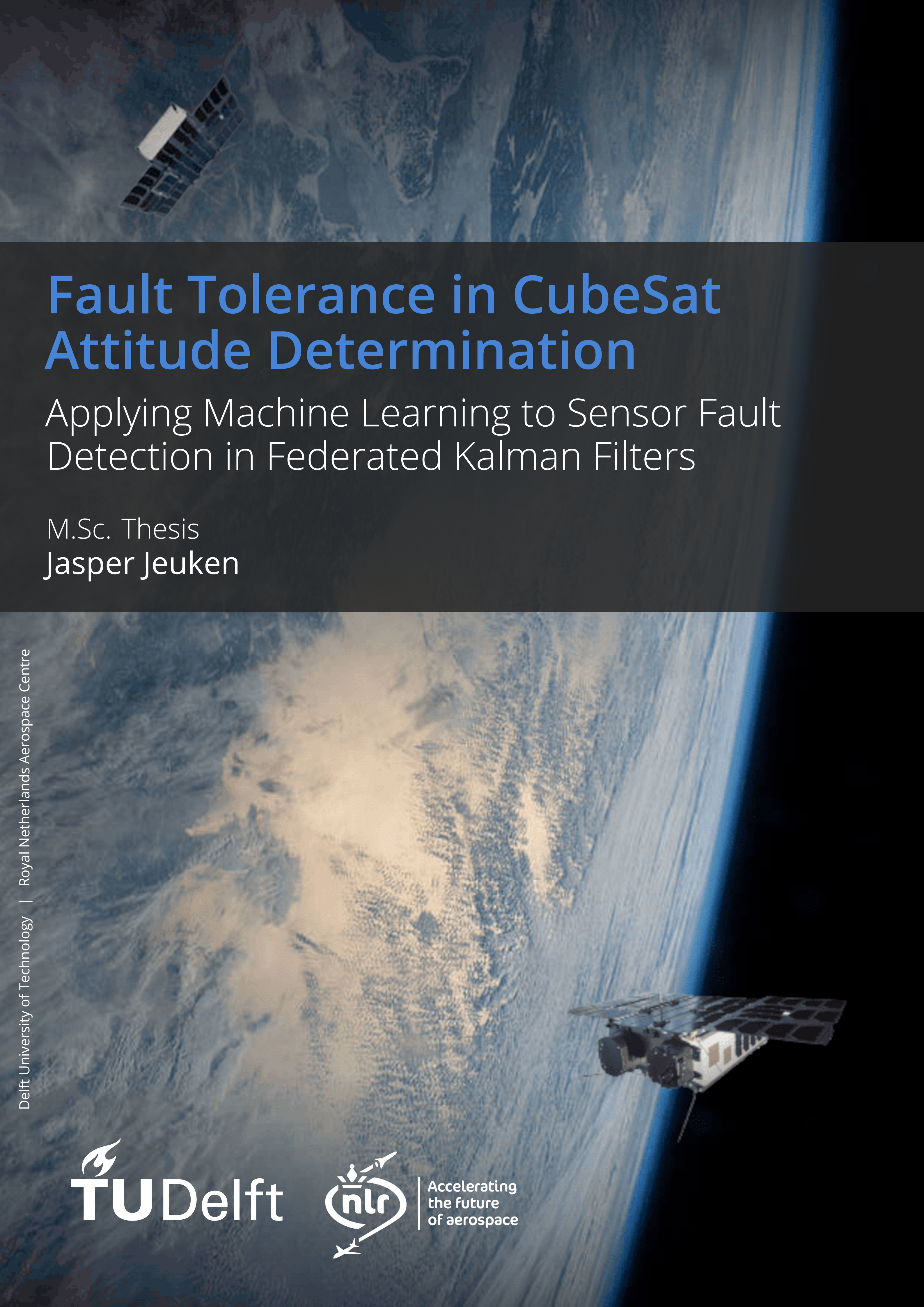
Fault Tolerance in CubeSat Attitude Determination

Applying Machine Learning to Sensor Fault Detection in Federated Kalman Filters

M.Sc. Thesis

Jasper Jeuken



Delft University of Technology | Royal Netherlands Aerospace Centre

TUDelft
nlr Accelerating the future of aerospace

[This page was intentionally left blank]

Fault Tolerance in CubeSat Attitude Determination

Applying Machine Learning to Sensor Fault Detection in Federated Kalman Filters

by

Jasper Jeuken

Student number: 4856716

To obtain the degree of Master of Science in Aerospace Engineering
at the Delft University of Technology.

Defended publicly on Tuesday 17 March 2026 at 09:00.

Duration:	June 2025 — March 2026	
University:	Department of Space Engineering, Faculty of Aerospace Engineering, Delft University of Technology (TU Delft)	
Host Institute:	Royal Netherlands Aerospace Centre (NLR)	
Committee:	Dr. A. Cervone	Chair
	Dr. J. Guo	Supervisor (TU Delft)
	Dr.ir. E. van Kampen	External examiner
	MSc. T.D. Landzaat	Supervisor (NLR)
Cover:	MilSpace2 satellites — NLR	

Preface

This thesis concludes my education as an Aerospace Engineer at the Delft University of Technology. Over the years I have learnt so much from so many people, and had the chance to work on projects I could never have imagined I would be part of. From working on a student-built rocket engine, to standing on the Shetland Islands helping test fire an orbital rocket, to researching ways to improve a vital spacecraft system, every step of the way provided me with valuable experiences for my studies, career, and life overall.

This thesis is the result of nine months of work, during which I came across many new ideas from experts, literature, and everyone around me. In the beginning the project seemed daunting, but over time I was able to grow my knowledge and skills to get accustomed to deeper topics. I am proud of the work I have achieved, and am grateful for all who brought me to this point in my life.

I would like to thank everyone who has supported me throughout the years and during this project. First, my supervisors, Jian and Tom, for supporting my work and fairly criticizing it along the way, and supporting the submission of an abstract to the IAC. My parents, who were always there to help me in any way possible. My brother, who I will follow the footsteps of in becoming a Master of Science. My friends, who I was always able to rely on to keep my spirits up. My cat, Cali, who contributed to this thesis by walking over my keyboard repeatedly. And last but not least, Sari (Wema), who sticks with me through thick and thin, laughs at my (bad) jokes, and is always there for me no matter the time or place.

Jasper Jeuken
Delft, March 2026

Part of the work presented in this report has been submitted as an abstract to the International Astronautical Congress (IAC) 2026 for presentation, within the 33rd IAA Symposium on Small Satellite Missions (Session: Generic Technologies for Small Satellites). At the time of writing, the submission is under review.

Abstract

The use of hardware redundancy in CubeSats is often limited by physical and budgetary constraints, making alternative approaches to fault tolerance essential for maintaining attitude determination performance. The extended and unscented Kalman filters are typically used to combine all sensor information in a centralized fashion. Federated Kalman filters offer improved fault isolation since each sensor group is associated with an independent local filter, whose estimates are fused by a master filter. Conventional anomaly detection relies on either a Mahalanobis distance- or residual-based measure. These methods require manual threshold selection and do not capture temporal patterns, limiting their effectiveness especially for gradual or subtle faults. In this work, machine learning (ML)-based alternatives are suggested and compared to these conventional approaches, showing a significant increase in detection performance while overcoming some limitations of the traditional methods. The increased computational load associated with these alternatives is assessed against typical microcontroller-based on-board computers used for attitude determination on CubeSats, which was found to be feasible under moderate inference rates. The results demonstrate that the use of ML-based detection within a federated Kalman filter can substantially enhance the reliability of CubeSat attitude determination systems.

Summary

The attitude determination system (ADS) of a CubeSat fuses measurements from multiple sensors into an accurate estimate of the spacecraft attitude. Faults in a sensor can significantly degrade the performance, and numerous CubeSat missions have encountered operational issues from undetected or unmitigated sensor faults.

Attitude determination traditionally employs the extended or unscented Kalman filters, which combine all sensor information in a single, centralized estimation. The federated Kalman filter (FKF) instead distributes the estimation process across several local filters, each linked to a specific sensor system. A master filter then fuses the local estimates. Because each local filter produces an intermediate estimate, detected faults can be attributed directly to the individual system causing it. They can be excluded from the final fusion step to isolate and recover from the fault.

Conventional detection of faults in the FKF relies on statistical properties. Two common methods use the Mahalanobis distance between the local and master filter, or the residual between the predicted and observed measurement. Both methods treat samples independently and require manually selected thresholds, limiting their ability to detect faults.

This research project shows the application of machine learning (ML)-based fault detection in this context, replacing the conventional approaches. Two ML-based methods were selected based on criteria including their expected performance and operational viability. The long short-term memory (LSTM) predictor is a recurrent neural network (RNN) which excels at learning temporal patterns. This method is computationally intensive but accurate. The second method, the isolation forest (iForest), is a more lightweight alternative which relies on randomly splitting the feature space into binary trees.

A simulation of the attitude behaviour of a CubeSat was developed, including models of typical attitude sensor hardware. Their measurements are fused by local filters using the unscented quaternion estimator (USQUE), before being combined to the final output forming a no-reset FKF. A variety of faults were injected in various operational scenarios. The response of the conventional and ML-based methods was measured and compared. The iForest offers no improvement to the conventional approaches. The LSTM predictor, however, shows a 10% increase in detection accuracy and reduces the time-to-detection by over threefold.

The computational impact of the ML-based methods was estimated using a static analysis. This is compared to a collection of microcontroller-based on-board computers (OBCs) used on CubeSats. The iForest developed in this project was deemed infeasible due to the large static model size not fitting into the memory of typical OBCs. This creates a bottleneck when repeatedly accessed during tree traversal. The LSTM predictor instead shows high computational load, though it remains feasible on higher-end units at limited inference rates.

In conclusion, the LSTM predictor is found to be a suitable ML-based alternative to conventional fault detection methods in the context of attitude determination using the FKF on CubeSats. The method benefits from the ability to incorporate the temporal history of the samples, allowing extraction of more detailed time-varying patterns compared to the point-wise conventional approaches.

Future works should explore reducing the shared reliance on gyroscope propagation to eliminate this as a single point of failure. Additionally, the large computational impact of the ML-based models could be reduced through optimization, including feature reduction, optimized implementations, and guided tuning. Hybrid strategies are also possible, combining conventional and ML-based detection.

Contents

List of Figures

List of Tables

List of Acronyms

ACS	attitude control system
Adam	adaptive moment estimation
ADCS	attitude determination and control system
ADS	attitude determination system
AE	autoencoder
AEKF	additive extended Kalman filter
AI	artificial intelligence
AOCS	attitude and orbit control system
AU	astronomical unit
AUC	area under the curve
CKF	cubature Kalman filter
COMP	comparison
COTS	commercial off-the-shelf
DOF	degree of freedom
ECEF	Earth-centered Earth-fixed
ECI	Earth-centered inertial
EKF	extended Kalman filter
FD	fault detection
FDD	fault detection and diagnosis
FDIR	fault detection, isolation, and recovery
FKF	federated Kalman filter
FLOP	floating-point operation
FN	false negative
FOV	field of view
FP	false positive
FPGA	field-programmable gate array
FPR	false positive rate
FTC	fault-tolerant control
GRP	generalized Rodrigues parameter
IFAC	International Federation for Automatic Control
iForest	isolation forest
IGRF	International Geomagnetic Reference Field
IQR	interquartile range
iTree	isolation tree
kNN	k-th nearest neighbour
LEO	low Earth orbit
LKF	linear Kalman filter
LOF	local outlier factor
LR	learning rate
LRD	local reachability density
LSTM	long short-term memory
MAE	mean absolute error
MEKF	multiplicative extended Kalman filter
MEMS	micro-electric mechanical system
ML	machine learning
MLP	multilayer perceptron
MRP	modified Rodrigues parameter
MSE	mean squared error
MSLE	mean squared logarithmic error
NATO	North Atlantic Treaty Organization
NN	neural network
OBC	on-board computer
OC-SVM	one-class support vector machine
OOL	out-of-limit
PCA	principal component analysis
PD	proportional-derivative
RAM	random access memory
RBF	radial basis function
RNN	recurrent neural network
ROC	receiver operating characteristic
SGP4	Simplified General Perturbations 4
STK	Systems Tool Kit
SVM	support vector machine
TLE	two-line element set
TN	true negative
TP	true positive
TPR	true positive rate
UKF	unscented Kalman filter
USQUE	unscented quaternion estimator
VAE	variational autoencoder

1 Introduction

In this chapter, the topic of the thesis will be introduced. The relevance of the research carried out in the project is first described in section 1.1. The section aims to place the topic into a broader context, showing the motivation behind the work. This is followed by a review of the relevant literature that is available on this topic in section 1.2, where a research gap is identified and justified. A research objective is developed based on this review, with accompanying research questions that specify the focus of this thesis. The research objective and questions are described in section 1.3. Finally, an outline of the thesis project and this report is provided in section 1.4.

1.1 | **Research Relevance**

To remain operational within the inhospitable conditions of space, an approach in which faults are expected and handled is necessary to make a fault-tolerant control (FTC) system. In space-flight, this is often implemented through hardware redundancy, where the function of a faulty component is transferred to an identical unit. For nanosatellites, such as CubeSats, the use of hardware redundancy is often difficult due to physical and budgetary limitations. Of 159 CubeSats launched before 2014, 86% either did not include redundancy for the electrical-, computing-, or communication system at all, or only for one of these major subsystems [1].

CubeSats are often considered to be less reliable than traditional larger satellites [2]. With the original purpose being education, a general requirement has always been low cost [3]. This drove the use of commercial off-the-shelf (COTS) components, and a reduction in testing prior to launch. In a statistical analysis of CubeSat missions, it was found that besides launch and deployment failures, common issues are communication, power generation, and a high spin rate [4]. The attitude determination and control system (ADCS) can play a role in many of these issues, such as due to improper pointing of antennae, suboptimal solar array pointing, or failure to control the attitude overall. In fact, a reliability study indicated that the ADCS is responsible for 11% of all first-year spacecraft failures, the largest contribution of any single subsystem [5]. Thus, reducing the impact of faults in this subsystem is an opportunity to increase mission reliability.

The attitude determination system (ADS), part of the ADCS, is responsible for ascertaining the orientation of the spacecraft with respect to some reference frame. The system typically relies on a variety of sensors, such as star trackers, Sun sensors, and gyroscopes. A process, sensor fusion, combines the information from these sensors into a single accurate estimate of the spacecraft orientation. When a sensor exhibits faulty behaviour, the performance of the system can be severely degraded. A sensor can, temporarily or permanently, become faulty for a number of reasons, including blinding by sunlight, mechanical degradation, and electrical shorting. Detecting these faults and preventing them from affecting the attitude estimate can greatly improve the reliability of the ADS. This strategy is studied in the engineering field of fault detection, isolation, and recovery (FDIR). This is not typically applied to the ADS or attitude control system (ACS) in CubeSats, mostly due to historically limited computational and power resources [6]. With rapid advancements in CubeSat technology, developing new approaches to applying FDIR in the ADS of CubeSats is a promising path to improving the success rate of missions.

There are many CubeSat missions where faults in the ADS had an impact. Not all of these cases are made public. However, university missions typically share results and findings, including any problems experienced and their effect on the mission. Below is a collection of university missions where faults in the ADS had an impact on the mission:

- The German *COMPASS-1* 1U CubeSat flight results indicate complete failure of the ADS to estimate attitude, leading to the active three-axis control never being put to use, significantly affecting the mission [7]. The cause for the failure of the ADS was twofold. First, the reference magnetic field contained a false rotation, causing the estimation using the magnetometer to fail. Second, the Sun sensors produced corrupted output due to incorrect threshold calibration. Combining these two facts rendered attitude determination impossible, and therefore active control impossible.
- The Estonian *ESTCube-1* satellite suffered from multiple issues with the ADS [8]. Besides a faulty Sun sensor that was not replaced before launch, it was found that the remaining Sun sensors produced errors that showed dependence on the angle with the Sun, influencing the sensor fusion process. Moreover, one of the four gyroscopes started malfunctioning some weeks after launch. The measurements are described as showing jumps between the incorrect and correct values.
- The *SwissCube* mission reported unexpected magnetometer readings [9]. They hypothesize this could be caused by magnetic sources within the spacecraft, or the failure of one of the magnetometer axes. Additionally, two Sun sensors on the same face malfunctioned and started providing constant noise signals.
- After the launch of the Finnish *Aalto-1* mission, it was found that two Sun sensors were not usable for estimation, and the gyroscope data was not being used correctly [10]. This affected the accuracy of the ADCS.

This is only a sample of the missions where the ADS experienced malfunctions, and similar instances are likely not all reported to the public. In any case, a reliable ADS is a necessary basis for the success of many CubeSat missions. Due to the limited potential for hardware redundancy, creating fault tolerance through other means is attractive. With increasing computational capacity on CubeSats, this area is receiving more interest as an approach to reduce the impact of anomalous sensor measurements. This research aims to contribute to this expanding knowledge by investigating novel approaches to make the ADS more fault-tolerant and robust.

1.2 | Literature Review

The ADCS of a spacecraft relies on a variety of sensors and actuators. The system must maintain control even under adverse conditions such as uncertainties, disturbances, and sensor faults. These conditions can severely degrade performance, or even pose a threat to the overall mission [11]. Statistical analysis has shown that 28-32% of failures affect the attitude and orbit control system (AOCS), the highest of any single subsystem [12, 13]. Additionally, a reliability analysis indicated that the AOCS contributes 32% to all small satellite failures in the first 30 days, and 26% after a year [14]. Thus, preventing issues in this subsystem from causing failures can greatly improve mission reliability. In order to maintain stability, reliability, and performance, control systems are developed which can effectively respond to faults. This approach is called fault-tolerant control (FTC) [11].

1.2.1 | Fault Tolerance

A distinction is made between two types of FTC system: passive and active. In passive FTC, a system is made failure-proof for a certain set of faults in the design stage, typically with hardware redundancy [15]. This is not always possible, especially for small satellites such as CubeSats, due to budgetary or physical constraints. The term *passive* indicates that no additional actions are taken by the system in response to faults. Active FTC, on the other hand, seeks to respond to faults as they occur by reorganizing the remaining system elements in real-time [15]. Thus, the term *active* refers to the fact that the system takes actions to adapt to new conditions, be it in hardware, software, or other means.

The two FTC types are illustrated through layout diagrams in Figure 1.1. Note that the active FTC system relies on fault detection and diagnosis (FDD). This refers to the determination of the presence and nature of a fault in the system. The information this provides is then used to decide if action should be taken.

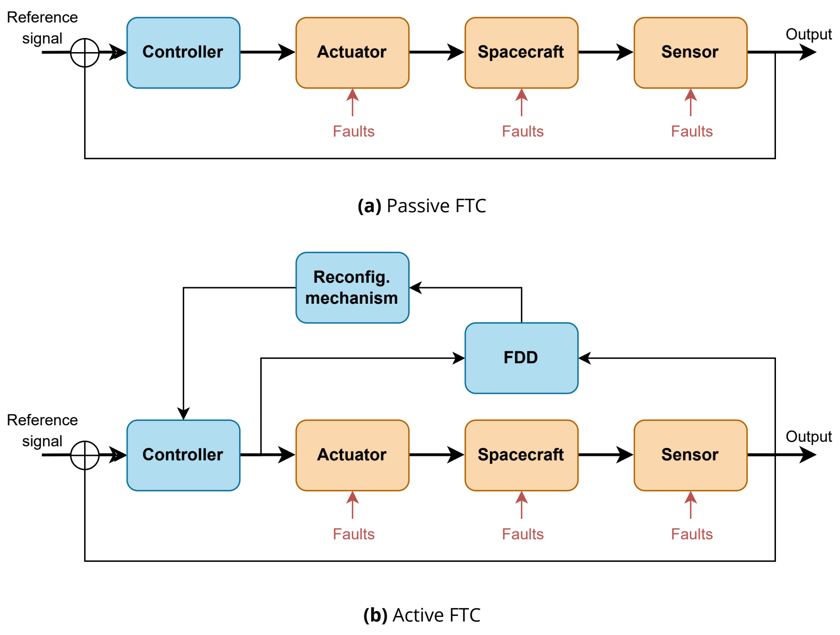


Figure 1.1: Layout comparison of active and passive FTC systems. Based on [11, 15].

One of the advantages of passive FTC is evident from Figure 1.1: the architecture is simple and has low computational demand. Conversely, the active FTC layout requires the implementation of an FDD algorithm and a reconfiguration mechanism. However, passive systems cannot account for faults that they were not designed for, and their performance is never optimal compared to an active approach for all scenarios [15].

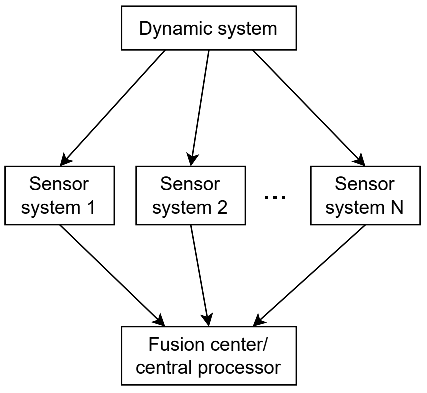
The goal of the active FTC system is thus to achieve all three steps encompassed by FDIR. First, the fault should be detected by a fault detection method. Next, the origin of the detected fault should be determined such that it can be isolated. Finally, the system is reconfigured to recover its functionality while the fault is present.

1.2.2 | Centralized vs. Distributed Sensor Fusion

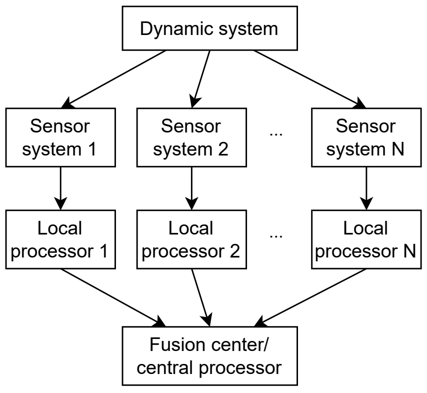
The ADS of a spacecraft typically consists of multiple sensors of different types, spatially distributed throughout the spacecraft to provide the desired viewing angles and coverage [16]. A sensor fusion system is necessary to combine all the observations into a single state estimate. Kalman filters are the most widely used algorithm for this purpose [17]. Improvements in parallel processing power and interest in fault-tolerant systems have lead to new adaptations and variations of the algorithm being developed [18]. The basis for these variations is the linear Kalman filter (LKF), originally developed in 1960 by R.E. Kalman [19].

The Kalman filter is an algorithm that can use a series of measurements to produce an estimate of unknown variables. The algorithm operates in two phases. First, in the prediction phase, the filter produces an estimate of the current state, typically with a physical model. Second, in the update phase, these estimates are updated using the new measurements, weighted based on their certainty. Common variations of the Kalman filter used in the ADS of spacecraft are the extended Kalman filter (EKF) and unscented Kalman filter (UKF), which are both equipped to handle the nonlinearity associated with the spacecraft attitude. Other variants, such as the cubature Kalman filter (CKF), are also used.

The traditional Kalman filtering technique consists of a single filter to process all information, which is a centralized approach [20]. This structure is shown in Figure 1.2a. Since all measurements are processed by one filter, there is minimal information loss and thus good optimality is achieved. Centralized variants of the Kalman filter are widely used for state estimation, spacecraft attitude determination being no exception [17].



(a) Centralized



(b) Distributed

Figure 1.2: Comparison of two approaches to Kalman filtering [21]

Due to the improvement of parallel processing power and interest in fault-tolerant systems, new filter structures were developed which distribute the filtering process over multiple stages, as is seen in Figure 1.2b. Local filters generate local estimates, typically using only a subset of the available information, which for the ADS is a subset of attitude sensors. A master filter combines these local estimates into a global solution. Depending on the architecture of the distributed filter, this can form the globally optimal solution [20].

Since the distributed architecture employs multiple local filters and a master filter, the required total number of computations is generally higher than the centralized approach which uses a single filter. However, it has been shown that running the local filters in parallel reduces the average computation per processor, leading to faster estimations [22]. In an application of a distributed filter for monitoring the health of a gas turbine engine, it was found that the developed distributed filter reduced the computational time by 24.1% compared to a centralized alternative [23].

Besides reducing the computational time, a distributed filter can accommodate sensors with different measurement frequencies by running the local filters asynchronously [18]. By updating at different rates, the local filters can match the speed of the measurements that they receive. The fastest rate does not dictate the speed at which the entire filter should run, only the local filter which processes that portion of the information. This reduces unnecessary computations and can increase throughput compared to a centralized approach.

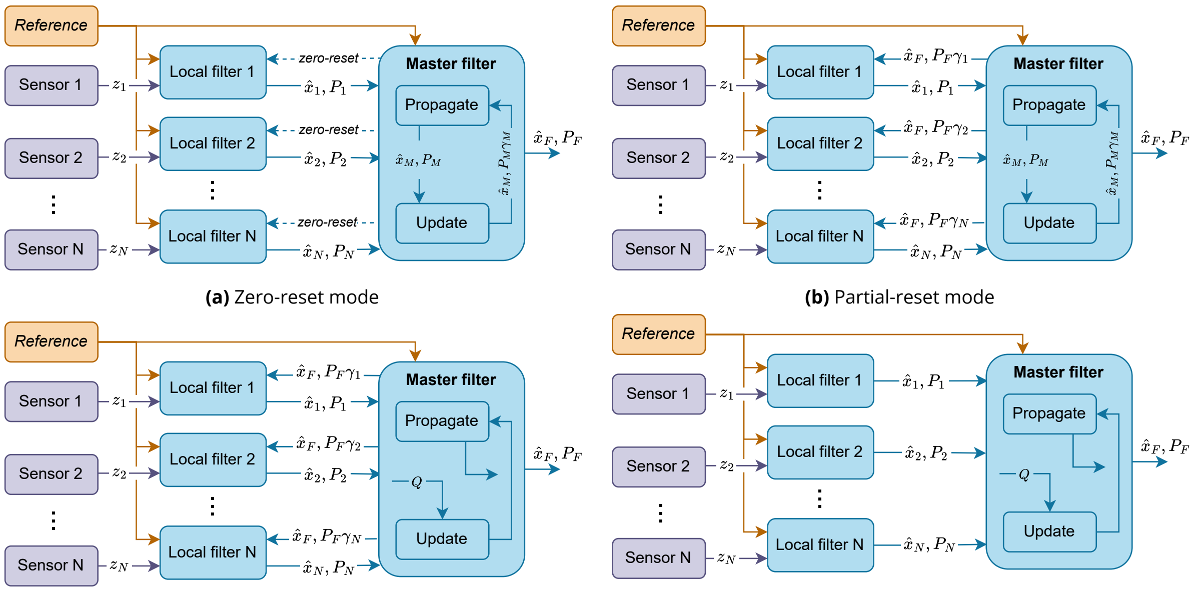
A final benefit of the distributed architecture, especially important for FTC systems, is its capacity for fault isolation [20]. Due to the independence of the local filters, any faults can inherently be isolated. A fault will affect one local filter, but not the other. Thus, by detecting if the output of the local filter is faulty, the associated sensor system is directly known. This improves the overall reliability of the system, since the multiple local filters are usable as backups [18].

1.2.3 | Federated Kalman Filter

A specific variant using the distributed approach is the federated Kalman filter (FKF). The foundational difference with other distributed methods is the information sharing principle. It was developed to improve the accuracy of the distributed filter, without compromising the fault detection capabilities [20]. Previous distributed filters, such as a cascaded filter, were subject to poor accuracy and even divergence [18]. This was due to local filter outputs not being sequentially random, and the fact that the local filter output and master filter output were not independent. The development of the FKF sought to improve this behaviour through a new principle: the information sharing principle. A formal definition is given by [24]:

The total system information can remain constant or decrease, but never increase, due to sharing.

In other words, double-counting of the same information should be avoided. This principle allows the master filter to treat the local filter estimates as statistically independent such that the globally optimal solution can be found [25]. Different strategies can be used to comply with this principle, four of which are primarily considered [18]: the full-, partial-, zero-, and no-reset modes. They differ in the method with which the total information is divided over the local and master filters. A comparison of the structure of a federated filter using the different modes is provided in Figure 4.2.



(a) Zero-reset mode

(b) Partial-reset mode

(c) Full-reset mode

(d) No-reset mode

Figure 1.3: Structures of federated Kalman filter in zero-, partial-, full-, and no-reset mode. Based on [18].

Due to its capacity for fault tolerance, the no-reset mode is most suitable for FTC systems [18]. In this mode, faults that affect one local filter have no path to affect any other local filters. This makes it trivial to exclude faulty elements from the master filter. In other modes, faults in one local filter would affect the others through the feedback mechanism, which is avoided if no feedback is present.

The FKF has been applied to CubeSat systems in various studies, such as formation flying satellites [26], positioning [27], as well as attitude determination [28, 29].

1.2.4 | Conventional Anomaly Detection

Traditionally, anomaly detection is performed through out-of-limit (OOL) checks with set upper and lower bounds [30]. Multiple variations of this approach exist, for example the use of soft limits which may temporarily be exceeded, with a hard limit as the ceiling. The literature involving the FKF for attitude determination specify two conventional methods of fault detection [31, 28]: the sensitivity factor and the measurement residual.

The sensitivity factor is a value similar to the Mahalanobis distance. It quantifies how well the difference between a local and master filter is explained by the uncertainty in the system [28]. This provides a numerical score to which a threshold can be applied, distinguishing nominal from anomalous points.

The second conventional method, the measurement residual, relies on a quantity calculated in the local filter itself. As part of the Kalman filter, the next measurement is predicted. The difference between this prediction and the observed measurement then serves as an indicator of anomalous behaviour, where a large difference indicates an unexpected measurement. Again, this provides a numerical score to which a threshold is applied.

Both of these methods rely heavily on the statistical properties of the filters, and do not take any temporal context into account. They only consider the current sample to judge whether an anomaly has occurred. This makes them less effective when attempting to detect gradual or subtle faults [6]. These relatively simple checks do not analyse patterns in the underlying data, and are therefore bound to miss anomalies [30]. Finally, the selection of a threshold presents a trade-off between detection speed and accuracy, and becomes difficult if no operational data is available.

1.2.5 | ML-based Anomaly Detection

Technology miniaturization allowed the space industry to build CubeSats using low-cost COTS components [32]. With the increasing processing power of COTS processors, the functional software requirements also increased, including the development of FDIR programs [33]. In a review of on-board computing in CubeSats, the application of artificial intelligence (AI) is specifically noted to be promising, including its application to FDIR functions [33].

Machine learning (ML), a subset of AI focused on learning from data, can be used to detect anomalies and malfunctions on-board the spacecraft. A neural network (NN) trained on simulation, or existing data if it is available, can identify faulty behaviour that might otherwise go undetected [30]. The historical limitations of computing and memory resources associated with CubeSats have been loosened, especially when hardware accelerators are implemented [33]. In contrast to the conventional detection methods, a data-based method is well-suited to learn and apply pattern recognition to detect more complex anomalous behaviour [30, 34].

Unsupervised ML-based anomaly detection has seen rapid advancement in recent times [35]. Unsupervised methods are more suited to the space industry than supervised or semi-supervised methods due to the general lack of available data containing anomalies or with annotations [36]. Additionally, the types of faults to be detected do not have to be limited in the design phase, improving the adaptability of the system to new behaviour.

Popular methods include the one-class support vector machine (OC-SVM), isolation forest (iForest), and local outlier factor (LOF) [35]. They are for example used for anomaly detection in spacecraft telemetry streams [37]. The LOF is a distance-based method that scores anomalies by how isolated they are from their local neighbourhoods [38]. The OC-SVM learns a mathematical boundary around the nominal data and flags points far away from this limit [39]. The iForest instead uses a set of binary trees where anomalies tend to be isolated in fewer splits [40].

Newer approaches use recurrence, such as in the long short-term memory (LSTM) approach, to inherently consider the temporal context [35]. With an LSTM predictor, the sample that follows a sequence is predicted, where the prediction error acts as an anomaly score. An LSTM autoencoder (AE) attempts to reconstruct the sequence, where the reconstruction error is used instead. These methods can learn temporal dependencies to detect even gradual or subtle faults, making them suitable candidates for time-series anomaly detection [35]. These LSTM-based methods, combined with other ML-based methods, are promising for application within the FKF.

1.2.6 | Research Gap

Table 1.1 lists a collection of published applications of fault detection and FKFs to the AOCS. First, it is noted that the research into the use of FKFs in the ADS is limited. Some studies focus on the application of the FKF solely, while others include fault tolerance. A common element is that when a fault detection method is implemented within an FKF, a simple OOL-based statistical technique is used. Specifically, the sensitivity factor or the measurement residual are used.

When different detection techniques are applied, including ML-based approaches, the FKF, or the sensor fusion approach in general, is not considered. The focus lies solely on the detection of faults. The detection methods listed in Table 1.1 show a great variety of possible approaches that are used across the AOCS. However, no literature could be identified which combines an ML-based fault detection method with the FKF for FDIR in a space application.

Table 1.1: Collection of applications of fault detection and FKF to AOCS-related components

Source	Detection method	Sensor fusion	System
[28]	Sensitivity factor / measurement residual	FKF (UKF)	ADS
[41]	Chi-square test + generalized likelihood ratio	Chained EKF	ADS
[42]	Luenberger observer Elman NN observer	—	Reaction wheels
[43]	Recurrent NN observer	—	Reaction wheels
[44]	Deep NN	—	ACS
[45]	Principal component analysis + statistical measures	—	ACS
[46]	Principal component analysis + statistical measures	—	Actuators
[47]	Kernel principal component analysis	—	Reaction wheels
[48]	T-S fuzzy model	—	Reaction wheels
[49]	Fuzzy adaptive estimator	—	Coupled spacecraft
[50]	Fuzzy fault tree	—	Flywheel
[51]	Convolutional NN	—	Attitude sensors
[52]	—	FKF (UKF)	ADS
[31]	Sensitivity factor	FKF	ADS
[53]	—	FKF (adaptive LKF)	ADS
[54]	Measurement residual	FKF (CKF)	Navigation

To summarize, the conventional detection methods do not analyse behavioural patterns, limiting their accuracy. Instead, ML-based methods offer several advantages, and have been shown to detect faults not apparent through conventional means [6]. While these methods come with their own limitations, such as increased computational load and the need for large amounts of data, the potential benefits make them an attractive option for improving fault detection in CubeSats. Finally, the FKF provides a fusion structure that inherently provides isolation of sensor systems while potentially reducing runtime and increasing information throughput. Thus, combining ML-based fault detection with the FKF is a concrete path to increasing the robustness of the ADS.

1.3 | Research Objective and Questions

This thesis aims to study the application of ML to FDIR in an FKF. Specifically, the output of the local filters, each of which corresponds to a subset of the sensor suite, is used to detect anomalies. The fault detection method relies on a trained classifier to determine if the output shows faulty behaviour. Only non-faulty local estimates are then included in the master fusion step.

This approach has the potential to detect more complicated fault scenarios than traditional approaches, such as OOL checking on statistical properties. The decentralized filter provides inherent isolation of sensor subsets, which can be used to create a fault-tolerant system. Improvements in the COTS hardware, which is used in CubeSats, provides the opportunity to implement such techniques, overcoming the historical challenge of limited computational resources.

The objective of this thesis is then summarized into the following statement:

Main Research Objective

The objective of this research project is to improve the reliability of the attitude determination system in a CubeSat by applying a machine learning-based classifier to fault detection within the sensor fusion process making use of a federated Kalman filter.

Of importance is the feasibility of applying a machine learning-based classifier in this context. Specifically, whether it is possible to replace simple fault detection methods with such a solution to provide better fault detection and isolation. Additionally, the limited hardware on most CubeSats should be taken into account. This leads to the following main research question:

Main Research Question

What is the accuracy and computational impact of a machine learning-based classifier in performing fault detection within a federated Kalman filter for CubeSat attitude determination?

The main research question is subdivided into the following sub-questions:

- **RQ1** Which ML-based methods are suitable for fault detection in an FKF?
- **RQ2** How effectively can ML-based classifiers distinguish between faulty and non-faulty subfilter outputs in an FKF?
- **RQ3** How accurate are ML-based classifiers in detecting faults within an FKF compared to conventional methods?
- **RQ4** What are the computational costs of deploying ML-based classifiers for fault detection in an FKF on CubeSat-class hardware?

The sub-questions above provide structure to the research project. First, method selection is considered in **RQ1**, which aims to identify suitable machine learning methods in this context. Next, **RQ2** and **RQ3** relate to the effectiveness of the developed solution, comparing the performance to reference methods. This is followed by a consideration of the limited capabilities of the hardware on a typical CubeSat in **RQ4**. The computational cost of the FKF with a machine learning-based classifier should be compared to standard solutions to determine the feasibility of using this approach on CubeSats.

1.4 | Thesis Outline

This report is divided into three parts. First, Part I gives the theoretical background required for the remainder of the report. This consists of a brief overview of the hardware comprising the ADS of a typical CubeSat (chapter 2), a description and taxonomy of fault tolerance methodology (chapter 3), an introduction to (distributed) Kalman filtering for attitude determination, specifically the FKF (chapter 4), and the metrics used to score detection performance (chapter 5).

Next, the developed fault detection system is described in Part II. This includes the simulation that was set up to evaluate sensor faults and their detection (chapter 6), as well as the selection and tuning of conventional and ML-based detection methods (chapter 7 and 8 respectively).

In Part III, the results of the research are presented and discussed. This includes a comparison of the responses to faults of the various detection methods (chapter 9), as well as the computational impact each method has on CubeSat hardware (chapter 10).

The conclusion of the research is presented in chapter 11, combined with recommendations for further research. In Appendix A and B, detailed descriptions of quaternion-based variants of the UKF and EKF are provided respectively. Finally, Appendix C shows the implementation of an attitude controller for tracking ground-based targets.

Part I

Theoretical Background

This thesis combines multiple disciplines, such as sensor fusion, fault tolerance, and machine learning. In order to provide context and relevant knowledge, this part of the report describes several important topics related to the research. Included are descriptions of the typical hardware in the attitude determination system (chapter 2), an introduction and overview of fault tolerance methodology (chapter 3), a more detailed illustration of the various Kalman filtering approaches (chapter 4), and the metrics used to score the performance of fault detection methods (chapter 5).

2
Attitude Hardware

A necessary component of the ADCS is the ability to observe and measure the attitude and angular rates of the spacecraft, as well as process their data into usable products. A variety of sensor systems have been developed that provide the ADCS with information about the current spacecraft state. This chapter describes various common sensor types and processing solutions. In general, attitude sensors are divided into two categories [55]: measuring the absolute attitude with respect to some external reference vector (section 2.1), or measuring centrifugal acceleration to determine the change in orientation (section 2.2). Their basic principles are modelled in chapter 6 for realistic fault injection, based on their functioning described in this chapter. Also given is a collection of typical computing units in section 2.3. This collection is used to assess the computational impact of the various fault detection methods in chapter 10.

2.1 | Absolute Sensors

Absolute attitude sensors measure the current attitude of the spacecraft in a specific reference frame. The most common sensors of this type are the star tracker, sun sensor, and magnetometer [55]. These are briefly described in the following sections.

2.1.1 | Star Trackers

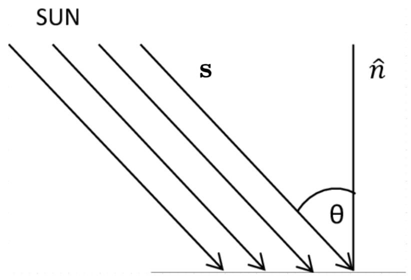
Star trackers rely on imaging the celestial sky to identify known stars, deducing the spacecraft attitude from their positions with respect to the sensor [55, 16]. The sensor typically consists of a digital camera and a processing unit. Star centroids are identified within the image and compared to an internal star catalogue. Once found, the orientation of the sensor with respect to the celestial reference frame can be determined.

Star trackers typically operate in two modes: lost-in-space and tracking. In lost-in-space mode, initial attitude acquisition is performed without prior knowledge. Once the attitude is determined, which typically takes a few seconds [16], the sensor can move into tracking mode. The next measurement can now use the knowledge of the previous measurement to speed up identification, increasing the update rate. Additionally, some star trackers measure the angular velocity of the spacecraft in addition to the absolute attitude.

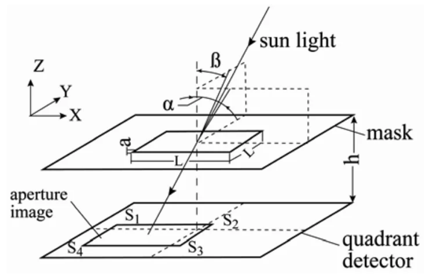
2.1.2 | Sun Sensors

Sun sensors measure the direction of incoming light, making it possible to deduce the attitude of the spacecraft based on the known position of the Sun. Generally, two types of Sun sensor are used: coarse and fine [55, 16]. Both require knowledge about the current position of the Sun, typically computed with an on-board algorithm, to determine the spacecraft attitude. A limitation of either type is that Sun sensors stop functioning when the spacecraft is in eclipse.

Coarse sensors are typically relatively simple analogue detectors that measure the angle of incoming light [16]. A photocell produces current proportional to the intensity of the light falling on it, as is depicted in Figure 2.1a. Light coming in perpendicularly to the surface of the photocell produces the highest current, while shallow angles produce lower current. Multiple coarse sensors can be combined to provide larger coverage and estimate the Sun vector direction [55]. In [16], a configuration of six coarse Sun sensors is proposed, with pairs of opposing normal directions. If these unit normals do not describe coplanar planes, the measured Sun vector can be computed.



(a) Cosine detector Sun sensor [55]



(b) Two-axis fine Sun sensor [56]

Figure 2.1: Comparison of Sun sensor principles

Fine Sun sensors, or digital Sun sensors, historically used linear photosensitive strips underneath arrays of slits or apertures [55]. Newer designs, however, are comparable to a star tracker in that they use an imaging module, but with a pinhole instead of other optics in front of them [16]. A possible approach is shown in Figure 2.1b, where incoming light falls onto one or more quadrants. In this configuration, analogue photocells are still used with specific placement. These can also be replaced with a digital sensor, which typically provide higher accuracy [16].

2.1.3 | Magnetometers

Magnetometers observe the magnitude and direction of the local magnetic field and compare this to a reference model in order to generate a vector attitude measurement [55]. Besides being lightweight and low power, magnetometers do not need an external view. However, due to the limited accuracy of the reference model and other error sources, they are not always accurate. Since the field strength decreases with distance, their use is typically limited to spacecraft below 1000 km, before magnetic biases dominate the measurement [55]. Most magnetometers are fluxgate magnetometers, which have a permeable core which is alternately magnetically excited. The flux caused by the Earth's magnetic field consistently acts in the same direction, which can be detected.

The reference system that is typically used to model the Earth's magnetic field is the International Geomagnetic Reference Field (IGRF) model [55]. This model expresses the magnetic potential function in terms of the position of the spacecraft. This field is then typically transformed to the Earth-centered inertial (ECI) or Earth-centered Earth-fixed (ECEF) frame when used in the ADS. The internal part of Earth's magnetic field varies slowly, which is also accounted for in the model by determining the time derivatives of the coefficients. There are short-term variations as well, such as due to electric currents in the ionosphere and magnetosphere, but these are not included [16]. Magnetometers can also suffer from drift over long time spans due to mechanical deformation degrading their accuracy, but this is typically not included in a measurement model due to the difficulty in modelling and its small influence on the overall magnetic field measurement [57].

2.2 | Gyroscopes

Gyroscopes, or gyros, are sensors which measure the angular velocity of a spacecraft. Historically, all gyros used a spinning mass which tends to remain fixed due to its angular momentum. The torque required to keep the angular momentum constant then gives a measure of the angular velocity of a spacecraft [16]. Due to the complexity of this approach, and the common failure of gyros due to wear and tear, other techniques were developed to measure the angular rates, specifically optical and Coriolis vibratory gyros.

Optical gyros make use of the Sagnac effect, which is a phase difference between two light beams travelling in opposite directions around a closed path that is caused by the path rotating [16]. The gyroscope observes this phase difference to estimate the angular rates. Since light travels extremely quickly, the phase difference is very small. To measure it, gyros can make the light travel around the closed path many times to magnify the difference, or an active laser medium can be used to transform the observation into a frequency measurement.

Coriolis vibratory gyros use the principle of Coriolis forces, which induces a vibrational mode in a structure that can be detected [16]. Various types exist, but gyros using micro-electric mechanical system (MEMS) technology are a new development which have become popular in CubeSats due to their small form factor and low power requirements, though they are typically less accurate than other variants [16].

2.3 | On-board Computers

The type of processing unit on a CubeSat can vary greatly, including microcontrollers, microprocessors, field-programmable gate arrays (FPGAs), or hybrid systems which could include a GPU [33]. The type of processor that the CubeSat on-board computer (OBC) possesses is the main factor determining the processing capabilities, speed, and power consumption of the unit. In this section, a description and collection of COTS units is provided, which is subsequently used in chapter 10 to assess the computational impact of a detection method.

Microcontrollers have traditionally been the preferred option, especially for early CubeSats missions with shorter durations and lower processing requirements [33]. These COTS components are small, low-power, and easily available. The most common architecture is those based on the ARM Cortex family of processors. There is a wide range of devices with different performance levels to be selected based on mission requirements.

Alternatives include the microprocessor, specifically radiation-hardened versions. With decades of flight heritage on larger missions, these are very reliable [33]. However, their use in CubeSat missions has been limited, largely due to the significant financial investment required for the development of suitable versions. Another alternative is the FPGA, which differs from traditional processors due to the use of reconfigurable logic gates that can perform multiple tasks simultaneously. Radiation-hardened versions have been shown to provide acceptable protection without the need for expensive radiation-hardened microprocessors. Finally, hybrid architectures have emerged that greatly improve performance, sometimes in one specific domain. This includes the use of hardware accelerators for on-board ML applications, though these increase power consumption.

To evaluate whether the methods selected in this research project could feasibly run on a CubeSat, the traditional microcontroller is used as the baseline. These efficient, light, and cost-effective processing units are commonly used in CubeSat missions, specifically those in low Earth orbit (LEO) or with shorter mission durations [33]. They are typically the least performant system out of the aforementioned architectures, and thus the most constraining option. In Table 2.1, a collection of COTS OBCs based on microcontrollers are collected, combined with their processing unit, memory sizes, and power consumption. Note that the listed power consumption cannot be directly compared due to differing specifications.

