectly labeling normal behavior as anomalous. It also experiences a delay in detecting the initial period of anomalies, leading to missed detection (false negative). However, it successfully detects the second anomaly (true positive).

For *current*_3, the algorithm performs well, accurately identifying both normal behavior (true negatives) and anomalous behavior (true positives).

In the telemetry *voltage*_1, the algorithm excels at accurately identifying both expected behavior (true negatives) and subsequent anomalies (true positives). However, it misses the first anomalous period, resulting in a false negative. It also produces false positives shortly before October 9th and fails to detect the second anomaly accurately, generating false positives towards the end of the anomalous period on October 13th.

Lastly, for *voltage*_2 the algorithm effectively detects the regular pattern, resulting in a true negative classification in all three test periods.

Overall, the performance of the algorithm and network varies depending on the telemetry and test period. While they demonstrate successful detection in some cases, there are instances of missed detections, false positives, and delays in anomaly identification. These findings highlight the need for further refinement and optimization of the algorithm to enhance its performance in all scenarios.

Table 6.1 summarizes the performance of the algorithm and network for each telemetry in the three test periods. The results indicate varying degrees of success and challenges in anomaly detection.

	Test Period 1	**Test Period 2**	**Test Period 3**
current_1	The algorithm fails to identify the initial abnormal period because it disregards the insignificant deviations. This can be characterized as a missed detection, or false negative. On the other hand, the algorithm accurately identifies the second anomaly, resulting in a true positive.	The algorithm's performance in this particular case is not meeting expectations. Surprisingly, it is incorrectly labeling normal behavior as anomalous, resulting in false positives. Additionally, it fails to identify the initial period of anomalies because it disregards insignificant deviations, leading to a missed detection or false negative. However, the algorithm accurately detects the second anomaly, which can be classified as a true positive.	The initial expected behavior is mistakenly identified as abnormal, resulting in a false positive. Additionally, it fails to identify the first period of abnormality, leading to a missed detection or false negative. However, the second anomaly is accurately detected, resulting in a true positive. Finally, the concluding period of expected behavior is correctly identified as normal, giving a true negative.

	Test Period 1	Test Period 2	Test Period 3
current_2	The algorithm accurately identifies normal behavior (true negatives). However, it only detects the first instance of anomalous behavior when the HI rises for the first time. On the other hand, it successfully detects the second anomalous behavior (true positive).	The algorithm effectively identifies normal behavior (true negative), but it detects anomalies three days after they have occurred in the data. There is a delay in raising an alarm until October 22nd, resulting in missed detection. However, from that point onward, the algorithm consistently identifies the events that have occurred (true positive).	The algorithm effectively identifies anomalous behavior (true positives) and accurately detects normal behavior (true negatives). However, there is a noticeable delay in detecting anomalous behavior due to the size of the sliding window.
current_3	The algorithm excels at accurately identifying normal behavior (true negatives) while also promptly detecting anomalous behavior (true positives).	The algorithm successfully identifies all true negatives except for a single momentary peak, but overall its detection accuracy remains intact.	The algorithm effectively identifies all instances of true negatives and even detects an anomaly near the peak on October 13, which qualifies as a true positive.

	Test Period 1	Test Period 2	Test Period 3
voltage_1	The algorithm excels at accurately identifying both the expected behavior (true negatives) and subsequent unusual behavior (true positives).	The algorithm demonstrates proficiency in accurately detecting both expected normal behavior (true negatives) and subsequent uncommon behavior (true positives).	The algorithm exhibits shortcomings in detecting anomalies: it initially misses the first anomalous period, resulting in a false negative. Additionally, it mistakenly identifies a false positive shortly before October 9th. Furthermore, for the second anomalous period, the algorithm fails to detect it accurately (true negative) and instead produces false positives towards the conclusion of the anomalous period on October 13th.
voltage_2	The algorithm effectively detects a regular pattern, resulting in a true negative classification.	The algorithm effectively detects a regular pattern, resulting in a true negative classification.	The algorithm effectively detects a regular pattern, resulting in a true negative classification.

Table 6.1: Summary of Algorithm Performance for Each Telemetry and Test Period.

6.4 Recommendations for Future Work

In the following section, potential avenues for future research and development in the field of DPHM are discussed. Additionally, areas for improvement and enhancement of the current

system are identified, along with opportunities for expanding the scope of this study. These recommendations aim to guide future researchers and practitioners in advancing the field of DPHM and further enhancing the capabilities of such systems.

- **Parameter Tuning and Architecture Optimization:** One potential area for future work is the implementation of a systematic approach for tuning the parameters of the training options and optimizing the architecture of the network. This can involve techniques such as Bayesian optimization, which allows for automatic tuning of hyperparameters to maximize the performance of the network. By fine-tuning the training options and exploring different network architectures, researchers can further optimize the DPHM system and potentially improve its predictive accuracy and efficiency.
- **Multivariate Analysis and Telemetry Integration:** Another promising direction for future research is to extend the network's capabilities to handle multivariate time series data. Currently, the DPHM system focuses on forecasting individual telemetry values. However, in practical scenarios, multiple telemetry time series may exhibit interdependencies and can provide valuable insights when analyzed collectively. By incorporating multivariate analysis techniques, researchers can explore how the behavior and performance of the network change when considering multiple telemetry variables simultaneously. This can potentially lead to a more comprehensive understanding of the system's health and enable more accurate predictions and diagnostics.
- **Comparative Analysis with Existing Systems:** An interesting avenue for future research is to perform a comparative analysis of the developed DPHM system with existing systems or methodologies in the field of satellite diagnostic and prognostic health monitoring. This comparative study can involve benchmarking the performance of the proposed system against established techniques or commercially available systems. By conducting such a comparison, researchers can assess the strengths and weaknesses of different approaches, identify areas of improvement, and gain insights into the unique contributions and advantages of the developed DPHM system. This analysis can help validate the effectiveness and competitiveness of the proposed system and provide valuable guidance for its practical implementation and deployment.

A

Appendix A

Listing A.1: MATLAB Script for Importing Data from CSV Files in a Folder

```
vars = {}; data = {};

[vars, data] = load_data_1(vars);

function [vars, data] = load_data_1(vars)
    % Ask the user to enter the number of telemetries
    vars.numOfTelemetry = input("Enter the Number of Telemetries: ");
    data.data = [];

    for i = 1:vars.numOfTelemetry
        [chosenfile, chosendirectory] = uigetfile({'*.csv*', 'All
            Files (*.csv*)'}, 'Select files to import', 'MultiSelect',
            'on');
        filePath = [chosendirectory chosenfile];

        if filePath ~= 0
            data.DataFileName = chosenfile;
            data.CompleteData{i} = readtable(filePath);
```

```
17          data.seriesdataHeader{i} = data.CompleteData{i}.Properties
                .VariableNames(1,:);
18          data.seriesdata{i} = table2array(data.CompleteData{i});
19          data.seriesdata{i} = transpose(data.seriesdata{i});
20          data.time{i} = data.seriesdata{i}(1,:);
21          data.datetime{i} = datetime(data.time{i}, 'ConvertFrom', '
                posixtime');
22      end
23
24      allfeatures{i} = data.seriesdataHeader{i};
25
26      % Display the list of features
27      disp('List of Telemetry:');
28      for j = 1:length(allfeatures{i})
29          disp(sprintf('%d. %s\n', j, allfeatures{i}{j}));
30      end
31
32      % Ask the user to select telemetry
33      vars.features{i} = input("Enter the Telemetry Number to be
            Selected: ");
34
35      % Display the selected telemetry
36      disp(sprintf('You selected the following telemetry from .csv %
            d: %s\n', i, allfeatures{i}{vars.features{i}}));
37      data.telemetry{i} = allfeatures{i}{vars.features{i}};
38
39      % Create a new array with only the selected telemetry
40      data.seriesdataWfeatures{i} = data.seriesdata{i}(vars.features
            {i},:);
41
42      % Find the indices of NaN values in the array
43      nanIndices = isnan(data.seriesdataWfeatures{i});
44
45      % Remove the NaN values from the array
```

```
        data.seriesdataWfeatures{i} = data.seriesdataWfeatures{i}(~
            nanIndices);
        data.time{i} = data.time{i}(~nanIndices);
        data.datetime{i} = data.datetime{i}(~nanIndices);
        data.data(i,:) = data.seriesdataWfeatures{i};
        data.seriesdataWfeatures{i} = transpose(data.
            seriesdataWfeatures{i});
    end

    fprintf('************************************************\n');
    fprintf('You selected the following telemetry:\n');
    for j = 1:vars.numOfTelemetry
        fprintf('%d. %s\n', j, data.telemetry{j});
    end
    fprintf('************************************************\n');
end
```

Listing A.2: MATLAB Script for Data Splitting into Train and Test Periods

```
vars.dateTrainStart = datetime('01-Jun-2019');
vars.dateTrainStop  = datetime('23-Jun-2019');

vars.dateTest = [
    datetime('24-Jun-2019'), datetime('07-Jul-2019');
    datetime('10-Sep-2019'), datetime('24-Sep-2019');
    datetime('01-Oct-2019'), datetime('15-Oct-2019')
];

[data, vars] = selection_of_training_data(vars, data);
data = selection_of_testing_data(vars, data);

function [data, vars] = selection_of_training_data(vars, data)
    for i = 1:vars.numOfTelemetry
        data.seriesdataWfeatures_inter{i} = data.
            seriesdataWfeatures_inter{i}';
```

```
16
17          data.X_Train{i} = [];
18          data.Y_Train{i} = [];
19
20          [vars.unixTrainStart, vars.unixTrainStop] =
                datetime2unixtimestemp(vars.dateTrainStart, vars.
                dateTrainStop);
21          [vars.indexTrainStart, vars.indexTrainStop] =
                find_closest_indexes(data, vars.unixTrainStart, vars.
                unixTrainStop);
22
23          data.X_Train{i} = data.seriesdataWfeatures_inter{i}(:, vars.
                indexTrainStart:vars.indexTrainStop-1);
24          data.Y_Train{i} = data.seriesdataWfeatures_inter{i}(:, vars.
                indexTrainStart+1:vars.indexTrainStop);
25          data.Train_Time = data.datetime_inter(vars.indexTrainStart:
                vars.indexTrainStop-1);
26      end
27  end
28
29  function data = selection_of_testing_data(vars, data)
30      for p = 1:size(vars.dateTest, 1)
31          for i = 1:length(data.seriesdataWfeatures)
32              Disp1Name = strcat('X_Test_', num2str(p));
33              Disp2Name = strcat('Y_Test_', num2str(p));
34              Disp3Name = strcat('Test_Time_', num2str(p));
35
36              data.(Disp1Name)  = [];
37              data.(Disp2Name)  = [];
38              data.(Disp3Name)  = [];
39
40              [vars.unixTestStart, vars.unixTestStop] =
                    datetime2unixtimestemp(vars.dateTest(p, 1), vars.
                    dateTest(p, 2));
```

```
41              [vars.indexTestStart, vars.indexTestStop] =
                    find_closest_indexes(data, vars.unixTestStart, vars.
                    unixTestStop);
42
43              data.(Disp1Name) = data.seriesdataWfeatures_inter{i}(:,
                    vars.indexTestStart:vars.indexTestStop-1);
44              data.(Disp2Name) = data.seriesdataWfeatures_inter{i}(:,
                    vars.indexTestStart+1:vars.indexTestStop);
45              data.(Disp3Name) = data.datetime_inter(vars.indexTestStart
                    :vars.indexTestStop-1);
46          end
47      end
48  end
49
50  function [indexStart, indexStop] = find_closest_indexes(data,
        unixStart, unixStop)
51      % Find the index of the exact match (if it exists)
52      exactIndexStart = find(data.time_inter == unixStart, 1);
53      exactIndexStop  = find(data.time_inter == unixStop, 1);
54      % If an exact match was found, return its index
55      if ~isempty(exactIndexStart)
56          indexStart = exactIndexStart;
57      end
58      if ~isempty(exactIndexStop)
59          indexStop = exactIndexStop;
60      end
61      % Otherwise, find the index of the closest value
62      [~, indexStart] = min(abs(data.time_inter - unixStart));
63      [~, indexStop]  = min(abs(data.time_inter - unixStop));
64  end
65
66  function [unixStart, unixStop] = datetime2unixtimestemp(dateStart,
        dateStop)
67      unixStart = posixtime(dateStart);
```

```
    unixStop = posixtime(dateStop);
end
```