Table 2.1: Collection of COTS CubeSat OBCs based on microcontrollers [33, 58]

OBC	Processing unit	Memory	Power consumption
TRISKEL by Alén Space	32-bit ARM Cortex-M7	• 2 MB Flash (program) • 1.4 MB SRAM • 4 MB MRAM • 1 Gb NAND Flash (data)	6 W (peak)

Continued on next page

Table 2.1: *continued from previous page*

OBC	**Processing unit**	**Memory**	**Power consumption**
OBC by EnduroSat	32-bit ARM Cortex-M7 (STM32)	• 2 MB (program) • 1 MB SRAM • 8 Mb MRAM • 2 MB NAND Flash (data)	1.5 W (peak)
NanoMind A3200 by GomSpace	32-bit RISC (AVR32)	• 32 MB SRAM • 32 kB FRAM • 128 MB Flash (data)	0.9 W
OBC by IMT	32-bit MIPS (PIC32MZ)	• 16 MB SRAM • 64 MB NOR Flash (housekeep) • 8 GB NAND Flash (data)	300 mW
SatBus 3C2 by NanoAvionics	32-bit ARM Cortex-M7 (STM32)	• 1 MB integrated RAM • 2 MB integrated flash • 256 MB NOR Flash (data) • 2×512 kB FRAM (data)	N/A
OBC-P4 by Space Inventor	32-bit ARM Cortex-M7 (SAME70)	• 384 kB SRAM • 32 kB FRAM (app) • 64 GB eMMC	N/A
Deep Thought by Spacemanic	32-bit ARM Cortex-M7 (SAMV71)	• 2048 kB Flash (memory) • 384 kB SRAM • 128 MB Flash (storage)	100 mW (avg)
iOBC by ISISPACE	32-bit ARM9	• 1 MB NOR Flash (code) • 64 MB SRAM • 256 kB FRAM • 4 GB Flash (storage)	400 mW (avg)

From this collection, the available working memory in the OBC varies between hundreds of kilobytes to tens of megabytes, and the long-term storage between hundreds of megabytes to several gigabytes. Most OBCs in this collection are based on the ARM Cortex-M7 processing unit, and all are a 32-bit platform. The throughput of the processor depends on several factors, including the clock frequency. A study found an ARM Cortex-M7 microprocessor running at 550 MHz to achieve a throughput of approximately 70 MFLOPs per second under optimal conditions [59].

The collection of microprocessor-based OBCs above is used as a baseline to assess the computational impact of various detection methods in chapter 10.

3 Fault Tolerance

To maintain stability, reliability, and performance even when components malfunction, a fault-tolerant control (FTC) approach is used [11]. Any minor fault could cause a large disruption of functionality, even risking complete failure of a spacecraft. Creating a system that is tolerant to these faults is necessary for reliable operation.

This chapter first provides an overview of the relevant terminology in section 3.1. A taxonomy of fault detection methods and typical faults is provided in section 3.2. This is followed by a more detailed description of ML-based fault detection methods in section 3.3.

3.1 | Terminology

It is important to clarify the relevant terminology used in fault tolerance. The distinction between concepts can affect the meaning of literature, which is spread across various fields and industries. Standardization efforts, such as by an advisory group to the North Atlantic Treaty Organization (NATO) [60], have lead to generally accepted terminology [61]. In this thesis, the terminology as proposed by the International Federation for Automatic Control (IFAC) in [62] is used. First, there are the terms *fault* and *failure*, which are often used interchangeably, which is not accurate. Their definitions are given by [62]:

Fault An unpermitted deviation of at least one characteristic property or parameter of the system from the acceptable/usual/standard condition.

Failure A permanent interruption of a system's ability to perform a required function under specified operating conditions.

Thus, a *fault* and a *failure* are distinct terms. In other words, a fault is any abnormal condition that affects a system, which in turn can lead to a failure where a functional unit can no longer perform its required function [60]. In order to prevent a fault from turning into a failure, various methods can be applied, also defined by [62]:

Fault detection Determination of the faults present in a system and the time of detection.

Fault isolation Determination of the kind, location, and time of detection of a fault. Follows fault detection.

Fault identification Determination of the size and time-variant behaviour of a fault. Follows fault isolation.

Fault diagnosis Determination of the kind, size, location, and time of detection of a fault. Follows fault detection, includes fault isolation and identification.

The first step in preventing a fault from leading to a failure is to detect that one has occurred. This is followed by a thorough diagnosis of the fault to determine its location, known as isolation, as well as its nature, known as identification. These steps combined are referred to as fault detection and diagnosis (FDD). The term fault detection, isolation, and recovery (FDIR) includes an additional step: recovery. This is the action which returns the system to a stable state, for example by switching off a faulty sensor [63].

There are more distinctions used throughout the field of FDIR, such as various fault classes and system metrics [60]. For the purposes of this thesis, the terminology outlined in this section is sufficient.

3.2 | Fault Detection

As aforementioned, an important component of FTC systems is the use of fault detection (FD) algorithms. The purpose of these methods is to determine that a fault has occurred. Though there is some inconsistency in literature [64], the FD methods can in general be categorized into three groups: signal-based, model-based, and data-based methods [64, 65, 66]. Hybrid techniques that combine elements from these categories also exist.

This section starts with an overview of the various types of faults that might occur in a sensor system, in subsection 3.2.1. This is followed by descriptions of the three categories of FD methods in subsection 3.2.2.

3.2.1 | Fault Types

There are multiple types of faulty behaviour a sensor can exhibit. The fault types can be broadly divided into two categories: abrupt and incipient [61, 67, 68].

Abrupt Faults

Abrupt faults appear suddenly and are typically significant, rapid deviations from the normal readings. The main contributor to this fault type is physical damage to a sensor system [67]. Since the deviations are usually obvious, they can typically be detected through simple methods. Fault types that belong to this category are [68]:

Spike Sudden, significant, and short-lived increase or decrease in sensor output.

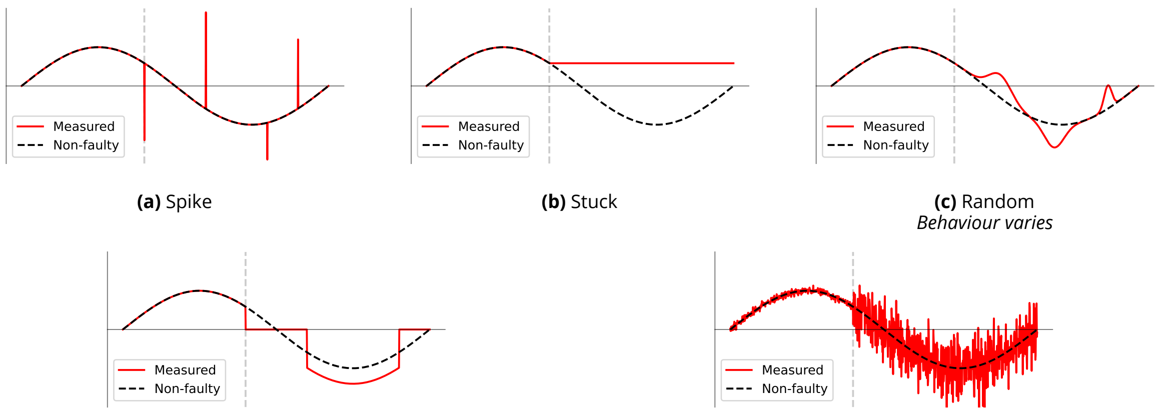
Stuck Sensor output becomes unresponsive and remains fixed in one state.

Random Unpredictable fluctuations in sensor output that are not consistent or systematic.

Short/open circuit Sensor hardware failure, typically indicated by unusually high or low output.

Noise Unusually or unacceptably high noise on the sensor output.

In Figure 3.1, the behaviour for each of these fault types is visualized. Note that this is an example not based on real data. Especially the output during a random or short/open circuit fault can vary greatly from sensor to sensor.



(a) Spike **(b)** Stuck **(c)** Random *Behaviour varies*

(d) Short/open circuit *Behaviour varies* **(e)** Noise

Figure 3.1: Examples of abrupt fault types

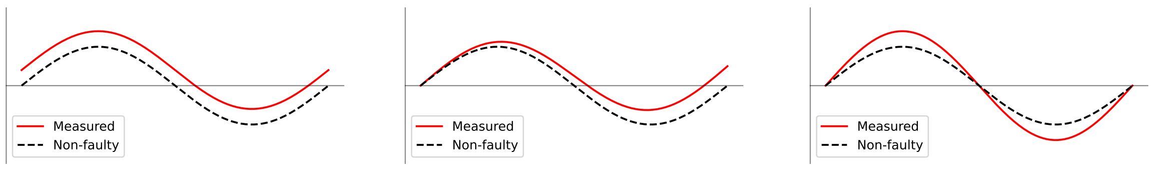
Incipient Faults

Incipient faults are gradual deviations that develop slowly over time [67]. This type of fault is harder to detect, especially in early stages [67, 68]. Fault types that belong to this category are listed below [68]. They are again visualized, in Figure 3.2.

Bias Sensor output has a consistent offset from the true value.

Drift Sensor output has a time-varying offset from the true value.

Gain Sensor gain changes leading to too high or too low sensor output.



(a) Bias **(b)** Drift **(c)** Gain

Figure 3.2: Examples of incipient fault types

3.2.2 | Detection Methods

As aforementioned, FD methods are generally grouped into three categories: signal-based, model-based, and data-based methods [64, 65, 66]. They are described in the following sections.

Signal-based Detection

Signal-based FD relies on the features of a measured signal. The faults in the system are reflected in these features, which can be used to make a diagnostic decision based on prior knowledge [61, 69]. The methods that rely on signal analysis are typically divided into three domains: time, frequency, and time-frequency [61, 69].

Time domain analysis is a natural approach for continuous dynamical processes. The most common technique is the use of thresholds, also called out-of-limit (OOL) checking [70, 71]. A pitfall is the knowledge that is required to set the values of the thresholds. Even with the necessary expertise, the signal properties may be unknown before actual operation [71]. Techniques in the frequency domain include Fourier transformations, typically applied for vibration analysis [69]. The two domains can be combined for signals which are transient and dynamic in a given time period [69].

Model-based Detection

Model-based detection was first described in 1971 by Beard [72]. The purpose was to replace hardware redundancy with analytical redundancy. Instead of carrying additional copies of hardware components, mathematical models could be used to detect faults. These methods rely on a mathematical or qualitative model of the process or system. The consistency between the measured and predicted output serve as the basis for the detection of anomalies. Model-based methods are generally divided into quantitative or qualitative techniques [64].

Qualitative models express the input-output relationship in qualitative terms [64]. This mainly consists of causal models and abstraction hierarchy, such as bond graphs or fault trees [73, 74]. Quantitative models describe the relationship in numerical terms. They are typically divided into state estimation, parameter estimation, and the parity space method [64]. Wide use is made of state estimation in attitude determination for spacecraft, specifically through Kalman filtering [75, 76]. Here, the measurement residual is a typical metric used for detection of faults.

Data-based Detection

Finally, data-based detection, also known as knowledge-based detection, relies on prior experience or data to discern whether a fault has occurred [65, 77]. Similar to the model-based methods, they are generally divided into qualitative and quantitative approaches [64, 77].

Qualitative data-based methods use human-readable, descriptive information to identify issues in a system [64]. The most common technique in this category is the expert-systems method, which represents a human's expertise as a set of rules [77]. Quantitative methods, on the other hand, essentially solve a pattern recognition problem [77]. They are largely divided into statistical and NN-based methods. Statistical methods include principal component analysis (PCA), partial least squares, and support vector machines (SVM). NN-based methods have the ability to approximate nonlinear functions with high accuracy, and can adapt based on training data [77]. These methods are generally categorized by their topology, which can be radial basis, recurrent dynamic, self-organizing, back propagation, and extension networks [77]. This methodology has seen wide application, also in the space domain [64]. Due to their relevance for this research project, ML-based approaches are discussed in more detail in the next section.

3.3 | ML-based Fault Detection

With an overall increase in data collection across all fields, research into automated analysis of this data has become more popular in recent years [35, 78]. Discerning whether a set of data is normal or anomalous is a difficult task due to the large variety of possible anomalies that can occur. In general, the ML-based methods for detecting these anomalies in time-series data can be divided into three categories: unsupervised, semi-supervised, and supervised methods [35, 78].

Supervised methods require a collection of data where anomalous behaviour is labelled, allowing detection of patterns and abnormal sequences. Semi-supervised methods combine labelled and unlabelled data, often specifically with only labelled normal behaviour. This approach is a commonly used strategy in anomaly detection literature [78]. Finally, unsupervised methods do not require any labelled data, but instead learn to recognize normal data patterns, thus classifying unexpected sequences as anomalous. For anomaly detection, unsupervised methods are often used when annotating large sets of data is difficult, and to avoid the risk of overfitting to specific faults in supervised methods [35].

A taxonomy of anomaly detection methods for time-series data is given in Figure 3.3. Three main categories are identified: distance-, density-, and prediction-based methods. Distance-based methods rely on a distance measure, such as Euclidean distance, to classify faulty behaviour between sets of samples. Density-based methods aim to measure the density of points in some representation such as graphs, trees, or histograms. Prediction-based methods detect anomalies by predicting the normal behaviour and comparing it to the observed behaviour.

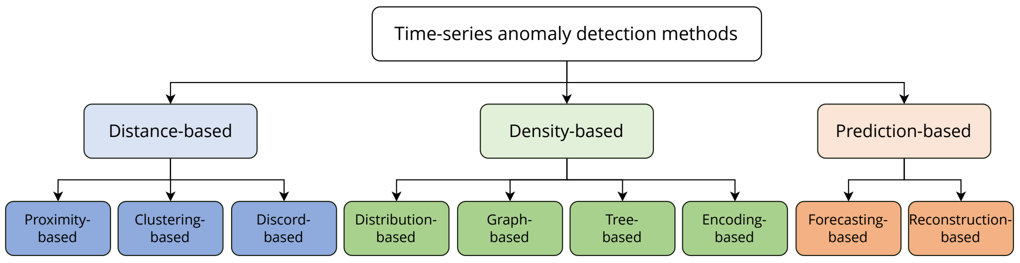


Figure 3.3: Time-series anomaly detection method taxonomy [78]

Each of these categories can be further divided into different groups. The following sections introduce these groups for distance- (subsection 3.3.1), density- (subsection 3.3.2), and prediction-based (subsection 3.3.3) methods.

3.3.1 | Distance-based Detection

Under the distance-based methods, algorithms are divided into three main types: proximity-, clustering-, and discord-based. These methods detect anomalies by computing and comparing distances between points or sequences in the data [35].

Proximity-based methods use the distance of a sample to a neighbourhood to distinguish normal from anomalous behaviour. A typical algorithm used in this category is the k-th nearest neighbour (kNN), which identifies collections of instances that are in proximity [78]. A common method that uses the kNN approach is the local outlier factor (LOF). This method computes a metric which measures the degree to which an instance is an outlier, given other instances. By applying a threshold, instances that deviate from the other observations are classed as anomalous.

Clustering-based methods create partitions of similar instances or sequences, then evaluate how a new instance fits into these partitions. A widely used method is the K-means method, which divides a dataset into multiple clusters [78]. The larger the distance of a new instance to the existing partitions, the more abnormal it is considered to be.

Finally, discord-based methods use an extension of the kNN technique, where now the distances to the k-th neighbour are compared, instead of the nearest neighbour. This is helpful for cases in which a few anomalies are clustered closely to the normal data points [78].

3.3.2 | Density-based Detection

Density-based methods use an alternative representation of the data to extract information, including graphs, trees, and histograms. These models are further divided into four groups: distribution-, graph-, tree-, and encoding-based methods [35].

Distribution-based methods build a distribution from statistical features. The models attempt to learn the statistical model behind the distribution of points or sequences, and infer whether a point is anomalous based on this. A commonly used method in this category, and in general, is the one-class support vector machine (OC-SVM) [35]. In this approach, instances are separated from some origin. Points that are far away from the separation hyperplane are considered anomalous.

Tree-based methods organize data into trees in order to isolate instances that are anomalous. The most common method in this category is the iForest [78]. In this technique, features are randomly split into partitions, each corresponding to a branch of the tree. Anomalies are more likely to be isolated quickly, and are located at a lower depth within the tree.

Graph-based methods represent time-series as a graph. The nodes and edges represent different types of points or sequences, and their change over time. Encoding-based methods use a latent space representation of the data. Anomalies are determined directly from these encoded sequences.

3.3.3 | Prediction-based Detection

Prediction-based methods determine what the expected behaviour should be based on training data. The error with the observed sequence then serves as an anomaly score. Within this category, methods are further divided into two groups: forecasting- and reconstruction-based [78].

Forecasting-based methods use a model to predict future points or sequences based on previous instances, directly incorporating the temporal context. The forecasted instance is then compared to the observed instance, yielding a prediction error as the anomaly score. A simple approach is the ARIMA model, whereas a more intricate technique is found in the long short-term memory (LSTM) network [35]. This latter approach is a type of recurrent neural network (RNN) which retains hidden information between steps for use during prediction.

Reconstruction-based methods are similar to forecasting-based methods. However, instead of forecasting the next point, they compress a sequence into a lower dimension, then attempt to reconstruct it [35]. The difference between the reconstructed and observed sequence is then the degree to which it is considered anomalous. An example of a method in this group is the autoencoder (AE), which attempts to learn the best latent space representation of a set of data and how to decompress that data to resemble the original. This can be expanded by using a variational autoencoder (VAE) for better representations, or combined with LSTM to better incorporate historical data [35].

3.4 | Highlighted Methods

Due to their relevance in this research project, five ML-based anomaly detection methods are highlighted in this section. These methods were introduced in section 1.2, and their specific selection is justified in more detail in section 8.1. The methods are the OC-SVM (subsection 3.4.1), the iForest (subsection 3.4.2), the LOF (subsection 3.4.3), the LSTM predictor (subsection 3.4.4), and the LSTM-AE/VAE (subsection 3.4.5).

3.4.1 | One-class Support Vector Machine

The one-class support vector machine (OC-SVM) is a boundary-based anomaly detection method that learns a decision function describing the region in feature space where the majority of points are located. Introduced in 2001 [39], this method differs from other SVM which rely on labelled data, whereas the OC-SVM is able to train on unlabelled, nominal data.

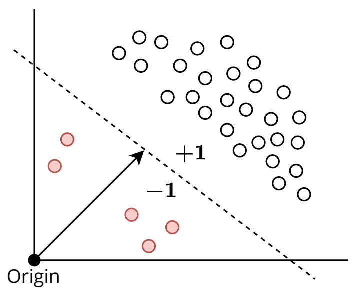


Figure 3.4: Visualization of hyperplane separation in OC-SVM. Based on [79]

The OC-SVM identifies a function that returns a positive value for regions containing most of the data, and a negative value everywhere else [39]. This is typically achieved by mapping the data into a high-dimensional feature space using a kernel function, and then finding a hyperplane that best separates the mapped data from the origin. Points far from this boundary are then considered anomalous. In Figure 3.4 the approach of the OC-SVM is illustrated.

The choice of kernel affects the ability of the OC-SVM to capture data. For example, the radial basis function (RBF) allows the method to capture nonlinear data, making it effective for high-dimensional and complex datasets [80]. Compared to other approaches, the OC-SVM provides a compact and mathematical decision boundary.

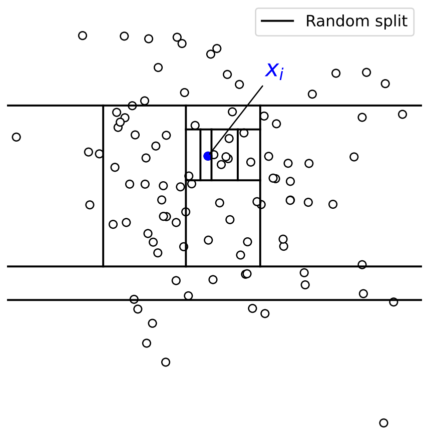
3.4.2 | Isolation Forest

The isolation forest (iForest) is a tree-based method that isolates anomalies instead of the conventional approach of learning the profile of normal points [40]. First described in 2008 [40], this technique was introduced as a novel method with linear time complexity and low memory requirements, as well as good accuracy and processing time, especially in large datasets. It also works well for high-dimensional problems and can be trained on unlabelled data of normal behaviour.

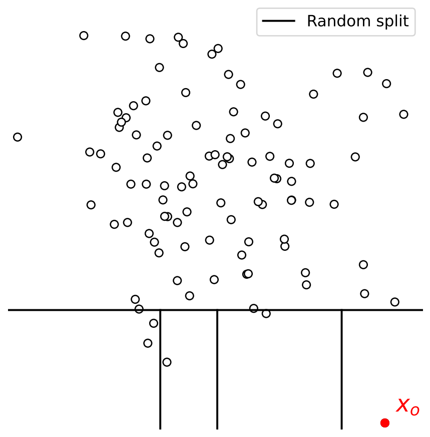
The iForest operates on the principle that anomalous points are inherently separated from normal points. Because there are typically fewer anomalies than normal instances, and anomalies have noticeably different values, they are more easily susceptible to being isolated [40]. This principle is visualized in Figure 3.5.

In Figure 3.5a, a normal point x_i is isolated from all other points by dividing the horizontal and vertical axes into sections. The same procedure is followed for an anomalous point x_o in Figure 3.5b. Separating the normal point requires more divisions in comparison to separating the anomalous points. This is the principle of isolation that the iForest relies on.

The iForest consists of an ensemble of isolation trees (iTrees), which use the isolation principle to isolate points. Within an iTree, a subset of all samples is considered. A random threshold in one of the feature dimensions is chosen, dividing the instances into two branches. This process of randomly splitting the samples continues until all instances are separated.



(a) Isolation of x_i (normal behaviour)



(b) Isolation of x_o (anomalous behaviour)

Figure 3.5: Isolation principle for normal and anomalous points. Based on [40]

Ultimately, the iForest contains a set of iTrees which are binary trees consisting of nodes. Each node specifies a threshold in one of the feature dimensions, splitting the samples into two child nodes. Since anomalies are more quickly isolated from other points, they are expected to exist closer to the root of the tree, where fewer splits have occurred. Normal points require many more partitions before being isolated, placing them deeper in the tree structure. An anomaly score can be calculated based on the depth of a point in the iTrees.

The performance of the iForest was found to converge quickly for only a limited number of iTrees. Additionally, the method performs better than some similar methods, such as the LOF, in both accuracy and execution time [40]. In contrast to most other ML-based methods, inference of an iForest does not require many computations, relying mostly on comparing a point to the iTree node thresholds.

3.4.3 | Local Outlier Factor

The local outlier factor (LOF) is a density-based anomaly detection method which identifies anomalies by comparing the local density of a point to that of its neighbours [38]. In contrast to global methods which rely on assumptions about the overall distribution, the LOF uses the local neighbourhood structure.

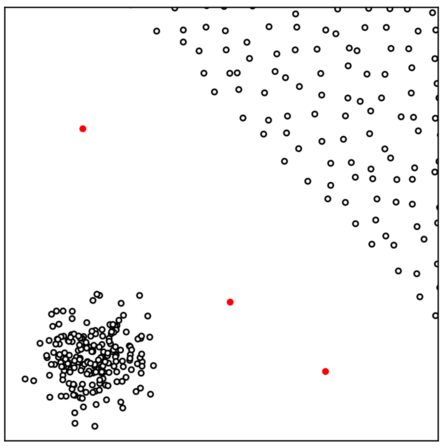


Figure 3.6: Visualization of varied local densities. Based on [38]

Developed in 2000 [38], the LOF computes the local reachability density (LRD) of each point. This metric captures how closely surrounded a point is by its k nearest neighbours. Anomalous points are typically in regions with significantly lower density compared to their local neighbourhood.

The LOF scores a point using the ratio of the average LRD of its neighbours and its own LRD. A value close to 1 then indicates that the point is in a region of comparable density to its neighbours. However, when values are substantially larger than 1, this suggests anomalous behaviour. An illustration of the varying local densities is shown in Figure 3.6. Here, anomalous points are located in sparse regions relative to its surroundings. The nominal clusters of points have different densities, meaning a global density value is not accurate. Due to the need to find local neighbours to a point, the method typically has a large computational impact. [81].

3.4.4 | LSTM Predictor

The long short-term memory (LSTM) predictor is a type of recurrent neural network (RNN) which can learn to retain specific information on its own. Before explaining the workings of the LSTM predictor, a simple neural network (NN) structure and the concept of LSTM cells are detailed.

NNs are well-suited for anomaly detection due to their ability to learn complex and nonlinear relationships. By training on a dataset with known patterns, a NN can develop an understanding of nominal behaviour, allowing it to identify deviations. The LSTM cell is an important extension due to its ability to retain sequential information and learn long-term dependencies.

Neural Network Structure

A fundamental type of NN is the multilayer perceptron (MLP). This network consists of connected layers of neurons which map some input, through one or more hidden layers, to an output. A possible configuration of an MLP is shown in Figure 3.7. Here, the network has an input layer with four elements, a hidden layer with five elements, and an output layer with a single element.

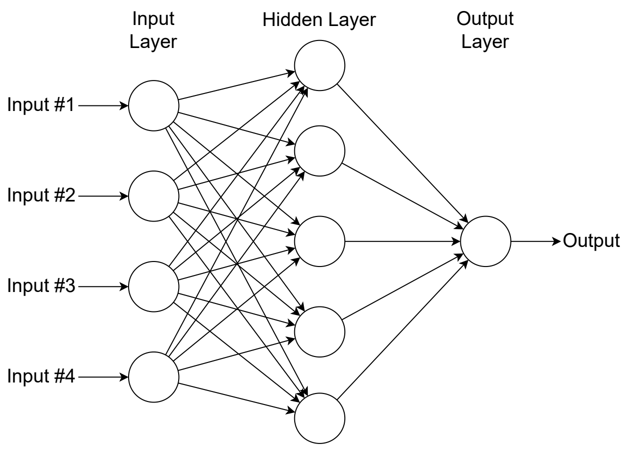


Figure 3.7: Structure of an MLP [82]

To calculate the value of a neuron, the values of the neurons in the previous layer are used. Each input is multiplied by a specific weight and summed, after which a specific bias value is added. Typically, an activation function is then used to calculate the value of the neuron. Without an activation function, the relationship between the inputs and output is linear. The function can be used to introduce nonlinearity. The process of finding the neuron value can be expressed as an equation:

$$y = f\left(\sum (x_i w_i) + b\right) \tag{3.1}$$

where y is the neuron output, x_i is the i-th input (or the output of the i-th neuron in the previous layer) and w_i its corresponding weight, b is the neuron bias, and f is some activation function, such as the sigmoid function. In the structure presented in Figure 3.7, this equation would be applied to calculate the value of each neuron in the hidden layer based on the values of the input layer, followed by the calculation of the output value based on the hidden layer. To calculate all neuron values at once, Equation 3.1 can be transformed into a matrix equation, where a weight matrix and bias vector are used instead of scalar values. The weight matrix specifies a weight for each combination of neuron in the previous and current layer.

In order for the network to fulfil a purpose, it is necessary to set the weights and biases such that the desired output is obtained. The process in which the network learns these parameters is called training. Typically, a set of data with known combinations of input and output is used. An optimizer adjusts the weights and biases gradually to minimize a loss function. This loss function effectively scores the performance of the network. Its gradient with respect to the parameters is typically calculated, indicating how the parameters should be adjusted in order to minimize loss.

Long Short-Term Memory

An extension of the core concept of the MLP is found in a recurrent neural network (RNN). These networks use loops to carry information between steps, acting as a form of memory. This allows the network to use past information for current predictions, and allows the network to learn sequential patterns, such as in time-series data. An MLP is not inherently able to achieve this.

A specific variant of an RNN is the long short-term memory (LSTM) network. This type was proposed in a 1997 paper to solve the problem of specific information blowing up or vanishing in other RNNs [83]. The network consists of one or more LSTM cells, forming a chain of repeating modules [84]. Each cell retains a state that is updated with the use of three gates. The structure of an LSTM cell is shown in Figure 3.8.

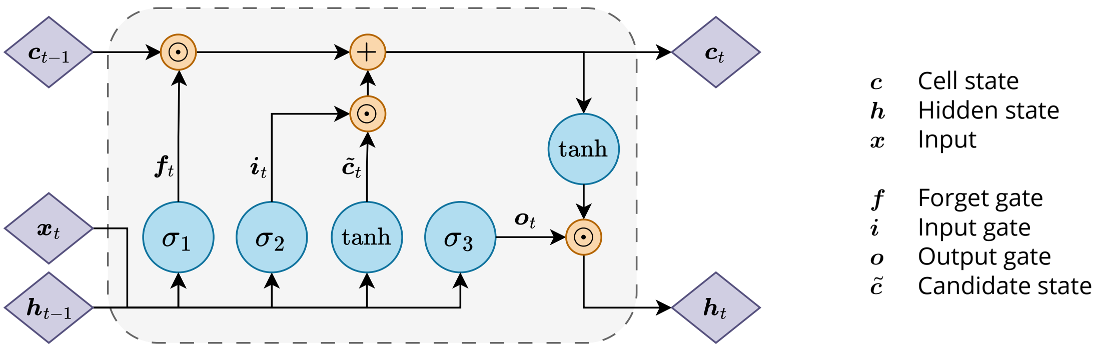


Figure 3.8: Structure of an LSTM cell. Based on [84].

In Figure 3.8, the cell state $\boldsymbol{c}$ flows through the top of the cell, and is updated from previous $t-1$ to current t. It maintains sequential information and allows the cell to retain knowledge from previous steps [84]. There are three so-called gates which control the cell state $\boldsymbol{c}_t$ and output of the cell $\boldsymbol{h}_t$. The forget-gate ($\boldsymbol{f}_t$) specifies which information should be erased from the old cell state, while the input-gate ($\boldsymbol{i}_t$) specifies what new information should be stored. The output-gate ($\boldsymbol{o}_t$) determines what contents should be exposed to the output. Each gate is a sigmoid (σ) NN layer. The sigmoid function causes the gate values to fall in the interval $[0, 1]$, where the value determines what portion of information is let through; a zero-value means nothing is passed through, a one-value means everything is passed through.

Stepping through the process, the first action is deciding which information should be forgotten through the forget-gate $\boldsymbol{f}_t$:

$$\boldsymbol{f}_t = \sigma_1 \left(W_f \, \boldsymbol{x}_t + U_f \, \boldsymbol{h}_{t-1} + \boldsymbol{b}_f \right) \tag{3.2}$$

where $\boldsymbol{h}_{t-1}$ is the previous output, $\boldsymbol{x}_t$ is the new input, W_f and U_f are the weight matrices of the forget-gate that act on the input and previous output respectively, and $\boldsymbol{b}_f$ is the bias of the forget-gate.

The second step is to process the new information. This consists of two actions: determining which information is kept through the input-gate $\boldsymbol{i}_t$, along with the candidate values $\tilde{\boldsymbol{c}}_t$:

$$\boldsymbol{i}_t = \sigma_2 \left(W_i \, \boldsymbol{x}_t + U_i \, \boldsymbol{h}_{t-1} + \boldsymbol{b}_i \right) \tag{3.3}$$

$$\tilde{\boldsymbol{c}}_t = \tanh \left(W_c \, \boldsymbol{x}_t + U_c \, \boldsymbol{h}_{t-1} + \boldsymbol{b}_c \right) \tag{3.4}$$

where W_i, U_i, and $\boldsymbol{b}_i$ are the weight matrices and bias of the input-gate, and W_c, U_c, and $\boldsymbol{b}_c$ are the weight matrices and bias of the cell state. The tanh function is used during the calculation of the candidate values for several reasons. First, it ensures the output is in the interval $[-1, 1]$ and has zero-mean. Second, it introduces nonlinearity, as was discussed before. Finally, during training the derivative of the function is needed, which is easy to compute for the tanh function.

The new cell state $\boldsymbol{c}_t$ can now be found by updating the previous state $\boldsymbol{c}_{t-1}$ with the forget-gate, the input-gate, and the candidate values:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{3.5}$$

where $\odot$ indicates pointwise multiplication.

The final step is to calculate the output $\boldsymbol{h}_t$ of the LSTM cell. The updated cell state $\boldsymbol{c}_t$ is run through the tanh function and multiplied by the output-gate, which determines what portion of the cell state is exposed:

$$\boldsymbol{o}_t = \sigma_3\left(W_o\, \boldsymbol{x}_t + U_o\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{o}}\right) \tag{3.6}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh\left(\boldsymbol{c}_t\right) \tag{3.7}$$

where W_o, U_o, and $\boldsymbol{b}_o$ are the weight matrices and bias of the output-gate. Again, the tanh function is used to ensure the output is in the interval $[-1, 1]$ with zero-mean, introduce nonlinearity, and provide a simple derivative for the training process.

Summarizing the internal process of the LSTM cell, the previous cell state is first updated using the previous output and new input. Part of the old information is forgotten, and part of the new information is stored. This updated state is then used to determine the new output of the cell.

LSTM Predictor

The LSTM network can be used for anomaly detection by training it to model normal data samples. The difference between the predicted normal sample and the observed sample is then a measure that can be used to identify anomalies [84].

A sequence of N consecutive samples is collected from the training dataset, $S = (\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_N)$ where each sample is a vector of dimension I. The LSTM predictor should predict the next sample $\hat{\boldsymbol{s}}_{N+1}$ based on the information in the window.

Multiple LSTM cells can be stacked allowing the model to learn patterns on different time scales, both long- and short-term [85]. In this context, stacking means that the output of one layer is connected to the input of the next layer. In Figure 3.8, this corresponds to connecting the output $\boldsymbol{h}_t$ of one cell to the input $\boldsymbol{h}_{t-1}$ of the next cell. To indicate which cell and which timestamp a variable corresponds to, the superscript (i) is added to indicate the i-th LSTM cell, besides subscript t to indicate time t. For example, the hidden state of the second cell at the tenth time step would be noted as $\boldsymbol{h}_{10}^{(2)}$.

Consider a model with three layers, $L = (1, 2, 3)$, where the LSTM cells have a hidden layer of dimension H. Each cell maintains its own hidden and internal state $\boldsymbol{h}_t^{(i)}$ and $\boldsymbol{c}_t^{(i)}$, which are initialized randomly at the beginning of the sequence as $\boldsymbol{h}_0^{(i)}$ and $\boldsymbol{c}_0^{(i)}$.

At each time step t, layer 1 receives the original input $\boldsymbol{s}_t$ from the training sequence. The subsequent layers receive the hidden state output of the layer before it:

$$\boldsymbol{x}_t^{(i)} = \begin{cases} \boldsymbol{s}_t & i = 1 \\ \boldsymbol{h}_t^{(i-1)} & i = 2, 3 \end{cases} \tag{3.8}$$

Starting at the first cell, each of the samples in the window S are processed one-by-one. The internal and hidden state of the first cell are updated at each step, leading to a sequence of N hidden state outputs $\left(\boldsymbol{h}_1^{(1)}, \boldsymbol{h}_2^{(1)}, \ldots, \boldsymbol{h}_N^{(1)}\right)$. This is the sequence provided to the second cell, which updates its own internal and hidden states step-by-step, producing its own sequence of outputs $\left(\boldsymbol{h}_1^{(2)}, \boldsymbol{h}_2^{(2)}, \ldots, \boldsymbol{h}_N^{(2)}\right)$. Finally, this is passed to the third cell, which produces the final sequence of outputs $\left(\boldsymbol{h}_1^{(3)}, \boldsymbol{h}_2^{(3)}, \ldots, \boldsymbol{h}_N^{(3)}\right)$. This stacked approach creates a deep architecture, where the first layer extracts short-term temporal features from the samples directly, the second layer transforms these into higher-level representations, and so on [85].

To generate the next sample $\hat{\boldsymbol{s}}_{N+1}$, the hidden state output of the final layer at the final time step is selected, $\boldsymbol{h}_N^{(3)}$. This serves as the learned summary of the entire input window S, incorporating both short- and long-term dependencies through the repeated gating across the LSTM layers. A dense layer, typically a linear layer, is used to transform this hidden state to a prediction of the sample:

$$\hat{\boldsymbol{s}}_{N+1} = W_{\text{dense}}\, \boldsymbol{h}_N^{(3)} + \boldsymbol{b}_{\text{dense}} \tag{3.9}$$

where W_{dense} and $\boldsymbol{b}_{\text{dense}}$ are the weight matrix and bias vector of the dense layer respectively. This transformation projects the abstract representation of the hidden state back to the original space of the data samples. The result is the model prediction of the next sample, based on the entire input window S.

During inference, the predicted sample $\hat{\boldsymbol{s}}_{N+1}$ is compared to the observed sample $\boldsymbol{s}_{N+1}$. The magnitude of the residual $\|\boldsymbol{s}_{N+1} - \hat{\boldsymbol{s}}_{N+1}\|$ serves as an anomaly score. When trained correctly, the model should predict normal samples with high accuracy, leading to small residuals. When an anomaly occurs the prediction deviates from the observed value, leading to a higher residual.

3.4.5 | LSTM (Variational) Autoencoder

The LSTM autoencoder (AE) is a reconstruction-based method which learns a compressed representation of normal patterns. It consists of two components: an encoder which reduces the input to the latent representation, and a decoder which attempts to reconstruct the original sequence from this representation. When trained on only nominal data, the reconstruction error provides a metric for the detection of anomalies [86].

During encoding, the input sequence is processed by one or more LSTM cells. The hidden state gradually stores information about the temporal structure of the data. After the final step, the hidden state forms a latent vector that summarizes the sequence. Then, the decoder, also a LSTM network, receives this latent vector and attempts to reconstruct the original sequence. This structure is visualized in Figure 3.9.

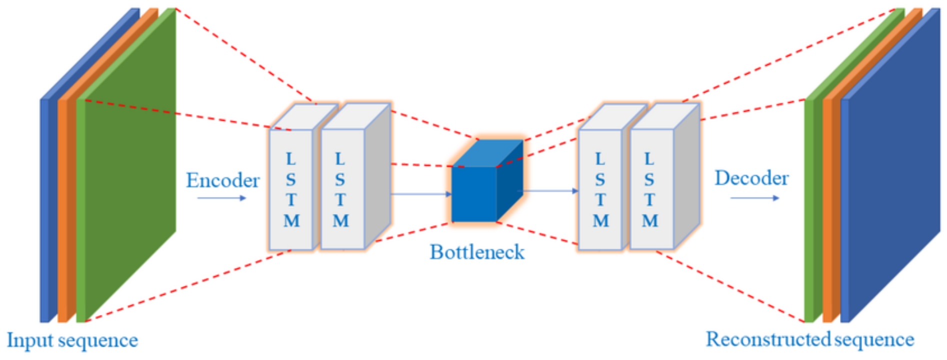


Figure 3.9: Sequence reconstruction with the LSTM-AE [87]

During training, the parameters are optimized to minimize the reconstruction error. Because the model is trained only on normal sequences, the network learns to reconstruct them. Deviations from the normal pattern lead to a higher reconstruction error, which serves as an anomaly score.

A variational autoencoder (VAE) extends this concept by introducing a probabilistic latent space. Rather than mapping the input sequence to a deterministic vector, the encoder outputs a distribution instead. The decoder then reconstructs the sequence by sampling from this distribution. Now, during inference, the likelihood of the latent representation is an additional indicator for anomalies. An LSTM VAE is thus useful when sequences contain complex or noisy behaviour, since the distribution captures uncertainty in the dynamics.

4 Kalman Filtering

To fuse the information from multiple sensors of different types, a sensor fusion system is required. Applying a Kalman filter is the most common technique to combine all measurements into a single estimate in the ADS of a spacecraft [17]. Many adaptations and variations have been developed over time to suit specific needs or constraints [18].

The traditional Kalman filtering technique consists of a single filter to process all information, which is a centralized approach. This structure is shown in Figure 4.1a, and is detailed in section 4.1. A decentralized, or distributed, approach has also been developed, where the processing is divided into multiple stages. The layout of such a system is shown in Figure 4.1b, and is described in section 4.2. For both of the approaches, numerous variations exist that are adapted to suit specific needs. One specific distributed variant, the FKF, is relevant for this thesis and is discussed in subsection 4.2.1.

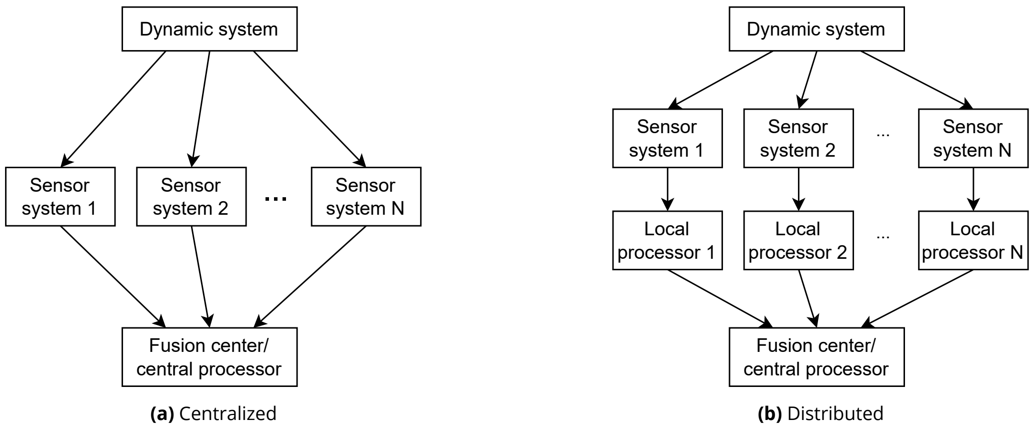


Figure 4.1: Comparison of two approaches to Kalman filtering [21]

4.1 | Centralized Kalman Filter

Centralized Kalman filters are those that use a single filter to combine all the available information into a solution [20], as is shown in Figure 4.1a. All sensor systems feed into one fusion center. This approach is most commonly used for state estimation in the ADS of spacecraft [17].

The original LKF developed in 1960 by R.E. Kalman is a centralized filter [19]. The two most common variations of the LKF in this application context are the EKF and UKF, which are designed to handle the nonlinearity associated with the spacecraft attitude. They are discussed in the following sections.

4.1.1 | Extended Kalman Filter

The original Kalman filter as described above is an estimator for linear systems. Since many physical systems, such as the motion of a spacecraft, are nonlinear, the filter was extended to work for these systems as well. The EKF uses Taylor expansions to linearize the nonlinear model about a working point [88]. Each cycle, the Jacobian is computed for the state and measurement models, which linearizes around the current estimate. A large error in the initial state can lead to the EKF giving poor accuracy or even diverging due to large linearization errors [89].

Quaternions are widely used for attitude estimation due to the absence of singularities which burden many other parameterizations [90]. They use the least amount of parameters to achieve this. A direct implementation of a quaternion-based EKF causes a violation of the unity norm constraint ($\boldsymbol{q}^T\boldsymbol{q} = 1$). This constraint specifies that the magnitude of the quaternion should remain 1 in order to represent an attitude. There are several variations of the EKF which use different techniques to circumvent this issue. Two common options are the additive extended Kalman filter (AEKF) and the multiplicative extended Kalman filter (MEKF) [91]. An overview of the MEKF is shown in Algorithm 1. It and the AEKF are described in more detail in Appendix B.

Both variants follow the same general approach as the original LKF. Before use, the current state and covariance are initialized. Additionally, the process noise and expected measurement noise are specified. Then, for each step of the algorithm, the state is propagated and then updated. During propagation, a physical model is applied to predict what the next state should be. This is followed by the update, where a new measurement is used to modify this prediction.

In the AEKF, the four elements of the quaternion are estimated as independent parameters. This approach can cause numerical issues, though there are techniques to ensure stability is maintained [92]. The MEKF, in contrast, uses a three-component error vector from a reference quaternion internally, which guarantees the output is a unit quaternion [91]. The MEKF typically requires less computation and is more stable than the AEKF, which is why it is generally considered superior [91].

4.1.2 | Unscented Kalman Filter

The UKF was developed to overcome some flaws of the EKF. Notably, the EKF linearizes the system dynamics around the current state estimate during propagation. This can be problematic if the system is highly nonlinear, potentially leading to suboptimal performance or divergence [89]. The UKF addresses this by using an unscented transform, which attempts to rebuild the probability distribution of a variable that passed through a nonlinear function. A set of specific sample points, sigma points, are selected around the current state estimate such that they represent the current mean and covariance. These sigma points are all propagated, and the solutions can be used to approximate the new mean and covariance.

The UKF is generally more accurate than the EKF since it better captures the uncertainty in the system state [89]. Additionally, since the UKF does not use a linear approximation, there is no need to calculate derivatives as is done in the EKF. Finally, the UKF is generally more stable than the EKF, especially for systems that are highly nonlinear.

Similar to the EKF, quaternions are a desired attitude representation. A direct implementation requires deriving the predicted mean using an averaged sum of quaternions, meaning there is no guarantee the resulting quaternion adheres to the unity norm constraint. To overcome this, the unscented quaternion estimator (USQUE) algorithm was developed [90]. This method uses a vector of generalized Rodrigues parameters (GRP) to represent an attitude error internally, similar to the MEKF. This three-component vector reintroduces singularities, avoiding which was the original reason for using a quaternion. However, since the vector only represents an error and typically remains small, the singularity is not reached in practice. An overview of USQUE is shown in Algorithm 2, and is described in more detail in Appendix A.