Listing A.3: MATLAB Script for Data Preprocessing

```
dataTrain        = data.X_Train{1};
numStepsTraining = round(numel(dataTrain));
indexTrain       = 1:numStepsTraining;
trainTime        = data.Train_Time;

% Choose one of the 3 testing periods
test_period = 1; % 1, 2, 3
if test_period == 1
    % Test Period 1 (24-Jun-2019 : 07-Jul-2019)
    dataTest        = data.X_Test_1;
    numStepsTesting = round(numel(dataTest));
    indexTest       = 1:numStepsTesting;
    testTime        = data.Test_Time_1;

elseif test_period == 2
    % Test Period 2 (10-Sep-2019 : 24-Sep-2019)
    dataTest        = data.X_Test_2;
    numStepsTesting = round(numel(dataTest));
    indexTest       = 1:numStepsTesting;
    testTime        = data.Test_Time_2;

elseif test_period == 3
    % Test Period 3 (01-Oct-2019 : 15-Oct-2019)
    dataTest        = data.X_Test_3;
    numStepsTesting = round(numel(dataTest));
    indexTest       = 1:numStepsTesting;
    testTime        = data.Test_Time_3;
end

%% Data Standardization
```

```
muData  = mean(data.data);
sigData = std(data.data);

% Train Data
TrainStandardizeddata = (dataTrain - muData) / sigData;

% Test Data
TestStandardizeddata = (dataTest - muData) / sigData;

%% Prepare Independent and Dependent Variables
numDays = 14;
numHours = 2;
maxLag = round(size(dataTrain, 2) / (numDays * 24)) * numHours;
vars.Lag = 1:maxLag;

% Train Data
XTrain = zeros(max(vars.Lag), size(TrainStandardizeddata, 2));
YTrain = TrainStandardizeddata(max(vars.Lag) + 1:end);
XrTrain = cell(size(XTrain, 2), 1);
YrTrain = zeros(size(YTrain, 2), 1);

for i = 1:size(XTrain, 2)
    XTrain(:, i) = TrainStandardizeddata(max(vars.Lag) + 1 - i:end - i
       );
    XrTrain{i} = XTrain(:, i);
    YrTrain(i) = YTrain(i);
end

% Test Data
XTest = zeros(max(vars.Lag), size(TestStandardizeddata, 2));
YTest = TestStandardizeddata(max(vars.Lag) + 1:end);
XrTest = cell(size(XTest, 2), 1);
YrTest = zeros(size(YTest, 2), 1);

```

```
for i = 1:size(XTest, 2)
    XTest(:, i) = TestStandardizeddata(max(vars.Lag) + 1 - i:end - i);
    XrTest{i} = XTest(:, i);
    YrTest(i) = YTest(i);
end
```

Listing A.4: MATLAB Script for CNN-LSTM Network Architecture Definition

```
vars.NetOption = "CNN-LSTM";
numFeatures = size(XTrain, 1);
numResponses = 1;
FiltZise = 5;

if strcmp(vars.NetOption, "CNN-LSTM")
    layers = [
        % Here input the sequence.
        sequenceInputLayer([numFeatures 1 1], 'Name', 'input')
        sequenceFoldingLayer('Name', 'fold')

        % From here, do your engineering design of your CNN feature
            extraction
        convolution2dLayer(FiltZise, 32, 'Padding', 'same', '
            WeightsInitializer', 'he', 'Name', 'conv', 'DilationFactor'
            , 1)
        batchNormalizationLayer('Name', 'bn')
        eluLayer('Name', 'elu')
        convolution2dLayer(FiltZise, 32, 'Padding', 'same', '
            WeightsInitializer', 'he', 'Name', 'conv1', 'DilationFactor
            ', 2)
        eluLayer('Name', 'elu1')
        convolution2dLayer(FiltZise, 32, 'Padding', 'same', '
            WeightsInitializer', 'he', 'Name', 'conv2', 'DilationFactor
            ', 4)
        eluLayer('Name', 'elu2')
        convolution2dLayer(FiltZise, 32, 'Padding', 'same', '
```

```
        WeightsInitializer', 'he', 'Name', 'conv3', 'DilationFactor
        ', 8)
    eluLayer('Name', 'elu3')
    convolution2dLayer(FiltZise, 32, 'Padding', 'same', '
        WeightsInitializer', 'he', 'Name', 'conv4', 'DilationFactor
        ', 16)
    eluLayer('Name', 'elu4')
    averagePooling2dLayer(1, 'Stride', FiltZise, 'Name', 'pool1')
    % Here you finish your CNN design and the next step is to
        unfold and flatten.
    sequenceUnfoldingLayer('Name', 'unfold')
    flattenLayer('Name', 'flatten')

    % From here, the RNN design.
    gruLayer(128, 'Name', 'gru1', 'RecurrentWeightsInitializer', '
        He', 'InputWeightsInitializer', 'He')
    lstmLayer(64, 'Name', 'gru2', 'RecurrentWeightsInitializer', '
        He', 'InputWeightsInitializer', 'He')
    dropoutLayer(0.25, 'Name', 'drop2')
    lstmLayer(32, 'OutputMode', 'last', 'Name', 'bil4', '
        RecurrentWeightsInitializer', 'He', '
        InputWeightsInitializer', 'He')
    dropoutLayer(0.25, 'Name', 'drop3')
    % Here finish the RNN design.

    % Use a fully connected layer with one neuron because you will
         predict one step ahead.
    fullyConnectedLayer(numResponses, 'Name', 'fc')
    regressionLayer('Name', 'output')
];

layers = layerGraph(layers);
layers = connectLayers(layers, 'fold/miniBatchSize', 'unfold/
    miniBatchSize');
```

```
end
```

Listing A.5: MATLAB Script for Training Options and Network Training

```
%% Training Options
optns.solverName = 'adam';
% 'sgdm', 'rmsprop', 'adam'
optns.Plots = 'training-progress';
% 'none', 'training-progress'
optns.Verbose = false;
% 'true', 'false'
optns.MaxEpochs = 450;
% positive integer
optns.MiniBatchSize = 64;
% positive integer
optns.Shuffle = 'every-epoch';
% 'once', 'never', 'every-epoch'
optns.InitialLearnRate = 0.00611;
% positive scalar
optns.LearnRateSchedule = 'piecewise';
% 'none', 'piecewise'
optns.LearnRateDropPeriod = 96;
% positive scalar
optns.LearnRateDropFactor = 0.25;
% positive scalar from 0 to 1
optns.GradientThreshold = 1;
% positive scalar
optns.ExecutionEnvironment = 'cpu';
% 'auto', 'cpu', 'gpu', 'multi-gpu', 'parallel'

options = trainingOptions(optns.solverName, ...
    'Plots', optns.Plots, ...
    'Verbose', optns.Verbose, ...
    'MaxEpochs', optns.MaxEpochs, ...
    'MiniBatchSize', optns.MiniBatchSize, ...
```

```
    'Shuffle', optns.Shuffle, ...
    'InitialLearnRate', optns.InitialLearnRate, ...
    'LearnRateSchedule', optns.LearnRateSchedule, ...
    'LearnRateDropPeriod', optns.LearnRateDropPeriod, ...
    'LearnRateDropFactor', optns.LearnRateDropFactor, ...
    'GradientThreshold', optns.GradientThreshold, ...
    'ExecutionEnvironment', optns.ExecutionEnvironment);

%% Train Hybrid Network
hybrid_network = trainNetwork(XrTrain, YrTrain, layers, options);
```

Listing A.6: MATLAB Script for Assessing Performance Metrics and Visualizing Results on Training Datasets

```
% Create a "Train Folder" for save the results
folder_path_train = 'H:\Arvotti\Results\TRAIN';
folder_contents_train = dir(folder_path_train);
num_subfolders_train = 1;

% Count the number of existing subfolders
for i = 1:length(folder_contents_train)
    if folder_contents_train(i).isdir && ~strcmp(folder_contents_train
        (i).name, '.') && ~strcmp(folder_contents_train(i).name, '..')
        num_subfolders_train = num_subfolders_train + 1;
    end
end

I_train = num_subfolders_train;

% Create a new folder with a unique name
if I_train < 10
    folder_name_train = "CASE_TRAIN_00" + I_train;
else
    folder_name_train = "CASE_TRAIN_0" + I_train;
end