Algorithm 1 Overview of MEKF

```
Initialize:
    $\hat{\boldsymbol{q}}_0^+$, $\hat{\boldsymbol{\beta}}_0^+$, $\hat{P}_0^+$, $Q$, $R$
Propagate:
    $\hat{\boldsymbol{\omega}}_k^+ \leftarrow \tilde{\boldsymbol{\omega}}_k - \hat{\boldsymbol{\beta}}_k^+$
    $\hat{\boldsymbol{q}}_{k+1}^- \leftarrow \Omega(\hat{\boldsymbol{\omega}}_k^+)\hat{\boldsymbol{q}}_k^+$
    $P_{k+1}^- \leftarrow F P_k^+ F^T + Q$
Update:
    for i = 0 to $N$ do
        $\boldsymbol{\delta q}_i \leftarrow \tilde{\boldsymbol{q}}_i \otimes \left(\hat{\boldsymbol{q}}_{k+1}^-\right)^*$
        $\boldsymbol{v}_{k+1,i} \leftarrow \boldsymbol{\delta p}_i^-(\boldsymbol{\delta q}_i)$
    end for
    $S_{k+1} \leftarrow H P_{k+1}^- H^T + R$
    $K_{k+1} \leftarrow P_{k+1}^- H^T S_{k+1}^{-1}$
    $\boldsymbol{\Delta x}_{k+1} \leftarrow K_{k+1}\ \boldsymbol{v}_{\boldsymbol{k}+1}$
    $\hat{\boldsymbol{q}}_{k+1}^+ \leftarrow \boldsymbol{\delta}\hat{\boldsymbol{q}}(\boldsymbol{\delta}\hat{\boldsymbol{p}}) \otimes \hat{\boldsymbol{q}}_{k+1}^-$
    $P_{k+1}^+ \leftarrow P_{k+1}^- - P_{k+1}^- K_{k+1} H$
```

Algorithm 2 Overview of USQUE

```
Initialize:
    $\hat{\boldsymbol{q}}_0^+$, $\hat{\boldsymbol{\beta}}_0^+$, $\hat{P}_0^+$, $Q$, $R$
Propagate:
    $\boldsymbol{\chi}_k(0) \leftarrow \hat{\boldsymbol{x}}_k^+$, $\hat{\boldsymbol{q}}_k^+(0) \leftarrow \hat{\boldsymbol{q}}_k^+$
    for i=1 to $2n$ do
        $\boldsymbol{\chi}_k(i) \leftarrow \hat{\boldsymbol{x}}_k^+ + \sigma_k(i)$
        $\hat{\boldsymbol{q}}_k^+(i) \leftarrow \boldsymbol{\delta q}_k^+(i) \otimes \hat{\boldsymbol{q}}_k^+$
    end for
    for i=0 to $2n$ do
        $\hat{\boldsymbol{q}}_{k+1}^-(i) \leftarrow \Omega\left(\hat{\boldsymbol{\omega}}_k^+(i)\right)\hat{\boldsymbol{q}}_k^+(i)$
    end for
    $\hat{\boldsymbol{x}}_{k+1}^- \leftarrow \frac{1}{n+\lambda}(\cdots)$
    $P_{k+1}^- \leftarrow \frac{1}{n+\lambda}(\cdots) + \bar{Q}_k$
    Update:
    $\boldsymbol{v}_{k+1} \leftarrow \tilde{\boldsymbol{y}}_{k+1} - \hat{\boldsymbol{y}}_{k+1}^-$
    $P_{k+1}^{vv} \leftarrow P_{k+1}^{yy} + R_{k+1}$
    $K_{k+1} \leftarrow P_{k+1}^{xy}\left(P_{k+1}^{vv}\right)^{-1}$
    $\hat{\boldsymbol{x}}_{k+1}^+ \leftarrow \hat{\boldsymbol{x}}_{k+1}^- + K_{k+1}\boldsymbol{v}_{k+1}$
    $P_{k+1}^+ \leftarrow P_{k+1}^- - K_{k+1} P_{k+1}^{vv} K_{k+1}^T$
    $\hat{\boldsymbol{q}}_{k+1}^+ \leftarrow \boldsymbol{\delta q}_{k+1}^+ \otimes \hat{\boldsymbol{q}}_{k+1}^-(0)$
```

4.2 | Distributed Kalman Filter

With improving processing power and an increased interest in fault-tolerant systems, new Kalman filter structures were developed which distribute the filtering process over multiple stages, as is shown in Figure 4.1b. Now, each sensor system is related to a local processor, which only receives information from that system. Each local filter independently produces a local estimate. Finally, in the fusion center, these local estimates are once again fused to obtain the final solution.

There are several benefits to distributing the sensor fusion process over the local processors. First, each local processor can now run at the frequency that matches the associated sensor system. In the centralized approach, if all measurements are to be used, the single filter must run at the highest frequency of the available sensor systems. If there are large discrepancies between the systems, this could lead to many unnecessary computations in which some measurements have not yet changed. By dividing the processing over multiple units, each unit can now independently match the sensor system frequency. This ensures all information is processed without unnecessary computation. A drawback is that this introduces the need for synchronization for the final fusion step.

A second benefit of the distributed approach is that it can reduce the computational time. Assuming it is supported by the hardware, the local filters can run now in parallel. The computations of the centralized approach are then divided over multiple processors. A case study into the health of a gas turbine engine found a decrease in runtime of 24.1% when using a distributed approach instead of a centralized alternative [23].

A final benefit which is especially important for this research project is the capacity for fault tolerance of a distributed filter [20]. A fault in one sensor system is isolated to a specific local processor, and does not necessarily affect any others. This means that the remaining local filters can act as backups, increasing the robustness of the filter [18].

A specific variant using the distributed approach is the FKF. This variant is discussed in more detail in the following section.

4.2.1 | Federated Kalman Filter

The difference between the FKF and other distributed methods is the information sharing principle. It was developed to improve the accuracy of the distributed filter, without compromising the fault detection capabilities [20]. Previous distributed filters, such as a cascaded filter, were subject to poor accuracy and even divergence [18]. This was due to local filter outputs not being sequentially random, and the fact that the local filter output and master filter output were not independent. The development of the FKF sought to improve this behaviour through a new principle: the information sharing principle. A formal definition is given by [24]:

The total system information can remain constant or decrease, but never increase, due to sharing.

In other words, double-counting of the same information should be avoided. This principle allows the master filter to treat the local filter estimates as statistically independent such that the globally optimal solution can be found [25]. Proper division of information is achieved by specifying the fraction of the total information, β, each subfilter receives. The master filter receives β_m, while the i-th local filter receives β_i. The principle then requires that the sum of all of these fractions is unity [25]:

$$\beta_m + \sum_{i=1}^{n} \beta_i = 1 \tag{4.1}$$

Different strategies can be used to comply with this equation, four of which are primarily considered [18]: the full-, partial-, zero-, and no-reset modes. They differ in the method with which the total information is divided over the local and master filters. A comparison of the structure of a federated filter using the different modes is provided in Figure 4.2. Note that the information sharing division is indicated in this figure using $\gamma_i = 1/\beta_i$.

Zero-reset Mode

The zero-reset mode (Figure 4.2a) retains all the long-term information in the master filter ($\beta_m = 1$) and none of the fused information in the local filters ($\beta_i = 0$). Each master filter cycle, the local filters are reset, such that they act as data compression filters with short-term memory only [18]. In this mode, it is possible for the local filters to use a reduced-order model due to the short estimation periods between the resets. Throughput is improved since the local filters do not need to wait for the master estimate.

Partial-reset Mode

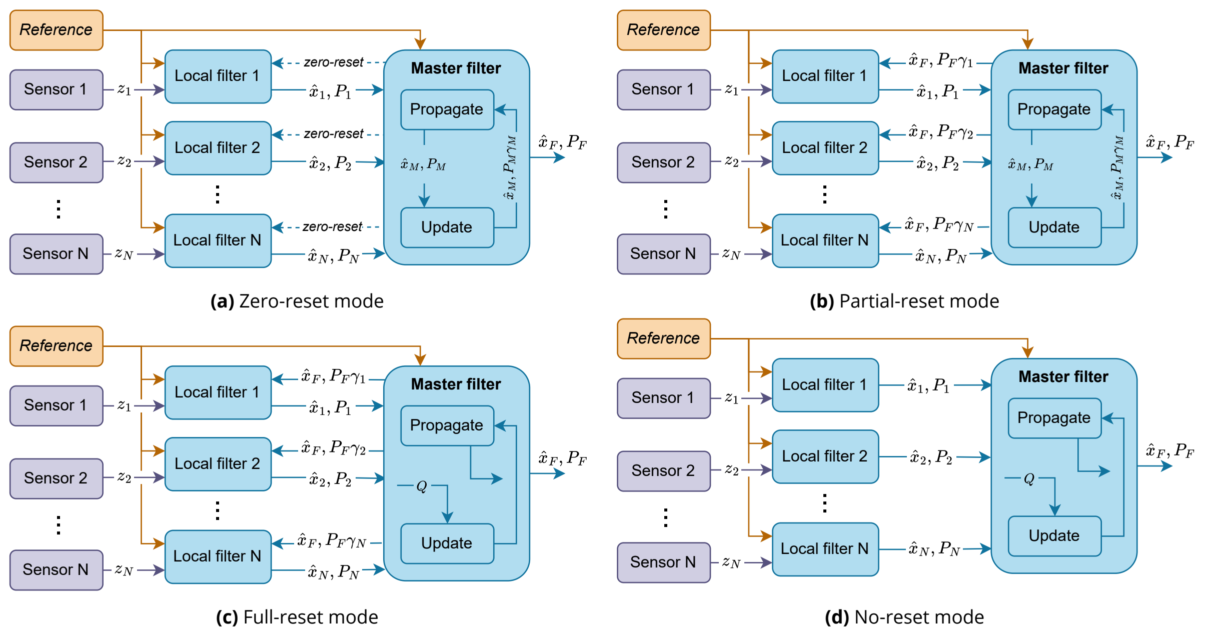
The partial-reset mode (Figure 4.2b) sees the long-term information shared by the master and local filters ($\beta_m \approx \beta_i \approx 1/(n+1)$). Feedback of the master fusion estimate is provided to the local filters, allowing them to reach higher accuracy than if they had operated independently [18]. Similar to the zero-reset mode, the local filters can use a reduced-order model to reduce computations.

Full-reset Mode

The full-reset mode (Figure 4.2c) has only the local filters share the master fusion solution ($\beta_i \approx 1/n$ and $\beta_m = 0$). Compared to the partial-reset mode, higher accuracy can be achieved in the local filters. This mode reduces the multipliers from $n + 1$ to n since there is no need for the master filter to retain information [18].

No-reset Mode

Finally, the no-reset mode (Figure 4.2d) is similar to the full-reset mode, except the information is directly stored by the local filters. Rather than split the fused solution, as is done in the partial- and full-reset mode, the local filters retain their own solutions without change. This mode is the least accurate but provides the best fault tolerance capacity [18].



(a) Zero-reset mode

(b) Partial-reset mode

(c) Full-reset mode

(d) No-reset mode

Figure 4.2: Structures of federated Kalman filter in zero-, partial-, full-, and no-reset mode. Based on [18].

The information sharing principle, in each of the different modes, allows for the solutions from all subfilters to be combined optimally through an additive algorithm [18]:

$$P_F = \left[{P_M}^{-1} + \sum_{i=1}^{N} {P_i}^{-1} \right]^{-1} = \left[{P_M}^{-1} + {P_1}^{-1} + {P_2}^{-1} + \cdots + {P_N}^{-1} \right]^{-1} \tag{4.2a}$$

$$\begin{aligned} \hat{\boldsymbol{x}}_F &= P_F \left[{P_M}^{-1} \hat{\boldsymbol{x}}_M + \sum_{i=1}^{N} {P_i}^{-1} \hat{\boldsymbol{x}}_i \right] \\ &= P_F \left[{P_M}^{-1} \hat{\boldsymbol{x}}_M + {P_1}^{-1} \hat{\boldsymbol{x}}_1 + {P_2}^{-1} \hat{\boldsymbol{x}}_2 + \cdots + {P_N}^{-1} \hat{\boldsymbol{x}}_N \right] \end{aligned} \tag{4.2b}$$

where subscript F indicates the final output, subscript M indicates the master filter output (if there is one), and subscripts $1 \dots N$ indicates the i-th subfilter output. The sharing principle avoids the need to maintain cross-covariances, and guarantees the above equation produces a correct final covariance matrix P_F. The sharing factor β from Equation 4.1 can then be incorporated to divide the final solution over the local and master filters, depending on the FKF mode:

$$P_j = P_F \, {\beta_j}^{-1} \qquad \text{for } j = 1, \dots, N, M \tag{4.3a}$$

$$\hat{\boldsymbol{x}}_j = \hat{\boldsymbol{x}}_F \qquad \text{for } j = 1, \dots, N, M \tag{4.3b}$$

5 Performance Metrics

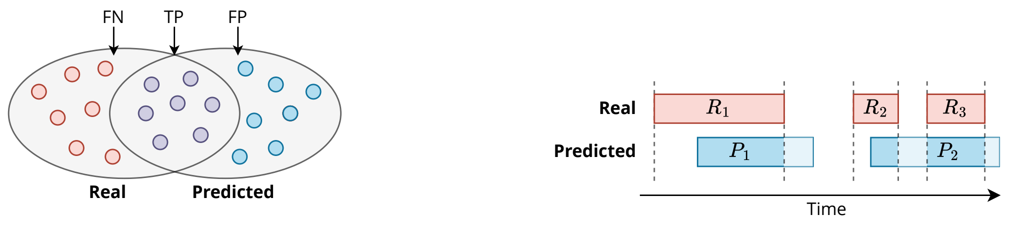
Several metrics were defined in order to quantify the performance of a detection method. First is the detection time, described in section 5.1. This is followed by a variation on the recall and precision, redefined for range-based anomalies. This is detailed in section 5.2. A visualization of sensitivity against specificity is described in section 5.3. The metrics are finally summarized in section 5.4.

5.1 | Detection Time

The first metric is the time to detection. This measures the time span between the actual start of the fault and the time at which the method first classifies an instance as an anomaly. For abrupt faults with a sudden deviation this is expected to be rapid due to the obvious nature of the fault. However, for gradual faults this is expected to be more subtle, leading to a greater detection time. A minimal detection time is desired in order to quickly exclude faulty sensors.

5.2 | Range-based Recall and Precision

The standard metrics for classifying performance in anomaly detection literature are precision and recall [35, 93]. Precision measures the accuracy of the detected anomalies, a high precision indicating a low false alarm rate. Recall, also called sensitivity, measures the ability of a method to find all faults, tied to the number of false negative instances.



(a) Traditional point-based

(b) Revisited range-based

Figure 5.1: Comparison of point-based and range-based anomalies. Based on [93]

These metrics suffer from the inability to represent domain-specific time series anomalies [93]. Specifically, the accuracy of many anomaly detection methods is misrepresented since the point-based metrics are applied to range-based anomalies. Range-based anomalies are those that occur over a consecutive sequence of points between a given start and end time, where no nominal data exists between the start and end of the range. This directly matches the need for this research project, where the simulated faults are consecutive stretches of time with constant faults. The difference between point-based and range-based anomalies is visualized in Figure 5.1. In cases where the detected and true ranges partially overlap, the position of this overlap could be of importance, for example to place emphasis on early detection. Additionally, the detection of an anomalous range in a single unit is sometimes preferred, instead of multiple separate detections that overlap. A 2018 paper [93] redefines the precision and recall metrics for range-based anomaly detection to be more effective for range-based anomalies.

Range-based Recall

The proposed redefinitions extend the classical metrics of precision and recall such that they can still be formulated with the new metrics [93]. Assumed are a set of real anomaly ranges $R = \{R_1, \dots, R_{N_r}\}$ and a set of predicted anomaly ranges $P = \{P_1, \dots, P_{N_p}\}$, where N_r and N_p are the number of real and predicted ranges respectively. The revisited recall Recall$_T$ is then defined as follows [93]:

$$\text{Recall}_T(R, P) = \frac{\sum_{i=1}^{N_r} \text{Recall}_T(R_i, P)}{N_r} \tag{5.1}$$

$$\text{Recall}_T(R_i, P) = \alpha \cdot \text{ExistenceReward}(R_i, P) + (1 - \alpha) \cdot \text{OverlapReward}(R_i, P) \tag{5.2}$$

Here, α is a factor between 0 and 1 that is tunable, representing the relative importance of rewarding existence and overlap. It is typically set to 0.5 to attribute equal importance [93]. Existence is defined as the detection method catching the existence of an anomaly in the first place, even by only a single point in the real range. Second, the overlap score rewards the extent to which the predicted ranges overlap with the real ranges, with a larger correctly predicted portion leading to a higher score. Here, the cardinality factor suggests detecting an anomaly with a single prediction is more valuable than detecting it in multiple fragments. The two scores are defined as:

$$\text{ExistenceReward}(R_i, P) = \begin{cases} 1 & \text{if } \sum_{j=1}^{N_p} |R_i \cap P_j| \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

$$\text{OverlapReward}(R_i, P) = \text{CardinalityFactor}(R_i, P) \cdot \sum_{j=1}^{N_p} \omega(R_i, R_i \cap P_j, \delta) \tag{5.4}$$

$$\text{CardinalityFactor}(R_i, P) = \begin{cases} 1 & \text{if } R_i \text{ overlaps with at most one } P_j \\ \gamma(R_i, P) & \text{otherwise} \end{cases} \tag{5.5}$$

Here, the function $\gamma()$ determines the overlap reward drop-off when multiple ranges overlap with a real range, $\omega()$ determines the reward for the degree to which the predicted and real ranges overlap, and $\delta()$ is a bias that influences $\omega()$ based on the relative locations of the predicted and real ranges, for example to attribute more importance to early detection. The selection of these functions is tunable according to the needs of the application. They are discussed in more detail in the section following the description of the revisited F_1-score. The Recall$_T$ score should be as close to 1.0 as possible.

Range-based Precision

The revisited precision Precision$_T$ is similarly defined. In this case, there is no need for an existence reward since precision by definition emphasizes prediction quality, where existence is not useful for judging this quality [93]:

$$\text{Precision}_T(R, P) = \frac{\sum_{i=1}^{N_p} \text{Precision}_T(R, P_i)}{N_p} \tag{5.6}$$

$$\text{Precision}_T(R, P_i) = \text{CardinalityFactor}(P_i, R) \cdot \sum_{j=1}^{N_r} \omega(P_i, P_i \cap R_j, \delta) \tag{5.7}$$

Again, the functions $\gamma()$, $\omega()$, and $\delta()$ are tunable according to the application. They can also be defined differently between precision and recall [93]. This is discussed in more detail in the section following the revisited F_1-score. The Precision$_T$ score should be as close to 1.0 as possible.

Range-based F_1-score

Similarly to the classical recall and precision, the F-score can be calculated which combines the two metrics [35, 93]. The score contains a factor β which is used to balance the importance between recall and precision. It is often set to $\beta = 1$ to indicate equal importance, essentially leading to their harmonic mean as the F_1-score$_T$:

$$F\text{-score}_T = \frac{(1+\beta^2)\cdot \text{Precision}_T \cdot \text{Recall}_T}{\beta^2 \cdot \text{Precision}_T + \text{Recall}_T} \tag{5.8}$$

$$F_1\text{-score}_T = \frac{2\cdot \text{Precision}_T \cdot \text{Recall}_T}{\text{Precision}_T + \text{Recall}_T} \tag{5.9}$$

The F_1-score$_T$ score should be as close to 1.0 as possible, similar to the Recall$_T$ and Precision$_T$ themselves.

Tunable Factors

As mentioned, there are several tunable functions and a weight that influence the scoring based on the application. These affect various properties of the revisited score and can be set according to the application in which they are used.

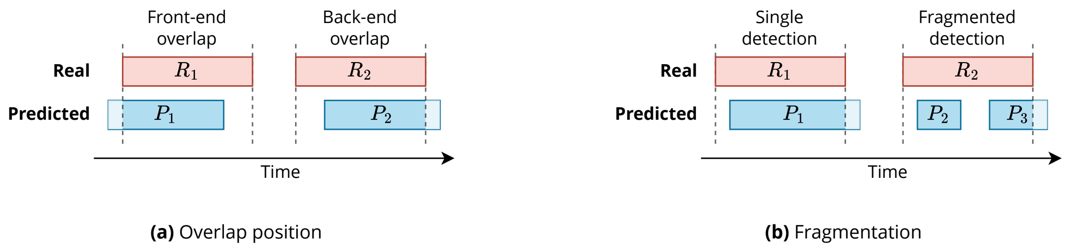


Figure 5.2: Overlap and fragmentation of real and predicted ranges

The function $\omega()$ measures the size, defined as the correctly predicted portion of a real range [93]. The larger a portion of the real range is detected, the higher the recall score becomes. This function is influenced by function $\delta()$, which accounts for the relative position of the correctly predicted portion of the real range. In some applications, it is important for certain sections of a range to be predicted more accurately than others, so the positional bias function $\delta()$ can be used to grant more weight to these sections. This is visualized in Figure 5.2a, where the overlaps of predicted ranges P_1 and P_2 and real ranges R_1 and R_2 have different relative positions. Several example definitions for the positional bias are provided [93]:

$$\delta(i, \text{AnomalyLength}) = \begin{cases} 1 & \text{flat bias} \\ \text{AnomalyLength} - i + 1 & \text{front-end bias} \\ i & \text{back-end bias} \end{cases} \tag{5.10}$$

For the application in this research project, early detection of faults is preferred. As such, a front-end bias is used to signify this importance. As suggested by the authors who developed the range-based metrics, this bias is applied to the recall, but not the precision [93]. The presence of a false positive is then considered uniformly bad, irrespective of its location.

Finally, the $\gamma()$ function is used to calculate the cardinality factor when multiple predicted ranges overlap a real range [93]. This factor should be inversely proportional to the number of ranges that overlap with the real range, causing more fragmented predictions to receive a lower score. This is shown in Figure 5.2b, where real range R_1 is detected with a single prediction, whereas R_2 is detected with multiple separate predictions. A typical example is given as:

$$\gamma(R_i, P) = \frac{1}{N_{\text{overlap}}} \tag{5.11}$$

where N_{overlap} is the number of predicted ranges from the set P that overlap with the real range R_i.

5.3 | Receiver Operating Characteristic

Another common metric used to evaluate and compare the performance of detection methods is the receiver operating characteristic (ROC). This curve depicts the relationship between sensitivity and specificity for varying threshold values. The area under the curve (AUC) is used as a single value describing the performance of the classifier.

The ROC depends on the category counts identified by the traditional confusion matrix. For a binary classifier, these are the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The y-axis corresponds to the true positive rate (TPR) or sensitivity, while the x-axis corresponds to the false positive rate (FPR) or $1 -$ specificity:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (5.12) \qquad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad (5.13)$$

For methods that provide real value scores, a threshold is selected. By varying the threshold, the counted categories are affected, in turn influencing the TPR and FPR. These changing rates are then depicted as the ROC curve. An example is shown in Figure 5.3, which depicts how the curves indicate various performance levels. In general, classifiers with better performance arch more into the upper-left quadrant of the graph, while worse classifiers arch into the lower-right quadrant. The grey diagonal line indicates the performance of a random classifier. The ROC-AUC is then the area underneath a curve in this diagram, which varies between 0.0 and 1.0. This score should be as close to 1.0 as possible to indicate better performance.

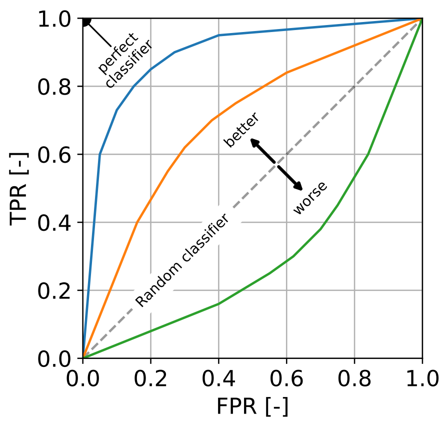


Figure 5.3: Example of ROC curves

Some criticisms of the ROC-AUC measure exist. First, the calculated score includes predictions that obtained insignificant sensitivity and specificity [94]. Additionally, the measure does not include information about precision or negative predictive value. The ROC-AUC should therefore not be used by itself, without inclusion of additional metrics.

5.4 | Overview

The selected metrics are summarized in Table 5.1. Combined, they quantify the relevant performance of a fault detection method within the application context of the FKF.

Table 5.1: Performance metrics used for method comparison

Metric	Description
Detection time	Time between actual start of fault and first detection
Recall$_T$	Range-based portion of real anomalies that are successfully detected
Precision$_T$	Range-based portion of detected anomalies that are real anomalies
F_1-score$_T$	Combination of range-based recall and precision
ROC-AUC	Sensitivity and specificity for changing thresholds

The detection time directly measures the response time to a fault, which should be minimized in order to exclude faults quickly. Next, the revisited recall and precision measure the accuracy of the detections with respect to the existence, size, position, and cardinality of the predictions. The F_1-score$_T$ metric combines these two scores into one, balancing their importance equally. Additionally, the ROC curves and the ROC-AUC provide additional insight. The performance of an anomaly detection method can be quantified with these metrics.

Part II

Simulation and Fault Detection

In order to evaluate the performance of ML-based fault detection methods and determine whether it improves upon conventional approaches, a simulation was created to emulate the response to various faults. In this part of the report, this simulation and the implemented fault detection methods are detailed. First, the structure of the simulation is described in chapter 6, including the models used for sensors and their faults, as well as their fusion in a federated structure. This is followed by chapter 7, in which a description of the conventional fault detection methods is given. They serve as the baseline for comparison to the ML-based methods that are selected and tuned in chapter 8.

6 Simulation

In this chapter, the simulation that was created to enable the training and testing of various fault detection methods is described. A robust suite of tools was implemented in order to provide flexible support for experimenting and simulation of various methods. First, the method with which reference data was generated is provided in section 6.1. This forms the basis of the simulation. This is followed by the models that are used to emulate attitude sensor behaviour, described in section 6.2. Associated with each sensor is a set of fault models, given in section 6.3. The scenarios which are used to evaluate the performance of the fault detection methods are described in section 6.4. Finally, the overall structure of the simulation is given in section 6.5.

6.1 | Data Generation

For the training of a machine learning model, a large quantity of data is required. The software Systems Tool Kit (STK) from Ansys[1] was used to simulate a variety of scenarios. The application can be used to model objects on land, in and under the sea, in the air, and in space [95]. The high-fidelity simulation of both position and orientation is used throughout various industries, including the aerospace industry. Customizable reports can be generated, making large quantities of data available for use in other applications.

6.1.1 | Satellite Definition

Before any data can be generated, a scenario with an object needs to be created. This object represents a physical entity, which for this research project is a satellite, specifically a CubeSat. This section covers the properties given to the satellite object.

STK simulates both the orbit and attitude of a satellite through time. For all simulations, the same reference satellite was used with properties of a typical 6U CubeSat [96]. A low Earth orbit (LEO) is selected, which is a typical operating environment for a CubeSat. An initial two-line element set (TLE) is propagated using the Simplified General Perturbations 4 (SGP4) propagator. The propagation accuracy is not crucial for investigating the attitude behaviour and is sufficient for the limited time period discussed later. A summary of the properties of the STK satellite object is listed in Table 6.1.

Table 6.1: Properties of STK satellite object

Property	Value	Unit
Orbit altitude	~ 500	[km]
Orbit eccentricity	~ 0	[-]
Orbit inclination	~ 100	[deg]
CubeSat size	$30 \times 20 \times 10$	[cm]

Continued on next page

[1] https://www.ansys.com/products/missions/ansys-stk

Table 6.1: *continued from previous page*

Property	**Value**	**Unit**
Mass	10	[kg]
Inertia matrix	diag$(416.67, 1083.33, 833.33)$	[kg cm^2]
Cross-sectional area	600	[cm^2]

On top of the properties listed in the table, a sensor object was attached to the body of the satellite, pointing outwards from one of the large faces of the CubeSat. This is shown in Figure 6.1, where the blue cone represents the view of the sensor. The sensor direction is used in one of the scenarios described in section 6.4. The specific cone angle, or field of view (FOV), is not relevant for this analysis. The thin yellow line indicates the path of the satellite, with red and yellow vectors indicating the velocity and Sun vector respectively.

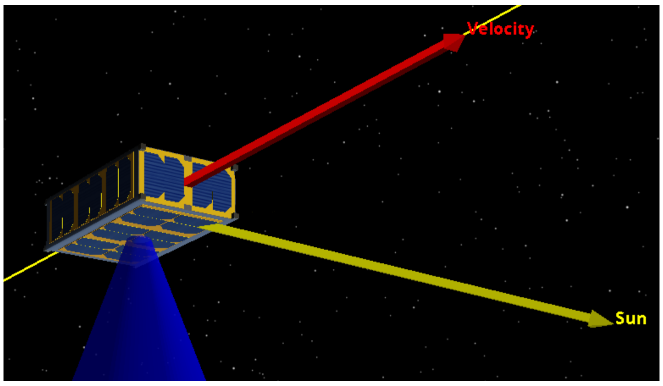


Figure 6.1: 3D view of STK satellite in default orientation

The orbit of the satellite is based on a TLE which is propagated using the SGP4 propagator. For the duration of the propagation (one day), this provides sufficient accuracy to investigate the attitude behaviour. Various mission phases in which this behaviour differs were simulated, which is described in more detail in section 6.4.

6.1.2 | **Data Output**

STK provides a customizable report generation tool, with which specific simulated variables can be exported. This data serves as the basis for the analysis performed in this thesis. In Table 6.2, the variables which are exported from STK are listed. Some variables are reported in several reference frames, such as the ECI or ECEF frames.

Table 6.2: Reported variables from STK simulation

Variable	**Frame**	**Unit**
Time	N/A	ISO8601 UTC
Quaternion	ECI, ECEF	[-]
Angular velocity	ECI, ECEF	[rad/s]
Euler angles	N/A	[rad]
Euler angle rates	N/A	[rad/s]
Position	ECEF	[m]
Velocity	ECEF	[m/s]
Acceleration	ECEF	[m/s^2]
Solar intensity	N/A	[%]
Latitude/longitude	Geodetic, geocentric	[deg]

Continued on next page

Table 6.2: *continued from previous page*

Variable	Frame	Unit
Orbital radius	N/A	[m]
Orbital altitude	N/A	[m]
Sun vector	ECEF, body	[m]
Moon vector	ECEF, body	[m]
Magnetic field vector	ECI, ECEF, body	[nT]
Total torque	ECI, body	[N m]

The collection of variables is sufficient to model sensor measurements and simulate sensor fusion for attitude determination of a spacecraft. This is detailed in the following sections.

6.2 | Sensor Models

In order to accurately model sensor faults, the working principles of various common attitude sensors were simulated. By simulating the underlying methodology, more realistic fault cases can be introduced. In this section, these sensor models are described. The subsequent fault models are then described in section 6.3.

The modelled sensors are the rate gyroscope (subsection 6.2.1) to measure the angular velocity of the craft, and three absolute attitude sensors: the magnetometer (subsection 6.2.2), the Sun sensor (subsection 6.2.3), and the star tracker (subsection 6.2.4). They are simulated to a level of detail that allows the injection of the faults described in section 6.3.

Typically, the magnetometer and Sun sensor, as well as other sensor types which rely on a single vector measurement, do not produce a quaternion measurement. This is due to ambiguity, where the attitude has a degree of freedom around the vector's axis. In the simulation, to simplify usage, the known attitude quaternion is used to fix the quaternion, selecting the quaternion which best aligns with the true attitude. The effect is that the sensor still relies on realistic principles, while also producing a quaternion attitude measurement.

6.2.1 | Gyroscope

A model of the measured angular rates $\tilde{\omega}$ by a multi-axis rate gyro is provided by [97]:

$$\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega} + \boldsymbol{\beta}_\omega + \boldsymbol{\eta}_\omega \quad (6.1) \qquad \dot{\boldsymbol{\beta}}_\omega = \boldsymbol{\eta}_\beta \quad (6.2)$$

where $\boldsymbol{\omega}$ is the true angular velocity, $\boldsymbol{\beta}_\omega$ is the measurement bias, and $\boldsymbol{\eta}_\omega$ and $\boldsymbol{\eta}_\beta$ are uncorrelated vectors following zero-mean Gaussian white noise with variances σ_ω^2 and σ_β^2 respectively.

In this model, the bias is not a constant offset but modelled using a random-walk process, causing drift over time as is the case for real sensors. For this reason, estimating and accounting for the gyroscope bias is typically included in sensor fusion for attitude determination.

The gyroscope measurement model is depicted in Figure 6.2. The angular velocity obtained from STK is taken as the true value, to which noise is added. The current sensor bias is added to obtain the angular velocity measurement. Noise is added to the bias value for the next measurement.

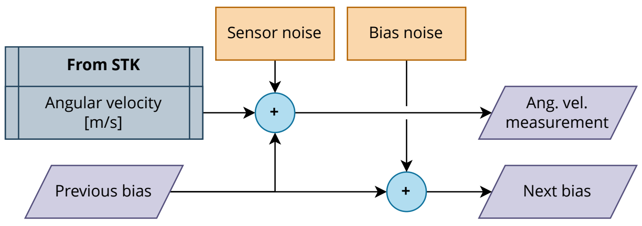


Figure 6.2: Gyroscope measurement model

6.2.2 | Magnetometer

Magnetometers function by observing the local magnetic field, and comparing it with a reference field model, typically the IGRF [55]. This model expresses the magnetic potential function V in terms of the position of the spacecraft:

$$V(r,\theta,\phi) = a\sum_{n=1}^{\infty}\left(\frac{a}{r}\right)^{n+1}\sum_{m=0}^{n}P_n^m \cos(\theta)\left(g_n^m \cos(m\phi) + h_n^m \sin(m\phi)\right) \tag{6.3}$$

where (r,θ,ϕ) are the geocentric spherical polar coordinates of the spacecraft, a is the equatorial radius of the Earth, P_n^m are associated Legendre functions, and g_n^m and h_n^m are Gaussian coefficients which are determined empirically. The reference magnetic field $\boldsymbol{B}_{\text{ref}}$ is then expressed in the spherical coordinates:

$$\boldsymbol{B}_{\text{ref}} = \begin{bmatrix} B_r \\ B_\theta \\ B_\phi \end{bmatrix} = -\nabla V(r,\theta,\phi) = -\begin{bmatrix} \frac{\partial V}{\partial r} \\ \frac{1}{r}\frac{\partial V}{\partial \theta} \\ \frac{1}{r \sin\theta}\frac{\partial V}{\partial \phi} \end{bmatrix} \tag{6.4}$$

This reference field is then typically transformed to the ECI or ECEF frame when used in the ADS. The measurement of the magnetometer is transformed from the sensor frame into the spacecraft body frame. By comparing the measured and reference fields, information about the spacecraft attitude is obtained.

To simulate the behaviour of this sensor, the process depicted in Figure 6.3 is followed. From STK, the body magnetic field is taken, to which angular noise is added. STK is set to use the IGRF model to calculate this vector. Next, the time and position of the spacecraft are used to calculate the reference field following Equation 6.3 and 6.4. Deriving an attitude from the two field vectors leads to a quaternion with a degree of freedom. The quaternion that best aligns with the quaternion provided by STK is selected. The model assumes there are sufficient sensor components to measure the three-axis magnetic field vector.

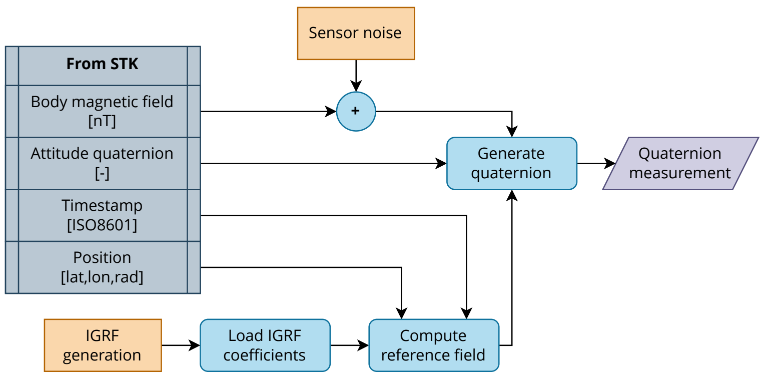


Figure 6.3: Magnetometer measurement model

6.2.3 | Sun Sensor

Sun sensors observe the direction of the incoming Sunlight to deduce the attitude of the spacecraft. The measured direction is compared to the expected position of the Sun, which is used to deduce attitude information.

To determine the expected Sun position, expressions using the J2000.0 epoch can be used. These are provided by the Astronomical Almanac, which is a joint publication by British and American institutions [98]. The procedure to determine the Sun position vector is described here.

First, the number of Julian centuries, T_{UT1}, is calculated from the epoch:

$$T_{\text{UT1}} = \frac{\text{JD}_{\text{UT1}} - 2\,451\,545}{36\,525} \tag{6.5}$$

$$\begin{aligned}\text{JD}_{\text{UT1}} &= 1\,721\,013.5 + 367\,\text{yr} - \text{int}\left(\frac{7}{4}\left(\text{yr} + \text{int}\left(\frac{\text{mo}+9}{12}\right)\right)\right) \\ &\quad + \text{int}\left(\frac{275\,\text{mo}}{9}\right) + \text{d} + \frac{60\,\text{hr} + \text{min} + \frac{\text{s}}{60^*}}{1\,440}\end{aligned} \tag{6.6}$$

Here, JD_{UT1} is the Julian date, calculated using the J2000.0 epoch (yr-mo-d hr:min:s). Note that in leap years, a value of 61 instead of 60^* should be used in the final term.

Next, the mean longitude of the Sun, $\phi_{M\odot}$, and the mean anomaly for the Sun, $M_\odot$, can be calculated:

$$\phi_{M\odot} = 280.460^\circ + 36\,000.771\,T_{\text{UT1}} \mod 360^\circ \tag{6.7}$$

$$M_\odot = 357.529\,109\,2^\circ + 35\,999.050\,34\,T_{\text{UT1}} \mod 360^\circ \tag{6.8}$$

Now, the ecliptic longitude, ϕ_{ecliptic}, can be approximated. Additionally, the ecliptic latitude, θ_{ecliptic}, never exceeds $0.000\,333^\circ$, and is often taken to be 0.0° [98]. The obliquity of the ecliptic, ε, is also approximated:

$$\phi_{\text{ecliptic}} = \phi_{M\odot} + 1.914\,666\,471\ \sin M_\odot + 0.019\,994\,643\ \sin 2M_\odot \tag{6.9}$$

$$\theta_{\text{ecliptic}} = 0^\circ \tag{6.10}$$

$$\varepsilon = 23.439\,291^\circ - 0.013\,004\,2\,T_{\text{UT1}} \tag{6.11}$$

Now, the position magnitude, $r_\odot$, is calculated from the mean anomaly, in astronomical units (AU):

$$r_\odot = 1.000\,140\,612 - 0.016\,708\,617\ \cos M_\odot - 0.000\,139\,589\ \cos 2M_\odot \tag{6.12}$$

Finally, the Sun position vector in AU, $\boldsymbol{r}_\odot$, is set up using the position magnitude, obliquity of the ecliptic, and ecliptic longitude:

$$\boldsymbol{r}_\odot = \begin{bmatrix} r_\odot\ \cos\phi_{\text{ecliptic}} \\ r_\odot\ \cos\varepsilon\ \sin\phi_{\text{ecliptic}} \\ r_\odot\ \sin\varepsilon\ \sin\phi_{\text{ecliptic}} \end{bmatrix} \tag{6.13}$$

This vector is in the ECI frame, and can be transformed to other frames where necessary. If the spacecraft position is known, it can be subtracted from the Sun position vector to get the vector between the spacecraft and Sun.

The method with which the Sun sensor is simulated is shown in Figure 6.4. From STK, the body-Sun vector is taken as a measurement by adding angular noise to it. The timestamp is used to predict the Sun position vector according to Equation 6.13, where the spacecraft position is subtracted to obtain the spacecraft-Sun vector. The measured body-Sun vector and predicted body-Sun vector are then compared to derive an attitude. Since deriving a quaternion from two vectors leads to a quaternion with a degree of freedom, the quaternion that best aligns with the quaternion provided by STK is selected.

The local solar intensity at the spacecraft position is taken into account. If this intensity is below 50%, meaning the spacecraft is within the penumbra or umbra, the Sun sensor does not provide a measurement.

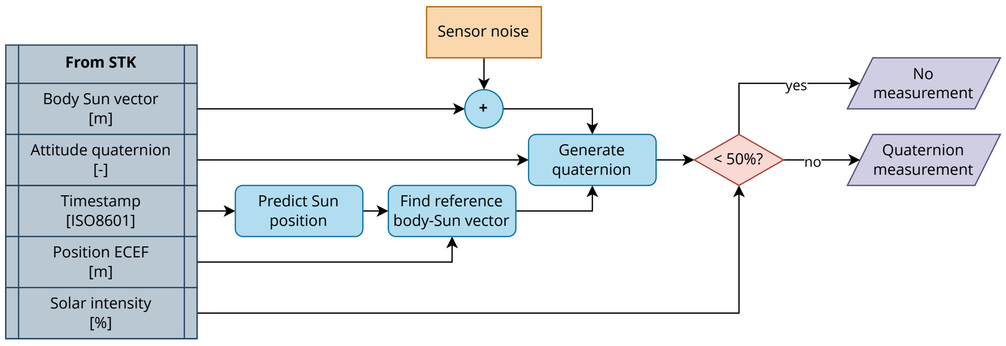


Figure 6.4: Sun sensor measurement model

The model assumes that, if not in eclipse, there are sufficient sensor modules on the spacecraft to measure the Sun vector in the body frame.

6.2.4 | Star Tracker

Star trackers observe the celestial sky and identify known star patterns to deduce the attitude of the sensor. They typically consist of a digital camera and processing unit. The camera module records an image of the visible sky, which is then used by a centroiding algorithm to locate stars in the image. The located stars are then identified by comparing to an internal star catalogue. Once found, the orientation of the sensor with respect to the celestial frame can be determined.

Modelling the full procedure of a star tracker is out of the scope of this research project. Instead, a simplified approach is used to still enable semi-realistic fault injection. This procedure is shown in Figure 6.5.

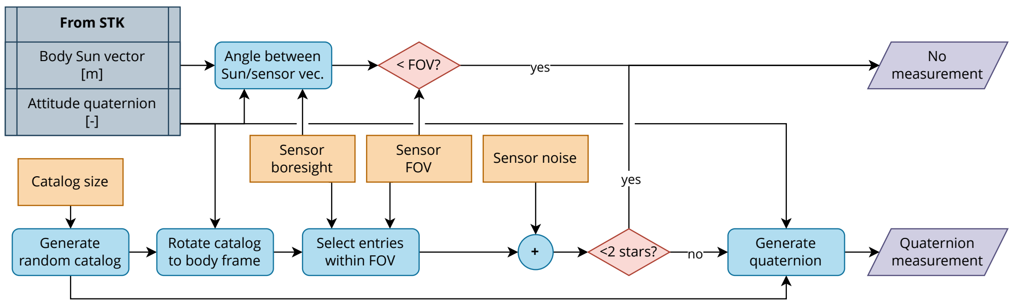


Figure 6.5: Star tracker measurement model

First, an internal catalogue is generated containing random vectors, mimicking the star vectors of a true star catalogue. Then, to generate a measurement, the internal catalogue is first rotated into the body frame using the reference quaternion from STK. Using the sensor's boresight and FOV, only the vectors that are visible to the sensor are selected from the catalogue, along with their positions from the non-rotated catalogue. Angular noise is added to the selected vectors to simulate centroiding variance, which now act as the measured vectors.

If enough stars are visible, Davenport's q-method is used to compute a quaternion based on the measured and expected catalogue vectors. The aim is to minimize the following loss function [99]:

$$L(A) = \frac{1}{2} \sum_{i=1}^{N} w_i |u_i - Av_i|^2 \tag{6.14}$$

where u_i is the i-th vector measurement and v_i is the corresponding i-th reference vector. w_i are weights for each observation. In Davenport's method, the objective function is then defined as:

$$g(A) = 1 - L(A) = \sum_{i=1}^{N} w_i U^T A V \tag{6.15}$$

where the goal is to find the optimal attitude matrix A based on the measurement and reference matrices U and V respectively. First, the attitude profile matrix is computed:

$$B = \sum_{i=1}^{N} w_i U V \tag{6.16}$$

This matrix is used to set up the matrix K, which is ultimately used to solve the optimization:

$$K = \begin{bmatrix} S - \sigma I_3 & \boldsymbol{z} \\ \boldsymbol{z}^T & \sigma \end{bmatrix} \tag{6.17}$$

$$\sigma = \text{trace}(B) \tag{6.18}$$

$$S = B + B^T \tag{6.19}$$

$$\boldsymbol{z} = \begin{bmatrix} B_{23} - B_{32} \\ B_{31} - B_{13} \\ B_{12} - B_{21} \end{bmatrix} \tag{6.20}$$

The optimal quaternion q is an eigenvector of the matrix K. The objective function is maximized if the eigenvector corresponding to the largest eigenvalue λ is selected:

$$Kq = \lambda q \tag{6.21}$$

Due to ambiguity, the sign of the quaternion could be opposite that of the true quaternion as obtained from STK. To simplify code usage, the sign of the quaternion is matched to that of the quaternion from STK. Finally, if the Sun vector is within the FOV of the sensor, no measurement is generated to simulate blinding of the camera module.

6.3 | Fault Models

In order to evaluate and compare fault detection methods, a realistic fault model is required. A sensor could exhibit one or more faults at a given time, and each sensor type might present unique fault cases. With the sensor models defined in section 6.2, a variety of faults were implemented to emulate the behaviour of a sensor under the presence of a fault.

The fault models are applied throughout the measurement process, and can affect both the ultimate sensor output and intermediate values. The various types of implemented faults and the sensors types for which they can apply are listed in Table 6.3.

The stuck fault type, and by extension the zero fault type, sees the output of a sensor hang on a constant value, either the last valid measurement or zero respectively. This is similar to the axis fault type, where one or more specific axes fail in this manner. During a complete fault, the sensor does not provide a value and the measurement is marked as invalid.

A noise fault introduces significant amounts of additional noise during the measurement. This is added on either the final output, such as for the gyroscope, or an intermediate value, such as for the other sensor types. The bias fault is similar and specific to the gyroscope, where the bias drift is significantly increased from normal behaviour.

The time- and position offset faults cause the variables used for estimating the reference magnetic field or reference Sun vector to deviate. This leads to an incorrect reference value, which affects the attitude measurement. Finally, the misalignment fault simulates a slight rotation of the expected sensor attitude within the body frame.

To evaluate the fault detection methods described in this research project, several fault cases were selected for analysis. These are described in more detail in section 6.4.

Table 6.3: Simulated sensor fault models

Fault type	Gyroscope	Magnetometer	Sun sensor	Star tracker	Description
Stuck fault	×	×	×	×	The output gets stuck at a constant value (by default, the value when the fault starts).
Zero fault	×	×	×	×	Similar to stuck fault. The output gets stuck at zero. For quaternions, output is an identity quaternion.
Complete fault	×	×	×	×	No output returned, invalid measurement.
Axis fault	×	×	×	×	One or multiple axes of the output or an intermediate variable stuck at a constant value.
Noise fault	×	×	×	×	Greatly increased measurement noise on the output or an intermediate variable.
Bias fault	×				Greatly increased gyroscope bias drift.
Time offset fault		×	×		Incorrect time used to calculate reference value (magnetic field or Sun vector).
Position offset fault		×	×		Incorrect position used to calculate reference value (magnetic field or Sun vector).
Misalignment fault		×	×	×	Sensor misalignment during simulated measurement (magnetic field, Sun vector, or star vectors).