```

```
data.new_folder_path_train = fullfile(folder_path_train,
    folder_name_train);
mkdir(data.new_folder_path_train);

%% Evaluate The Network: Train Period
YPred_Train = predict(hybrid_network,XrTrain,'ExecutionEnvironment',
    optns.ExecutionEnvironment,'MiniBatchSize',numFeatures);
YPred_Train = YPred_Train';

YPred_Train = sigTrain.*YPred_Train+muTrain;
YTrain      = sigTrain.*YTrain+muTrain;

%% Metrics Calculation [Train Data]
Errors_train = YTrain - YPred_Train;

MSE_train_total = mean(Errors_train.^2);
RMSE_train_total = sqrt(MSE_train_total);
NRMSE_train_total = RMSE_train_total / mean(YTrain);
ErrorMean_train_total = mean(Errors_train);
ErrorStd_train_total = std(Errors_train);

% NRMSE
for i = 1:size(YTrain, 2)
    mse_train(1, i) = (Errors_train(1, i)).^2;
    rmse_train(1, i) = sqrt(mse_train(1, i));
    nrmse_train(1, i) = rmse_train(1, i) / (max(YTrain) - min(YTrain))
        ;
end
nrmse_mean_train = mean(nrmse_train);
nrmse_std_train  = std(nrmse_train);

% Plot The Results [Train Data]
Train_Time = trainTime(1, max(vars.Lag) + 1:end);
Disp1Name = "Predicted_Outcomes_vs_Observed_Values_[Train_Data]_(" +
```

```
    data.telemetry{1} + ")";
Disp2Name = "Errors_[Train_Data]_(" + data.telemetry{1} + ")";
Disp3Name = "Error_Histogram_[Train_Data]_(" + data.telemetry{1} + ")
    ";
Disp4Name = "NRMSE_[Train_Data]_(" + data.telemetry{1} + ")";
Disp5Name = "NRMSE_Histogram_[Train_Data]_(" + data.telemetry{1} + ")
    ";

% Predicted Outcomes vs Observed Values
figure('Name', 'Predicted_Outcomes_vs_Observed_Values_[Train_Data]', '
    NumberTitle', 'off')
hold on
title("Predicted Outcomes vs Observed Values [Train Data] (" + data.
    telemetry{1} + ")", 'Interpreter', 'none')
plot(Train_Time, YTrain, 'LineWidth', vars.lineWidth)
plot(Train_Time, YPred_Train, '.-', 'LineWidth', vars.lineWidth);
xlabel("Time")
xlim([Train_Time(1) Train_Time(end)]);
grid minor;
ylabel("Telemetry Value")
legend(["Observed Values" "Predicted Outcomes " + vars.NetOption], '
    Location', "best")
set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
    FontWeight', 'bold');
set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
    XMinorTick', 'on', 'YMinorTick', 'on', ...
     'XGrid', 'on', 'YGrid', 'on', 'TickDir', 'in', 'TickLength', [.015
         .015], ...
     'FontName', 'avantgarde', 'FontSize', vars.fontSize, 'FontWeight',
         'normal');
savefig(fullfile(data.new_folder_path_train, sprintf('%s.fig',
    Disp1Name)));

% Errors
```

```
figure('Name', 'Errors_[Train_Data]', 'NumberTitle', 'off');
hold on
title("Errors [Train Data] (" + data.telemetry{1} + ")", 'Interpreter', 'none');
subtitle(['Error Mean = ' num2str(ErrorMean_train_total) ', Error StD = ' num2str(ErrorStd_train_total)]);
plot(Train_Time, Errors_train, "LineWidth", vars.lineWidth, 'Color', '0.64,0.08,0.18');
xlabel('Time');
xlim([Train_Time(1) Train_Time(end)]);
grid minor;
ylabel('Error');
set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', 'FontWeight', 'bold');
set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', 'XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on', ...
    'TickDir', 'in', 'TickLength', [.015 .015], ...
    'FontName', 'avantgarde', 'FontSize', vars.fontSize, 'FontWeight', 'normal');
savefig(fullfile(data.new_folder_path_train, sprintf('%s.fig', Disp2Name)));

% Error Histogram
figure('Name', 'Error_Histogram_[Train_Data]', 'NumberTitle', 'off');
hold on
title("Error Histogram [Train Data] (" + data.telemetry{1} + ")", 'Interpreter', 'none');
subtitle(['Error Mean = ' num2str(ErrorMean_train_total) ', Error StD = ' num2str(ErrorStd_train_total)]);
histogram(Errors_train, "LineWidth", vars.lineWidth, 'FaceColor', '0.64,0.08,0.18');
grid minor;
xlabel('Error Magnitude');
```

```
99  ylabel('Occurrences')
100 set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
    FontWeight', 'bold');
101 set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
    XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
    , ...
102     'TickDir', 'in', 'TickLength', [.015 .015], ...
103     'FontName', 'avantgarde', 'FontSize', vars.fontSize, 'FontWeight',
        'normal');
104 savefig(fullfile(data.new_folder_path_train, sprintf('%s.fig',
    Disp3Name)));
105
106 % NRMSE
107 figure('Name', 'NRMSE_[Train_Data]', 'NumberTitle', 'off');
108 hold on
109 title("NRMSE [Train Data] (" + data.telemetry{1} + ")", 'Interpreter',
     'none');
110 subtitle(['NRMSE Total = ' num2str(NRMSE_train_total)...
111     ', NRMSE Mean = ' num2str(nrmse_mean_train)...
112     ', NRMSE StD = ' num2str(nrmse_std_train)]);
113 plot(Train_Time, nrmse_train, 'LineWidth', vars.lineWidth, 'Color', '
    0.49,0.18,0.56');
114 xlabel('Time');
115 xlim([Train_Time(1) Train_Time(end)]);
116 grid minor;
117 ylabel('NRMSE');
118 set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
    FontWeight', 'bold');
119 set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
    XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
    , ...
120     'TickDir', 'in', 'TickLength', [.015 .015], ...
121     'FontName', 'avantgarde', 'FontSize', vars.fontSize, 'FontWeight',
        'normal');
```

```
122 savefig(fullfile(data.new_folder_path_train, sprintf('%s.fig',
        Disp4Name)));
123
124 % NRMSE Histogram
125 figure('Name', 'NRMSE_Histogram_[Train_Data]', 'NumberTitle', 'off');
126 hold on
127 title("NRMSE Histogram [Train Data] (" + data.telemetry{1} + ")", '
        Interpreter', 'none');
128 subtitle(['NRMSE Total = ' num2str(NRMSE_train_total)...
129     ', NRMSE Mean = ' num2str(nrmse_mean_train)...
130     ', NRMSE StD = ' num2str(nrmse_std_train)]);
131 histogram(nrmse_train, 'LineWidth', vars.lineWidth, 'FaceColor', '
        0.49,0.18,0.56');
132 grid minor;
133 xlabel('NRMSE Magnitude');
134 ylabel('Occurrences')
135 set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
        FontWeight', 'bold');
136 set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
        XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
        , ...
137     'TickDir', 'in', 'TickLength', [.015 .015], ...
138     'FontName', 'avantgarde', 'FontSize', vars.fontSize, 'FontWeight',
            'normal');
139 savefig(fullfile(data.new_folder_path_train, sprintf('%s.fig',
        Disp5Name)));
140 save(fullfile(data.new_folder_path_train,'workspace.mat'));
```

Listing A.7: MATLAB Script for Assessing Performance Metrics and Visualizing Results on Testing Datasets

```
1 % Create a "Test Folder" to save the results
2 folder_path_test = 'H:\Arvotti\Results\TEST';
3 folder_contents_test = dir(folder_path_test);
4 num_subfolders_test = 2;
5
```

```
for i = 1:length(folder_contents_test)
    if folder_contents_test(i).isdir && ~strcmp(folder_contents_test(i).name, '.') && ~strcmp(folder_contents_test(i).name, '..')
        num_subfolders_test = num_subfolders_test + 1;
    end
end
I_test = num_subfolders_test;

% Create a new folder with a unique name inside the specified path
if I_test < 10
    folder_name_test = "CASE_TEST_00" + I_test;
elseif I_test >= 10
    folder_name_test = "CASE_TEST_0" + I_test;
end

data.new_folder_path_test = fullfile(folder_path_test, folder_name_test);
mkdir(data.new_folder_path_test);

%% Evaluate The Network: Test Period
YPred_Test = predict(hybrid_network, XrTest, 'ExecutionEnvironment', optns.ExecutionEnvironment, 'MiniBatchSize', numFeatures);
YPred_Test = YPred_Test';

YPred_Test = sigTest .* YPred_Test + muTest;
YTest = sigTest .* YTest + muTest;

%% Metrics Calculation [Test Data]
Errors_test = YTest - YPred_Test;

MSE_test_total = mean(Errors_test.^2);
RMSE_test_total = sqrt(MSE_test_total);
NRMSE_test_total = RMSE_test_total / mean(YTest);
ErrorMean_test_total = mean(Errors_test);
```

```
ErrorStd_test_total = std(Errors_test);

% NRMSE
nrmse_test = [];
for i = 1:size(YTest, 2)
    mse_test(1, i) = mean(Errors_test(1, i)).^2;
    rmse_test(1, i) = sqrt(mse_test(1, i));
    nrmse_test(1, i) = rmse_test(1, i) / (max(YTest) - min(YTest));
end
nrmse_mean_test = mean(nrmse_test);
nrmse_std_test = std(nrmse_test);

% Plot The Results [Test Data]
Test_Time = testTime(1, max(vars.Lag) + 1:end);

Disp1Name = "Predicted_Outcomes_vs_Observed_Values_[Test_Data]_(" +
    data.telemetry{1} + ")";
Disp2Name = "Errors_[Test_Data]_(" + data.telemetry{1} + ")";
Disp3Name = "Error_Histogram_[Test_Data]_(" + data.telemetry{1} + ")";
Disp4Name = "NRMSE_[Test_Data]_(" + data.telemetry{1} + ")";
Disp5Name = "NRMSE_Histogram_[Test_Data]_(" + data.telemetry{1} + ")";

% Predicted Outcomes vs Observed Values
figure('Name', 'Predicted_Outcomes_vs_Observed_Values_[Test_Data]', '
    NumberTitle', 'off')
hold on
title("Predicted Outcomes vs Observed Values [Test Data] (" + data.
    telemetry{1} + ")", 'Interpreter', 'none')
plot(Test_Time, YTest, 'LineWidth', vars.lineWidth);
plot(Test_Time, YPred_Test, '.-', 'LineWidth', vars.lineWidth)
xlabel("Time")
xlim([Test_Time(1) Test_Time(end)]);
grid minor;
ylabel("Telemetry Value")
```

```
68 legend(["Observed Values", "Predicted Outcomes " + vars.NetOption], '
       Location', "best")
69 set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
       FontWeight', 'bold');
70 set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
       XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
       , 'TickDir', 'in', 'TickLength', [.015 .015], 'FontName', '
       avantgarde', 'FontSize', vars.fontSize, 'FontWeight', 'normal');
71 savefig(fullfile(data.new_folder_path_test, sprintf('%s.fig',
       Disp1Name)));
72
73 % Errors
74 figure('Name', 'Errors_[Test_Data]', 'NumberTitle', 'off');
75 hold on
76 title("Errors [Test Data] (" + data.telemetry{1} + ")", 'Interpreter',
        'none');
77 subtitle(['Error Mean = ' num2str(ErrorMean_test_total) ', Error StD =
        ' num2str(ErrorStd_test_total)]);
78 plot(Test_Time, Errors_test, "LineWidth", vars.lineWidth, 'Color', '
       0.64,0.08,0.18');
79 xlabel('Time');
80 xlim([Test_Time(1) Test_Time(end)]);
81 grid minor;
82 ylabel('Error');
83 set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
       FontWeight', 'bold');
84 set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
       XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
       , 'TickDir', 'in', 'TickLength', [.015 .015], 'FontName', '
       avantgarde', 'FontSize', vars.fontSize, 'FontWeight', 'normal');
85 savefig(fullfile(data.new_folder_path_test, sprintf('%s.fig',
       Disp2Name)));
86
87 % Error Histogram
```

```
figure('Name', 'Error_Histogram_[Test_Data]', 'NumberTitle', 'off');
hold on
title("Error Histogram [Test Data] (" + data.telemetry{1} + ")", '
    Interpreter', 'none');
subtitle(['Error Mean = ' num2str(ErrorMean_test_total) ', Error StD =
     ' num2str(ErrorStd_test_total)]);
histogram(Errors_test, "LineWidth", vars.lineWidth, 'FaceColor', '
    0.64,0.08,0.18');
grid minor;
xlabel('Error Magnitude');
ylabel('Occurrences')
set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
    FontWeight', 'bold');
set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
    XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
    , 'TickDir', 'in', 'TickLength', [.015 .015], 'FontName', '
    avantgarde', 'FontSize', vars.fontSize, 'FontWeight', 'normal');