6.4 | Scenarios

To evaluate the performance of various fault detection methods, several scenarios are set up which can be simulated to encompass different behaviours seen in CubeSats. This includes both varying satellite orbit and attitude conditions, as well as different fault modes. First, in subsection 6.4.1, the various scenarios for the attitude behaviour of the satellite are discussed. These cases are used to generate the reference data from STK. This is followed by subsection 6.4.2, which describes the various faults that are applied on top of the sensor models.

The combination of STK scenarios and fault cases leads to a matrix of situations. Due to the expense in evaluating even a single situation with all methods, only a limited set of scenarios and fault cases is selected, representing various realistic mission phases and possible faults.

6.4.1 | STK Scenarios

Within STK, various scenarios are simulated in which a typical CubeSat might be situated. Each case consists of a 24-hour period, with reported variables each second, as described in subsection 6.1.2. The simulated variants are listed in Table 6.4.

Table 6.4: Simulated scenarios using STK

Scenario	Description
Default	Nominal orbit with the sensor aligned to nadir, with the smallest side facing the ECI velocity vector.
Target tracking	Same as default, except the spacecraft is oriented such that the sensor tracks predefined ground targets when nearby.
Safe	Spacecraft oriented such that the normal of the largest side with solar panels tracks the Sun vector.
Tumbling	Spacecraft is in a precessing spin of ~ 1 [rev/min].

In the default case, the satellite maintains an attitude in which the sensor is aligned with the nadir-direction and the smallest face of the CubeSat is aimed in the velocity direction to minimize drag, as depicted in Figure 6.1. This scenario describes a nominal situation with no sudden deviations.

In the target tracking case, the satellite maintains the same attitude as in the default case, until it approaches specified ground target locations. Once nearby, the satellite rotates to aim the sensor cone onto the ground target, and tracks it until it leaves line-of-sight. This scenario describes nominal operation of a CubeSat used for Earth observation, which requires sensor pointing to perform ground measurements. A VBScript file is used to define a simple PD controller using the target and the current attitude. More details on the simulation of target tracking can be found in Appendix C.

In the safe case, the satellite tracks the Sun position vector such that the largest side with solar panels is always facing the Sun. In this case, a safe mode is simulated where the satellite simply maintains power generation.

In the tumbling case, the satellite follows a precessing spin about its longest axis. The satellite spins with a rate of 1.2 revolutions per minute, while the precession has a rate of 0.6 revolutions per minute, at a nutation angle of 30 degrees. This simulates the CubeSat in a tumbling state, which is typically the case after deployment from the launch vehicle, or when the ability to control the attitude is lost.

These four scenarios describe common situations a CubeSat could be placed in. The default case presents smooth, non-deviating attitude behaviour. The simulation of the safe mode is similar, except the satellite is rotated from its default position and is tracking the Sun with one face. The target tracking case adds periodic, non-repeating movements based on which ground target it is approaching. Finally, the tumbling scenario adds a case with fast and complex movement.

6.4.2 | Fault Cases

Using the reference data for each case in subsection 6.4.1, simulations of the sensor fusion process can be performed. For each case, different faults are applied to the reference data. These faults are listed in Table 6.5. They are selected to cause varied effects on the sensor output, and approximate the behaviour seen in real CubeSat missions, such as described in section 1.1.

Table 6.5: Simulated fault cases

Case	**Description**
No faults	No faults are applied, only regular sensor models are used.
Star tracker stuck	Star tracker repeatedly stuck at constant value.
Magnetometer zero	Magnetometer repeatedly stuck at zero value.
Sun sensor axis	Sun sensor repeatedly has faulty axis, stuck at zero.

The case in which no faults occur serves is used to train the unsupervised methods, which requires nominal, non-faulty data. The unsupervised methods can learn the patterns of this nominal data to then identify when deviations from this pattern occur.

While it is classified as an abrupt fault in the sense that it has a sudden onset, the star tracker stuck fault causes the anomalous measurements to slowly deviate from the true values, especially when the underlying data is slow-varying. This is the case for the attitude of a CubeSat, which typically does not vary quickly. Thus, when the output becomes constant, it slowly separates from the true behaviour.

The magnetometer fault is another abrupt fault which represents a sudden large deviation to the system. Instead of a growing deviation, the sensor output now immediately jumps from the true value to zero. This large shock is generally easier to detect than the gradual behaviour of the stuck fault.

Finally, the Sun sensor fault is another abrupt fault, but instead of completely losing the sensing ability, the sensor still provides measurements using the corrupted axis. The magnitude of the deviation caused by this fault depends on the state of the spacecraft. If the measurement is contained entirely within the two functioning axes, losing the third axis has no effect. However, if the third axis does contain part of the measurement, it will affect the sensor output.

The faults are repeated with varying lengths and intervals. Within each scenario, which spans one day, the fault is repeated every 2000 s ($\sigma = 100$ s). The duration of each occurrence is set to 300 s ($\sigma = 50$ s). With these settings, the faults occur roughly 43 times in each scenario, with enough time between occurrences for the system to return to a stable state.

Each of the fault cases is combined with each of the STK scenarios described in Table 6.4. The case where no faults are applied is used as the training data. The training process is described in more detail in section 8.2. The remaining three fault cases are the test cases, where each combination with the STK scenarios leads to a total of 12 pairings.

6.5 | Architecture

The data generated by STK, described in section 6.1, is used for the simulation of sensor measurements according to the models given in section 6.2. These are possibly affected by the fault models described in section 6.3, in the combinations specified in section 6.4. The simulated sensor outputs then need to be fused into a state estimate of both the attitude and the gyroscope bias. The simulation structure is visualized in Figure 6.6.

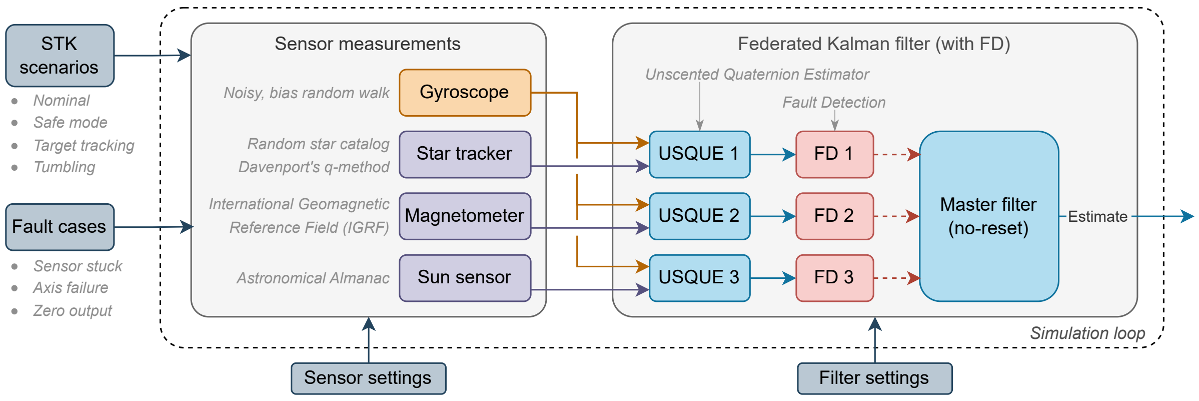


Figure 6.6: Structure of the simulation

The simulation is run for a combination of STK scenario and fault case, as specified in section 6.4. The scenario used in STK determines the basis for the data that the sensors are simulated on top of. In the simulation, all sensors are assumed to operate at the same update rate of 1 Hz to simplify code timing.

The gyroscope, star tracker, magnetometer, and Sun sensor are simulated according to the models described in section 6.2. This allows for realistic injection of a fault based on the current fault case. The gyroscope is combined with each of the remaining absolute sensors in a local filter based on USQUE. This algorithm is commonly used in literature involving the FKF for attitude determination [28, 52]. Here, the gyroscope acts as the reference system, and its angular velocity is used to propagate the state. It typically provides measurements at a fast rate, providing a consistent baseline for all local filters. While this introduces a single point of failure, this is the typical approach when the FKF is applied [90, 28]. The use of the gyroscope requires the filters to estimate its bias, besides the attitude itself.

The three estimates generated by the local filters are processed by a fault detection method. The selection of the conventional and ML-based fault detection methods is discussed in chapter 7 and 8 respectively. The application of a fault detection method allows specific local filters to be excluded from the master fusion process. If the local estimate is deemed anomalous, it is not considered. Finally, the master filter fuses the local estimates that passes fault detection. This, combined with the local filters, forms the FKF.

The settings of various sensor parameters that were used for the simulation are provided in Table 6.6. The specified values are based on literature, such as [90], or representative COTS CubeSat sensors.

Table 6.6: Sensor settings used for the simulation

Sensor	Parameter		Value	Unit
Gyroscope	σ_ω	measurement standard deviation[2]	3×10^{-4}	[rad/$\sqrt{s}$]
	σ_β	bias walk standard deviation[2]	3×10^{-5}	[rad/$s^{3/2}$]
	$\beta_{\omega,0}$	initial bias	0.1	[deg/hr]
Star tracker	σ_{star}	measurement standard deviation[3]	4	[arcsec]
	—	FOV[3]	20	[deg]
	—	random catalogue size	2000	[-]
Magnetometer	σ_{mag}	measurement standard deviation[4]	100	[nT]
	—	IGRF generation	14	[-]
	n_{max}	maximum IGRF degree	5	[-]
Sun sensor	σ_{Sun}	measurement standard deviation[5]	0.5	[deg]
	—	required solar intensity	50	[%]

Besides the sensor parameters, the USQUE and FKF also present several parameters. The settings used for the local and master filters in the simulation is shown in Table 6.7. The same settings are used for all local filters and the master filter. The use of these parameters for USQUE is further detailed in Appendix A. The specific values were selected based on literature and tuned through experimentation to obtain a stable system.

Table 6.7: Filter settings used for the simulation

Parameter		Value	Unit
$\hat{q}_0^+$	initial quaternion estimate	taken from STK	
$\hat{\beta}_0^+$	initial bias estimate	0.1	[deg/hr]
P_{att}	initial attitude covariance elements	1×10^{-3}	[-]
P_{bias}	initial bias covariance elements	0.1	[(deg/hr)^2]
$\hat{\sigma}_q$	quaternion standard deviation estimate	0.01	[-]
$\hat{\sigma}_\omega$	gyroscope standard deviation estimate	1×10^{-4}	[rad/$\sqrt{s}$]
$\hat{\sigma}_\beta$	gyroscope bias standard deviation estimate	1×10^{-5}	[rad/$s^{3/2}$]
a	GRP parameter	1.0	[-]
λ	USQUE filter scale	1.2	[-]
—	FKF mode	No-reset	[-]

There is no feedback of information from the master filter to the local filters, meaning the FKF operates in no-reset mode. This means there is no path for a sensor system to affect any local filter but the one it is associated with. The simple structure is the least accurate of the four modes suggested in subsection 4.2.1 [18], which was confirmed through experimentation. However, due to the aforementioned lack of interaction between sensor systems, this mode also provides the best isolation of faults and is recommended for its fault tolerance capacity [90]. Additionally, the deviation in attitude compared to other modes observed during experimentation never exceeded several percent, which was deemed acceptable.

[2]Based on PG400 gyroscope by AAC Clyde Space, piMAGGYRO by SkyFox Labs
[3]Based on Star Tracker by KU Leuven, Sagitta Star Tracker by ARCSEC
[4]Based on CubeMag GEN 2 by CubeSpace
[5]Based on Fine Sun Sensor by Bradford Space, SS200 by AAC Clyde Space

7 Conventional Detection

To determine the relative performance of the ML-based methods under study, two conventional fault detection methods were implemented to serve as the baseline for comparison. These methods do not use ML, but instead rely on the statistical properties of the filters.

As identified in the review in chapter 1, literature first suggests the use of the sensitivity factor, which is detailed in section 7.1. Second, the local filters calculate a residual, which can also be used to determine when faults occur. This is described in section 7.2. Finally, these methods rely on selecting a value for the anomaly threshold. The process with which these were chosen is described in section 7.3.

7.1 | Sensitivity Factor

In a study where the application of the FKF to formation-flying satellites is considered, the use of a sensitivity factor S is suggested to detect whether a local filter should be considered faulty [26]. The variable is similarly applied in other studies [28]. The value can be calculated at each iteration of the filter, as:

$$S = \left(\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F\right)^T \left(P_i + P_F\right)^{-1} \left(\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F\right) \tag{7.1}$$

Here, $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{x}}_F$ are the states of the i-th local and master filter respectively, and P_i and P_F are the state covariance matrices of the i-th local and master filter respectively.

The sensitivity factor is equivalent to the square of the Mahalanobis distance. The sum of the local and master filter covariance, P_i and P_F, represents the worst-case expected spread of $\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F$. The magnitude of the sensitivity factor then indicates whether this difference could be explained by the expected noise distribution, or it might be an outlier. Higher values indicate the point is more likely to be an outlier.

A threshold can be applied, $S > S_{\text{max}}$, to indicate whether the local filter should be excluded from the master filter. The value of S_{max} can be selected based on the Chi-square distribution, and optimized by experiment for the particular application [26].

7.2 | Measurement Residual

Another commonly chosen value for anomaly detection is the measurement residual [61, 26, 54], $\boldsymbol{v}$, which is already computed in each local filter. The residual, or innovation, is the difference between the expected measurement and the observed measurement:

$$\boldsymbol{v} = \tilde{\boldsymbol{y}} - \hat{\boldsymbol{y}} \tag{7.2}$$

Here, $\tilde{\boldsymbol{y}}$ is the observed measurement, and $\hat{\boldsymbol{y}}$ is the predicted measurement.

The covariance of this residual is often also calculated, for example in Equation A.28 for USQUE, or Equation B.12 for AEKF and MEKF. The diagonal of this matrix represents the variance of the residual. A common approach is to divide the measurement residual v by the standard deviations $\boldsymbol{\sigma}_v$ obtained from the diagonal of the residual covariance matrix [98, 26]:

$$\text{ratio} = \left\| \frac{\boldsymbol{v}}{\boldsymbol{\sigma}_v} \right\| = \left\| \frac{\tilde{\boldsymbol{y}} - \hat{\boldsymbol{y}}}{\boldsymbol{\sigma}_v} \right\| \tag{7.3}$$

Since the mean of the residual should be zero, this metric is equivalent to the commonly used z-score. The ratio captures how large the residual is compared to how large it is expected to be due to uncertainty in the system. By using the variance of the residual, the calculated ratio becomes a simple metric that indicates how many standard deviations away the measurement is from the prediction. Points with a ratio of below 3 are expected to occur 99.7% of the time. When points continuously exceed some threshold, they can be classified as anomalies [98, 26].

A limitation of detecting faults through the measurement residual in the FKF architecture, is that each local filter typically relies on a small subset of sensors. This can cause instances where no measurements are available, for example from Sun sensors when the spacecraft is in eclipse. No measurement residual can be generated, thus fault detection is not possible. One remedy is to consider the sensor faulty during these periods, though this would then discard the propagated state from the master filter.

7.3 | Threshold Selection

Both conventional methods rely on the manual selection of a threshold which, if exceeded, indicates an anomaly has occurred. Choosing a value for this threshold requires knowledge about the expected values during normal operation, and is a trade-off between detection speed and the amount of false positives that are caught.

With a low threshold, anomalies are detected quickly, but more false positives are introduced. A high threshold instead reduces the number of false positives, but it takes longer for anomalies to cross it. This is visualized in Figure 7.1. Here, the effect of the threshold value on the detection performance is shown, using the sensitivity factor in the default scenario where the star tracker experiences the stuck fault as an example.

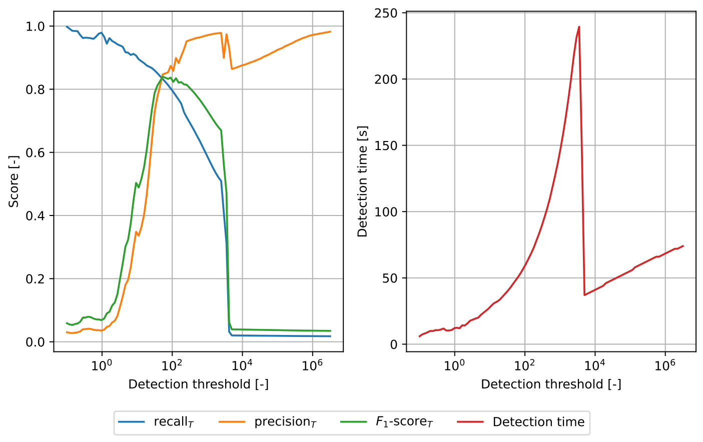


Figure 7.1: Threshold sweep for the sensitivity factor, applied to the default scenario with the star tracker stuck fault

The expected behaviour is observed. With an increasing threshold value, the detection time grows. At the same time, the detection accuracy increases until a maximum accuracy is reached. Further increasing the threshold leads to a slow decrease in accuracy, until at some point the detection fails completely. The value of the threshold thus determines the detection performance, but the conventional methods require a manual selection.

The measurement residual ratio inherently provides a statistical measure. The value of the threshold determines how many points are captured. A 3σ threshold was selected to capture 99.7%

of nominal points. When the measurement residual ratio exceeds this value, it is considered anomalous. This is a typical approach when a residual-based detection is implemented [98].

The threshold for the sensitivity factor was selected using a statistical hypothesis test. Under nominal operation, the difference between the local and master filter states, $\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F$, follows a normal distribution described by the sum of their covariance matrices, $P_i + P_F$:

$$\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F \sim \mathcal{N}(0, P_i + P_F) \tag{7.4}$$

The measure follows a chi-square distribution based on the degrees of freedom (DOF) and false alarm probability α. The sensitivity factor uses the 6-dimensional state, leading to a distribution with 6 DOF. Similar to the selection of the measurement residual ratio threshold, a 3σ limit is used. This corresponds to a false alarm probability of $\alpha = 0.27$%. This leads to the following threshold:

$$S_{\text{max}} = \chi^2_{\text{DOF},1-\alpha} = 20.06 \text{ [-]} \tag{7.5}$$

The selected thresholds of the conventional methods are summarized in Table 7.1

Table 7.1: Selected thresholds for the conventional fault detection methods

Method	**Threshold**	**Description**
Sensitivity factor	20.06	χ^2-distribution with 6 DOF and $\alpha = 0.27$%
Measurement residual	3	3σ capture of 99.7% of points

8 ML-based Detection

The conventional methods are used as the baseline for comparison to the ML-based methods. In this chapter, two ML-based methods are selected through a trade-off in section 8.1. Following this, the selection of features for training is described in section 8.2, accompanied by the training process itself. Finally, the various hyperparameters of the selected methods are tuned in section 8.3.

8.1 | Method Selection

To improve the performance of the fault detection within the FKF when compared to the conventional methods, a suitable ML-based classifier ideally incorporates the feature history to extract more information, instead of relying only on the current set of features. Many existing methods can be adapted to incorporate the history through a sliding window, while others are purpose-built to handle historical data.

In this section, a trade-off is performed between several candidate methods, based on a set of criteria. These criteria are first described, followed by an evaluation of the candidates. This leads to the trade-off, where a sensitivity analysis is performed to confirm the reliability of the results.

8.1.1 | Criteria

In order to compare the candidate methods, a set of criteria was defined for the trade-off. These aim to envelop the important characteristics of each method, relevant to the use within an FKF for fault detection. All criteria are described in the following sections, and an overview is provided afterwards.

Accuracy Each ML method has strengths and weaknesses. Combined with the general level of performance of a method, this can indicate what level of accuracy the method might achieve in this application context. Accuracy is a fundamental metric in fault detection, since the primary objective is to correctly identify faults while minimizing false positives and negatives. For this application, inaccurate detections can lead to unnecessary exclusion of a sensor group, while allowing faults to go undetected affects the final solution. The expected performance depends strongly on the data and type of fault, as well as the architecture of the method.

Temporal Modelling This criterion evaluates the ability of a method to consider historical information. This criterion is related to the accuracy criterion. However, it is used separately to place importance on early detection and reduction of false alarms, especially for gradual or intermittent faults. Methods without the ability to inherently incorporate temporal context generally performance worse in this area than those that do [35]. These methods can still use sliding windows, but this does not fully replace the modelling of sequences directly.

Online Capability As part of **RQ4**, an important consideration for each method is the computational load it would impose on the spacecraft when deployed in orbit. Spacecraft typically operate with limited processing power, so an algorithm with high accuracy but excessive computational demand might be infeasible. This criterion balances performance with expected operational viability.

Interpretability This criterion refers to the transparency of a model during both training and operational use. Understanding the behaviour of a safety critical system is important in ensuring its performance across all cases. Methods with interpretable outputs can facilitate human experts when dealing with flagged anomalies after the fact. The application of ML tools can often introduce behaviour that is hard or impossible to predict. Thus, a helpful metric is the degree to which the method can be understood by a human.

Robustness The final criterion considers the robustness of the method. This encompasses a variety of characteristics of the method. First, some methods are susceptible to noise or outliers, which can greatly affect performance. Second, most methods require tuning of specific parameters to improve the performance of the classifier. This tuning can vary depending on the operating context, and requires expert interaction to set up. Some methods are more sensitive to variation of these parameters than others. Thus, this criterion scores a method on how well it handles unclean data, and the level to which an expert is required to adjust the algorithm.

Overview

The criteria defined in the previous sections are summarized in Table 8.1. Weights are assigned to each criterion, which are multiplied with the respective scores for each method to obtain the total weighted score.

Table 8.1: Criteria for the ML-based method trade-off

Criteria	Weight	Description
Accuracy	25%	Expected performance
Temporal modelling	20%	Ability to extract information from sample history
Online capability	25%	Resource usage and ability to run in real-time
Interpretability	15%	Degree to which output can be understood
Robustness	15%	Resilience to unclean data, need for tuning

The selected weights reflect the prioritization between performance and deployability. Accuracy and online capability are the two criteria that directly correspond to this balance, and are thus heavily weighted. This is followed by the ability of a method to consider temporal context, which recognizes the importance of capturing time-varying patterns in contrast to point-based detectors. Finally, interpretability and robustness share the same weight. Once a model is validated, some black-box behaviour can be tolerated if the output is shown to be reliable and consistent. Similarly, if a method is shown to handle diverse and uncertain conditions, this provides confidence in the approach. The selected weights thus balance technical performance and practical deployment, which aligns with the requirements for its use within the FKF on a CubeSat.

The selected criteria and weights together provide a comprehensive framework for assessing the various candidate methods. They capture the essential differences between detection performance, operational feasibility, and practical use within the FKF structure.

8.1.2 | Candidates

Several candidate methods were selected for the trade-off. They were chosen to span the space defined by the criteria in the previous section. The specific application context of FKF anomaly detection in a spacecraft was considered, which presents several baseline requirements.

Not included in the trade-off are supervised methods. These detectors require labelled examples of every fault type for training, which is typically not available for most space missions. Additionally, these methods have difficulty in detecting unforeseen anomalies not included in the training data.

First, the OC-SVM is a classical, well-studied method. This distribution-based method is relatively lightweight and serves as a common baseline across ML literature. The iForest is a tree-based detector. This method presents an alternative density-based methodology to the OC-SVM, while still differing in architecture from parametric or deep-learning models. The LOF is a distance-based method, again a fundamentally different approach to the previous two detectors, considering the relative proximity of samples.

The LSTM predictor is a temporal prediction-based model, directly addressing the limitations of the other point-wise methods, both conventional and ML-based. This allows the method to detect slow-developing faults more rapidly. The final candidate, the LSTM-AE/VAE, is another temporal, deep-learning method. It completes the spectrum of candidates with a reconstruction-based method.

The selected candidates have methodological diversity. They use fundamentally different detection principles, including methods from the three major categories defined in section 3.3, as is summarized in Table 8.2. The LOF represents the distance-based methods, while the iForest and OC-SVM represent density-based methods. The LSTM predictor and LSTM-AE/VAE are prediction-based detectors. The candidates span lightweight to heavy models, and include both point-wise and temporal detectors. They respect the mission constraints, being unsupervised and with a light to moderate expected computational footprint. The methods are detailed in section 3.4.

Table 8.2: Candidates for the ML-based method trade-off

Category	**Group**	**Method**
Density-based	Distribution-based Tree-based	OC-SVM iForest
Distance-based	Proximity-based	LOF
Prediction-based	Forecasting-based Reconstruction-based	LSTM predictor LSTM-AE/VAE

In the following sections, these candidate methods are scored on the selected criteria. A score of one through five is given based on literature and relative comparison between the methods. This allows for the methods to ultimately be compared numerically. A sensitivity analysis is included to highlight the variance in the results when the scores are adjusted.

One-Class Support Vector Machine The OC-SVM generally achieves moderate performance because it models nonlinear boundaries effectively for well-defined nominal data, though this depends on the definition of the kernel (3/5) [100, 80]. It lacks the inherent ability to consider the temporal history except through external methods such as a sliding window (2/5), where it is specifically noted that the method cannot capture the spatial and temporal dependencies simultaneously [100]. Its inference is relatively efficient, though it scales with the number of support vectors and can become slow for large models (3/5) [80, 101, 81]. The method is robust if the parameters are well-chosen, though this requires tuning of very sensitive hyperparameters (3/5) [80]. Due to its deterministic nature, this method is among the most stable [35]. The output of the OC-SVM is somewhat interpretable if a linear kernel is used, but when using a nonlinear kernel such as a RBF, the resulting boundary and support vectors are difficult to understand (2/5) [35]. The scores are summarized in Table 8.3.

Table 8.3: Trade-off scores for OC-SVM
Scored criteria (1–5), higher is better

Candidate	**Accuracy**	**Temporal**	**Online**	**Interpret.**	**Robust**
OC-SVM	3	2	3	2	3

Isolation Forest The iForest performs similarly to the OC-SVM in terms of detection accuracy (3/5) [35, 101, 81]. Its inference is more efficient, though it depends significantly on the structure of the model (4/5) [81, 101]. The method also has a minimal amount of hyperparameters that require tuning, and is generally resilient to noise (4/5). Inherent temporal awareness is absent similar to the OC-SVM, where external alternatives are needed (2/5). Finally, the partitioning of features provides a decent indication of the output structure, though this becomes obscured when using a large amount of trees (4/5). The scores are summarized in Table 8.4.

Table 8.4: Trade-off scores for iForest
Scored criteria (1–5), higher is better

Candidate	Accuracy	Temporal	Online	Interpret.	Robust
iForest	3	2	4	4	4

Local Outlier Factor The LOF is slightly worse in accuracy compared to the previous methods, specifically in higher dimensions (2/5) [81, 101]. Additionally, inference is very inefficient due to the need to perform neighbour searches each time a sample is added (1/5), with one study showing a 75 times increase in inference time compared to the iForest [81]. The method provides no inherent temporal aspect (2/5) [35]. Its density ratio based on the feature space offers a decent interpretable measure, though it is not trivial to trace it back to specific behaviour (3/5). Its robustness is limited by the need to select a neighbourhood size that balances noise and sensitivity characteristics (2/5). The scores are summarized in Table 8.5.

Table 8.5: Trade-off scores for LOF
Scored criteria (1–5), higher is better

Candidate	Accuracy	Temporal	Online	Interpret.	Robust
LOF	2	2	1	3	2

LSTM Predictor Forecasting using LSTM significantly improves the modelling power, both in accuracy (4/5), and especially temporal modelling by explicitly learning sequential dependencies (5/5) [35]. The prediction- and reconstruction-based approaches were found to be the best candidates for anomaly detection in time-series data so far in this respect [35]. They can identify both abrupt and gradual faults well. Their inference requires a single forward pass, the computational cost of which depends highly on the model architecture, but involves numerous matrix operations (3/5) [102]. Interpretability is poor since the anomaly scoring is hidden in the network weights, though the prediction error is a useful metric (2/5). Finally, the robustness depends on the design of the network, though once trained the inference stability is typically good. The method is able to handle noisy data reasonably well, but also contains several hyperparameters that require tuning (3/5) [35]. The scores are summarized in Table 8.6.

Table 8.6: Trade-off scores for LSTM predictor
Scored criteria (1–5), higher is better

Candidate	Accuracy	Temporal	Online	Interpret.	Robust
LSTM predictor	4	5	3	2	3

LSTM-AE/VAE The approach using LSTM-AE/VAE achieves the highest expected accuracy due to the ability to learn patterns with probabilistic likelihood, an improvement compared to the next-step prediction of the LSTM predictor (5/5) [35]. Similar to the LSTM predictor, they inherently consider temporal context, though large transitions can be smoothened out (4/5). This method additionally has similar computational demands, though autoencoders can be more expensive (2/5). Interpretability is similarly low as reconstruction errors are difficult to trace back to specific behaviour, through the reconstruction error is a useful metric (2/5). Finally, the method is robust, though they introduce several hyperparameters that require tuning (3/5). The scores are summarized in Table 8.7.

Table 8.7: Trade-off scores for LSTM-AE/VAE
Scored criteria (1–5), higher is better

Candidate	Accuracy	Temporal	Online	Interpret.	Robust
LSTM-AE/VAE	5	4	2	2	3

8.1.3 | Trade-off

With the scores defined for each candidate method in the previous sections, a comparison is shown in Figure 8.1. The scores are also summarized in Table 8.8, where the weighted total score is calculated using the scores and weights for each criterion.

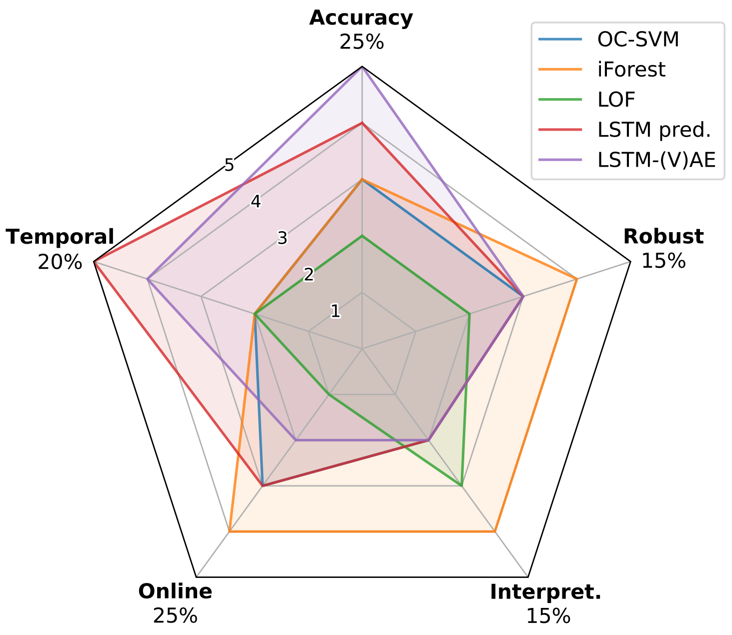


Figure 8.1: Comparison of trade-off scores for ML-based methods

Studying the results, the two options using LSTM are superior to the other methods in the expected accuracy and incorporation of temporal context. However, this comes at the cost of being less computationally efficient. On top of this, they are not as interpretable as the other approaches. In general, the iForest is a well-rounded option, except for the lack of temporal modelling.

The weighted scores show the LSTM predictor being ranked the highest, shortly followed by the iForest, then the LSTM-AE/VAE. The LSTM predictor is ahead by 0.15 points, and the gap between the iForest and LSTM-AE/VAE is only 0.05. The remaining methods, the OC-SVM and LOF, rank much lower and are behind by over 0.60 points.

The scores indicate that a sensitivity analysis is warranted. While a ranking of methods was generated, slight differences in the criteria weights or candidate scores could affect the results significantly. In the following section, this is investigated before concluding the trade-off.

Table 8.8: ML-based method trade-off matrix
Scored criteria (1–5), higher is better

	Accuracy	**Temporal**	**Online**	**Interpret.**	**Robust**	
Candidate	25%	20%	25%	15%	15%	**Total**
OC-SVM	3	2	3	2	3	2.65
iForest	3	2	4	4	4	3.35
LOF	2	2	1	3	2	1.90
LSTM predictor	4	5	3	2	3	3.50
LSTM-AE/VAE	5	4	2	2	3	3.30

8.1.4 | Sensitivity Analysis

The trade-off results have several of the candidates rank with very similar total scores. In order to determine how sensitive the outcome is to changes in the criteria weights and candidate scores, a sensitivity analysis was performed. This is described in the following sections.

Criteria Weights The criteria weights were each varied to observe how the trade-off outcome would change. Each criterion weight was adjusted by $\pm 5\%$ (p.p.) and the new total scores for each method were recorded. These scores after the adjustment are shown in Figure 8.2 and Table 8.9.

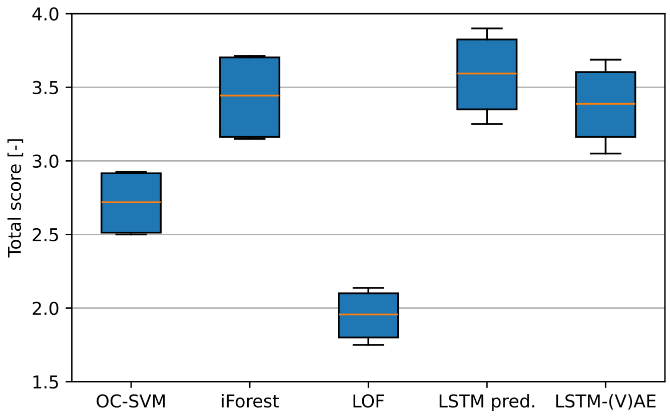


Figure 8.2: Score sensitivity to varying criteria weights by ± 5 p.p.

Table 8.9: Score sensitivity to varying criteria weights by ± 5 p.p. ($n = 10$)

Candidate	**Mean** [-]	1σ [-]
OC-SVM	2.715	0.196
iForest	3.435	0.257
LOF	1.950	0.153
LSTM predictor	3.585	0.259
LSTM-AE/VAE	3.380	0.245

Varying the criteria weights shows that the LSTM predictor, iForest, and LSTM-AE/VAE are indeed closely matched in score. The LSTM predictor pulls ahead noticeably, with the iForest being subtly ahead of the LSTM-AE/VAE method. When the weight of the temporal modelling criterion is lowered, the iForest surpasses both methods and ranks the highest. There is no case in which the LSTM-AE/VAE achieves the same. The OC-SVM and LOF methods significantly trail behind the others, with the LOF method having by far the lowest score range.

Candidate Scores Besides the criteria weights, the scores for each candidate and criterion were also varied. Each score was adjusted by ± 1 point to study its effect on the rankings. Again, the ranking and its standard deviation after the adjustment is shown in Figure 8.3 and Table 8.10.

The variation of the candidate scoring shows similar results to the variation of the criteria weights in Figure 8.2. The OC-SVM and LOF methods remain at the bottom of the ranking. Again, the LSTM predictor shows the best results on average, with the iForest now winning the trade-off in 10 out of 50 runs. The LSTM-AE/VAE method is subtly behind the iForest, but only wins the trade-off in a single case.

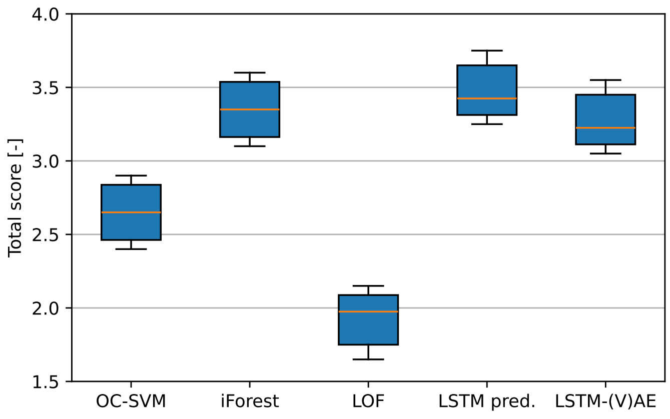


Figure 8.3: Score sensitivity to varying candidate scores by ± 1

Table 8.10: Score sensitivity to varying candidate scores by ± 1 ($n = 50$)

Candidate	**Mean** [-]	1σ [-]
OC-SVM	2.650	0.194
iForest	3.350	0.205
LOF	1.925	0.187
LSTM predictor	3.480	0.194
LSTM-AE/VAE	3.275	0.187

8.1.5 | Outcome

Considering the results of the trade-off and sensitivity analysis, the following summarized ranking is found as shown in Table 8.11. The LSTM predictor is consistently the best ranked method on average, with strong accuracy and temporal modelling at the cost of higher computational load. The iForest is a lightweight alternative with good interpretability, though it lacks inherent modelling of temporal context, besides using a sliding window or similar approach.

The LSTM-AE/VAE algorithm follows closely, with even greater accuracy than the LSTM predictor, but also with a higher computational cost. Finally, the OC-SVM is a balanced option with limited accuracy and efficiency, whereas the LOF method is less accurate and very inefficient.

Table 8.11: ML-based method trade-off outcome

Rank	Candidate
1	LSTM predictor
2	iForest
3	LSTM-AE/VAE
4	OC-SVM
5	LOF

For this research project, the LSTM predictor and iForest methods were selected based on this ranking. The first as an accurate but heavy method incorporating the temporal context, and the latter as a lightweight point-wise alternative. The iForest is used as a point-wise method to emphasize the difference to sequential methods, and additionally serves as a more direct comparison to the conventional approaches.

8.2 | Features and Training

After the selection of the ML-based fault detection methods, the process with which they are trained was developed. Before the method can be used to infer whether a given sample is anomalous, it requires learning from data what is and what is not an anomaly.

Several features were derived from the available data within the simulation, which would also be available during real-world operation. These are first described in the following section. This is followed by a description of the training process for the unsupervised methods.

8.2.1 | Feature Selection

In the distributed architecture of the FKF, each local filter operates on a subset of the sensor suite. In the no-reset mode, as described in subsection 4.2.1, these provide independent state and covariance estimates which are subsequently combined by the master filter into the master state and covariance estimate. Fault detection in this structure relies on identifying inconsistency between the local and master estimate, or abnormal behaviour in the local filter itself. To enable data-driven fault detection, a set of features was defined to capture both internal statistical properties of each local filter and their consistency with the master filter. The following sections describe the selected features and provide details on their suitability for fault detection.

Trace of Local Covariance The trace of the state covariance matrix, the sum of its diagonal elements, represents the total uncertainty estimated by the i-th local filter. Under normal conditions, the trace remains within predictable bounds, the range depending on the characteristics of the sensor group and the applied filter.

When a sensor fault occurs, the residual becomes inconsistent with the noise statistics of the model. Consequently, the elements of the covariance matrix tend to increase as the filter attempts to adapt to the unexpected measurements. This increase in total uncertainty is directly captured in the covariance trace.

The first two features, f_1 and f_2, represent the trace of the attitude and gyroscope bias components of the i-th local filter state covariance matrix P_i respectively:

$$f_1 = \mathrm{Tr}\left(P_{i,\text{att}}\right) \quad (8.1) \qquad f_2 = \mathrm{Tr}\left(P_{i,\text{bias}}\right) \quad (8.2)$$

The traces increase significantly when the sensor group becomes less informative. This provides a scalar measure of the confidence of the local filter.

Determinant of Local Covariance The determinant of the state covariance matrix is similar to the trace, differing in that the determinant incorporates correlations between states. The value indicates the generalized variance, or spread of the distribution. In a fault scenario, the uncertainty might not expand only in magnitude, but also in orientation in state space. The determinant complements the trace, being sensitive to any coupling that may arise during faults.

The third feature, f_3, represents the determinant of the i-th local filter state covariance matrix P_i:

$$f_3 = \det(P_i) \tag{8.3}$$

The determinant is sensitive to coupled increases in uncertainty and estimation degradation. It provides geometric information about the spread of the state uncertainty.

Mahalanobis Distance The Mahalanobis distance provides a measure of the statistical consistency between the i-th local filter estimate $\hat{\boldsymbol{x}}_i$ and the master estimate $\hat{\boldsymbol{x}}_F$. Under normal conditions, the local and master filters should remain consistent, where the differences between their states are covered by the expected uncertainty. A fault in a local filter leads to a local estimate that diverges from the master filter beyond the expected statistical limit.

The fourth feature, f_4, uses the Mahalanobis distance to capture the divergence between the i-th local filter and master filter:

$$f_4 = \sqrt{(\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F)^T (P_i + P_F)^{-1} (\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F)} \tag{8.4}$$

The Mahalanobis distance increases when the local estimate becomes inconsistent with the master estimate.

Quaternion Difference Both the local and master filters produce an estimate of the attitude of the spacecraft as a quaternion. The angle between the quaternions estimated by the i-th local filter and master filter provides a measure of the attitude discrepancy between them.

Sensor faults that affect the estimated attitude will cause the difference between the local and master estimates to grow. This feature captures this inconsistency as a scalar value in the rotational domain.

The fifth feature, f_5, is the angular difference between the quaternion estimates of the i-th local filter and the master filter:

$$f_5 = 2\,\arccos(\hat{q}_i^*\,\hat{q}_F) \tag{8.5}$$

The angular difference captures the attitude estimation deviation between the filters.

Bias Difference The norm of the difference between the local and master estimates of the gyroscope bias provides a similar measure to the angular quaternion difference. The value indicates the disagreement in the estimated gyroscope bias between the local and master filter.

A faulty sensor group can cause its corresponding local filter to compensate for the fault by adjusting the bias estimate incorrectly. This can help distinguish between deviations caused by noise, and systematic sensor faults.

The sixth feature, f_6, is the norm of the difference between the i-th local filter and master filter estimates of the gyroscope bias:

$$f_6 = \|\hat{\boldsymbol{\beta}}_i - \hat{\boldsymbol{\beta}}_F\| \tag{8.6}$$

This difference can show slowly developing faults, and is useful for distinguishing between transient and systematic behaviour.

Local Residual The residual, or innovation, $\boldsymbol{v}$, of the local filter is the difference between the actual measurement $\tilde{\boldsymbol{y}}$ and the predicted measurement $\hat{\boldsymbol{y}}$. During normal behaviour, the residual should behave following white noise with a covariance consistent with the assumed measurement noise.

When a sensor fault occurs, the residuals exhibit increased magnitude or directional bias. Monitoring the norm of the residual therefore provides a direct indication of the statistical consistency of the measurements.

The seventh feature, f_7, is the norm of the residual of the i-th local filter:

$$f_7 = \|\boldsymbol{v}_i\| = \|\tilde{\boldsymbol{y}}_i - \hat{\boldsymbol{y}}_i\| \tag{8.7}$$

The value provides a direct measure of the consistency of the sensor measurements.

Overview

The features described in the previous sections are summarized in Table 8.12. Through the comparison of the local and master filter, as well as the statistical properties of the local filter itself, the selected variables provide insight into the inconsistency and uncertainty of the solution, as scalar values. They provide the relevant information for a model to learn the difference between nominal and anomalous behaviour, where the model learns during training which combinations of features are decisive.

Table 8.12: Selected features for ML training

Feature		Equation	Indication
f_1	Attitude covariance trace	Tr $\left(P_{i,\text{att}}\right)$	Attitude estimate confidence
f_2	Bias covariance trace	Tr $\left(P_{i,\text{bias}}\right)$	Bias estimate confidence
f_3	Covariance determinant	det (P_i)	Spread of state uncertainty
f_4	Mahalanobis distance	$\sqrt{(\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F)^T (P_i + P_F)(\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_F)}$	Local-master consistency
f_5	Quaternion difference	2 arccos $(\hat{q}_i^* \, \hat{q}_F)$	Local-master attitude deviation
f_6	Bias difference	$\|\hat{\boldsymbol{\beta}}_i - \hat{\boldsymbol{\beta}}_F\|$	Local-master bias deviation
f_7	Filter residual	$\|\boldsymbol{v}_i\| = \|\tilde{\boldsymbol{y}}_i - \hat{\boldsymbol{y}}_i\|$	Sensor measurement consistency

The selected features f_1 through f_7 are grouped into a single feature vector $\boldsymbol{f}$:

$$\boldsymbol{f} = \begin{bmatrix} f_1 & \cdots & f_7 \end{bmatrix}^T \tag{8.8}$$

This feature vector can be computed for every time step. For the data where no faults are present, these vectors form the training data for the ML models. During training, the model learns the patterns in the feature vectors to be able to distinguish between nominal and anomalous data. Then, to infer from the model, the feature vector can be calculated for other datasets to classify faults using the trained system.

Before the features are used for training, they should be normalized to a standard range. This pre-processing step transforms the features into a common range such that features with greater value ranges cannot dominate those with smaller ranges [103].

Normalization When all features are normalized to the range of $[0, 1]$, it allows the model to learn which features are more important than others by itself. If one feature had presented a larger possible range of values, it would falsely dominate those with smaller possible ranges. For parametric models, such as the LSTM predictor, the weights and biases would first have to be adapted to overcome this range difference before being able to incorporate the actual anomalous behaviour. This would introduce confusion during the training process and can negatively affect the performance [103].

Some information is lost when the features are normalized, such as absolute scale and outlier magnitude. The ML model does not need absolute units to learn to detect anomalies, more important is how the features behave relative to their typical behaviour. Normalized features still contain temporal patterns and relative magnitude changes. Normalization discards the information that the model should not use for learning, and avoids it being biased toward larger features.

Since each of the features selected above presents a wide range of possible values, they are normalized individually to a range between 0 and 1. Many normalization methods exist, each with a different impact on performance. Several techniques were considered:

- **Z-score**: use the mean and standard deviation to rescale to zero-mean and unit variance.

$$f_i' = \frac{f_i - \mu_i}{\sigma_i} \tag{8.9}$$

- **Min-max**: scale the data between the lower and upper bounds to the range $[0, 1]$.

$$f_i' = \frac{f_i - \text{min}(f_i)}{\text{max}(f_i) - \text{min}(f_i)} \tag{8.10}$$

- **Robust median**: similar to Z-score, but uses the median and interquartile range (IQR).

$$f_i' = \frac{f_i - \text{med}(f_i)}{\text{IQR}(f_i)} \tag{8.11}$$

- **Logistic sigmoid**: use logistic sigmoid function, squishing the outliers exponentially.

$$f_i' = \frac{1}{1 + e^{-q_i}} \quad \text{where} \quad q_i = \frac{f_i - \mu_i}{\nu \cdot \sigma_i} \tag{8.12}$$

In the above equations f_i and f_i' denote the i-th non-normalized and the normalized feature respectively, μ_i and σ_i are the mean and standard deviation of feature f_i respectively, and ν is a parameter for the logistic sigmoid normalization, set to $\nu = 1$.