savefig(fullfile(data.new_folder_path_test, sprintf('%s.fig',
    Disp3Name)));

% NRMSE
figure('Name', 'NRMSE_[Test_Data]', 'NumberTitle', 'off');
hold on
title("NRMSE [Test Data] (" + data.telemetry{1} + ")", 'Interpreter',
    'none');
subtitle(['NRMSE Total = ' num2str(NRMSE_test_total)]);
plot(Test_Time, nrmse_test(1, :), 'LineWidth', vars.lineWidth, 'Color'
    , '0.49,0.18,0.56');
xlabel('Time');
xlim([Test_Time(1) Test_Time(end)]);
grid minor;
ylabel('NRMSE');
set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
    FontWeight', 'bold');
```

```
111 set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
    XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
    , 'TickDir', 'in', 'TickLength', [.015 .015], 'FontName', '
    avantgarde', 'FontSize', vars.fontSize, 'FontWeight', 'normal');
112 savefig(fullfile(data.new_folder_path_test, sprintf('%s.fig',
    Disp4Name)));
113
114 % NRMSE Histogram
115 figure('Name', 'NRMSE_Histogram_[Test_Data]', 'NumberTitle', 'off');
116 hold on
117 title("NRMSE Histogram [Test Data] (" + data.telemetry{1} + ")", '
    Interpreter', 'none');
118 subtitle(['NRMSE Total = ' num2str(NRMSE_test_total) ', NRMSE Mean = '
     num2str(nrmse_mean_test) ', NRMSE StD = ' num2str(nrmse_std_test)
    ]);
119 histogram(nrmse_test(1, :), "LineWidth", vars.lineWidth, 'FaceColor',
    '0.49,0.18,0.56');
120 grid minor;
121 xlabel('NRMSE Magnitude');
122 ylabel('Occurrences');
123 set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
    FontWeight', 'bold');
124 set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
    XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
    , 'TickDir', 'in', 'TickLength', [.015 .015], 'FontName', '
    avantgarde', 'FontSize', vars.fontSize, 'FontWeight', 'normal');
125 savefig(fullfile(data.new_folder_path_test, sprintf('%s.fig',
    Disp5Name)));
126
127 %% Anomaly Detection
128 lowerThreshold = prctile(abs(Errors_test), 0);
129 upperThreshold = prctile(abs(Errors_test), 95);
130 window_size = maxLag;
131 anomalies = [];
```

```
132 nrmse_anomaly = [];
133
134 if data.telemetry{1} == "current_1"
135     nrmse_threshold = 3.55;
136     hi_upper_th = 100;
137 elseif data.telemetry{1} == "voltage_1"
138     nrmse_threshold = 1.5;
139     hi_upper_th = 100;
140 elseif data.telemetry{1} == "current_2"
141     nrmse_threshold = 0.33;
142     hi_upper_th = 100;
143 elseif data.telemetry{1} == "voltage_2"
144     nrmse_threshold = 1.4;
145     hi_upper_th = 100;
146 elseif data.telemetry{1} == "current_3"
147     nrmse_threshold = 1.4;
148     hi_upper_th = 100;
149 end
150
151 for i = 1:size(Errors_test, 2)
152     if i > window_size && i < size(Errors_test, 2)
153         anomalies(1, i) = abs(Errors_test(1, i)) < lowerThreshold |
                abs(Errors_test(1, i)) > upperThreshold;
154         if anomalies(1, i)
155             % Calculate NRMSE in a window of window_size timestamps
                    before the anomaly
156             windowStart = i - window_size;
157             windowEnd = i;
158             window = Errors_test(1, windowStart:windowEnd);
159             nrmse_anomaly(1, i) = sqrt(mean(window.^2)) / (max(
                    Errors_train) - min(Errors_train)); % PA3.0
160
161             if nrmse_anomaly(1, i) < nrmse_threshold
162                 anomalies(1, i) = 0;
```

```
                nrmse_anomaly(1, i) = 0;
            end
        end
    else
        anomalies(1, i) = 0;
        nrmse_anomaly(1, i) = 0;
    end
end

anomalies = logical(anomalies);
anomalyTimes = Test_Time(anomalies);

HI = [];
for i = 1:size(Errors_test, 2)
    if nrmse_anomaly(1, i) < nrmse_threshold
        HI(1, i) = 0;
    elseif nrmse_anomaly(1, i) >= nrmse_threshold && nrmse_anomaly(1,
       i) < hi_upper_th
        HI(1, i) = nrmse_anomaly(1, i) * 100;
    elseif nrmse_anomaly(1, i) >= hi_upper_th
        HI(1, i) = 100;
    end
end

% Health Index
Disp6Name = "Health_Index_[Test_Data]_(" + data.telemetry{1} + ")";
figure('Name', 'Health_Index_[Test_Data]', 'NumberTitle', 'off');
hold on
title("Health Index [Test Data] (" + data.telemetry{1} + ")", '
   Interpreter', 'none');
plot(Test_Time, HI, 'LineWidth', vars.lineWidth, 'Color', '
   1.00,0.00,0.00');
grid minor;
ylim([0, 100]);
```

```
194 xlim([Test_Time(1), Test_Time(end)]);
195 xlabel('Time');
196 ylabel('Health Index');
197 set(gca, 'FontSize', vars.fontSize, 'FontName', 'Adobe Kaiti Std R', '
    FontWeight', 'bold');
198 set(gca, 'Box', 'on', 'LineWidth', vars.lineWidth, 'Layer', 'top', '
    XMinorTick', 'on', 'YMinorTick', 'on', 'XGrid', 'on', 'YGrid', 'on'
    , 'TickDir', 'in', 'TickLength', [.015 .015], 'FontName', '
    avantgarde', 'FontSize', vars.fontSize, 'FontWeight', 'normal');
199 savefig(fullfile(data.new_folder_path_test, sprintf('%s.fig',
    Disp6Name)));
200 save(fullfile(data.new_folder_path_test, 'workspace.mat'));
```