The robust median normalization method was selected through experimentation as it provides the best representation of the feature ranges once normalized in this case. Since all features, except the angular quaternion difference, show variations across multiple orders of magnitude, their base-10 logarithms were used before applying the robust median normalization. The resulting normalized features vary correctly in the range of $[0, 1]$. The normalized features f_1' through f_7' are once again collected in a vector $\boldsymbol{f}'$ for processing by the models:

$$\boldsymbol{f}' = \begin{bmatrix} f_1' & \cdots & f_7' \end{bmatrix}^T \tag{8.13}$$

The response of four of the features to a stuck fault is shown in Figure 8.4. The depicted normalized features show anomalous behaviour during the fault period. The Mahalanobis distance (f_4') and quaternion difference (f_5') gradually increase over the duration of the fault. The stuck fault causes the measurements and true data to diverge, which is observed as these features increasing from near-zero to near-one. At the same time, the attitude covariance trace (f_1') and bias difference (f_6') show different behaviour. They rapidly rise to a constant value for the duration of the fault, before spiking after the fault ends and returning to nominal values.

Each feature provides the model with different information about the state of the local and master filter. The response of the ML-based detection methods is greatly dependent on these normalized values. Their responses to a variety of faults is analysed in more detail in chapter 9.

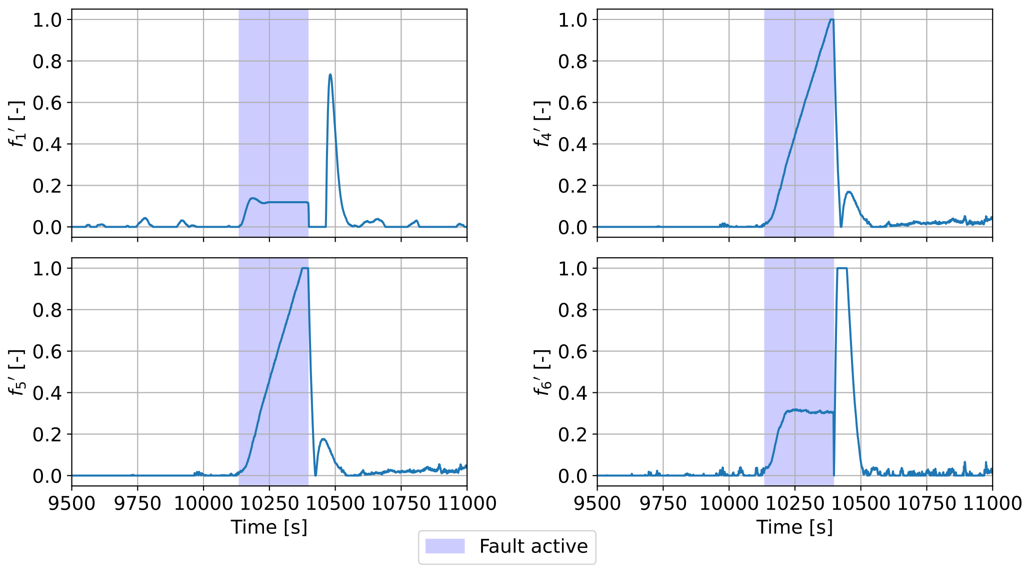


Figure 8.4: Normalized features during an occurrence of the stuck fault

8.2.2 | Training Process

In order to use an ML-based method to detect faults, data is needed to train the method in distinguishing normal from abnormal behaviour. Within the space industry, there is generally a lack of large amounts of real data, especially with labelled instances [36]. Unsupervised methods are able to train on unlabelled, non-anomalous data to learn the normal behaviour. Then, when given more data that does contain faults, it should detect these anomalous events.

Each of the scenarios is simulated once without the presence of any faults and once with the presence of a fault. This creates two datasets: the training and testing datasets. During training, the normalized features are calculated for each timestep. These are used to train the ML-based methods. Due to the different nature of the two selected methods, the iForest and LSTM predictor, their training approaches also differ.

The iForest processes the feature vectors separately by randomly splitting features to generate a multitude of iTrees. The LSTM predictor instead collects a consecutive sequence of samples as well as the sample directly following it. Using back-propagation, it updates the model parameters such that the sample following the sequence is more accurately predicted. Due to memory limitations, training an LSTM predictor on all scenarios at once was not feasible. Instead, the model was trained in stages, separating the data for each of the four scenarios. This means the model does not see data from other scenarios in each training stage.

The Sun sensor is repeatedly unable to provide measurements due to the regular occurrence of eclipse periods. If these periods were to be included in the training data, the models would consider these periods as nominal. However, this creates confusion between the start of a fault and the start of an eclipse period. To avoid this, the eclipse periods are excluded from the training dataset for the Sun sensor. This has the consequence that eclipse periods are considered anomalous for this sensor type. This is acceptable since during these periods there are no measurements available.

Both the iForest and LSTM predictor provide an anomaly score. The iForest bases this score on the depth within the iTrees, while the LSTM predictor bases this on the prediction error. To avoid setting a manual threshold, as is required for the conventional methods, the following approach is used. After training, the anomaly scores for the training data are computed, where the upper 3σ bound is taken to be the threshold for inference, similar to the approach taken for the conventional methods. This captures 99.7% of the normal behaviour, and anomalies are expected to exceed this threshold consistently.

Before calculating the detection performance, a post-processing step is applied which prevents single outliers from tripping the detection. Three consecutive samples scoring above the threshold are needed to trip the detection. This step is applied to the output of both the conventional and ML-based methods. This filter prevents outliers of one or two consecutive samples from tripping detection.

8.3 | Tuning

The two selected ML-based detection methods each have several parameters that can be adjusted to change their performance. A range of values is evaluated to determine their optimal values. In the following sections, the parameters of the iForest and LSTM predictor are tuned.

The default scenario in which the star tracker exhibits the stuck fault is used to evaluate the performance of the methods at each parameter iteration. For abrupt, large deviations, such as with the zero fault, the effect of tuning is less visible. The obvious nature of these faults leads to rapid detection with most methods. Instead, with the stuck fault, the deviation slowly grows, leading to more variation in detection time. The tuning has a more visible effect on the detection performance. The default scenario is used due to the absence of any rapid movement that could be falsely seen as anomalous.

8.3.1 | Isolation Forest

The iForest method contains several parameters that should be tuned to optimize its performance. The parameters considered for the tuning are listed in Table 8.13. Also listed is the range in which each parameter was varied, observing its effect on the scoring metrics.

Table 8.13: Tunable parameters for the iForest

Parameter	Description	Tune range
Ensemble size	Number of iTrees that are trained in the iForest	25 to 200
Maximum samples	Maximum portion of the total samples used by each iTree	5% to 40%
Contamination	Assumed proportion of outliers in the training dataset	0.01% to 10%

The ensemble size specifies the number of individual trees the iForest is made up of. When it was originally developed, it was noted that the iForest becomes accurate for a relatively low number of iTrees [40]. The lower the amount of trees, the more efficient the iForest is due to the lower number of required comparisons. The maximum number of samples dictates how many of the total samples are available to each iTree. This parameter helps control the isolation process and can allow an iTree to become specialized, where each tree operates on a different set of samples which can include different anomalies or even no anomalies at all [40]. Finally, the contamination parameter specifies the expected fraction of outliers in the training data, affecting the calculation of detection threshold.

Ensemble Size In Figure 8.5 the influence of the contamination parameter on the performance metrics is shown. From the graphs, there is a clear relationship visible between the contamination and performance. A positive correlation exists between the contamination and recall score. The precision score shows a local maximum at a contamination of 0.1%. Consequently, the F_1-score shows a similar maximum at 0.1%. However, the detection time at this value is also maximal, and rapidly decreases with higher contamination values while the F_1-score decreases more slowly.

The number of iTrees also influences the performance. Especially at lower values such as 25 and 50 trees, the scores are significantly different from the other values. Starting at 75 trees and higher, the scores show little variation when increasing the number of trees further.

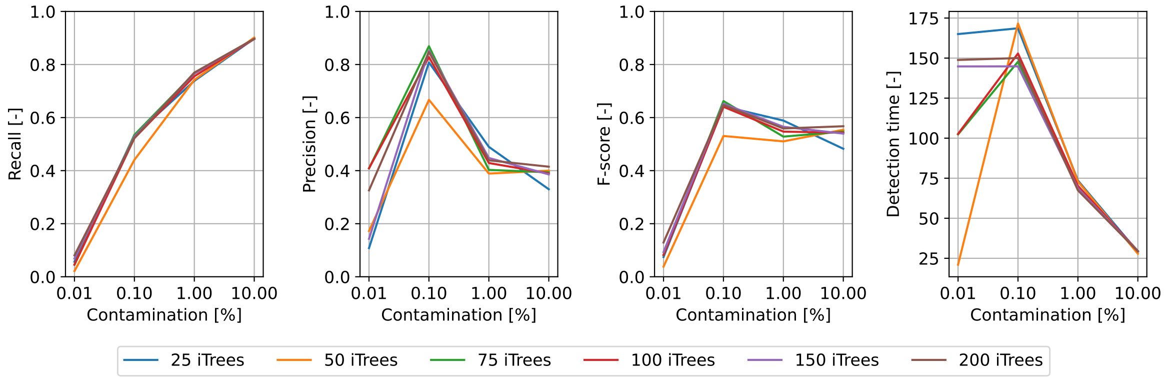


Figure 8.5: Effect of contamination on iForest performance (maximum samples: 20%)

The influence of the maximum sample parameter was also determined, visible in Figure 8.6. Again, there is a clear relationship between the parameter and the performance. Similar to the contamination, an increasing value generally leads to a higher recall score. The effect on the precision score is smaller, where the behaviour is more erratic for lower numbers of iTrees. With 150 or 200 trees, the behaviour is more stable, and shows a local maximum between 40 and 60%. Consequently, the F_1-score grows with higher values, with a maximum between 40 and 60%. The detection time generally reduces with higher values for the maximum sample parameter. Within the range where the F_1-score is maximum, the detection time is between 100 and 125 seconds.

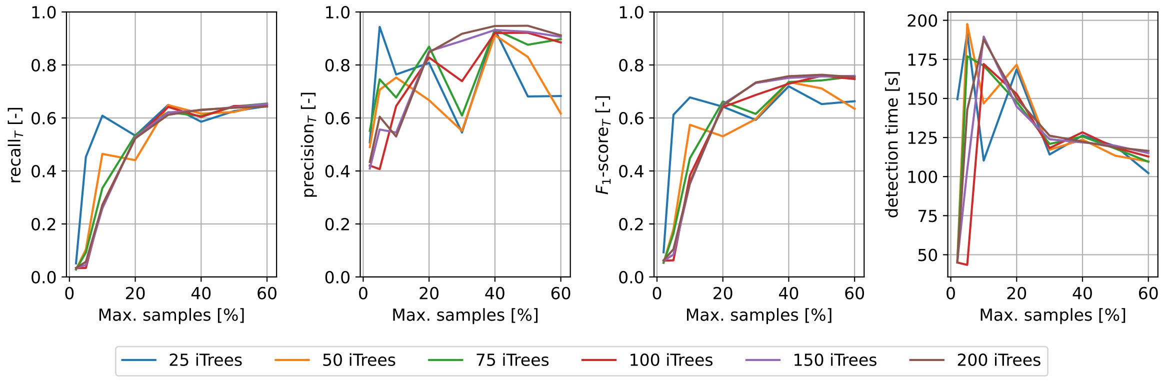


Figure 8.6: Effect of maximum sample parameter on iForest performance (contamination: 0.1%)

From the investigation of the contamination parameter, it was determined that a number of trees above 75 is preferable. For the maximum sample parameter, the number of trees should be 150 or more to provide stable behaviour. Therefore, the optimal number of trees is set to be 150.

Contamination and Maximum Samples In order to select optimal values for the remaining two parameters, contamination and the maximum sample size, their combined influence over the performance metrics is depicted in Figure 8.7. The trends identified before again appear in these matrices. Increasing contamination and maximum sample percentage both lead to increased recall. At higher contamination values, 1% or above, the dependence on the maximum sample percentage becomes generally insignificant. To maximize recall, both parameters should be set as high as possible.

The precision matrix shows a clear region with an optimal score, with a contamination of 0.1% and maximum sample of 40 to 60%. Values surrounding this region show moderate results, while more distant regions show very low precision. To maximize precision, the contamination should be close to 0.1% and the maximum sample percentage above 40%.

The F_1-score combines the precision and recall metrics, which is optimal for a contamination of 1% and a maximum sample of 60%. The F_1-score at this location is 0.797. Finally, the detection time is generally best for high contamination, where the influence of the maximum sample percentage disappears. At the location with optimal F_1-score, the detection time is 65.1 seconds.

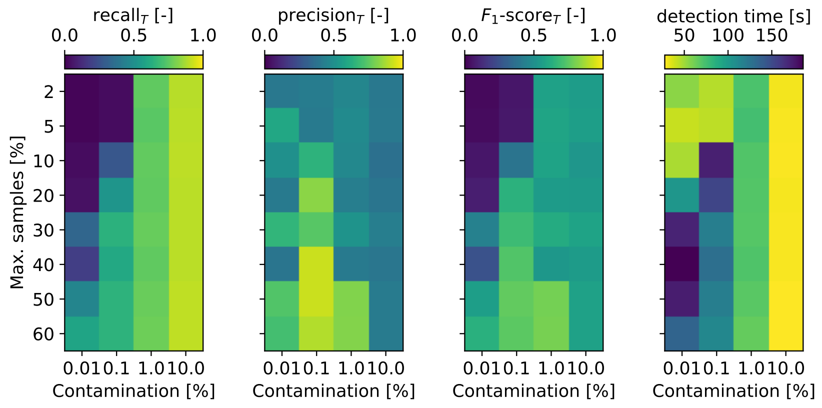


Figure 8.7: Effect of maximum sample and contamination parameters on iForest performance (number of iTrees: 150)

iForest Architecture

In order to optimize the accuracy of the iForest method, the point with the optimal F_1-score is selected. This occurs at a contamination of 1.0% and a maximum sample size of 60%. On top of this, the number of iTrees was previously selected to be 150. This leads to the tuned parameters summarized in Table 8.14.

Table 8.14: Tuned iForest architecture

Parameter	Value
Number of iTrees	150
Maximum samples	60%
Contamination	1%

8.3.2 | LSTM Predictor

The LSTM predictor contains more tunable parameters than the iForest. These all have a unique influence on the performance of the model. The parameters considered for tuning are listed in Table 8.15, along with the ranges that they were varied in for tuning to observe the effect on performance.

Table 8.15: Tunable parameters for the LSTM predictor

Parameter	Description	Tune range
Loss function	Function to minimize during training	MSE, MAE, MSLE
Learning rate	Size of weight adjustments during training	10^{-7} to 10^{1}
No. of LSTM cells	Number of stacked LSTM cells	1 to 7
Hidden layer size	Number of neurons in hidden layer	8 to 256
Window size	Length of input sequence from history	2 to 100
Dropout	Probability of dropout between LSTM layers	0 to 50%

Loss Function During training, the model adjusts its weights and biases to minimize some loss function. The choice of this function influences how the model learns to approximate the system dynamics. The error that is calculated between the predicted and observed data guides the training process and shapes how the model responds to anomalies. As a result, the definition of this loss function is part of the tuning process for optimizing the performance of the model.

Three commonly used regression losses for prediction were evaluated:

- **Mean absolute error (MAE)**: uses the absolute difference between the predicted and observed features.
- **Mean squared error (MSE)**: uses the squared difference between the predicted and observed features.
- **Mean squared logarithmic error (MSLE)**: uses the squared difference between the logarithms of the predicted and observed features.

These three methods were selected because they represent the primary families of error penalization: linear (MAE), quadratic (MSE), and logarithmic (MSLE). Each captures a different bias with direct relevance to the anomaly detection. The performance of the three loss functions was assessed using the performance metrics, which is depicted in Figure 8.8.

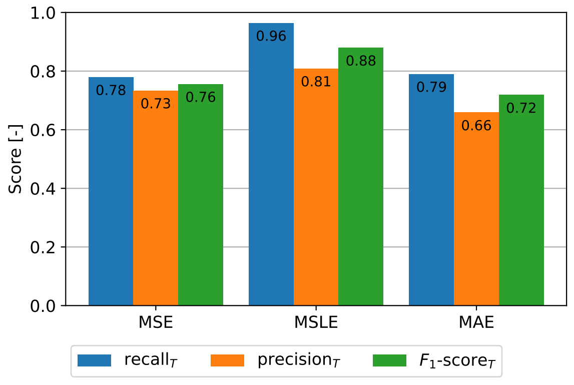


Figure 8.8: Effect of loss function on LSTM predictor performance (window size: 50, no. of hidden layers: 4, hidden layer size: 64, learning rate: 1.92×10^{-3}, dropout: 0%)

The highest score across all metrics was achieved using MSLE (F_1-score$= 0.88$). This outperforms both MSE (F_1-score$= 0.76$) and MAE (F_1-score$= 0.72$). Based on these results, MSLE was selected as the loss function for subsequent development of the model. Its strong recall suggests strong sensitivity to anomalous behaviour, while its higher precision also indicates better distinction between normal and faulty data. This makes MSLE the more effective loss function out of the three assessed methods.

Learning Rate The learning rate (LR) controls the step size of the optimizer during the gradient-based updates to the model weights. This parameter is a primary factor in convergence speed and training stability. To select a suitable LR for the LSTM predictor, the LR range test described by Smith in 2017 [104] is used.

The range test involves steadily increasing the LR over several iterations while recording the loss, in this case the MSLE. The range test produces a characteristic curve showing the relationship between the LR and the MSLE. This is depicted for this case in Figure 8.9.

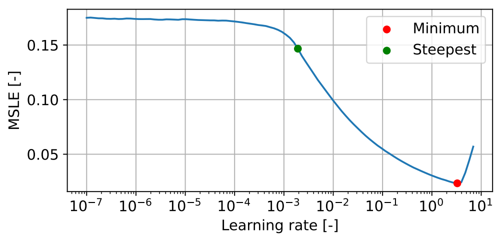


Table 8.16: Identified points on the LR-loss curve for the LSTM predictor

Location	**LR** [-]	**MSLE** [-]
Steepest descent	1.92×10^{-3}	0.1467
Minimum	3.27	0.0234

Figure 8.9: Effect of LR on LSTM predictor training loss

The plot starts with a plateau for low values of the LR. In this region, the steps are too small to reduce the loss. This is followed by a section of decline in MSLE as the LR grows and training starts to become stable. This decline continues up to some minimum loss, at which point the MSLE starts rapidly increasing when the optimizer becomes unstable.

Two points are identified on the graph: the location where the loss declines most rapidly, the steepest negative gradient, and the location at which the loss first starts to diverge, at the minimum loss. These are tabulated in Table 8.16. A common choice for the LR falls between this steepest gradient and the minimum, with typically an order of magnitude subtracted from the minimum to avoid divergence [104].

Selecting an LR is a balance between training speed and robustness. Higher values lead to faster training, but the larger steps can introduce instability. Lower values are more precise, but take longer during training. A conservative LR was selected, favouring robustness over training speed, using the location of the steepest gradient (LR$= 1.92 \times 10^{-3}$) as the base LR for training. This value yields stable and consistent training, avoiding the instability observed for higher LRs.

Number and size of LSTM Cells Stacking multiple LSTM cells increases the depth of the neural network, where the hidden state of one layer serves as the input to the next layer. Increasing the number of stacked cells allows the model to better learn temporal connections, where some layers capture short-term dependencies while other layers model capture long-term patterns [85]. A deeper architecture thus increases the capacity of the model to represent the data, but this also introduces many more trainable parameters. Besides increasing training time, this increases the risk of overfitting, can increase instability, or reduce the sensitivity to subtle faults if the short-term dynamics are obscured by other layers.

In Figure 8.10, the effect of the number of stacked LSTM cells on the performance is shown. A clear dependence is visible. If the number of cells is too small (< 2), the shallow architecture is not sufficient to capture the patterns needed to distinguish anomalies from normal points. On the other hand, if the number of cells is too large (> 6), the detection also fails, showing very high recall (0.998) but very low precision (0.002). This indicates that the model is effectively treating most points as anomalies, which could be consistent with overfitting or other instabilities.

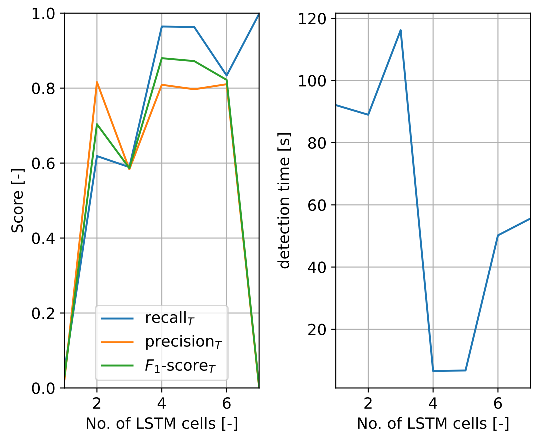


Figure 8.10: Effect of number of cells on LSTM predictor (window size: 50, hidden layer size: 64, learning rate: 1.92×10^{-3}, dropout: 0%)

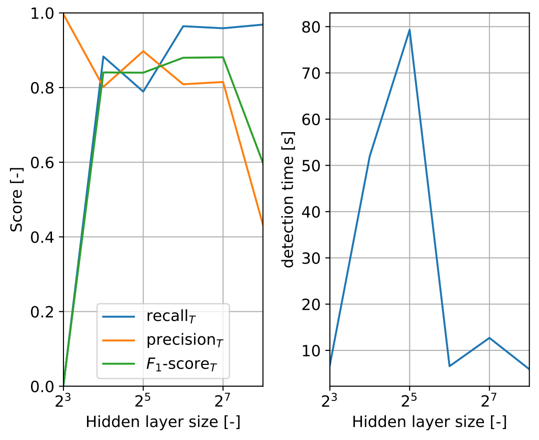


Figure 8.11: Effect of hidden layer size on LSTM predictor (no. of hidden layers: 4, window size: 50, learning rate: 1.92×10^{-3}, dropout: 0%)

Performance improves substantially with two stacked cells compared to a single cell, suggesting that the additional depth enables the model to detect anomalies more accurately. However, further increasing to three cells sees a slight loss in performance, indicating that adding more depth does not guarantee better performance. The strongest overall performance is obtained at either 4 or 5 cells with very similar scores (F_1-score$= 0.880$ and F_1-score$= 0.872$ respectively). Both configurations achieve high recall and strong precision. These models also achieve the fastest detection times (6.58 and 6.73 seconds respectively). This indicates they not only detect the anomalies accurately, but also quickly. This combination suggests that the model has sufficient capacity without introducing unnecessary additional complexity. Finally, when further increasing the number of cells, the performance reduces and eventually collapses.

Based on these metrics, the model with four stacked LSTM cells is selected. It achieves the highest F_1-score (0.880) and fastest detection time (6.58 seconds). A model with five cells performs similarly, but offers no clear advantage to the four-cell model while increasing the required computation, both in training and inference. The four-cell model provides an effective balance between capacity, accuracy, and detection time.

Next, the hidden layer size of the LSTM was tuned. This size determines the dimension of the cell's internal state and, similar to the number of cells, affects the capacity of the model to represent temporal dependencies. A larger hidden layer allows the network to learn more complex patterns by increasing the number of trainable parameters. However, increasing the size also raises the risk of overfitting, increases training time, and can reduce sensitivity to subtle faults. Layers that are too small can lack the necessary capacity to detect anomalies, particularly subtle ones where temporal context is important.

Figure 8.11 shows the effect of the hidden layer size on the performance. Again, a clear dependence is observed. A hidden size of 8 yields very low recall (0.003) but very high precision (0.996), indicating the model almost never correctly predicts anomalies. This suggests the model lacks the necessary capacity. Increasing the hidden size to 16 or 32 shows great increase in the accuracy of the model with high precision and recall. However, at this size, the detection time has also greatly increased (52.0 and 79.3 seconds respectively). The longer delays suggest that, while these configurations have enough capacity for detection, their smaller size may limit the early detection of subtle faults.

The best performance is observed for layer sizes of 64 and 128, with the highest F_1-scores observed in the tested range (0.880 and 0.881 respectively). Detection delays have decreased rapidly as well (6.58 and 12.7 seconds respectively), showing the ability of the models to accurately and quickly detect faults. At larger sizes, such as 256, the performance starts to degrade. The recall remains high while the precision drops, suggesting many rapid detections which are not accurate, which in turn explains why the detection time is still low (5.92 seconds).

Considering both the detection accuracy and detection time, a hidden layer size of 64 is selected. While this setting performs marginally worse in accuracy than the option with a size of 128, its detection time is almost half that of the larger model. The reduced complexity, as well as accurate and fast detection, gives this architecture the best balance between representation and computational efficiency.

Window Size and Dropout The window size defines the number of past steps that are provided as input to the model to make a prediction with. This effectively determines the amount of temporal context the model receives. A smaller window leads the model to focus on short-term connections, while longer windows provide more long-term information. A larger window may introduce irrelevant information which makes optimization more difficult, or lead to smoothing of the representation which can reduce the sensitivity to subtle faults.

In Figure 8.12, the relationship between the window size and the performance is shown. For very small window sizes (2 to 10), a moderately accurate model is produced with F_1-scores ranging between 0.7 and 0.8. The detection times are also moderately rapid, between 30 and 50 seconds. The stronger recall compared to the precision suggests the model can detect anomalies, but lacks the context needed to avoid false positives.

At a window size of 25 a large degradation in performance occurs. Both the recall and precision drop, leading to a much lower F_1-score (0.514) and higher detection time (107.2 seconds). This suggests that the model receives more context, but only partially learns the patterns, not enough for stable detection. The strongest performance then occurs at a window size of 50, where both high recall and precision lead to a high F_1-score (0.880). Additionally, the detection time is the lowest of all tested values (6.58 seconds). This indicates a good balance between temporal context and responsiveness. The window size of 75 also performs well in accuracy, but the detection time has increased drastically (104.8 seconds). Further increasing the window size sees the detection time reduce again (31.5 seconds), with similar accuracy (F_1-score$=0.867$). The sharp spike in detection time at a window size of 75 suggests that longer temporal contexts introduce different behaviour in the response of the model. This is likely due to the model becoming biased to long-term behaviour, requiring larger deviations for detection.

Considering both the detection accuracy and detection time, the model with a window size of 50 performs the best. This provides the model with enough temporal context to learn both short- and long-term patterns, making it responsive and accurate.

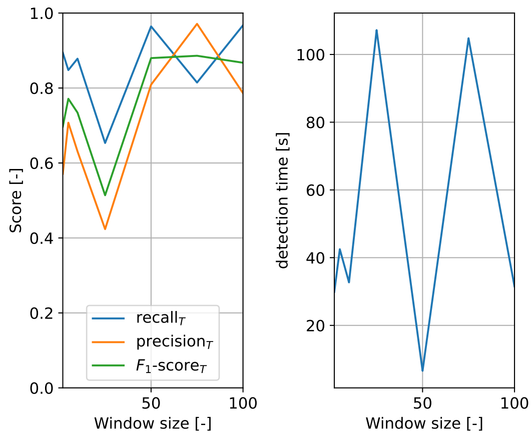


Figure 8.12: Effect of window size on LSTM predictor (hidden layer size: 64, no. of hidden layers: 4, learning rate: 1.92×10^{-3}, dropout: 0%)

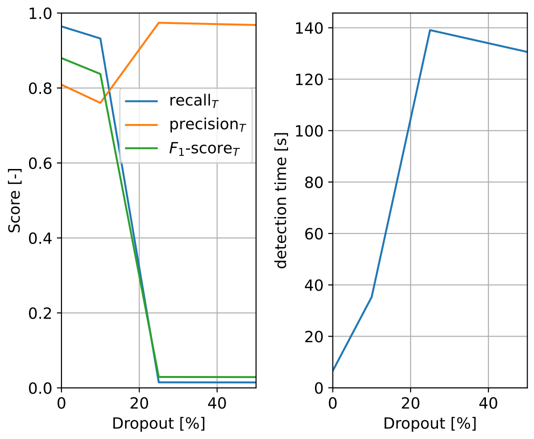


Figure 8.13: Effect of dropout factor on LSTM predictor (window size: 50, hidden layer size: 64, no. of hidden layers: 4, learning rate: 1.92×10^{-3})

Finally, dropout is a technique which can be used to mitigate overfitting by randomly disabling part of the neurons during training. In this case, dropout is applied between the LSTM cells, where it acts as a mechanism to prevent co-adaptation between layers, encouraging the model to be more robust [105]. Excessive dropout can disrupt the propagation of information between cells, which can lead to instability during training.

The effect of the dropout probability on the performance is shown in Figure 8.13. The results show a strong negative impact of dropout on both accuracy and detection time. Without dropout (0%), the model shows the best overall performance. This suggests the model does not suffer from overfitting in this configuration. Introducing increasing probabilities of dropout, the accuracy and detection time degrade. This suggests that even a small amount of dropout disrupts the information passing through the layer, leading to a worse model.

Because the anomaly detection relies heavily on learning patterns from temporal context, the dropout has significant effects on the performance of the model. Given the observed results, a dropout of 0% is selected. The configuration provides the best accuracy and detection time. Any higher probabilities indicate that the connections between cells are affected negatively. The use of dropout is therefore not justified in this case.

LSTM Predictor Architecture

Based on the observed results of the tuning process, an LSTM predictor with the architecture summarized in Table 8.17 is used. The tuning results show that this configuration has the capacity to extract temporal patterns from context, is robust and stable, and provides accurate and timely detection of anomalies.

Table 8.17: Tuned LSTM predictor architecture

Parameter	Value
Loss function	MSLE
Learning rate	1.92×10^{-3}
No. of LSTM cells	4
Hidden layer size	64
Window size	50
Dropout	0%

Besides these tuned parameters, the adaptive moment estimation (Adam) algorithm was used during training to minimize the loss. This optimizer is considered the best optimizer for most scenarios [106]. The batch size, which specifies the amount of data that is used before a weight update, was set to 32. At this value, reasonable training times were achieved with high accuracy.

Part III

Results and Discussion

With the conventional and ML-based fault detection methods defined, their responses to faults were simulated. This part of the report first presents these responses and compares the achieved performances in chapter 9. This is followed by an analysis of the computational impact, comparing the expected load to typical CubeSat-class hardware, in chapter 10.

9 Detection Performance

To compare the performance of the selected conventional and ML-based methods, their responses to the defined faults and scenarios was simulated. One fault case is highlighted in section 9.1 to discuss the behaviour of each method and their effect on the final FKF output. Subsequently, their performance across all scenarios and fault cases is determined and compared in section 9.2.

9.1 | Fault Response

To observe how each method responds to the same fault, one combination of scenario and fault case is highlighted in this section. The case in which the magnetometer has a zero fault in the default scenario is presented, with the response of each of the methods plotted. A small subset of the full dataset is shown for clarity, containing three consecutive fault periods. The true and estimated quaternions and gyroscope biases of the selected subset are shown in Figure 9.1.

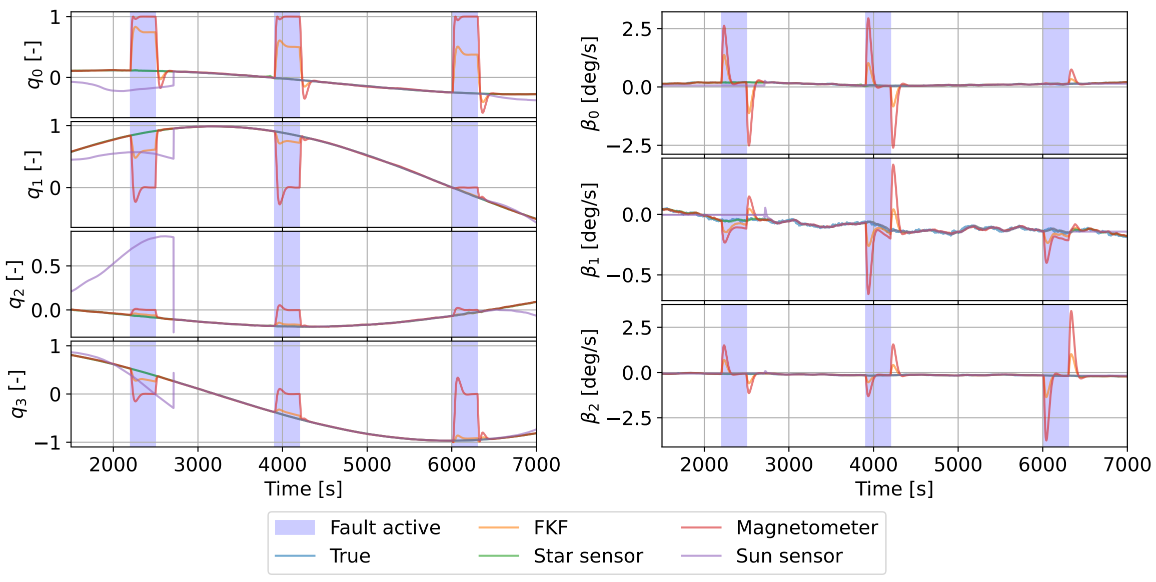


Figure 9.1: True and estimated history of quaternion and bias elements during three repeated faults

In this figure, it is important to note that the plots relating to the three attitude sensors are the output of the local filter assigned to that sensor. For example, the plot 'Magnetometer' is the solution estimated by the local filter assigned to the magnetometer. During a zero fault, the magnetometer returns a zero output. This fault is active in the blue regions. Some additional comments on the behaviour observed in this graph are warranted.

The quaternion elements are shown to smoothly vary over time with two exceptions. First is the Sun sensor filter, which reports deviating estimates at the start and near the end of the graph. This is due to the spacecraft being in eclipse during this time, leading to no Sun sensor measurements being available. The corresponding local filter must rely solely on the gyroscope data for which the estimated bias can no longer be updated, causing the propagated state to slowly diverge over time. This does not significantly affect the FKF output, however. The covariance of the Sun sensor filter estimate greatly increases during this period, leading the FKF to largely ignore this output according to Equation 4.2b.

The second, more relevant exception is the magnetometer filter, which expectedly deviates during the fault periods. During the fault, the magnetometer measurements suddenly deviate greatly from the true attitude. The supposed rapid movement that this deviation suggests is absorbed in the filter by increasing the estimated gyroscope bias. The propagated state, which relies on this bias, is used to predict the next measurement. By adjusting the bias, this prediction is then able to more closely follow the anomalous measurements, which the filter deems optimal.

Since the filter is still processing measurements and the predicted measurement is reconciled with the anomalous data, the covariance does not increase as much as is the case for the Sun sensor filter during eclipses. This leads the FKF to still consider the magnetometer filter output as trustworthy, causing the FKF estimate to also deviate from the true quaternion value. This deviation is exaggerated for the first fault instance, where the Sun sensor is not functioning. Only the magnetometer and star tracker are available to the FKF here, whereas for the two later instances the Sun sensor is also available to counteract the magnetometer deviation.

This effect where the FKF falsely attributes the deviating measurements to a large change in gyroscope bias can be lessened by adjusting the parameters of the filter, such as the process or measurement noise. This would make it more difficult for the filter to adjust the gyroscope bias when anomalous measurements are processed. However, this only slows the false attribution effect and does not completely remove it. Additionally, large changes in these parameters can make the filter unstable.

In the following sections the responses of the conventional and ML-based methods to this subset of faults are discussed. In the final section, the effect that the application of FD has on the FKF output is shown.

9.1.1 | Conventional Methods

The two conventional methods, relying on the measurement residual and sensitivity factor, were simulated for the above dataset containing three instances of the magnetometer fault. Their responses are discussed in this section.

Measurement Residual

First, the response of the measurement residual ratio to the faults is shown in Figure 9.2. During the nominal periods, the value is stable below a value of 1, indicating the measurement residual deviates by at most 1σ. However, large spikes in the ratio are observed both after the start of a fault and after the end of a fault.

The spikes correspond to those observed in the estimated gyroscope bias in Figure 9.1. At the start of a fault, the measurement suddenly deviates from the prediction, causing a large residual which trips the detection. However, as the filter slowly adjusts the gyroscope bias to compensate, they are brought more in line causing the residual to shrink over time. The ratio returns to below 3σ, where the detection method falsely suggests the data is no longer anomalous.

At the end of the fault, a similar effect happens. A nominal measurement becomes available again, suddenly deviating from the anomalous data on which the filter had settled. This causes a large residual, which is then slowly reconciled by returning the gyroscope bias to nominal levels.

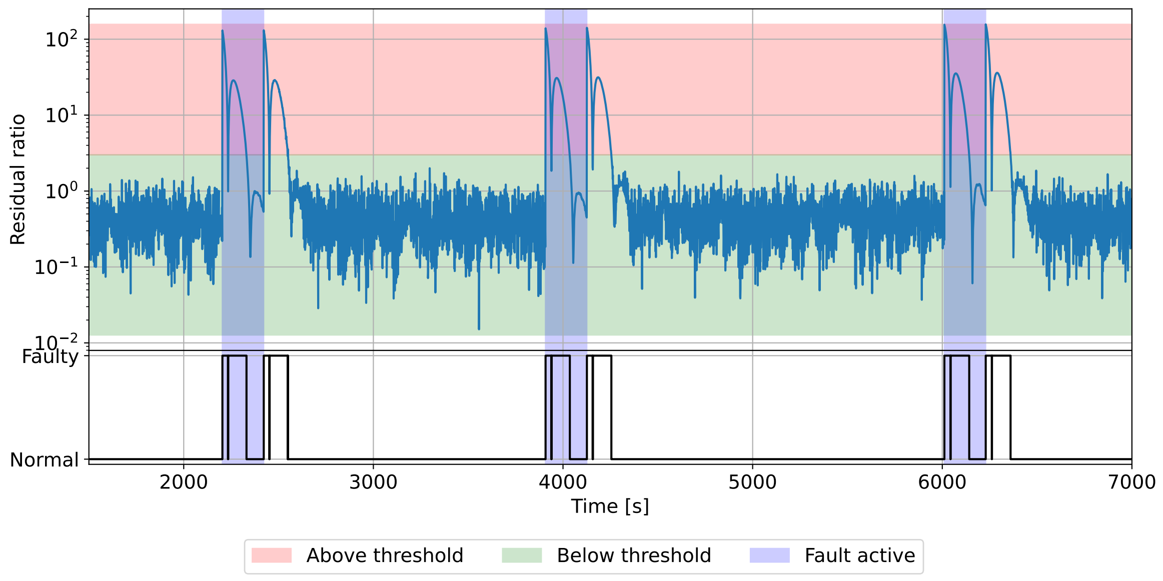


Figure 9.2: Response of measurement residual to three magnetometer zero faults

The result of this effect is that this method is not able to capture the entirety of the fault, only responding to the start and end of the fault when the sudden change has not yet been compensated for. By adjusting filter settings this effect can be slowed, but it would still appear for long-duration faults. The method does not incorporate any information besides the difference between the prediction and measurement, and is therefore sensitive to modelling mismatches in the filter itself. While the effect can be reduced, it requires more tuning and cannot be fully eliminated.

Sensitivity Factor

Next, the response of the sensitivity factor to the same faults is shown in Figure 9.3. Here, it is observed that the calculated factor during nominal periods sways in a larger range, between roughly 0.1 and 20, and in a more irregular pattern than the measurement residual ratio. The sensitivity factor includes information from the master filter state and covariance, which are affected by the other local filters. This leads to a more complex response than when only a single metric from one local filter is considered, such as the measurement residual.

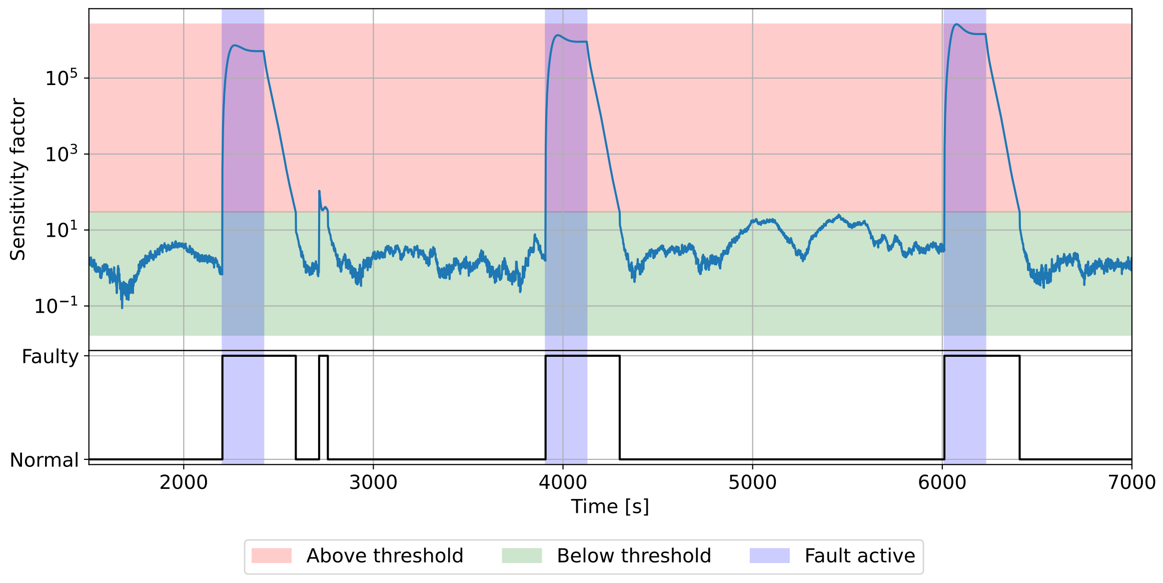


Figure 9.3: Response of sensitivity factor to three magnetometer zero faults

Once a fault becomes active, the factor rapidly increases by several orders of magnitude. It then remains stable and nearly constant for the remainder of the fault. Once the fault ends, the measure gradually decreases, returning to nominal levels over a period of roughly 150 seconds. A false positive is present after the first fault instance, where the factor rises above the threshold for roughly 50 seconds.

The sensitivity factor relies on the difference between the local filter and the master filter. When the filter assigned to the magnetometer starts to deviate, the remaining filters do not. This causes the state of the magnetometer to deviate from the master state, which is compared to the expected uncertainty to decide when it has become anomalous. Even when the magnetometer filter has compensated for the deviation in the gyroscope bias, the difference to the master filter remains. This information allows this detection method to more reliably capture the entirety of the fault.

Of the conventional methods, detection based on the sensitivity factor is found to be more reliable than the measurement residual alternative. The residual-based approach is sensitive to modelling mismatches, only responding to the starts and ends of a fault. The sensitivity factor, on the other hand, incorporates the master filter state for more consistent detection.

9.1.2 | ML-based Methods

Following the conventional methods, the ML-based methods were also simulated for the same three fault instances. The responses of the iForest and LSTM predictor are discussed in this section.

Isolation Forest

The response of the iForest to the faults is shown in Figure 9.4. During normal operation, the anomaly score hovers at roughly 0.45. When the fault is active, this rapidly increases to a value of approximately 0.65. Once the fault ends, the score remains high for some time before decreasing back to its original level.

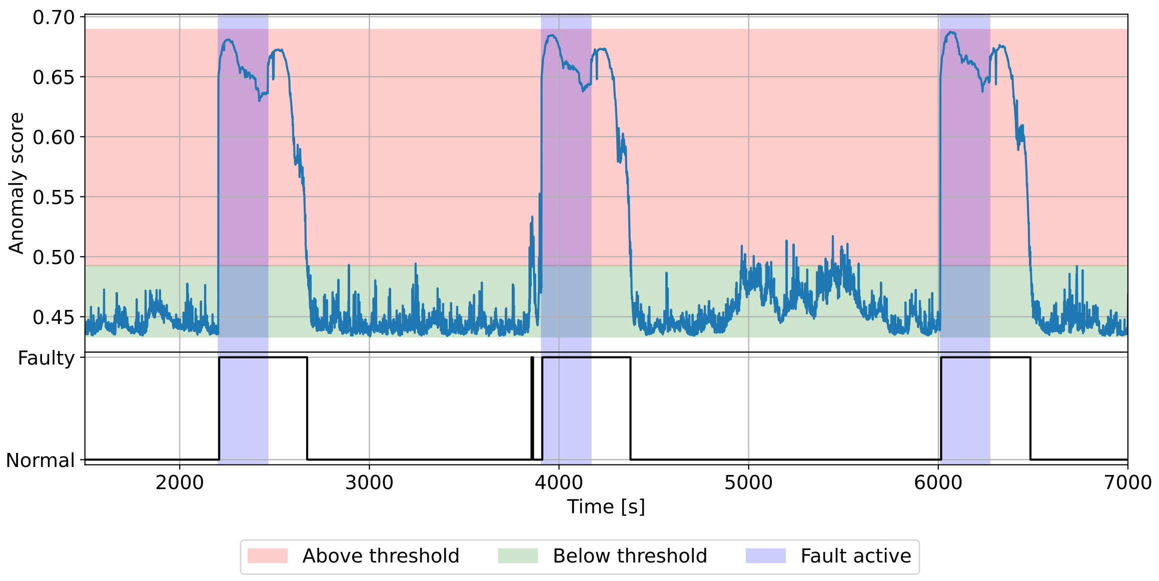


Figure 9.4: Response of iForest to three magnetometer zero faults

The response of the iForest shows similarity to that of the sensitivity factor. However, there is a brief false positive before the second fault instance for the iForest, not corresponding to the one observed for the sensitivity factor. Additionally, between the second and third fault instance, the score rises closer to the detection threshold. Several outliers even cross the threshold, but they do not trip the detection due to the post-processing step requiring three consecutive detections, which is applied to the output of both the conventional and ML-based methods.

While the responses of the iForest and sensitivity factor are similar, the iForest takes more time to settle back to nominal levels after the fault has ended, roughly 200 seconds instead of 150 seconds. The sensitivity factor directly measures the statistical consistency between the local and master filter states using their covariances. It reacts immediately as the filters are converging after the fault ends. The iForest relies on patterns learned from past data, taking longer to classify the transient states as normal.

LSTM Predictor

Finally, the response of the LSTM predictor to the faults is depicted in Figure 9.5. Similar to the sensitivity factor and iForest, once a fault starts the anomaly score quickly spikes to a large value. However, instead of remaining at this high value, the score settles at a lower constant value, though still above the detection threshold. At the end of a fault, another spike is observed before settling to original levels.

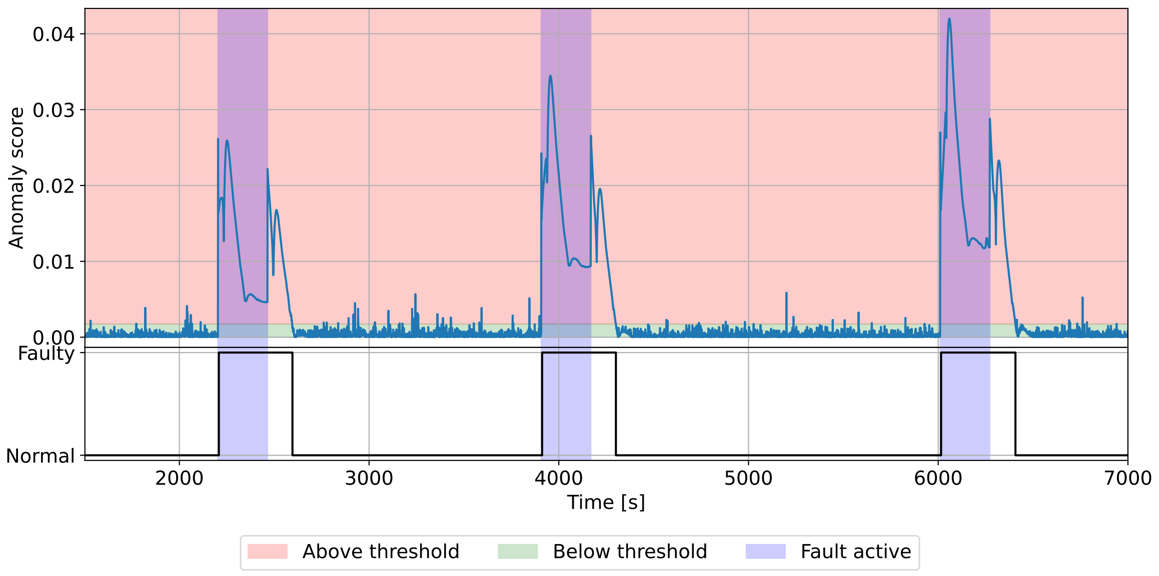


Figure 9.5: Response of LSTM predictor to three magnetometer zero faults

This behaviour is explained by the selected window size of 50 samples. When this window contains the transition from normal to faulty operation, or the opposite, a high anomaly score is found. The combination of anomalous and nominal data points make prediction of the next sample difficult. However, when the window only contains faulty samples, the next sample is more easily predicted, but it does not adhere to the nominal pattern. This leads to a lower anomaly score than during the transitions, though it is still larger than during normal operation. Additionally, one of the underlying features includes the bias difference between the local and master filter, which also shows similar spikes directly after the transition between states, as is seen in Figure 9.1. This compounds the effect.

Both ML-based methods show the ability to detect the faults correctly. The calculated anomaly scores rise upon the start of a fault, and remain above the detection threshold for the duration of the fault. After the fault has passed, the calculated score settles to nominal levels again.

9.1.3 | Fusion Result

The FD methods are used to exclude the output of the local filter assigned to the magnetometer from the FKF fusion when the detection is tripped. In Figure 9.6, the same dataset as was used in the previous analysis is depicted. However, now the final fused outputs of the FKF are shown for a system with no FD and one with FD provided by the LSTM predictor.

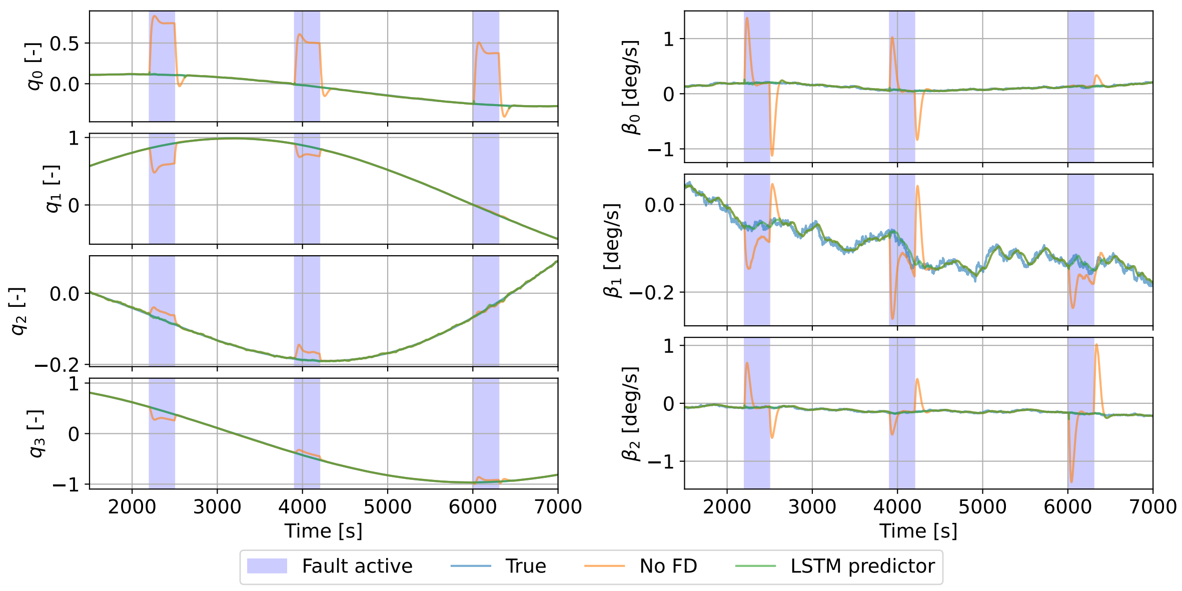


Figure 9.6: Attitude estimation with and without FD

When no FD is applied, the output is significantly affected during the fault periods, with the same original behaviour as shown in Figure 9.1. However, when a FD method is implemented, in this example the LSTM predictor, the final output remains much closer to the true state. Both the quaternion and bias estimates remain unaffected, showing the addition of FD in the FKF effectively isolated the sensor fault. This difference between the fusion system with and without FD is quantified in Table 9.1.

Table 9.1: Fusion output error with and without FD

Method	**Quaternion error** cumulative [-]	**Bias error** cumulative [deg/s]
No fault (baseline)	16.3	1.5
No FD	614.6	9.3
Sensitivity factor	16.4	1.5
Measurement residual	335.5	2.0
iForest	16.6	1.6
LSTM predictor	16.7	1.5

The cumulative error between the true and fused output is calculated for each FD method, for both the quaternion and bias estimates. It is observed that both errors are greatly reduced when FD is applied with results comparable to the baseline case in which no faults are present. All methods, except for the measurement residual, show nearly identical results for this fault case.

To minimize the cumulative error, faults should be identified as quickly as possible. When faulty data is included in the FKF fusion, the estimated attitude deviates from the true attitude of the spacecraft. In the next section, the accuracy and response time of the detection methods is therefore compared.

9.2 | Method Comparison

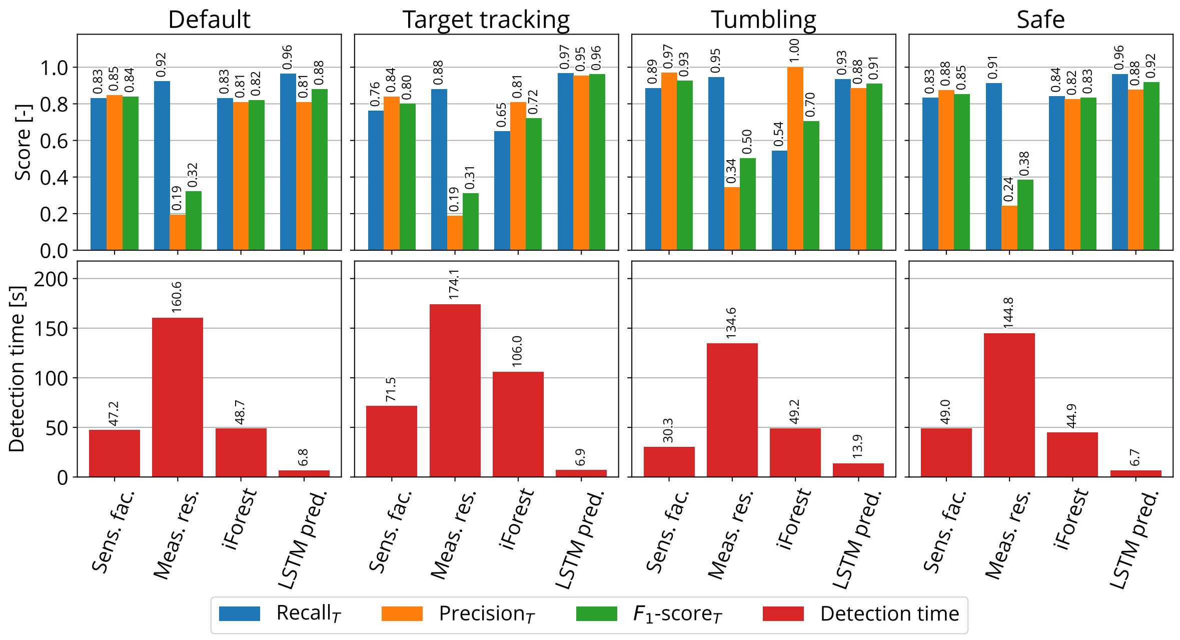
To compare the overall performance of the conventional and ML-based methods, all scenarios and fault cases described in section 6.4 are considered. Each detection method was applied to each combination of scenario and fault case, leading to a large set of performance metrics. In this section, the results are compared per fault case first, followed by an aggregate comparison.

The performance metrics as defined in chapter 5 are used for this comparison: recall, precision, F_1-score, and detection time, as well as the ROC curves.

9.2.1 | Star Tracker Stuck Fault

The first presented fault case is the stuck fault occurring in the star tracker. The performance of the detection methods in this case is shown in Figure 9.7 for each method and STK scenario.

The measurement residual performs poorly across all scenarios. The F_1-score is low (0.31-0.50) and the detection time is large (138.7-162.8 seconds). As was discussed in the previous section, the filter adapts to the faulty measurements by adjusting the gyroscope bias, keeping the predicted measurement in line with the anomalous measurement. Since the stuck fault only causes a gradual deviation from the true value over time, the filter can keep the measurement residual small. The threshold is not crossed until the deviation outgrows the ability to adjust the gyroscope bias, producing a long detection time and correspondingly low detection accuracy.



Scenario	Method	Recall$_T$	Precision$_T$	F_1-score$_T$	Detection time [s]
Default	Sens. fac.	0.83	0.85	0.84	47.2
Default	Meas. res.	0.92	0.19	0.32	160.6
Default	iForest	0.83	0.81	0.82	48.7
Default	LSTM pred.	0.96	0.81	0.88	6.8
Target tracking	Sens. fac.	0.76	0.84	0.80	71.5
Target tracking	Meas. res.	0.88	0.19	0.31	174.1
Target tracking	iForest	0.65	0.81	0.72	106.0
Target tracking	LSTM pred.	0.97	0.95	0.96	6.9
Tumbling	Sens. fac.	0.89	0.97	0.93	30.3
Tumbling	Meas. res.	0.95	0.34	0.50	134.6
Tumbling	iForest	0.54	1.00	0.70	49.2
Tumbling	LSTM pred.	0.93	0.88	0.91	13.9
Safe	Sens. fac.	0.83	0.88	0.85	49.0
Safe	Meas. res.	0.91	0.24	0.38	144.8
Safe	iForest	0.84	0.82	0.83	44.9
Safe	LSTM pred.	0.96	0.88	0.92	6.7

Figure 9.7: Performance across scenarios for star tracker stuck fault

The sensitivity factor shows more consistent results, with F_1-scores between 0.80 and 0.93 across scenarios. The detection time is improved significantly compared to the measurement residual, between 30.3 and 71.5 seconds.

The LSTM predictor shows comparable but slightly better metrics in terms of accuracy, outperforming the sensitivity factor in all scenarios except the tumbling scenario (F_1-score between 0.88 and 0.96). The largest improvement is in the detection time, which is drastically lowered to between 6.7 and 13.9 seconds. As was expected, the sequence-based detection method is able to more rapidly detect the slow deviation of the sensor, whereas the point-wise sensitivity factor requires the difference to grow more first.

The iForest shows slightly worse accuracy when compared to the sensitivity factor, with F_1-scores between 0.70 and 0.83. Especially in the target tracking and tumbling scenarios, the sensitivity factor outperforms the iForest. A similar result is observed for the detection time, which is roughly equal to the sensitivity factor in the default and safe scenarios, but significantly longer in the target tracking and tumbling scenarios.

Besides these performance metrics, the ROC curves were calculated for each method, which is shown in Figure 9.8. Note that these rely on the traditional counts of true/false positives/negatives, instead of the range-based metrics depicted in Figure 9.7, as described in chapter 5.

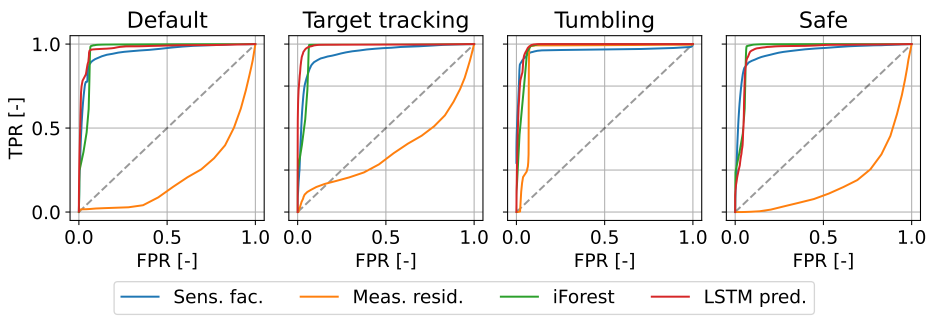


Figure 9.8: ROC curves across scenarios for star tracker stuck fault

All methods except the measurement residual show consistently good curves tending towards high TPR at low FPR, with the iForest and LSTM predictor performing slightly better than the sensitivity factor in general. This suggests they are more threshold-agnostic. The measurement residual performs poorly, often worse than the random classifier.

9.2.2 | Magnetometer Zero Fault

Continuing with the second fault case in which the magnetometer output goes to zero, the results as shown in Figure 9.9 are obtained.

Again, the measurement residual performs poorly (F_1-scores between 0.38 and 0.51), though the detection time has improved. The more abrupt and large deviation helps the method to more quickly and accurately detect the anomaly. As was shown in Figure 9.2, the detection spikes at the start and end of a fault, but fails to capture its entire duration, leading to low detection accuracy.

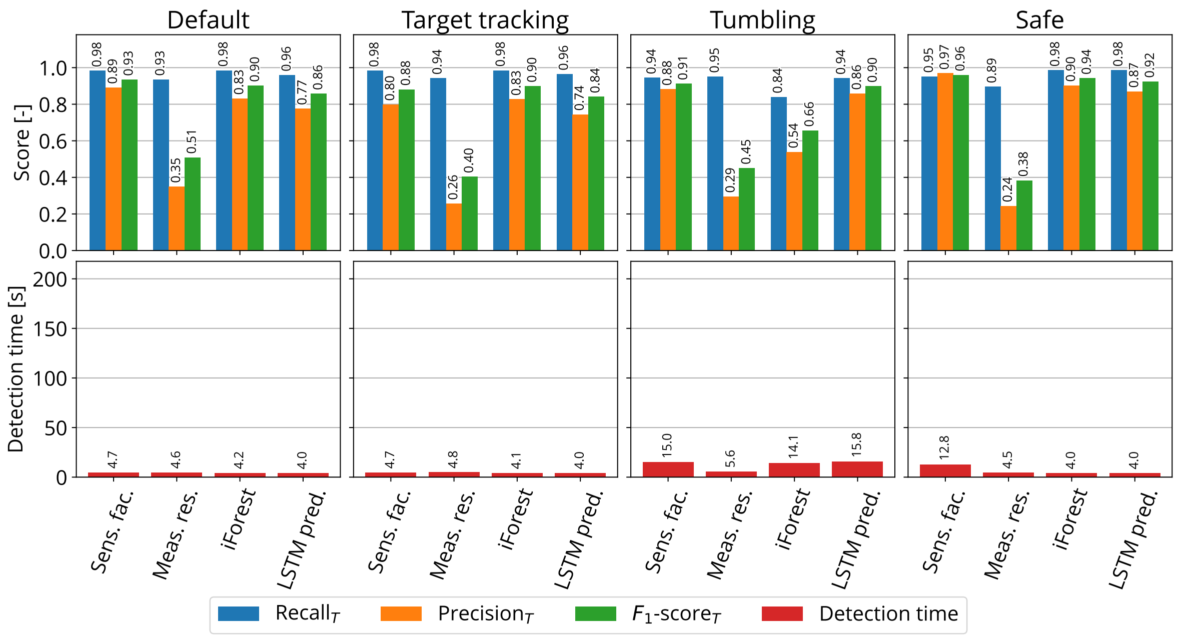


Figure 9.9: Performance across scenarios for magnetometer zero fault

The sensitivity factor shows greater accuracy (F_1-scores between 0.88 and 0.96), as well as fast detection times (between 4.7 and 15.0 seconds). Again, the abrupt nature of the fault leads to a more obvious deviation that is picked up by the methods. This time, the LSTM predictor shows accuracy slightly below that of the sensitivity factor, with F_1-scores varying between 0.84 and 0.92. The detection time, however, is even faster than in the previous fault case, showing 4.0 seconds for all cases except the tumbling case where it is 15.8 seconds. It is important to consider the addition of the post-processing step which requires three consecutive detections before tripping. At the simulation rate of 1 Hz, this means the anomaly score has already risen above the threshold after the first anomalous data point.

The iForest shows accuracy comparable to the sensitivity factor with F_1-scores in the range 0.90-0.94, except in the tumbling scenario where it drops to 0.66. The detection time has greatly improved compared to the previous fault case and is en par with the LSTM predictor, between 4.0 and 14.1 seconds.

Again, the ROC curves for each method were plotted, shown in Figure 9.10. Similar to the star tracker stuck fault, the sensitivity factor, iForest, and LSTM predictor perform well. In contrast to the previous fault case, the measurement residual now performs considerably better, though still not at the level of the other methods. Again, the more abrupt and obvious nature of the fault makes for easier detection compared to the more gradual stuck fault.

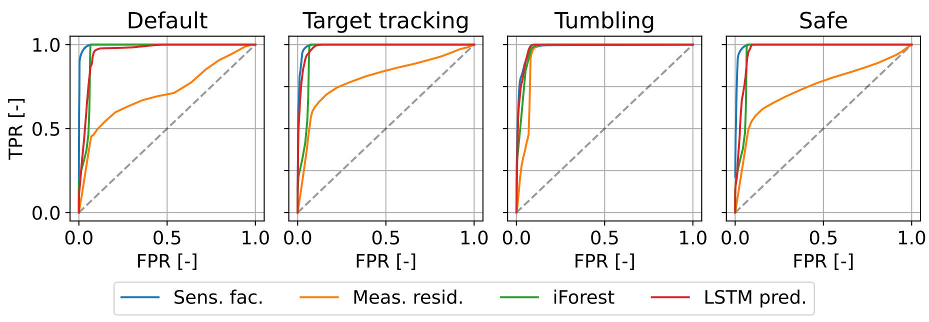


Figure 9.10: ROC curves across scenarios for magnetometer zero fault

9.2.3 | Sun Sensor Axis Fault

The third and final fault case, where one of the three axes of the Sun sensor fails, produces the results shown in Figure 9.11. First, it is noted that all methods perform considerably worse in terms of accuracy than in the other fault cases.

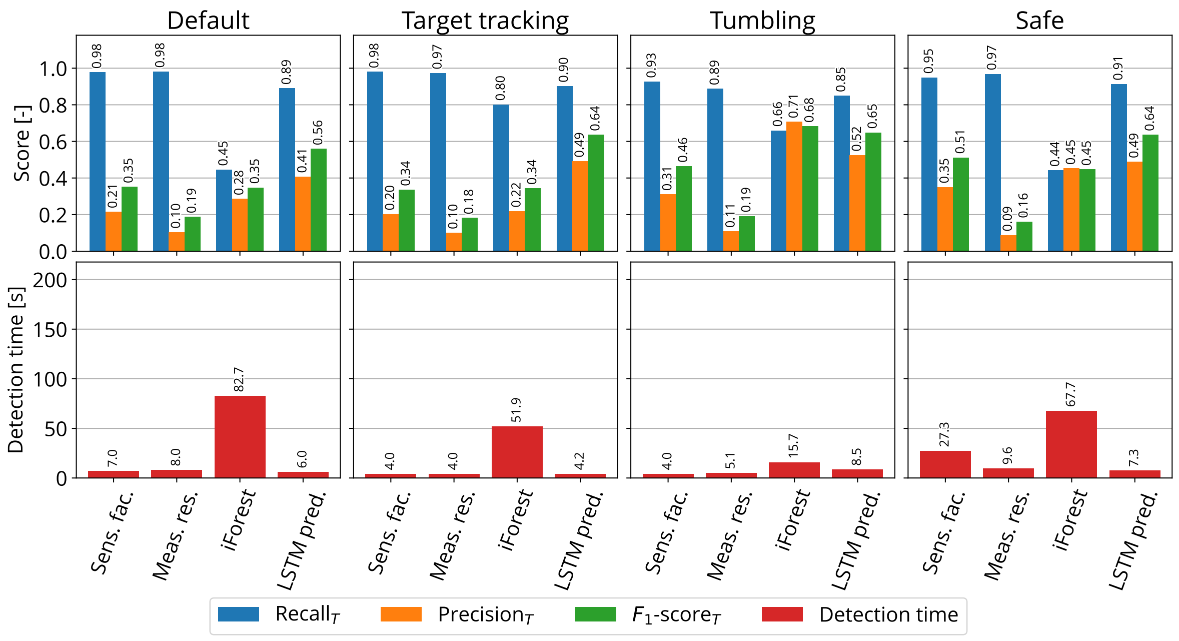


Figure 9.11: Performance across scenarios for Sun sensor axis fault

There are several causes for this decrease in accuracy. First, the attitude based on the faulty Sun sensor measurement still closely resembles the true attitude in many orientations. Except for specific attitudes, the estimate does not diverge significantly, making fault detection difficult. Additionally, eclipse periods impact the available instances of faults. The eclipse periods are left out of consideration due to the inability for faults to be observed within them. This reduces the available number of fault instances for each dataset.

Again, the measurement residual performs the worst, with very low F_1-scores between 0.16 and 0.19. The reported detection time is low, between 4.0 and 9.6 seconds. However, this can be explained by observing the high recall score but low precision score. The method is triggering repeatedly in short spikes, meaning a detection is often present right after a fault starts. This is not due to accurate detection of the faults, but rather random chance.

The sensitivity factor shows significantly worse results compared to the previous fault cases. The accuracy is much lower, with F_1-scores dropping to between 0.34 and 0.51. Similar to the measurement residual, a high recall but low precision score is observed in all scenarios. The reported detection times, between 4.0 and 27.3 seconds, suffer from the same issue as those of the measurement residual. Due to the low detection accuracy, the low detection time is not trustworthy. The iForest shows comparable results with F_1-scores between 0.34 and 0.45, except in the tumbling case, where the iForest is the best-performing method with an F_1-score of 0.68. The detection time in this case is 15.7 seconds.

Finally, the LSTM predictor performs the best across the scenarios on average. The reported accuracy and detection times are much more consistent, with F_1-scores between 0.56 and 0.65, and detection times between 4.2 and 8.5 seconds. This method is the only method that produces an F_1-score over 0.5 across all scenarios, making it the only method that can detect the Sun sensor fault in daylight with moderate accuracy.

9.2.4 | **Overall Comparison**

The achieved scores and detection times of each method are collected visually in Figure 9.12. The means and standard deviations across scenarios and fault cases were calculated and are tabulated in Table 9.2. Note that the reported metrics reflect the variation between the scenarios and fault cases, not statistical uncertainty.

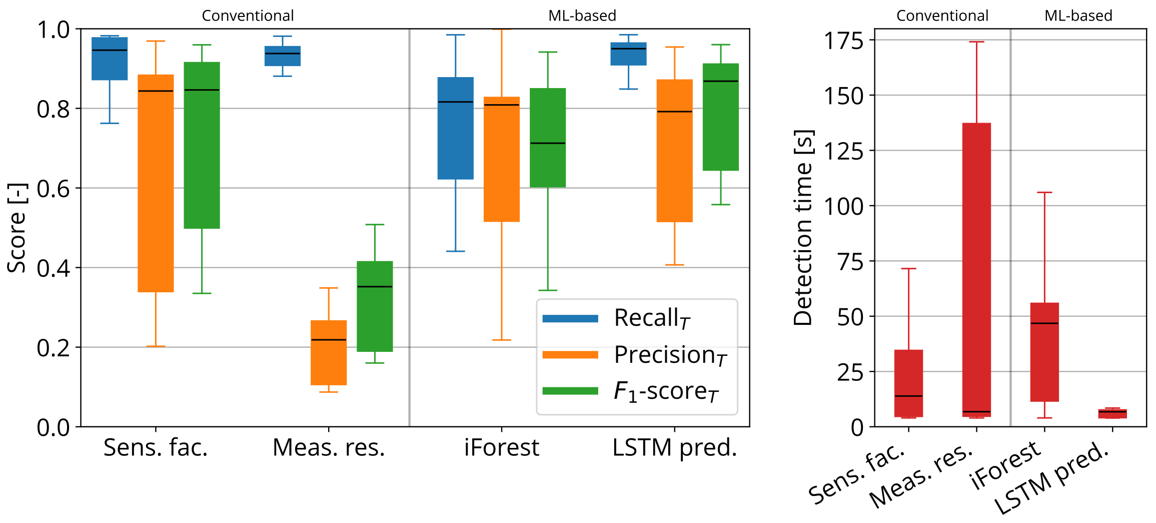


Figure 9.12: Comparison of scores and detection times across scenarios and fault cases

Table 9.2: Average performance across scenarios and fault cases

Method	Recall$_T$ mean (SD) [-]	Precision$_T$ mean (SD) [-]	F_1-score$_T$ mean (SD) [-]	Detection time mean (SD) [s]	ROC-AUC mean (SD) [-]
Sensitivity factor	0.9168 (0.07)	0.6787 (0.30)	0.7303 (0.23)	23.1 (21.4)	0.8073 (0.20)
Measurement residual	0.9326 (0.03)	0.2090 (0.09)	0.3316 (0.12)	55.0 (70.2)	0.5580 (0.23)
iForest	0.7503 (0.19)	0.6832 (0.24)	0.6910 (0.20)	41.1 (32.1)	0.7716 (0.09)
LSTM predictor	0.9356 (0.04)	0.7233 (0.18)	0.8051 (0.14)	7.34 (3.69)	0.8778 (0.14)

As was observed in Figure 9.7, 9.9, and 9.11, the measurement residual has the lowest average accuracy (F_1-score of 0.33) and longest average detection time (55.0 seconds). The poor accuracy, combined with the high detection time, makes this an unreliable detection method. This is reflected in Figure 9.12, where the method shows consistently high recall but low precision. Additionally, the detection time is low in some scenarios, but very high in others. The low detection times are mostly not trustworthy due to the low detection accuracy. The approach taken when using the measurement residual is the only one that does not incorporate the difference between the local and master filter, basing the anomaly score only on the variables in the local filter. This lack of information, combined with the inability to detect gradual faults well, leads to this being the worst-performing method.

The other conventional method, the sensitivity factor, performs much better. With an average F_1-score of 0.73, this method yields moderately accurate detection. Similarly, a moderate average detection time of 23.1 seconds is found. The inclusion of the difference between the local and master filter, as well as the incorporation of the state covariances, provides this method with enough information to detect anomalies with higher accuracy and speed. However, the method shows significant variance between scenarios and fault case. This is observed in Figure 9.12, where the IQR of the F_1-score varies between 0.50 and 0.91 across scenarios and fault cases. The performance of the method thus depends highly on the spacecraft and anomaly behaviour.

The iForest shows a similar but lower average accuracy with an F_1-score of 0.69. The accompanying average detection time is much greater when compared to the sensitivity factor, at 41.1 seconds. This method, similar to the sensitivity factor, is a point-wise method, not considering the temporal history. The iForest relies on similar underlying information as the sensitivity factor, though encoded as normalized features. This makes it difficult to surpass its performance without additional context. In contrast to the sensitivity factor, however, the iForest shows more consistent results for varying cases with an IQR for the F_1-score of 0.61 to 0.85.

The second ML-based detection method, the LSTM predictor, yields the best results, both in terms of average accuracy (F_1-score of 0.81) and detection time (7.34 s). Additionally, the variance between scenarios and fault cases is significantly lower, indicating more stable behaviour across these situations. The IQR for the F_1-score observed in Figure 9.12 is between 0.64 and 0.91. The sequence-based detection allows the method to learn temporal patterns, both long- and short-term. This leads to more rapid detection across all scenarios, especially with gradual faults.

Since the sensitivity factor is a more reliable conventional baseline, the ML-based methods are compared to this method. The percentual difference between them and the sensitivity factor are shown in Table 9.3.

Table 9.3: Performance of ML-based methods compared to sensitivity factor

Method	Recall$_T$	Precision$_T$	F_1-score$_T$	Detection time	ROC-AUC
iForest	-18.2%	$+0.7\%$	-5.4%	$+77.7\%$	-4.4%
LSTM predictor	$+2.1\%$	$+6.6\%$	$+10.2\%$	-68.3%	$+8.7\%$

The iForest does not improve the detection performance of the sensitivity factor. While the overall accuracy is comparable, only 5.4% lower than that of the sensitivity factor, the detection time is drastically longer, increasing by 77.7%.

The LSTM predictor, on the other hand, improves upon all performance metrics of the sensitivity factor. The F_1-score is increased by over 10%. More significant is the reduction in detection time, which is reduced by over a factor of three when using the LSTM predictor instead of the conventional method. This improvement is especially significant and can clearly be observed in the comparison of Figure 9.12. The LSTM predictor method never exceeds a detection time of 15.8 seconds across all simulated scenarios and fault cases.

To summarize, the measurement residual proves unreliable due to its sensitivity to modelling mismatches in the local filter. The sensitivity factor is the more reliable conventional method which incorporates information from the master filter to produce moderate detection performance. The iForest shows less dependence on the fault case and scenario, but has worse performance overall, especially in detection time. Finally, the LSTM predictor is the best-performing method, increasing the detection accuracy and especially improving the detection time.

10 Computational Impact

To determine the computational impact of the various anomaly detection methods, a static analysis was performed to characterize their efficiency. While the methods discussed in this report are implemented as Python code, these are not representative of optimized or embedded solutions. Therefore, measuring the execution time and memory usage of the Python code is not sufficient to compare the detection methods.

Counting the number of elementary operations, such as floating-point operations (FLOPs), and estimating memory usage are widely used to compare algorithms independent of specific implementations [107]. This type of analysis provides a hardware-independent baseline for estimating the computational cost. Due to behaviour that is difficult to predict, such as memory access and other hardware effects, the result does not always correlate perfectly with runtime [107]. However, it is a well-recognized method that allows for a general comparison of the methods despite these limitations.

The following sections aim to estimate the computational cost and the size of memory used for inference at one timestep for each of the methods. First are the conventional methods: the sensitivity factor (section 10.1) and measurement residual (section 10.2). They are followed by the ML-based methods: the iForest (section 10.3) and LSTM predictor (section 10.4). The results of the methods are compared in section 10.5, accompanied by an evaluation of their potential use on CubeSat-class hardware. Training costs are not included in this analysis, since it is presumed the model is trained on the ground and only inferred from on-board.

The computational cost is determined using two classes of operations. First, the number of FLOPs required for one inference step is counted. This is defined as an arithmetic operation on a floating-point number, including multiplication, addition, subtraction, and division. For simplicity, computations such as square root or exponentiation are also counted as one FLOP. The FLOP count indicates the volume of computation that is required. The second class of operations, supplementing the FLOP count, is the number of comparisons (COMPs) that are performed. These are conditional evaluations, such as determining which of two values is larger or smaller. Comparisons are particularly important in tree-based models, such as the iForest. Comparisons are typically executed by different units of a processor, and captures a second source of computational effort. The comparison of the calculated score and the detection threshold is not included in this count since this is present for all methods while being insignificant compared to the actual calculation.

The memory usage of the methods is also divided into two classes. First, the static memory is defined as the memory that is permanently required to store a method's parameters or other model attributes, such as the weights and biases of the LSTM network. This represents the inherent footprint of a model, independent of how many samples are being processed. The static memory also places a constraint on the hardware it is run on, determining whether the model can fit on a device at all. The second class of memory is the runtime memory. This consists of all the temporary variables, intermediate values, or other memory that is created or needed during inference. The runtime memory describes the size of the intermediate computations and indicates the transient demands of the method. A 32-bit system is assumed, since this is the predominant architecture used in typical OBCs, as was identified in section 2.3.

A common operation in the following sections is the matrix multiplication. The number of FLOPs required for the multiplication of two matrices of sizes $n \times p$ and $p \times m$ is [108]:

$$\text{FLOP}_{\text{matmul}}(n, p, m) = nm(2p - 1) \tag{10.1}$$

10.1 | Sensitivity Factor

The number of FLOPs needed to calculate the sensitivity factor is found by examining Equation 7.1. First, the states of the local and master filter, $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{x}}_F$ respectively, are subtracted (element-wise). The size of these state vectors is denoted as n. Thus, n FLOPs are needed for the subtraction. It is assumed this result is stored for both uses in the equation.

Next, the covariance matrices of the states, P_i and P_F are added (element-wise). Since these matrices are both $n \times n$, this requires n^2 FLOPs. The resulting matrix is inverted. This operation, when using Cholesky decomposition, requires $n^3 + n^2 + n$ FLOPs, including square roots as one FLOP [108].

Now, the result of the vector subtraction and matrix inverse are multiplied. Following Equation 10.1, this requires $n(2n - 1) = 2n^2 - n$ FLOPs. The result of this computation is multiplied with the subtracted vectors again, adding an additional $2n - 1$ FLOPs.

The sensitivity factor does not rely on any comparisons, thus uses 0 COMPs. The total number of FLOPs required for the calculation of the sensitivity factor at one time step is then the summation of the previous parts:

$$\text{FLOP}_{\text{SF}} = n^3 + 4n^2 + 3n - 1 \tag{10.2}$$

$$\text{COMP}_{\text{SF}} = 0 \tag{10.3}$$

Considering the states in this case consist of $n = 6$ elements, the calculation of the sensitivity factor requires 377 FLOPs.

To estimate the runtime memory usage of this calculation, it is assumed that the result of the vector subtraction ($1 \times n$), matrix addition ($n \times n$), and intermediate matrix multiplication result ($1 \times n$) are stored. This is a total of $n^2 + 2n$ elements. The sensitivity factor does not require permanent storage of any parameters, and thus has a static memory usage of 0 bytes. Assuming 32-bit floating-point values, each element requires 4 bytes:

$$\text{MEM}_{\text{SF}}^{\text{runtime}} = 4(n^2 + 2n) \tag{10.4}$$

$$\text{MEM}_{\text{SF}}^{\text{static}} = 0 \tag{10.5}$$

Again, using $n = 6$, this results in an estimated runtime memory usage of 192 bytes.

10.2 | Measurement Residual

The number of FLOPs required for calculating the measurement residual value similarly follows from Equation 7.3. It is assumed the residual $\boldsymbol{v}$ is already calculated within the filter, and is therefore not included in the FLOP count of this method.

First, the square root of the diagonal of the innovation covariance P_v is calculated. Here, the size of the residual vector, and thus the dimension of the covariance matrix, is denoted as n. Counting one square root operation as one FLOP [108], this requires n FLOPs. Next, the residual $\boldsymbol{v}$ is divided by the square root of the diagonal (element-wise), adding n FLOPs. Finally, the norm of this vector is calculated. This requires $2n$ FLOPs.

The measurement residual, like the sensitivity factor, does not require any comparisons, thus uses 0 COMPs. Summing the previous contributions, the number of FLOPs required to calculate the measurement residual ratio is found, leading to:

$$\text{FLOP}_{\text{resid}} = 4n \tag{10.6}$$
$$\text{COMP}_{\text{resid}} = 0 \tag{10.7}$$

Considering a quaternion measurement of $n = 4$ elements, this results in 16 FLOPs.

Assuming the calculated standard deviation vector, as well as the result of the subsequent vector division, are stored between operations, $2n$ elements are placed in runtime memory. Similar to the sensitivity factor, no parameters need to be stored permanently, thus the static memory usage is 0 bytes. Again assuming 32-bit floating-point numbers, this leads to:

$$\text{MEM}_{\text{resid}}^{\text{runtime}} = 8n \tag{10.8}$$
$$\text{MEM}_{\text{resid}}^{\text{static}} = 0 \tag{10.9}$$

Again, using $n = 4$, an estimated runtime memory usage of 32 bytes is found.

10.3 | **Isolation Forest**

The resource usage for the iForest differs from the conventional methods, and in fact most ML-based methods. Instead of performing large computations, inference of an iForest relies mostly on comparisons between the sample and pre-defined thresholds.

The iForest consists of T iTrees, each of which contains ψ samples. During inference for some sample s, the path length h through the i-th iTree is determined, as $h_i(s)$. Then, the average path length over all iTrees, $E(h(s))$, is used to calculate an anomaly score. Assuming the iTree is balanced, the depth of the tree is $\lceil \log_2(\psi) \rceil$ [40]. Following from the maximum percentage of samples used by one tree, derived in section 8.3, $\psi = 60\%\ N$, where N is the total number of samples in the training dataset.

At each internal node of the iTree, the sample is compared to the threshold, and a child node is selected. For simplicity, this is assumed to count as 1 comparison. Using the assumption that the tree is balanced, at most $\lceil \log_2(\psi) \rceil - 1$ comparisons are performed in each iTree in the worst-case scenario. The cost of the entire forest is then the cost of one iTree multiplied by the number of iTrees T.

Once all trees have been traversed, the average path length $E(h(s))$ across all iTrees is calculated. This requires $T - 1$ additions and one division for a total of T FLOPs. The anomaly score is then calculated as [40]:

$$2^{-\frac{E(h(s))}{c(n)}}$$

Here, $c(n)$ is a constant determined from training. This calculation thus requires one division (1 FLOP), one negation (1 FLOP), and one exponentiation (1 FLOP) for a total of 3 FLOPs. While exponentiation is typically more expensive than other arithmetic operations, it is counted as 1 FLOP for simplicity.

The total number of FLOPs and COMPs required for inference of the iForest is then:

$$\text{FLOP}_{\text{IF}} = T + 3 \tag{10.10}$$
$$\text{COMP}_{\text{IF}} = T(\lceil \log_2(\psi) \rceil - 1) \tag{10.11}$$

The selected number of iTrees in section 8.3 is $T = 150$. The number of samples for each tree can be found using the specified 60% of the total samples used for each iTree. In this case, the model was trained using $345\,600$ total samples, leading to a number of samples per iTree of $\psi = 207\,360$. Substituting these values into Equation 10.10 and 10.11 leads to a required 153 FLOPs and 2550 COMPs.

Next, to estimate the static memory usage of the iForest, the distinction between internal and leaf nodes has to be made. Internal nodes typically store four elements: the index of the feature that is being split, a split threshold, and pointers to the left and right children nodes [40]. It is assumed each of these is a 32-bit value. Leaf nodes do not need to store children or split information. Instead, they typically store a single value: the depth of the leaf node in the tree [40]. This is used as the path length when traversing the tree, avoiding the need to keep track of it separately. This is also assumed to be a 32-bit number, resulting in:

$$\text{MEM}_{\text{IF-internal}}^{\text{static}} = 16 \text{ [bytes]} \tag{10.12}$$

$$\text{MEM}_{\text{IF-leaf}}^{\text{static}} = 4 \text{ [bytes]} \tag{10.13}$$

The number of internal nodes in an iTree is $N_{\text{internal}} = \psi - 1$ and the number of leaf nodes is $N_{\text{leaf}} = \psi$ under the worst-case assumption that the tree is full [40]. The memory used by a tree is then the sum of the memory used for internal nodes and the memory used for leaf nodes:

$$\text{MEM}_{\text{IF-tree}}^{\text{static}} = N_{\text{internal}}\ \text{MEM}_{\text{IF-internal}}^{\text{static}} + N_{\text{leaf}}\ \text{MEM}_{\text{IF-leaf}}^{\text{static}} = 20\psi - 16 \tag{10.14}$$

The runtime memory usage is much lower than the static memory usage. The found path lengths for each iTree need to be stored before averaging, leading to T values. Other variables, such as tracking variables needed for tree traversal, depend heavily on the implementation and are assumed negligible for simplicity. The static memory used by the entire iForest is the static memory used for one iTree multiplied by the number of iTrees T. This results in:

$$\text{MEM}_{\text{IF}}^{\text{runtime}} = 4T \tag{10.15}$$

$$\text{MEM}_{\text{IF}}^{\text{static}} = T\ \text{MEM}_{\text{tree}} = T(20\psi - 16) \tag{10.16}$$

Using the tuned value of $T = 150$ iTrees, and the value of ψ found before, this results in a static memory usage of 6.221×10^8 bytes (622.1 MB) and runtime memory usage of 600 bytes.

10.4 | LSTM Predictor

The resource usage of the LSTM predictor involves large matrix multiplications and related operations. To determine the number of FLOPs and memory used for inference, the processing of one window, as described in subsection 3.4.4, is followed step-by-step.

To update the state and hidden output for one LSTM cell based on one input, the following computations are performed:

- Forget-, input-, and output gate (Equation 3.2, 3.3, and 3.6 respectively)
- Cell state candidate and update (Equation 3.4 and 3.5 respectively)
- Cell output (Equation 3.7)

The calculation of the gates and the candidate cell state follow a similar structure:

$$\text{activation}\,(W\,x_t + U\,h_{t-1} + b)$$

where W and U are weight matrices, b is a bias vector, and either a sigmoid or tanh activation function is used. The hidden state h is a vector of dimension H. The input x is a vector whose dimension is noted as I.

For the calculation of one gate, the required number of FLOPs for the first matrix multiplication ($W\,x_t$) follows from $\text{FLOP}_{\text{matmul}}(H, I, 1)$ and for the second matrix multiplication ($U\,h_{t-1}$) from $\text{FLOP}_{\text{matmul}}(H, H, 1)$. The bias vector addition adds H FLOPs. The resulting vector is passed through the activation function, the sigmoid. The sigmoid is defined as $1/(1 + \exp(-x))$, showing a division, addition, exponentiation, and negation, for an assumed total of 4 FLOPs. Applying this to the vector of size H thus adds $4H$ FLOPs. Combined, this leads to the required number of FLOPs for the computation of one gate output:

$$\text{FLOP}_{\text{LSTM-gate}}(I, H) = \text{FLOP}_{\text{matmul}}(H, I, 1) + \text{FLOP}_{\text{matmul}}(H, H, 1) + H + 4H = 2H^2 + 2IH + 3H \tag{10.17}$$

This computation occurs four times, for the input-, forget-, and output gate, as well as the cell candidate values. Besides this, the cell state is updated and the output hidden state is computed. This first operation, described by Equation 3.5, consists of two element-wise vector multiplications and a vector addition, each of dimension H. This combines to $3H$ FLOPs. The second operation, computing the output state in Equation 3.7, involves applying the tanh activation function on the cell state. The number of FLOPs depends on implementation and the input value, but for rational numbers and single-precision is roughly eight FLOPs [109]. Thus, to apply the function to the cell state of size H, this yields $8H$ FLOPs. The result is multiplied with the output gate (element-wise), adding H FLOPs. Combining these operations with four gate computations leads to the required number of FLOPs for one LSTM cell:

$$\text{FLOP}_{\text{LSTM-cell}}(I, H) = 4 \cdot \text{FLOP}_{\text{LSTM-gate}}(I, H) + 3H + 9H = 8H^2 + 8IH + 24H \tag{10.18}$$

The value for the input dimension I depends on which cell is considered. The first cell receives an input sample s_t of dimension S, while subsequent cells receive the hidden state output of the previous cell, with dimension H. Denoting the number of LSTM layers as L, this yields the required FLOPs for the computation of all stacked cells:

$$\begin{aligned}\text{FLOP}_{\text{LSTM-stack}}(S, H, L) &= \text{FLOP}_{\text{lstm-cell}}(S, H) + \sum^{L-1} \text{FLOP}_{\text{lstm-cell}}(H, H) \\ &= 8H^2 + 8SH + 24H + (L-1)\left(16H^2 + 24H\right)\end{aligned} \tag{10.19}$$

Considering an input window containing N samples, each sample is processed one-by-one by each of the cells in the stack. The final output of the final cell is then passed through the dense layer, which performs a final weight and bias computation before obtaining the predicted next sample:

$$\begin{aligned}\text{FLOP}_{\text{LSTM-predict}}(S, H, L, N) &= N \cdot \text{FLOP}_{\text{LSTM-stack}}(S, H, L) + \text{FLOP}_{\text{matmul}}(S, H, 1) + S \\ &= N\left(8H^2 + 8SH + 24H + (L-1)\left(16H^2 + 24H\right)\right) + S(2H-1) + S\end{aligned} \tag{10.20}$$

Finally, the anomaly score is calculated as the norm of the difference between the predicted and observed sample. This leads to a subtraction of S FLOPs, and a vector norm of $2S$ FLOPs. The total required FLOPs for anomaly detection with the LSTM predictor is then the summation of the FLOPs required for prediction and the calculation of the score. The LSTM predictor does not require any comparisons for inference. This results in:

$$\begin{aligned}\text{FLOP}_{\text{LSTM}}(S, H, L, N) &= \text{FLOP}_{\text{LSTM-predict}}(S, H, L, N) + 3S \\ &= N\left(8H^2 + 8SH + 24H + (L-1)\left(16H^2 + 24H\right)\right) + S(2H-1) + 4S\end{aligned} \tag{10.21}$$

$$\text{COMP}_{\text{LSTM}} = 0 \tag{10.22}$$

The tuned hyperparameters of the LSTM predictor were found in section 8.3: sample dimension $S = 7$, hidden dimension $H = 64$, window size $N = 50$, and number of stacked LSTM cell $L = 4$. Using these values in Equation 10.21 yields a required $11\,956\,117$ FLOPs.