References

[1] Nanosats database. [Online]. Available: https://www.nanosats.eu/

[2] K. O'Donnell and G. Richardson, "Small satellite trending & reliability 2009-2018," *The Aerospace Corporation*, 2019.

[3] X.-Y. Ji, Y.-Z. Li, G.-Q. Liu, J. Wang, S.-H. Xiang, X.-N. Yang, and Y.-Q. Bi, "A brief review of ground and flight failures of chinese spacecraft," *Progress in Aerospace Sciences*, vol. 107, pp. 19–29, 2019.

[4] J. Schumann, O. J. Mengshoel, and T. Mbaya, "Integrated software and sensor health management for small spacecraft," in *2011 IEEE Fourth International Conference on Space Mission Challenges for Information Technology*, 2011, pp. 77–84.

[5] S. Xie, X. Peng, X. Zhong, and C. Liu, "Fault diagnosis of the satellite power system based on the bayesian network," in *2013 8th International Conference on Computer Science & Education*, 2013, pp. 1004–1008.

[6] D. Pan, D. Liu, J. Zhou, and G. Zhang, "Anomaly detection for satellite power subsystem with associated rules based on kernel principal component analysis," *Microelectronics Reliability*, vol. 55, no. 9, pp. 2082–2086, 2015.

[7] T. Yairi, N. Takeishi, T. Oda, Y. Nakajima, N. Nishimura, and N. Takata, "A data-driven health monitoring method for satellite housekeeping data based on probabilistic clustering and dimensionality reduction," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 53, no. 3, pp. 1384–1401, 2017.

[8] H. Fang, H. Shi, Y. Dong, H. Fan, and S. Ren, "Spacecraft power system fault diagnosis based on dnn," in *2017 Prognostics and System Health Management Conference (PHM-Harbin)*, 2017, pp. 1–5.

[9] M. Suo, M. Zhang, D. Zhou, B. Zhu, and S. Li, "Fault diagnosis of satellite power system using variable precision fuzzy neighborhood rough set," in *2017 36th Chinese Control Conference (CCC)*, 2017, pp. 7301–7306.

[10] S. Dheepadharshani, S. Anandh, K. B. Bhavinaya, and R. Lavanya, "Multivariate time-series classification for automated fault detection in satellite power systems," in *2019 International Conference on Communication and Signal Processing (ICCSP)*, 2019, pp. 0814–0817.

[11] S. K. Ibrahim, A. Ahmed, M. A. E. Zeidan, and I. Ziedan, "Machine learning techniques for satellite fault diagnosis," *Ain Shams Engineering Journal*, vol. 11, pp. 45–56, 2020.

[12] B. Xiao and S. Yin, "A deep learning based data-driven thruster fault diagnosis approach for satellite attitude control system," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 10, pp. 10 162–10 170, 2021.

[13] S. Mengqi, Y. Xinqing, L. Guiming, and H. Zhongmin, "Fuzzy-based analysis of thermal effects on component failure for leo satellites," in *2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, 2021, pp. 933–943.

[14] A. Rahimi, K. D. Kumar, and H. Alighanbari, "Failure prognosis for satellite reaction wheels using kalman filter and particle filter," *Journal of Guidance Control and Dynamics*, vol. 43, pp. 585–588, 2020.

[15] J. Wang, H. Zheng, Q. Li, H. Wu, and B. Zhou, "Prognostic for on-orbit satellite momentum wheel based on the similitude method," in *2015 Prognostics and System Health Management Conference (PHM)*, 2015, pp. 1–5.

[16] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2306–2318, 2017.

[17] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering and System Safety*, vol. 172, no. C, pp. 1–11, 2018.

[18] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Söderström, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[19] J. Dong, Y. Ma, and D. Liu, "Deep learning based multiple sensors monitoring and abnormal discovery for satellite power system," in *2019 International Conference on Sensing, Diagnostics, Prognostics, and Control (SDPC)*, 2019, pp. 638–643.

[20] C.-G. Huang, H.-Z. Huang, and Y.-F. Li, "A bidirectional lstm prognostics method under multiple operational conditions," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 11, pp. 8792–8802, 2019.

[21] D. Pan, Z. Song, L. Nie, and B. Wang, "Satellite telemetry data anomaly detection using bi-lstm prediction based model," in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1–6.

[22] Y. Wang, Y. Wu, Q. Yang, and J. Zhang, "Anomaly detection of spacecraft telemetry data using temporal convolution network," in *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2021, pp. 1–5.

[23] J. Murphy, J. E. Ward, and B. M. Namee, "Machine learning in space: A review of machine learning algorithms and hardware for space applications," in *Irish Conference on Artificial Intelligence and Cognitive Science*, 2021.

[24] M. Sirajul Islam and A. Rahimi, "Fault prognosis of satellite reaction wheels using a two-step lstm network," in *2021 IEEE International Conference on Prognostics and Health Management (ICPHM)*, 2021, pp. 1–7.

[25] M. ElDali and K. D. Kumar, "Fault diagnosis and prognosis of aerospace systems using growing recurrent neural networks and lstm," in *2021 IEEE Aerospace Conference (50100)*, 2021, pp. 1–20.

[26] D. Han, J. Yu, M. Gong, Y. Song, and L. Tian, "A remaining useful life prediction approach based on low-frequency current data for bearings in spacecraft," *IEEE Sensors Journal*, vol. 21, no. 17, pp. 18 978–18 989, 2021.

[27] C. Wang, Z. Zhu, N. Lu, Y. Cheng, and B. Jiang, "A data-driven degradation prognostic strategy for aero-engine under various operational conditions," *Neurocomputing*, vol. 462, pp. 195–207, oct 2021.

[28] Z. Zeng, G. Jin, C. Xu, S. Chen, Z. Zeng, and L. Zhang, "Satellite telemetry data anomaly detection using causal network and feature-attention-based lstm," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–21, 2022.

[29] A.-E. R. Abd-Elhay, W. A. Murtada, and M. I. Youssef, "A reliable deep learning approach for time-varying faults identification: Spacecraft reaction wheel case study," *IEEE Access*, vol. 10, pp. 75 495–75 512, 2022.

[30] V. Muthusamy and K. D. Kumar, "Failure prognosis and remaining useful life prediction of control moment gyroscopes onboard satellites," *Advances in Space Research*, vol. 69, no. 1, pp. 718–726, 2022.

[31] X. Wei, X. Mu, T. Jiang, W. Liu, and Z. Zeng, "Fault diagnosis method of spacecraft control systems based on pca-resnet," *Journal of Physics: Conference Series*, vol. 2258, no. 1, p. 012062, apr 2022.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] H. Sanchez. Mathworks. [Online]. Available: https://it.mathworks.com/matlabcentral/fileexchange/91360-time-series-forecasting-using-hybrid-cnn-rnn