The static memory consists of the weights and biases contained within the LSTM cells, as well as the final dense layer. Each gate uses two weight matrices and one bias vector, sized based on the input dimension I and hidden dimension H. Again assuming each element is a 32-bit floating-point number:

$$\text{MEM}^{\text{static}}_{\text{LSTM-gate}}(I, H) = 4\left(H^2 + IH + H\right) \tag{10.23}$$

Each cell consists of four gates, leading to the static memory usage of one LSTM cell:

$$\text{MEM}^{\text{static}}_{\text{LSTM-cell}}(I, H) = 4 \cdot \text{MEM}^{\text{static}}_{\text{LSTM-gate}} = 16\left(H^2 + IH + H\right) \tag{10.24}$$

The size of the input depends again on which cell is considered. The first cell has input dimension S, while the remaining cells have input dimension H:

$$\begin{aligned}\text{MEM}^{\text{static}}_{\text{LSTM-stack}}(S, H, L) &= \text{MEM}^{\text{static}}_{\text{LSTM-cell}}(S, H) + \sum^{L-1} \text{MEM}^{\text{static}}_{\text{LSTM-cell}}(H, H) \\ &= 16\left(H^2 + SH + H\right) + (L-1)\left(32H^2 + 16H\right)\end{aligned} \tag{10.25}$$

Finally, the dense layer adds a weight matrix ($S \times H$) and bias vector ($S \times 1$):

$$\begin{aligned}\text{MEM}^{\text{static}}_{\text{LSTM}}(S,H,L) &= \text{MEM}^{\text{static}}_{\text{LSTM-stack}}(S,H,L) + 4(SH+S) \\ &= 16\left(H^2+SH+H\right)+(L-1)\left(32H^2+16H\right)+4(SH+S)\end{aligned} \tag{10.26}$$

Using the tuned values of $S = 7$, $H = 64$, and $L = 4$, this results in an estimated static memory usage of $471\,836$ bytes (471.8 kB).

Part of the runtime memory are the cell states ($H \times 1$) and hidden states ($H \times 1$) that persist for each cell during inference. Thus, for each cell, $8H$ bytes are stored. With L cells, this becomes $8LH$ bytes. Next, it is assumed that the values of each of the four gates ($H \times 1$) are temporarily stored, as well as the intermediate results of the candidate cell state and output activation (both $H \times 1$). This adds six vectors of H elements each, adding $24H$ bytes for all cells. Here, it is assumed this memory is reused between calculations. Summing the contributions yields:

$$\text{MEM}^{\text{runtime}}_{\text{LSTM}} = 8LH + 24H \tag{10.27}$$

Using the values $L = 4$ and $H = 64$, this results in a runtime memory usage of 3584 bytes (3.58 kB).

10.5 | Comparison

The results of the static analysis are summarized in Table 10.1. Note that these are estimates that do not exactly correlate to real-world performance directly due to variations in the ultimate implementation and hardware effects that have not been accounted for. However, they serve as order-of-magnitude values for comparison of the methods. Additionally, the amounts listed in the table represent one inference. To effectively make use of the FKF, multiple local filters should exist, thus requiring multiple instances of the fault detection method. In this simulation, three local filters were used, thus the values in this table should be multiplied by three to obtain the total fault detection cost.

Table 10.1: Comparison of estimated resource usage for one inference from static analysis

Group	**Method**	**Computation**		**Memory usage**	
		FLOPs	COMPs	Static	Runtime
Conventional	Sensitivity factor	377	0	0 B	192 B
	Measurement residual	16	0	0 B	32 B
ML-based	iForest	153	2550	622.1 MB	600 B
	LSTM predictor	1.20×10^7	0	471.8 kB	3.58 kB

The conventional methods are substantially less computationally demanding than the ML-based alternatives. Both conventional approaches require only minimal computation, combined with negligible memory requirements. Notably, there is also no need to store any model parameters. The ML-based methods show greater requirements, with the iForest having the largest model size, and the LSTM predictor the most demanding computations.

To evaluate whether the methods could feasibly run on CubeSat hardware, two constraints were considered: the memory constraint and the compute constraint. First, it should be determined whether the static and runtime memory fits within the budget of the OBC. Second, the required number of FLOPs and comparisons should be executed within reasonable real-time margins. This analysis is based in part on the collection of OBCs identified in section 2.3.

10.5.1 | Memory Constraint

Both conventional methods require no static memory, with runtime memory in the order of tens to hundreds of bytes. This memory footprint is negligible compared to the listings in Table 2.1, fitting comfortably within any OBC's memory constraints.

The iForest requires hundreds of megabytes to store the trained model, depending in large part on the number of samples used by each iTree. None of the listed OBCs in Table 2.1 provide random access memory (RAM) in this range. The most capable units provide several gigabytes of flash memory, though this type of storage can be too slow for the repeated tree traversal that is required for the iForest method. When the majority of the model cannot be stored in RAM, the access latency greatly increases [110]. Storing and accessing such a large model on a microcontroller platform is therefore infeasible. The runtime memory of several hundred bytes is again negligible, but irrelevant since the static memory requirement already exceeds the available capacity by an order of magnitude.

The LSTM predictor requires several hundred kilobytes of static memory, and several kilobytes of runtime memory. The runtime requirement fits comfortably on most options listed in Table 2.1. The static requirement is, however, more constraining. Lower-end units with only a few hundred kilobytes of RAM likely cannot store the entire model while maintaining other memory overhead. Some higher-end options provide more capacity, likely enough to store the model. This was also observed in previous studies, such as by Horne et al. in 2023 [111]. Here, a similar model was used in combination with a similar processor based on the 32-bit ARM Cortex-M7. Thus, use of the LSTM predictor on a CubeSat OBC using a microcontroller is feasible memory-wise, though requiring higher-end options.

10.5.2 | Compute Constraint

Again, both conventional methods present negligible computational requirements. Even at very high sampling rates, the required workload would likely correspond to microseconds of computation.

The iForest requires a similar amount of FLOPs compared to the conventional methods. However, several thousand comparisons are introduced during tree traversal. Each comparison involves memory access and branching, which can be expensive depending on the architecture. As was identified before, the model cannot fit into RAM. Thus, if the model were to be placed in long-term memory, such as flash memory, accessing parameters would make real-time traversal infeasible. Even though the FLOP count is small, the tree traversal cost renders this method infeasible.

With approximately 1.2×10^7 FLOPs, the LSTM predictor is the most computationally demanding method by far. Assuming the OBC is indeed capable of 70 MFLOPs per second as found in the aforementioned study [59], it could be feasible to run the LSTM predictor next to other application overhead. However, the update frequency would be limited, likely not exceeding 1 Hz.

10.5.3 | Summary

Both conventional methods require negligible computation and memory capacity, requirements which are easily met by even low-end microcontrollers used in CubeSat OBCs. The ML-based methods are less straightforward.

The iForest suffers from a very significant model size which cannot fit in the RAM capacity of any microcontroller analysed in section 2.3. An alternative would be to store the model in flash memory, but this greatly slows memory accesses which makes tree traversal infeasible. Therefore, the iForest is considered infeasible, unless more powerful OBC alternatives are used or the implementation is optimized.

The constraining static memory size is linearly dependent on the number of iTrees and the number of samples the model is trained with. In the implementation in this project, over 200 thousand samples are used to train each iTree. Reducing this number can greatly reduce the memory footprint of the iForest, but this would simultaneously affect performance. To illustrate, the largest RAM capacity listed in section 2.3 is 64 MB. Assuming the entire capacity would be available to store the iForest model, only about 10% of the currently used training samples could be used per iTree. As was observed during tuning of the hyperparameters in section 8.3, this would drastically lower the F_1-score.

The LSTM predictor model fits on several of the listed higher-end microcontrollers, which is confirmed by a case study with a similar model and microcontroller [111]. The necessary computations would be feasible under limited inference rates. With optimization, the viability of this method could be further improved.

11 Conclusion

The research objective identified in chapter 1 specified the desire to improve the reliability of the attitude determination system (ADS) in CubeSats by applying machine learning (ML) to fault detection (FD) within the federated Kalman filter (FKF). Subsequently, the main research question was defined as follows:

> *What is the accuracy and computational impact of a machine learning-based classifier in performing fault detection within a federated Kalman filter for CubeSat attitude determination?*

In section 11.1, this question is answered based on the results presented in this report. This is followed by recommendations for future research into this topic, presented in section 11.2.

11.1 | Answers to Research Questions

In chapter 1, several sub-questions of the main research question were defined. In the following sections, conclusions are drawn using the results and observations obtained throughout the research project.

RQ1: Which ML-based methods are suitable for fault detection in an FKF?

In this research project, two ML-based FD methods were selected through a trade-off process. Five candidate methods were chosen based on literature and expert input, ranging from lightweight and classical methods, to deep-learning temporal models. Unsupervised methods are preferred due to the general lack of training data in the space industry, especially datasets containing annotated anomalies. Additionally, these methods can detect faults not seen during training.

Using weighted scoring and a sensitivity analysis, the long short-term memory (LSTM) predictor and isolation forest (iForest) were selected for application within the context of FD in the FKF. First, the LSTM predictor is a type of recurrent neural network (RNN) which excels at recognizing temporal patterns. Its ability to consider the history of variables allows it to detect even gradual faults more rapidly. It comes with a significant computational cost due to the large amounts of matrix operations during inference.

The iForest is a more computationally lightweight and simple alternative. It differs from most other ML-based methods in that no weights or biases are learned. Instead, many binary trees, isolation trees (iTrees), are created which randomly divide samples into branches. This makes the method robust to noise and simple to implement.

Beyond their theoretical suitability, the selection of the LSTM predictor and iForest reflects a distinction between temporal and point-wise FD. The FKF produces time-varying signals, and methods capable of exploiting this, such as the LSTM predictor, are expected to have an inherent advantage in this context. Point-wise methods, like the iForest, operate on individual samples and therefore cannot capture faults that evolve over time as well.

As is shown in the subsequent research questions, experimental results confirm this expectation. The extent to which each method can separate faulty from nominal data is shown in the answer to RQ2. This is compared to the conventional methods in the answer to RQ3.

RQ2: How effectively can ML-based classifiers distinguish between faulty and non-faulty subfilter outputs in an FKF?

To determine the behaviour of the selected ML methods in response to faults, a detailed simulation was developed. A reference CubeSat was placed in four operational scenarios: tumbling, target tracking, safe mode, and a nominal mode. Sensor models for the gyroscope, star tracker, magnetometer, and Sun sensor were developed and implemented, allowing for the injection of various fault types.

The gyroscope was combined with each of the other sensors to create three local filters, using the unscented quaternion estimator (USQUE) algorithm. Their local estimates are fused by a master filter, forming an FKF in no-reset mode. Three types of fault were simulated for each scenario:

- **Stuck fault**: sensor output becomes constant, gradually deviating from the true value.
- **Axis fault**: partial failure, one of the three sensor axes fails.
- **Zero fault**: complete failure, sensor only produces zero output.

The response of the ML-based detection methods to these faults was recorded and analysed. Performance metrics were used to quantify their ability to detect faults. First, range-based alternatives to the traditional recall and precision were used and subsequently combined to a range-based F_1-score. In addition, the duration between the onset of a fault and the first detection of that fault was measured. Finally, the receiver operating characteristic (ROC) curves using the traditional category counts were generated. These scores were obtained for each method in each separate combination of scenario and fault case.

The results show that both methods were able to identify the occurrence of faults reliably. During nominal operation, the automatically selected threshold is rarely exceeded. Upon the start of a fault, the anomaly score increases drastically and remains above the threshold for the duration of the fault, before returning to nominal levels.

The iForest achieved a mean F_1-score of 0.69 across all scenarios and fault types, while the LSTM predictor exceeds this result with an F_1-score of 0.81. Both show limited sensitivity to the scenario and fault case. In terms of time-to-detection, the LSTM predictor again outperforms the iForest, with mean times of 7.34 and 41.1 seconds respectively. Here, the LSTM predictor is very consistent, providing rapid detection in all considered cases. The iForest shows more variance, with detection times reaching 180 seconds for the gradual stuck fault.

Overall, the LSTM predictor was found to be more reliable than the iForest. The latter method only considers the current timestep, meaning a gradual deviation is more difficult to detect until it has grown significantly. This leads to a larger time between the onset and detection of a fault. In contrast, the LSTM predictor considers the temporal history, allowing it to detect these same deviations more rapidly.

Taken together, the results show that ML-based methods are effective fault detectors in the FKF. They consistently separate faulty from nominal behaviour, respond reliably across scenarios, and produce limited false alarms. The LSTM predictor is highly effective, while the iForest, despite being less responsive, still achieves reliable detection in most cases.

RQ3: How accurate are ML-based classifiers in detecting faults within an FKF compared to conventional methods?

The selected ML-based methods were compared to two conventional methods taken from literature: the sensitivity factor and measurement residual ratio. Using the simulation of a CubeSat in several scenarios, affected by multiple different faults, their accuracy was scored using the same performance metrics.

The obtained results indicate that of the conventional methods, the sensitivity factor is superior. It provides moderately accurate detection with a mean F_1-score of 0.73 and a mean detection time of 23.1 seconds, though the performance shows significant variance with scenario and fault type. The measurement residual ratio fails to capture the entirety of some faults, especially when they are gradual in nature, showing sensitivity to the Kalman filter falsely attributing anomalies to gyroscope bias. This effect can be reduced through tuning of the filter, but not fully eliminated for long-duration faults. As a result, the measurement residual received very low accuracy scores, never exceeding an F_1-score of 0.51. The sensitivity factor is therefore used as the baseline for comparison of the conventional and ML-based methods.

The iForest shows similar accuracy to that of the sensitivity factor with an F_1-score 5.4% lower, though it is more scenario- and fault-agnostic. Its average time-to-detection has drastically increased by 77.7%.

The LSTM predictor was shown to be the most reliable detection method overall, improving all metrics found for the sensitivity factor. Its average accuracy is the highest of the considered methods with little variance, indicating situation- and fault-agnostic detection, boasting a 10.2% increase in F_1-score compared to the sensitivity factor. Furthermore, this method detected faults within 15.8 seconds across all tested scenarios and faults, with the mean of 7.34 seconds being over three times lower than the mean detection time of the conventional method. The inclusion of temporal context in its architecture allows the LSTM predictor to detect smaller changes when compared to the other point-wise methods.

Based on these results, the point-wise iForest offers no improvement to the conventional detection using the sensitivity factor. The LSTM predictor, however, shows a clear improvement. With a mean F_1-score 10.2% higher than that of the sensitivity factor, and a threefold reduction in detection time, this method significantly improves upon the conventional fault detection.

RQ4: What are the computational costs of deploying ML-based classifiers for fault detection in an FKF on CubeSat-class hardware?

A static analysis was performed to estimate the computational impact of inference using the conventional and ML-based fault detection methods. For this analysis, both the computation and memory constraints were considered. A collection of commercial off-the-shelf (COTS) microcontroller-based on-board computers (OBCs) served to indicate the typical hardware available for attitude determination and control system (ADCS) on a CubeSat.

The computation constraint was evaluated by determining the number of floating-point operations (FLOPs) and comparisons that are needed for one inference. While these do not perfectly correlate with real-world performance, they act as an indicator for the volume of computation that needs to occur. The memory constraint was evaluated by determining the static and runtime memory needs of each method. The static memory entails the storage of the trained parameters for the models, while the runtime memory determines the dynamic memory needed for inference, such as for temporary data or intermediate results.

From the analysis, the conventional methods were shown to require minimal computation and memory, negligible compared to the typical CubeSat OBCs. The iForest relies mostly on comparisons within the iTrees, which leads to many memory accesses. The static memory required to store the model far exceeds the typical capacity of the random access memory (RAM) in an OBC. The alternative, storing the model in long-term memory, would lead to the required memory accesses becoming a bottleneck. This makes the use of the iForest on this hardware infeasible. To employ this method, more powerful hardware is needed, or the model size should be reduced.

In contrast, the bottleneck for the LSTM predictor is the high computational volume, five orders of magnitude greater than that of the sensitivity factor. The static memory requirement is now much lower and comfortably fits on most OBCs. The large number of matrix multiplications leads to a significant portion of computational resources being used, even on higher-end options. It would be possible to use the LSTM predictor on this hardware at limited inference rates, likely not exceeding 1 Hz depending on the processing unit. Again, reducing the computational volume through optimization can improve the viability further.

11.2 | Recommendations

Throughout the research project several areas were identified which could benefit from further research. These are collected and detailed in the following sections.

11.2.1 | Gyroscope Dependency

In the formulation of the FKF presented in this research project, each local filter requires the measurements of the gyroscope to generate the propagated state. However, this introduces a single point of failure, which is not desirable for a fault-tolerant system. Ideally, each local filter relies solely on one sensor or sensor group. Any faults in this group would only affect the corresponding local filter. Currently, a fault in the gyroscope would affect all three local filters.

There are several approaches that could be used to decouple the gyroscope from the local filters. For example, many star trackers produce an angular velocity measurement besides the absolute attitude measurement. Alternatively, the filters can estimate the angular rate themselves. This could be used to replace the gyroscope data, making the local filter solely rely on one sensor group. Investigating approaches which reduce the reliance of the local filters on the same sensor can further improve the reliability of the ADS.

11.2.2 | Optimization

During the selection and implementation of the ML-based fault detection methods in this study, the hyperparameters of the selected methods were tuned to maximize detection accuracy. However, no consideration was given to the model size and computational load during the tuning process. As was found during the computational load analysis, the model size of the iForest is problematic, while the large computational volume of the LSTM predictor takes up a significant part of the available resources of most OBCs on CubeSats.

Instead, the models could be optimized for performance while considering these computational constraints. Limits on the computational volume and memory usage can be used to guide the tuning process, setting hard limits on the range of values for the hyperparameters. Moreover, the methods in this research project were trained on 7-dimensional features which could be reduced to decrease computational load. Additionally, optimized implementations of the methods, including pruning and quantization, should be studied on comparable hardware. A more detailed study of computational viability would provide better insight into the feasibility of these methods as an alternative to the conventional methods on CubeSat hardware.

11.2.3 | Fault Detection

The fault detection methods presented in this report each provide a numerical anomaly score. If the selected threshold is exceeded, the local filter is excluded from the master filter such that the information from the anomalous filter is not included in the final estimate. Instead of merely detecting the presence of a fault, it could be classified instead. This provides information about the nature of the fault and how to resolve it, either manually or automatically.

The anomaly score could also be used as a form of confidence instead of a hard cut-off based on a threshold. The confidence in the output could act as a weight in the master filter. It could then rely more on the local filters which are deemed more trustworthy, and less on those seen as more anomalous.

The iForest was applied as a point-wise detector in this project. Incorporation of temporal context could be achieved by implementing a sliding window. While other methods, such as the LSTM predictor, inherently include this information, this external approach could improve detection performance, especially for gradual faults.

The implementation of a hybrid method could be used to combine the benefits of the conventional and ML-based methods. A conventional approach with a low threshold could be used as an early-warning system. The more expensive ML-based methods are only used if this early detection is tripped in order to refine the detection. Studying these alterations of the methodology presented in this report could lead to a solution that more closely approaches an operational system.

11.2.4 | Other Recommendations

The ability of the FKF to decouple the update rates of each of the filters was not used in this research project. In the simulation presented in this report, all filters run at the same frequency for simplicity. However, the data throughput can be increased if the filter rates match the measurement rates of the associated sensors. This presents new challenges, such as synchronization for the master filter update.

Many ML-based models trained on simulations show performance degradation when applied to real-world data. To determine whether the models trained in this project still perform well on real satellite data, more investigation is needed. This requires collecting datasets from the ADS of in-orbit satellites which contain anomalies. The models can then be applied, allowing for comparison between the simulated and real performance.

To determine the performance of the ML-based methods in this project, a single fault was introduced in one sensor at a time. While rare, it could be possible for multiple faults to occur simultaneously. When multiple sensors suffer from a fault, the FKF receives less true information to base its estimate on. In the structure used in this project, if two out of three sensors became faulty simultaneously, the third nominal sensor might falsely appear to deviate from the other sensors. The response of the fault detection methods to these more complicated fault scenarios should be studied to determine whether they are still reliable under these conditions.

References

[1] A.O. Erlank and C.P. Bridges. "Reliability analysis of multicellular system architectures for low-cost satellites". In: *Acta Astronautica* 147 (2018), pp. 183–194. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2018.04.006.

[2] National Academies of Sciences, Engineering, and Medicine. *Achieving Science with CubeSats: Thinking Inside the Box*. Washington, DC: The National Academies Press, 2016. ISBN: 978-0-309-44263-3. DOI: 10.17226/23503.

[3] T. Villela et al. "Towards the Thousandth CubeSat: A Statistical Overview". In: *International Journal of Aerospace Engineering* 2019.1 (2019), p. 5063145. DOI: 10.1155/2019/5063145.

[4] E. Kulu. "CubeSats & Nanosatellites — 2024 Statistics, Forecast and Reliability". In: *Proceedings of the International Astronautical Congress, IAC*. Vol. 2024. 2024.

[5] J.K. Wayer, J.F. Castet, and J.H. Saleh. "Spacecraft attitude control subsystem: Reliability, multi-state analyses, and comparative failure behavior in LEO and GEO". In: *Acta Astronautica* 85 (2013), pp. 83–92. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2012.12.003.

[6] C. Hajiyev and H.E. Soken. *Fault Tolerant Attitude Estimation for Small Satellites*. 1st ed. Boca Raton, United States: CRC Press, 2020. ISBN: 9781351248839.

[7] A. Scholz et al. "Flight results of the COMPASS-1 picosatellite mission". In: *Acta Astronautica* 67.9 (2010), pp. 1289–1298. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2010.06.040.

[8] A. Slavinskis et al. "Flight Results of ESTCube-1 Attitude Determination System". In: *Journal of Aerospace Engineering* 29.1 (2016), p. 04015014. DOI: 10.1061/(ASCE)AS.1943-5525.0000504.

[9] S. Rossi et al. "The SwissCube's technologies results after four years of flight". In: *Proceedings of the International Astronautical Conference*. 2013, pp. 23–27.

[10] M. Rizwan Mughal et al. "Aalto-1, multi-payload CubeSat: In-orbit results and lessons learned". In: *Acta Astronautica* 187 (2021), pp. 557–568. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2020.11.044.

[11] M.N Hasan, M. Haris, and S. Qin. "Fault-tolerant spacecraft attitude control: A critical assessment". In: *Progress in Aerospace Sciences* 130 (2022), p. 100806. ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2022.100806.

[12] J.K. Wayer, J.F. Castet, and J.H. Saleh. "Spacecraft attitude control subsystem: Reliability, multi-state analyses, and comparative failure behavior in LEO and GEO". In: *Acta Astronautica* 85 (2013), pp. 83–92. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2012.12.003.

[13] X.Y. Ji and J. Wang. "Statistical Analysis of Spacecraft Failure in Full-Life Based on STED". In: *2019 IEEE 10th International Conference on Mechanical and Aerospace Engineering (ICMAE)*. 2019, pp. 89–96. DOI: 10.1109/ICMAE.2019.8880888.

[14] R.P. Perumal et al. "Small Satellite Reliability: A Decade in Review". In: (2021). URL: https://digitalcommons.usu.edu/smallsat/2021/all2021/244/.

[15] J. Jiang and X. Yu. "Fault-tolerant control systems: A comparative study between active and passive approaches". In: *Annual Reviews in Control* 36.1 (2012), pp. 60–72. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2012.03.005.

[16] F.L. Markley and J.L. Crassidis. *Fundamentals of Spacecraft Attitude Determination and Control*. 1st ed. New York, United States: Springer, 2014. ISBN: 978-1-4939-0802-8.

[17] C. Wei et al. "An overview of prescribed performance control and its application to spacecraft attitude system". In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 235.4 (2021), pp. 435–447. DOI: 10.1177/0959651820952552.

[18] N.A. Carlson. "Federated filter for fault-tolerant integrated navigation systems". In: *IEEE PLANS '88.,Position Location and Navigation Symposium, Record. 'Navigation into the 21st Century'.* 1988, pp. 110–119. DOI: 10.1109/PLANS.1988.195473.

[19] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552.

[20] P.J. Lawrence and M.P. Berarducci. "Comparison of federated and centralized Kalman filters with fault detection considerations". In: *Proceedings of 1994 IEEE Position, Location and Navigation Symposium - PLANS'94*. 1994, pp. 703–710. DOI: 10.1109/PLANS.1994.303380.

[21] X. Shen et al. "Globally optimal distributed Kalman filtering fusion". In: vol. 55. 2012, pp. 512–529. DOI: 10.1007/s11432-011-4538-7.

[22] S. Roy, R.H. Hashemi, and A.J. Laub. "Square root parallel Kalman filtering using reduced-order local filters". In: *IEEE Transactions on Aerospace and Electronic Systems* 27.2 (1991), pp. 276–289. DOI: 10.1109/7.78303.

[23] F. Lu et al. "A Hybrid Kalman Filtering Approach Based on Federated Framework for Gas Turbine Engine Health Monitoring". In: *IEEE Access* 6 (2018), pp. 9841–9853. DOI: 10.1109/ACCESS.2017.2780278.

[24] N.A. Carlson and M.P. Berarducci. "Federated Kalman Filter Simulation Results". In: *NAVIGATION* 41.3 (1994), pp. 297–322. DOI: 10.1002/j.2161-4296.1994.tb01882.x.

[25] N.A. Carlson. "Information-sharing approach to federated Kalman filtering". In: *Proceedings of the IEEE 1988 National Aerospace and Electronics Conference*. 1988. DOI: 10.1109/NAECON.1988.195221.

[26] M. Ilyas et al. "Federated Unscented Kalman Filter design for multiple satellites formation flying in LEO". In: *2008 International Conference on Control, Automation and Systems*. 2008, pp. 453–458. DOI: 10.1109/ICCAS.2008.4694683.

[27] J. Shi et al. "Federated Kalman Filter-Based Fusion of LEO and GNSS Positioning". In: *IEEE 99th Vehicular Technology Conference*. 2024, pp. 1–5. DOI: 10.1109/VTC2024-Spring62846.2024.10683248.

[28] J. Bae and Y. Kim. "Attitude Estimation for Satellite Fault Tolerant System Using Federated Unscented Kalman Filter". In: *International Journal of Aeronautical and Space Sciences* 11.2 (June 2010), pp. 80–86. DOI: https://doi.org/10.5139/IJASS.2010.11.2.080.

[29] F. Ni, H. Quan, and W. Li. "Micro-satellite attitude determination based on federated Kalman filter using multiple sensors". In: *Automotive, Mechanical and Electrical Engineering*. CRC Press, 2017, pp. 175–179.

[30] J. Mess, F. Dannemann, and F. Greif. "Techniques of Artificial Intelligence for Space Applications - A Survey". In: *European Workshop on On-Board Data Processing (OBDP2019)*. Vol. 2019. 2019. URL: elib.dlr.de/129255.

[31] L. Zhang et al. "Federated nonlinear predictive filtering for the gyroless attitude determination system". In: *Advances in Space Research* 58.9 (2016), pp. 1671–1681. ISSN: 0273-1177. DOI: 10.1016/j.asr.2016.07.023.

[32] A. Poghosyan and A. Golkar. "CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions". In: *Progress in Aerospace Sciences* 88 (2017), pp. 59–83. ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2016.11.002.

[33] A. Cratere et al. "On-Board Computer for CubeSats: State-of-the-Art and Future Trends". In: *IEEE Access* 12 (2024), pp. 99537–99569. DOI: 10.1109/ACCESS.2024.3428388.

[34] J. Murphy, J.E. Ward, and B. Mac Namee. "Machine Learning in Space: A Review of Machine Learning Algorithms and Hardware for Space Applications." In: *AICS* 3105 (2021), pp. 72–83. URL: ceur-ws.org/Vol-3105/paper21.pdf.

[35] N. Mejri et al. "Unsupervised anomaly detection in time-series: An extensive evaluation and analysis of state-of-the-art methods". In: *Expert Systems with Applications* 256 (2024), p. 124922. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2024.124922.

[36] S. Ramachandran, M. Rosengarten, and C. Belardi. “Semi-Supervised Machine Learning for Spacecraft Anomaly Detection and Diagnosis”. In: *2020 IEEE Aerospace Conference*. 2020, pp. 1–10. DOI: 10.1109/AERO47225.2020.9172454.

[37] Z. Zeng et al. “Spacecraft Telemetry Anomaly Detection Based on Parametric Causality and Double-Criteria Drift Streaming Peaks over Threshold”. In: *Applied Sciences* 12.4 (2022). ISSN: 2076-3417. DOI: 10.3390/app12041803.

[38] M.M. Breunig et al. “LOF: identifying density-based local outliers”. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’00. Dallas, Texas, USA: Association for Computing Machinery, 2000, pp. 93–104. ISBN: 1581132174. DOI: 10.1145/342009.335388.

[39] B. Schölkopf et al. “Estimating the Support of a High-Dimensional Distribution”. In: *Neural Computation* 13.7 (2001), pp. 1443–1471. DOI: 10.1162/089976601750264965.

[40] F.T. Liu, K.M. Ting, and Z. Zhou. “Isolation Forest”. In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[41] F.N. Pirmoradi, F. Sassani, and C.W. de Silva. “Fault detection and diagnosis in a spacecraft attitude determination system”. In: *Acta Astronautica* 65.5 (2009), pp. 710–729. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2009.03.002.

[42] Z.Q.Li, L. Ma, and K. Khorasani. “A Dynamic Neural Network-based Reaction Wheel Fault Diagnosis for Satellites”. In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. 2006, pp. 3714–3721. DOI: 10.1109/IJCNN.2006.247387.

[43] H.A. Talebi and K. Khorasani. “A Neural Network-Based Multiplicative Actuator Fault Detection and Isolation of Nonlinear Systems”. In: *IEEE Transactions on Control Systems Technology* 21.3 (2013), pp. 842–851. DOI: 10.1109/TCST.2012.2186634.

[44] B. Sun et al. “Fault Identification for a Closed-Loop Control System Based on an Improved Deep Neural Network”. In: *Sensors* 19.9 (2019). ISSN: 1424-8220. DOI: 10.3390/s19092131.

[45] F. Bingqing et al. “Anomaly detection of spacecraft attitude control system based on principal component analysis”. In: *2017 29th Chinese Control And Decision Conference (CCDC)*. 2017, pp. 1220–1225. DOI: 10.1109/CCDC.2017.7978704.

[46] O. Nasri et al. “Spacecraft Actuator Diagnosis with Principal Component Analysis: Application to the Rendez-Vous Phase of the Mars Sample Return Mission”. In: *Journal of Control Science and Engineering* 2015.1 (2015), p. 204918. DOI: 10.1155/2015/204918.

[47] I. Gueddi, O. Nasri, and K. Ben Othman. “Spacecraft Reaction Wheels Fault Diagnosis: Interval Kernel Principal Component Analysis”. In: *IFAC-PapersOnLine* 54.21 (2021). Control Conference Africa CCA, pp. 222–227. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2021.12.038.

[48] A. Li, M. Liu, and Y. Shi. “Adaptive sliding mode attitude tracking control for flexible spacecraft systems based on the Takagi-Sugeno fuzzy modelling method”. In: *Acta Astronautica* 175 (2020), pp. 570–581. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2020.05.041.

[49] Y. Mei et al. “Fuzzy adaptive sliding mode fault estimation and fixed-time fault-tolerant control for coupled spacecraft based on SE(3)”. In: *Aerospace Science and Technology* 126 (2022), p. 107673. ISSN: 1270-9638. DOI: 10.1016/j.ast.2022.107673.

[50] S. Liu et al. “A New Fault Diagnosis Model of Flywheel System based on Belief Rule Base and Fuzzy Fault Tree Analysis”. In: *2021 CAA Symposium on Fault Detection, Supervision, and Safety for Technical Processes (SAFEPROCESS)*. 2021, pp. 1–6. DOI: 10.1109/SAFEPROCESS52771.2021.9693539.

[51] R. Gallon et al. “Convolutional neural network design and evaluation for real-time multivariate time series fault detection in spacecraft attitude sensors”. In: *Advances in Space Research* 76.5 (2025), pp. 2960–2976. ISSN: 0273-1177. DOI: 10.1016/j.asr.2025.06.068.

[52] C. Fan et al. “Federated sigma point filter for multi-sensor attitude and rate estimation of spacecraft”. In: *Seventh International Symposium on Instrumentation and Control Technology: Optoelectronic Technology and Instruments, Control Theory and Automation, and Space Exploration*. Ed. by Jiancheng Fang and Zhongyu Wang. Vol. 7129. International Society for Optics and Photonics. SPIE, 2008, p. 712929. DOI: 10.1117/12.807457.

[53] T. Xu. “Research on Federated Kalman Filter for Integrated Navigation System”. In: *2011 Third Pacific-Asia Conference on Circuits, Communications and System (PACCS)*. 2011, pp. 1–4. DOI: 10.1109/PACCS.2011.5990287.

[54] F. Xu, G. Gao, and L. Ni. “A New Adaptive Federated Cubature Kalman Filter Based on Chi-Square Test for SINS/GNSS/SRS/CNS Integration”. In: *Mathematical Problems in Engineering* 2022.1 (2022), p. 7588265. DOI: 10.1155/2022/7588265.

[55] J.R. Wertz. *Spacecraft Attitude Determination and Control*. 1st ed. Dordrecht, The Netherlands: D. Reidel Publishing Company, 1978. ISBN: 978-9027709592.

[56] C. Lee et al. “Sun Tracking Systems: A Review”. In: *Sensors* 9 (May 2009), pp. 3875–3890. DOI: 10.3390/s90503875.

[57] J. Ge et al. “Characterization and Calibration of Measurement Error Associated With Attitude Drift of a Coil Vector Magnetometer”. In: *IEEE Transactions on Instrumentation and Measurement* 71 (2022), pp. 1–9. DOI: 10.1109/TIM.2022.3192855.

[58] *ISIS On board computer*. TJA1043. ISISPACE. 2012.

[59] S. Xiao et al. “A Neural Network-Based Real-time Casing Collar Recognition System for Downhole Instruments”. In: *arXiv preprint arXiv:2512.22901* (2025).

[60] Advisory Group for Aerospace Research and Development. *Fault Tolerant Considerations and Methods for Guidance and Control Systems*. Tech. rep. AGARD-AG-289. Neuilly sur Seine, France: NATO, July 1987.

[61] R. Isermann. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. 1st ed. Berlin, Germany: Springer, 2006. ISBN: 978-3-540-24112-6.

[62] R. Isermann and P. Ballé. “Trends in the application of model-based fault detection and diagnosis of technical processes”. In: *Control Engineering Practice* 5.5 (1997), pp. 709–719. ISSN: 0967-0661. DOI: 10.1016/S0967-0661(97)00053-1.

[63] B. Bittner et al. “An Integrated Process for FDIR Design in Aerospace”. In: *Model-Based Safety and Assessment*. Ed. by F. Ortmeierand A. Rauzy. Cham, Germany: Springer, 2014, pp. 82–95. ISBN: 978-3-319-12214-4.

[64] S.H. Pourtakdoust et al. “Advanced fault detection and diagnosis in spacecraft attitude control systems: Current state and challenges”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 237.12 (2023), pp. 2679–2699. DOI: 10.1177/09544100231157132.

[65] S. Altaf. “Smart Sensor Network Organization: Sensor Data Fusion and Industrial Fault Traceability”. Available: hdl.handle.net/10292/9271. Doctoral thesis. Auckland, New Zealand: Auckland University of Technology, Nov. 2015.

[66] S. Yin, S.X. Ding, and D. Zhou. “Diagnosis and Prognosis for Complicated Industrial Systems—Part II”. In: *IEEE Transactions on Industrial Electronics* 63.5 (2016), pp. 3201–3204. DOI: 10.1109/TIE.2016.2538745.

[67] N. Hasan, S.U. Jan, and I. Koo. “Sensor Fault Detection and Classification Using Multi-Step-Ahead Prediction with an Long Short-Term Memoery (LSTM) Autoencoder”. In: *Applied Sciences* 14.17 (2024). ISSN: 2076-3417. DOI: 10.3390/app14177717.

[68] Daoliang Li et al. “Recent advances in sensor fault diagnosis: A review”. In: *Sensors and Actuators A: Physical* 309 (2020), p. 111990. ISSN: 0924-4247. DOI: 10.1016/j.sna.2020.111990.

[69] Z. Gao, C. Cecati, and S.X. Ding. “A Survey of Fault Diagnosis and Fault-Tolerant Techniques—Part I: Fault Diagnosis With Model-Based and Signal-Based Approaches”. In: *IEEE Transactions on Industrial Electronics* 62.6 (2015), pp. 3757–3767. DOI: 10.1109/TIE.2015.2417501.

[70] A. Wander and R. Förstner. “Innovative Fault Detection, Isolation and Recovery Strategies On-Board Spacecraft: State of the Art and Research Challenges”. In: *Deutscher Luft- und Raumfahrtkongress*. 2013. URL: api.semanticscholar.org/CorpusID:44234357.

[71] J. Mansell. "Deep Learning Fault Protection Applied to Spacecraft Attitude Determination and Control". In: (July 2020). DOI: 10.25394/PGS.12739202.v1.

[72] R.V. Beard. "Failure accomodation in linear systems through self-reorganization". Available: hdl.handle.net/1721.1/16415. Doctoral thesis. Boston, United States: Massachusetts Institute of Technology, Feb. 1971.

[73] B.O. Bouamama A.K. Samantaray. *Model-based Process Supervision: A Bond Graph Approach*. 1st ed. London, United Kingdom: Springer, 2008. ISBN: 978-1-84800-158-9.

[74] W.S. Lee et al. "Fault Tree Analysis, Methods, and Applications: A Review". In: *IEEE Transactions on Reliability* R-34.3 (1985), pp. 194–203. DOI: 10.1109/TR.1985.5222114.

[75] L. He et al. "Developments of attitude determination and control system of microsats: A survey". In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 235.10 (2021), pp. 1733–1750. DOI: 10.1177/0959651819895173.

[76] A. Fekih. "Fault diagnosis and Fault Tolerant Control design for aerospace systems: A bibliographical review". In: *American Control Conference*. 2014, pp. 1286–1291. DOI: 10.1109/ACC.2014.6859271.

[77] Z. Gao, C. Cecati, and S.X. Ding. "A Survey of Fault Diagnosis and Fault-Tolerant Techniques—Part II: Fault Diagnosis With Knowledge-Based and Hybrid/Active Approaches". In: *IEEE Transactions on Industrial Electronics* 62.6 (2015), pp. 3768–3774. DOI: 10.1109/TIE.2015.2419013.

[78] P. Boniol et al. "Dive into time-series anomaly detection: A decade review". In: (2024).

[79] H. Alashwal, S. bin deris, and R. Othman. "One-Class Support Vector Machines for Protein Protein Interactions Prediction". In: *International Journal of Biomedical Science* 1 (Jan. 2006).

[80] M. Goldstein and S. Uchida. "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data". In: *PLOS ONE* 11.4 (Apr. 2016), pp. 1–31. DOI: 10.1371/journal.pone.0152173.

[81] E.F. Agyemang. "Anomaly detection using unsupervised machine learning algorithms: A simulation study". In: *Scientific African* 26 (2024), e02386. ISSN: 2468-2276. DOI: 10.1016/j.sciaf.2024.e02386.

[82] H.Hassan et al. "Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: case study El Burullus Lake". In: *International Water Technology Journal* 5 (Dec. 2015).

[83] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[84] K.P. Tran, H.D. Nguyen, and S. Thomassey. "Anomaly detection using Long Short Term Memory Networks and its applications in Supply Chain Management". In: *IFAC-PapersOnLine* 52.13 (2019). 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019, pp. 2408–2412. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2019.11.567.

[85] M. Hermans and B. Schrauwen. "Training and analyzing deep recurrent neural networks". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 190–198.

[86] Z.Z. Darban et al. "Deep Learning for Time Series Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 57.1 (Oct. 2024). ISSN: 0360-0300. DOI: 10.1145/3691338.

[87] J. Do, A.B. Kareem, and J. Hur. "LSTM-Autoencoder for Vibration Anomaly Detection in Vertical Carousel Storage and Retrieval System (VCSRS)". In: *Sensors* 23 (Jan. 2023), p. 1009. DOI: 10.3390/s23021009.

[88] J. Hartikainen, A. Solin, and S. Särkkä. "Optimal Filtering with Kalman Filters and Smoothers". In: 2011. URL: api.semanticscholar.org/CorpusID:61187405.

[89] E.A. Wan and R. Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*. 2000, pp. 153–158. DOI: 10.1109/ASSPCC.2000.882463.

[90] J.L. Crassidis and F.L. Markley. “Unscented Filtering for Spacecraft Attitude Estimation”. In: *Journal of Guidance, Control, and Dynamics* 26.4 (2003), pp. 536–542. DOI: 10.2514/2.5102.

[91] H.A. Hassan et al. “A Comparative Analysis Between the Additive and the Multiplicative Extended Kalman Filter for Satellite Attitude Determination”. In: (2023).

[92] F.L. Markley. “Attitude Estimation or Quaternion Estimation?” In: *Journal of the Astronautical Sciences* 52 (2004), pp. 221–238. DOI: 10.1007/BF03546430.

[93] N. Tatbul et al. “Precision and Recall for Time Series”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018.

[94] D. Chicco and G. Jurman. “The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification”. In: *BioData Mining* 16.1 (Feb. 2023), p. 4. ISSN: 1756-0381. DOI: 10.1186/s13040-023-00322-4.

[95] L.M. McNeil and T.S. Kelso. *Spatial temporal information systems: an ontological approach using STK®*. Taylor and Francis, 2013. ISBN: 9781466500495.

[96] The CubeSat Program. *CubeSat Design Specification (rev. 14.1)*. Tech. rep. 2022.

[97] C. Karlgaard and H. Schaub. “Adaptive Huber-Based Filtering Using Projection Statistics: Application to Spacecraft Attitude Estimation”. In: *AIAA Guidance, Navigation and Control Conference and Exhibit*. 2012. DOI: 10.2514/6.2008-7389.

[98] D.A. Vallado. *Fundamentals of Astrodynamics and Applications*. 4th ed. Hawthorne, United States: Microcosm Press, 2013. ISBN: 978-1881883203.

[99] Y. Yang. *Spacecraft Modeling, Attitude Determination, and Control: Quaternion-Based Approach*. 1st ed. Boca Raton, United States: CRC Press, 2019. ISBN: 978-1-138-33150-1.

[100] Y. Zhang et al. “Unsupervised Deep Anomaly Detection for Multi-Sensor Time-Series Signals”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.2 (2023), pp. 2118–2132. DOI: 10.1109/TKDE.2021.3102110.

[101] J. Zeng, B. Cao, and R. Tian. “Quality Monitoring for Micro Resistance Spot Welding with Class-Imbalanced Data Based on Anomaly Detection”. In: *Applied Sciences* 10.12 (2020). ISSN: 2076-3417. DOI: 10.3390/app10124204.

[102] M.J.C.S. Reis and C. Serôdio. “Edge AI for Real-Time Anomaly Detection in Smart Homes”. In: *Future Internet* 17.4 (2025). ISSN: 1999-5903. DOI: 10.3390/fi17040179.

[103] D. Singh and B. Singh. “Investigating the impact of data normalization on classification performance”. In: *Applied Soft Computing* 97 (2020), p. 105524. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2019.105524.

[104] L.N. Smith. “Cyclical Learning Rates for Training Neural Networks”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2017, pp. 464–472. DOI: 10.1109/WACV.2017.58.

[105] G.E. Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *CoRR* abs/1207.0580 (2012). DOI: 10.48550/arXiv.1207.0580.

[106] R. Sebastian. *An overview of gradient descent optimization algorithms*. 2017. DOI: 10.48550/arXiv.1609.04747. arXiv: 1609.04747 [cs.LG].

[107] F. Lopez, L. Karlsson, and P. Bientinesi. “FLOPs as a Discriminant for Dense Linear Algebra Algorithms”. In: *Proceedings of the 51st International Conference on Parallel Processing*. ICPP ’22. Bordeaux, France: Association for Computing Machinery, 2023. ISBN: 9781450397339. DOI: 10.1145/3545008.3545072.

[108] R. Hunger. *Floating point operations in matrix-vector calculus.* Tech. rep. Associate Institute for Signal Processing, 2005.

[109] N.H.F. Beebe. “Accurate hyperbolic tangent computation”. In: (May 1993).

[110] K. Alizadeh et al. “LLM in a flash: Efficient large language model inference with limited memory”. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 12562–12584.

[111] R. Horne et al. “Anomaly Detection Using Deep Learning Respecting the Resources on Board a CubeSat”. In: *Journal of Aerospace Information Systems* 20.12 (2023), pp. 859–872. DOI: 10.2514/1.I011232.

[112] B.N. Stovner et al. “Attitude estimation by multiplicative exogenous Kalman filter”. In: *Automatica* 95 (2018), pp. 347–355. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2018.05.038.

[113] R. Zanetti and K.J. DeMars. “Joseph Formulation of Unscented and Quadrature Filters with Application to Consider States”. In: *Journal of Guidance, Control, and Dynamics* 36.6 (2013), pp. 1860–1864. DOI: 10.2514/1.59935.

A
Unscented Quaternion Estimator

Quaternions are an appealing attitude representation due to the lack of singularities. For both the extended Kalman filter (EKF) and unscented Kalman filter (UKF), a direct implementation using quaternion representation leads to an issue: there is no guarantee the estimated quaternion adheres to the unity norm constraint [90]. In the EKF this is due to the linear measurement updates, while for the UKF this is caused during propagation, when an averaged sum of quaternions would be used. In [90], an alternative approach is developed which avoids this problem by using a three-component vector to represent the quaternion error vector internally. This reintroduces possible singularities, between 180 and 360 degrees, but these are not encountered in practice since the vector only represents an attitude error, not an absolute attitude.

The algorithm proposed in [90] is called the unscented quaternion estimator (USQUE). The attitude error vector is used during propagation to avoid the unity issue, while the updates are still performed using quaternion multiplication. In the internal state vector $\hat{\boldsymbol{x}}$, the estimated attitude error is represented using a vector of generalized Rodrigues parameters (GRP) $\boldsymbol{\delta\hat{p}}$:

$$\hat{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{\delta\hat{p}} \\ \hat{\boldsymbol{\beta}} \end{bmatrix} \tag{A.1}$$

where $\hat{\boldsymbol{\beta}}$ is the estimated gyroscope bias vector. The algorithm contains parameters to allow for manual placement of the singularity between 180 and 360 degrees, by adjusting the representation of the GRP vector.

The algorithm as described in [90] uses a quaternion definition with the vector part first and the scalar part after ($\boldsymbol{q} = [\boldsymbol{\varrho} \quad q_4]$). However, to adhere to the standard used for this thesis, the algorithm was slightly adjusted to use the opposite definition ($\boldsymbol{q} = [q_1 \quad \boldsymbol{\varrho}]$), which introduces some subtle differences.

In the following sections, the USQUE algorithm will be described. First, an overview of the notation and common functions is given in section A.1. This is followed by the initialization of the filter in section A.2. Finally, the two main steps of the algorithm are given: the propagation step (section A.3) and the measurement update step (section A.4).

A.1 | Notation

In the following description of the USQUE algorithm, specific notations are used throughout the equations. This section gives a brief summary of these notations.

Several subscripts and superscripts are used to denote types of variables. These are listed below. Note that the symbol $\square$ is a placeholder for any other symbol or combination of symbols.

$\hat{\square}$	estimate
$\tilde{\square}$	measured
$\square^-$	pre-update value
$\square^+$	post-update value
$\square_k$	value at time k

A quaternion $\boldsymbol{q}$ consists of a scalar part q_1 and a vector part $\boldsymbol{\varrho}$:

$$\boldsymbol{q} = \begin{bmatrix} q_1 & \boldsymbol{\varrho}^T \end{bmatrix}^T \tag{A.2}$$

with $q_1 = \cos(\vartheta/2)$ and $\boldsymbol{\varrho} = \begin{bmatrix} q_2 & q_3 & q_4 \end{bmatrix} = \hat{\boldsymbol{e}}\sin(\vartheta/2)$, where $\hat{\boldsymbol{e}}$ is the axis of rotation and ϑ is the angle of rotation. Note that the original derivation in [90] follows the opposite standard, placing the scalar part behind the vector part. This introduces some subtle differences in the description of USQUE.

The attitude matrix of a quaternion $A(\boldsymbol{q})$ is defined using the two matrices $\Xi(\boldsymbol{q})$ and $\Psi(\boldsymbol{q})$:

$$A(\boldsymbol{q}) = \Xi^T(\boldsymbol{q})\Psi(\boldsymbol{q}) \tag{A.3}$$

$$\Xi(\boldsymbol{q}) = \begin{bmatrix} -\boldsymbol{\varrho}^T \\ q_1 I_{3\times3} + [\boldsymbol{\varrho}\times] \end{bmatrix} \tag{A.4}$$

$$\Psi(\boldsymbol{q}) = \begin{bmatrix} -\boldsymbol{\varrho}^T \\ q_1 I_{3\times3} - [\boldsymbol{\varrho}\times] \end{bmatrix} \tag{A.5}$$

$$[\boldsymbol{a}\times] = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{A.6}$$

where $I_{3\times3}$ is a 3×3 identity matrix, and $[\boldsymbol{a}\times]$ is the cross product matrix of vector $\boldsymbol{a}$. Successive rotation of quaternions $\boldsymbol{q}'$ and $\boldsymbol{q}$ can then be accomplished with quaternion multiplication ($\otimes$):

$$\boldsymbol{q}' \otimes \boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}' & \vdots & \Psi(\boldsymbol{q}') \end{bmatrix} \boldsymbol{q} = \begin{bmatrix} \boldsymbol{q} & \vdots & \Xi(\boldsymbol{q}) \end{bmatrix} \boldsymbol{q}' \tag{A.7}$$

Finally, the conjugate $\boldsymbol{q}^*$ of a quaternion $\boldsymbol{q}$ is given by:

$$\boldsymbol{q}^* = \begin{bmatrix} q_1 & -\boldsymbol{\varrho}^T \end{bmatrix}^T \tag{A.8}$$

During the propagation, the GRP representation $\boldsymbol{\delta p}$ of the local error-quaternion $\boldsymbol{\delta q} = \begin{bmatrix} \delta q_1 & \boldsymbol{\delta\varrho}^T \end{bmatrix}^T$ is used:

$$\boldsymbol{\delta p} = f\frac{\boldsymbol{\delta\varrho}}{a + \delta q_1} \tag{A.9}$$

where a is a parameter in the range $[0, 1]$, and f is a scale factor. Using $a = 0$ and $f = 1$ makes $\boldsymbol{\delta p}$ equivalent to the Gibbs vector, while $a = f = 1$ gives the standard vector of modified Rodrigues parameters (MRP). In [90] it is proposed to take $f = 2(a + 1)$ such that $\|\boldsymbol{\delta p}\|$ is equal to ϑ for small errors.

The opposite transformation, from $\boldsymbol{\delta p}$ to $\boldsymbol{\delta q}$, is given by:

$$\delta q_1 = \frac{-a\,\|\boldsymbol{\delta p}\|^2 + f\sqrt{f^2 + (1-a^2)\,\|\boldsymbol{\delta p}\|^2}}{f^2 + \|\boldsymbol{\delta p}\|^2} \tag{A.10a}$$

$$\boldsymbol{\delta\varrho} = f^{-1}\,(a + \delta q_1)\,\boldsymbol{\delta p} \tag{A.10b}$$

A.2 | Initialization

The USQUE algorithm starts by selecting an initial attitude quaternion estimate $\hat{\boldsymbol{q}}_0^+$ and gyroscope bias $\hat{\boldsymbol{\beta}}_0^+$. The 6×6 initial covariance matrix P_0^+ is also set up, where the upper-left 3×3 block P_{att} corresponds to the attitude error angles, and the lower-right 3×3 block P_{bias} corresponds to the bias:

$$P_0^+ = \begin{bmatrix} P_{\text{att}} & 0_{3\times3} \\ 0_{3\times3} & P_{\text{bias}} \end{bmatrix} \tag{A.11}$$

where $0_{3\times3}$ is a 3×3 zero matrix. The initial state vector $\hat{\boldsymbol{x}}_0^+$ is set such that the initial attitude error is zero:

$$\hat{\boldsymbol{x}}_0^+ = \begin{bmatrix} 0 \\ \hat{\boldsymbol{\beta}}_0^+ \end{bmatrix} \tag{A.12}$$

The process noise covariance $\bar{Q}_k$ can then be derived based on the time step size Δt, as well as σ_ω^2 and σ_β^2 which are variances associated with the gyroscope model, such as is described in subsection 6.2.1. In the USQUE algorithm, a trapezoidal approximation of the process noise covariance $\bar{Q}_k$ is used, given by:

$$\bar{Q}_k = \frac{\Delta t}{2} \begin{bmatrix} \left(\sigma_\omega^2 + \frac{1}{6}\sigma_\beta^2 \Delta t^2\right) I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & \sigma_\beta^2 I_{3\times3} \end{bmatrix} \tag{A.13}$$

Finally, the measurement covariance matrix R_k is created, assumed to be a diagonal matrix:

$$R_k = \text{diag}\left(\sigma_1^2 \quad \sigma_2^2 \quad \dots \sigma_N^2\right) \tag{A.14}$$

where diag$()$ is a diagonal matrix with the diagonal entries σ_i as the standard deviation of the i-th vector measurement, such as from a star tracker or Sun sensor.

A.3 | Propagation

For propagation in the UKF, a set of sigma points is generated. These are points distributed around the current estimate using information from the current covariance. In USQUE, the distribution of these points is represented as a matrix with $2n$ columns, σ_k, based on the current covariance P_k^+:

$$\sigma_k = \left[+\sqrt{(n+\lambda)\left(P_k^+ + \bar{Q}_k\right)} \quad \vdots \quad -\sqrt{(n+\lambda)\left(P_k^+ + \bar{Q}_k\right)}\right] \tag{A.15}$$

where n is the number of elements in the state, in this case $n = 6$, and λ is a scaling factor that defines the spread of the sigma points, typically kept at $\lambda = 1$. An efficient method to calculate the matrix square root is through Cholesky decomposition. The sigma points are then defined as:

$$\boldsymbol{\chi}_k(i) = \begin{bmatrix} \boldsymbol{\chi}_k^{\delta p}(i) \\ \boldsymbol{\chi}_k^{\beta}(i) \end{bmatrix} \tag{A.16a}$$

$$\boldsymbol{\chi}_k(0) = \hat{\boldsymbol{x}}_k^+ \tag{A.16b}$$

$$\boldsymbol{\chi}_k(i) = \hat{\boldsymbol{x}}_k^+ + \sigma_k(i) \qquad \text{for } i = 1, 2, \dots, 12 \tag{A.16c}$$

There are $2n + 1$ sigma points, where the 0-th point is simply the current estimate $\hat{\boldsymbol{x}}_k^+$, and the i-th point is the same estimate but with an offset defined by σ_k. Each sigma point contains a part for the GRP error vector $\boldsymbol{\chi}_k^{\delta p}$, and a part for the bias $\boldsymbol{\chi}_k^{\beta}$.

For each of the sigma points, the corresponding error quaternion $\boldsymbol{\delta q}_k^+(i) = \left[\delta q_{1_k}^+(i) \quad \boldsymbol{\delta \varrho}_k^+(i)\right]$ can be calculated following the transformation described by Equation A.10:

$$\delta q_{1_k}^+(i) = \frac{-a\left\|\boldsymbol{\chi}_k^{\delta p}(i)\right\|^2 + f\sqrt{f^2 + (1-a^2)\left\|\boldsymbol{\chi}_k^{\delta p}(i)\right\|^2}}{f^2 + \left\|\boldsymbol{\chi}_k^{\delta p}(i)\right\|^2} \qquad \text{for } i = 1, 2, \dots, 12 \tag{A.17a}$$

$$\boldsymbol{\delta \varrho}_k^+(i) = f^{-1}\left(a + \delta q_{1_k}^+(i)\right)\boldsymbol{\chi}_k^{\delta p}(i) \qquad \text{for } i = 1, 2, \dots, 12 \tag{A.17b}$$

Note that this calculation is not necessary for the 0-th point, since it is not used in the next step.

For each sigma point, the calculated error quaternion can be used to compute the full quaternion $\hat{\boldsymbol{q}}_k^+(i)$ for that point:

$$\hat{\boldsymbol{q}}_k^+(0) = \hat{\boldsymbol{q}}_k^+ \tag{A.18a}$$

$$\hat{\boldsymbol{q}}_k^+(i) = \boldsymbol{\delta q}_k^+(i) \otimes \hat{\boldsymbol{q}}_k^+ \qquad \text{for } i = 1, 2, \dots, 12 \tag{A.18b}$$

Note that, since the 0-th sigma point is based on the current state estimate, also the 0-th quaternion is the current quaternion estimate.

Next, these quaternions corresponding to the sigma points can be propagated to the pre-update value $\hat{\boldsymbol{q}}_{k+1}^-$. The discrete-time equivalent of the relationship $\dot{\boldsymbol{q}}(t) = 1/2\,\Xi\left[\boldsymbol{q}(t)\right]\boldsymbol{\omega}(t)$ is used:

$$\hat{\boldsymbol{q}}_{k+1}^-(i) = \Omega\left(\hat{\boldsymbol{\omega}}_k^+(i)\right)\hat{\boldsymbol{q}}_k^+(i) \qquad \text{for } i = 0, 1, \dots, 12 \tag{A.19a}$$

$$\Omega(\hat{\boldsymbol{\omega}}_k^+) = \begin{bmatrix} \cos\left(\frac{1}{2}\left\|\hat{\boldsymbol{\omega}}_k^+\right\|\Delta t\right) & -\hat{\boldsymbol{\psi}}_k^{+\,T} \\ \hat{\boldsymbol{\psi}}_k^+ & \cos\left(\frac{1}{2}\left\|\hat{\boldsymbol{\omega}}_k^+\right\|\Delta t\right) I_{3\times3} - \left[\hat{\boldsymbol{\psi}}_k^+\times\right] \end{bmatrix} \tag{A.19b}$$

$$\hat{\boldsymbol{\psi}}_k^+ = \sin\left(\frac{1}{2}\left\|\hat{\boldsymbol{\omega}}_k^+\right\|\Delta t\right)\frac{\hat{\boldsymbol{\omega}}_k^+}{\left\|\hat{\boldsymbol{\omega}}_k^+\right\|} \tag{A.19c}$$

The matrix $\Omega(\hat{\boldsymbol{\omega}}_k^+)$ is a 4×4 matrix that incorporates the estimated angular velocity of the spacecraft $\hat{\boldsymbol{\omega}}_k^+$. This estimate uses the current measured angular velocity $\tilde{\omega}_k$, and subtracts the estimated bias from the i-th gamma point: $\hat{\boldsymbol{\omega}}_k^+(i) = \tilde{\boldsymbol{\omega}}_k - \boldsymbol{\chi}_k^\beta(i)$. A quaternion can be multiplied with the matrix to propagate it using this angular velocity.

Error quaternions $\boldsymbol{\delta q}_{k+1}^-(i) = \begin{bmatrix}\delta q_{1_{k+1}}^-(i) & \boldsymbol{\delta\varrho}_{k+1}^-(i)\end{bmatrix}$ that correspond to the propagated quaternions $\hat{\boldsymbol{q}}_{k+1}^-(i)$ can be calculated for each sigma point:

$$\boldsymbol{\delta q}_{k+1}^-(i) = \hat{\boldsymbol{q}}_{k+1}^-(i) \otimes \left(\hat{\boldsymbol{q}}_{k+1}^-(0)\right)^{-1} = \hat{\boldsymbol{q}}_{k+1}^-(i) \otimes \left(\hat{\boldsymbol{q}}_{k+1}^-(0)\right)^* \qquad \text{for } i = 0, 1, \dots, 12 \tag{A.20}$$

Here, the 0-th propagated quaternion $\hat{\boldsymbol{q}}_{k+1}^-(0)$ is taken as the basis from which the error is calculated. This makes $\boldsymbol{\delta q}_{k+1}^-(0)$ an identity quaternion. Note that the quaternion reciprocal (q^{-1}) is used, but for a unit quaternion, this is equal to the conjugate (q^*). Thus, only if the quaternion is a unit quaternion does the above relationship hold.

Following the procedure described by Equation A.9, the propagated sigma points χ_{k+1} can be found by transforming the error quaternions:

$$\boldsymbol{\chi}_{k+1}^{\delta p}(0) = 0 \tag{A.21a}$$

$$\boldsymbol{\chi}_{k+1}^{\delta p}(i) = f\frac{\boldsymbol{\delta\varrho}_{k+1}^-(i)}{a + \delta q_{1_{k+1}}^-(i)} \qquad \text{for } i = 1, 2, \dots, 12 \tag{A.21b}$$

$$\boldsymbol{\chi}_{k+1}^{\beta}(i) = \boldsymbol{\chi}_k^\beta(i) \qquad \text{for } i = 0, 1, \dots, 12 \tag{A.21c}$$

Note that the gyro bias is assumed to be constant, which is valid for small timesteps.

From the propagated points, the predicted mean $\hat{\boldsymbol{x}}_{k+1}^-$ and predicted covariance P_{k+1}^- can be computed:

$$\hat{\boldsymbol{x}}_{k+1}^- = \frac{1}{n+\lambda}\left(\lambda\boldsymbol{\chi}_{k+1}(0) + \frac{1}{2}\sum_{i=1}^{2n}\boldsymbol{\chi}_{k+1}(i)\right) \tag{A.22}$$

$$\begin{aligned} P_{k+1}^- = \frac{1}{n+\lambda}\Bigg(&\lambda\left[\boldsymbol{\chi}_{k+1}(0) - \hat{\boldsymbol{x}}_{k+1}^-\right]\left[\boldsymbol{\chi}_{k+1}(0) - \hat{\boldsymbol{x}}_{k+1}^-\right]^T \\ &+ \frac{1}{2}\sum_{i=1}^{2n}\left[\boldsymbol{\chi}_{k+1}(i) - \hat{\boldsymbol{x}}_{k+1}^-\right]\left[\boldsymbol{\chi}_{k+1}(i) - \hat{\boldsymbol{x}}_{k+1}^-\right]^T\Bigg) + \bar{Q}_k \end{aligned} \tag{A.23}$$

where $\bar{Q}_k$ is the process noise covariance, given by Equation A.13. With a predicted mean and covariance, the propagation step is concluded. An overview of the process is provided in Figure A.1.

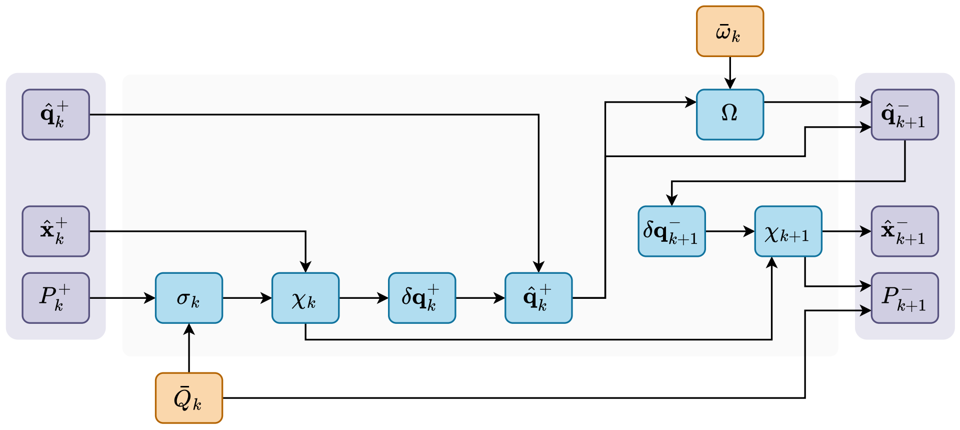


Figure A.1: Propagation step of the USQUE algorithm

A.4 | Measurement Update

After propagating the mean and covariance using the gyroscope measurement, the predicted values are updated using quaternion measurements. These can come from a variety of sensors, such as star trackers, sun sensors, or magnetometers.

The propagated quaternions, calculated in Equation A.19a, are used to calculate $2n+1$ gamma points $\gamma_{k+1}(i)$. Each gamma point represents the expected measurement for the corresponding sigma point. Assuming the measurements are provided as quaternions, the expected quaternion would be the propagated quaternion $\hat{\boldsymbol{q}}_{k+1}^-(i)$. The i-th gamma point is then a column vector of N quaternions, where N is the number of measurements:

$$\gamma_{k+1}(i) = \begin{bmatrix} \hat{\boldsymbol{q}}_{k+1}^-(i) \\ \vdots \\ \hat{\boldsymbol{q}}_{k+1}^-(i) \end{bmatrix}_{\text{size } N} \qquad \text{for } i = 0, 1, \dots, 12 \tag{A.24}$$

In the case that measurements are provided as vectors, they can be rotated using the quaternion attitude matrix, $A(\hat{\boldsymbol{q}}_{\boldsymbol{k}+1}^-(\boldsymbol{i}))$, as derived in Equation A.3.

The mean of the observations $\hat{\boldsymbol{y}}_{k+1}^-$ and the output covariance P_{k+1}^{yy} can be calculated from these gamma points:

$$\hat{\boldsymbol{y}}_{k+1}^- = \frac{1}{n+\lambda}\left(\lambda\gamma_{k+1}(0) + \frac{1}{2}\sum_{i=1}^{2n}\gamma_{k+1}(i)\right) \tag{A.25}$$

$$\begin{aligned} P_{k+1}^{yy} = \frac{1}{n+\lambda}\Bigg(&\lambda[\gamma_{k+1}(0) - \hat{\boldsymbol{y}}_{k+1}^-][\gamma_{k+1}(0) - \hat{\boldsymbol{y}}_{k+1}^-]^T \\ &+ \frac{1}{2}\sum_{i=1}^{2n}[\gamma_{k+1}(i) - \hat{\boldsymbol{y}}_{k+1}^-][\gamma_{k+1}(i) - \hat{\boldsymbol{y}}_{k+1}^-]^T\Bigg) \end{aligned} \tag{A.26}$$

With the mean of observations $\hat{\boldsymbol{y}}_{k+1}^-$, the innovation $\boldsymbol{v}_{k+1}$ can be computed:

$$\boldsymbol{v}_{k+1} = \tilde{\boldsymbol{y}}_{k+1} - \hat{\boldsymbol{y}}_{k+1}^- \tag{A.27}$$

where $\tilde{\boldsymbol{y}}_{k+1}$ is the actual measurement, a column vector of the N measured quaternions.

With the output covariance P_{k+1}^{yy}, two more matrices can be computed: the innovation covariance P_{k+1}^{vv} and the cross-correlation matrix P_{k+1}^{xy}. They are calculated as:

$$P_{k+1}^{vv} = P_{k+1}^{yy} + R_{k+1} \tag{A.28}$$

$$P^{xy}_{k+1} = \frac{1}{n+\lambda}\left(\lambda[\boldsymbol{\chi}_{k+1}(0) - \hat{\boldsymbol{x}}^-_{k+1}][\boldsymbol{\gamma}_{k+1}(0) - \hat{\boldsymbol{y}}^-_{k+1}]^T \right.$$
$$\left. + \frac{1}{2}\sum_{i=1}^{2n}[\boldsymbol{\chi}_{k+1}(i) - \hat{\boldsymbol{x}}^-_{k+1}][\boldsymbol{\gamma}_{k+1}(i) - \hat{\boldsymbol{y}}^-_{k+1}]^T\right) \tag{A.29}$$

where R_{k+1} is the measurement covariance matrix, given by Equation A.14. With these matrices, the Kalman filter gain K_{k+1} can be calculated:

$$K_{k+1} = P^{xy}_{k+1}(P^{vv}_{k+1})^{-1} \tag{A.30}$$

This gain can now be used to update the predicted mean and covariance, from $\hat{\boldsymbol{x}}^-_{k+1}$ and P^-_{k+1}, to $\hat{\boldsymbol{x}}^+_{k+1}$ and P^+_{k+1}:

$$\hat{\boldsymbol{x}}^+_{k+1} = \hat{\boldsymbol{x}}^-_{k+1} + K_{k+1}\boldsymbol{v}_{k+1} \tag{A.31}$$
$$P^+_{k+1} = P^-_{k+1} - K_{k+1}P^{vv}_{k+1}K^T_{k+1} \tag{A.32}$$

The updated state vector $\hat{\boldsymbol{x}}^+_{k+1} = \begin{bmatrix}\boldsymbol{\delta}\hat{\boldsymbol{p}}^{+T}_{k+1} & \hat{\boldsymbol{\beta}}^{+T}_{k+1}\end{bmatrix}^T$ contains the updated GRP error vector $\boldsymbol{\delta}\hat{\boldsymbol{p}}^+_{k+1}$, which can be transformed to an error quaternion $\boldsymbol{\delta q}^+_{k+1} = \begin{bmatrix}\delta q^+_{1_{k+1}} & \boldsymbol{\delta\varrho}^{+T}_{k+1}\end{bmatrix}^T$ following Equation A.10:

$$\delta q^+_{1_{k+1}} = \frac{-a\left\|\boldsymbol{\delta}\hat{\boldsymbol{p}}^+_{k+1}\right\|^2 + f\sqrt{f^2 + (1-a^2)\left\|\boldsymbol{\delta}\hat{\boldsymbol{p}}^+_{k+1}\right\|^2}}{f^2 + \left\|\boldsymbol{\delta}\hat{\boldsymbol{p}}^+_{k+1}\right\|^2} \tag{A.33a}$$
$$\boldsymbol{\delta\varrho}^+_{k+1} = f^{-1}\left(a + \delta q^+_{1_{k+1}}\right)\boldsymbol{\delta}\hat{\boldsymbol{p}}^+_{k+1} \tag{A.33b}$$

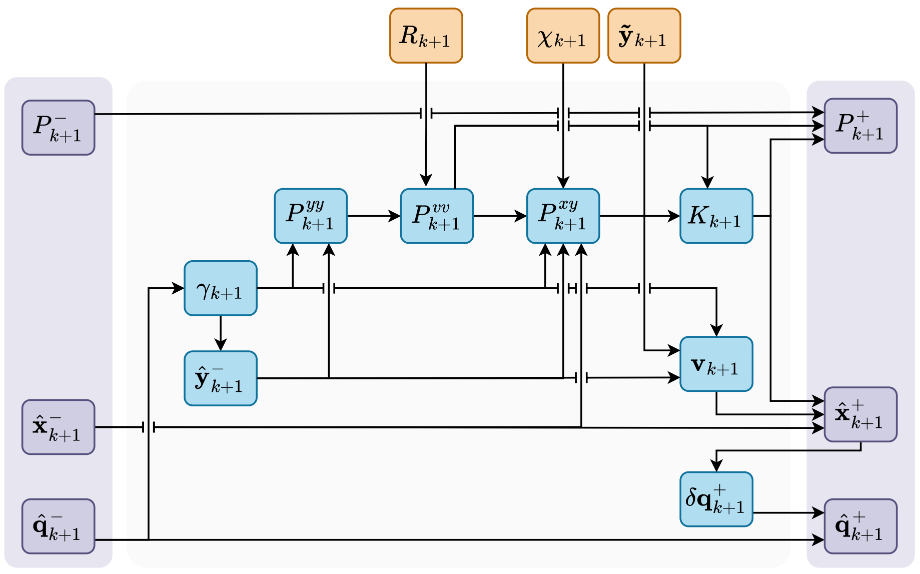
This error quaternion can finally be used to update the quaternion estimate to $\hat{\boldsymbol{q}}^+_{k+1}$:

$$\hat{\boldsymbol{q}}^+_{k+1} = \boldsymbol{\delta q}^+_{k+1} \otimes \hat{\boldsymbol{q}}^-_{k+1}(0) \tag{A.34}$$

With this updated quaternion, the measurement update step is completed. For the next propagation, the state is adjusted to set the GRP attitude error $\boldsymbol{\delta}\hat{\boldsymbol{p}}^+_{k+1}$ to zero:

$$\hat{\boldsymbol{x}}^+_{k+1} = \begin{bmatrix}0 \\ \hat{\boldsymbol{\beta}}^+_{k+1}\end{bmatrix} \tag{A.35}$$

An overview of the measurement update step is provided in Figure A.2.



R_{k+1}
χ_{k+1}
$\tilde{\mathbf{y}}_{k+1}$
P^-_{k+1}
P^+_{k+1}
P^{yy}_{k+1}
P^{vv}_{k+1}
P^{xy}_{k+1}
K_{k+1}
γ_{k+1}
$\hat{\mathbf{y}}^-_{k+1}$
$\mathbf{v}_{k+1}$
$\hat{\mathbf{x}}^-_{k+1}$
$\hat{\mathbf{x}}^+_{k+1}$
$\delta\mathbf{q}^+_{k+1}$
$\hat{\mathbf{q}}^-_{k+1}$
$\hat{\mathbf{q}}^+_{k+1}$

Figure A.2: Measurement update step of the USQUE algorithm

B

Quaternion Extended Kalman Filter

The extended Kalman filter (EKF) is a popular estimator for spacecraft attitude determination [75]. The quaternion is a desirable attitude representation due to the lack of singularities, with the lowest possible number of parameters to achieve this [92]. Several strategies exist to incorporate the use of quaternions into an EKF. In this appendix, the common variations additive extended Kalman filter (AEKF) and multiplicative extended Kalman filter (MEKF) are discussed in section B.1 and B.2 respectively.

The AEKF treats the four components of the quaternion as independent parameters and estimates them directly [92]. The MEKF estimates a three-component deviation from a reference quaternion instead. The AEKF corresponds to a standard EKF implementation, directly estimating the quaternion. This strategy is prone to numerical issues such as ill-conditioned covariance matrices, though there are methods to ensure stability [92]. The MEKF output is guaranteed to be a unit quaternion and in general has lower computation time due to estimating three parameters instead of four. For these reasons, among others, the MEKF generally considered superior [91].

In section A.1, the general notation for the equations used in this appendix can be found, including the definitions for a quaternion.

B.1 | Additive Filter

In this section, the AEKF algorithm is described. This algorithm treats the quaternion elements as independent parameters, which are estimated directly. The term additive refers to the method of updating the state estimate, which is done through addition in the AEKF. This inherently violates the unity norm constraint, in contrast to the multiplicative variant. However, with the correct accommodations the two filters have been shown to be identical, though the AEKF requires more computations [112].

Initialization

The state vector for the AEKF contains an estimate of the quaternion $\hat{\boldsymbol{q}}$ directly, as well as the gyroscope bias $\hat{\boldsymbol{\beta}}$:

$$\hat{\boldsymbol{x}} = \begin{bmatrix} \hat{\boldsymbol{q}} \\ \hat{\boldsymbol{\beta}} \end{bmatrix} \tag{B.1}$$

An initial quaternion $\hat{\boldsymbol{q}}_0^+$ and initial bias $\hat{\boldsymbol{\beta}}_0^+$ are selected. The initial covariance matrix P_0^+ is also chosen, which will be a 7×7 matrix with the upper-left 4×4 block corresponding to the quaternion and the lower-right 3×3 block corresponding to the bias.

Deriving the Jacobian of the state with respect to the angular velocity results in matrix W. The standard deviations σ_ω and σ_β corresponding to the gyroscope model in subsection 6.2.1 are then used to find the process covariance Q:

$$W = \frac{\Delta t}{2} \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix} \tag{B.2}$$

$$Q = \begin{bmatrix} W(\sigma_\omega^2\, I_{4\times4})W^T & 0_{4\times3} \\ 0_{3\times4} & \sigma_\beta^2\, I_{3\times3} \end{bmatrix} \tag{B.3}$$

The measurement covariance matrix R is assumed to be a diagonal matrix for a quaternion measurement:

$$R = \sigma_q^2\, I_{4\times4} \tag{B.4}$$

The matrix can be extended into a $4N \times 4N$ diagonal block matrix for N measurements.

Propagation

In the first step of the AEKF cycle, a prediction is made for the quaternion and bias vector. This is based on a gyroscope measurement $\tilde{\boldsymbol{\omega}}_k$, which is corrected using the current bias estimate into the estimated true angular rate $\hat{\boldsymbol{\omega}}_k^+$:

$$\hat{\boldsymbol{\omega}}_k^+ = \tilde{\boldsymbol{\omega}}_k - \hat{\boldsymbol{\beta}}_k^+ \tag{B.5}$$

The quaternion is then propagated using the propagation matrix $\Omega(\hat{\boldsymbol{\omega}}_k^+)$ derived in Equation A.19b:

$$\hat{\boldsymbol{q}}_{k+1}^- = \Omega(\hat{\boldsymbol{\omega}}_k^+)\hat{\boldsymbol{q}}_k^+ \tag{B.6}$$

The bias is assumed to be constant:

$$\hat{\boldsymbol{\beta}}_{k+1}^- = \hat{\boldsymbol{\beta}}_k^+ \tag{B.7}$$

Finally, the covariance matrix is predicted using the state transition matrix F:

$$F = \left[\begin{array}{cc|ccc} 1 & \frac{\Delta t}{2}\hat{\boldsymbol{\omega}}_k^+ & \frac{\Delta t}{2}q_x & \frac{\Delta t}{2}q_y & \frac{\Delta t}{2}q_z \\ & & -\frac{\Delta t}{2}q_w & \frac{\Delta t}{2}q_z & -\frac{\Delta t}{2}q_y \\ -\frac{\Delta t}{2}\hat{\boldsymbol{\omega}}_k^+ & -\frac{\Delta t}{2}\left[\hat{\boldsymbol{\omega}}_k^+\times\right] & -\frac{\Delta t}{2}q_z & -\frac{\Delta t}{2}q_w & \frac{\Delta t}{2}q_x \\ & & \frac{\Delta t}{2}q_y & -\frac{\Delta t}{2}q_x & -\frac{\Delta t}{2}q_w \\ \multicolumn{2}{c}{0_{3\times4}} & & I_{3\times3} & \end{array}\right] \tag{B.8}$$

$$P_{k+1}^- = F P_k^+ F^T + Q \tag{B.9}$$

where $[\hat{\boldsymbol{\omega}}_k^+\times]$ is the cross product matrix of $\hat{\boldsymbol{\omega}}_k^+$ as defined in Equation A.6, and q_w, q_x, q_y, q_z are elements of the quaternion estimate $\hat{\boldsymbol{q}}_{k+1}^- = [q_w\ q_x\ q_y\ q_z]$.

Measurement Update

After prediction, the predicted state is updated using N quaternion measurements $\tilde{\boldsymbol{q}}$. The N measurements are stacked in a column vector $\tilde{\boldsymbol{y}}_{k+1}$. The predicted measurements $\hat{\boldsymbol{y}}_{k+1}^-$ are simply the predicted quaternion $\hat{\boldsymbol{q}}_{k+1}^-$. The innovation $\boldsymbol{v}_{k+1}$ can then be calculated:

$$\boldsymbol{v}_{k+1} = \tilde{\boldsymbol{y}}_{k+1} - \hat{\boldsymbol{y}}_{k+1}^- = \begin{bmatrix} \tilde{\boldsymbol{q}}_1 \\ \vdots \\ \tilde{\boldsymbol{q}}_N \end{bmatrix}_{\text{size } N} - \begin{bmatrix} \hat{\boldsymbol{q}}_{k+1}^- \\ \vdots \\ \hat{\boldsymbol{q}}_{k+1}^- \end{bmatrix}_{\text{size } N} \tag{B.10}$$

The sensitivity matrix H is then used to calculate the innovation covariance matrix S_{k+1}. Since the measurements are directly a quaternion, this leads to the following:

$$H = \begin{bmatrix} I_{4\times4} & 0_{4\times3} \end{bmatrix} \tag{B.11}$$

$$S_{k+1} = H P_{k+1}^- H^T + R \tag{B.12}$$

The innovation covariance and sensitivity matrix are then used to calculate the Kalman gain matrix K_{k+1}:

$$K_{k+1} = P_{k+1}^- H^T S_{k+1}^{-1} \tag{B.13}$$

The Kalman gain K_{k+1} can then be used to update the state using the innovation $\boldsymbol{v}_{k+1}$:

$$\hat{\boldsymbol{x}}^{+}_{k+1} = \hat{\boldsymbol{x}}^{-}_{k+1} + K_{k+1}\boldsymbol{v}_{k+1} \tag{B.14}$$

Finally, the covariance matrix can similarly be updated:

$$P^{+}_{k+1} = (I_{7\times 7} - K_{k+1}H)P^{-}_{k+1}(I_{7\times 7} - K_{k+1}H)^{T} + K_{k+1}RK^{T}_{k+1} \tag{B.15}$$

The Joseph form covariance update is used here for increased numerical accuracy [113].

B.2 | Multiplicative Filter

The MEKF is similar to the AEKF, but instead of estimating the quaternion elements directly, it estimates a 3-element error vector $\boldsymbol{\delta\hat{p}}$. This vector contains generalized Rodrigues parameters (GRP), which is an attitude representation that introduces a singularity. However, since the vector only represents an attitude error, the singularity is never reached in practice. This leads to the following internal state:

$$\hat{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{\delta\hat{p}} \\ \hat{\boldsymbol{\beta}} \end{bmatrix} \tag{B.16}$$

The GRP representation can be obtained from an error quaternion $\boldsymbol{\delta q}$, as described in Equation A.9. In reverse, an error quaternion can be obtained from the GRP representation using Equation A.10.

The propagation step is equal to that of the AEKF. The only change is that the state transition matrix F is resized to match the smaller 6-element state:

$$F = \begin{bmatrix} -\frac{\Delta t}{2}\left[\hat{\boldsymbol{\omega}}^{+}_{k}\times\right] & \begin{matrix} -\frac{\Delta t}{2}q_w & \frac{\Delta t}{2}q_z & -\frac{\Delta t}{2}q_y \\ -\frac{\Delta t}{2}q_z & -\frac{\Delta t}{2}q_w & \frac{\Delta t}{2}q_x \\ \frac{\Delta t}{2}q_y & -\frac{\Delta t}{2}q_x & -\frac{\Delta t}{2}q_w \end{matrix} \\ 0_{3\times 3} & I_{3\times 3} \end{bmatrix} \tag{B.17}$$

where $[\hat{\boldsymbol{\omega}}^{+}_{k}\times]$ is the cross product matrix of $\hat{\boldsymbol{\omega}}^{+}_{k}$ as defined in Equation A.6, and q_w, q_x, q_y, q_z are elements of the quaternion estimate $\hat{\boldsymbol{q}}^{-}_{k+1} = [q_w\ q_x\ q_y\ q_z]$.

In the measurement update, the multiplicative approach replaces the additive approach of the AEKF. The calculation of the innovation $\boldsymbol{v_{k+1}}$ is now performed by calculating an error quaternion and representing it with a GRP vector:

$$\begin{aligned} \boldsymbol{\delta q}_i &= \tilde{\boldsymbol{q}}_i \otimes (\hat{\boldsymbol{q}}^{-}_{k+1})^{*} \\ \boldsymbol{v_{k+1}} &= \begin{bmatrix} \boldsymbol{\delta p}^{-}_{1}(\boldsymbol{\delta q}_1) \\ \vdots \\ \boldsymbol{\delta p}^{-}_{N}(\boldsymbol{\delta q}_N) \end{bmatrix} \end{aligned} \tag{B.18}$$

where $\boldsymbol{\delta q}_i$ is the error quaternion corresponding to the i-th measurement $\tilde{\boldsymbol{q}}_i$, $(\hat{\boldsymbol{q}}^{-}_{k+1})^{*}$ is the conjugate of the propagated quaternion, and $\boldsymbol{\delta p}$ is the GRP representation of $\boldsymbol{\delta q}$ following Equation A.9.

Similar to the state transition matrix, the sensitivity matrix H is also resized to match the new state size:

$$H = \begin{bmatrix} I_{3\times 3} & 0_{3\times 3} \end{bmatrix} \tag{B.19}$$

The Kalman gain matrix K_{k+1} is then used to calculate the state update $\boldsymbol{\Delta x}_{k+1}$ containing a change in $\boldsymbol{\delta\hat{p}}$ and $\hat{\boldsymbol{\beta}}$:

$$\boldsymbol{\Delta x}_{k+1} = K_{k+1}\,\boldsymbol{v_{k+1}} = \begin{bmatrix} \boldsymbol{\delta\hat{p}} \\ \boldsymbol{\Delta\hat{\beta}} \end{bmatrix} \tag{B.20}$$

The bias prediction is straightforwardly updated as $\hat{\boldsymbol{\beta}}^{+}_{k+1} = \hat{\boldsymbol{\beta}}^{-}_{k+1} + \boldsymbol{\Delta\hat{\beta}}$. The predicted quaternion is updated by transforming the GRP error vector into an error quaternion:

$$\hat{\boldsymbol{q}}^{+}_{k+1} = \boldsymbol{\delta\hat{q}}(\boldsymbol{\delta\hat{p}}) \otimes \hat{\boldsymbol{q}}^{-}_{k+1} \tag{B.21}$$

where $\boldsymbol{\delta\hat{q}}$ is the error quaternion corresponding to GRP error vector $\boldsymbol{\delta\hat{p}}$ following Equation A.10.

Finally, the covariance update follows the approach in the AEKF, as given by Equation B.15.

C
Target Tracking Simulation

One of the scenarios simulated in Ansys Systems Tool Kit (STK), as described in section 6.4, is the target tracking scenario. In this simulation, the spacecraft maintains a default position until it approaches one of several specified ground targets. Once line-of-sight with the target is established, the satellite uses a controller to orient its sensor to face the target. While passing over the target, the controller tracks the target until it leaves line-of-sight, at which point it returns to the default position.

Several large cities were randomly selected as ground targets with enough spacing to avoid conflicts but provide passes at regular intervals. These targets are shown on a world map in Figure C.1. The location of the target is provided to the controller by STK once the satellite has approached it.

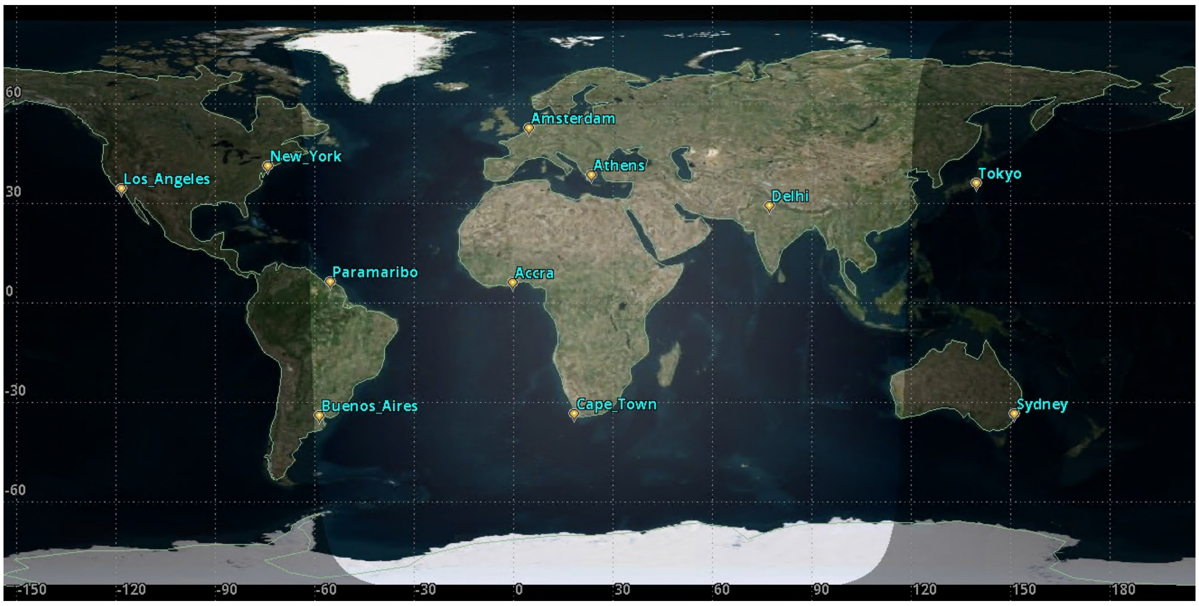


Figure C.1: Selected ground targets in STK for the target tracking scenario

To simulate the tracking behaviour in STK, a custom controller must be implemented that adheres to this logic. A VBScript was used for this purpose, following an example provided by STK. A proportional-derivative (PD) controller is used, based on the data provided by STK at a given timestep. The code used in the controller is shown in Listing C.1.

First, the entry point of the code is `Function VB_feedback` on line 5. Note that this function must have the same name as the script file, in this case `VB_feedback.vbs`. When called by STK, it is specified whether the controller is being registered or a compute step should be performed.

If the controller is being registered, `Function VB_feedback_register` on line 22 is called. This function specifies the inputs that are needed for the controller, and which outputs it provides. Once registered, STK can start using the expected output of the compute function by providing its expected inputs. In this case, four inputs are required:

- The current epoch (`time` , line 29)
- The derivative of the attitude, or angular rate (`att` , line 30)
- The difference between the current and target attitude (`erratt` , line 31)
- The inertia matrix of the spacecraft (`IMtx` , line 32)

The controller provides one output:

- A torque vector (`Torque` , line 26)

Once the function is registered, STK will call the compute function, `Function VB_feedback_compute` on line 37, at each timestep with the required inputs. The variables are extracted from the input, and gain values of $k = 0.2$ and $c = 1.2$ were set through experimentation. The control law is applied to calculate the three elements of the torque vector. This is finally supplied as the output.

When the spacecraft is in line-of-sight to one of the defined targets, the attitude error to this target is automatically calculated and provided to the compute function by STK. If there is no target in sight, the attitude error to the default position is provided instead. With this automated switching, the controller attempts to correct the spacecraft attitude smoothly. The gain values were tuned to provide a relatively fast response with minimal overshoot.

Listing C.1: VBScript code implementing a PD controller for STK

```
1  Dim VB_feedback_init
2  Dim VB_feedback_Inputs
3  Dim VB_feedback_Outputs
4
5  Function VB_feedback(argArray)
6      Dim retVal
7
8      If IsEmpty(argArray(0)) Then
9          retVal = VB_feedback_compute(argArray) ' do compute
10     ElseIf argArray(0) = "register" Then
11         VB_feedback_init = -1
12         retVal = VB_feedback_register() 'do register
13     ElseIf argArray(0) = "compute" Then
14         retVal = VB_feedback_compute(argArray) 'do compute
15     Else
16         retVal = Empty 'bad call
17     End If
18
19     VB_feedback = retVal
20 End Function
21
22 Function VB_feedback_register()
23     ReDim argStr(5)
24
25     'Outputs
26     argStr(0) = "ArgumentType = Output ; Type = Parameter ; ArgumentName = Torque ; Name =
           Torque ; BasicType = Vector "
27
28     'Inputs
29     argStr(1) = "ArgumentType = Input ; ArgumentName = time ; Name = Epoch "
30     argStr(2) = "ArgumentType = Input ; ArgumentName = att ; Type = Attitude ; Derivative =
           Yes "
31     argStr(3) = "ArgumentType = Input ; ArgumentName = erratt ; Type = Attitude ; RefName =
           Body ; RefSource = Satellite/SAT_NAME "
32     argStr(4) = "ArgumentType = Input ; ArgumentName = IMtx ; Type = Inertia ; Name = Inertia
            "
33
```

```
    VB_feedback_register = argStr
End Function

Function VB_feedback_compute(stateData)
    If VB_feedback_init < 0 Then
        Set VB_feedback_Inputs = g_GetPluginArrayInterface("VB_feedback_Inputs")
        Set VB_feedback_Outputs = g_GetPluginArrayInterface("VB_feedback_Outputs")
        VB_feedback_init = 1
    End If

    'get input values
    Dim att, erratt, IMtx
    att = stateData(VB_feedback_Inputs.att)
    erratt = stateData(VB_feedback_Inputs.erratt)
    IMtx = stateData(VB_feedback_Inputs.IMtx) 'returned as 9x1 array instead of 3x3

    'create return value and torque vector
    ReDim returnValue(1)
    ReDim torque(3)

    'set gain values
    Dim k, c
    k = 0.2
    c = 1.2

    'apply feedback control law
    ReDim temp(3)
    temp(0) = (k * erratt(0) * erratt(3)) + (c * att(4))
    temp(1) = (k * erratt(1) * erratt(3)) + (c * att(5))
    temp(2) = (k * erratt(2) * erratt(3)) + (c * att(6))

    torque(0) = -1 * IMtx(0) * temp(0) + -1 * IMtx(3) * temp(1) + -1 * IMtx(6) * temp(2)
    torque(1) = -1 * IMtx(1) * temp(0) + -1 * IMtx(4) * temp(1) + -1 * IMtx(7) * temp(2)
    torque(2) = -1 * IMtx(2) * temp(0) + -1 * IMtx(5) * temp(1) + -1 * IMtx(8) * temp(2)

    'set torque as return value
    returnValue(VB_feedback_Outputs.Torque) = torque
    VB_feedback_compute = returnValue
End Function
```